Design, Analysis and Testing of Efficient Memristive Neural Networks

Anu Bala

School of Engineering, Computing and Mathematics

Oxford Brookes University, Oxford, UK

A Thesis submitted for the degree of

Doctor of Philosophy

August 2022

Abstract

Neuromorphic systems are gaining significant importance in an era where CMOS digital techniques are reaching physical limits due to more power consumption, chip area and complex structures. Thus, memristors have appeared as promising devices not only in the area of neural systems but also in the fields of logic design, memory design, sensor and cryptography systems, etc. due to their non-volatility, nanoscale size and physical layout.

We present memristor-based neural network designs and implementations in this work. This work first simulates the different memristor modelling techniques in a high-level language (especially in C++). The multi-layer neural network is simulated based on this modelling. The results demonstrated that the linear and non-linear separable functions performed well with this modelling technique. This simulation modelling can be used to reduce the simulation time of very large memristor based neural networks. Then, an improved learning method for ex-situ training and a novel circuit design for activation function are also presented in this work. Based on these designs and learning methods, various single and multi-layer memristive crossbar neural architectures are implemented and tested in SPICE. Experimental results show that this learning method performed efficiently while implementing memristor-based neural networks in ex-situ. The results also demonstrated that the memristive crossbar-based multi-layer neural networks with the proposed circuit occupy less chip area as compared to existing circuit designs used for activation functions. Lastly,

this work analyses the fault tolerance behaviours of memristor-based neural networks and evaluates the yield of memristor crossbar array-based neural networks using the Markov chains. The repairability process is also considered while evaluating the yield. Our analysis also shows that a well-designed network can function effectively with up to 50% faulty memristors with negligible impact on the learning capabilities of the network. The analysis is useful when designing a network with redundancy. The results also show that the yield can be improved with redundancies and a higher ratio of SA0 faults.

Publications

Research Papers from this Thesis

A. Bala, S. Khandelwal, A. Jabir and M. Ottavi, "Yield Evaluation of Faulty Memristive Crossbar Array-based Neural Networks with Repairability." The 28th IEEE International Symposium on On-Line Testing and Robust System Design. Torino, Italy Sept 2022 (**Accepted**).

A. Bala, X. Yang, A. Adedotun, S. Khandelwal, and A. Jabir, "Memristor Crossbar based Learning Method for Ex-situ Training in Neural Network." Book entitled "Nanoscale Memristor Device and Circuits Design", Elsevier, Book Chapter (**Submitted**).

A. Bala, X. Yang, A. Adeyemo and A. Jabir, "Efficient and Low Overhead Memristive Activation Circuit for Deep Learning Neural Networks" Journal of Low Power Electronics (2019) ISSN: 1546-1998 eISSN: 1546-2005.

A. Bala, X. Yang, A. Adeyemo and A. Jabir, "A Memristive Activation Circuit for Deep Learning Neural Networks", IEEE 8th International Symposium on Embedded Computing and System Design (ISED), India, 2018, pp.1-5.

A. Bala, A. Adeyemo, X. Yang and A. Jabir, "Learning Method for Ex-situ Training of Memristor Crossbar based Multi-Layer Neural Network" 2017 IEEE 9th International Congress on Ultra Modern Telecom and Control Systems, in Munich, Germany, October 2017, doi:10.1109/ICUMT.2017.8255181 (**Best Paper award**).

A. Bala, A. Adeyemo, X. Yang, and A. Jabir, "High level abstraction of memristor model for neural network simulation," in Embedded Computing and System Design (ISED), 2016 Sixth International Symposium on, IEEE, 2016, pp. 318–322.

Research Papers in Allied Areas

S. Khandelwal, **A. Bala**, V. Gupta, M. Ottavi, E. Martinelli and A. Jabir, "Fault Modeling and Simulation of Memristor based Gas Sensors," 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS), 2019, pp. 58-59, doi: 10.1109/IOLTS.2019.8854459.

X. Yang, A. Adeyemo, **A. Bala** and A. Jabir, "Novel Techniques for Memristive Multifunction Logic Design" in special issue Modeling, Optimization and Simulation for High Performance and Ultra-Low Power Circuits and Systems" with Integration, the VLSI Journal, available online 15 Sept, 2017. DOI: 10.1016/j.vlsi.2017.09.005.

X. Yang, A. Adeyemo, **A. Bala** and A. Jabir, "Parasitic effects on memristive logic architecture," 2017 27th International Symposium on Power

and Timing Modeling, Optimization and Simulation (PATMOS), Thessaloniki, 2017, pp. 1-5. doi: 10.1109/PATMOS.2017.8106983.

X. Yang, A. Adeyemo, **A. Bala** and A. Jabir, "Novel memristive logic architectures," 2016 IEEE 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), Bremen, Germany, 2016, pp. 196-199. doi:10.1109 / PATMOS.2016.7833687. (**Best poster award**).

Adeyemo, Adedotun, X. Yang, **A. Bala**, and A. Jabir, "Analytic models for crossbar write operation," in Embedded Computing and System Design (ISED), 2016 Sixth International Symposium on, IEEE, 2016, pp. 313-317.

Adeyemo, Adedotun, X. Yang, **A. Bala**, J. Mathew, and A. Jabir, "Analytic models for crossbar read operation," in IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS), 2016, pp. 3.

Declaration

This thesis is submitted to Oxford Brookes University in accordance with the requirements for the award of Doctor of Philosophy in the School of Engineering, Computing and Mathematics. It has not been submitted for any other degree or diploma of any examining body. Some parts of the work presented in this thesis have previously appeared in the published papers listed in the publications section. Except where specifically acknowledged, it is all the work of the Author.

Anu Bala, August, 2022

Acknowledgement

God Almighty has always loved me, and I am truly grateful for that. This work is the result of weeks of hard work, persistence, and dedication.

My sincere appreciation goes to my supervisor and my research team members, Dr. Abusaleh Jabir, Prof. Dhiraj Pradhan, Dr. Adeyemo Adedotun, Dr. Xiaohan Yang and Dr. Saurabh Khandelwal, for their continuous support, guidance and assistance for the duration of this project. Without their help, this work simply could not have been accomplished. I would also like to express my special thanks to Dr. Jimson Matthew for his support at the initial stage of my work.

I am thankful to my friends, Tajinder Kaur, Harpreet Kaur, Dr. Mona Eisa, and Dr. Maite. They are always there for me.

I also express my heartfelt gratitude to my husband (Capt. Amandeep Singh) and my son (Imaan Partap Singh Badwal) for their unconditional love and support. I also offer my big thanks to my parents and my wonderful family (the Badwal family). They have always encouraged and supported me.

Contents

List of Figures

List of Tables

List of Acronyms

0T1M	Zero Transistor One Memristor
IT1M	One Transistor One Memristor
AF	Activation Function
ANNs	Artificial Neural Networks
CLA	Concurrent Learning Algorithm
CMOS	Complementary Metal-Oxide-Semiconductor
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
HRS	High Resistance State
I-V	Current-Voltage
IC	Integrated Circuits
LCA	Locally Competitive Algorithm
LRS	Low Resistance State
MC	Markov Chain
MRL	Memristor Rationed Logic
MSE	Mean Square Error
PD	Poisson Distribution
PE	Processing Elements
RAM	Random Access Memory

ReLU	Rectified Linear Unit
ReRAM	Resistive Random Access Memory
SAFs	Stuck-at-Faults
SRAM	Static Random Access Memory
STDP	Spike Time Dependent Plasticity
TEAM	ThrEshold Adaptive Memristor
TiO_2	Titanium dioxide
VTEAM	Voltage ThrEshold Adaptive Memristor

Chapter 1

Introduction

1.1 Motivation & Problem Background

Recent years have seen the incorporation of Artificial Neural Networks (ANNs) into a variety of commercial products and services, including mobile devices and cloud computing. Recent studies have suggested [18, 46] that when huge amounts of computational power are available, ANNs have the potential to perform exceptionally well for large-scale realistic learning applications. However, their usability is limited by their computational intensity due to area and power constraints. Therefore, a new specific hardware design strategy is needed to overcome these limitations. Historically, the number of transistors in Integrated Circuits (IC) has about doubled every two years, as predicted by Moore's law [56]. As the number of transistors on the ICs has increased, it has become apparent that there is a limit to which the transistors can be shrunk. Unless other alternatives to silicon-based transistors or new architecture developments occur, there is a high possibility that processor performance will stay static. Researchers have been investigating a variety of options for overcoming such obstacles. Moreover, the implementation of powerful machine learning algorithms like backpropagation is inefficient for general-purpose digital hardware (i.e., the von Neumann architecture). One of the main reasons for this is the physical separation between the memory arrays that store the synaptic weight values and the arithmetic module that calculates the updated rules.

In the previous two decades, various Hardware Neural Network chips have been implemented, known as neurochips. These chips are based on analog, digital and hybrid electronic technologies, as well as Field Programmable Gate Array (FPGA) and Random Access Memory (RAM) based technologies [36, 52, 55, 74]. The majority of Hardware Neural Networks (HNNs) are designed and implemented using the Complementary Metal-Oxide Semiconductor (CMOS) technology [52]. The major challenge in these designs is scalability, as they are not used for large-scale realistic applications. These designs are also area and power hungry.

The effectiveness of the HNNs design is largely dependent on the compromises that are made between accuracy, chip area, and processing speed. Digital neural systems provide a high degree of accuracy, but at the expense of a larger chip area, slower speed, and higher power consumption. In contrast to digital implementations, analog implementations are typically more efficient in terms of chip area and processing speed, but their accuracy is limited due to the spatial non-uniformity of analog components and their non-ideal responses [19, 21, 53]. Another major issue with analog HNNs is weight storage. In these system implementations, weights are normally stored in electronic devices such as capacitors, resistors, and floating gate transistors. Due to charge leakage, the synaptic weights of capacitors have a limited retention time; therefore, they need to have their dynamic weights updated at frequent intervals. Resistors are fixed and cannot be altered after they have been manufactured. Thus, they are only suitable for use in non-learning hardware. That is the issue of using traditional resistors for weight storage. In conjunction with analog multipliers, floating gate transistors have been employed effectively as synapses. Nevertheless, this method suffers from significant non-linearity in synaptic weightings. Moreover, non-volatile weight storage is another big challenge in analog HNN implementations.

A tried-and-tested approach to designing information processing systems is neuromorphic engineering. Neuromorphic networks are more advantageous in cases where parallel computation is required. A high number of processing elements (PE) are

required for efficient neuromorphic computing [84]. Researchers in the area of neuromorphic computing are also exploring ways to reduce the amount of chip area needed to mimic the massive computing ability of the human brain. Existing CMOS technology lacks the density and connectivity required for more advanced neuromorphic architectures. The neuromorphic systems simulated using current nano technologies are magnitudes below the integration density of neurons in the human brain [87, 109]. Humans with millisecond-operating neurons can execute arbitrary picture recognition tasks in tens to hundreds of milliseconds, whereas even the most powerful computers would require hours, if not days, to do the same. This gap between digital computing and biology (especially the human brain) motivates the development of technologies with higher connection densities than CMOS can provide.

A novel device called 'memristor' has recently created new opportunities not only in the area of HNNs and neuromorphic computing but also in other big applications like logic and memory designs. A memristor is a non-volatile variable resistor, which was predicted from theory by Leon Chua [14] in 1971, and it represents the relationship between charge (q) and flux (φ). The scientists at Hewlett-Packard Laboratories achieved its physical manifestation in 2008 [82] . After that, memristive behaviour has been demonstrated by numerous research teams using different materials. Moreover, nanoscale fabrication of this device is also possible, and it retains its previous state even when the power is switched off. Furthermore, physical memristors can be built in a dense grid called a crossbar. A memristor occupies $4F^2$ area in the crossbar (where F is the device feature size) which is 36 times smaller than an SRAM memory cell. One of the most promising uses of memristors is in the implementation of artificial neural networks, as a memristor crossbar is capable of performing several multiply-add operations in parallel and updating the conductance of different memristors simultaneously. Thus, a memristor can be used as a synaptic device in the design of HNNs. Since the device operates similarly to a synapse and the memristance can be gradually changed by the external electrical signal, the memristor is

also being investigated in neuromorphic applications.

1.2 Methodology

The work in this thesis covers the research, technology, and application categories in the field of neural network designs. All the memristor crossbar-based neural circuits were implemented and simulated using a combination of C++, LTspice and MATLAB. The training was provided in C++ and MATLAB. The memristive neural circuits were designed and implemented using LTspiceIV. Various components of LTspice are used, e.g voltage sources, comparators, etc. The SPICE memristor models and comparators were developed to aid simulation with EDA tools. Different linear and non-linear datasets, pattern classifiers and benchmark datasets, e.g. the iris dataset, were used to test the memristor based neural network designs. The pattern classifiers were chosen randomly to examine the functionality of the improved learning method and test the proposed memristive circuit designs. The complexity of the pattern classifiers was not considered at this stage. The processes in flowchart shown in Fig. 1.1 are used in each chapter from Chapter 3 to Chapter 6. In each chapter, we defined the problem and proposed a solution. Then, designed, trained and implemented the memristive neural networks based on the proposed solutions and the results are evaluated at the end of each chapter.

1.3 Contributions

Efficient memristor-based neural network circuit designs are required. This thesis examines the memristor crossbar-based neural network architectures. It also examines the memristive neural networks based on ex-situ training and activation function. We have also examined the fault-tolerance and yield of memristor crossbar based neural circuits.

The following is a list of key contributions of this thesis:

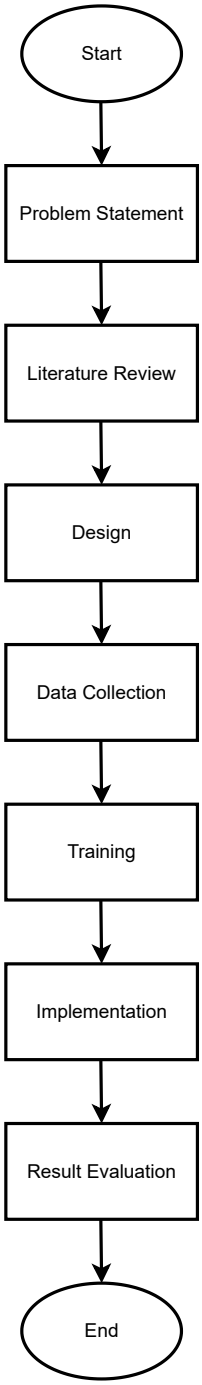


Figure 1.1: Flowchart of the research process.

1. The SPICE simulation of large memristor crossbar arrays is time consuming. A novel memristor modelling is provided using a high-level language. Memristor models are simulated using this modelling approach, especially in C++. We designed a C++ framework to simulate various linear and non-linear separable functions in order to reduce the simulation time.

2. In order to implement memristive neural networks in hardware, improved learning methods, mapping schemes and more accurate computations in the memristor crossbar architectures are required. The author presents an improved learning method for ex-situ training. A voltage-divider technique is considered for learning during training in software. This method is very close to the hardware computation of the memristor crossbar. Weights are updated using this learning method while training in software so that similar results can be produced while writing these weights as conductance in memristor crossbar implemented in LTspice. The results show that more accurate computations can be achieved in the LTspice implementation of memristive crossbar neural circuits using this learning method, as well as a reduction in training time.

3. The non-linearity of activation functions poses yet another challenge while designing compact hardware. A novel circuit for activation function (AF) is presented. This circuit design is based on memristor MIN functionality. Two memristors, along with a comparator, are used to design the circuit. The proposed circuit approximates the Rectified Linear Unit (ReLU) activation function widely used in the training of neural networks. Different multi-layer memristive neural circuit implementations are carried out using this function in order to check performance. Our circuit design is area efficient and helps to train the network faster.

4. Neural networks can accept a limited number of faulty synaptic weights. However, a high defect rate dramatically reduces the accuracy of the matrix-vector

calculations. The author analyses the error tolerance of memristor crossbar neural circuits with stuck-at-faults (SAFs). To examine fault tolerance in the memristor crossbar architecture, random memristors were chosen to be stuck in either a low or high resistance state. The results of the analysis show that the neural circuits based on memristors are capable of learning even when some of the memristors in the memristor crossbar are faulty. The author also calculates the yield of a memristor crossbar array used for neural networks using a Markov chain, which provides flexibility without sacrificing accuracy and representativeness. The benefit of the repair process is also considered while using the Markov chain model. The results show the increase in yield using redundancies and with the increase of SAFs that can be repaired.

1.4 Outline of the Thesis

This thesis is organised as follows:

- Chapter 2 describes the background theories of a memristor and its physical model. Some existing models of memristors are also discussed in this chapter. Additionally, the memristor-based crossbar neural architecture and some of the existing training methods for neural networks have been introduced. Existing write and read schemes of memristors in the memristor crossbar array are also explained. This chapter also reviewed the existing circuits used for activation functions. Furthermore, it also reviewed the fault-tolerance and yield in the memristor crossbar array, alongside a review of other literature relevant to the contribution of this thesis.

- Chapter 3 presents the novel memristor modelling in a high-level language. Different memristor models are simulated using C++. Various linear and non-linear separable functions are simulated using the proposed modelling approach.

- In Chapter 4, an improved learning method for memristor based neural networks is proposed. This learning algorithm is used for the ex-situ training method. It is based on the voltage-divider sensing technique. The backpropagation algorithm along with the voltage-divider technique is used for learning and updating the weights in software. Various single and multi-layer memristive neural networks are trained in software while using this learning method for training and then implemented memristive crossbar circuits for these networks in LTspice with consideration of sneak-paths. These networks are also trained and implemented without using the voltage-divider technique. This chapter also compares both the results.

- Chapter 5 presents a novel circuit design for the activation function of a neural network. This chapter starts by describing the memristor-based MIN-MAX operation. Then, a circuit design based on memristor MIN functionality is proposed. It also demonstrated its results using LTspice. This chapter also shows the novel implementation and analysis of different multi-layer memristor crossbar based neural circuits. Furthermore, this chapter compares the proposed activation circuit with other existing activation circuits used in memristive neural networks.

- Chapter 6 begins with the examination of the fault tolerance within the memristor crossbar neural circuits in the presence of stuck-at-faults (SAFs). To evaluate the tolerance of faulty memristors, two distinct memristor-based structures are considered. This chapter also estimates the yield of a memristor-based crossbar array using a Markov chain when SAFs are present. Another method used for estimation and comparison is the Poisson distribution. Different sizes of memristor crossbar arrays are considered. We also considered different fault ratios while calculating the yield. The results are evaluated with or without redundancies.

- Chapter 7 concludes and summarises the work presented in this thesis. Other possible extensions of the research conducted in this thesis are also discovered.

Chapter 2

Background and Literature Review

2.1 Introduction

A memristor is a two-terminal non-volatile passive element with varying resistance. A pinched hysteresis loop is the recognising feature of a memristor. This fourth basic passive element was first theorised by Leon Chua in 1971 [14], and it predicts the relationship between charge and magnetic flux. One of the earliest physical memristor devices was fabricated in 2008 by Hewlett-Packard (HP) Labs [82] based on Chua's original definition [14].

This chapter introduces the memristor, which is commonly referred to as the fourth circuit element. An overview of the memristor device, its existing models, and its applications in different areas are explored. A full description of neural networks, memristor crossbar structure and its use in the implementation of memristor-based neural circuits is also presented in this chapter. Various architectures of memristor crossbars as well as their pros and cons are also discussed. This chapter presents the background relevant to the contribution of this thesis.

2.2 Background

2.2.1 The Theory of Missing Element

In the field of device electronics, there already exists three fundamental circuit elements denoted by their respective symbols: resistor R, capacitor C and inductor L.

These three components are passive elements, which are able to store or dissipate charges. The behaviours of each of the three elements are defined through a linear relationship between two (out of four) circuit variables. These variables are voltage v, Current i, Charge q, and Flux φ as shown in Fig. 2.1. There are six possible combinations of these four parameters. A resistor (R) is characterised by the relationship between voltage and current as $dv = Rdi$. A capacitor (C) is characterised by the relationship between charge and voltage as $dq = Cdv$. Similarly, an inductor (L) is characterised by the relationship between magnetic flux and current: $d\varphi = Ldi$. Leon Chua noticed the missing link between charge and flux and first theoretically described it in his paper [14]. In 1971, Chua introduced a new passive element named the memristor, which exhibits the relationship between charge q, and flux φ [14] as shown in Eq. 2.1. Fig. 2.2 represents the symbol illustrated in Chua's paper [14].

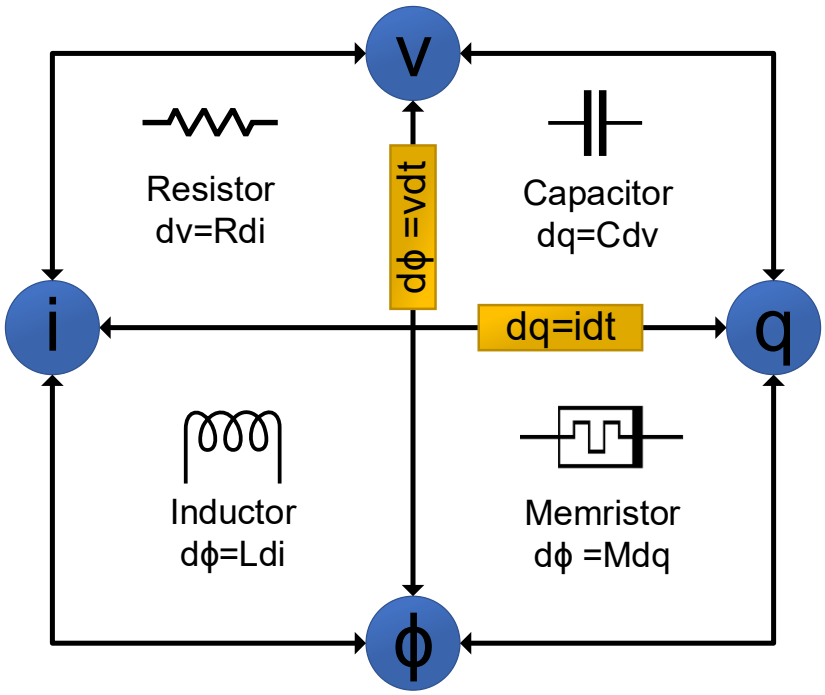


Figure 2.1: Circuit variables relationship diagram.

$$M = \frac{d\varphi}{dq} \tag{2.1}$$

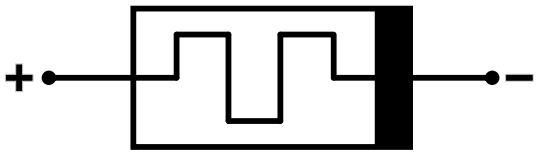


Figure 2.2: Memristor symbol originally proposed by Leon Chua.

Eq. 2.2 and 2.3 describe the flux and charge as these equations represent the time integral of voltage and current.

$$\varphi = \int v(t)dt \Rightarrow \frac{d\varphi}{dt} = v(t) \tag{2.2}$$

$$q = \int i(t)dt \Rightarrow \frac{dq}{dt} = i(t) \tag{2.3}$$

A charge dependent memristance is defines as

$$M(q) = \frac{d\varphi/dt}{dq/dt} \tag{2.4}$$

After substituting Eq. 2.2 and Eq. 2.3 into Eq. 2.4, the Eq. 2.4 becomes

$$M(q) = \frac{v(t)}{i(t)} \tag{2.5}$$

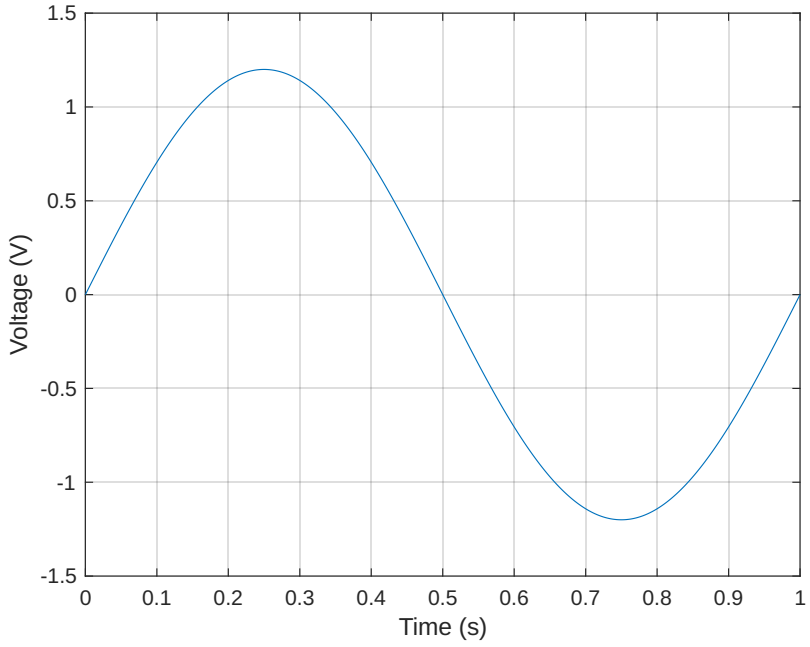
Thus, Eq. 2.5 represents memristance described by Ohm's Law.

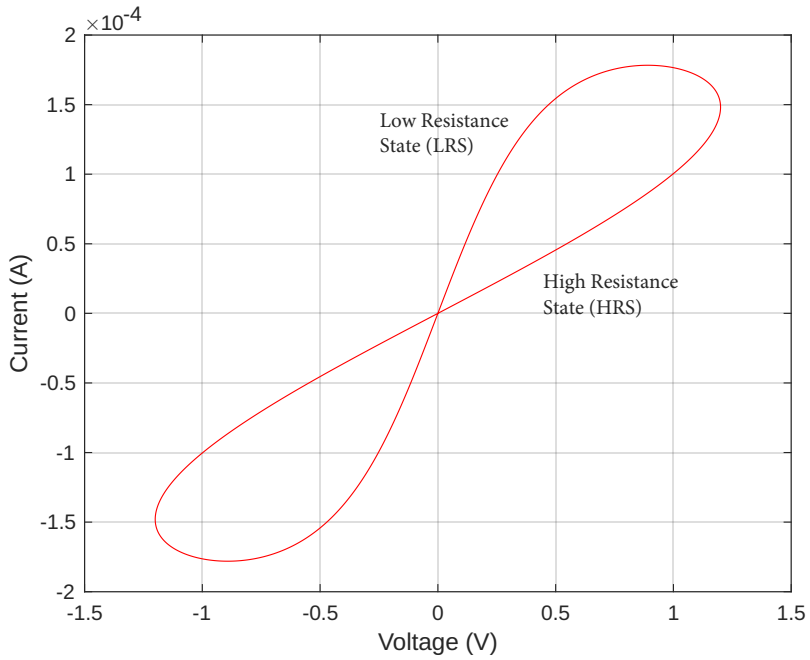
Similarly, a flux based memristance described as Eq. 2.6 is derived from Eq. 2.7

$$M(\varphi) = \frac{dq/dt}{d\varphi/dt} \tag{2.6}$$

$$M(q) = \frac{i(t)}{v(t)} \tag{2.7}$$

Hence, memristance depends on the voltage or current time integral. Any resistive device displaying a current-voltage curve in the shape of a pinch-hysteresis loop as shown in Fig. 2.3b is considered as a memristor [15] for sinusoidal input as shown in Fig. 2.3a.





(b)

Figure 2.3: (a) Sinusoidal voltage input and (b) I-V characteristic of a memristor.

2.2.1.1 HP Memristor based on TiO_2

In 2008, Hewlett-Packard (HP) Labs used a thin film structure of titanium dioxide (TiO_2) to fabricate the first physical memristor device [82]. This was based on the theory of a solid-state element called a memristor, which predicts the relationship between charge and magnetic flux. It consists of two thin films sandwiched between the pair of platinum electrodes as shown in Fig. 2.4(a). It is divided into two regions known as the doped and the undoped regions. The undoped region contains titanium dioxide TiO_2 and the doped region contains poor-oxygen titanium dioxide TiO_{2-x}. The undoped region contains more oxygen vacancies than the doped region and has high resistance, whereas the doped region acts as a semiconductor due to positive dopants and has low resistance. Both the doped and undoped regions are modelled as resistors and the result is equivalent to two resistors in series, as shown in Fig. 2.4(b). Memristor switching is based on the voltage and current passing through it. Its resistance changes from high resistance state (HRS) R_{off} to low resistance state (LRS) R_{on} and vice-versa based on the bias of voltage applied to its terminals. Ohm′s law, as shown in Eq. 2.8, describes the relationship between current and voltage of a memristor. In Fig. 2.4(a), D represents the total length of the device (the doped and the undoped region) and w represents the length of the doped region.

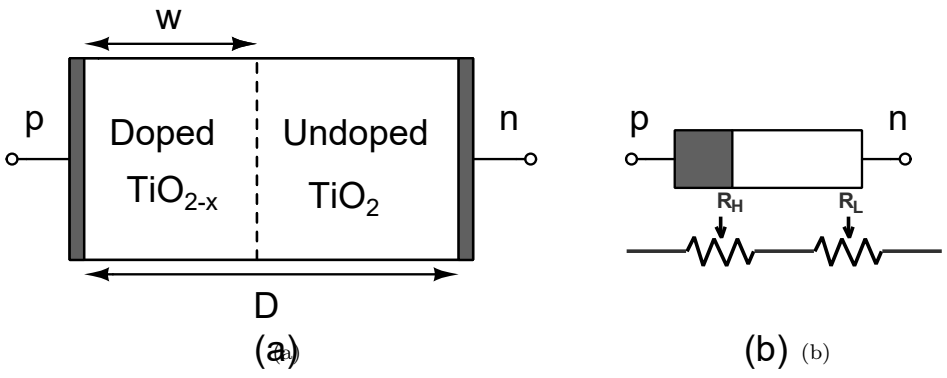


Figure 2.4: (a) TiO_2 Memristor model structure and (b) equivalent circuit.

$$V(t) = M(t)i(t) \tag{2.8}$$

The mathematical model for the resistance of a memristor M is as follows [82]:

$$M(t) = R_{on}\frac{w(t)}{D} + R_{off}\left(1 - \frac{w(t)}{D}\right) \tag{2.9}$$

The state variable w is defined as:

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{on}}{D} i(t) \tag{2.10}$$

$$w(t) = \mu_v \frac{R_{on}}{D} \int_0^t i(t)dt \tag{2.11}$$

where μ_v represents the average mobility of ions.

2.2.2 Memristor Applications

After the realisation of the first memristor model, researchers have initiated significant experiments to illustrate the applications of memristors in different areas. As a result, memristors have been proposed for a broad range of applications, such as neuromorphic systems [32, 61, 77] non-volatile memories [83], chaotic circuits [8, 57] and logic circuit designs [93, 102]. Neuromorphic system designs have gained lots of attention since the invention of the memristor. To simulate neurons and synapses in an electronic regime requires an implementation with very low power consumption. As a result, electronic synapses are more difficult to simulate. However, the memristor structure mimics the biological synapse. Thus, memristors are gaining significant interest in this area as they can be used as a synapse with negligible power requirements [32, 61]. It has been demonstrated experimentally [61] that a memristor emulator is designed with three electronic neurons connected by two memristor emulator synapses, which shows associative memory function. Memristors are also used to replicate the functionality of synapses in [32, 77] to create efficient neuromorphic hardware systems with high density and connectivity. Controlled voltage pulses are utilized to alter the conductance of a memristor in the same way as neuron spikes are used to alter synaptic weights during the learning process in the human brain.

Another widely researched application area for memristors is non-volatile memory design, and memristors are becoming more viable candidates in computing memories. Memristors are replacing transistors (which are used in conventional memory) in the design of denser memory structures as they are used as the main memory cells. It is also possible to construct hybrid CMOS/memristor memory structures using CMOS building blocks for the control and peripheral circuits and the memristor as the memory unit [83].

Another interesting application of memristors is logic circuits. The use of memristors has been proposed in many different ways to perform logic operations. Memristors are associated with CMOS structures in certain logic families to perform logic functions while voltage levels represent the logical values. Moreover, memristors can also be used as computational components inside logic gates [93, 102] or as reconfigurable switches for FPGA-like architectures [7, 81, 83]. In another method, memristors are the primary building blocks as the resistance of the memristor is considered as the logical state, where logic 0 is represented by high resistance and logic 1 is represented by low resistance [7]. This method is ideal for architectures of crossbar arrays and it can be implemented into a regular crossbar based on a memristor, which is widely used for memories [75].

The chaotic system is one of the useful applications of the memristor, where the memristor as a non-linear device can be used for random number generation and encryption. Chua designed the memristor model [31] to generate a chaotic attractor with negative conductance and a capacitor. It was clearly based on simulation, but memristor-based chaotic systems were initiated by simplicity and functionality. The simple chaotic oscillator [57] is demonstrated using an inductor-capacitor-memristor series circuit. A HP memristor based chaotic circuit is published in [8]. In this circuit, two memristors are used in an antiparallel connection.

2.2.3 Memristor Modelling

Since the exciting 2008 discovery of non-volatile memristive behaviour in titanium dioxide (TiO_2)-based nanofilms at HP Labs [82], both academia and industry have been searching for innovative memristive materials and manufacturing technologies. There are already numerous device models that attempt to characterize both the current–voltage (i–v) behaviour and the device dynamics. Memristors are applicable to a wide variety of applications. In each application, different memristor characteristics are expected. An element with the ability to calculate, control, and store data after calculation is required in logic and memory applications. They must be able to read and write quickly. A suitable model is required to design, evaluate, and simulate memristor-based circuits and applications. The model with fast switching time and higher R_{on} and R_{off} is required for memristor based neural applications. The higher ratio between R_{on} and R_{off} resistances is also required for storing Boolean data in a memristor. Other essential properties for memristor applications are scalability, low power consumption and compatibility with conventional CMOS. This section will describe the most significant memristor models and window functions currently available.

2.2.3.1 Linear Ion Drift Model

A linear ion drift model is proposed in [82]. In this model, it is assumed that a physical device of width D consists of two regions. One region named as w contains a high concentration of dopants that is the state variable of the system and the other region of width $(D - w)$ is the oxide region. The region includes the dopants has a higher conductance than the oxide region. Hence, the device is modelled as two resistors are connected in series and it is assumed that the linear ion drift and ohmic conductance are in uniform field, and the average ion mobility rate of the ions is equal μ_v. The i-v relationship of this model is shown in Fig. 2.5 for sinusoidal input pulse. The following equations describe the linear model:

$$v(t) = M(t)i(t) \tag{2.12}$$

where M(t) is the memristance of the memristor which is described as:

$$M(t) = R_{on}\frac{w(t)}{D} + R_{off}\left(1 - \frac{w(t)}{D}\right) \tag{2.13}$$

The Eqn. 2.12 becomes 2.14 by substituting the Eqn. 2.13 in Eqn. 2.12

$$v(t) = \left(R_{on}\frac{w(t)}{D} + R_{off}\left(1 - \frac{w(t)}{D}\right)\right)i(t) \tag{2.14}$$

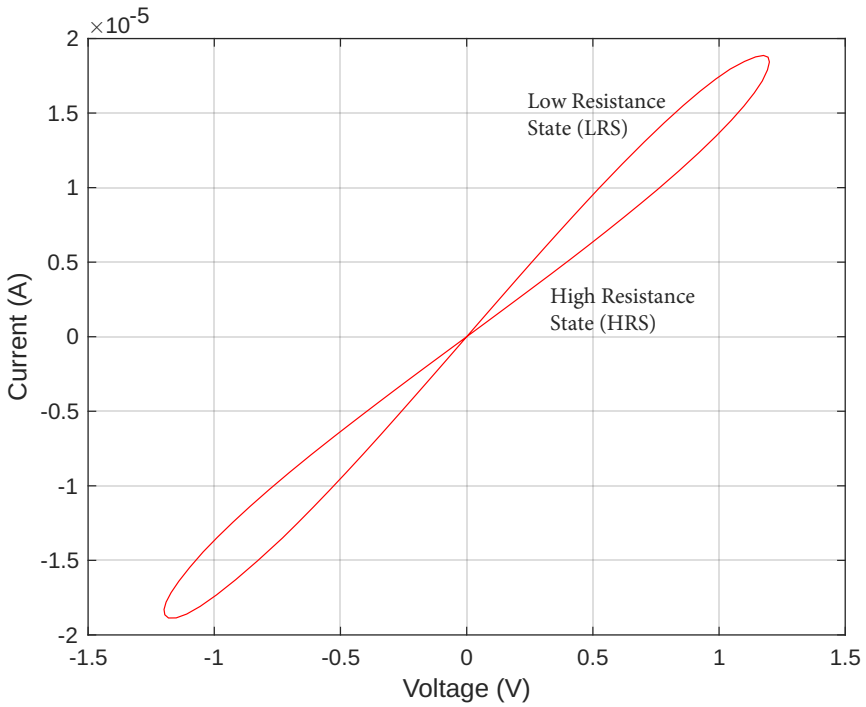


Figure 2.5: I-V characteristic of linear ion drift model.

