# Designing Neuromorphic Computing Systems with Memristor Devices

by

**Amr Mahmoud Hassan Mahmoud**

B.Sc. in Electrical Engineering, Cairo University, 2011

M.Sc. in Engineering Physics, Cairo University, 2015

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

**Doctor of Philosophy**

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Amr Mahmoud Hassan Mahmoud

It was defended on

May 29$^{th}$, 2019

and approved by

Yiran Chen, Ph.D., Associate Professor
Department of Electrical and Computer Engineering

Amro El-Jaroudi, Ph.D., Associate Professor
Department of Electrical and Computer Engineering

Samuel J. Dickerson, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Natasa Miskov-Zivanov, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Bo Zeng, Ph.D., Assistant Professor
Department of Industrial Engineering

Dissertation Director: Yiran Chen, Ph.D., Associate Professor

**Designing Neuromorphic Computing Systems with Memristor Devices**

Amr Mahmoud Hassan Mahmoud, PhD

University of Pittsburgh, 2019

Deep Neural Networks (DNNs) have demonstrated fascinating performance in many real-world applications and achieved near human-level accuracy in computer vision, natural video prediction, and many different applications. However, DNNs consume a lot of processing power, especially if realized on General Purpose GPUs or CPUs, which make them unsuitable for low-power applications.

On the other hand, neuromorphic computing systems are heavily investigated as a potential substitute for traditional von Neumann systems in high-speed low-power applications. One way to implement neuromorphic systems is to use memristor crossbar arrays because of their small size, low power consumption, synaptic like behavior, and scalability. However, these systems are in their early developing stages and still have many challenges to be solved before commercialization.

In this dissertation, we will investigate designing of neuromorphic computing systems, targeting classification and generation applications. Specifically, we introduce three novel neuromorphic computing systems. The first system implements a multi-layer feed-forward neural network, where memristor crossbar arrays are utilized in realizing a novel hybrid spiking-based multi-layered self-learning system. This system is capable of on-chip training, whereas for most previously published systems training is done off-chip. The system performance is evaluated using three different datasets showing improved average failure error by 42% than previously published systems and great immunity against process variations.

The second system implements an Echo State Network (ESN), as a special type of recurrent neural networks, by utilizing a novel memristor double crossbar architecture. The system has been trained for sample generation, using the Mackey-Glass dataset, and simulations show accurate sample generation within a 75% window size of the training dataset. Finally, we introduce a novel neuromorphic computing for real-time cardiac arrhythmia classification. Raw ECG data is directly fed to the system, without any feature extraction, and hence reducing classification time and power consumption. The proposed system achieves an overall accuracy of 96.17% and requires only 34 *ms* to test one ECG beat, which outperforms most of its counterparts.

For future work, we introduce a preliminary neuromorphic system implementing a deep Generative Adversarial Network (GAN), based on ESNs. The system is called ESN-GAN and it targets natural video generation applications.

## Table of Contents


**Acknowledgment** . . . . . xv

**1.0 Introduction** . . . . . 1

1.1 Dissertation Goal and Organization . . . . . 6

**2.0 Memristor Devices in Crossbar Arrays Configuration** . . . . . 8

2.1 What is a Memristor? . . . . . 8

2.2 Memristors in Crossbar Arrays . . . . . 12

2.2.1 Different Cell Structure to Overcome Sneak Paths . . . . . 12

2.2.2 Cell Structure Adopted in This Dissertation . . . . . 18

**3.0 Hybrid Spiking-based Multi-Layered Self-Learning Neuromorphic System Based on Memristor Crossbar Arrays** . . . . . 20

3.1 Introduction . . . . . 20

3.2 Preliminary . . . . . 22

3.2.1 Memristor Devices and Model Used . . . . . 22

3.2.2 Neural Network Crossbar Arrays . . . . . 23

3.2.3 Integrate-and-Fire Circuit (IFC) . . . . . 24

3.3 Design Methodology and Hardware Implementation . . . . . 25

3.3.1 Proposed Training Scheme . . . . . 25

3.3.2 Hardware Implementation of Two Layers Crossbar Array . . . . . 27

3.4 System Evaluation . . . . . 33

3.4.1 Ideal System Performance . . . . . 34

3.4.2 Memristor Process Variations . . . . . 36
3.5 Conclusions . . . . . 37
**4.0 Hardware Implementation of Echo State Networks using Memristor Double Crossbar Arrays** . . . . . 40
4.1 Introduction . . . . . 40
4.2 Preliminary . . . . . 43
4.2.1 Memristor Devices and Model Used . . . . . 43
4.2.2 Echo State Networks and Reservoir Computing . . . . . 44
4.3 Proposed Architecture and Design Procedure . . . . . 46
4.3.1 Generic ESN Hardware Implementation . . . . . 47
4.3.2 Design Procedure . . . . . 50
4.4 System Evaluation . . . . . 52
4.4.1 Ideal System Performance . . . . . 53
4.4.2 Memristor Process Variations . . . . . 54
4.5 Conclusions . . . . . 58
**5.0 Real-time Cardiac Arrhythmia Classification using Memristor Neuromorphic Computing System** . . . . . 59
5.1 Introduction . . . . . 59
5.2 Preliminary . . . . . 61
5.2.1 Memristor Devices and Model Used . . . . . 61
5.2.2 Feed-Forward Neural Network Crossbar Arrays . . . . . 62
5.3 Proposed Architecture and Design Procedure . . . . . 63
5.3.1 Two-Layers Memristor Crossbar Arrays . . . . . 63

5.3.2 Training Procedure . . . . . 66
5.4 System Evaluation . . . . . 67
5.4.1 Memristor Process Variations . . . . . 69
5.5 Conclusions . . . . . 70
**6.0 Conclusion and Future Work** . . . . . 71
6.1 Video Prediction using GAN . . . . . 72
6.1.1 Training *Disc* . . . . . 75
6.1.2 Training *Gen* . . . . . 75
6.2 Video Prediction using ESN-GAN . . . . . 77
6.3 Experiments and Results . . . . . 78
6.3.1 Training Details . . . . . 78
6.3.2 Quantitative Metrics . . . . . 79
6.3.3 Qualitative Evaluation . . . . . 81
6.4 Future Work . . . . . 81
**Bibliography** . . . . . 84