2.2.3.2 Window Functions

The major issue in the linear model is boundary issue. The window function is used to prevent the state variable to go beyond its defined bounds [0 D] and also add non-linearity close to the boundaries. When the state variable w is at the boundaries,

its derivative is multiplied by a window function so that it is forced to 0. Some of the major window functions and their properties are listed in the Table 2.1.

Table 2.1: Comparison of window functions.

Window Function	Joglekar [34]	Biolek [6]	Prodromakis [64]
f(w)	$1-(2w/D-1)^{2p}$	$1-(w/D-stp(-i))^{2p}$	$j(1-[(w-0.5)^2+0.75]^p$
Impose nonlinear Drift	Yes	Yes	Yes
Scalability	No	No	Yes
Solve Boundary Effect	No	No	Yes
Compatible Memristive Models	Linear/Nonlinear/TEAM	Linear/Nonlinear/TEAM	Linear/Nonlinear/TEAM
Symmetric	Yes	Yes	Yes

2.2.3.3 Non-Linear Ion Drift Model

A voltage-controlled memristor with non-linear relationship between the voltage and the internal state derivative is considered as non-linear ion drift model. It was proposed in [35] and also shows asymmetric behaviour of switching. In this model, the state variable w is bounded between zero and one. The following equation shows the relationship between current and voltage in this model.

$$i(t) = w(t)^n \beta \sinh\left(\alpha v(t)\right) + \chi\left[\exp\left(\gamma v(t)\right) - 1\right] \tag{2.15}$$

where α, β, χ, γ are the fitting parameters of the device and n is the variable that defines how the state variable affects the current. The state variable w will be close to 1, while the device is in ON position, and the current is determined by $\beta \sinh(\alpha v(t)$. The current is estimated by $\chi[exp(\gamma v(t)) - 1]$, w will shift close to 0 when it is in OFF position.

The differential equation of state variable w is defined as:

$$\frac{dw}{dt} = a f(w) v(t)^m \tag{2.16}$$

where $f(w)$ is a window function and a and m are constants.

2.2.3.4 Simmons Tunnel Barrier Model

In [62], a more detailed physical model was proposed. Due to an exponential dependency on the movement of the ionized dopants, that is, changes within the state variable, this model assumes nonlinear and asymmetric switching behavior. There is a resistor in series with an electron tunnel barrier in this model, rather than two resistors in series as in the linear drift model. The state variable w represents the width of the tunnel. The current and voltage relationship in Simmons tunnel model is defines as:

$$i(t) = \tilde{A}(x, v_g)\phi_1(v_g, x)\exp\left(-B(v_g, x)\cdot\phi_1(v_g, x)^{1/2}\right) - \tilde{A}(x, v_g)\left(\phi_1(v_g, x) + e|v_g|\right) \times \exp\left(-B(v_g, x)\cdot\left(\phi_1(v_g, x) + ev_g\right)^{1/2}\right) \tag{2.17}$$

$$v_g = v - i(t)R_s \tag{2.18}$$

2.2.3.5 ThrEshold Adaptive Memristor (TEAM) Model

The Simmons tunnel barrier model is the most precise physical memristor model. However, without an explicit relationship between current and voltage, this model is very complex and not generic in nature. The TEAM model is proposed in [41] and is considered as a simple and generic memristor model. As it has undefined current-voltage relationship which can be uninhibitedly chosen from any I-V relationship.

The relationship between current and voltage in this model is defined by the following equations as the state variable x of the memristor changes linearly.

$$v(t) = \left[R_{on} + \frac{R_{off} - R_{on}}{x_{off} - x_{on}}\left(x - x_{on}\right)\right]\cdot i(t) \tag{2.19}$$

The memristance changes in TEAM model depends on the state variable x exponen-

tially and current-voltage equation becomes

$$v(t) = R_{on} e^{(\lambda / x_{off} - x_{on})(x - x_{on})} \cdot i(t) \tag{2.20}$$

where λ is a fitting parameter and R_{on} and R_{off} are resistance at the boundaries, which fulfil

$$\frac{R_{off}}{R_{on}} = e^{\lambda} \tag{2.21}$$

2.2.3.6 Voltage ThrEshold Adaptive Memristor (VTEAM) Model

The VTEAM model is an extension of the TEAM model proposed by Kvatinsky et al. [42]. As opposed to the TEAM model that depends on threshold current, the VTEAM model exhibits a threshold voltage. For some logic and memory applications, a threshold voltage memristor is more suited than a threshold current memristor models. The VTEAM model combines the benefits of the TEAM model with the addition of a threshold voltage rather than a threshold current. The VTEAM model has an undefined current-voltage relationship that can be chosen freely from any of the current-voltage characteristics. The following equations represent the voltage-controlled time-invariant memristor device:

$$\frac{dw}{dt} = f(w, v) \tag{2.22}$$

$$i(t) = G(w, v)v(t) \tag{2.23}$$

where w and $v(t)$ are the state variable and voltage of the memristor device. $f(w, v)$ represents the function of the derivative of the state variable w. $G(w, v)$ is the conductance of the device and $i(t)$ represents the current passing through the device. In the VTEAM model, the derivative of the state variable is represented by Eq. 2.24

$$\frac{dw(t)}{dt} = \begin{cases} k_{\text{off}} \left(\frac{v(t)}{v_{\text{off}}} - 1 \right)^{\alpha_{\text{off}}} f_{\text{off}}(w), & 0 < v_{\text{off}} < v \\ 0, & v_{\text{on}} < v < v_{\text{off}} \\ k_{\text{on}} \left(\frac{v(t)}{v_{\text{on}}} - 1 \right)^{\alpha_{\text{on}}} f_{\text{on}}(w), & v < v_{\text{on}} < 0 \end{cases} \tag{2.24}$$

where k_{off}, k_{on}, α_{off}, α_{on} are constants. k_{off} and k_{on} represent positive and negative values. The functions $f_{off}(w)$ and $f_{on}(w)$ represent the state variable w's dependency on the derivative of the state variable. These functions act as window functions, limiting the state variable to $w \in [w_{on}, w_{off}]$ limits.

$$\frac{dx}{dt} = g\left((V(t))\, f\left(x(t)\right)\right) \tag{2.25}$$

Fig. 2.6 shows the I-V relationship of the VTEAM model for sinusoidal.

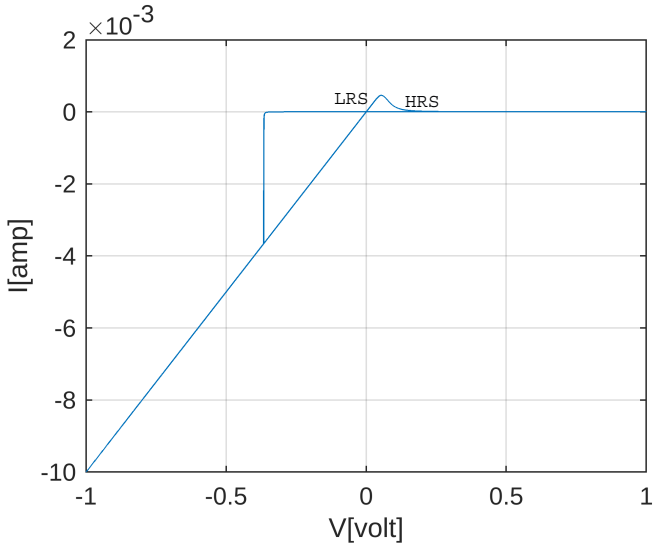


Figure 2.6: I-V characteristic of VTEAM.

```
    %legend('X/DV')
end
```

2.2.3.7 Yakopcic Memristor Model

This memristor model is proposed in [98] based on the current-voltage relationship proposed in [45]. The switching behaviour of this model is similar to the one presented in [62]. The model uses several fitting parameters to simulate the effects of the various devices due to the broad variance in the design and function of the memristor devices.

This model is designed to use in the applications where fast switching time is required like high speed neuromorphic circuits. In this thesis, most of our memristive neural network implementation is based on this memristor model. The I-V relationship of this device model is defined by

$$I(t) = \begin{cases} a_1 x(t) \sinh\left(bV(t)\right), & V(t) \geq 0 \\ a_2 x(t) \sinh\left(bV(t)\right), & V(t) < 0 \end{cases} \tag{2.26}$$

where $x(t)$ is the state variable and a_1, a_2 and b are the fitting parameters.

The I-V characteristic equation of this memristor model shown in Eq. 2.26 is based on hyperbolic sinusoid to account for the metal insulator metal junction. Beyond a specific voltage threshold, the device's conductivity increases due to hyperbolic sinusoid shape. The three parameters a_1, a_2 and b shown in Eq. 2.26 are used to determine a particular type of memristor model. The thickness of the dielectric layer in the memristor is represented by a_1 and a_2 and the parameter b is related to the conductivity of the device.

In this model, the change of the state variable depends on two functions $g(V(t))$ and $f(x(t))$ as shown in Eq. 2.27 and Eq. 2.28, Eq. 2.29. The function $g(V(t))$ is used to implement the threshold voltage that must be exceeded to cause a change in the value of the state variable. To ensure that $g(V(t))$ starts from zero once the voltage threshold is crossed, an exponential value subtracted from the equation in the simulations is used as a constant term. The voltage thresholds at the positive and negative leads of a memristor model are (V_p, V_n). Depending on the current state of the device, the function $f(x(t))$ is used for the partition of the state variable motion into two separate regions. Until the point x_p or x_n, the state variable motion is constant. At this point, the movement of the state variable is restricted by an exponential function decaying at a rate of p or n.

$$g(V(t)) = \begin{cases} A_p \left(e^{V(t)} - e^{V_p}\right), & V(t) > V_p \\ -A_n \left(e^{-V(t)} - e^{V_n}\right), & V(t) < -V_n \\ 0, & -V_n \leq V(t) \leq V_p \end{cases} \tag{2.27}$$

$$f(x) = \begin{cases} e^{-\alpha_p(x-x_p)} w_p(x, x_p), & x \geq x_p \\ 1, & x < x_p \end{cases} \tag{2.28}$$

$$f(x) = \begin{cases} e^{\alpha_n(x+x_n-1)} w_n(x, x_n), & x \leq 1 - x_n \\ 1, & x > 1 - x_n \end{cases} \tag{2.29}$$

The function $W_p(x, x_p)$ is a window function in Eq. 2.30 that assures $f(x)$ equals 0 when $x(t)$=1. When the current flow is reserved, $w_n(x, x_n)$ in Eq. 2.31 prevents $x(t)$ from becoming less than 0.

$$w_p(x, x_p) = \frac{x_p - x}{1 - x_p} + 1 \tag{2.30}$$

$$w_n(x, x_n) = \frac{x}{1 - x_n} \tag{2.31}$$

In each of the memristor devices, Eq. 2.32 is employed to simulate state variable motion. The parameter η in the equation is used to identify the direction of the state variable motion.

$$\frac{dx}{dt} = \eta g\left((V(t)\right) f\left(x(t)\right) \tag{2.32}$$

The I-V relationship of this memristor model is shown in Fig. 2.7 for sinusoidal voltage.

Table 2.2: Comparison of different memristor models.

Memristor Model	**Linear Ion Drift Model [14]**	**Non-linear Ion Drift Model [35]**	**Simmons Tunnel Barrier Model [62]**	**TEAM Model [41]**	**VTEAM Model [42]**	**Yakopcic Model [98]**
Control Structure	Current	Voltage	Current	Current	Voltage	Voltage
State Variable	$0 \leq w \leq D$	$0 \leq w \leq 1$	$a_{off} \leq w \leq a_{on}$	$x_{off} \leq w \leq x_{on}$	$w_{off} \leq w \leq w_{on}$	$0 \leq w \leq 1$
I-V Relationship	Explicit	Explicit	Ambiguous	Explicit	Undefined	Ambiguous
Threshold Existence	No	No	Practically exists	Yes	Yes	Yes
Generic	No	No	No	Yes	Yes	Yes
Accuracy	Low	Low	High	Sufficient	Sufficient	Moderate

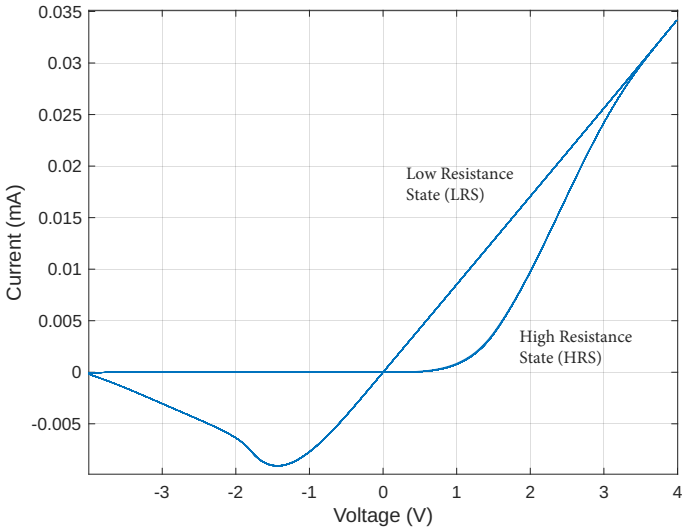


Figure 2.7: I-V characteristic of Yakopcic memristor model.

2.2.4 Artificial Neural Networks

The human brain is the inspiration of artificial neural network. Neurons and synapses are two basic elements of any neural network. In Artificial Neural Networks (ANNs), hundreds of neurons are connected with synapses and organized in single or multiple layers. Each synapse contains the weight parameter. Each neuron constitutes the weighted sum of inputs and passes the information to the threshold function as represented by Eq. 2.33. The threshold function provides non-linearity to the network. Fig. 2.8 shows a basic structure of a neural network. Weight parameters can be adjusted according to learning rule during training process. Numerous neural network structures have been developed for pattern recognition, character recognition, control signal and so on. Feedforward neural networks, Convolutional neural networks and Recurrent Hopfield neural networks are some of the examples of ANNs.

$$Y_j = F(\sum\nolimits_i X_i W_{i,j} + b) \tag{2.33}$$

X_i and $W_{i,j}$ represent input and weight for each neuron and b is the bias in the network.

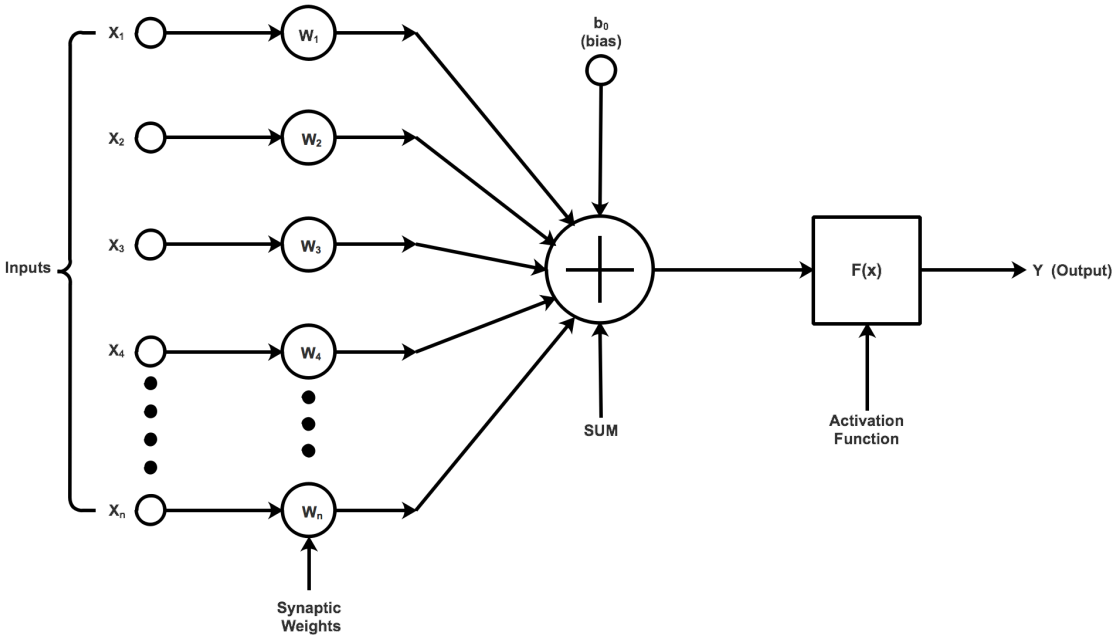


Figure 2.8: The basic structure of a neural network.

2.2.5 Memristor Crossbar Array

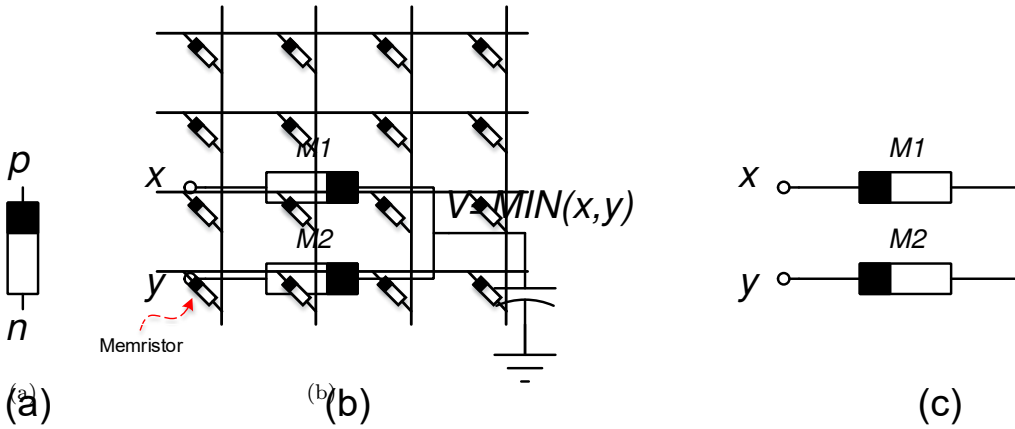


Figure 2.9: (a) Symbol of a single memristor and (b) memristor crossbar circuit.

Memristor crossbar architecture, comprises a horizontal and a vertical set of bars that lies perpendicular to on each other. A memristor is positioned between each horizontal and vertical intersection as shown in Fig. 2.9. Highly dense structure can be built using memristor crossbar due to its independent access of each device in the crossbar without any interference from other devices. One of the nanowires connects devices in the same row together and another nanowire connects devices in

the same column together. The value of each device can be programmed by applying the balanced voltage to the row (column) of the desired device and the corresponding row (column) are grounded. The remaining lines are left floating or partly biased as discussed in later sections.

2.2.6 Vector-Matrix Multiplication in the Memristor Crossbar Array

Matrix-vector multiplication in the memristor crossbar array provides high accuracy, fast computation, and low design cost [29]. In the memristor crossbar used for neuromorphic systems, any continuous range of conductance values can be stored in memristors. The memristor crossbar array is ideally suited for neural network implementation by nature. Firstly, the memristor crossbar array has the fundamental property of supporting a large number of signal connections in a small footprint, which is one of the most significant characteristics of any synapse network. Secondly, the operation that dominates in a model of a memristor-based neural network is the weighted combination of input signals, which imitates the dendritic potential [65]. The memristor crossbar structure $N \times M$ shown in Fig. 2.10(a) is assumed to demonstrate its vector-matrix multiplication functionality where $I = V \times M$ [63, 71]. Here, V is the vector of inputs denoted as $V = \{V_1, V_2 .... V_n\}$ that are applied to the rows simultaneously. The matrix M is the matrix of $N \times M$ whose coefficients are the conductance values of memristors at each crosspoint and the currents flowing through the columns are represented by the vector $I = \{I_1, I_2 ... I_m\}$. The dot product calculation at the end of each column is shown in Fig. 2.10(b). Assume that $G_{1,1}$ is the conductance value of memristor M_1 at row and column 1 and $G_{2,1}$ is the conductance of memristor M_2 at row 2 and column 1. When the voltage V_1 is applied to row R_1, a current I_1 at column G_1 is given by $I_1 = V_1 * G_{1,1}$. Similarly, if a voltage V_2 is applied to row R_2, the current I_2 in column G_1 is calculated as $I_2 = V_2 * G_{2,1}$. The total current I at the given column G_1 is given by $I = I_1 + I_2$, i.e. $V_1 * G_{1,1} + V_2 * G_{2,1}$ that is the sum of the dot product of the vectors $\{V_1, V_2\}$ and $\{G_{1,1}, G_{2,1}\}$.

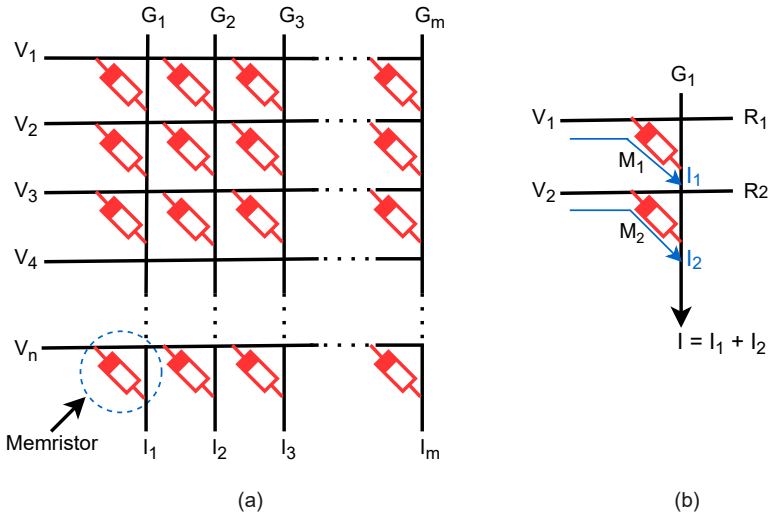


Figure 2.10: (a) A memristor crossbar based vector-matrix multiplication and (b) dot product at the bottom of each column.

2.3 Literature Review

In this thesis, we designed and implemented memristive neural circuits using our improved learning method and activation function. We also analysed the fault tolerance of memristive neural networks and evaluated the yield of the memristor crossbar array. Hence, Section 2.3.1 describes some of the existing neural circuits based on memristors. Section 2.3.2 gives a brief description of the write and read operations used in memristive neural networks. The overview of training methods for memristor-based neural networks is described in Section 2.3.3.

2.3.1 Memristor based Neural Network Circuits

A number of studies investigated memristor-based synapse, neuron, and neural network circuit designs. A number of neural architectures are proposed based on memristors. In this section, we introduce some existing memristor-based neural circuits. Some of these structures use single memristors and some use pairs of memristors to gain higher synaptic density and connectivity in the design of neural circuits. Thus, the study shows that the memristor crossbar circuits used in neuromorphic systems inherently provide high synaptic density and connectivity.

2.3.1.1 Neuromorphic Memristor Crossbar Array

For the first work on memristor-based neuromorphic designs [3, 70, 76, 77, 105], a memristor crossbar array is used to connect rows of pre-synaptic and post-synaptic neurons like a grid. Additionally, it has also been demonstrated that memristors can be used to implement Spike Time Dependent Plasticity (STDP) learning in a manner similar to biological synapses. The work proposed in [105] demonstrates that the memristor crossbar grid can be used to implement highly dense neural networks in visual image processing. Some of these studies did not use any circuit level simulator like SPICE to demonstrate, while others used EDA tools for implementation but did not mention how the sneak-paths affect the circuit level simulation in memristor-based neuromorphic networks.

2.3.1.2 Single Memristor based Memristive Neural Structures

Single memristor-based single neuron circuits that require low power have been developed for self-training [88] and supervised learning [68]. A circuit proposed in [10] represents a single synapse circuit and its simulation in HSPICE shows that it is able to operate similarly to biological learning. A Hebbian learning mechanism is used in this paper. To illustrate associative memory using a fundamental memristor circuit, a simple memristor-based neural network is developed and demonstrated in [61]. This network is implemented by simulating 2 memristors and 3 CMOS neurons using microcontrollers. All these circuits demonstrate that a memristor could be used as a synaptic weight, which is an interesting possibility. However, these publications do not explain how large multi-neuron systems based on memristors can be utilised to solve problems.

2.3.1.3 One Transistor One Memristor Circuit (1T1M)

The performance of a memristor is analysed [101] based on a 1T1M crossbar structure using SPICE simulation. The circuit is designed so that each of the memristors can be independently written to and accessed without interfering with the operation

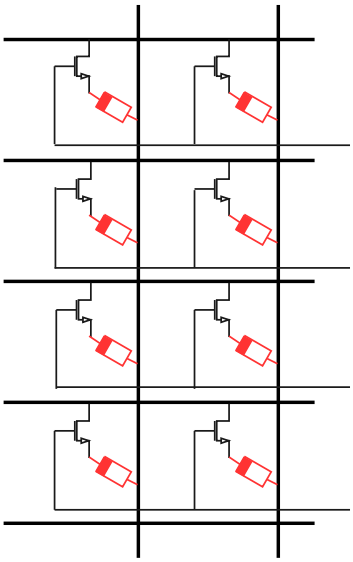


Figure 2.11: 1T1M structure.

of the other devices due to the presence of alternate current paths. A dot product engine [29] has also been developed using the 1T1M crossbar structure to execute vector-matrix multiplication in neuromorphic computing. Each crosspoint of a 1T1M crossbar circuit contains an isolation transistor that allows an individual memristor access as shown in Fig. 2.11. During write/read operations, this approach isolates/blocks unselected memristor cells such that no current flows through them. This architecture has minimum power consumption since only selected cells have current paths. Even though sneak-paths were minimised, the most significant disadvantage of this structure is the additional area required by these added transistors; which has a negative impact on the array density and footprint of the architecture. Moreover, in 3-D integration, layer-by-layer stacking is much harder to implement with the 1T1M architecture than with the 0T1M architecture [40]. The memristor neural circuit proposed in [78] uses two transistors and one memristor for a single synapse. The circuit is implemented on-chip using an online gradient decent training algorithm.

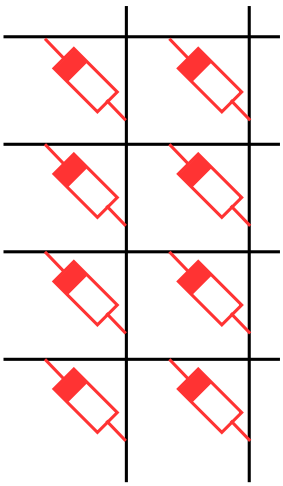


Figure 2.12: 0T1M structure.

2.3.1.4 Zero Transistor One Memristor Circuit (0T1M)

The 0T1M crossbar circuit does not contain any isolation transistors (or other devices) at the cross points. A single memristor at each cross point is used in this circuit design, as shown in Fig. 2.12. Thus, this crossbar circuit provides more density than the 1T1M crossbar structure, which is desirable for ex-situ training in memristor-based neural networks. However, the read from a 0T1M crossbar is a bit challenging because of the lack of isolation devices. Non-linear separable functions are trained using a memristor-based crossbar [96]. A multi-layer neural network is also implemented using a memristor-based crossbar circuit without any isolation device [26]. However, in-situ training is provided in these papers.

2.3.1.5 Memristor Bridge Circuits

Memristor bridge circuits are proposed [2, 38] where four identical memristors are used to store synaptic weights as shown in Fig. 2.13. Here, M1, M2, M3 and M4 represent the memristance of memristors with positive and negative polarities. V_{in} and V_{out} represent voltage applied at the input and output, respectively. Based on the sensed voltage, these bridge circuits have the ability to perform positive, zero

or negative synaptic weighting. When a strong positive or negative pulse is applied to the input, the memristance of each memristor increases or decreases based on its polarity. The advantage of this circuit is that it has the ability to provide both the polarities (positive and negative) on weighted products. However, these architectures have the disadvantage of requiring weight programming (to adjust the weights) and weight processing (via a specific circuit). The application of these circuits in a crossbar array for large-scale pattern recognition has not yet been investigated.

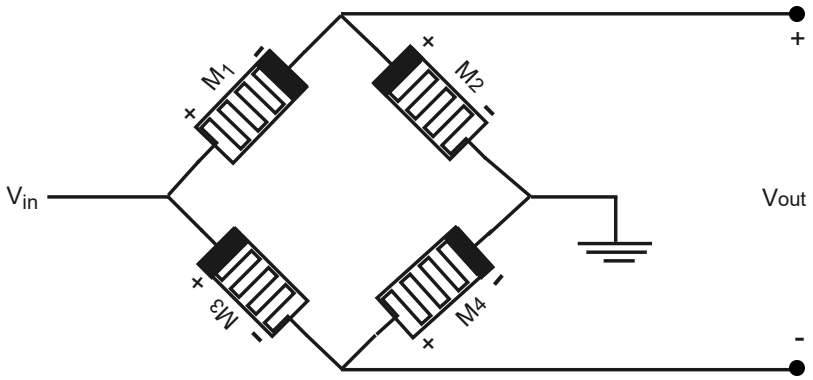


Figure 2.13: Memristor bridge circuit.

2.3.1.6 Memristive Crossbar with Constant-term Circuit

A new memristor-based crossbar array architecture is proposed [85] where a single memristor crossbar array and constant-term circuit are used to represent synaptic weight as shown in Fig. 2.14. Here, $V_{in,j}$ is the input voltage at the j^{th} row, $g_{j,k}$ represents the conductance of the memristor between the j^{th} and k^{th} column and $V_{out,k}$ is the output voltage at column k. The final output is calculated with $(1/R_b - g^{-}_{j,k})$, where $1/R_b$ is the conductance of the constant term circuit. If $g_{j,k} > 1/R_b$, it gives the negative polarity. If, on the other hand, $g_{j,k} < 1/R_b$, it gives the positive polarity. Thus, it provides both positive and negative polarity on a weight product. However, the application of these circuits in a large crossbar array has not yet been implemented.

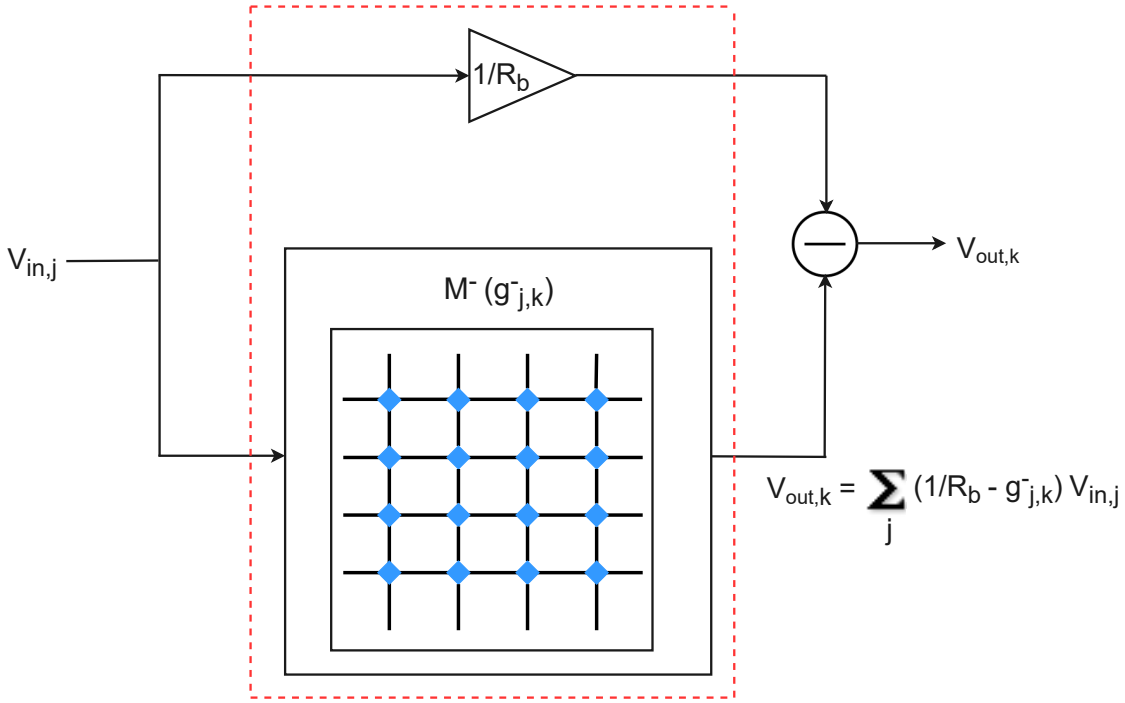


Figure 2.14: Memristor crossbar with constant-term circuit.

2.3.1.7 The Copy Memristor Crossbar

The circuit used in [47] utilized two crossbars (one original and one copy) for the same weight values. The copy crossbar is not directly connected to the input layer. One crossbar is used for error propagation in the forward direction, while the other version is utilised in the backward direction. It is difficult to save an exact copy of the memristor crossbar without a complicated feedback write mechanism, as the memristors frequently contain some degree of noise due to their switching characteristics.

2.3.2 Write and Read Operations in Memristive Crossbar Array

Programming a memristor crossbar is a bit difficult since there is a substantial amount of variation existing between different memristor devices [104]. When the identical voltage pulses are applied to write different memristors, this results in different amounts of resistance changes in these different devices in the crossbar. As a consequence of this, at each step of iteration, a new updated pulse is applied and the new

state of the memristor device is calculated. Thus, this is called an iterative feed-back writing process.

The read operation is more challenging than the write operation due to alternate current paths (called sneak-paths) in the memristor crossbars [48]. These sneak paths have more effect on large memristor crossbars than on smaller ones.

2.3.2.1 Write Operations in Memristor Crossbar

Various research [32, 63] showed that there is a significant relationship between the memristor's conductance change, voltage amplitude, and pulse width. Programming a memristor in the crossbar includes a suitable write voltage across it. The write voltage (V_w) should be greater than the threshold voltage (V_{th}) [20]. During a write operation, only the desired memristor should receive the V_w pulse. The write voltage ($V_{w/2}$) is applied to the selected row in which a memristor has to be written while the voltage ($-V_{w/2}$) is applied to the selected column and vice-versa in order to increase or decrease the value of the desired memristor as shown in Fig. 2.15. The rest of the rows and columns are set to 0 volt.

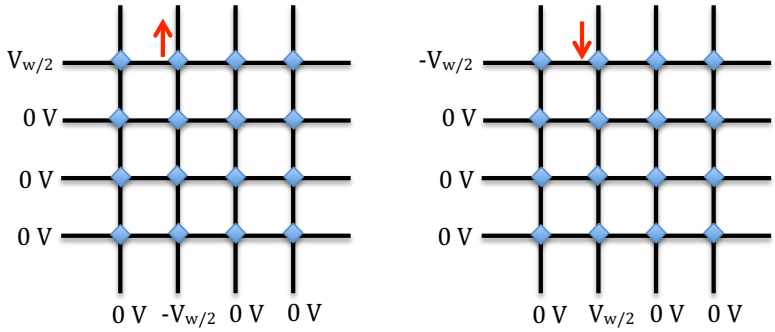


Figure 2.15: Writing a single memristor (represents with blue dot) in crossbar shows (a) increase the memristor's conductance (upward arrow) and (b) decrease the memristor's conductance (downward arrow).

The voltage shown in Fig. 2.16 is the write voltage pulses that are applied to the memristor crossbar. The V_{th} is considered as 4V [97]. It shows two write pulses, one is applied to the row and the other is applied to the column of the desired memristor.

Thus, the desired memristor achieves the voltage above threshold. The write voltage is set to 2.5V and -2.5V for row and column, respectively, to increase or decrease the memristance during write operation.

The state variable value of a memristor lies between 0 and 1, and it corresponds to the change in conductance value of each memristor. Fig. 2.17 represents the state variable value. It shows how the memristor value changes while applying the write pulses.

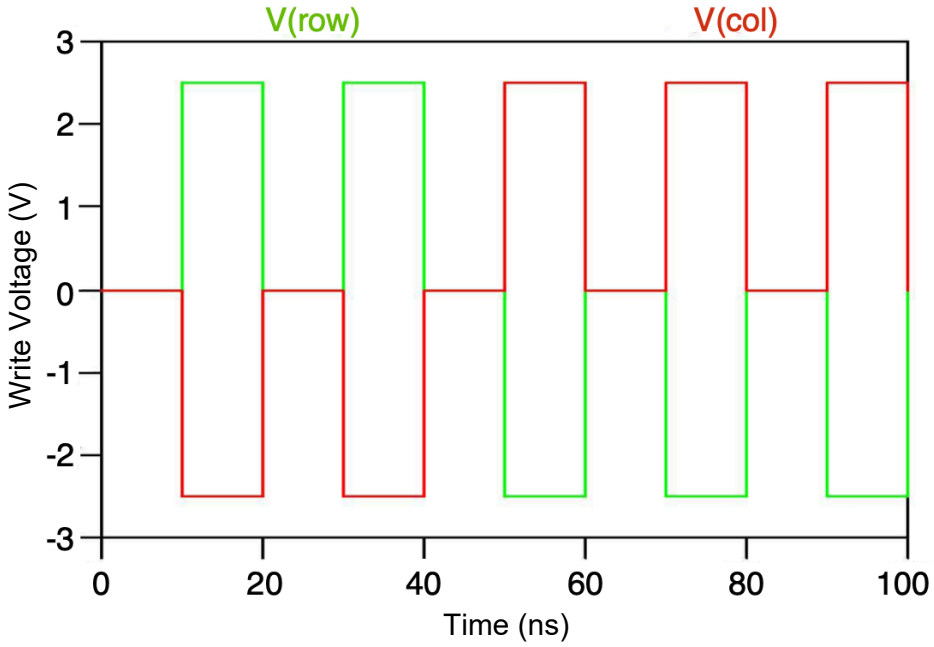


Figure 2.16: Write voltage pulses.

2.3.2.2 Read Operations in Memristor Crossbar

In this section, we described two of the reading schemes that are widely used to read memristors in the implementation of memristive neural circuits. (a) Voltage divider based read circuit (b) Transimpedance amplifier based read circuit. We also used these techniques to implement memristive neural circuits in this thesis.

Voltage Divider based Read Circuit: As we mentioned earlier, the read operation is a bit challenging compared to the write operation. The read voltage (V_r)

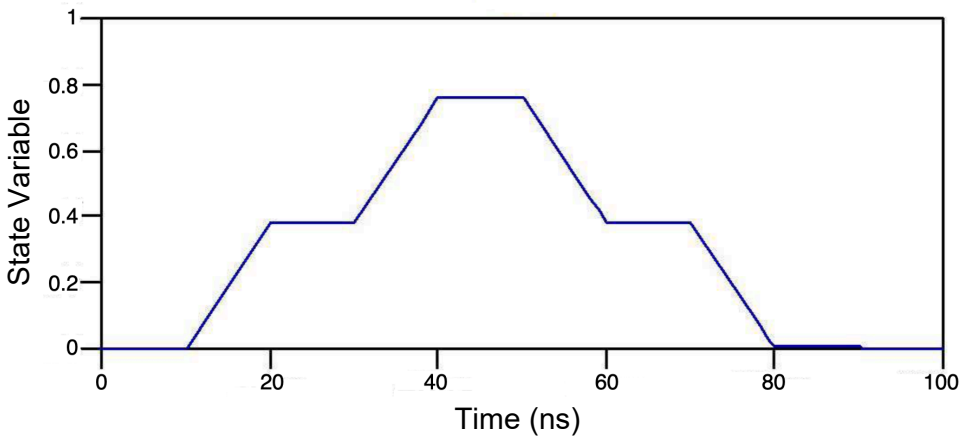


Figure 2.17: Memristor state variable value.

should be less than the memristor threshold voltage (V_{th}) so that it cannot affect the memristance value. The state of a memristor can be read by using the voltage divider sensing technique. A sense resistor (R_s) is connected in series to the memristor in order to convert the current from the memristor into a voltage signal. The read voltage (V_r) is applied to the row of the desired memristor and 0V to the rest of the rows in order to read a single memristor as shown in Fig. 2.18. The output voltage (V_o) and current (I_o) are computed according to Eq. 2.34 and Eq. 2.35. It represents the output of the voltage divider between the sense resistor and the resistance (R_m) of the memristor itself.

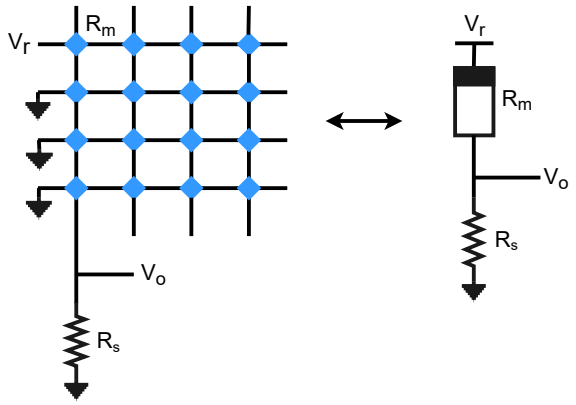


Figure 2.18: Reading the desired memristor in the crossbar using voltage divider.

$$V_o = V_r \frac{R_s}{R_s + R_m} \tag{2.34}$$

$$I_o = \frac{V_r}{R_s + R_m} \tag{2.35}$$

Read Circuit with Amplifier: In this read circuit, a transimpedance amplifier is used at the end of the corresponding column of the memristor crossbar as shown in Fig. 2.19. A read voltage, $-V_r$ is applied to the corresponding row of the memristor crossbar array and 0V is set to the rest of the columns and rows in the crossbar in order to read a single desired memristor in the crossbar array. The output voltage V_0 is calculated as shown in Eq. 2.36. Here, V_r and R_f are constants and V_o is only the function of R_m.

$$V_o = \frac{V_r R_f}{R_m} \tag{2.36}$$

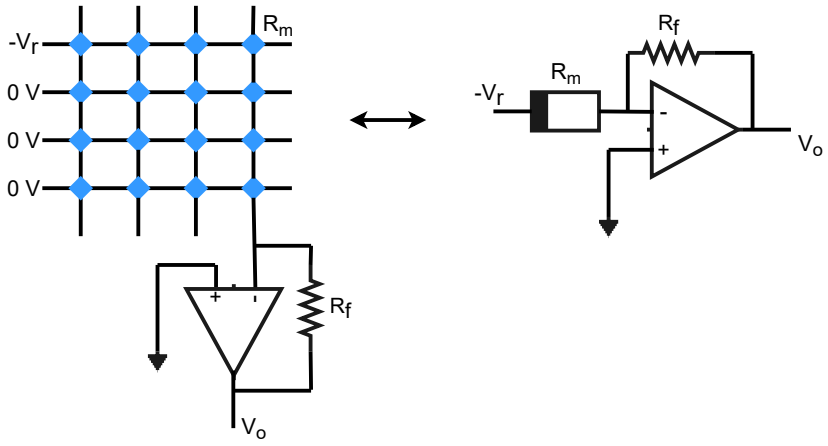


Figure 2.19: Reading the desired memristor in the crossbar using amplifier.

2.3.3 Training Methods

The training process is one of the major parts of any neural network. During the training process, a network is trained using some learning algorithms like perceptron learning [67], backpropagation [69] and the weights are updated. There are basically

two training methods that can be used to implement memristive neural networks: *ex-situ* and *in-situ* training methods [5].

2.3.3.1 Ex-situ Training

The work presented in [5, 37, 95] is based on ex-situ training. In the *ex-situ* approach, training is provided in software; weights are calculated and then calculated weights are imported into the physical memristor crossbar. Thus, an on-chip training circuit is not required in this approach. Individual memristor devices in the memristive crossbar array are programmed to set the pre-determined weight value of a memristor. A sequence of write and read pulses are applied to alter the conductance of a memristive device. The key benefits of *ex-situ* training are that any complex learning algorithm could be considered to train the network and these are easy to implement. However, it is less efficient than the *in-situ* method while importing the pre-determined weights into the networks and this approach is less tolerant to stuck at faults in crossbars. The key issue in *ex-situ* training is the programming of the memristor crossbar due to the variability of different memristive devices. Thus, applying the same voltage pulse to a similar set of memristors does not necessarily result in the memristors having similar final conductance values. Moreover, the feedback programming process [4] is required to programme the individual memristor to achieve target resistance within the precision boundary. Higher precision weight values would be required for the implementation of complex neural networks [5, 95].

2.3.3.2 In-situ Training

The work in [5, 37, 47, 78] used the in-situ training approach to implement neural circuits based on memristors. The *in-situ* method utilizes a physical memristor crossbar to represent the weights and network during the training process. The weight adjustment of all the memristor devices in the network is implemented in hardware during the training process because the training circuit is directly on the chip and the training data is used to programme the memristor crossbar. Thus, it provides a more

efficient and parallel weight update process than the *ex-situ* approach. However, updating the weight of each memristor device is a lengthy and time-consuming process. The main problems with in-situ training are the existence of diffusion in the switching behaviour of memristors and the fact that the rate of change of conductance depends not only on the voltage pulses but also on the current state of the memristor [5].

At this level of hardware memristive neural network implementation using these training methods, there is a high requirement for finding efficient mapping schemes and crossbar designs to achieve fast and accurate dot product operations. There is also a need to find more efficient ways of tuning memristor resistance values [29].

2.3.4 Activation Functions in Memristor-based Neural Circuits

A nonlinear differentiable activation function is required in any multi-layer neural network as described in Section 2.2.4. The widely used activation functions are sigmoid, tanh and the rectified linear unit (ReLU) [1, 5, 25, 26, 29]. The work in [2] used a set of transistors to perform an activation function. Along with the memristor bridge circuit, a separate circuit consisting of set of a transistors is used to implement the activation function.

The work presented in these papers [26, 29, 95] used operational amplifiers to perform activation functions. The activation function implemented in [26] closely approximates the activation function as shown in Eq. 2.37

$$f(x) = \frac{1}{1 + e^{-x}} - 0.5 \tag{2.37}$$

The activation function implemented in [29] as shown in Eq. 2.38, which closely approximates the Softmax function.

$$f(x) = \frac{1}{1 + exp(-x)} \tag{2.38}$$

The work [95] implements an activation circuit that is relatively close to the sigmoid activation function. To achieve this desirable function, an amplifier at the output of the circuit is used to activate the non-linearity of the circuit. To simulate the amplifier transfer function, a piecewise linear equation with a slope and a set of bounds is used. Most of the activation functions in these papers are implemented using operational amplifiers which incur significant hardware overhead and closely approximates the sigmoid function.

2.3.5 Fault-tolerance and Yield of Memristor-based Circuits

The performance of memristor-based neuromorphic systems can be affected by variations in conductance values, operational faults, manufacturing defects, and other factors [11, 86, 91]. In the majority of cases, neural networks can tolerate a limited number of defective synaptic weights; however, a high defect rate drastically reduces the accuracy of matrix-vector calculations. The retraining and weight mapping procedures are utilised to train the neural network containing defective memristors [12, 86, 107]. A low manufacturing yield is still a major challenge in a Resistive RAM (RRAM)-based computing system (RCS) due to the limitations of the immature fabrication technology. Researchers [11] claim that only 63% of the RRAM cells on a 4-Mb HfO_2-based RRAM test chip are fault-free, and roughly 10% of the RRAM cells on the device are affected by Stuck-at-Faults (SAFs). Due to manufacturing defects, the memristors can be stuck-at a low or high resistive state, which can degrade the performance of memristor neural circuits [11, 86, 108].

The proposed retraining techniques [12, 50] tolerate SAFs by using the inherent fault tolerance of neural networks. This strategy is useful when it is possible to retrain a fault-tolerant network with comparable recognition accuracy. However, when the proportion of faults exceeds the inherent fault tolerance of the neural network, performance will be degraded. The work in [49, 92] proposed approaches where the memristors or rows are swapped to reduce the overall errors. The rows are swapped

frequently to reduce the error. However, the approach requires one to go back if the current step leads to the worst solution.

The researchers proposed methods [11, 92] to diagnose the faulty memristors. In [11], memristor cells are tested in a way that is similar to the way conventional RAM is tested. This proposed approach is called March C* testing, which is similar to conventional March testing [9]. This method tests each cell one at a time. However, it is a time-consuming process to test large crossbars. In [92], faulty memristor cells are tested by using a combination of March testing and sneak-currents generated by stuck-at-memristors to reduce testing time. It is assumed that such currents are generated in a certain region of the crossbar. This method requires fewer cycles, up to 30% fewer than the method proposed in [11].

2.4 Summary

This chapter covered the detailed description of memristor related concepts and the artificial neural network. Existing memristor models are also discussed. Memristor-based crossbar architecture is also explained. This chapter also described the relationship between the memristor and synapse and how a memristor crossbar array can be used to compute the vector-matrix multiplication of a neural network. The literature review section of this chapter identified and briefly reviewed the existing work related to memristor-based neural circuits. The chapter also reviewed the training methods and activation circuits used for memristive neural circuit designs. Fault-tolerance and memristor-based yield evaluation are also reviewed in this section. Each of the works reviewed has advantages and disadvantages. The work in this thesis is aimed towards improving the design of memristor-based neural circuit and learning methods in order to widen the neural application area of the memristor.

Chapter 3

Baseline Research: High Level Abstraction of Memristive Neural Network

3.1 Introduction

The neuromorphic systems are gaining significant importance in an era where CMOS digital techniques are reaching physical limits due to more power consumption, chip area and complex structures [27, 28, 30]. High density and high connectivity are required to build highly efficient and fast computing neuromorphic systems [32]. The current CMOS technology does not provide high density and connectivity required for neuromorphic systems. It also increases the design complexity of neuromorphic systems as well as consumes much more power to perform neuromorphic functions. Thus, the major bottleneck in the neuromorphic systems is to build high density neural networks. Hence, the memristor appeared as a promising device in this research area due to their nanoscale size, non-volatility and low power requirements. A major benefit of using memristors is their ability to produce high density crossbar arrays.

In this chapter, high level memristor modelling in C++ is presented. Numerous ways of memristor modelling in SPICE [6, 43], Verilog A [24, 44] and MATLAB [23, 106] have been developed but barely any model exists in literature that is developed in high-level languages especially in C++. The results of the C++ tool were verified against those of the LTspice tool. A smaller memristor crossbar based neural network

(2-bit parity function) can be simulated in less than a minute by the developed C++ script. The same crossbar needs extra design, and training work in LTspice with simulation time in hours (Approx. 5 hours). As the crossbar size increases, the simulation time in LTspice increases. The key to implementing neural network using memristor model simulated in this platform is to motivate the successful simulation of massive neural networks using this platform in the near future and to reduce simulation time of large neural networks. Electronic Design Automation (EDA) tools increase the complexity of connectivity of massive neural networks and take more time to simulate. As the size of the memristor crossbar increases, the simulation time also increases. Implementation of complex learning algorithms is another major problem in hardware neural network. For example, backpropagation algorithm is very useful learning scheme in neural networks. However, its implementation in hardware neural network is difficult due to synapse design and computation complexity.

The chapter starts by describing the characteristics of memristor as synapse in Section 3.2. It also explains the memristive structures and how these structures are utilised to implement dense neural networks. The Section 3.3 proposes the implementation of high level memristor models in C++. A single layer and multi-layer neural networks are simulated and evaluated using C++ memristor model and demonstrated the results in Section 3.4. The Section 3.4.3 presents pattern classification and the results. The chapter is concluded in Section 3.5.

3.2 Memristor as a Synapse

The relationship between memristor and Spike Time Dependent Plasticity (STDP) learning was first introduced in [3, 77]. Research in [32, 61] illustrates that memristors have the potentiality to associate memory and represent the learning and memory effects in neural network due to their non-volatility. Memristors can preserve the training output for a long time even without power source and imitate the behaviour of biological synapse in brain. A memristor was fabricated with analog

switching behaviour in [32] and it was demonstrated that it can be used as synapse in neuromorphic applications. The conductance of the memristor can be altered by directing charge or flux through it. Using high density crossbar grid, memristor can achieve equal or higher synaptic density than that of brain tissue [77] due to its physical layout. Memristor crossbar structure has the capability of matrix storage and vector-matrix multiplication. Various neuromorphic systems are developed by using memristive crossbar circuits in [13, 100] which illustrates the potential of memristive crossbar structure in massive neural network implementation. Hence, memristors can be effectively used to build neural networks with a large number of neurons and synaptic connections.

Numerous architectures are being proposed for the implementation of neural network. A hardware architecture for multilayer neural networks is proposed in [2] based on memristor bridge-synapse. Memristor bridge-synapse [38] considered as a problem solver of non-volatile weight storage in analog neural network implementation and is able to perform the sign synaptic weighting. The research work in [63, 73] experimentally demonstrates the fully operational artificial neural network and effective implementation of the locally competitive algorithm (LCA) for feature extraction based on integrated memristor crossbar array.

3.3 High Level Memristor Models Simulation using C++

In this section, we simulated different memristor models with C++.

3.3.1 Memristor Model with Non-linear Ion Drift

We have implemented C++ memristor model based on the model proposed in [6]. We also used this model to implement memristive neural networks further in this chapter. The Eq. 3.1 to Eq. 3.5 describe the mathematical modelling of this device.

$$V(t) = M(x)i(t) \tag{3.1}$$

$$M(x) = R_{\text{on}}(x) + R_{\text{off}}(1 - x) \tag{3.2}$$

Where,

$$x = \frac{w}{D}\epsilon(0, 1) \tag{3.3}$$

$$\frac{dx}{dt} = Ki(t)f(x) \tag{3.4}$$

$$K = \frac{\mu v R_{\text{on}}}{D^2} \tag{3.5}$$

$$f(x) = 1 - (x - stp(-i(t))^{2p} \tag{3.6}$$

Where,

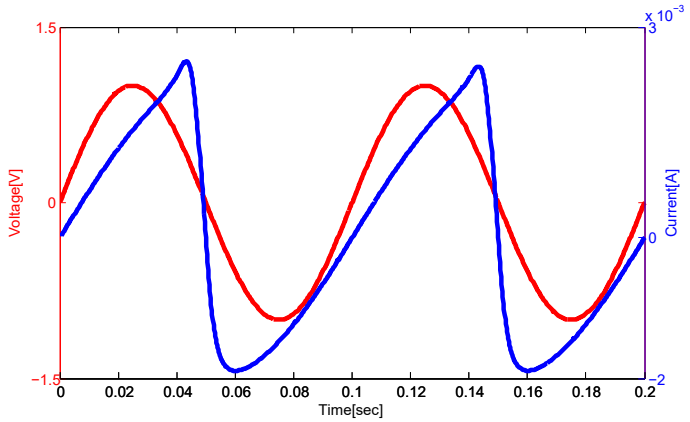
$$stp(i) = \begin{cases} 1, & i \geq 0 \\ 0, & i < 0 \end{cases} \tag{3.7}$$

$x\epsilon[0, 1]$ will drift non-linearly when w is close to 0 or D. i is the current in a memristor device. stp is a step function which behaves according to the bias of voltage applied. p is a positive integer. As the value of p increases, the difference between the linear and nonlinear drift models vanishes. The memristor model presented in [6] yields the memristance values between R_{on} and R_{off} in continuous fashion. Hence, it can be used to implement an analog synapse.

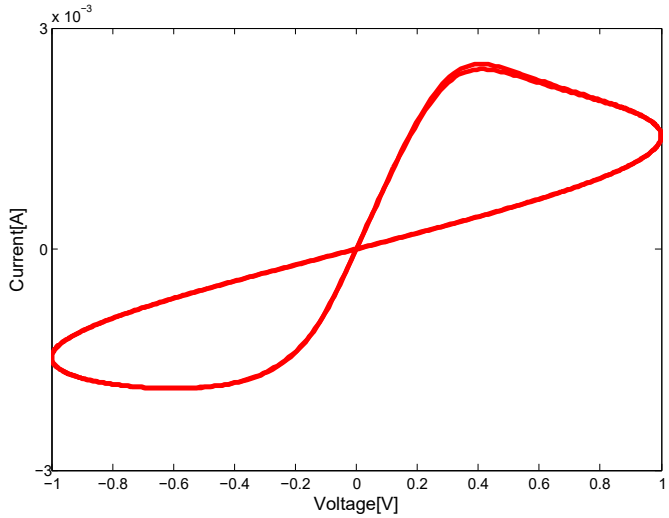
Fig. 3.1 represents the I-V curve obtained through C++ modelling for memristor device described above. The sinusoidal voltage has been used as input to attain this curve and considered R_{on} as 100Ω and R_{off} as 1KΩ. Fig. 3.2 shows the C++ code for the memristor model equations mentioned above.

3.3.2 Yakopcic Memristor Model

This memristor model is proposed in [98] based on the current-voltage relationship presented in [45]. The Eq. 3.8 shows the I-V characteristic of this memristor model. The detailed description of this model is given in Section 2.2.3.6.



(a)



(b)

Figure 3.1: (a) Input voltage and current pulses and (b) simulated I-V curve for given voltage.

$$I(t) = \begin{cases} a_1 x(t) \sinh\left(bV(t)\right), & V(t) \geq 0 \\ a_2 x(t) \sinh\left(bV(t)\right), & V(t) < 0 \end{cases} \tag{3.8}$$

We have simulated this model with C++ and the I-V curve generated by $C++$ modelling for this memristor device is shown in Fig. 3.3(b). As an input, the sinusoidal voltage shown in Fig. 3.3(a) is applied.

3.3.3 Voltage ThrEshold Adaptive Memristor (VTEAM) Model

To characterize the behaviour of voltage-controlled memristors, A Voltage ThrEshold Adaptive Memristor (VTEAM) Model is proposed by Kvatinsky et al. [42]. This

$$x = \frac{w}{D}\epsilon(0,1) \tag{4}$$

$$dx/dt = Ki(t)f(x) \tag{5}$$

$$K = \frac{\mu v R_{\text{on}}}{D^2} \tag{6}$$

ıe relationship between current and voltage. s the value of current through memristor sents the input voltage of memristor. $M(x)$ istance of memristor $i.e.$ the sum of doped on of the memristor. w is the width of D is the total length of the device. The R_{off} represent the minimum and maximum device respectively.The value of the state valuated using Eq. (5) where μv represents ıe ions. An extensive electrical field can mall voltages. As a result, it can produce arities in ionic transport. Hence, to model tion and non-linear drift, a window function own in Eq. (7).

$$c) = 1 - (x - stp(-i(t))^{2p} \tag{7}$$

n-linearly when w is close to 0 or D. del in [12] yields the memristance values R_{off} in continuous fashion. Hence, it can be an analog synapse.

nts the I-V curve obtained through C++ nristor device described above. The sinu- been used as input to attain this curve and 100Ω and R_{off} as $1\text{K}\Omega$. The listing 1 shows the memristor model equations mentioned

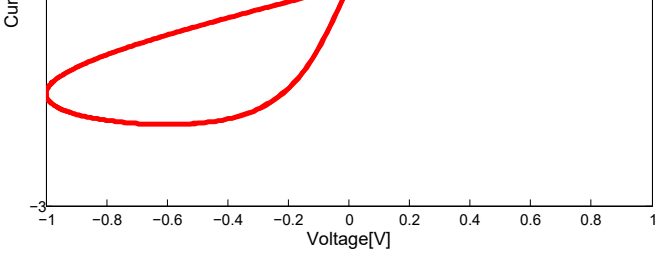


Fig. 2: Simulated I-V curve for given voltage

```
void nonlinear()
{
  double t0=0;
  float X0=0.076;   // initial value of state variable
  vector<double> X;
  X.push_back(X0);
  vector<double> t;
  t.push_back(t0);
  int nsteps=1000;
  double T=0.2;
  B=(D*D)/UV;
  K=Ron/B;      //constant mentioned in Biolek model

  double dt=(double)T/(nsteps);

  for(int i=0; i<nsteps;  i++)
   {
     t.push_back(t[i] + dt);
   }
  for(int i=0; i<t.size();  i++)
   {
     V.push_back(1*sin(2*PI*t[i]*10));  //input voltage

     M.push_back((Ron*X[i])+(Roff*(1-X[i]))); //memristance

     I.push_back((1*sin(2*PI*t[i]*10))/
               ((Ron*X[i])+(Roff*(1-X[i]))));  //current

     W.push_back(wfunction(X[i],p,I[i])); //window function

     dx.push_back(dt*K*I[i]*W[i]);

     X.push_back(X[i] + dx[i]);  //state variable
   }
}
```

Listing 1: C++ Code for Memristor Model Equations

Figure 3.2: C++ code for memristor model equations.

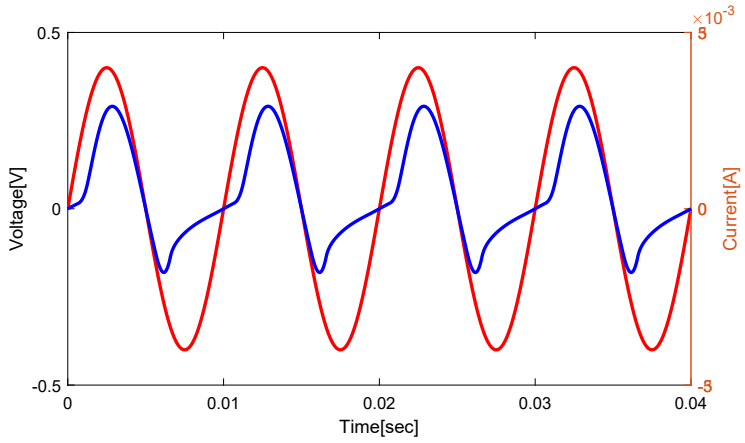
model is explained in detail in Section 2.2.3.6. The current-voltage relationship for this model is represented by:

$$i(t) = G(w, v)v(t) \tag{3.9}$$

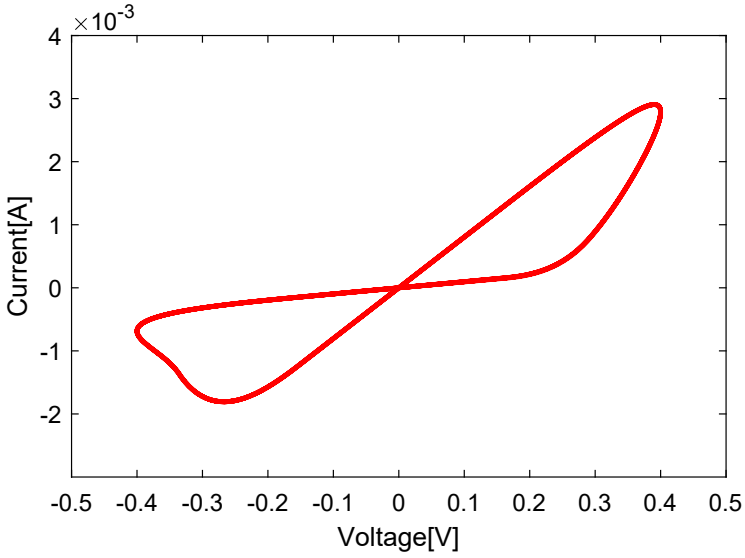
Fig. 3.4(b) shows the I-V curve simulated with C++ for the VTEAM model for a given voltage shown in Fig. 3.4(a).

3.4 Memristor based Neural Network Simulation

This section explains the simulation of single-layer and multi-layer neural networks. We performed different linear and non-linear separable functions on these networks. For this implementation, we used high level memristor modelling in C++ as proposed in the previous section.



(a)

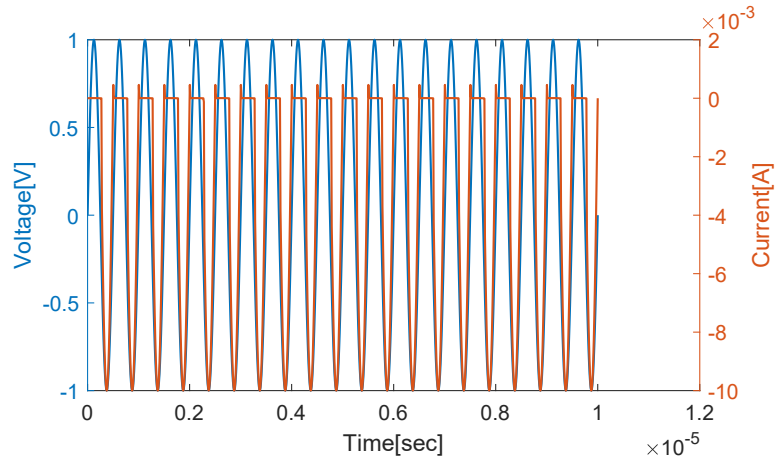


(b)

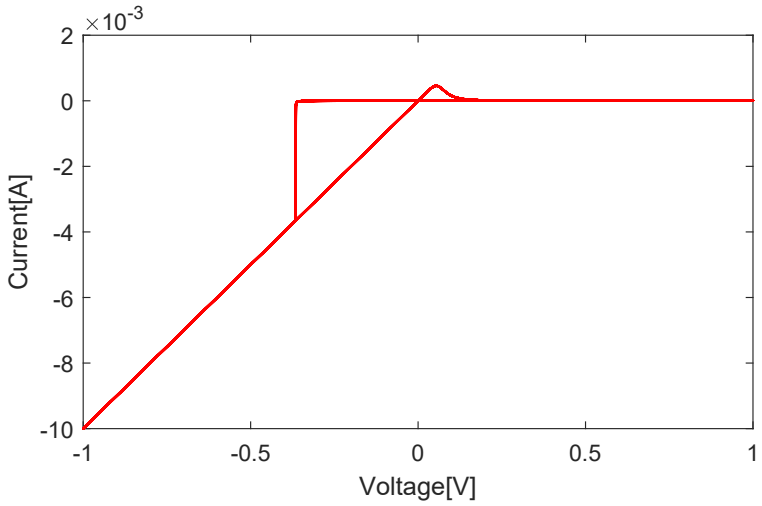
Figure 3.3: (a) Input voltage and current pulses and (b) simulated I-V curve with C++.

3.4.1 Single Layer Network Simulation

This section illustrates the classification of linear functions using a single-layer neural network. Fig. 3.5 shows a single-layer perceptron network with two inputs and one output. Each synapse contains some weight value. Then, each neuron multiplies the input value with its correspondance weight value and passes the result to the next neuron. The neuron at the output layer performs the activation function on the sum product of input and weight values.



(a)



(b)

Figure 3.4: (a) Input voltage and current pulses and (b) I-V curve simulated for VTEAM.

Let X_i represent the input values and $W_{j,i}$ represent the weight values. Then the dot product is calculated by:

$$O_j = \sum_i X_i W_{j,i} \tag{3.10}$$

and the final neuron output is given by:

$$Y = f(O_j) \tag{3.11}$$

In Eq. 3.11, f is the activation function. Fig. 3.6 displays the corresponding

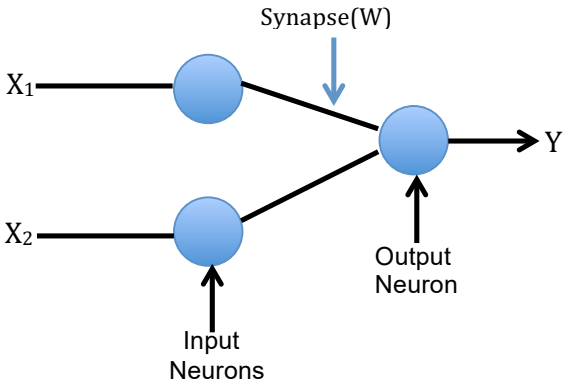


Figure 3.5: A single layer perceptron network.

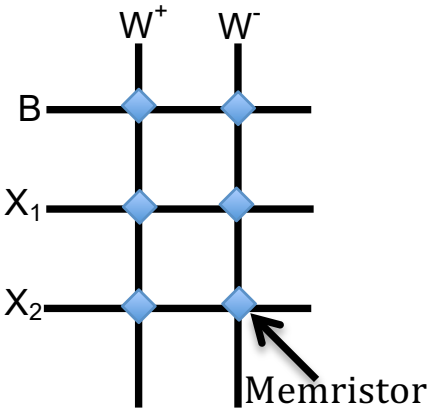


Figure 3.6: Memristive crossbar schematic for single layer perceptron network.

memristive crossbar circuit for a single-layer perceptron neural network. The rows represent the input values and the columns the outputs. X_1 and X_2 are inputs and B is considered as bias in a memristive crossbar. Memristor acts as a synapse and interconnects each row with a column. In a memristive crossbar, two memristors are used for a single synapse so that a negative effect can be presented on total weight values. Conductance values are considered as weights in the memristor based neural network. Weight values are calculated and updated according to the conductance values of memristors in a neural network. The value of W in the memristive crossbar circuit is given by:

$$W = W^+ - W^- \tag{3.12}$$

The C++ software environment was adopted to develop and train the network. We provided the training for this network to learn 2-bit AND and OR logic operations. The memristor model in [6] was used due to its simplicity and popularity in literature. To reduce simulation time, we trained the network as a crossbar array structure. A perceptron learning algorithm was applied to train a single perceptron crossbar neural network. Eq. 3.13 and Eq. 3.14 were used to calculate the error and weight update.

$$E = A_j - Y_j \tag{3.13}$$

$$\Delta W = \eta E X_{i,j} \tag{3.14}$$

In 3.14, η is a constant learning function, and E represents an error function having values of either +1, -1, or 0 (corresponding to weight increase, decrease, or no change, respectively). $X_{i,j}$ represents the input value according to row and coloumn. Y_j and A_j are the final and desired outputs of the network. Fig. 3.7 and Fig. 3.8 represent the results of AND and OR operations. The network learned AND function in just 4 epochs and OR function in 5 epochs.

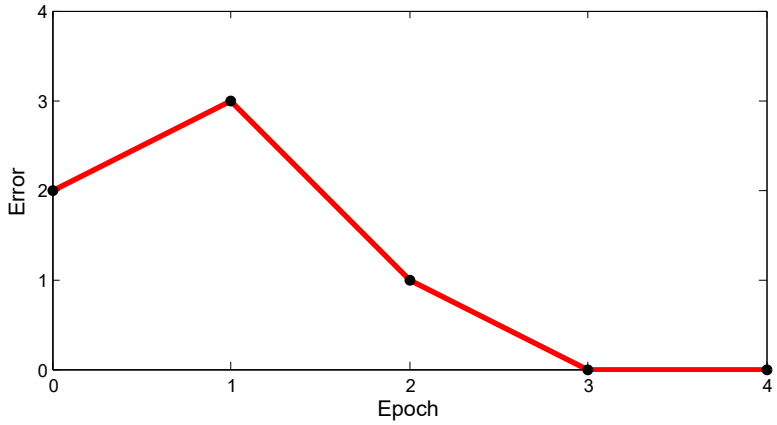


Figure 3.7: Training error in each epoch for AND logic function.

3.4.2 Multi-layer Network Simulation

A multi-layer neural network is required for the classification of non-linear separable functions. We implemented a multi-layer neural network with one and two hidden

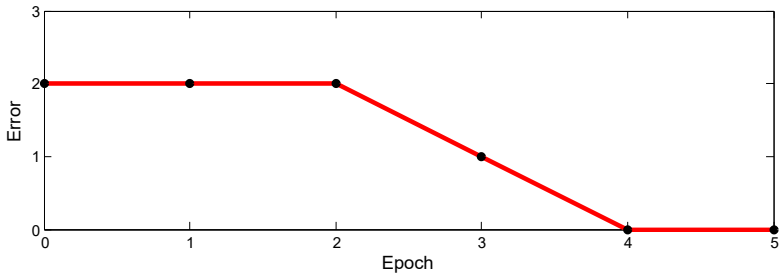


Figure 3.8: Training error per epoch for OR logic function.

layers to classify 2-bit or 3-bit parity functions. Different learning algorithms are used for training.

3.4.2.1 Two-layer Network Simulation

This section explains the simulation of a two-layer neural network to perform non-linear separable operations. Fig. 3.9 displays a two-layer neural network. The network contains one hidden layer, consisting of four hidden neurons. This network has been trained to classify a 2-bit XOR function.

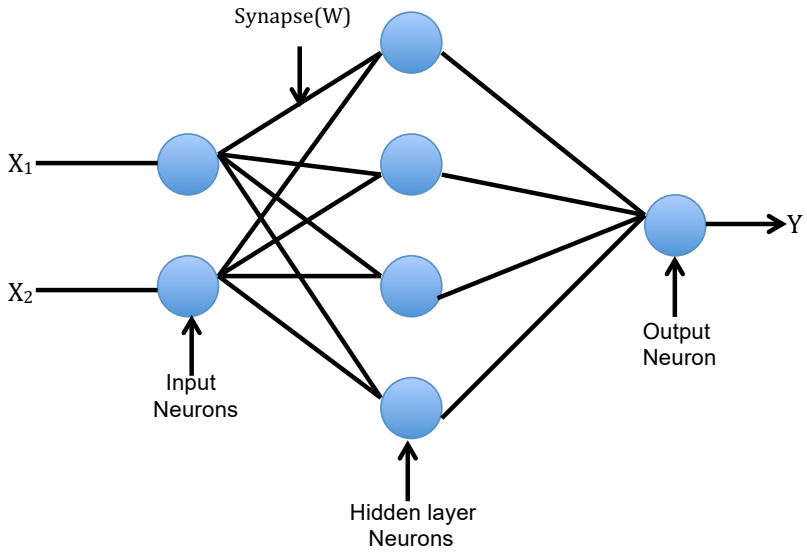


Figure 3.9: Two-layer neural network.