## List of Tables


1 Error vector mapping for different values of $\delta_j^{L2}$ . . . . . 32

2 Properties of the used datasets and their corresponding crossbar systems. . . 33

3 Different parameters for the dataset and system under evaluation. . . . . 53

4 Records used to test the proposed system. . . . . 67

5 Proposed method in comparison with other studies in literature. . . . . 68

6 Different parameters used in training both systems. . . . . 78

7 Comparison between GAN [1] and ESN-GAN architecture. . . . . 79

## List of Figures


1 NAND flash memory vs. Cross-point ReRAM. Source: Crossbar Inc 1. . . . . 2

2 Memory Wall phenomenon shows the performance gap between processors and DRAM over years. . . . . 3

3 Power consumption vs clock frequency of recent devices. From [2]. Reprinted with permission from AAAS. . . . . 5

4 Quad-copter powered by a brain-like neuromorphic chip. From [3]. . . . . 6

5 The four fundamental circuit quantities, namely current $I$, charge $Q$, voltage $V$, and magnetic flux $\Phi$, along with the six relationships that connect them. The blue relation shows the missing circuit element, the memristor. . . . . 9

6 Memristor structure suggested by HP. . . . . 11

7 Crossbar memristors array. The gray path ((a) and (b)) shows the desired signal track in order to read the value stored in the gray memristor. The red paths (b) show the undesired tracks that the signal might flow in. . . . . 13

8 Memristor crossbar array with 1T1M cells. . . . . 15

9 Memristor crossbar array with 1S1M cells [4]. . . . . 15

10 I-V characteristics of the FAST selector device with different thresholds [4]. © 2016 IEEE. . . . . 16

11 The digital images dataset used in evaluating the system proposed in [4]. (a) shows the standard dataset representing the digits from "0" to "5", while (b) shows the images after introducing some noise. . . . . 17

12 The pattern recognition failure rate of the neuromorphic system using 1S1M structure [4]. © 2016 IEEE. . . . . 17

13 The $I$-$V$ and $G$-$V$ curves of (a) an ideal cell; (b) a 1T1M cell; and (c) 1S1M cell [4]. © 2016 IEEE. . . . . 19

14 Neural network implementation using memristor crossbar arrays. (a) shows a conventional diagram for 4×2 neural network while (b) shows the crossbar implementation of that network. . . . . 25

15 Integrate-and-Fire Circuit. (a) shows the circuit diagram of the IFC as proposed in [5], while (b) shows our simulation for that circuit for the number of spikes (y-axis) against current values (x-axis) for different $V_{\text{ref}}$. . . . . 26

16 Hardware implementation of a two layer memristor crossbar array. A 4×3×2 neural network diagram is shown in the lower left corner, and its circuit implementation is shown in the rest of the figure. . . . . 29

17 Training the second crossbar in Fig. 16. (a) shows the legend for different colors, while (b) depicts the assumed value for the error vector $\delta$ and the input vector $\eta \times x$. (c) and (d) show the only two required steps to program the whole crossbar. . . . . 31

18 Ideal system evaluation for the architecture shown in Fig. 16. (a) shows a comparison between average failure error for conventional two-layers neural network (blue), two-layers spiking-based system (green), and one-layer spiking-based system (yellow). Results are shown for three datasets, namely Digits [5], Cancer1, and Thyroid1 [6], and (b) shows the average failure error for Digits (red), Cancer1 (blue), and Thyroid1 (black) versus $IFC_{max}$ of the intermediate stages IFC. . . . . 35

19 System evaluation, after taking into account process variations, for the architecture shown in Fig. 16. (a) shows a comparison between average failure error for the two-layers spiking-based system, using online (solid) and offline (dashed) training, versus standard deviation $\sigma$ for Thyroid1 (blue), Cancer1 (red), and Binary Digits (black). (b) shows the average number of epochs for online trained systems versus standard deviation $\sigma$ for the same datasets. . . 38

20 Neural network implementation using memristor crossbar arrays. (a) shows a conventional diagram for 4×2 neural network while (b) shows the crossbar implementation of that network. . . . . 41

21 Generic Echo State Network (ESN) with an input layer, reservoir layer, and output layer. . . . . 44

22 Hardware implementation of ESN. (a) shows a 2x4x1 ESN with random connections inside the reservoir and (b) depicts the hardware implementation of this network model. both figures are color matched. . . . . 48

23 Summation amplifier (orange block) in Fig. 22(b) (a) shows the circuit implementation of the amplifier and (b) depicts the amplifier IV characteristics (blue curve) versus *tanh(x)* characteristics (red curve). . . . . 49

24 Design procedure steps to implement memristor crossbar arrays for ESNs. . . 51

25 Performance comparison between the proposed hardware implementation (red) and the software implementation (blue) of ESN for Mackey-Glass dataset. . . 55

26 Comparison between the actual testing dataset (green) versus the ESN output, *y(t)*, for (a) software implementation (blue) and (b) hardware implementation (red). The results shown are for M=1000. . . . . 56

27 System evaluation, after taking into account process variations, for the proposed hardware implementation. It depicts the MSE versus the reservoir size M, for different values of $\sigma$. . . . . 57

28 Neural network implementation using memristor crossbar arrays. (a) shows a conventional diagram for 4×2 feed-forward neural network while (b) shows the memristor crossbar array implementation of that network. . . . . 63

29 Hardware implementation of the proposed two-layers memristor crossbar array. A 300×210×5 feed-forward neural network diagram is shown in the lower left corner, and its circuit implementation is shown in the rest of the figure. . . . 64

30 Summation amplifier (yellow block) in Fig. 29 (a) shows the circuit implementation of the amplifier and (b) depicts the amplifier IV characteristics (blue curve) versus *tanh(x)* characteristics (red curve). . . . . 65