The activation function and weight update function from the Concurrent Learning Algorithm (CLA) proposed in [51] were adopted. CLA is a biologically inspired supervised learning algorithm that employs gradient descent to minimize the network

error and it helps to speed up the training process. The derivative of the arc tangent is used as the activation function. Eq. 3.15 and Eq. 3.16 are used to perform activation function and weight update while providing the training to a multi-layer neural network.

$$Y = \frac{1}{1+O_j^2} \tag{3.15}$$

$$\Delta W = \eta E \frac{1}{1+O_j^2} X_{i,j} \tag{3.16}$$

Fig. 3.10 shows the error at the output layer throughout the training process of a non-linear separable function. After seven epochs, the error reaches zero and the network is fully learned. From the results of the analysis, it is demonstrated that training in a two-layer network using a C++ memristor model successfully classifies the non-linear separable functions.

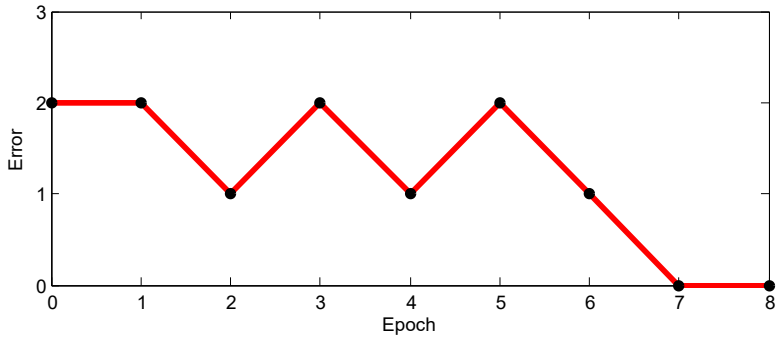


Figure 3.10: Error at output layer in each epoch throughout the training process of XOR pattern.

3.4.2.2 Three-layer Network Simulation

This section describes the implementation of a three-layer neural network to perform a three-bit parity function. Fig. 3.11 displays a three-layer neural network. The network contains two hidden layers. As shown in Fig. 3.11, hidden-layer-1 consists of four hidden neurons and hidden-layer-2 contains two hidden neurons. We used this three-layer network to perform non-linear separable functions. We trained a network

of 3 input × 4 hidden-1 × 2 hidden-2 × 1 output nodes to classify the 3-bit parity function.

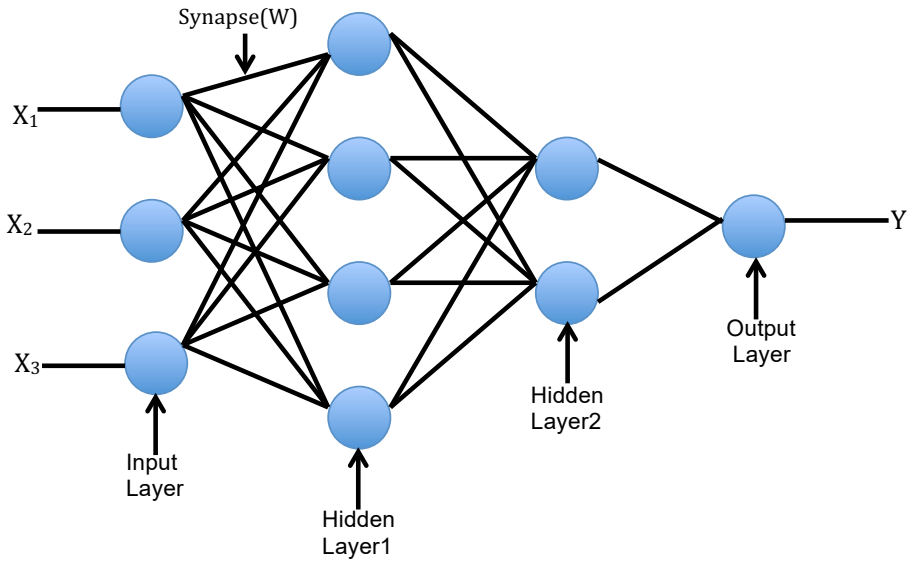


Figure 3.11: Three-layer neural network.

The backpropagation learning algorithm [69] has been used to train this network. Backpropagation is one of the powerful algorithms in artificial neural networks. However, its implementation is difficult in hardware. The Following are the steps of this algorithm that have been used.

*Step*1: Feedforward Computation: Calculate the dot product by multiplying input values with their corresponding weight values. Then, Apply the activation function to the dot product and calculate the output at each layer by using the following equations:

$$O_j = \sum_i X_i W_{j,i}$$
$$Y_j = f(O_j) \tag{3.17}$$

Here, i and j represent input and hidden layer1 nodes. Eq. 4.4 shows the output and activation function at hidden layer1.

The following 4.6 represents the output and activation function at hidden layer2.

$$O_k = \sum_j Y_j W_{k,j}$$
$$Y_k = f(O_k) \tag{3.18}$$

Here, Y_j is the output of the previous layer and act as input at this layer. $W_{k,j}$ shows the weight values between the hidden-layer-1 and hidden-layer-2.

The following Eq. 3.19 represents the output and activation function at the output layer which is represented by o.

$$O_o = \sum_k Y_k W_{o,k}$$
$$Y_o = f(O_o) \tag{3.19}$$

Here, Y_k is the output of the hidden-layer-2 and considered as an input to the output layer. $W_{o,k}$ is the weight value between the hidden-layer-2 and the output layer.

*Step*2: Backpropagation to the output layer: Calculate the error at output layer based on the following equation:

$$E_o = A_o - Y_o \tag{3.20}$$

Here, E_o is the error at the output layer, and A_o and Y_o represent the actual and desired outputs and backpropagate the error to the previous layer.

*Step*3: Backpropagation to the hidden-layer-2: Determine the error at the hidden-layer-2 using the following equation:

$$E_{h2} = \sum_j E_o W_{j,k} \tag{3.21}$$

Here, E_{h2} is the error at the output layer. $W_{j,k}$ represents the weight values of synapse connecting k with previous layer neuron j and E_o is the error which is back propagated from the output layer.

*Step*4: Backpropagation to the hidden-layer-1: Calculate Error at the hidden-layer-1 using following equation:

$$E_{h1} = \sum_i E_{h2} W_{i,j} \tag{3.22}$$

Here, E_{h1} is the error at hidden-layer-2. $W_{i,j}$ represents the weight values of synapse connecting j with previous layer neuron i and E_{h2} back propagates error from the hidden-layer-2

*Step*5: Weight updates: Update the weight using following equations at each layer:

$$\Delta W_{o,k} = \eta E_o Y_o Y_{o,k} \tag{3.23}$$

$$\Delta W_{k,j} = \eta E_{h2} Y_k Y_{k,j} \tag{3.24}$$

$$\Delta W_{j,i} = \eta E_{h1} Y_j Y_{j,i} \tag{3.25}$$

The above mentioned Eq. 3.23, Eq. 3.24 and Eq. 3.25 are used to update the weights connected with the output layer, hidden-layer-2 and hidden-layer-1 respectively.

*Step*6: All the steps are repeated until the error becomes zero.

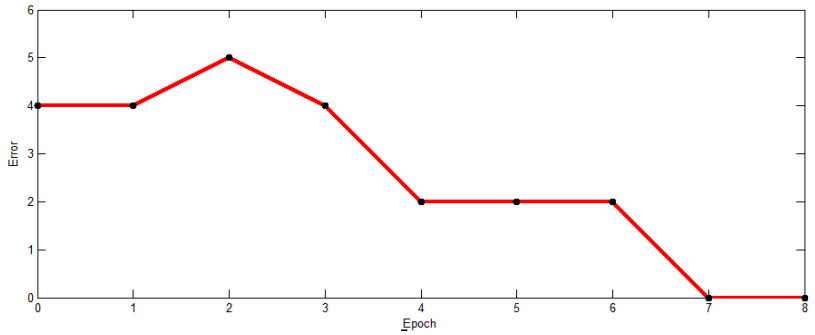


Figure 3.12: Epoch required to learn 3-bit parity function.

Fig. 3.12 shows the number of epochs required to learn three-bit parity function. As shown Fig. 3.12, it takes seven epochs to learn. After seven epochs, the error becomes zero and the network has learnt the three-bit parity function successfully.

Fig. 3.13 displays the errors at each layer in the three-layer network during the training process. Due to maximum connections, hidden-layer-1 shows the maximum number of errors as compared to hidden-layer-2 and the output layer. Hidden-layer-2 has more connections than the output layer, so it has more errors than the output

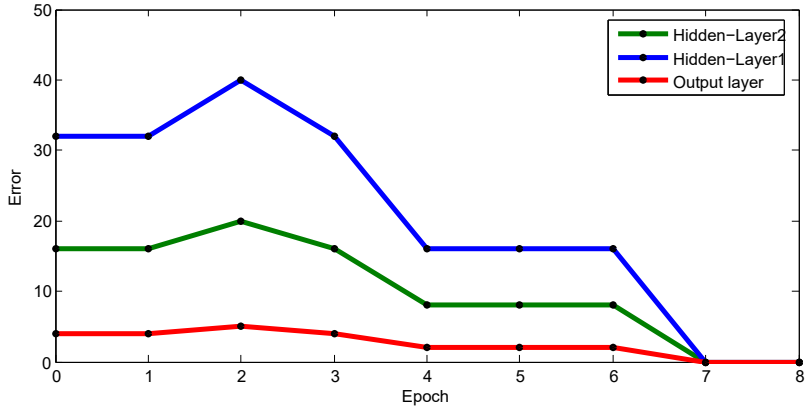


Figure 3.13: Error at each layer while training 3-bit parity function on three-layer network.

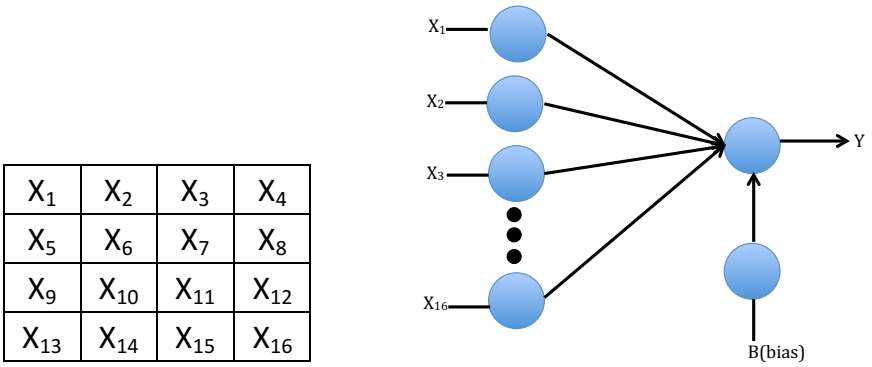
layer. Hence, more connections imply more weights and more weights contain more errors.

Our simulation results successfully demonstrate that high level memristor modelling could be used to train and simulate the larger neural networks. It shows the successful results with the backpropagation learning algorithm. Hence, any complex learning algorithm used in software training could be used to train the network through this modelling.

3.4.3 Simulation of Pattern Classifier

This section presents pattern classification for 4×4 binary images using a single-layer perceptron network. The architecture used to implement this pattern classifier is based on the memristive crossbar array-based schematic explained in Section 3.4.1. The network has been trained through perceptron learning algorithm with varying constant learning rates. The learning rate has a significant effect on training speed and generalization accuracy while using different learning algorithms for training [90]. Thus, it is important to choose a learning rate efficiently for the network as it can maximize the accuracy of the network. Fig. 3.14 displays 4×4 binary pixel image and single layer perceptron network for pattern classification.

The network consists of 16 binary inputs and one bias input. The value of bias is fixed and considered as +1, whereas the values of inputs and outputs are +1 or -1.



(a) (b)

Figure 3.14: (a) 4×4 binary image and (b) single layer perceptron network for pattern classification.

The set of patterns considered for classification in Fig. 3.15 and Fig. 3.16 represent the alphabets 'F' and 'J' and their noisy versions. First patterns in Fig. 3.15 and Fig. 3.16 are ideal patterns for letters 'F' and 'J' and rest are noisy patterns. The training set contained 50 patterns and some of these patterns are repeated.

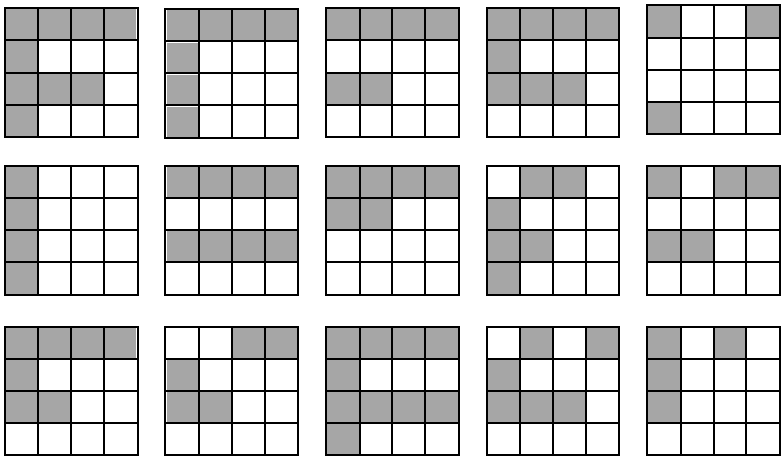


Figure 3.15: Set of input patterns for 'F'.

The grey and white pixels of images are represented by +1 and -1 respectively and the output value for pattern 'F' is considered as +1 and 'J' is -1. Fig. 3.17 shows the number of epochs required for average error to reach zero at six different constant learning rates. The network is learned only in 3 epochs at learning rate 0.02, 0.03 and 0.04 and error became zero as shown in Fig. 3.17. The network is learned in 18 epochs, 12 epochs and 7 epochs at learning rate 0.002, 0.003 and 0.004 respectively.

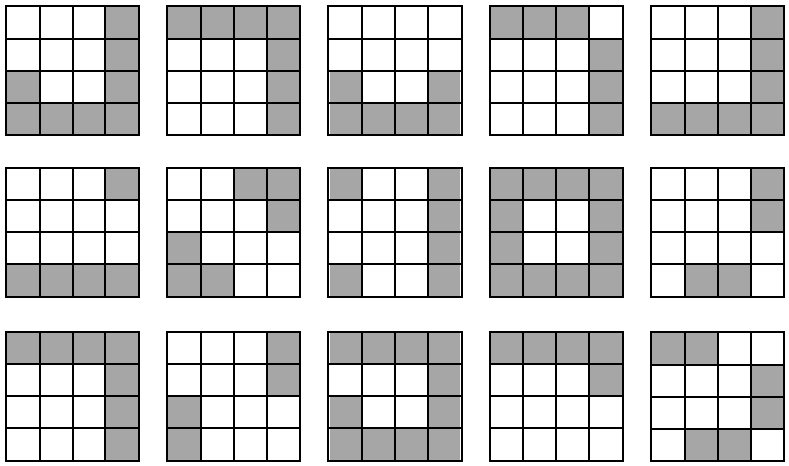


Figure 3.16: Set of input patterns for 'J'.

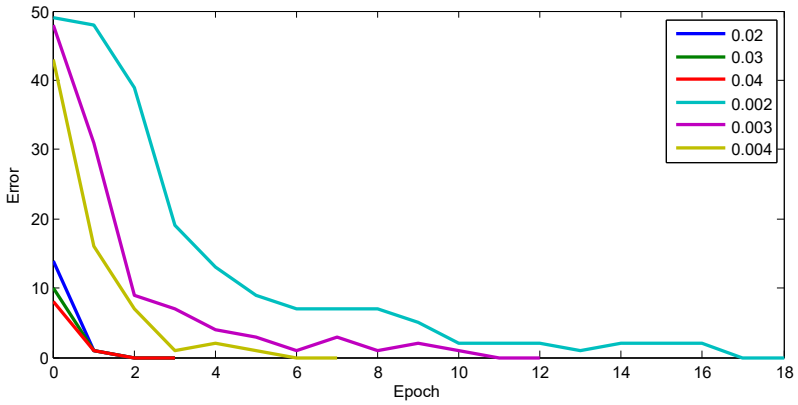


Figure 3.17: Number of epochs required for average error to reach zero at six different learning rates.

The results shows that network is successfully learned the patterns at six different learning rates.

After training, the network was tested using test data sets. Fig. 3.18 shows the test pattern sets that were used to check the performance of the network. The tested data set contained 14 test patterns of 'F' and 'J'. The patterns considered for testing are different from the training data set. The Fig. 3.19b illustrates the performance of the network during the testing process. Moreover, the results show the generalization accuracy of the network at different learning rates that have been chosen to train the network. The general idea of performing the training at different learning rates is to check the compatibility of our proposed modelling and to find the fastest and most

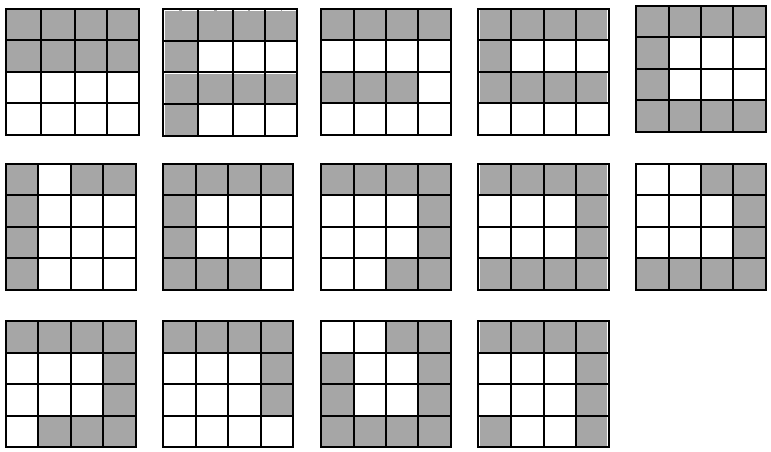
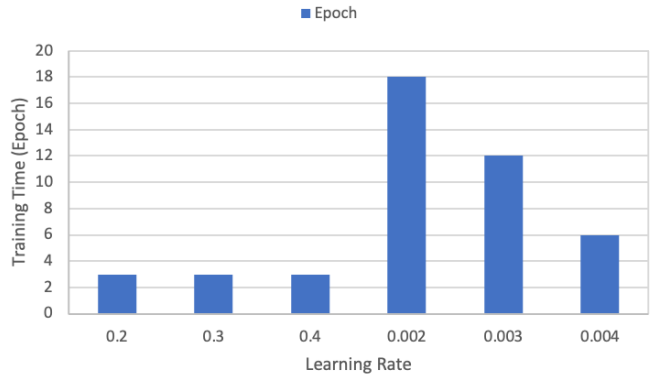
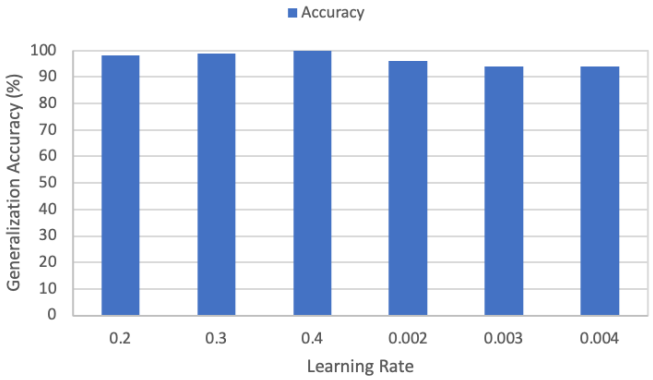


Figure 3.18: Set of patterns for testing process.



(a)



(b)

Figure 3.19: (a) Number of Epochs required to learn pattern classifier and (b) Generalization accuracy of pattern classifier.

accurate learning rate for this pattern classifier. The guidelines provided in [90] are used to determine the efficient learning rate. The network worked satisfactorily at

all the learning rates. The learning rate 0.004 has less total sum squared error(tss) than other two smaller learning rates. However, the results show that the learning rate of 0.002, 0.003 and 0.004 require more training time as well as provide less generalization accuracy than 0.02, 0.03 and 0.04 learning rates, so these could be ignored. The learning rate of 0.04 takes the same amount of training time as 0.03 and 0.02. However, it has less total sum squared(tss) error and provides maximum generalization accuracy. Thus, it is considered the most efficient and fastest learning rate for this pattern classifier.

3.5 Summary

A better understanding of the relationship between the memristor and biological synapse is essential for using the memristors in the implementation of memristor based neural networks. This chapter begins by introducing the properties of the memristor as a synapse. A novel implementation of memristor models in a high-level language is proposed. C++ is used to simulate these models. To the best of our knowledge, there is no memristor model implemented with C++ in literature. We used the memristive crossbar architecture to simulate and train single-layer and multi-layer neural networks using C++ memristor modelling. The linear and non-linear separable problems are simulated based on a memristor based crossbar array using C++. Different training algorithms are used to train different networks. We also simulated the pattern classifier at different learning rates. The aim of using this environment is to reduce the simulation time of large-scale neural networks based on memristors. Using this environment, the circuit simulated faster than with EDA tools. The simulation results successfully demonstrated the classification of linear and non-linearly separable problems such as AND, OR and XOR logic functions as well as patterns using high-level C++ memristor modelling in the simulation of neural networks.

Chapter 4

Learning Method for Ex-situ Training of Memristor Crossbar based Multi-layer Neural Network

4.1 Introduction

The Memristor is being considered as a game changer for the realization of neuromorphic hardware systems due to its similarity with biological synapse as we discussed in Section 3.2. Recent studies have shown that the physical layout, nano-scale size, and low power use of the memristor crossbar make it a good choice for hardware implementation of neural networks with high density and high performance at low power. This chapter proposes a learning method for implementing memristive multi-layer neural networks using an ex-situ training method. We mimic the behaviour of the memristor crossbar in software training process to achieve more accurate and close computations to hardware. We use voltage-divider sensing technique to calculate the dot product in this method. To demonstrate the accuracy and effectiveness of this method, different patterns and non-separable functions are implemented using memristor crossbar structures. The results demonstrate that more accurate computations can be produced using this learning method for ex-situ. It also reduces the learning time of functions.

In order to train any neural network, specific training methods are required. The weight value of each neuron in the neural network is calculated and updated using

the training methods. Two basic training methods are used to train the networks: ex-situ and in-situ. During the ex-situ training process, the training is provided in software; final weights are calculated and imported into the hardware by programming individual memristor. In the in-situ method, the weight value of each memristor device has been updated after each iteration during the training process in hardware. The key benefit of the ex-situ method is that any learning algorithm can be used for training, which is difficult in the in-situ method. However, achieving accuracy in ex-situ is difficult due to a lack of efficient mapping schemes and device variability. Section 2.3.3 describes these methods in more detail.

Recent studies show that different mapping schemes and neuromorphic circuits are proposed for ex-situ to achieve more accurate computations between hardware and software [29, 95]. This chapter presents the modified training method for ex-situ. The memristor crossbar behaviour has been studied deeply from literature and we tried to model the same behaviour of the crossbar in software training. We calculated the dot product using a voltage divider so that software computations could be matched more closely to hardware computations and more accuracy could be achieved. Thus, the network trained in software mimics the memristor crossbar in hardware.

The rest of the chapter is organized as follows: Section 4.2 describes the neuron circuit, the proposed learning method and the experimental setup for the implementation of a memristive neural network based on the ex-situ training method. Section 4.3 describes memristor crossbar based single-layer and multi-layer neural networks implementations using the proposed learning method introduced in Section 4.2. It also shows the experimental results and comparison between the training method using simple dot products and our proposed method. The chapter is concluded in Section 4.4.

4.2 Neural Network Design and Proposed Learning Method

One of the methods that can be utilised while training a neural network is known as ex-situ training as, we discussed in Section 2.3.3. In this method, the weights that are trained by a software implementation are programmed into the system. In this section, we explain our modified learning algorithm for the ex-situ training method. Before going to our modified method, we will discuss the important components that are used in the implementation of memristive neural circuits.

4.2.1 A Neuron as a Memristive Circuit

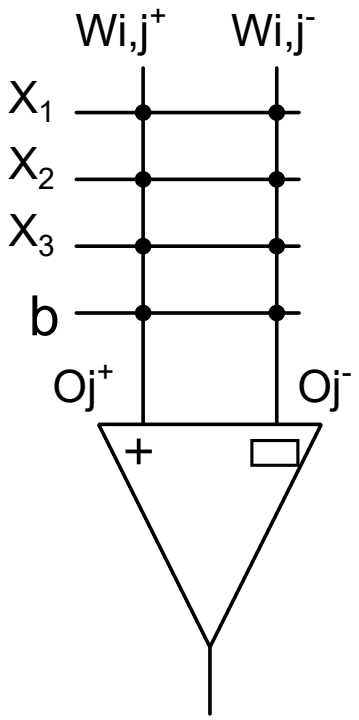


Figure 4.1: A neuron implemented with memristors as synapses.

The schematic in Fig. 4.1 displays a memristive circuit for implementing a neuron. This memristive neuron is the building block of memristive neural networks. Multiple memristive neurons are connected together to train and implement ex-situ multi-layer neural networks. The training has been provided in software and final weight values

have been written in memristor crossbar implemented in hardware. Conductance values are considered as weight values in the memristor based neural network circuits. So, the weights are updated according to the conductance of memristors in memristive neural networks. The 0T1M approach is used in the circuit as shown in Fig. 4.1 as 0T1M memristor crossbars are much denser and smaller in area than 1T1M crossbar structures as discussed in Section 2.3.1.

In this circuit, the rows represent the input values and the columns the outputs. X_1, X_2 and X_3 are inputs and b is considered as the bias in this structure. The memristor acts as a synapse and interconnects each row with the column. Two memristors are used per synapse to present either a positive or negative effect on total weight values. For the input X_1 as shown in Fig. 4.1, if $W_{i,j^+} > W_{i,j^-}$, then the synapse associated with input X_1 has a positive weight. Similarly, if $W_{i,j^-+} < W_{i,j^-}$, then the synapse associated with input X_1 has a negative weight. Using the following equation, the value of W in the memristor crossbar array is always calculated.

$$W = W_{i,j^+} - W_{i,j^-} \tag{4.1}$$

The weight value in the memristor crossbar is obtained by subtracting W_{i,j^-} from W_{i,j^+}.

All the inputs, along with the memristors of each column, are connected to a comparator as shown in Fig. 4.1. The output of the comparator at the bottom of the circuit represents the output of each neuron. If the output of the column (O_{j^+}) on the positive side of the comparator is greater than the column output (O_{j^-}) on the negative side of the comparator, then the pair of memristors used per synapse represents a positive weight and vice versa. The comparator gives an output as two discrete values as it behaves as an ideal threshold function.

The Dot product is the most important part of any neural network algorithm. The output of any neuron depends on the dot product and activation function in neural networks. The dot product of any neural network in software is represented by

Eq. 4.2. However, any simple memristor crossbar array provides the output as shown in Eq. 4.3.

$$Vout_j = \sum_i X_i W_{i,j} \tag{4.2}$$

Here, X_i represents the input values and $W_{j,i}$ represents the weight values in software neural networks.

$$Vout_j = \frac{\sum_i Vin_i C_{i,j}}{\sum_i C_{i,j}} \tag{4.3}$$

Vin_i and $C_{i,j}$ represent the voltage and conductance values corresponding to each input i.

The output at each column in the memristor crossbar is calculated as Eq. 4.3. It represents a voltage divider between the memristor conductance values in a simple crossbar array.

Synaptic Weight Precision: The number of memristors used for each synapse and the conductance range of the memristor devices determine the precision of memristor-based synaptic weights. The range of synaptic weights for single and two memristors used for each synapse is shown in Table 4.1. In this table, the C_{max} is maximum conductance and C_{min} represents minimum conductance of a memristor. $\bar{C}$ represents the difference between the positive and negative weight in a design with only using a single memristor per synapse [78]. With two memristors per synapse, the range of synaptic weights is therefore twice that with just one.

Table 4.1: Synaptic weight precision for memristor-based synapse.

	Memristors per Synapse	
	Two Memristors	Single Memristor
Minimum weight	$C_{min} - C_{max}$	$C_{min} - \bar{C}$
Maximum weight	$C_{max} - C_{min}$	$C_{max} - \bar{C}$
Range	$2(C_{max} - C_{min})$	$C_{max} - C_{min}$

4.2.2 Proposed Learning Method for *Ex-situ* Training

In the proposed training method, the output at each column has been calculated using the voltage divider sensing technique as our training process mimics the memristor crossbar in hardware. This learning method is based on backpropagation learning algorithm [69]. However, as it is the most popular learning algorithm, its implementation in hardware is difficult [21, 53]. The backpropagation algorithm iteratively propagates error signals from the output layer to calculate the required weight change in the hidden layers. This is a relatively complicated and expensive procedure to perform in electronic circuits. The simple memristor crossbar structure uses a voltage divider between the resistances connected in series and parallel to provide the output at each column as described in the previous section. Hence, all the networks have been trained using a more efficient architecture based on voltage divider. The arctan function is used as an activation function. Two weight values per synapse are used in the learning process. The backpropagation learning algorithm along with our proposed dot product and weight update equations has been used to train the networks. The following are the steps of the learning algorithm that have been used to train the neural networks in software.

1. Initialize the weights with low random conductance values.

2. Calculate the dot product at the bottom of each column of the output layer and the hidden layer as the networks are trained in crossbar fashion. The output value of each column is calculated using the following equations as it represents the voltage divider. Then apply the activation function.

$$O_j = \frac{\sum_i X_i W_{i,j}}{\sum_i W_{i,j}} \qquad Y_j = f(O_j) \tag{4.4}$$

where,

$$O_j = O_j^+ - O_j^- \tag{4.5}$$

Here, i and j represent input and hidden-layer nodes. Eq. 4.4 shows the dot product and activation function of each neuron at the hidden-layer.

The following Eq. 4.6 represents the output and activation function at the hidden-layer.

$$\begin{aligned} O_o &= \frac{\sum_j Y_j W_{j,o}}{\sum_j W_{j,o}} \\ Y_o &= f(O_o) \end{aligned} \tag{4.6}$$

where,

$$O_o = O_o^+ - O_o^- \tag{4.7}$$

The output of the hidden-layer is considered as an input to the output layer as shown in Eq. 4.6. $W_{j,o}$ is the weight value between the hidden-layer and the output layer.

3. Calculate the error at the output layer and the hidden-layer using the following equations:

$$E_o = T_o - Y_o \tag{4.8}$$

Here, E_o is the error at the output layer. T_o and Y_o represent the target and actual outputs and backpropagate the error to the previous layer. E_o represents an error function having values either +1, -1, or 0 (corresponding to weight increase, decrease, or no change respectively).

$$E_h = \sum\nolimits_j E_o W_{j,o} \tag{4.9}$$

Here, E_h is the error at hidden-layer. $W_{j,o}$ represents the weight values of synapse connecting neuron o with previous layer neuron j and E_o is the error which is back propagated from the output layer.

4. Update the weights using following equations at each layer:

$$\Delta W_{j,o} = \eta E_o \frac{1}{1+O_o^2} X_{j,o} \tag{4.10}$$

$$\Delta W_{i,j} = \eta E_h \frac{1}{1+O_j^2} X_{i,j} \tag{4.11}$$

$$W^+_{j,o,new} = W^+_{j,o,old} + \Delta W_{j,o} \tag{4.12}$$

$$W^-_{j,o,new} = W^-_{j,o,old} - \Delta W_{j,o} \tag{4.13}$$

$$W^+_{i,j,new} = W^+_{i,j,old} + \Delta W_{i,j} \tag{4.14}$$

$$W^-_{i,j,new} = W^-_{i,j,old} - \Delta W_{i,j} \tag{4.15}$$

Here, η is a constant learning function. The above mentioned Eq. 4.10 and Eq. 4.11 determine the total amount of weight change at the output layer and the hidden-layer. Eq. 4.12 and Eq. 4.13 are used to update the weights connected with the output layer and Eq. 4.14 and Eq. 4.15 are used to update the weights of the hidden-layer.

5. All the steps from step 2 are repeated for each data pattern until the error becomes zero.

4.2.3 Experimental Setup

A single and multi-layer layer neural networks based on memristors have been trained and implemented to perform linear and non-linear separable functions and pattern classifiers. All the networks described in this chapter have been trained in software using the proposed learning method, and then final weights are written into the memristor crossbar neural circuits in hardware. We also trained these networks using simple dot product and compared the results. C++ is used for training and LTspice

is used for hardware implementation of these networks. The alternate current paths are also considered while implementing memristive neural circuits in LTspice.

The memristor model published in [97, 98] has been utilized for the implementation of the memristor crossbar in hardware due to its high resistance ratio (R_{off}/R_{on}=10^6) and fast switching time. The device's entire resistance range can be switched in 10ns. The maximum and minimum resistance values used for this simulation are 125MΩ and 125KΩ [97, 98]. The detailed simulation results for this model are shown in 5.3.4. 0T1M crossbar structure is simulated in LTspice to implement these networks. 0T1M memristor crossbar structures are significantly denser and consume less area than the 1T1M structures. Write and read operations have been performed on memristor crossbar in LTspice to program the final pre-determined weights. In Section 2.3.2.1 and Section 2.3.2.2, these operations are described in detail. The resistance of each memristor is set according to the target resistance calculated during training in the software. It is complex to read a single memristor in a 0T1M memristor crossbar array due to sneak-paths. If the resistances of other memristors in the crossbar array are known, then the expected value of V_o for a target value of R_m in the memristor reading circuit shown in Section 2.3.2.2 can be calculated using software.

4.3 Implementation of Neural Network based on Improved Learning Method

This section explains the implementation and results of single-layer and multi-layer neural networks based on modified learning algorithm. We have implemented different non-linear separable functions and pattern classifiers using memristive crossbar circuits. First, the networks have been trained in software using general dot product and proposed dot product calculations in the learning process. Then, the final weight values as conductance values of memristors that are calculated in software have been directly updated in memristor based crossbar structure in hardware. The

networks have been tested and the experimental results produced by both methods are compared.

4.3.1 Single Layer Neural Network

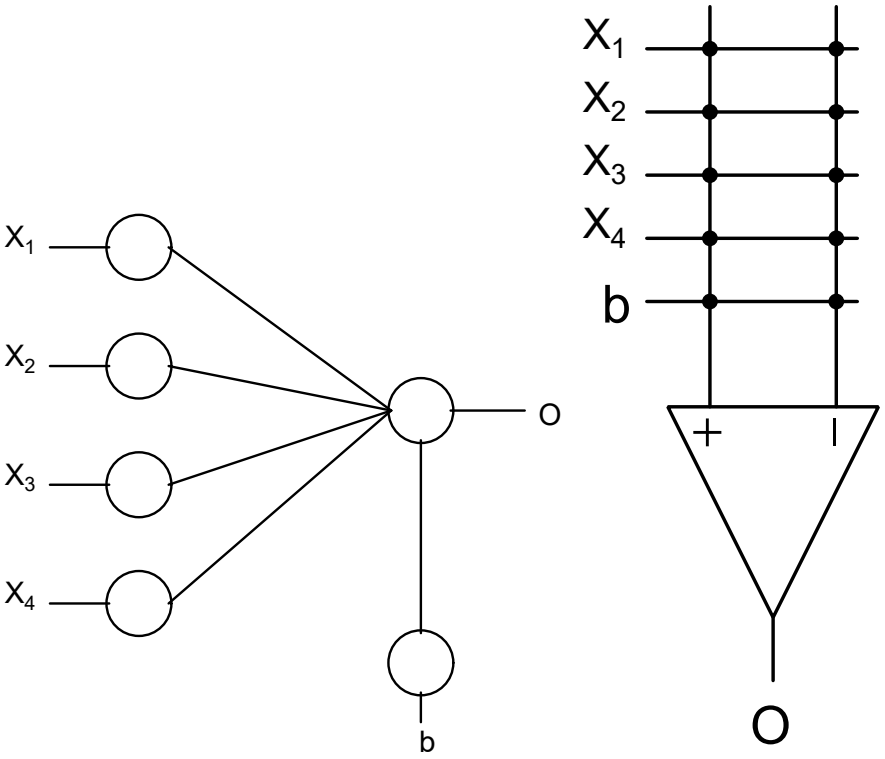
This section illustrates the classification of a linear function and a pattern classifier using a single layer neural network. The network configuration for these networks is represented by x→y where the network has x input and y output neurons.