31 Misclassification error, after taking into account memristor process variations, for the proposed system architecture. . . . . 69

32 Illustration of the working mechanism of GAN. . . . . 73

33 Multiple scales architecture proposed in [1]. . . . . 74

34 ESN-GAN architecture. . . . . 77

35 Comparison between the training loss functions of (a) *Disc* and (b) *Gen* networks for both ESN-GAN (blue) and CNN-GAN (red). . . . . 80

36 Comparison between (a) PSNR and (b) Sharp Difference for both ESN-GAN (blue) and CNN-GAN (red). . . . . 82

37 Sample batch showing the sequence of frames used in training, the next generated and real frame. This batch was recorded at the last training epoch. . . . . 83

## Acknowledgment

Over the past few years, I have enjoyed pursuing a doctor of philosophy degree in Electrical Engineering at University of Pittsburgh, and finally the journey has come to an end. I am greatly thankful to Allah and many other people for helping me out through this journey and becoming the human I am right now.

First and foremost, I would like to express my indebtedness and gratefulness to my adviser, Prof. Yiran Chen. I would not have reached this far without his continuous guidance and support. He was always pushing me forward and constantly provided me with new research ideas. His insight, enthusiasm and invaluable advises - not only in scientific research, but also in other aspects of life - have great impacts on me, personally and scientifically.

I also would like to extend my gratitude to Prof. Hai Li for teaching me invaluable skills in communicating ideas effectively and logically, critical thinking, documentation, and reporting.

I am also grateful to Professor Mahmoud El-Nokali and Professor Amro El-Jaroudi for their help and support through out my journey here. They always supported me with invaluable advice whenever I was frustrated and helpless. In addition, their amazing performance as instructors was a beacon to follow.

During my study at Pitt, I had the opportunity to learn from great professors. In particular, I would like to thank Prof. Zhi-Hong Mao, Prof. Jun Yang, Prof. Samuel J. Dickerson, Prof. Natasa Miskov-Zivanov, and Professor Bo Zeng for the help they provided me, either being their student, helping me to become a better TA, or reviewing my dissertation.

For the past few years here at Pitt, I had collaborated with many colleagues and made even more friends. Specifically, I would like to thank Mohamed Bayoumy, Rashad Eletreby, Aya Fawzy, Khaled Sayed, and Yassin Khalifa for all the fruitful discussions we had which resulted in top-notch research. Many thanks to Prof. Ahmed Dallal, Moataz elsisy, Moataz Abdulhafez, and all my friends for the good times we spent together and the unforgettable moments we shared. Also, I would like to extend my warmest thanks to Sandy Weisberg and Caitlin Mathis of the department of Electrical and Computer Engineering Administration for all the efforts and help they provided me to make my life a bit easier.

Last but not least, my deepest gratitude to my parents, my wife, my sister, and my brother for their unconditional love, support, and patience. They had to deal with many ups and downs during the different phases of my journey, but they never stopped believing in me. Without them, I would have never been able to successfully finish my degree.

It has been a great journey, full of joy, achievements, depression, learning, and many life experiences that I wouldn't regret doing it all over again.

*To my family and all my friends*

## 1.0 Introduction

Nowadays, nonvolatile memory represents a large variety of memory technologies which can store the information, even if not powered. Flash memory, which was firstly invented by Fujio Masuoka [7], is considered as one of the most versatile non volatile memory technologies used nowadays. They were commercialized by Intel in 1988 [8] and since then, Flash memory has been heavily utilized in numerous applications, ranging from massive data storage like Solid State Drives (SSDs), to consumer electronics such as digital cameras, mobile phones. However, as semiconductors technologies nodes continue shrinking down, the development of Flash memory becomes very challenging due to quantum physical limits and device reliability [9].

Researchers are now heavily working seeking alternatives for nonvolatile memories. New emerging memory devices, such as Resistive memory (ReRAM) [9, 10], Phase Change Memory (PCM) [11, 12], Ferroelectric RAM (FeRAM) [13, 14], and Spin-Transfer Torque RAM (STT-RAM) [15, 16] are considered potential candidates for next generation of nonvolatile memories. Among these technologies, ReRAM is considered a potential candidate for massive data storage because of its extremely high density, great scalability, low power consumption, and good compatibility with traditional CMOS technology [9,10,17]. Many of the well-known companies in the memory industry, such as Samsung and IBM, have adopted ReRAM technology. Fig. 1 released by [18] compared ReRAM technology to the present NAND flash memory, indicating that ReRAM technology advance in all the important aspects.

ReRAM devices are usually made of a metal-oxide layer sandwiched between two metal layers. Numerous oxide materials with fast resistive switching characteristics were studied, including $SiO_x$, $Ta_2O_5$, NiO, and $Al_2O_3$ [9]. However, no prototype was reported until 2002 when a 64-bit ReRAM array based on perovskite oxide devices was fabricated at 0.5μm CMOS process [19]. Following, Baek *et al.* [20] integrated a Transition Metal Oxide (TMO) ReRAM in 0.18μm CMOS technology. Shortly after in 2008, HP Labs [21] described the ReRAM devices with analogue resistive states as memristors, proving the existence of the fourth basic circuit element, predicted by professor Leon Chua in 1971 [22]. Since then, extensive efforts have been given to ReRAM development and applications. For instance, not only ReRAM technology can be used as high density memory, but also can be used to implement re-configurable systems [23] and matrix-based computation [24].

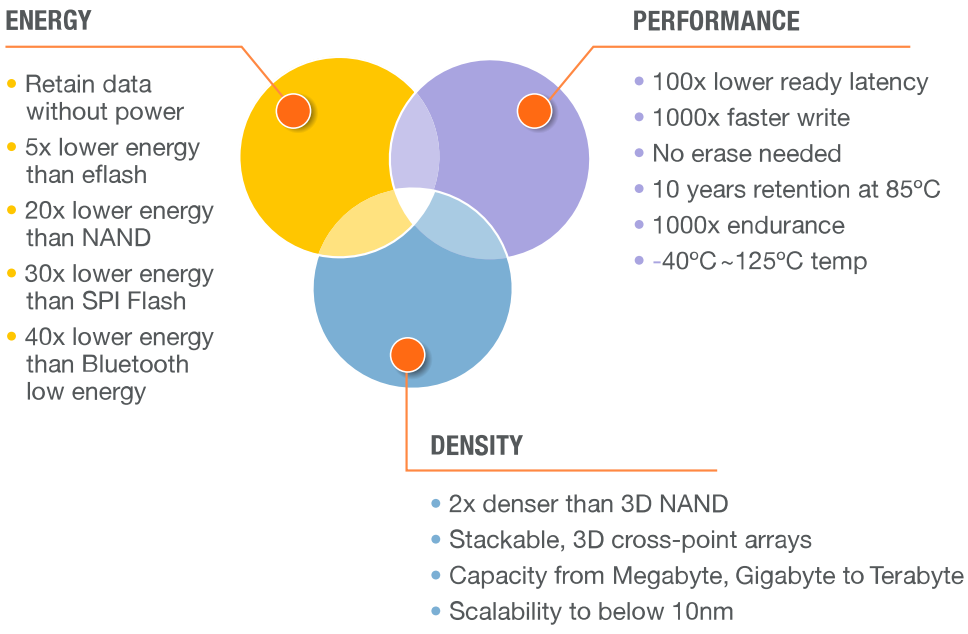



Figure 1: NAND flash memory vs. Cross-point ReRAM. Source: Crossbar Inc 1.

On the other hand, memristor technology is also used to develop neuromorphic computing systems, enabling a new computing solution with extremely high efficiency. Researchers have been heavily investigating neuromorphic systems as a potential substitute for the traditional von Neumann systems, as the latter cannot cope with the rapid growth of data amounts, processing power, and power efficiency. This problem is due to the well known "Memory Wall" phenomenon [25], which is shown in Fig. 2. The gap between the performance of processors and DRAM is increasing over the years, creating an urgent need for systems that don't separate data processing and storage in different locations. Neuromorphic computing systems can do this job as they mimic the working mechanism of human brains, where data are stored and processed at the same location [26], thus, offer great potentials.

In the 1980s, neuromorphic computing was firstly proposed, and that time it referred to implementing the computation in neural systems by utilizing a specific VLSI hardware

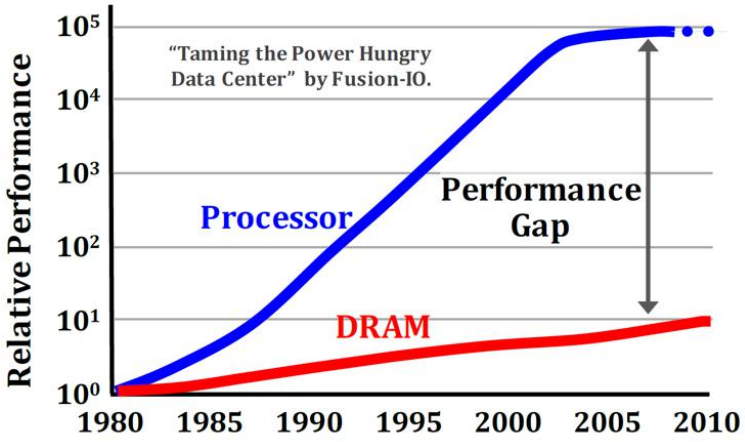



Figure 2: Memory Wall phenomenon shows the performance gap between processors and DRAM over years.

system [27]. Many mixed-signal CMOS systems have been developed aiming to achieve high computation speed with low cost [28, 29]. A group of MIT researchers [30] have proposed such a system, called the "brain chip". It's a 400-transistor chip that mimics the analog signal transmission in human brains [30]. In addition, neuromorphic computation can also be realized on Field-Programmable Gate Array (FPGA) or Field-Programmable Analog Array (FPAA) for low power signal processing and reconfigurability [31, 32]. Alternatively, IBM reported TrueNorth – a spike-timing based bio-synaptical chip, in which synapses were built with SRAM cells in a crossbar structure providing extremely low power and energy consumption in data transferring [2]. In addition, Fig. 3, taken from [2], depicts the power consumption versus the clock frequency of the human brain and current CPUs. As can be told, the human brain is at least 2.5 folds less than modern CPUs, yet, still very powerful in tasks like recognition and classification, and certainly outperforms any current neural network model. Such an extreme low power consumption is truly appealing, especially in embedded applications which have severe restrictions on power consumption. This is another reason why scientists move toward brain inspired computing, like the ones discussed in this dissertation.

As an application of embedded systems powered by neuromorphic chips, HRL's Center for Neural and Emergent Systems developed a quad-copter prototype, powered by a neuromorphic chip [3], as sown in Fig. 4. The chip was powered by 576 silicon neurons, took in data from the aircraft's optical, ultrasound, and infrared sensors as it learned to differentiate between three different rooms. The chip only weighed 18 grams and consumed 50 *mWatt*. It's worth mentioning that this project was funded by DARPA's neuromorphic SyNAPSE project, which tells how much the biggest funding entities are moving in this direction [3].