4.3.1.1 Four Bit Linear Function

The Fig. 4.2a shows a single layer network and Fig. 4.2b displays the corresponding memristive crossbar circuit for four input linear functions. The rows represent the input values and the columns the outputs. 4→1 network configuration is used. In a memristive crossbar, X_1, X_2, X_3 and X_4 are all inputs and b is considered the bias. The memristor acts as a synapse and interconnects each row with the column. All the inputs, along with the pair of memristors are, connected to a comparator. The comparator output is the output of each neuron at each layer. The Fig. 4.3 represents the number of epochs required for both the methods to learn linear functions. The proposed method required fewer epochs to learn the function as compared to the simple dot product method as shown in Fig. 4.3. After hardware implementation of both methods, the general dot product method shows a 6.25% error rate during testing. However, our method shows zero error as shown in Table 4.2.

4.3.1.2 Pattern Classifier

This section presents pattern classification for 3×3 binary images using a single-layer network. The network configuration used for this pattern is represented by 9→2. The network has been trained through a perceptron learning algorithm with modified dot product and weight update equations. Fig. 4.10 displays 3×3 binary pixel image and a single-layer perceptron network for pattern classification. The network consists of nine inputs, one bias and two outputs. The value of bias is fixed and considered



(a) (b)

Figure 4.2: (a) Single layer network and (b) memristor neuron crossbar circuit for linear function.

as 1, whereas the values of inputs and outputs are 1 or 0. The memristor crossbar for pattern classification contains 10×4 memristors in total, including bias as two memristors per synapse.

The set of patterns considered for classification in Fig. 4.5 and Fig. 4.6 represents the alphabets 'X' and 'T' and their noisy versions. The first patterns in Fig. 4.5 and Fig. 4.6 are ideal patterns for letters 'X' and 'T' and the rest are noisy patterns. The grey and white pixels of images are represented by 1 and 0 respectively and the output value for pattern 'X' is considered as 1 and 'T' is 0. The training set contained 50 patterns and some of these patterns are repeated.

The results in Fig. 4.7 show the number of epochs required for the average error to reach zero. The network is learned only in 3 epochs by the proposed method. However, the simple dot product method takes two more epochs to learn. During the implementation and testing in hardware, our method shows 100% accuracy than

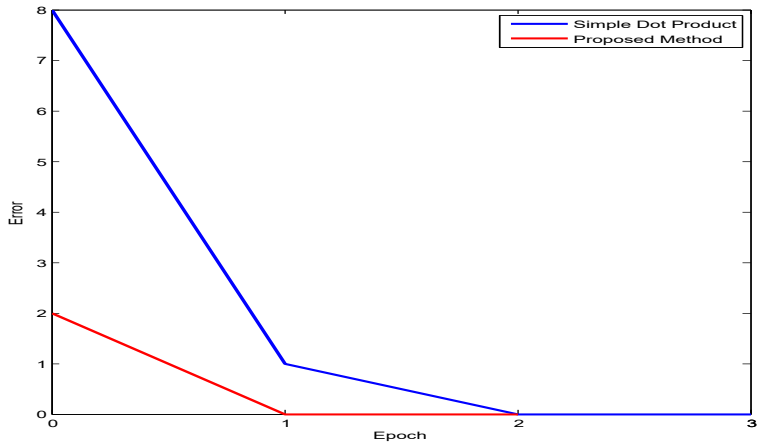
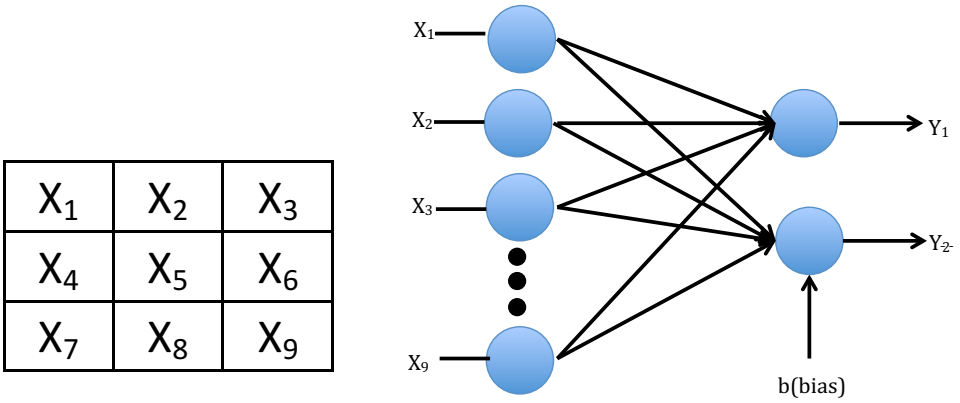


Figure 4.3: Training error in each epoch for four input linear function.



(a) (b)

Figure 4.4: (a) 3×3 binary image and (b) single layer network for pattern classification.

other method as mentioned in Table 4.2.

4.3.2 Multi-layer Neural Network

A multi-layer neural network is required for the classification of complex non-linear separable functions. We implemented the multi-layer neural networks with one and two hidden layers to classify three-bit parity functions and full adder function.

4.3.2.1 Two-layer Network Implementation

The two-layer neural network is utilized to perform three-bit parity and full adder functions. The configuration for these networks is represented by x→y→z, where the network has x input neurons, y hidden and z output neurons.

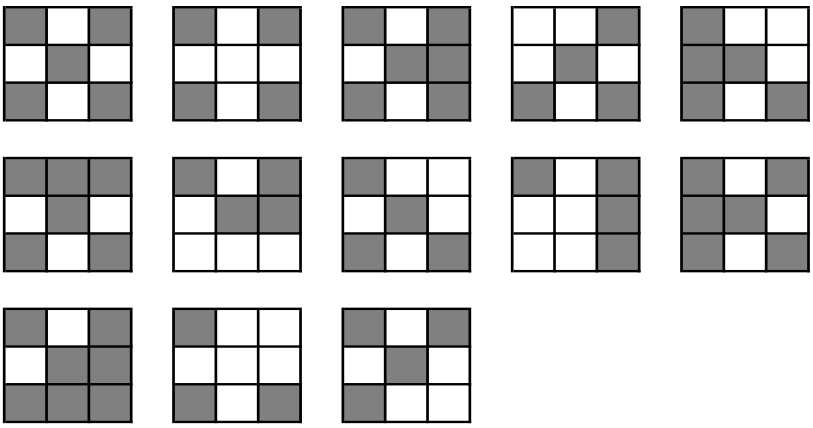


Figure 4.5: Set of input patterns for 'X'.

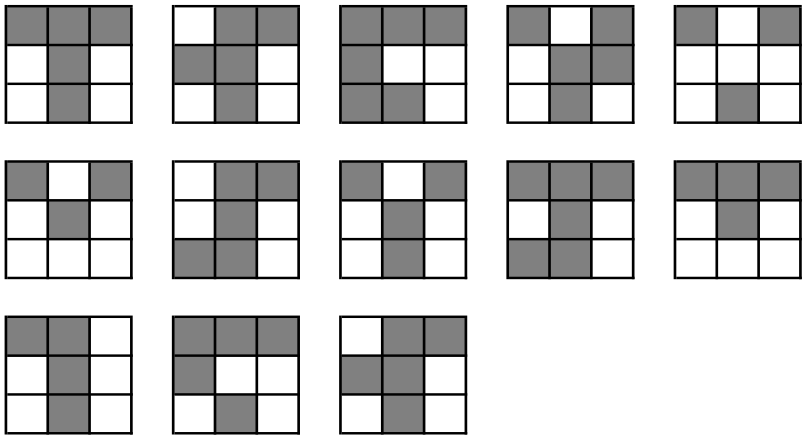


Figure 4.6: Set of input patterns for 'T'.

Three Bit Parity Function The network in Fig. 4.8a displays a two-layer neural network for three input parity function. This network is represented by 3→4→1. The network contains one hidden-layer consisting of four hidden neurons. The Fig. 5.4 displays the memristor crossbar neuron circuit. In this circuit, the first layer consisting of 4×8 memristor crossbar and the output layer consists of 5×2 including bias. The Fig. 4.9 represents the results of training with both methods. The network with the proposed method learns more quickly as compared to another method. The proposed method requires less than 10 epochs to train the network whereas simple dot product network learns three input parity function in almost 70 epochs and its error rate is more than the proposed method while implementing and testing in hardware as

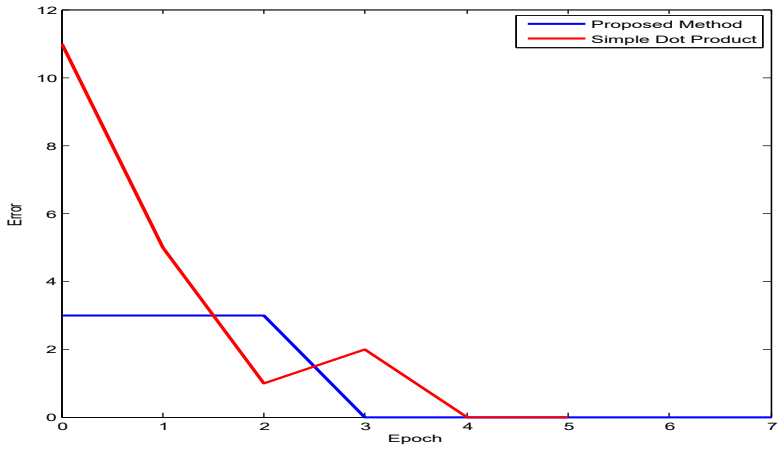


Figure 4.7: Number of epochs required for average error to reach zero.

shown in Table 4.2.

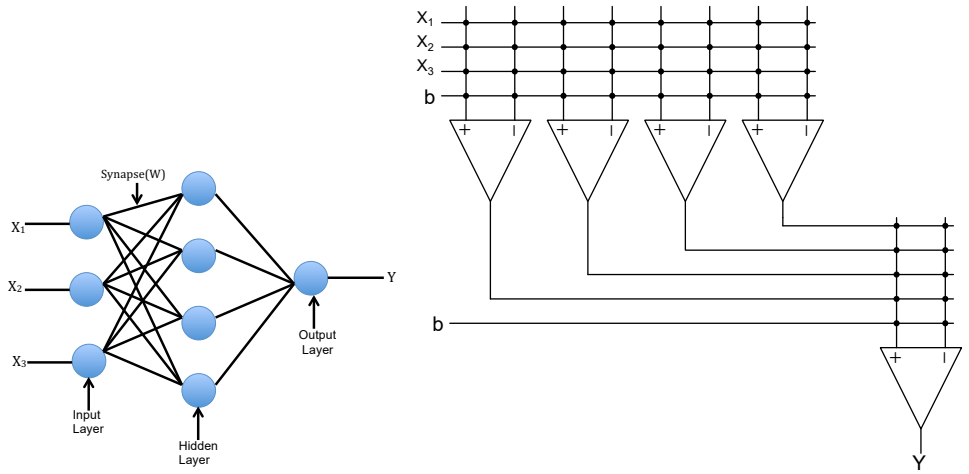


Figure 4.8: (a) Two-layer network and (b) memristor based neuron crossbar circuit for three-bit parity function.

Full Adder Function The full adder function has also been trained and implemented using the two-layer neural network. The two-layer network and its corresponding memristor crossbar circuit for full adder function are displayed in Fig. 4.10a and Fig. 4.10b. The network configuration used for this network is represented by 3→4→2. This network contains 3-inputs neurons along with one bias, four hidden neurons and two output neurons. One output is for sum and the other is for carry in full adder function. In memristive crossbar, the first-layer and output-layer have

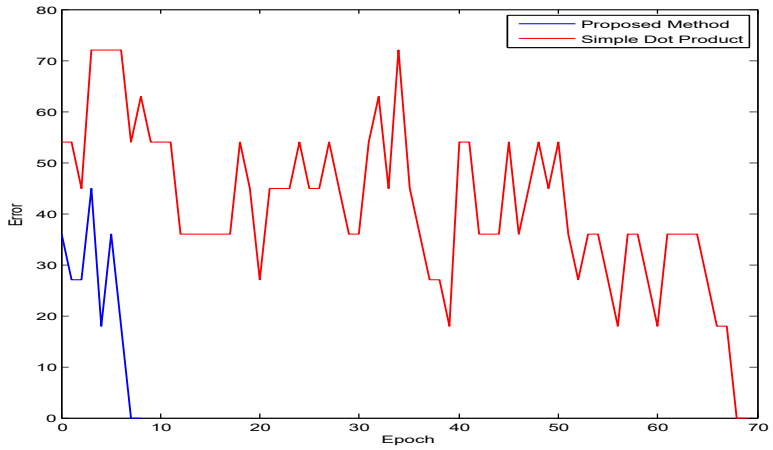
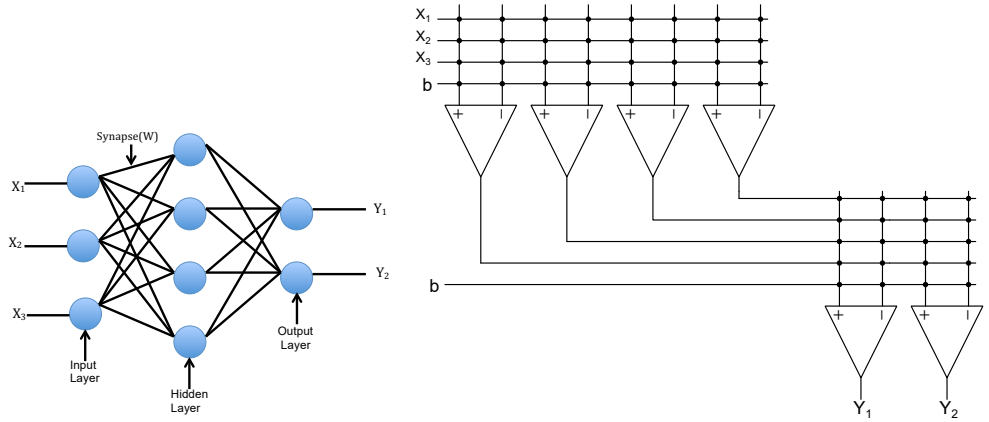


Figure 4.9: Epochs required to learn three-bit parity function on two-layer network.

been implemented using 4×8 and 5×4 memristor crossbars respectively. The error reaches zero after 40 epochs using the proposed method and it requires more than 200 epochs to learn with the simple dot product method. Table 4.2 shows a 50% error rate during testing and implementation in hardware.



(a) (b)

Figure 4.10: (a) Two-layer network and (b) memristor crossbar circuit for full adder function.

Binary Counter The three-bit binary up counter function is also considered for training and implementation. 3→6→3 neural network configuration is used for this function. The network and its corresponding memristor crossbar neural circuit represenation are shown in Fig. 4.12a and Fig. 4.12b. This network has three inputs, six hidden neurons and three outputs. The inputs represent the present state and the

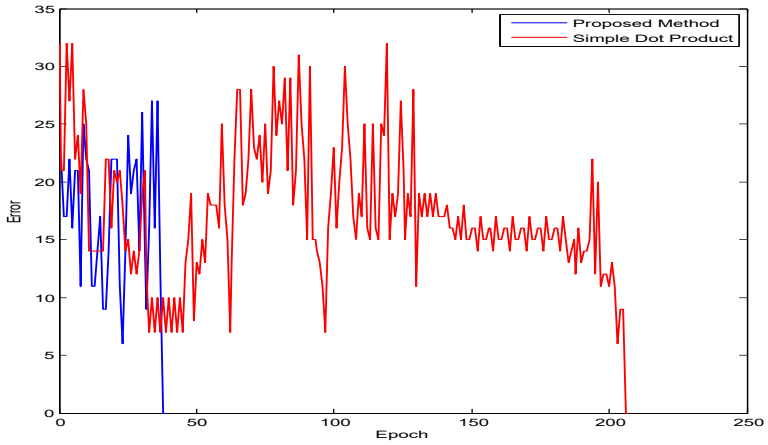
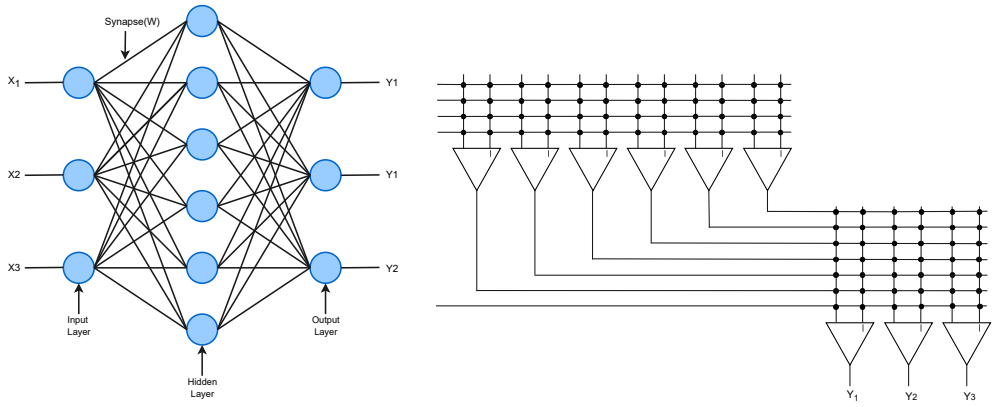


Figure 4.11: Error during training process of full adder function.

output represents the next state of the counter. As shown in Fig. 4.12b, 4×8 memristors are used at the first layer and 7×6 memristors are used at the output layer during the implementation of the memristor crossbar circuit in LTspice. The training results are shown in Fig. 4.13 and it shows that the network learns more faster using the modified learning method than the simple dot product. The implementation and testing results of the memristor circuit in LTspice are shown in Table 4.2 and it shows more accuracy than other simple dot product method.



(a) (b)

Figure 4.12: (a) Neural network representation and (b) memristor crossbar circuit for binary counter function.

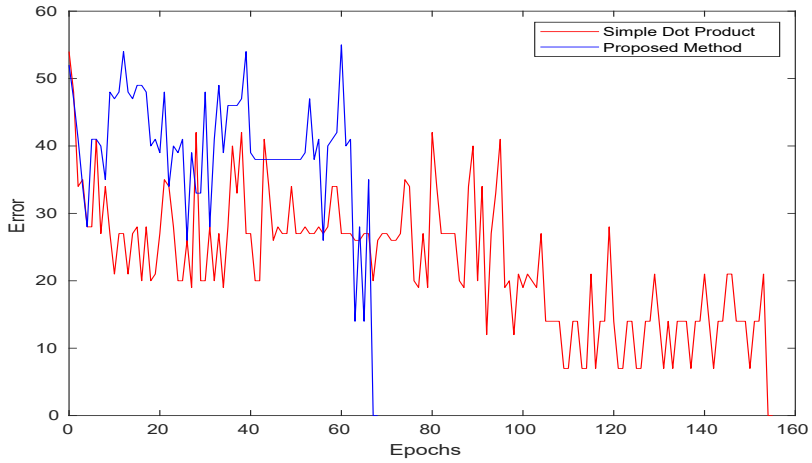


Figure 4.13: Epoch required during the training of binary counter.

4.3.2.2 Three-layer Network Implementation

This section describes the implementation of the three-layer neural network to perform a three-bit parity function. A three-layer neural network is shown in Fig. 4.14 for three input parity function. The network consists of 3 input $\times$ 4 hidden-1 $\times$ 2 hidden-2 $\times$ 1 output nodes to classify 3-bit parity function. Hidden-layer1 consisting of four hidden neurons and hidden-layer2 contains two hidden neurons as shown in Fig. 4.14. We used this three-layer network to perform the non-linear separable function. The memristive crossbar circuit corresponding to this network, which is implemented in hardware is shown in Fig. 4.14b. In hardware, the first-layer is implemented using 4×8 crossbar, the second-layer is implemented using a 5×4 crossbar and the final-layer contains a 3×2 crossbar.

Fig. 4.15 shows the number of epochs required to learn a three-bit parity function with both methods. The simple dot product learning method takes more epochs to learn as shown in Fig. 4.15. In the proposed method, after nine epochs, an error becomes zero and the network is learned a three-bit parity function successfully. The proposed method also shows better results compared to other method during simulation and testing of the memristive crossbar circuit in hardware as shown in Table 4.2. The network configuration for this network is represented by x$\rightarrow$y1$\rightarrow$y2$\rightarrow$z where the network has x inputs, y1 hidden layer1, y2 hidden layer2 neurons and z

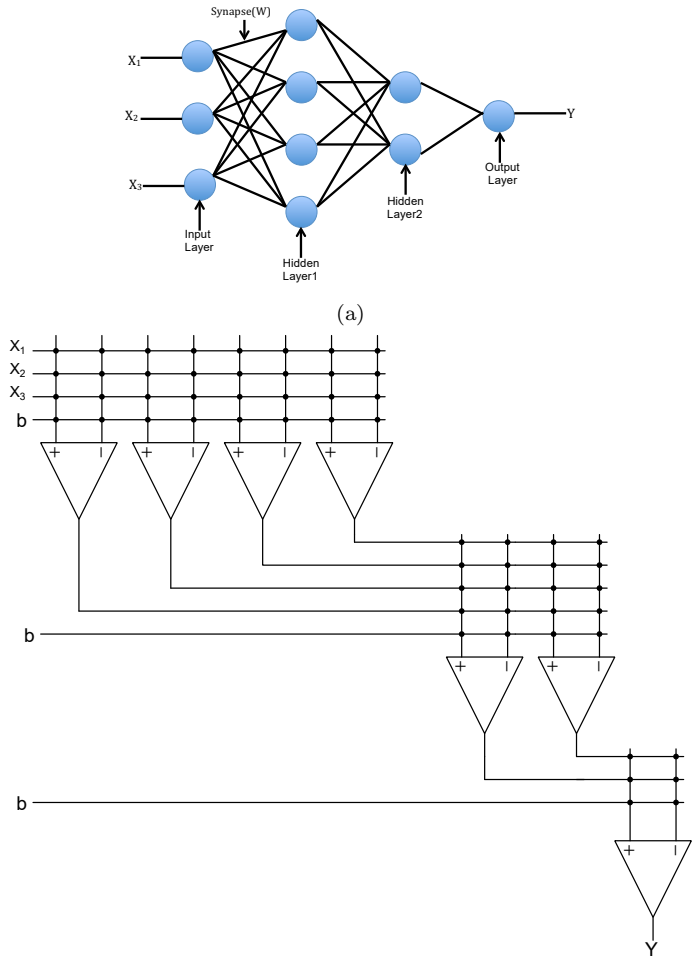


Figure 4.14: (a) Three-layer network and (b) memristive neuron crossbar circuit.

output neurons as shown in Table 4.2.

Our simulation results have successfully demonstrated that the modified learning method is capable of training complex non-linear functions successfully using multi-layer crossbar networks. The experimental results also show that it provides the training faster than using a simple dot product in a learning algorithm. Moreover, it provides more accuracy while using it with the ex-situ method. We achieved a zero error rate with the implementation of these small networks using this method. Our aim is to mimic the behaviour of the memristor crossbar in software training to obtain more close computations to hardware in ex-situ. It also works well with the back propagation learning algorithm. Hence, more accuracy could be achieved

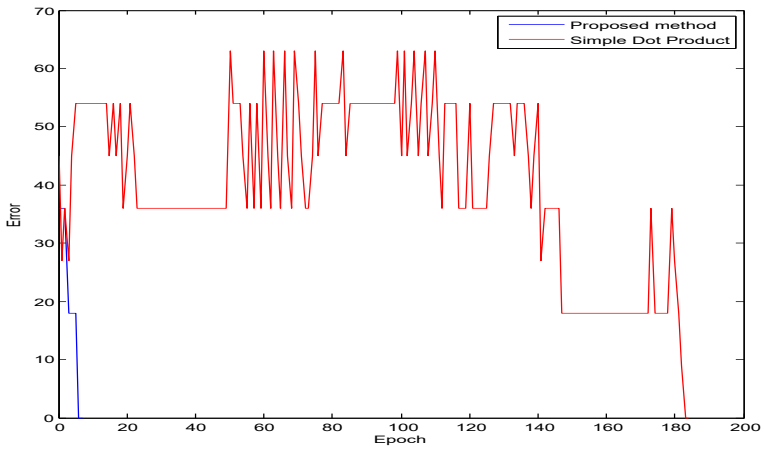


Figure 4.15: Error during training process of three-bit parity function on three-layer network.

for more massive networks using ex-situ processes by finding ways to produce more accurate computations.

Table 4.2: Comparison of simple dot product method with proposed method.

Function	Network Conf.	Simple Dot Product		Proposed Method	
		Epoch	Error Rate (%)	Epoch	Error Rate (%)
Four-bit-AND	4→1	2	6.25	1	0
Pattern Classifier	9→2	4	2	3	0
Three-bit-parity	3→4→1	68	37.5	07	0
Fulladder	3→4→2	202	50	38	0
Binary Counter	3→6→3	154	20	67	10
Three-bit-parity	3→4→2→1	183	25	08	12.5

4.4 Summary

An improved learning method has been proposed in this chapter. This learning method is based on the voltage-divider technique and the backpropagation algorithm. The output of each memristive neuron is calculated using a voltage divider while training the network in software. We used the proposed learning method with ex-situ training to simulate and train single-layer and multi-layer memristive neural networks by using C++ and the LTspice environment. Firstly, the networks are trained in software using the proposed learning method and the final weights are calculated. The networks trained in software mimic the memristor crossbar in hardware. Therefore, the voltage-divider has been used to calculate the dot product in the learning process. Then, the memristor crossbar array is simulated in the LTspice and the conductance values of each memristor are adjusted according to the weights calculated during the training process. The read and write schemes are used to update the weights in the memristive crossbar structure. We also trained these networks using simple dot products and compared the results. The aim of the proposed learning method is to reduce the error rate while implementing networks in hardware with ex-situ training methods. It also reduces learning time in software. The simulation results have successfully demonstrated the classification of linear and non-linearly separable problems and provide more accuracy during the implementation of the ex-situ method in memristive crossbar array-based neural networks.

Chapter 5

Efficient and Low Overhead Memristive Architecture for Activation Functions in Deep Neural Networks

5.1 Introduction

Along with the learning method, an activation function deserves similar consideration in a neural network. An activation function introduces non-linearity into the network, which helps the network with learning and performing complex functions. Thus, an activation function is an essential part of an artificial neural network. The widely used activation functions are sigmoid, tanh and rectified linear unit (ReLU) as we discussed in Section 2.3.4. The ReLU activation function, introduced very recently [58], is becoming more popular in deep neural networks due to its simplicity. It also provides sparsity and aids in eliminating the gradient problem that is hard to handle with sigmoid and tangent functions. Moreover, it has been shown that deep networks can be trained efficiently using ReLU even without pre-training [25]. Most of the activation functions used in memristive neural networks in the literature are approximated as sigmoid, tanh and piecewise linear functions [1, 5, 26]. The majority of these functions are implemented using only operational amplifiers, which incur significant hardware overhead. In contrast, considering the merits and demer-

its of the existing approaches, in this chapter, we present a novel implementation of the ReLU activation function based on memristor MIN functionality. To the best of our knowledge, this is the first circuit that approximates an activation function using memristors. We show with experimental results that the proposed circuit is significantly area efficient as well as aids fast learning.

The rest of the chapter is organized as follows: The circuit for activation function in memristive neural networks is proposed in Section 5.2. Firstly, the MIN-MAX functionality based on memristors is reviewed in Section 5.2.1. Then, Section 5.2.2 describes the proposed memristive activation circuit and the experimental details of the network simulation. Section 5.3 illustrates memristor crossbar architectures used for implementing different multi-layer neural networks and experimental setups. Section 5.4 evaluates the results and compare the results with existing approaches. Finally, the chapter is concluded in Section 5.5.

5.2 Proposed Circuit for Activation Function

We proposed an activation function circuit using memristors. It is based on memristive MIN functionality. So, we introduced the memristor based MIN-MAX circuits first in this section. Then, we described our proposed activation function circuit for memristive neural networks.

5.2.1 The MIN-MAX Function

The MIN-MAX circuits are becoming crucial building blocks in fuzzy systems and many artificial neural networks [72]. However, the major obstacle to the existing designs is the area complexity. As a result of their nanoscale size characteristics, memristor-based MIN-MAX circuits are emerging as highly efficient circuit designs [39]. Fig. 5.1(a) represents a diagram of a single memristor, where it is assumed that when the voltage difference between the p terminal and the n terminal is higher than the threshold voltage, the memristor switches to a low resistance state (R_{on});

otherwise, it switches to a high resistance state (R_{off}) [103]. Fig. 5.1(b) shows a two Memristor Rationed Logic (MRL) MIN circuit [103], where M1 and M2 represent memristors. The MIN-MAX functions are defined by the following equations:

$$MIN(V_x, V_y) = \begin{cases} V_x, & V_x \leq V_y \\ V_y, & otherwise. \end{cases} \tag{5.1}$$

$$MAX(V_x, V_y) = \begin{cases} V_x, & V_x \geq V_y \\ V_y, & otherwise. \end{cases} \tag{5.2}$$

Assuming that V_x and V_y are the voltages at the x and y terminals, respectively. In MIN circuit operation, if $V_x > V_y$, then the voltage V follows V_y. As the current flows from the memristor M1 to the memristor M2, the voltage appearing at the positive terminal of M1 is smaller than the negative terminal of M2. Hence, M1 switches to the 'OFF' state (a high resistance state) and M2 switches to the 'ON' state (a low resistance state). If $V_x = V_y$, then the voltage V approximately equals to V_x or V_y. As M1 and M2 appear in parallel, no current flows between the inputs. If $V_x < V_y$, then the voltage V follows V_x. M1 switches to the 'ON' state and M2 switches to the 'OFF' state.

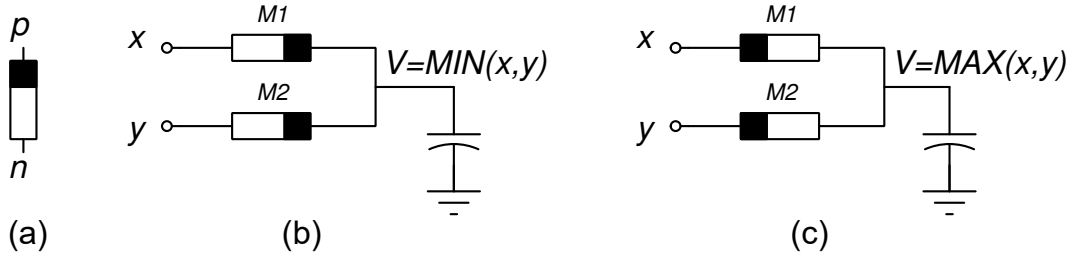


Figure 5.1: (a) Single memristor, (b) memristor based MIN circuit and (c) memristor based MAX circuit.

Fig. 5.1(c) shows a memristor based MAX circuit. The MAX circuit operations are vice-versa to the MIN circuit operations. If $V_x > V_y$, then the voltage V follows V_x and approximately equal to $V_x - V_{M_1}$, where V_{M_1} is the voltage drop across M1. As the current flows from the memristor M2 to memristor M1, the voltage appearing at

the positive terminal of M1 is higher than the negative terminal of M2. Hence, M2 switches to 'OFF' state. If V_x=V_y, then the voltage V approximately equals to V_x or V_y. As M1 and M2 appear in parallel and no current flows between the inputs. If V_x<V_y, then the voltage V follows V_y and V equal to $V_y - V_{M_2}$, where V_{M_2} is the voltage drop across M1. Hence, M1 switches to the 'OFF' state and M2 switches to the 'ON' state.

5.2.2 Memristive Neuron Schematic with Proposed Circuit for Activation Function

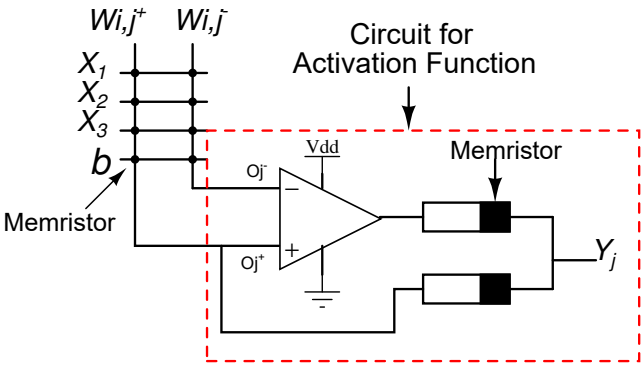


Figure 5.2: A single neuron crossbar along with memristor MIN based activation function.

Fig. 5.2 shows the proposed architecture of a memristive neuron, comprising a comparator and an MRL MIN circuit, for realizing the ReLU activation function. The following equation represents the functionality of this circuit:

$$Y_j = \begin{cases} O_j^+, & O_j^+ \geq O_j^- \\ 0, & otherwise, \end{cases} \tag{5.3}$$

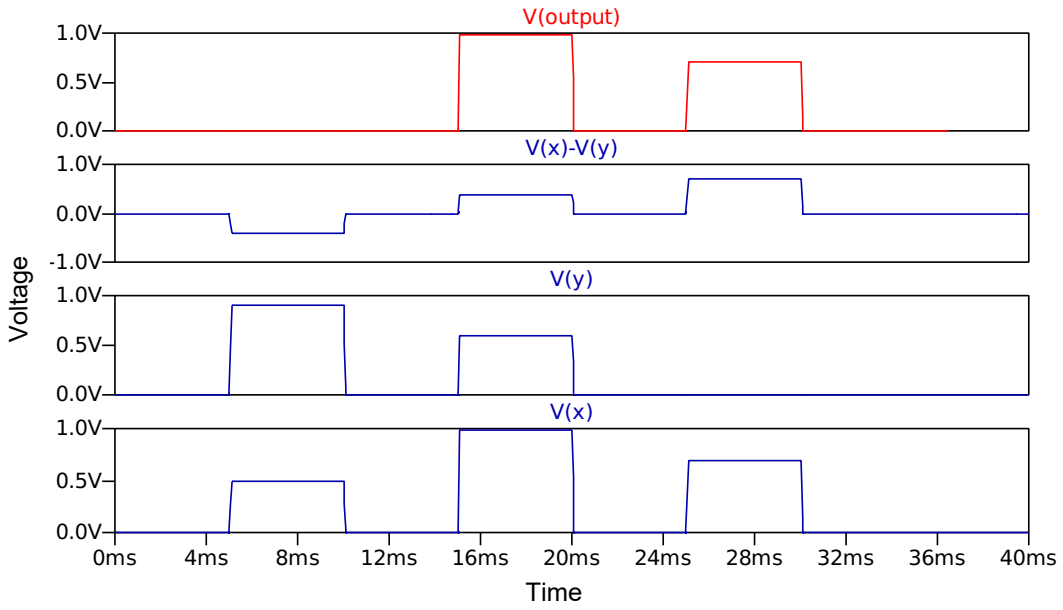


Figure 5.3: Simulation results of the circuit in Fig. 5.2.

$$W_{i,j} = W_{i,j^+} - W_{i,j^-} \tag{5.4}$$

where Y_j is the final output of this circuit. O_j^+ and O_j^- represent the dot product at each column representing a single neuron, as two memristors are required to represent a single weight ($W_{i,j}$). If $O_j^+ > O_j^-$, then it will have a positive effect on the total dot product (O_j). Otherwise, it will have a negative effect. According to Eq. 5.3, if the total dot product produces a negative effect, then it will be considered 0 at the final output. Otherwise, the higher dot product value between the two columns is considered the final output of a neuron. The total dot product of each neuron can be calculated by using the following equation:

$$O_j = O_j^+ - O_j^-. \tag{5.5}$$

The comparator in Fig. 5.2 provides an output of either zero or V_{dd}, where V_{dd} is assumed to be the supply voltage. Then, the memristor based MIN function is used to calculate the minimum value between the comparator output and the dot product at the column on the positive side (O_j^+).

The SPICE simulation result of the proposed activation circuit is shown in Fig. 5.3. Here, $V(x)$ and $V(y)$ are assumed as positive (O_j^+) and negative (O_j^-) inputs of the comparator, respectively. Additionally, $V(x) - V(y)$ represents the total dot product (O_j) and $V(output)$ shows the final output (Y_j) of the circuit. If the total dot product has a negative effect, then the final output will be zero; otherwise, it follows $V(x)$.