All the previous implementation of neuromorphic systems were based on conventional CMOS technologies. However, with the rising of the new emerging devices, such as spin-tronic devices and memristors, new neuromorphic architectures are widely investigated. For instance, a neuromorphic hardware using spin-tronic devices in crossbar structure was proposed by [33]. The design achieved more than 15× lower energy consumption, comparing to the state of art CMOS designs. Memristor cells emerge as one of the most attractive candidates because of their unique features, such as natural synaptic-like behavior, low energy, excellent scalability, and CMOS compatibility [34]. Many research entities have adopted memristors as an indispensable building block in their neuromorphic systems. For instance,

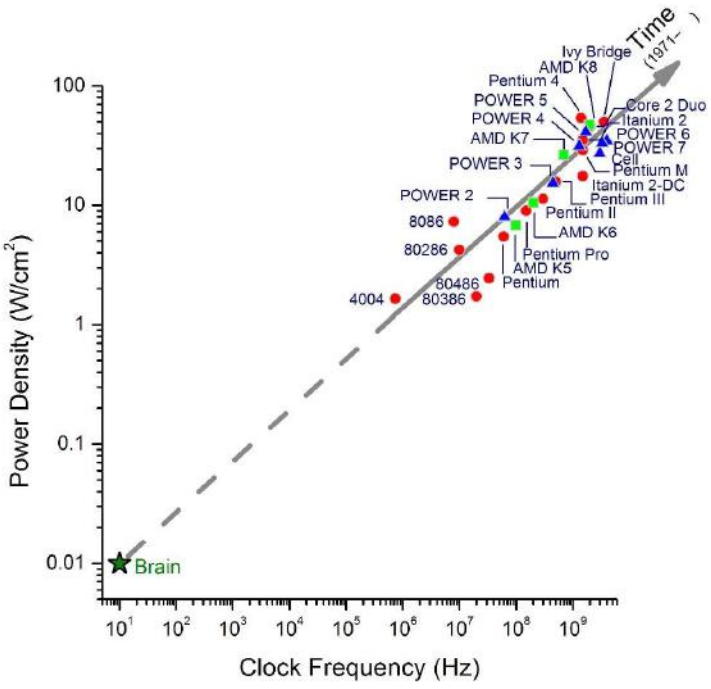



Figure 3: Power consumption vs clock frequency of recent devices. From [2]. Reprinted with permission from AAAS.

Hu *et al.* [24] proposed a neuromorphic engine with memristor crossbar in an analog computing approach, where the input and output signals are represented by analog signals with high parallel computation.

## 1.1 Dissertation Goal and Organization

Implementing neuromorphic computing systems using memristors is still in the early developing stages. Many obstacles and challenges need to be overcome before this technology becomes mature enough for mass production and commercialization. Our main goal in this dissertation is to investigate and tackle some of the resilient challenges that hinder the development of neuromorphic computing systems. Our main focus will be on neuromorphic computing systems for classification and generation purposes. Specifically, the rest of this dissertation is organized as follows:

Figure 4: Quad-copter powered by a brain-like neuromorphic chip. From [3].

- Chapter 2 will discuss, in details, the historical background of memristors, how they are configured in crossbar arrays, and which of these configurations will be adopted in the rest of this dissertation.
- Chapter 3 discusses our neuromorphic system implementation of multi-layer feed-forward neural networks. We propose a novel hybrid spiking-based multi-layered self-learning neuromorphic system and provide a detailed discussion on how this system helps alleviating most of the problems in current systems. The system performance is evaluated using three different datasets showing improved average failure error by 42% than previously published systems.
- Chapter 4 will illustrate our neuromorphic system implementation of Echo State Networks (ESNs), as a special type of Recurrent Neural Networks (RNNs). We provide a novel double memristor crossbar arrays architecture, as the main building block of our proposed system. In addition, we will shed some light on the current neuromorphic systems implementing ESNs, their issues, and how our system helps eliminating them.
- In Chapter 5, we exploit the small footprint of memristor crossbar arrays, along with low power consumption and fast response, to propose a neuromorphic system for embedded biomedical devices. The system implements real-time cardiac arrhythmia classification of five different beat types, using raw ECG data, with overall accuracy of 96.17% and 34 *ms* only to test one ECG beat, outperforming most of its counterparts.
- Finally, in Chapter 6, we discuss our projection of the future in developing neuromorphic systems, by introducing deep Generative Adversarial Network (GAN), their current progress, and our preliminary proposed ESN-based GAN. We will also present some of our preliminary results and some ideas to improve the whole system.

## 2.0 Memristor Devices in Crossbar Arrays Configuration

### 2.1 What is a Memristor?

Memristor, the fourth circuit element along with resistor, capacitor, and inductor, was theorized by Leon Chua in 1971 [35], and later generalized in 1976 [36]. According to the relations governing the three passive circuit elements at this time, Chua predicted, by analogy, that there has to be another passive circuit element. He named it memristor.

To elaborate, the four fundamental circuit quantities, namely the current $I$, electric charge $Q$, the voltage $V$, and the magnetic flux $\Phi$. These relationships are connected together, by the means of the derivative operator, as depicted in Fig. 5. The voltage $V$ is the time derivative of the magnetic flux $\Phi$, while The current $I$ is the time derivative of the charge $Q$. The three two-port passive circuit elements back then relate the four quantities as follows: resistors relate current to voltage via Ohm's Law ($dV = R.dI$), inductors relate magnetic flux to current ($d\Phi = L.dI$), and finally capacitors relate voltage to charge ($dQ = C.dV$).

Chua noticed that, from symmetry and for the sake of completeness, there has to be a fourth two-port passive element which relates the magnetic flux and charge ($d\Phi = M.dQ$). And from the expected behavior of such an element, Chua named it memristor. Moreover, Chua proved that the behavior of the memristors could not be mimicked by any combination of the three other passive circuit elements alone, and that it would require an active circuit of about 25 transistors to mimic its behavior [35].

It can be shown that [35]:

$$V(t) = M(Q(t)).I(t) \tag{2.1}$$

$$M(Q(t)) \equiv d\Phi(Q)/dQ \tag{2.2}$$

where *M(Q(t))* is called the memristance and it has the same unit as the resistance. To get the sense of the memristance represents, eq. 2.2 says that memristance, *M(Q(t))*, is function of the charge $Q$, or, in other words, the integration of the current $I$ flowing through the device with respect to time. Accordingly, memristor can be thought of as a two port passive device whose resistance is variable, depending on the current $I$ passing through.

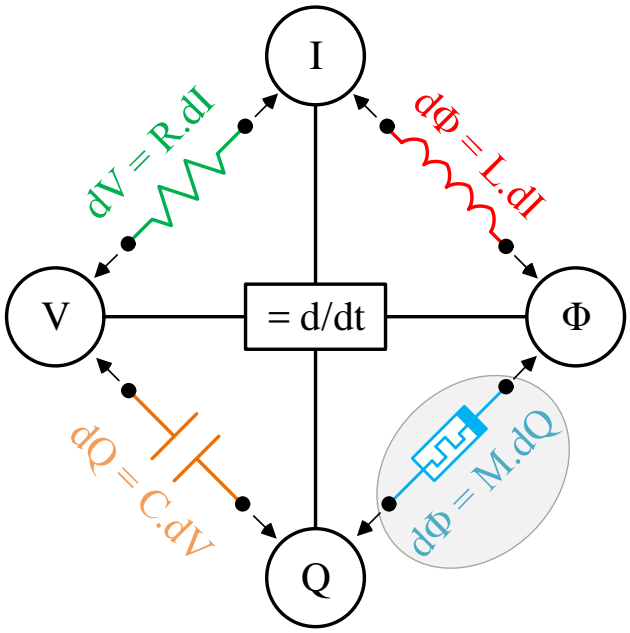



Figure 5: The four fundamental circuit quantities, namely current $I$, charge $Q$, voltage $V$, and magnetic flux $\Phi$, along with the six relationships that connect them. The blue relation shows the missing circuit element, the memristor.

It can also be deduced from the above relation that the memristance is constant when no current is applied (the time integral of zero current would result in a constant charge, thus constant memristance). Hence, the memristor keeps the last memristance value it reached, which can be thought of as a memory effect. And since it relates the voltage $V$ and the current $I$, as in eq. 2.1, the name memristor (short for memory resistor) seemed to be suitable.

It wasn't until 1976 when Chua *et al.* generalized the memristor behavior into memristive devices [36]. Memristive devices are two-port devices with varying memristance as well, however, they differ from memristors in the way their memristance changes. Chua *et al.* defined the memristance of memristive devices as a function of some internal state, $w \in R^n$, which, in turn, a function of the current passing through the device (eq. 2.3 and eq. 2.4). On the other hand, the memristance of memristor devices is a direct function of the charge $Q$. The equations representing these nonlinear dynamics of memristive devices are [36]:

$$V(t) = \mathcal{R}(w, I).I(t) \tag{2.3}$$

$$\frac{dw}{dt} = f(w, I) \tag{2.4}$$

where $\mathcal{R}(w, I)$ is the generalized resistance (without loss of generality, memristance and resistance can be interchangeable for memristive devices) of the device, and $w$, as mentioned above, is an internal state. *f(w,I)* is a function of $w$ and the current $I$.

Since 1976, memristive devices were just theory without any physcial evidence of their existence. However, this changed in 2008 when HP Labs made the first connection between the hypothesized memristor and a $TiO_2$-based device [21]. Fig. 6 depicts the anatomy of the device fabricated by HP, which is made of a doped $TiO_{2-x}$ layer and an undoped $TiO_2$ layer

sandwiched between two platinum electrodes. The doping here refers to the oxygen vacancies and $x$ represents their quantity. The width of the doped $TiO_{2-x}$ layer is considered as the internal state variable, $w$. Upon applying a voltage signal across the device, the oxygen vacancies drifts along with the electric field, causing a change in the doped $TiO_{2-x}$ width and changing the overall resistance of the device. On the opposite, removing the applied voltage signal won't revert back the oxygen vacancies to their original position as they have very low mobility [21].

The representation of eq. (2.3) and (2.4) are [21]:

$$V(t) = \left(R_{on}\frac{w(t)}{D} - R_{off}\left(1 - \frac{w(t)}{D}\right)\right).I(t) \tag{2.5}$$

$$\frac{dw}{dt} = \mu_v \frac{R_{on}}{D}.I(t) \tag{2.6}$$

where $R_{off}$ is the resistance when $w(t) = 0$, and $R_{on}$ is the resistance when $w(t) = D$. The internal state variable $w(t)$ has the limits of $0 < w(t) < D$. $\mu_v$ is the average mobility of the dopants.

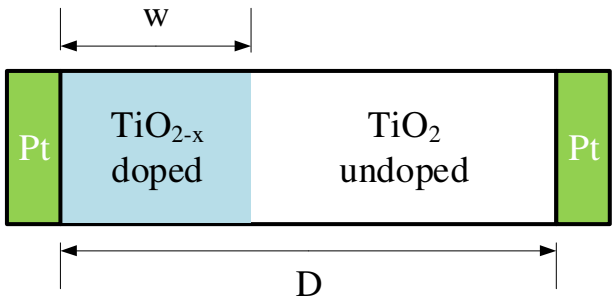



Figure 6: Memristor structure suggested by HP.

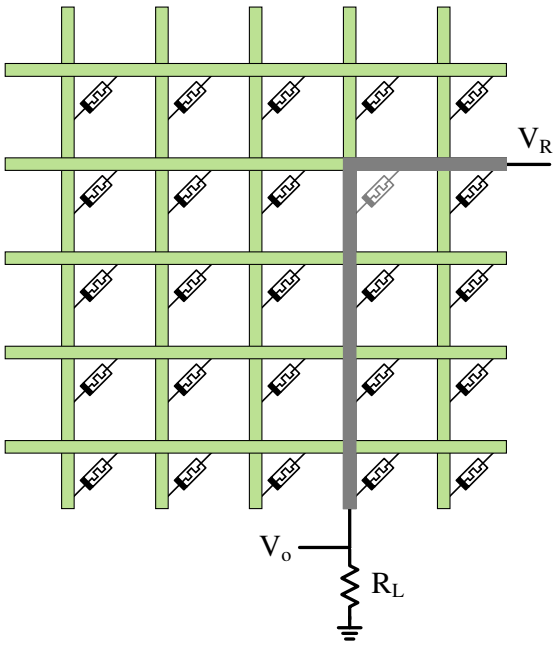
In 2009, Di Ventra and Chua expanded the definition of memristive systems to accommodate meminductors and memcapacitors. Like memristor, the inductance and capacitance of meminductors and memcapacitors depend on the history of the applied signal and the state of these devices [37, 38]. Since then, memristive devices acquired the attention of different scholars worldwide. In fact, many research entities and companies have adopted the memristor as a new emerging device and conducted countless experiments. It can be used in many applications, ranging from memory chips, analog circuits, and neuromorphic computing, which is the main focus of this dissertation.