5.3 Multi-layer Neural Network Implementation using Proposed Circuit

This section illustrates memristive crossbar architectures for multi-layer neural networks.

5.3.1 Functions Implementation using Two-layer Memristive Architecture

The two-layer circuit design is utilized to implement non-separable functions as follows.

Three Bit Parity Function

A three-bit parity function has been simulated using the schematic as shown in Fig. 5.4. Here, three inputs and one bias are used. Two memristors are used to compute a single weight value as it determines the positive and negative effects on the total dot product of neurons at each layer. The outputs of the neurons at the final layer are thresholded. Thus, a comparator is used at the output layer. In Fig. 5.4, 4×8 memristors are used to implement a crossbar at the first layer and the final-layer contains 5×2 memristors along with bias. 4×2 memristors are used at the hidden-layer for the activation circuit. Fig. 5.12 shows the epochs used by each activation function during training to learn a three-bit parity function.

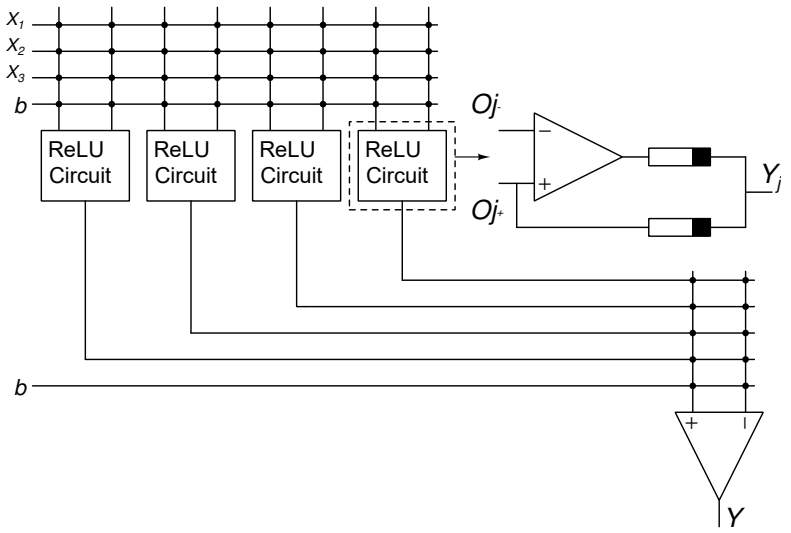


Figure 5.4: Memristor based crossbar for 3-bit parity.

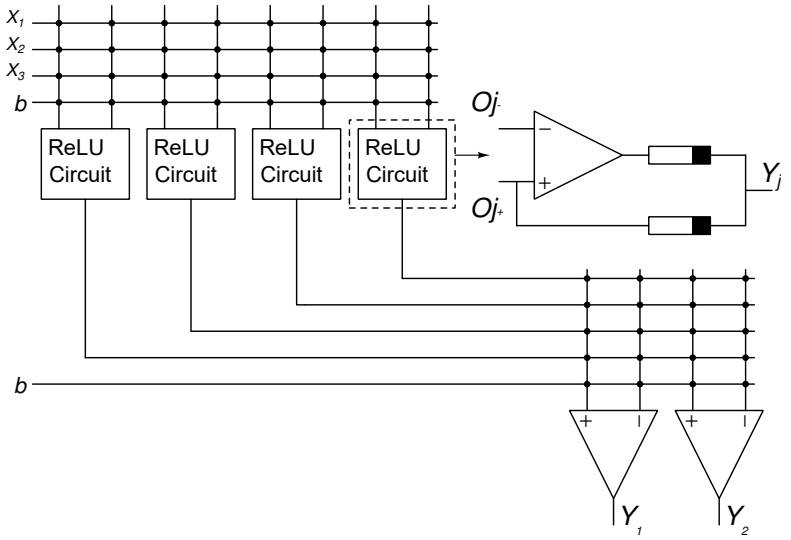


Figure 5.5: Memristor crossbar structure for full adder function.

Full Adder Function

The memristor based crossbar structure for full adder function is displayed in Fig. 5.5. The network consists of 3-input neurons, 4-hidden neurons and 2-output neurons where the two outputs represent sum and carry respectively. 4×8 memristors are used by the first-layer and 5×4 memristors are used by the output-layer in the memristor crossbars. 4×2 memristors are used to implement an activation function at the hidden-layer. Fig. 5.13 shows the error rate of the full adder function during

training. The results in Fig. 5.13 show the total number of epochs required for training using each activation function and the proposed activation function takes less time for training than other activation functions.

5.3.2 Function Implementation using Three-layer Memristive Architecture

This section presents a three-layer memristor crossbar based neural network implementation. Fig. 5.6 unveils the memristor crossbar schematic for three-bit parity function. The network topology used for this function is 3→4→2. The first-layer is consisting of 4×8 memristors; the second-layer contains 5×4 memristors and the output-layer contains 3×2 memristors. To implement activation circuit, two memristors are used at each layer for a single output neuron. Thus, 4×2 memristors are used at hidden-layer1 and 2×2 are utilized at hidden-layer2. Fig. 5.14 shows the total epochs required for training using each activation function.

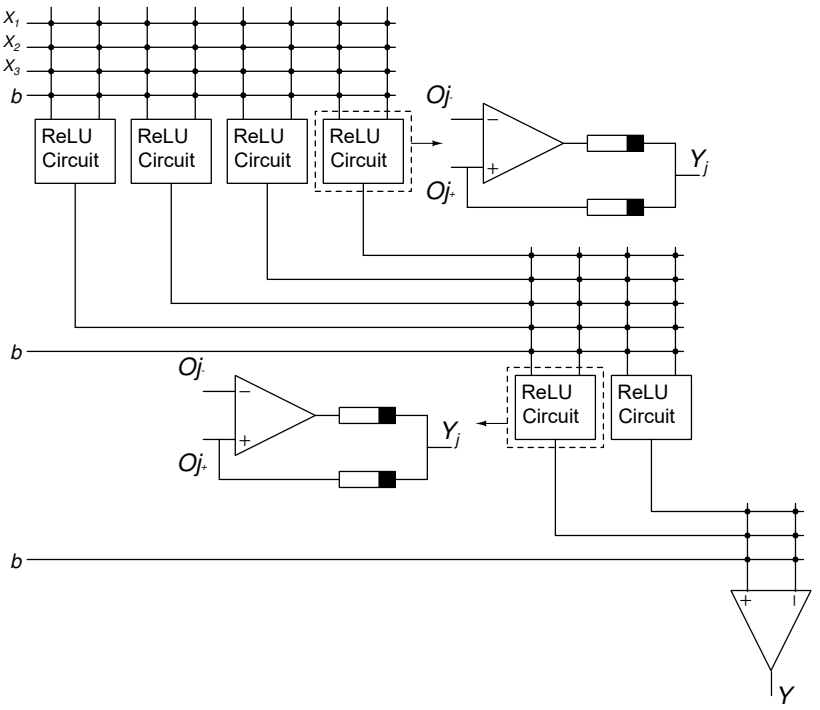


Figure 5.6: Memristive crossbar circuit for parity function.

5.3.3 Implementation of Pattern Classifiers

The Fig. 5.7 shows a 4×4 binary image and its corresponding neural structure. The network includes 17 inputs including one bias, and two outputs. The hidden-layer contains six neurons. The bias value is fixed and is considered as 1. As two memristors are used per synapse, the memristor crossbar circuit contains 17×12 memristors at hidden-layer and 7×4 at the output-layer.

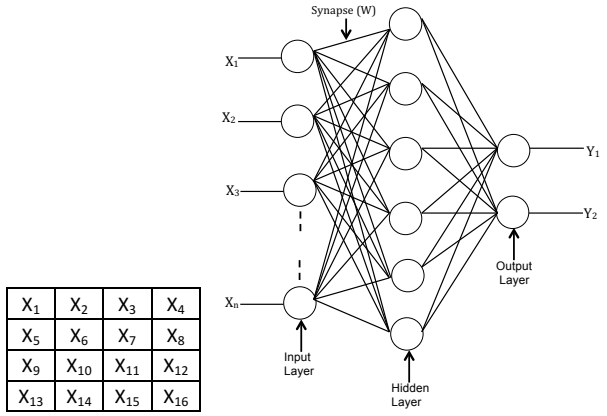
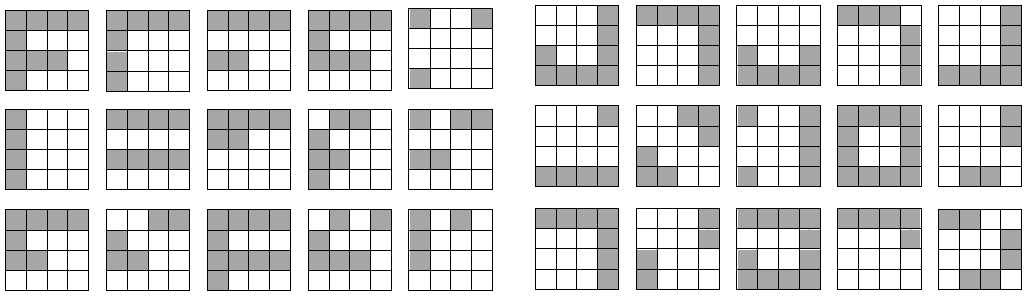


Figure 5.7: (a) Binary image and (b) multi-layer neural network structure.

The Fig. 5.8 displays the set of patterns used for classification. Two alphabets 'F' and 'J' are used. The training set consists of 50 patterns and the Fig. 5.8 shows some of the patterns. The same sets of training patterns are used for testing. Fig. 5.8(a) represents the set of a pattern for alphabet 'F' and Fig. 5.8(b) shows the patterns for alphabet 'J'. Noisy versions of these alphabets are also considered. The First pattern in each set of patterns is an ideal pattern and the rest are noisy versions as shown in Fig. 5.8. The grey pixels of images are represented by 1 and white are represented by 0. The output for alphabet 'F' is considered as 1 and alphabet 'J' is 0.

The result in Fig. 5.15 illustrates the number of epochs required by the network to reach zero or minimum error during the training process.



(a) (b)

Figure 5.8: (a) Set of patterns for 'F' and (b) set of patterns for 'J'.

Iris Classification

Iris dataset [22] contains 3 classes and each class has 50 instances, where each class represents a different type of iris plant called setosa, versicolour and virginica. There are four attributes. The dataset contains 150 patterns in total. First, the data is normalized and then divided into training and testing sets. 90% data are considered for training and 10% is used for testing. The network consists of four input neurons, six hidden neurons, three output neurons and one bias. The configuration for this network is 4→6→3. The result in Fig. 5.9 shows the mean square error during the training of iris dataset.

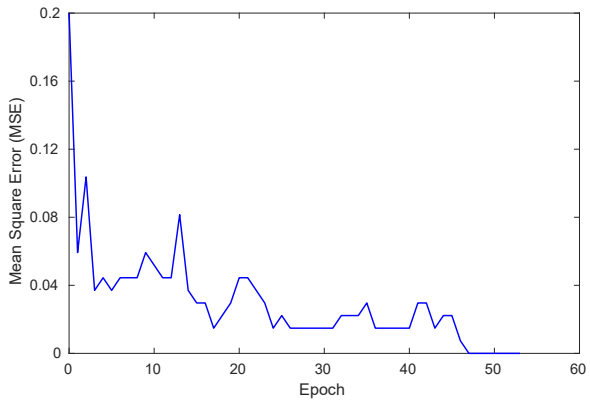


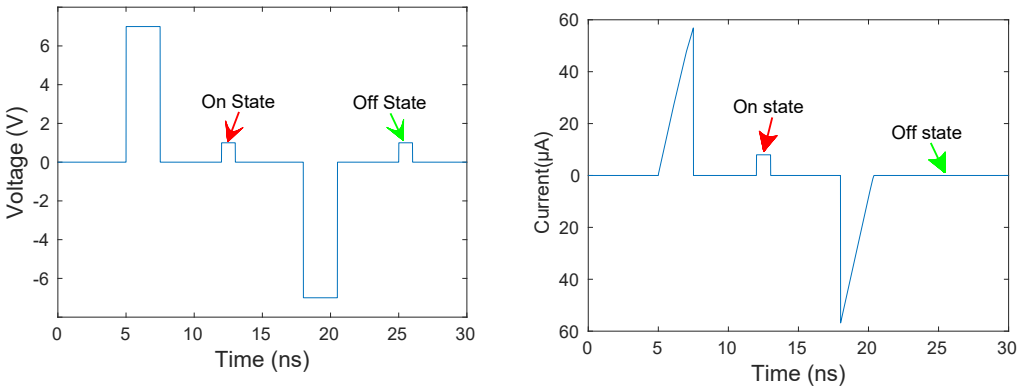
Figure 5.9: The mean square error during training of iris pattern recognition.

5.3.4 Experimental Setup

Memristor based multi-layer networks are trained and simulated with proposed activation circuit. These networks are designed to implement pattern classifiers and 3-input parity functions. The memristive neural circuits shown in Fig. 5.4, Fig. 5.5, Fig. 5.6, and Fig. 5.7 are first trained with algorithm implemented in C++, with a focus on its crossbar architecture, based on the backpropagation algorithm [69]. In the learning process, the modified dot product and weight update equations are used that we proposed in the previous chapter. This method of learning is very efficient and robust, however, its hardware implementation is difficult. All the weights are initialized with low conductance values. After training, the final conductance values are calculated. Eq. 4.3 is used to compute the dot product at each column as it represents the computation of a simple memristor crossbar.

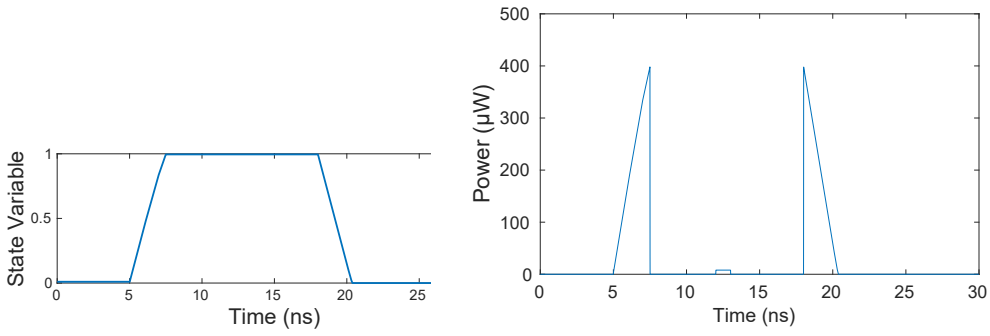
Once the memristive architecture has been modelled and trained in software, the circuit is simulated in LTspice based on the memristor model in [97, 98]. This memristor device is selected as it yields higher resistance ratio (R_{off}/R_{on}=10^6) as well as fast switching time, which is highly desirable for neural network implementations. The simulation results of this memristor device are shown in Fig. 5.10 and Fig. 5.11. The voltage and current waveforms are displayed in Fig. 5.10. A +7V/-7V pulse is applied to switch the memristor device successfully from high resistance state into low resistance state and vice-versa. Fig. 5.11 shows the state variable value of this device that lies between 0 and 1 and the power in this memristor device with respect to time which is calculated by multiplying the voltage and current pulses displayed in Fig. 5.10. The state variable plot shows that the device is successfully switched by applying the voltage waveform as shown in Fig. 5.10. These simulation results show that this device provides the low switching time in nanoseconds. Thus, it will provide more precise results while using in crossbar simulation. The R_{off} (high resistance) and R_{on} (low resistance) are 125MΩ and 125KΩ respectively. All the memristors in the crossbar are programmed in LTspice according to the resistance values pre-

calculated in software using write and read schemes described in Section 2.3.2.1 and Section 2.3.2.2. The implementations in LTspice also consider the alternate current paths in the memristor crossbar. Thus, a sequence of alternate write and read pulses are used to program each memristor in crossbar to achieve the desired conductance value. The research [32, 63] show a strong correlation between the change of memristor state and width, an amplitude of the applied voltage. The pulse width much smaller than the pulse to fully switch the memristor is used to write and the pulse of 1V is used to read a single memristor. The 0T1M approach is used in these circuits, as it is denser and smaller in area as compared to the 1T1M circuit designs.



(a) (b)

Figure 5.10: Simulation results display (a) Voltage pulse and (b) current pulse.



(a) (b)

Figure 5.11: Simulation results display (a) State variable and (b) power.

5.4 Experimental Evaluations

The experimental results of training a three-bit parity, full adder and pattern classifier are shown in Fig. 5.12, Fig. 5.13, Fig. 5.14 and Fig. 5.15 respectively. The mean square error (MSE) during training reaches zero or minimum error value for ReLU much faster than the tanh and sigmoid functions. Additionally, zero error was found after crossbar simulation and testing the inputs in LTspice. A 3-bit weight precision was enough to successfully classify all the functions and patterns. Thus, the non-separable functions and pattern classifiers are successfully trained in software and tested in LTspice by using the schematics in Fig. 5.4, Fig. 5.5, Fig. 5.6, and Fig. 5.7. The proposed architecture also provides significant area benefits compared to the circuits presented in [5, 26] as shown in table 5.1. In this table, sizes of a comparator and a memristor are generously assumed to be $750F^2$ [94] and $4F^2$ [33] respectively. As shown in columns two and three, the areas reported by the existing techniques are almost two and three times more than the proposed method just for a single neuron. Clearly, the area increases substantially as the number of neurons and hidden layers increase, e.g., as shown in the third row and fourth row of the table, which corresponds to the three-bit parity function in Fig. 5.4 and Fig. 5.6 respectively. Moreover, this activation function helps to reduce training time as shown in Fig. 5.12, Fig. 5.13, Fig. 5.14 and Fig. 5.15. Thus, fast computations with less training time can be achieved. However, it takes more time to program the memristors in crossbar which is a generic issue in all memristor based crossbar structures.

	Technique of [5]	Technique of [26]	Proposed Technique
Single neuron	2250	1500	758
Four neurons	9000	6000	3032
Six neurons	13500	9000	4548

Table 5.1: Area (F^2) benefit of proposed technique.

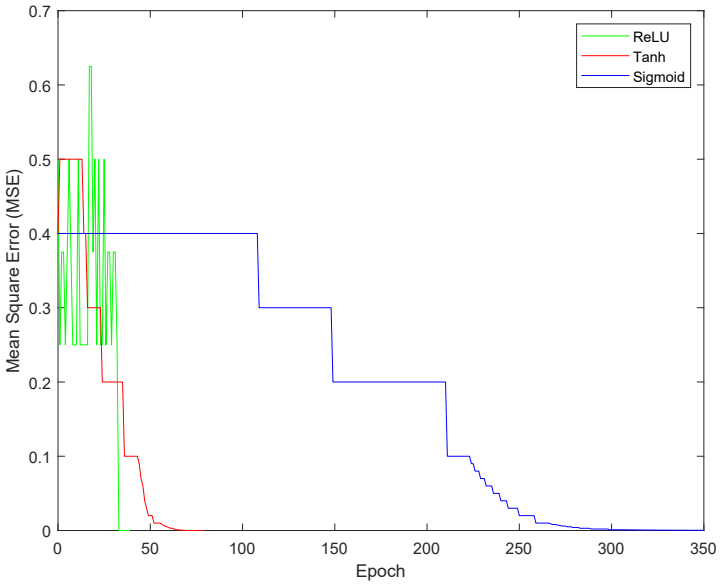


Figure 5.12: Epochs required for mean square error to reach zero or minimum.

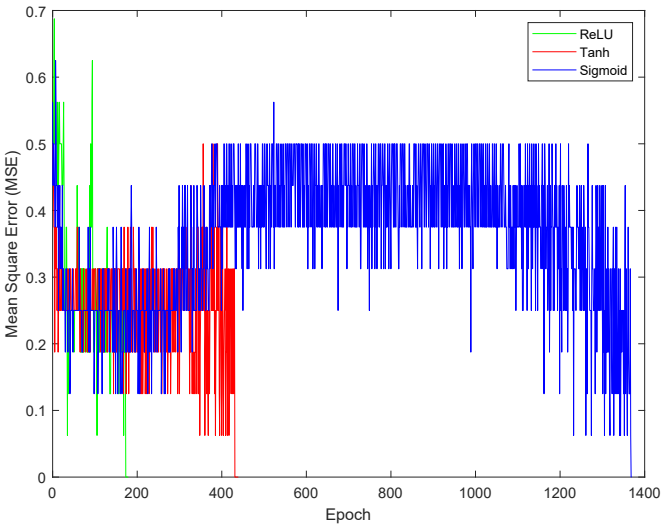


Figure 5.13: Error during training process of full adder function.

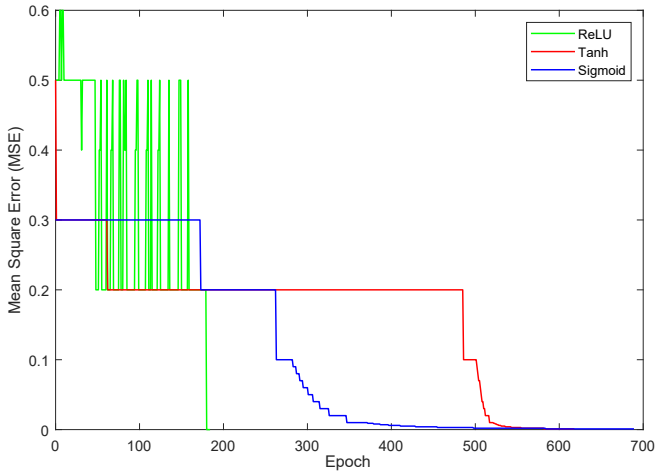


Figure 5.14: The mean square error on 3-layer network during training.

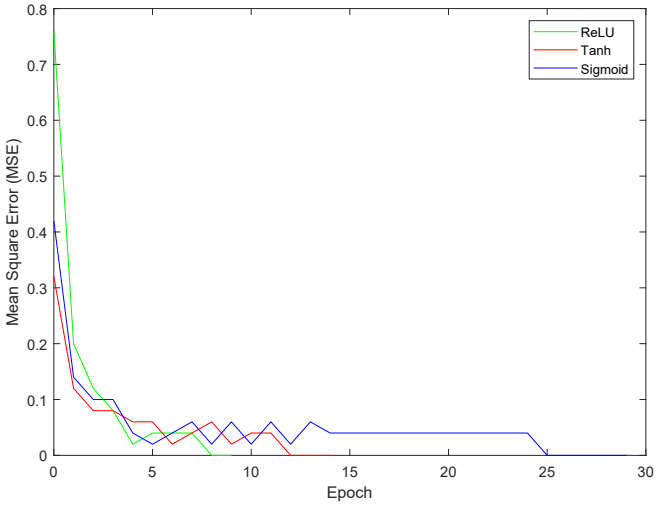


Figure 5.15: The total epochs required for mean square error to reach zero.

5.5 Summary

This chapter represented a novel memristive architecture for realizing the ReLU activation function. This circuit for activation function is based on memristive MIN functionality. Firstly, we introduced the existing MIN-MAX circuit operations based on memristors. Secondly, we proposed an activation circuit based on memristor MIN functionality. The proposed circuit comprises a single comparator and two memristors configured to realize the MIN function. Based on the proposed activation circuit,

three-bit parity functions using two-layer and three-layer memristive neural networks are simulated. We have also simulated different pattern classifiers using our proposed activation circuit in multi-layer memristive neural circuit designs. Finally, the experimental results are evaluated based on these simulations and these results are compared with existing approaches in terms of hardware requirements. Experimental results, based on a multi-layer neural networks showed that the proposed architecture requires significantly lower hardware compared to existing approaches. In addition, the results also demonstrated that non-separable functions and pattern classifiers can be trained and simulated successfully using this circuit. The proposed architecture can also reduce computational costs during the training of the network in software, as well as help speed up the training process due to the simplicity of the underlying activation function. Therefore, the proposed approach could be much more effective for the training and implementation of deep neural networks compared to the existing approaches.

Chapter 6

Fault-Tolerance and Repairability of Memristive Crossbar-based Neural Networks

6.1 Introduction

A memristive crossbar architecture provides fast, low power circuits for high precision matrix-vector multiplication. The computational accuracy of memristor-based neural circuits can be considerably affected by stuck-at-faults (SAFs) in memristor devices, as it is difficult to avoid stuck-at-faults during the manufacturing processes. Therefore, these faults can limit the recognition accuracy of memristor-based neural circuits [11, 91].

This chapter first analyses the error tolerance of memristive crossbar neural networks in the presence of stuck-at-faults. To investigate fault tolerance within the memristor crossbar structures, random memristors in the crossbar array were selected to be stuck in just high resistance state. Up to 50% of the memristors are considered faulty as only one memristor from a pair of memristors (considered a synapse) is selected as faulty to avoid the possibility of having a faulty pair of memristors [99]. Two different memristor based structures are considered to observe the tolerance of faulty memristors. We analyse the neural circuits based on memristors that are able to learn even with faulty memristors.

Secondly, this chapter evaluates the yield of a memristor-based crossbar array of

artificial neural networks in the presence of SAFs. A technique based on the Markov chains is used to estimate the yield in the presence of stuck-at-faults. This method provides a high degree of accuracy. Another method that is used for analysis and comparison is based on the Poisson distribution, which uses the sum of all repairable fault patterns. A fault repair mechanism is also proposed when evaluating the yield of the memristor crossbar array. The results demonstrate that the yield can be improved with redundancies and a higher repairable stuck-at-fault ratio.

The rest of the chapter is organized as follows: Section 6.2 explains the fault-tolerance analysis in memristive neural circuits. Further, in Section 6.2.1, the memristive crossbar structures used for fault-tolerance are described and explain how the memristive neural circuits are tested in the presence of faulty memristors. The results are also discussed in Section 6.2.2. Section 6.3 describes the yield evaluation of memristor-based neural circuits. Section 6.3.1 describes the global yield model and the yield models used for evaluation in this chapter. Section 6.3.2 explains the simulation of a memristor crossbar array with SAFs. It also explains the state diagram and transition rates of the Markov chain that are used to evaluate the yield. The yield evaluation results are demonstrated in Section 6.3.3. Section 6.4 compares the results of both the methods used for yield calculations. The chapter concludes in Section 6.5.

6.2 Fault Tolerance Analysis in Memristive Neural Circuits

In this section, the fault-tolerance of memristive neural networks is analysed. Two different memristor crossbar-based neural structures are considered to check the fault-tolerance of the memristor crossbars. One structure considered for this analysis is the 0T1M memristor crossbar structure, as explained in Section 2.3.1.4 and the other structure used for analysis is the memristive crossbar with constant-term circuit, which is described in Section 2.3.1.6.

6.2.1 Multi-layer Non-linear Separable Functions

The non-linear separable function is trained and tested in the presence of faulty memristors in a multi-layer memristive crossbar neural structure. The Schema presented in Fig. 6.1 and Fig. 6.2 are used to test this functionality. A 3-input logic function is trained and tested based on these memristive neural circuits. A 3→4→1 network configuration is used in these circuits. Three input neurons, four hidden neurons and one output neuron are considered. Here, X1, X2 and X3 are considered as inputs, while Y and b represent output and bias, respectively. Eight data patterns are applied as inputs to X1, X2 and X3. Input 1 is always applied to b (bias). The crossbar inputs are applied as a different set of voltages for each iteration of the training. Then the results are evaluated from the output of the comparator at the end of each column. The perceptron algorithm is used to train the networks. The comparator performs the threshold function in this circuit.

In the circuit shown in Fig. 6.2, a single memristor crossbar along with one constant term circuit is used. The voltage follower is used to calculate the output (Y_c) from the constant-term column and this output is applied to each comparator at the end of each column in memristor crossbar. Thus, this circuit provides both positive and negative polarity on the total dot product.

6.2.2 Faulty Memristors in Memristor Crossbar

Firstly, to observe fault tolerance within memristor structures, random memristors are set to high resistance states so that they cannot change their initial states. To avoid defects at a single synapse (pair of memristors), the memristors stuck at high resistance states are considered in either of the columns in the memristor structure. It was assured that at least one memristor should be working in each pair of synapses. Then, the memristor crossbars are trained with faulty memristors. From 10% to 50% of memristors are considered defective to observe the tolerance of each memristor neural learning structure. Four different test trials have been done on memristor

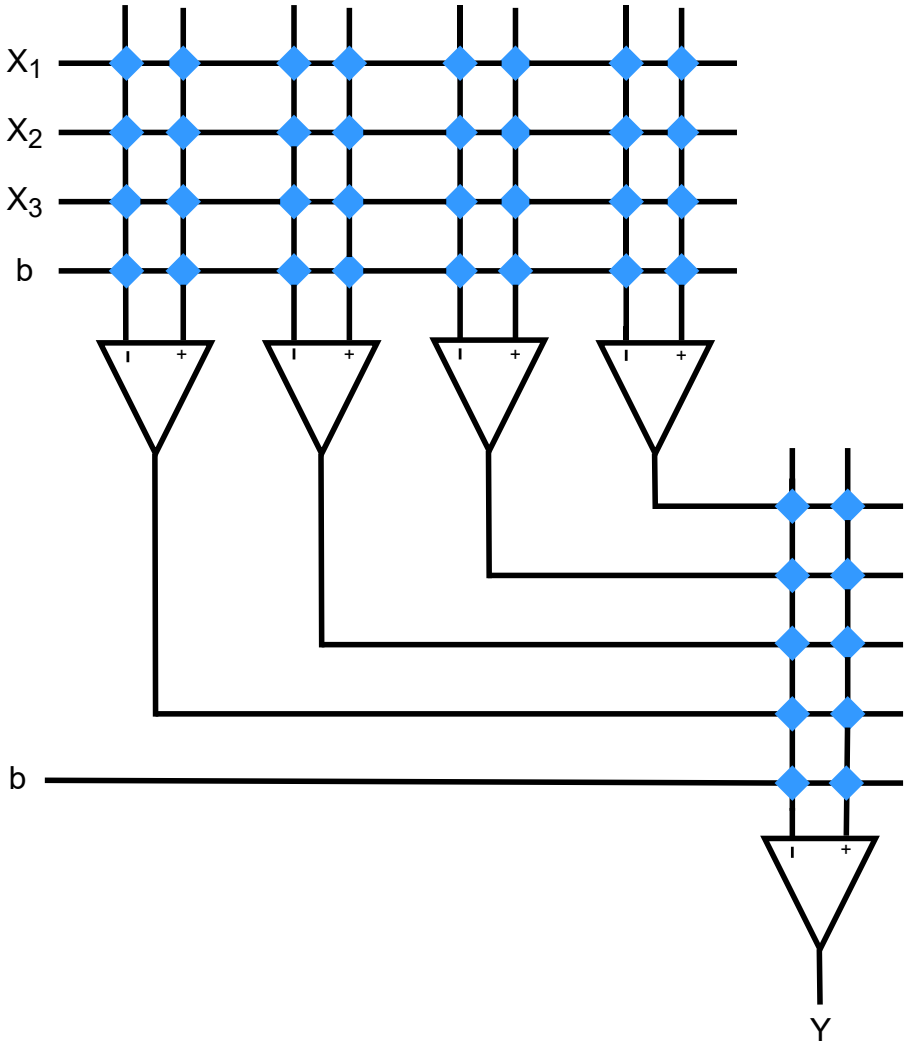


Figure 6.1: 0T1M Memristor crossbar-based neural circuit used for fault-tolerance.

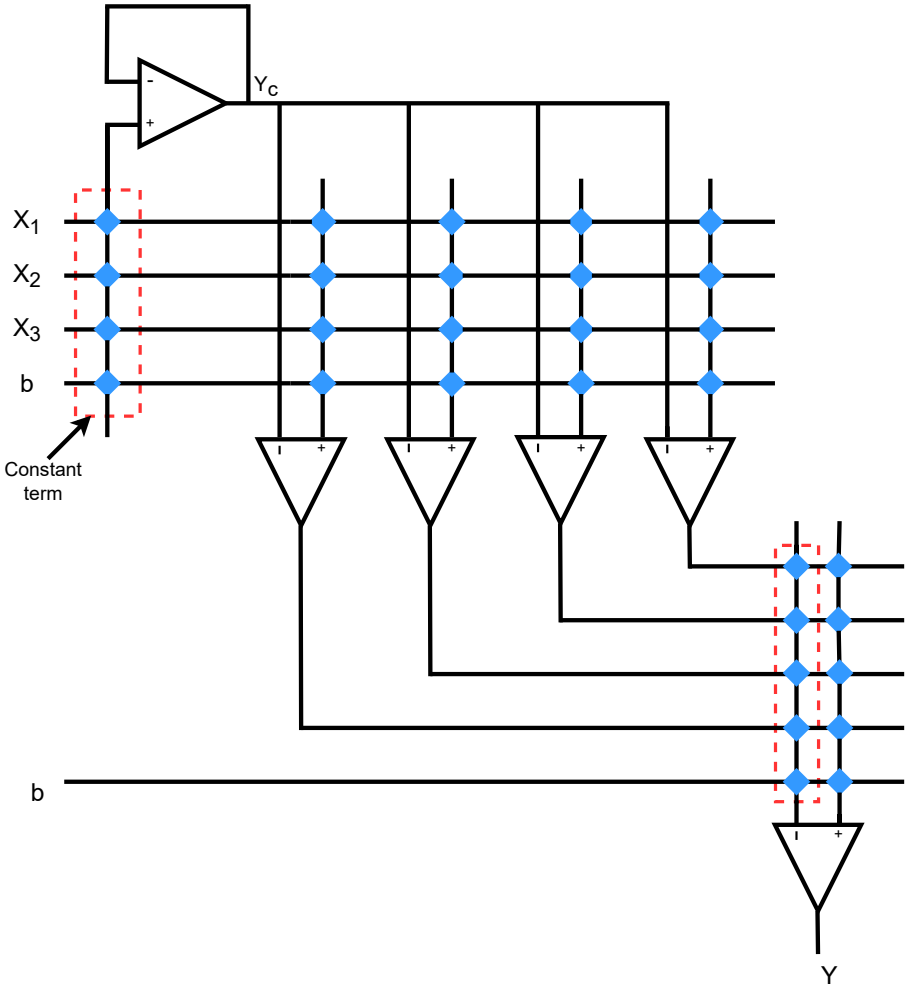


Figure 6.2: Memristor crossbar with constant-term circuit used for fault-tolerance.

crossbar structures with random weight initialization values. In a memristive neural structure with constant term, no faulty memristor was considered in constant term column to ensure that at least one memristor is working in a pair. Thus, in this circuit, the faulty memristors are only in the columns of a single crossbar. Table 6.1 shows the results of 0T1M memristive neural structures after training with faulty memristors. Table 6.2 shows the results of memristive neural structures with constant term circuit after training with faulty memristors. Fig. 6.3, Fig. 6.4, Fig. 6.5 and Fig. 6.6 show the epochs required for training of the non-linear separable function in different test trials for the 0T1M multi-layer neural circuit. Fig. 6.7, Fig. 6.8, Fig. 6.9 and 6.10 show the epochs required for training in each test for the multi-layer memristive neural circuit with constant-term for training the non-linear separable function. The results show that both the memristive structures with faulty memristors are able to learn non-separable functions with faulty memristors. In only one trial, the memristor neural structure with constant term was unable to learn after 230 epochs with 50% faulty memristors. In some test trials, the networks took longer time to learn than in other trials.

Table 6.1: Fault tolerance analysis in 0T1M memristor crossbar.

	Error Percentage	10%	20%	30%	40%	50%
Epoch	Test1	80	64	90	92	94
	Test2	197	237	99	64	198
	Test3	61	130	121	115	147
	Test4	36	55	28	92	135

Table 6.2: Fault tolerance analysis in memristor crossbar with constant-term circuit.

	Error Percentage	10%	20%	30%	40%	50%
Epoch	Test1	91	53	208	190	41
	Test2	80	90	99	163	**230**
	Test3	75	43	38	32	190
	Test4	45	57	60	90	163

To analyse the fault-tolerance of memristive neural circuits, the C++ and LT-spice is used. C++ is used for training and LTspice is used for memristor crossbar

implementation and calculations. It is better to represent the memristor grid more precisely by examining the real memristor circuit in LTspice. An existing memristor model published in [97, 98] was used in this work to model the memristor device in LTspice. This memristor device has been chosen due to its high minimum resistance value and large resistance ratio.