## 2.2 Memristors in Crossbar Arrays

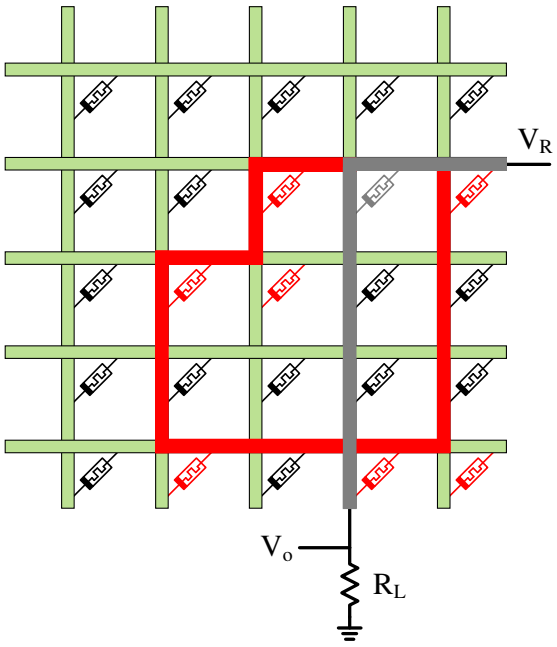
For the scope of this thesis, we are interested in studying memristor as the building block in neuromorphic computing systems. Those systems usually are built using crossbar arrays where memristors exist on each intersection, as shown in Fig. 7. An element is accessed by applying a read signal to the row and column that contains it. However, other paths may contribute in this reading phase, thus giving a false data or even ruining it. This problem is called Sneak Paths problem. Fig. 7(a) shows the desired path, and Fig. 7(b) shows the desired path along with the undesired sneak paths. This problem was addressed by many researchers [4, 5, 39, 40] and many solutions were developed in the last two years.

### 2.2.1 Different Cell Structure to Overcome Sneak Paths

Two of the solution proposed seemed to be very promising, in terms of scalability and speed. The first solution was proposed by many researchers [5, 41, 42], where they used




(a)




(b)

Figure 7: Crossbar memristors array. The gray path ((a) and (b)) shows the desired signal track in order to read the value stored in the gray memristor. The red paths (b) show the undesired tracks that the signal might flow in.

1-Transistor-1-Memristor (1T1M) cell in a crossbar array configuration. In this structure, transistors act a selector device to help alleviate the impact of sneak paths. The design of the 1T1M cell is shown in Fig. 8, where $cell_{ij}$ only contributes a current to $VL_j$ when the $HL_i$ is selected. The effective conductance of the $cell_{ij}$ in this case becomes $\hat{G}_{ij} = G_{ij} \,//\, G_{on}$ where $G_{on}$ is the conductance of the access transistor at ON state. When $HL_i$ is grounded, the transistor is off and its extremely small conductance $G_{off}$ causes $\hat{G}_{ij} \approx 0$.

Liu *et al.* [5] adopted the 1T1M cell and used it to build two neuromorphic systems: 1) a 32 × 32 crossbar implementing a feed-forward neural network and 2) 32 × 64 crossbar implementing a Hopfield network. Both systems were used for digital image recognition. Simulations for both systems showed excellent tolerance on the noisy images and process variations. Moreover, The feed-forward system (Hopfield) system achieved 50% reduction in power consumption and with only 1.45% (5.99%) increase in the average recognition failure rate, compared to previously published systems [5].

The second solution is to use a highly nonlinear selective resistor device instead of a transistor. Such a cell is called 1-Selector-1-Memristor (1S1M) and it was originally proposed in [43], and further investigated in [4]. Fig. 9 shows a crossbar array with 1S1M cells. The selector device conducts current only when the voltage applied on it exceeds a certain threshold (ON state); otherwise, the current going through is almost zero (OFF state). A typical selector should have the following features: high selectivity (i.e., high OFF/ON resistance ratio), fast set (OFF⟶ON) and relax (ON⟶OFF), and high endurance.

Among different selector devices investigated, the Field Assisted Superlinear Threshold (FAST) selector seems to be the device with the most excellent performance, compared to others [43]. This device is composed of a Superlinear Threshold Layer (STL) sandwiched

between two electrodes, where a conducting channel is formed upon applying the threshold electric field [43]. As shown in Fig. 10, the threshold electric field (hence, the threshold voltage) can be controlled by adjusting the thickness of the STL layer. It worth mentioning that the OFF and ON current in the figure is limited by the tester noise and current compliance, respectively. The device exhibits promising characteristics such as a current density larger

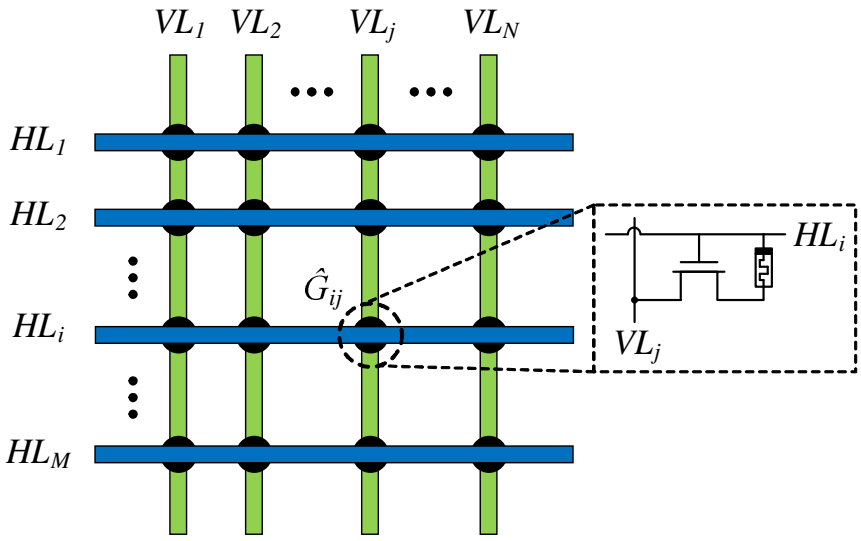



Figure 8: Memristor crossbar array with 1T1M cells.