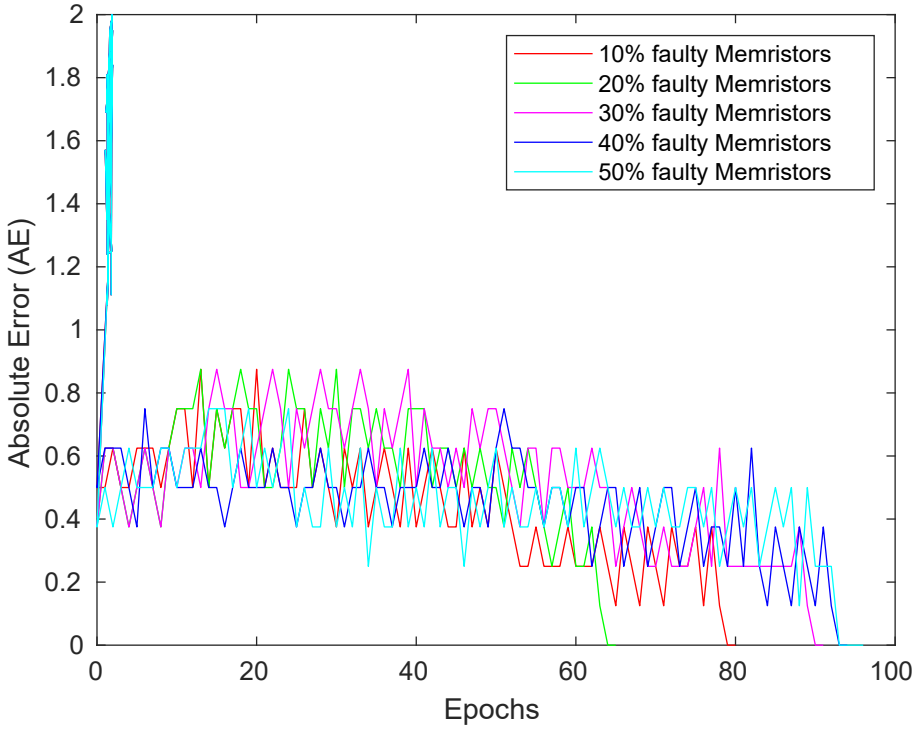


Figure 6.3: Error per epoch for test1 with varying faulty memristors in 0T1M neural circuit.

6.3 Yield Evaluation in Memristor Crossbar-based Neural Structures

Variations in conductance values, operational faults, manufacturing defects, and other factors can affect the performance of memristor-based neuromorphic systems [11, 86, 91]. Due to manufacturing defects, the memristors can be stuck-at a low or high resistive state, which can degrade the performance of memristor neural circuits [11, 86, 108].

In most cases, neural networks can accept a limited number of faulty synaptic

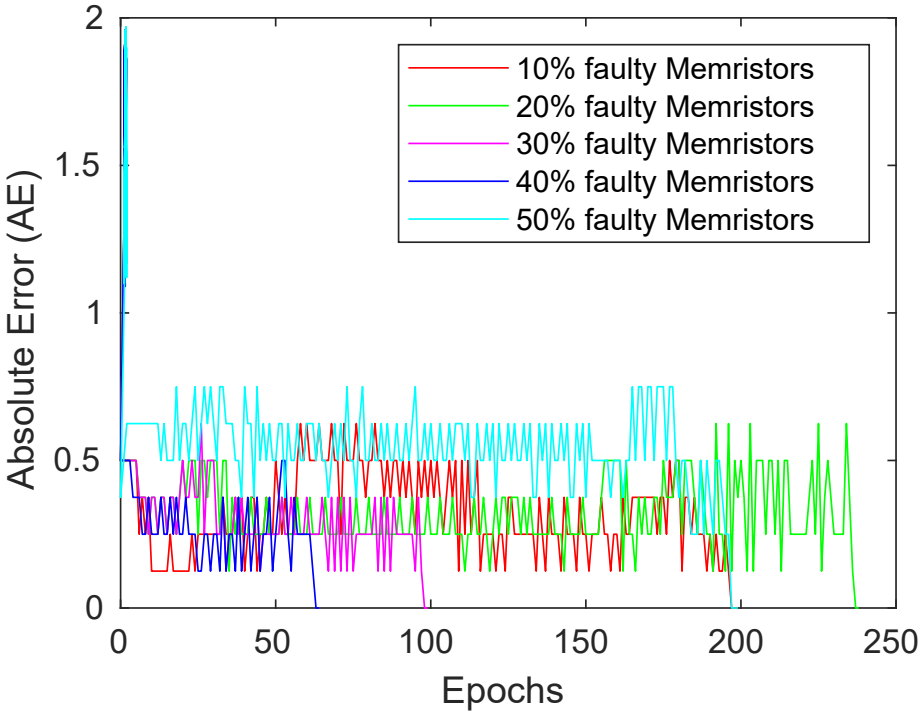


Figure 6.4: Error per epoch for test2 with varying faulty memristors in 0T1M neural circuit.

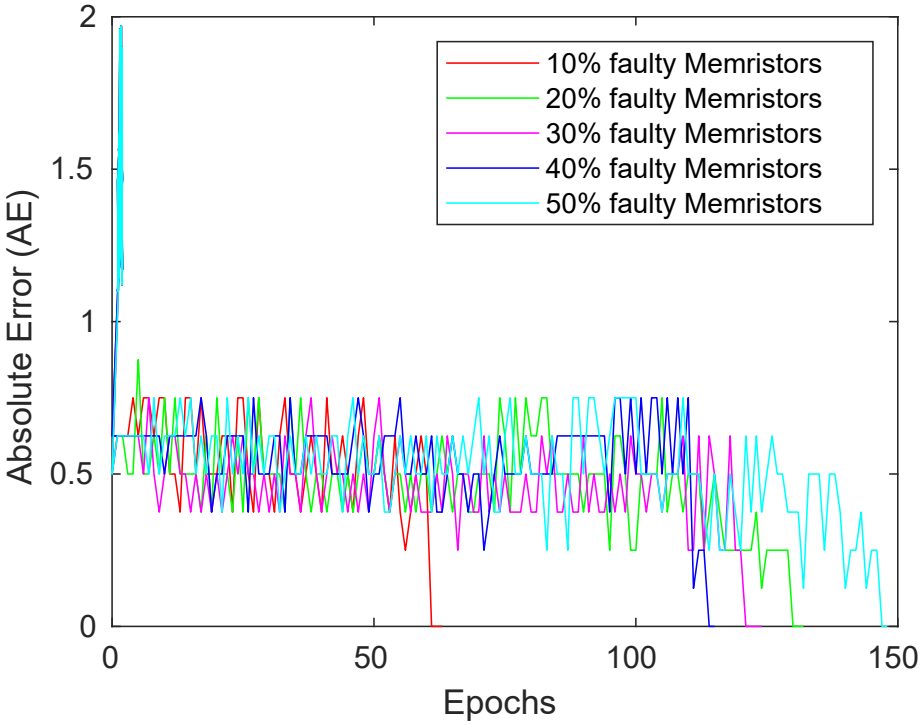


Figure 6.5: Error per epoch for test3 with varying faulty memristors in 0T1M neural circuit.

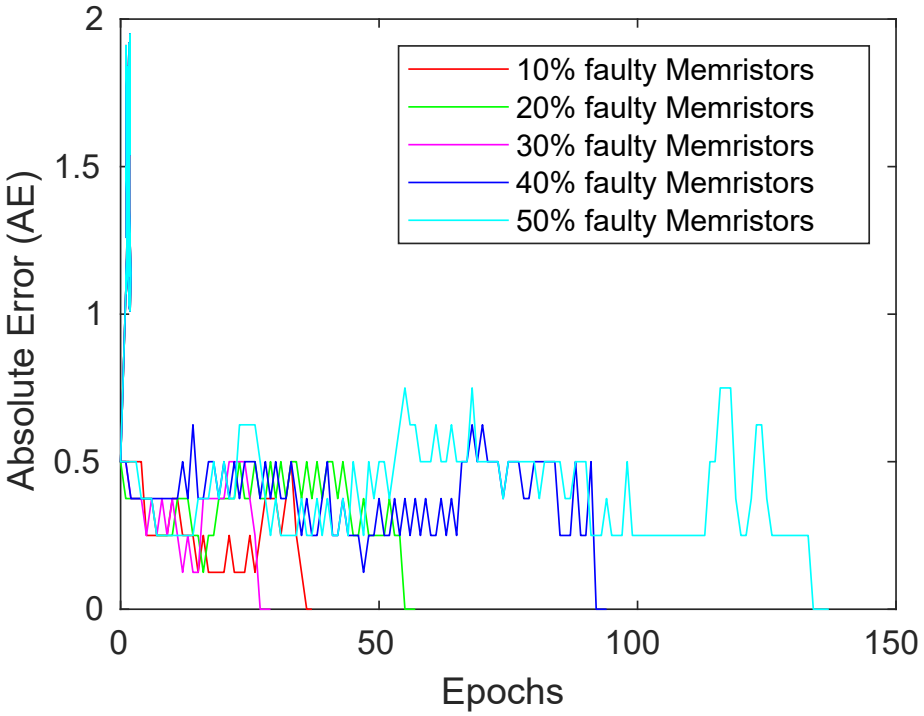


Figure 6.6: Error per epoch for test4 with varying faulty memristors in 0T1M neural circuit.

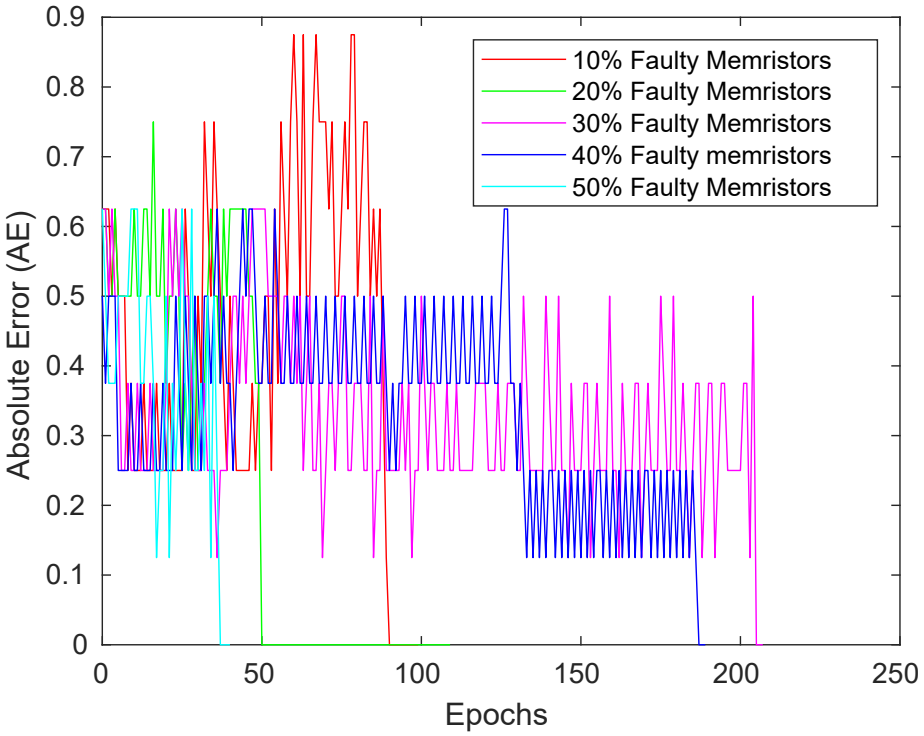


Figure 6.7: Error per epoch for test1 with varying faulty memristors in memristor neural circuit with constant term.

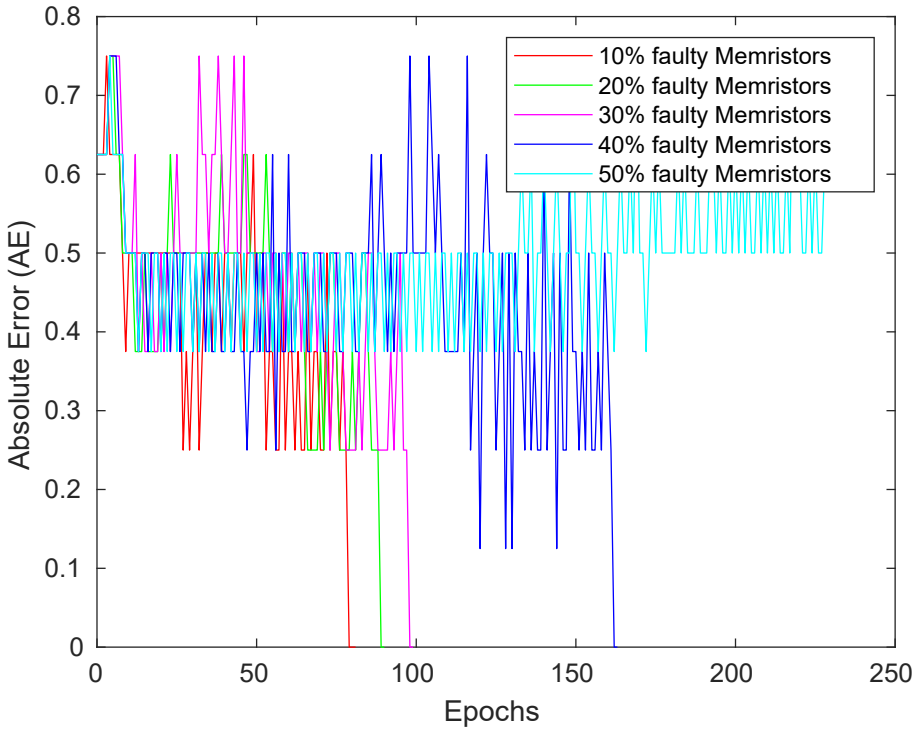


Figure 6.8: Error per epoch for test2 with varying faulty memristors in memristor neural circuit with constant term.

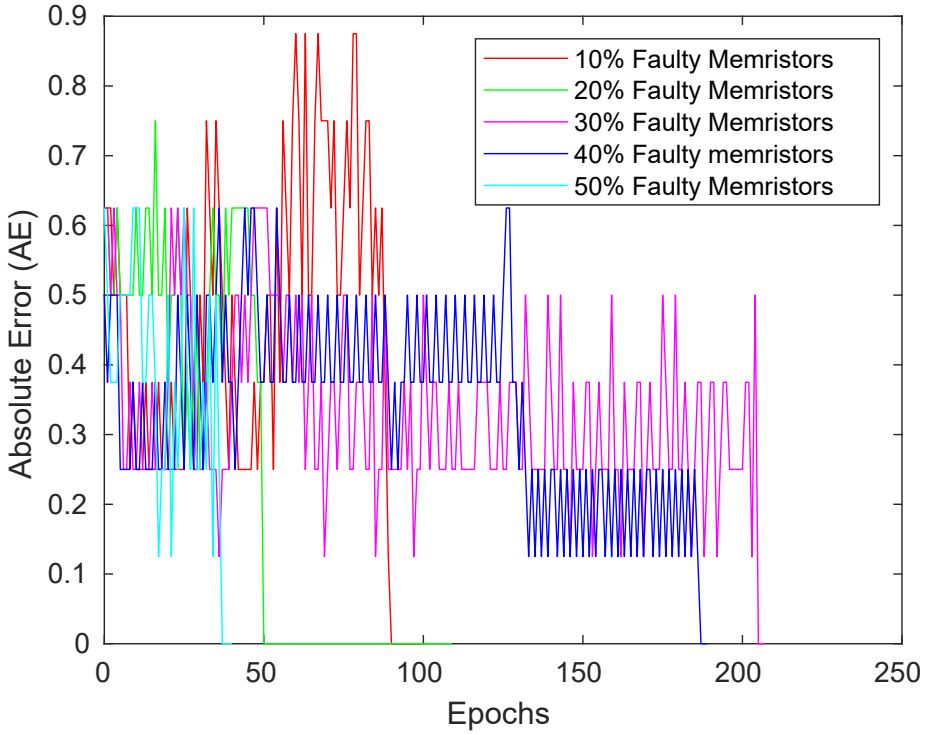


Figure 6.9: Error per epoch for test3 with varying faulty memristors in memristor neural circuit with constant term.

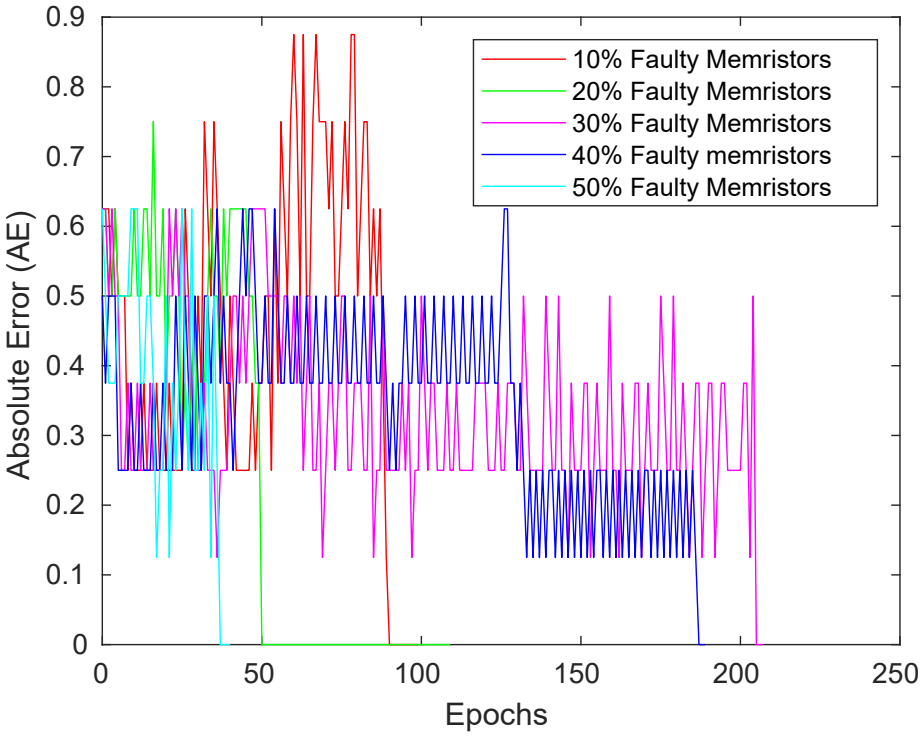


Figure 6.10: Error per epoch for test4 with varying faulty memristors in memristor neural circuit with constant term.

weights; however, a high defect rate dramatically reduces the accuracy of the matrix-vector calculations. The retraining and weight mapping processes are used to train the neural network with faulty memristors [12, 86, 107].

In this section, the yield of a memristor crossbar array used for neural networks is calculated using a Markov chain, which provides ease of use without sacrificing accuracy and representativeness. The benefit of the repair process is also considered while using the Markov chain model. The Poisson distribution approach proposed in [89] is also used for analysis and comparison as it is a faster industry-based approach for embedded SRAMs. Different aspects of these two evaluation techniques are also discussed.

6.3.1 Yield Evaluation Methods

In this section, we define some of the key terms used in the rest of the chapter. [16, 79, 80]. This section also explains the methods used for calculating yield.

- Yield: The yield is described as the probability of chip acceptability during the manufacturing process. As the number of elements increases, the risk of device failure also expands.

- Fault Types: The fault types are represented as $FT = (1, 2, ..., F)$ and the number of such types is denoted by F. These fault types were introduced in [80, 79]. The average number of type " i " faults is denoted by λ_i, where $(i = 1, 2, ..., F)$.

- Circuit Types: The "circuit types" are defined as $(1, 2.....l)$. Faults can occur in the "circuit types" and the number of such kinds is denoted by l.

6.3.1.1 General Yield Formula

The basic formula [80] based yield estimation for a chip with F potential fault types is defined by the fault pattern (FP) vector:

$$FP = (i_1, i_2, ..., i_F) \tag{6.1}$$

where, the average number of type " i " faults is denoted by λ_i, where $(i = 1, 2, ..., F)$.

Based on the above, the yield is defined by:

$$Y = \sum_{fixable FP_i} Prob(FP_i) \tag{6.2}$$

6.3.1.2 The Markov Chain Modelling

The Markov chains [59, 66] are a type of stochastic model that represents a series of probable events in which the next state's probabilities are entirely based on the events in the current state, not the previous states.

Definition: A Markov chain is a 'memoryless' discrete-time process in which all knowledge of the past states comprises the current state. This indicates that the

present state (at time t-1) is enough to predict the following state's probability (at time t).

A labelled directed graph $G = (V, E)$ can be used to describe a Markov chain with state-space V and transition matrix P, where the edges are defined by nonzero probability transitions.

$$E = (u, v)|P_{u,v} > 0 \tag{6.3}$$

Here, an edge from u to v is labelled by the probability $P_{u,v}$. The matrix will be $N \times N$ if the Markov chain has N potential states. This matrix's rows must add up to one. An $N \times 1$ Initial State Vector is also included in a Markov chain.

In Markov chains, a higher-order transition matrix is used to determine the probability of that transition occurring over a number of steps.

6.3.1.3 The Poisson Distribution

Let λ_0 be the mean number of faults of each type in a memristor crossbar array. The probability that a crossbar array has k faults is determined by the Poisson distribution function as shown in Eq. 6.4.

$$P(k) = \frac{e^{-\lambda_0}\lambda_0^k}{k!}, \quad for \;\; k = 0, 1, 2 \cdots \tag{6.4}$$

The probability of a chip having no faults is known as the yield, which is determined for k = 0 by Eq. 6.5, i.e., if there is no redundancy on the chip.

$$Y = P(0) = e^{-\lambda_0} \tag{6.5}$$

The additivity of faults is a particularly valuable characteristic of the Poisson distribution model. After repairing the memristor crossbar array with the stated redundancy, the yield is calculated as the sum of the probabilities of different types of faults by using Eq. 6.6.

$$Y = \sum P_{SA0}(i) P_{SA1}(j) \tag{6.6}$$

Here, the repair process is defined as a method that can correct the majority of the faults and Y is the yield after repairing various types of faults, e.g. (i+j).

6.3.2 Stuck-at-Faults Simulation

The probability of stuck-at-faults in the memristor crossbar is higher than the other faults, like operational faults and variation in conductance values, and these faults have an adverse effect on the performance of memristor crossbar-based neural networks [50, 91]. While a neural network can usually accept a limited number of faulty weights, a higher rate of SAFs, especially SAFs at low resistance states, drastically reduces the computational accuracy.

To mitigate this, re-training techniques have been proposed for memristive crossbar arrays that tolerate SAFs by utilizing the inherent fault tolerance of neural networks [12, 50]. These techniques are useful if a fault-tolerant network can be retrained with similar recognition accuracy. However, when the percentage of defects is higher than the neural network's natural fault tolerance, performance suffers.

To estimate the yield, the Markov model introduced in [60] is used in the memristor crossbar array. This model is based on the approach presented in [17] as it provides accurate results. In this evaluation, two types of stuck-at-faults are considered, termed $SA0$ (High Resistive State) and $SA1$ (Low Resistive State). $SA0$ is considered a repairable fault as it is assumed to be re-trained in the neural network. $SA1$ fault is considered an unrepairable fault as it is hard to re-train in the network as this type of faulty memristor produces the maximum amount of current that can pass through it. It is more challenging to train other memristors to have a resistance that is lower than this faulty memristor [50, 99]. We considered two and four spare columns to replace the faulty memristors in the crossbar array while evaluating yield.

First, a 4×4 memristor crossbar with two spare columns is considered for yield evaluation as shown in Fig. 6.11 and its corresponding state transition diagram is shown in Fig. 6.12. Here, circles denote the states and the edges denote the transition from one state to another. It starts from state G, which represents the 'GOOD' state and ends at state F, which represents the 'FAIL' state. 'GOOD' state implies no faults in the crossbar and 'FAIL' implies that there is no further state to consider and the system fails. One single fault is processed at a time. Each state is represented by (i, j) where i represents the spare column and j represents the repairable faults. The next state depends on the present state in this algorithm.

The transition rate, denoted by λ is represented by the edge between two transition states. It is the weighted sum of all possible faults that can produce that transition. The weight represents the number of elements that are affected by that fault. The transition rates with the Markov model are shown in Table 6.3. As shown in this table, the transition rate $\lambda_{G,(i^1,j^0)}$ is from a state with no spare column to a state with one spare column, where i represents the spare column and j represents the number of repairable faults. The transition rate is calculated by considering the faults that are responsible for generating that transition. For example, the transition rate $\lambda_{SA0(n_c n_r)}$ is the probability of having a fault in any row and column in the memristor crossbar, where the number of rows is indicated by n_r and the number of columns is indicated by n_c in Table 6.3. The transition rates for a 4×4 crossbar state diagram calculated using a Markov chain are illustrated in Table 6.3. The values of the fault rates must be divided by the number of columns or rows, whichever is greater (or their product). This is because, when describing a transition rate, all defects are considered to be independent and their weighted probabilities are assumed [60].

The yield is calculated by solving the following equation with continuous-time Markov chain (CTMC):

$$P = P(0).A^k \tag{6.7}$$

$P(0)$ is a vector whose elements are the probability of being in a given state $(G, 1, 2, ..., F)$,

and A is the generating matrix whose elements are the transition rates and k is the

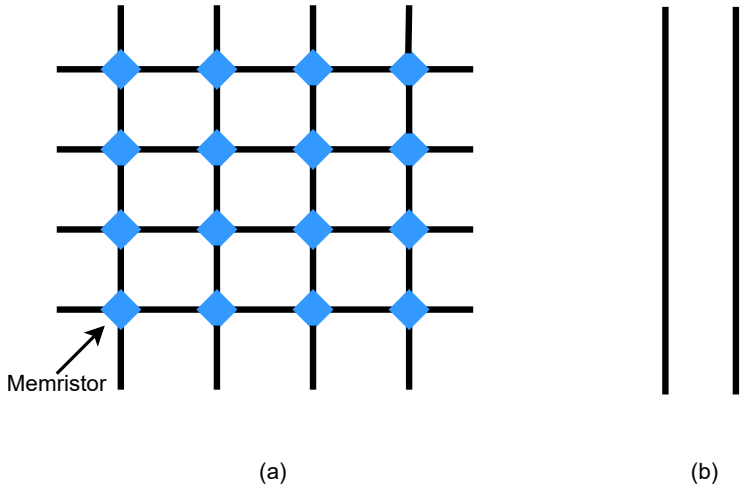


Figure 6.11: (a) A 4×4 memristor crossbar array (b) two spare columns.

6.3.3 Results and Discussion

The yield analysis of the memristor crossbar is done using MATLAB [54]. The simulation results for a 4×4 crossbar array are shown in Fig. 6.13. The average defect rate for this varies from 0 to 40 with a step size of $\Delta\lambda$=10^{-3}. The results show that the yield of a 4×4 crossbar with no spare column is less than with two spare columns. Additionally, different fault probabilities are considered to compare the yield of the crossbar as shown in Fig. 6.13. Three probability cases are considered as follows:

- $SA0$>$SA1$. In this case, we considered the ratio of the $SA0$ fault higher than the $SA1$ fault.
- $SA0$<$SA1$. In this case, the ratio of $SA1$ faults is considered higher than $SA0$ faults.

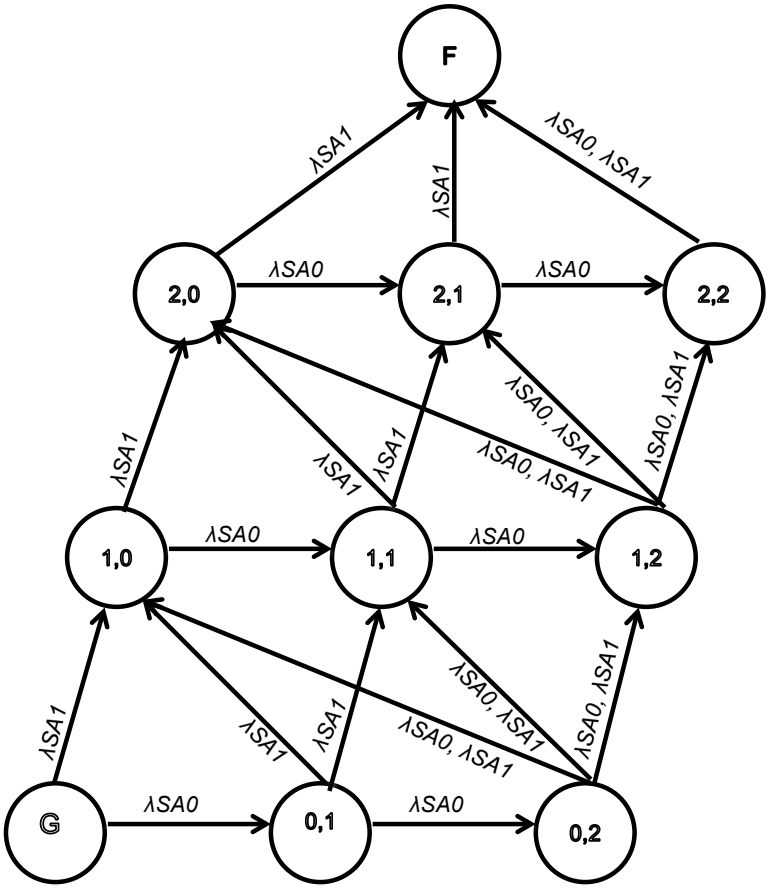


Figure 6.12: State diagram for 4×4 memristor crossbar array.

Table 6.3: Transition Table

Transition Rates	Weighted sum of probable faults
$\lambda_{G,(i^0,j^1)}$	$\lambda_{SA0}(n_c n_r)$
$\lambda_{G,(i^1,j^0)}$	$\lambda_{SA1}(n_c n_r)$
$\lambda_{(i^0,j^n),(i^0,j^{n+1})}$	$\lambda_{SA0}(n_c n_r - n)$
$\lambda_{(i^0,j^n),(i^1,j^0)}$	$\lambda_{SA1}(n_c - (n_c - 1))(n_r - n)$
$\lambda_{(i^0,j^{n=m}),(i^1,j^0)}$	$\lambda_{(SA1+SA0)}(n_c - (n_c - 1))(n_r - n)$
$\lambda_{(i^0,j^n),(i^1,j^n)}$	$\lambda_{SA1}(n_c - n)n_r$
$\lambda_{(i^0,j^{n=m}),(i^1,j^n)}$	$\lambda_{(SA1+SA0)}(n_c - n)n_r$
$\lambda_{(i^1,j^{n-1}),(i^1,j^n)}$	$\lambda_{SA0}(n_c - 1)n_r - (n - 1)$
$\lambda_{(i^1,j^{n-1}),(i^2,j^0)}$	$\lambda_{SA1}(n_c - n)n_r$
$\lambda_{(i^1,j^{n=m}),(i^2,j^0)}$	$\lambda_{(SA1+SA0)}(n_c - n)n_r$
$\lambda_{(i^1,j^n),(i^2,j^{n-1})}$	$\lambda_{SA1}(n_c - (n_c - 1))(n_r - n)$
$\lambda_{(i^1,j^{n=m}),(i^2,j^{n-1})}$	$\lambda_{(SA1+SA0)}(n_c - (n_c - 1))(n_r - n)$
$\lambda_{(i^2,j^{n-1}),(i^2,j^n)}$	$\lambda_{SA0}(n_c - 2)n_r - (n - 1)$
$\lambda_{(i^2,j^{n-1}),F}$	$\lambda_{SA1}(n_c - 2)n_r - (n - 1)$

- $SA0$=$SA1$. In this case, the ratio of both faults is considered equal.

With the increasing percentage of $SA0$ faults the yield increases. The yield is higher in the case where the ratio of $SA0$ faults is higher.

As we discussed in the previous section, if the percentage of SAFs increases beyond the internal tolerance of the neural network, then the computational accuracy decreases. Additionally, SAF probabilities increase with the larger array sizes. The yield is calculated for different-sized memristor crossbar arrays. The chosen array sizes are 32×32, 128×128, and 256×256. For all these array sizes, the yield has been calculated with zero and two spare columns and different fault ratios have been considered. The plots reported in Fig. 6.14, Fig. 6.15, and Fig. 6.16 show the yield for 32×32, 128×128, and 256×256 memristor crossbar arrays with and without redundancies. In Fig. 6.14, the value of λ_0 is varying from 0 to 100 with a varying step of 10^{-3}. The value of λ_0 varies from 0 to 40 in the plots reported in Fig. 6.15 and Fig. 6.16. All the resulting plots show that the yield is higher with spare columns as compared to no spare columns. In terms of fault ratio, results show that with the increase in repairable fault ratio, the yield increases. Fig. 6.17 and Fig. 6.18

illustrate the yield with two, four and no spare columns for the array sizes 256×256 and 512×512, respectively. Both SAFs are considered equal in these cases and the value of λ_0 is varied from 0 to 30. The results demonstrate that adding more redundancies could boost the yield. As a result, crossbar array yield can be improved by using effective re-training methods for memristive neural networks with SAFs and redundancies.

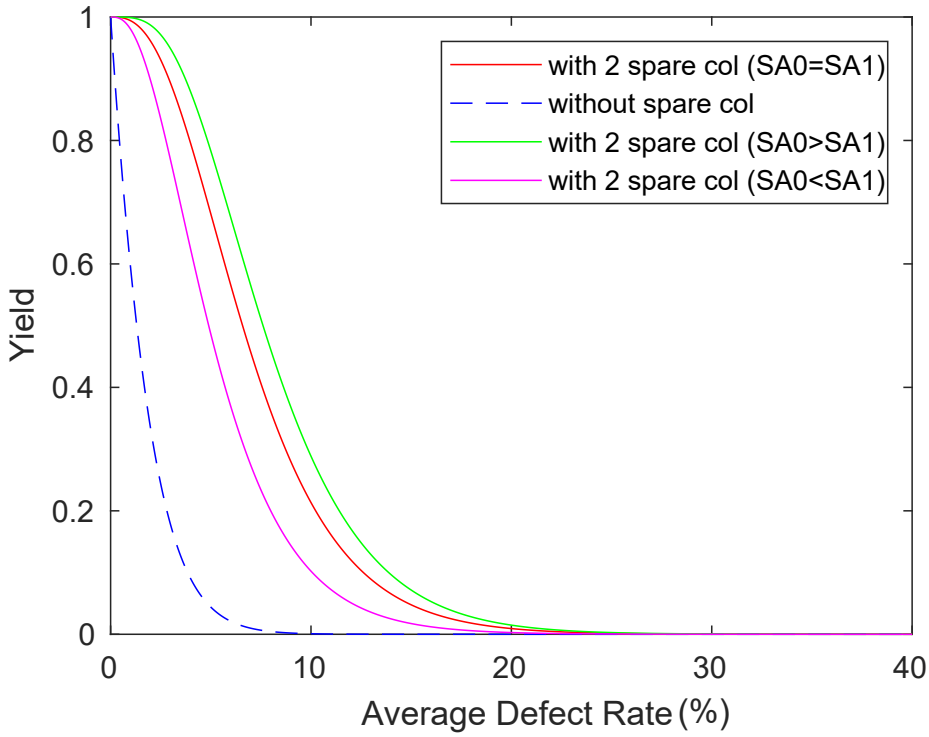


Figure 6.13: Yield evaluation with varying fault ratio for 4×4 memristor crossbar array.

Table 6.4: Comparison table for array size 4×4.

	λ_0=0.2			λ_0=0.4			λ_0=2			λ_0=4		
	PD	MC	Difference	PD	MC	Difference	PD	MC	Difference	PD	MC	Difference
no spare col	0.9047	0.9042	-0.0005	0.8178	0.8162	-0.0016	0.3383	0.3323	-0.006	0.0916	0.0898	-0.0018
two spare col	0.9998	0.9999	1e-04	0.9988	0.9996	0.0008	0.891	0.9606	0.0696	0.5312	0.7945	0.2633
four spare col	1	1	0	1	1	0	0.9930	1	0.007	0.8730	1	0.127

Table 6.5: Comparison table for array size 32×32.

	λ_0=0.2			λ_0=0.4			λ_0=2			λ_0=4		
	PD	MC	Difference	PD	MC	Difference	PD	MC	Difference	PD	MC	Difference
no spare col	0.8178	0.8177	-1e-04	0.8178	0.8177	-1e-04	0.3383	0.3371	-0.0012	0.0916	0.0903	-0.0013
two spare col	0.9988	0.9989	1e-04	0.9988	0.9989	1e-04	0.891	0.9054	0.0144	0.5312	0.5711	0.0399
four spare col	1	1	0	1	1	0	0.9930	0.9951	0.0021	0.8730	0.9040	0.031

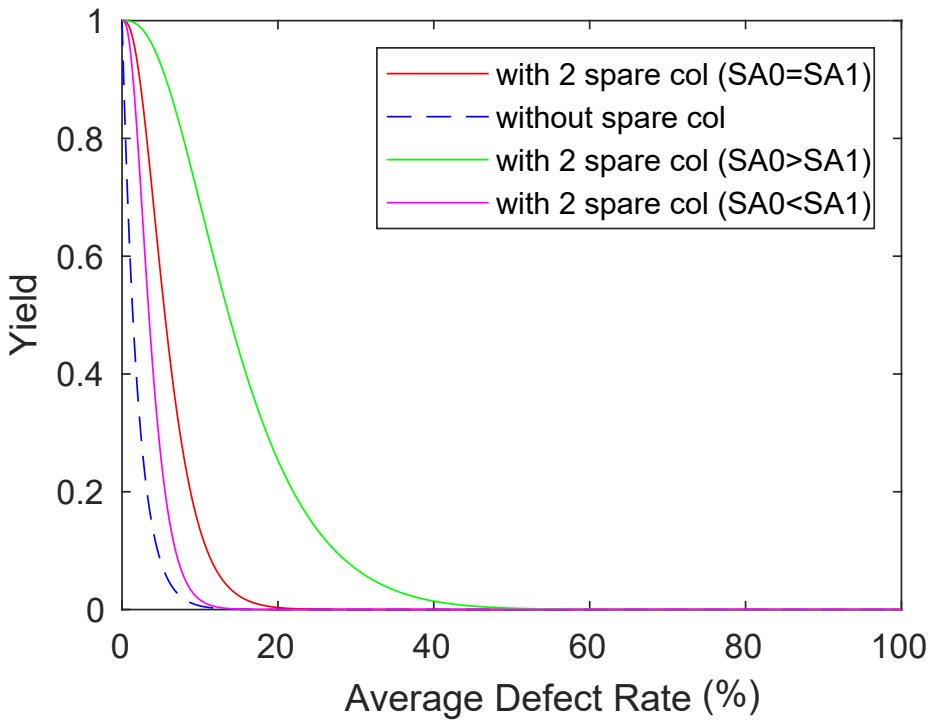


Figure 6.14: Yield evaluation with varying fault ratio for 32×32 memristor crossbar array.

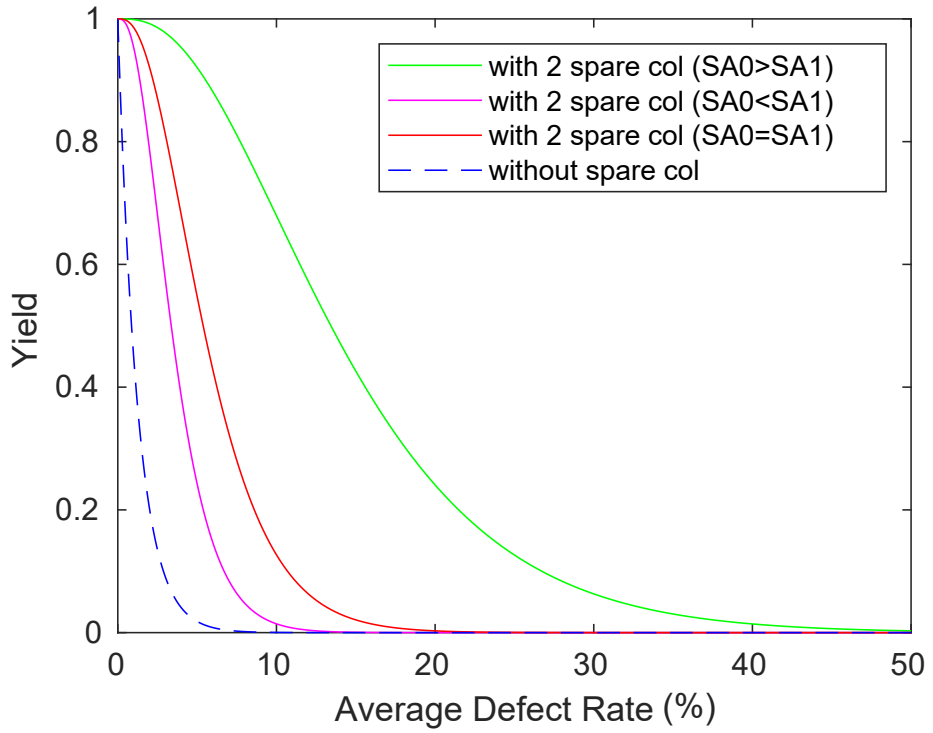


Figure 6.15: Yield evaluation with varying fault ratio for 128×128 memristor crossbar array.

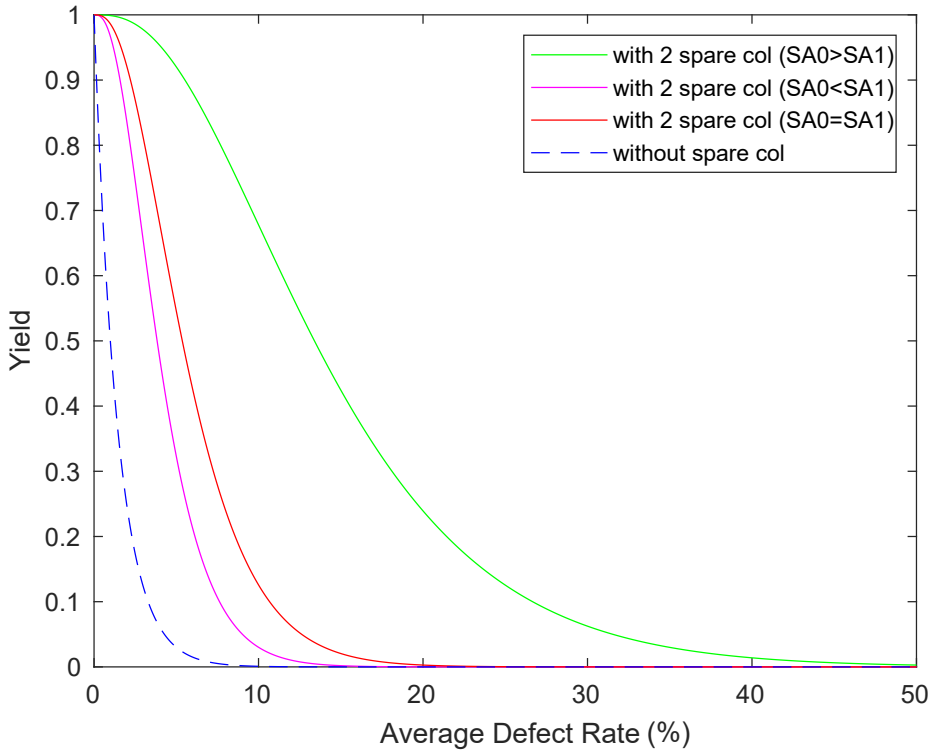


Figure 6.16: Yield evaluation with varying fault ratio for 256×256 memristor crossbar array.

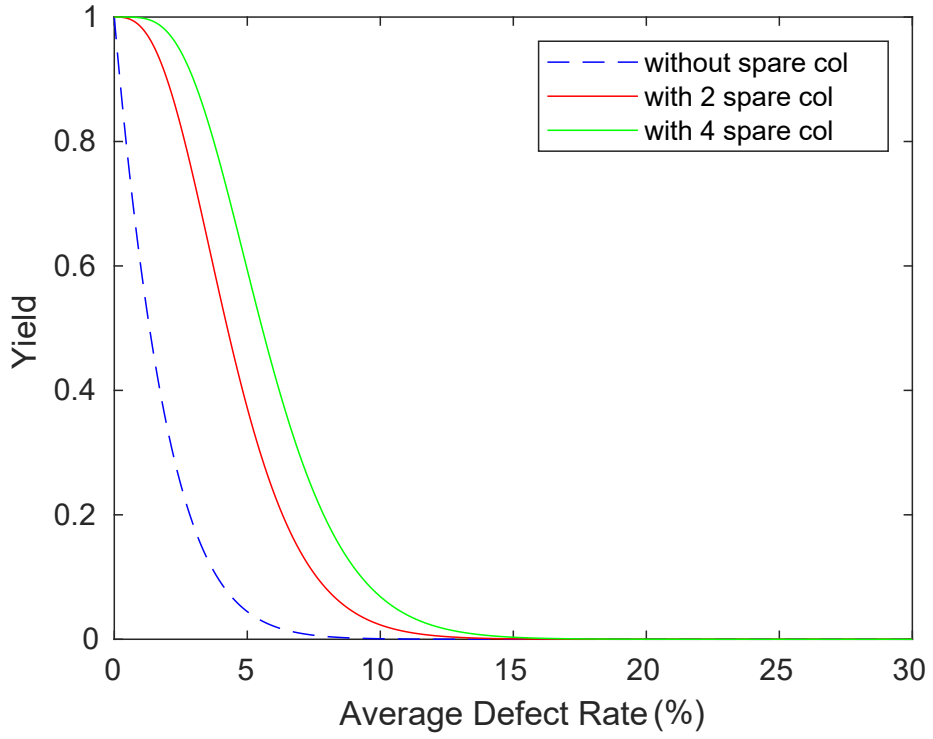


Figure 6.17: Yield evaluation for 256×256 memristor crossbar array.

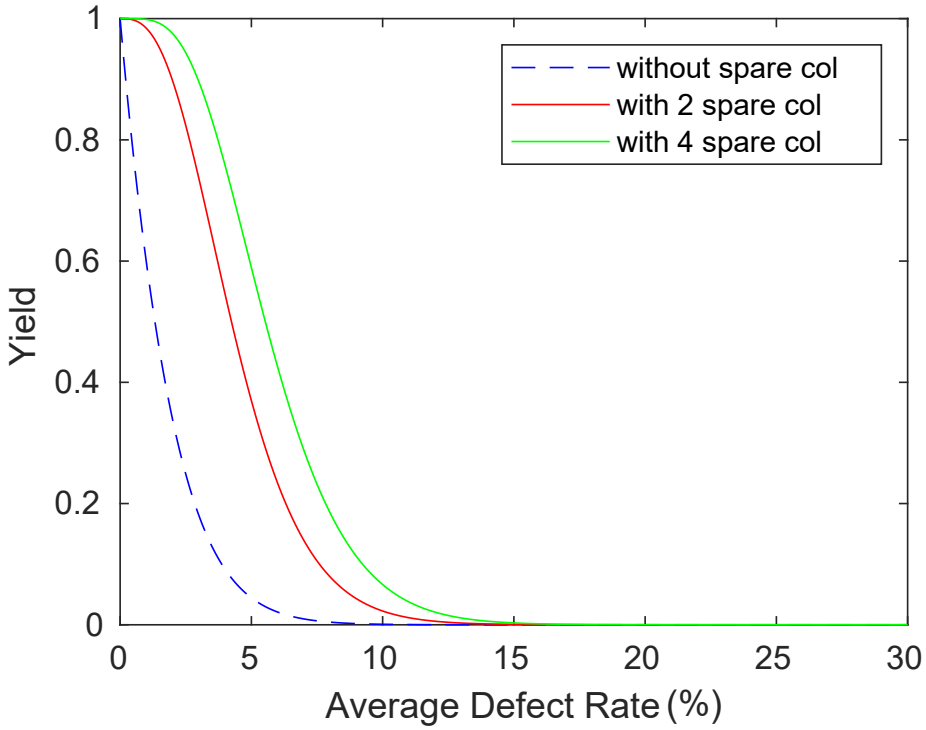


Figure 6.18: Yield evaluation for 512×512 memristor crossbar array.

Table 6.6: Comparison table for array size 128×128.

	λ_0=0.2			λ_0=0.4			λ_0=2			λ_0=4		
	PD	MC	Difference	PD	MC	Difference	PD	MC	Difference	PD	MC	Difference
no spare col	0.9047	0.9047	0	0.8178	0.8177	-1e-04	0.3383	0.3379	-0.0004	0.0916	0.0911	-0.0005
two spare col	0.9998	0.9998	0	9988	9989	1e-04	0.891	0.8988	0.0078	0.5312	0.5501	0.0189
four spare col	1	1	0	1	1	0	0.9930	0.9936	0.0006	0.8730	0.8812	0.0082

6.4 Comparison

The yield calculated by using two methods is compared in this section. For comparison, the Poisson distribution [89] and the Markov chain model [60] are used to calculate yield. Many characteristics of these two methods are different. The Markov chain uses a repair procedure that evolves in real-time, whereas the Poisson distribution uses a static probabilistic analysis. Yield is evaluated using these methods by considering different average defect rates and array sizes. The results are calculated and compared using two, four and no spare columns as shown in Table 6.4, Table 6.5 and Table 6.6. In these tables, PD represents the Poisson distribution and MC represents the Markov chain method. The execution processes of these two methods are also different in terms of memory size and aspect ratio. The same fault ratio is

used in both methods. However, in terms of memory size and aspect ratio, the Poisson distribution method does not contain these. In terms of complexity, the Markov chain approach is more complex than the Poisson distribution as it uses matrix multiplication as shown in Eq. 6.7. However, the number of matrix multiplications in this equation is proportional to the desired accuracy, which is higher for lower values of execution steps $\Delta\lambda$ and the highest value of the average fault number λ_0. On the other hand, for available spare columns and a given number of faults, the Poisson distribution method determines if there is a repair configuration authorised by the available spare columns for each possible defect. As a result, it is required to compute fewer operations than the Markov chain. However, the Poisson distribution ignores the fault overlaps; it does not consider two horizontal pairs overlapping or two stuck at faults on the same column. As the position of the faults also affects the vector-matrix multiplication in memristive neural networks. We considered a lower to higher average defect rate for comparison and these values are considered based on the data used in [60]. The evaluated yield results for comparison are shown in Table 6.4, Table 6.5 and Table 6.6. The results show that both methods produce close or similar results for smaller values of λ_0 and the yield calculated by using the Poisson distribution is underestimated for higher values of average defect rate. As a result, higher defect rates should be considered in order to obtain more accurate results, and the Markov chain method is more effective for memristor crossbar based neural networks as it considers fault positions and array sizes while evaluating the yield and provides more accurate results for larger array sizes.

6.5 Summary

In the first part of this chapter, the fault-tolerance of memristive neural networks is analysed in the presence of faulty memristors. These faulty memristors are considered to be stuck at high resistance states. Then, the network is trained along with these faulty memristors in the crossbar array. A number of trials were conducted to check the performance of memristive crossbar neural circuits in the presence of faulty memristors. C++ and LTspice are used to train and test these networks. The results of fault-tolerance show that both the networks are able to learn most of the non-separable functions even with 50% faulty memristors. However, it takes a longer time to learn. Some of the tests learn faster than others. In the second part of this chapter, the yield of a memristor crossbar array is calculated based on a Markov chain approach. Yield is estimated for different sized memristor crossbar arrays with varying fault ratios. A Poisson distribution approach is also considered to evaluate the yield and compare. The yield is calculated by considering zero, two, and four redundancies for chosen array sizes. The Markov chain is more complex than the Poisson distribution in terms of complexity. However, in terms of flexibility and accuracy, the Markov chain provides more accuracy with a higher average defect rate, as the results are shown. It is more flexible because the crossbar array can consider defect positions, which is also helpful in re-training methods used for memristive crossbar neural networks with stuck-at-faults. Thus, to obtain more accuracy, higher values of the average defect rate should be chosen. Hence, the Markov chain can be more effective for calculating the yield for memristive crossbar arrays used for deep neural networks because it provides more accurate results with a higher average defect rate.

Chapter 7

Conclusions and Future Work

Memristors have the ability to be fabricated with a higher synaptic density than brain tissue due to their physical layout. A Memristor can be laid out as a high-density crossbar. The key advantage of the memristor crossbar is that it performs a large number of multiply-add computations in parallel in the analog domain. This thesis presented the learning method and activation function for efficient design and implementation of memristive neural networks. The chapters in this thesis also analysed the fault-tolerance and evaluated the yield of memristor crossbar-based neural networks in the presence of stuck-at-faults (SAFs).

7.1 Summary of the Thesis

The work presented in this thesis represents the design and implementation of memristor-based neural networks. It covered different areas of memristive neural networks that are explained below:

- Electronic Design Automation (EDA) tools increase the complexity of connectivity of massive neural networks and take more time to simulate. As the size of the memristor crossbar increases, the simulation time also increases. Therefore, a novel implementation of memristor models in a high-level language, especially in C++ was proposed in this thesis. Various single-layer and multi-layer memristive neural networks were simulated successfully using this modelling

approach. The implementation of memristive neural networks in this platform reduces the simulation time. Using this environment, the circuit was simulated faster than with EDA tools. Thus, it motivates the successful and faster simulation of massive neural networks using this platform in the near future.

- Recent studies show that different mapping schemes and neuromorphic circuit designs are required for ex-situ to achieve more accurate computations between hardware and software. This thesis presented an improved training method for ex-situ. The proposed training method is based on the voltage-divider technique and the backpropagation algorithm. The simulation results have successfully demonstrated the classification of linear and non-linearly separable problems with consideration of sneak-paths and provided more accuracy during the implementation of the ex-situ method in memristive neural networks. The results also showed a reduced learning time in software.

- The majority of activation functions in memristive neural networks are implemented using only operational amplifiers, which incur significant hardware overhead. A novel circuit design for the activation function of neural networks was proposed in this thesis. This circuit was designed using memristors, which approximated the ReLU activation function. The results showed that the proposed architecture required significantly lower hardware compared to existing approaches and helped to speed up the training process due to the simplicity of the underlying activation function.

- Stuck-at-faults have a significant impact on the computational accuracy of memristor-based neural networks. This thesis presented the fault-tolerance analysis and yield evaluation of memristor-based neural structures in the presence of SAFs. The results of the fault-tolerance tests showed that the memristive neural networks are able to learn the majority of non-separable functions even with 50% faulty memristors. The results of the yield evaluation showed that

the yield can be improved by adding redundancies and increasing the SA0 fault ratio.

- Various parameterized simulation tools in C++ and MATLAB were developed to aid various research contributions in this work. Working with emerging technologies is currently hindered by the limitations of existing EDA tools, specifically for designing new devices like memristors. Memristor models are not included in these EDA tools like resistors, capacitors and other components. Thus, SPICE memristor models were developed to aid simulation with EDA tools that made them difficult to converge. Thus, all these limitations slow down the design and simulation processes of larger crossbar circuits.

7.2 Future Research

Although the proposed work in this thesis demonstrated the successful implementation of memristive neural networks, there are still several opportunities to improve and extend the work. This section lists some of those opportunities:

- The simulation of memristive neural networks with C++ does not consider any sneak-paths and line resistances in chapter3. This can be done by inducing sneak-paths and line resistances in these simulations.

- A smaller set of multi-layer neural networks has been implemented and tested for this improved learning method. The massive memristive neural networks can be trained and tested using this learning method in ex-situ. Moreover, this learning method can be used with some improved weight mapping schemes for larger and more challenging datasets in memristive neural networks.

- The novel circuit design for activation function is used in ex-situ training method. It can also be implemented and demonstrated in an in-situ training method by considering larger datasets for deep neural networks.

- The fault-tolerance analysis of memristor-based neural circuits considers faults in either of the columns to ensure that one memristor in each pair should be working. Faulty memristors can be considered as a pair to check performance and the performance can also be checked with faults stuck in low resistance states. For yield analysis, more detailed analysis can be done using larger array sizes and more methods can be considered for evaluating yield.

Bibliography

[1] S. P. Adhikari, H. Kim, R. K. Budhathoki, C. Yang, and L. O. Chua. A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(1):215–223, Jan 2015.

[2] Shyam Prasad Adhikari, Changju Yang, Hyongsuk Kim, and Leon O. Chua. Memristor bridge synapse-based neural network and its learning. *IEEE Transactions on Neural Networks and Learning Systems*, 23(9):1426–1435, 2012.

[3] Ahmad Afifi, Ahmad Ayatollahi, and Farshid Raissi. STDP implementation using memristive nanodevice in CMOS-Nano neuromorphic networks. *IEICE Electronics Express*, 6(3):148–153, 2009.

[4] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23(7):075201, 2012.

[5] Fabien Alibart, Elham Zamanidoost, and Dmitri B Strukov. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nature communications*, 4, 2013.

[6] D Biolek, V Biolkova, and Z Biolek. SPICE model of memristor with nonlinear dopant drift. *Radioengineering*, 2009.

[7] Julien Borghetti, Gregory S Snider, Philip J Kuekes, J Joshua Yang, Duncan R Stewart, and R Stanley Williams. Memristive switches enable stateful logic operations via material implication. *Nature*, 464(7290):873–876, 2010.

[8] Arturo Buscarino, Luigi Fortuna, Mattia Frasca, and Lucia Valentina Gambuzza. A chaotic circuit based on hewlett-packard memristor. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(2):023136, 2012.

[9] Michael Bushnell and Vishwani Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science & Business Media, 2004.

[10] Kurtis D. Cantley, Anand Subramaniam, Harvey J. Stiegler, Richard A. Chapman, and Eric M. Vogel. Neural learning circuits utilizing nano-crystalline silicon transistors and memristors. *IEEE Transactions on Neural Networks and Learning Systems*, 23(4):565–573, 2012.

[11] Ching-Yi Chen, Hsiu-Chuan Shih, Cheng-Wen Wu, Chih-He Lin, Pi-Feng Chiu, Shyh-Shyuan Sheu, and Frederick T Chen. Rram defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Transactions on Computers*, 64(1):180–190, 2014.

[12] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 19–24, 2017.

[13] Myonglae Chu, Byoungho Kim, Sangsu Park, Hyunsang Hwang, Moongu Jeon, Byoung Hun Lee, and Byung-Geun Lee. Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron. *IEEE Transactions on Industrial Electronics*, 62(4):2410–2419, 2015.

[14] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[15] Leon Chua. Resistance switching memories are memristors. *Applied Physics A*, 102(4):765–783, 2011.

[16] B Ciciani and G Iazeolla. A straightforward yield model for fault-tolerant vlsi memory chips. In *IFIP TC-10 Conference on "Design Methodology in VLSI and computer architecture," Sept*, 1988.

[17] Bruno Ciciani and Giuseppe Iazeolla. A markov chain-based yield formula for vlsi fault-tolerant chips. *IEEE transactions on computer-aided design of integrated circuits and systems*, 10(2):252–259, 1991.

[18] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[19] Fernando Morgado Dias, Ana Antunes, and Alexandre Manuel Mota. Artificial neural networks: a review of commercial hardware. *Engineering Applications of Artificial Intelligence*, 17(8):945–952, 2004.

[20] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[21] Sorin Draghici. Neural networks in analog hardware—design and implementation issues. *International journal of neural systems*, 10(01):19–42, 2000.

[22] Dheeru Dua and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[23] Hazem Elgabra, Ilyas AH Farhat, Ahmed S Al Hosani, Dirar Homouz, and Baker Mohammad. Mathematical modeling of a memristor device. In *IEEE International Conference on Innovations in Information Technology (IIT)*, pages 156–161, 2012.

[24] Ahmed A. Emara, Mohamed M. Aboudina, and Hossam A. H. Fahmy. Corrected and accurate verilog-a for linear dopant drift model of memristors. In *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 499–502, 2014.

[25] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[26] Raqibul Hasan, Tarek M. Taha, and Chris Yakopcic. On-chip training of memristor crossbar based multi-layer neural networks. *Microelectron. J.*, 66(C):31–40, August 2017.

[27] Jennifer Hasler and Harry Bo Marr. Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in neuroscience*, 7:118, 2013.

[28] Mark Horowitz. Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014.

[29] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication. In *2016*

53nd acm/edac/ieee design automation conference (dac), pages 1–6. IEEE, 2016.

[30] Giacomo Indiveri and Timothy K Horiuchi. Frontiers in neuromorphic engineering. *Frontiers in neuroscience*, 5:118, 2011.

[31] Makoto Itoh and Leon O Chua. Memristor oscillators. *International journal of bifurcation and chaos*, 18(11):3183–3206, 2008.

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

[33] Sung Hyun Jo and Wei Lu. Cmos compatible nanoscale nonvolatile resistance switching memory. *Nano Letters*, 8(2):392–397, 2008. PMID: 18217785.

[34] Yogesh N Joglekar and Stephen J Wolf. The elusive memristor: properties of basic electrical circuits. *European Journal of physics*, 30(4):661, 2009.

[35] J Joshua Yang, M-X Zhang, Matthew D Pickett, Feng Miao, John Paul Strachan, Wen-Di Li, Wei Yi, Douglas AA Ohlberg, Byung Joon Choi, Wei Wu, et al. Engineering nonlinearity into memristors for passive crossbar applications. *Applied Physics Letters*, 100(11):113501, 2012.

[36] Seul Jung and Sung su Kim. Hardware implementation of a real-time neural network controller with a dsp and an fpga for nonlinear systems. *IEEE Transactions on Industrial Electronics*, 54(1):265–271, 2007.

[37] Irina Kataeva, Farnood Merrikh-Bayat, Elham Zamanidoost, and Dmitri Strukov. Efficient training algorithms for neural networks based on memristive crossbar circuits. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[38] Hyongsuk Kim, Maheshwar Pd. Sah, Changju Yang, Tamás Roska, and Leon O. Chua. Memristor bridge synapses. *Proceedings of the IEEE*, 100(6):2061–2070, 2012.

[39] Martin Klimo and Ondrej Such. Memristors can implement fuzzy logic. *CoRR*, abs/1110.2074, 2011.

[40] C Kügeler, M Meier, R Rosezin, S Gilles, and R Waser. High density 3d memory architecture based on the resistive switching effect. *Solid-state electronics*, 53(12):1287–1292, 2009.

[41] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser. Team: Threshold adaptive memristor model. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):211–221, 2013.

[42] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny. Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, 2015.

[43] S. Kvatinsky, K. Talisveyberg, D. Fliter, A. Kolodny, U. C. Weiser, and E. G. Friedman. Models of memristors for SPICE simulations. In *IEEE 27th Convention of, Electrical Electronics Engineers in Israel (IEEEI)*, pages 1–5, Nov 2012.

[44] Shahar Kvatinsky, Keren Talisveyberg, Dmitry Fliter, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Verilog-a for memristor models. *CCIT Technical Report*, 801, 2011.

[45] Mika Laiho, Eero Lehtonen, Alex Myles Thomas Russell, and Piotr Dudek. Memristive synapses are becoming reality. 2010.

[46] Quoc V Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

[47] Boxun Li, Yuzhi Wang, Yu Wang, Yiran Chen, and Huazhong Yang. Training itself: Mixed-signal training acceleration for memristor-based neural network. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 361–366. IEEE, 2014.

[48] Eike Linn, Roland Rosezin, Carsten Kügeler, and Rainer Waser. Complementary resistive switches for passive nanocrossbar memories. *Nature materials*, 9(5):403–406, 2010.

[49] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. Vortex: Variation-aware training for memristor x-bar. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[50] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai Li. Rescuing memristor-based neuromorphic design with high defects. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[51] MR McLean. Concurrent learning algorithm and the importance map. In *Network Science and Cybersecurity*, pages 239–250. Springer, 2014.

[52] Janardan Misra and Indranil Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1-3):239–255, 2010.

[53] Perry Moerland and Emile Fiesler. Neural network adaptations to hardware implementations. Technical report, IDIAP, 1997.

[54] Cleve B Moler. *Numerical computing with MATLAB*. SIAM, 2004.

[55] A.J. Montalvo, R.S. Gyurcsik, and J.J. Paulos. Toward a general-purpose analog vlsi neural network with on-chip learning. *IEEE Transactions on Neural Networks*, 8(2):413–423, 1997.

[56] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[57] Bharathwaj Muthuswamy and Leon O Chua. Simplest chaotic circuit. *International Journal of Bifurcation and Chaos*, 20(05):1567–1580, 2010.

[58] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[59] James R Norris and James Robert Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

[60] Marco Ottavi. Evaluating the yield of repairable srams for ate: Ieee transactions on instrumentation and measurement. *IEEE transactions on instrumentation and measurement*, 2006.

[61] Yuriy V Pershin and Massimiliano Di Ventra. Experimental demonstration of associative memory with memristive neural networks. *Neural networks*, 23(7):881–886, 2010.

[62] Matthew D Pickett, Dmitri B Strukov, Julien L Borghetti, J Joshua Yang, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. Switching dynamics in titanium dioxide memristive devices. *Journal of Applied Physics*, 106(7):074508, 2009.

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

[64] Themistoklis Prodromakis, Boon Pin Peh, Christos Papavassiliou, and Christofer Toumazou. A versatile memristor model with nonlinear dopant kinetics. *IEEE transactions on electron devices*, 58(9):3099–3105, 2011.

[65] Ulrich Ramacher and Christoph Von der Malsburg. *On the construction of artificial brains*. Springer Science & Business Media, 2010.

[66] Gareth O Roberts. Markov chain concepts related to sampling algorithms. *Markov chain Monte Carlo in practice*, 57:45–58, 1996.

[67] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[68] Garrett S. Rose, Robinson Pino, and Qing Wu. A low-power memristive neuromorphic circuit utilizing a global/local training mechanism. In *The 2011 International Joint Conference on Neural Networks*, pages 2080–2086, 2011.

[69] Stuart Russell and Peter Norvig. Ai a modern approach. *Learning*, 2(3):4, 2005.

[70] Maheshwar Pd. Sah, Changju Yang, Hyongsuk Kim, and Leon O Chua. Memristor circuit for artificial synaptic weighting of pulse inputs. In *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1604–1607, 2012.

[71] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[72] Reza Shayanfar, Abdollah Khoei, and Khayrollah Hadidi. Multi-input voltage-mode min-max circuit. *2010 18th Iranian Conference on Electrical Engineering*, pages 413–416, 2010.

[73] P. M. Sheridan, C. Du, and W. D. Lu. Feature extraction using memristor networks. *IEEE Transactions on Neural Networks and Learning Systems*, 27(11):2327–2336, Nov 2016.

[74] T. Shima, T. Kimura, Y. Kamatani, T. Itakura, Y. Fujita, and T. Iida. Neuro chips with on-chip back-propagation and/or hebbian learning. *IEEE Journal of Solid-State Circuits*, 27(12):1868–1876, 1992.

[75] G Snider. Computing with hysteretic resistor crossbars. *Applied Physics A*, 80(6):1165–1172, 2005.

[76] Greg S Snider. Cortical computing with memristive nanodevices. *SciDAC Review*, 10:58–65, 2008.

[77] Greg S Snider. Spike-timing-dependent learning in memristive nanodevices. In *IEEE International Symposium on Nanoscale Architectures*, pages 85–92, 2008.

[78] Daniel Soudry, Dotan Di Castro, Asaf Gal, Avinoam Kolodny, and Shahar Kvatinsky. Memristor-based multilayer neural networks with online gradient descent training. *IEEE Transactions on Neural Networks and Learning Systems*, 26(10):2408–2421, 2015.

[79] Charles H Stapper, Frederick M Armstrong, and Kiyotaka Saji. Integrated circuit yield statistics. *Proceedings of the IEEE*, 71(4):453–470, 1983.

[80] Charles H Stapper, Andrew N McLaren, and Martin Dreckmann. Yield model for productivity optimization of vlsi memory chips with redundancy and partially good product. *IBM Journal of Research and Development*, 24(3):398–409, 1980.

[81] Dmitri B Strukov and Konstantin K Likharev. Cmol fpga: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, 16(6):888, 2005.

[82] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[83] Dmitri B Strukov, Duncan R Stewart, Julien Borghetti, Xuema Li, M Pickett, G Medeiros Ribeiro, Warren Robinett, G Snider, John Paul Strachan, Wei Wu, et al. Hybrid cmos/memristor circuits. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1967–1970. IEEE, 2010.

[84] Philip C Treleaven. Neurocomputers. *Neurocomputing*, 1(1):4–31, 1989.

[85] Son Ngoc Truong and Kyeong-Sik Min. New memristor-based crossbar array architecture with 50-% area reduction and 48-% power saving for matrix-vector multiplication of analog neuromorphic computing. *JSTS: Journal of Semiconductor Technology and Science*, 14(3):356–363, 2014.

[86] Onur Tunali and Mustafa Altun. A survey of fault-tolerance algorithms for reconfigurable nano-crossbar arrays. *ACM Computing Surveys (CSUR)*, 50(6):1–35, 2017.

[87] Özgür Türel, Jung Hoon Lee, Xiaolong Ma, and Konstantin K Likharev. Neuromorphic architectures for nanoelectronic circuits. *International Journal of Circuit Theory and Applications*, 32(5):277–302, 2004.

[88] Hui Wang, Hai Li, and Robinson E. Pino. Memristor-based synapse design and training scheme for neuromorphic computing architecture. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–5, 2012.

[89] Xiaopeng Wang, Marco Ottavi, and Fabrizio Lombardi. Yield analysis of compiler-based arrays of embedded srams. In *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 3–10. IEEE, 2003.

[90] D. R. Wilson and T. R. Martinez. The need for small learning rates on large problems. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, volume 1, pages 115–119 vol.1, 2001.

[91] Lixue Xia, Wenqin Huangfu, Tianqi Tang, Xiling Yin, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, and Huazhong Yang. Stuck-at fault tolerance in rram computing systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):102–115, 2017.

[92] Lixue Xia, Mengyun Liu, Xuefei Ning, Krishnendu Chakrabarty, and Yu Wang. Fault-tolerant training with on-line fault detection for rram-based neural computing systems. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[93] Qiangfei Xia, Warren Robinett, Michael W Cumbie, Neel Banerjee, Thomas J Cardinali, J Joshua Yang, Wei Wu, Xuema Li, William M Tong, Dmitri B Strukov, et al. Memristor- cmos hybrid integrated circuits for reconfigurable logic. *Nano letters*, 9(10):3640–3645, 2009.

[94] C. Yakopcic. Method for ex-situ training in memristor-based neuromorphic circuit using robust weight programming method. *Electronics Letters*, 51:899–900(1), June 2015.

[95] C. Yakopcic, R. Hasan, and T. M. Taha. Memristor based neuromorphic circuit for ex-situ training of multi-layer neural network algorithms. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, July 2015.

[96] C Yakopcic, R Hasan, TM Taha, M McLean, and D Palmer. Memristor-based neuron circuit and method for applying learning algorithm in spice. *Electronics Letters*, 50(7):492–494, 2014.

[97] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino. Memristor spice model and crossbar simulation based on devices with nanosecond switching time. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, Aug 2013.

[98] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers. A memristor device model. *IEEE Electron Device Letters*, 32(10):1436–1438, Oct 2011.

[99] Chris Yakopcic, Raqibul Hasan, and Tarek M Taha. Tolerance to defective memristors in a neuromorphic learning circuit. In *NAECON 2014-IEEE National Aerospace and Electronics Conference*, pages 243–249. IEEE, 2014.

[100] Chris Yakopcic and Tarek M Taha. Energy efficient perceptron pattern recognition using segmented memristor crossbar arrays. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2013.

[101] Chris Yakopcic, Tarek M. Taha, Guru Subramanyam, Robinson E. Pino, and Stanley Rogers. Analysis of a memristor based 1t1m crossbar architecture. In *The 2011 International Joint Conference on Neural Networks*, pages 3243–3247, 2011.

[102] X. Yang, A. A. Adeyemo, A. Jabir, and J. Mathew. High-performance single-cycle memristive multifunction logic architecture. *Electronics Letters*, 52(11):906–907, 2016.

[103] Xiaohan Yang, Adedotun Adeyemo, Anu Bala, and Abusaleh Jabir. Novel techniques for memristive multifunction logic design. *Integration, the VLSI Journal*, 2017.

[104] Shimeng Yu, Yi Wu, and H-S Philip Wong. Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory. *Applied Physics Letters*, 98(10):103514, 2011.

[105] Carlos Zamarreño-Ramos, Luis A Camuñas-Mesa, Jose A Pérez-Carrasco, Timothée Masquelier, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in neuroscience*, 5:26, 2011.

[106] Karel Zaplatilek. Memristor modeling in MATLAB & SIMULINK. In *Proceedings of the European computing conference*, pages 62–67, 2011.

[107] Baogang Zhang, Necati Uysal, Deliang Fan, and Rickard Ewetz. Handling stuck-at-faults in memristor crossbar arrays using matrix transformations. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 438–443, 2019.

[108] Fan Zhang and Miao Hu. Defects mitigation in resistive crossbars for analog vector matrix multiplication. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 187–192. IEEE, 2020.

[109] WS Zhao, Guillaume Agnus, Vincent Derycke, A Filoramo, JP Bourgoin, and C Gamrat. Nanotube devices based crossbar architecture: toward neuromorphic computing. *Nanotechnology*, 21(17):175202, 2010.