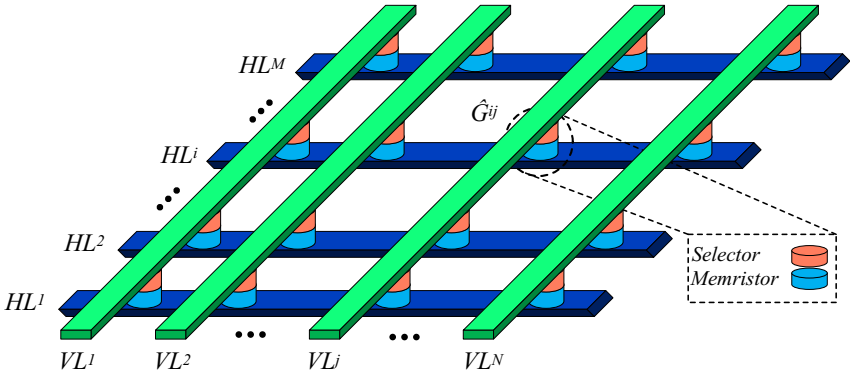



Figure 9: Memristor crossbar array with 1S1M cells [4].

than $5\times10^6$ A/cm$^2$ for device area as low as 15 nm×100 nm, high endurance over $1\times10^8$ cycles with set and relax time $\sim$ 50 ns [4].

Yan *et al.* [4] also implemented the same Hopfield-based neuromorphic computing system presented in [5], but using the 1S1M cells instead of 1T1M. For the comparison to be fair, they also used the same digital images, representing number from "0" to "5" in 8×4 binary images as shown in Fig. 11.

The recognition error rate of this system is shown in Fig.12. where the $x$-axis represents the probability of introducing error in a single bit and the $y$-axis is the recognition failure rate. According to this result, the use of 1S1M lowers the system's accuracy significantly,

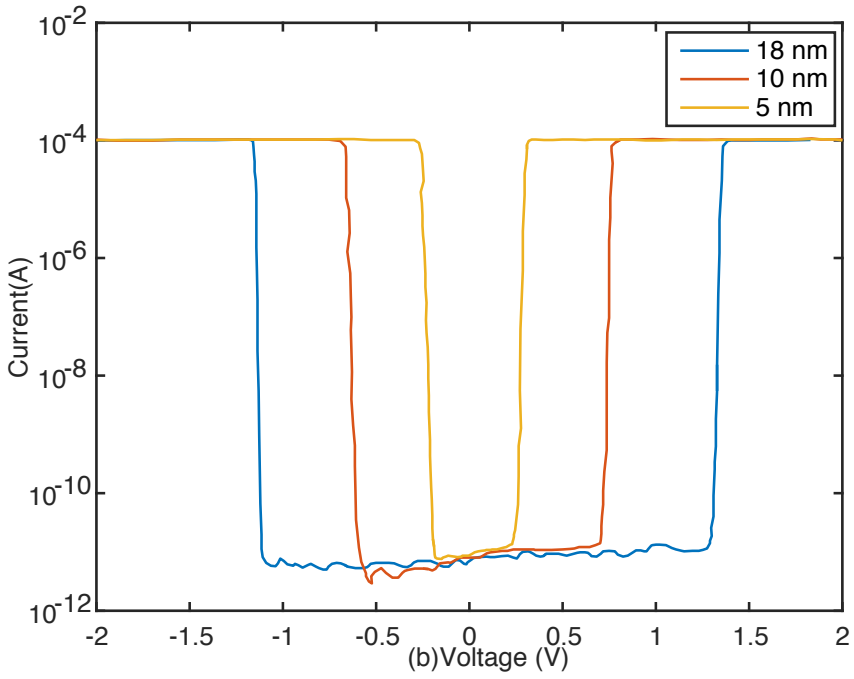
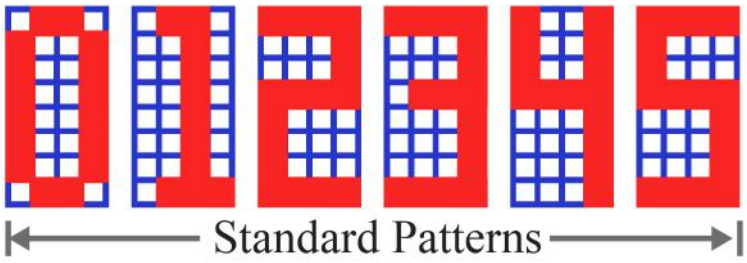
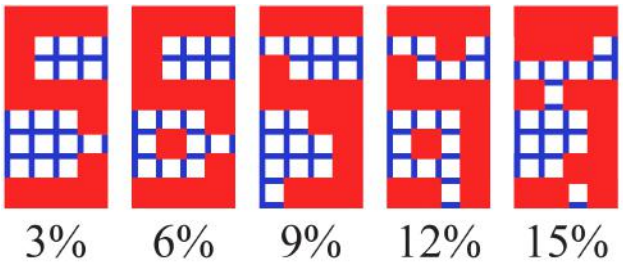



Figure 10: I-V characteristics of the FAST selector device with different thresholds [4]. © 2016 IEEE.




(a)




(b)

Figure 11: The digital images dataset used in evaluating the system proposed in [4]. (a) shows the standard dataset representing the digits from "0" to "5", while (b) shows the images after introducing some noise.

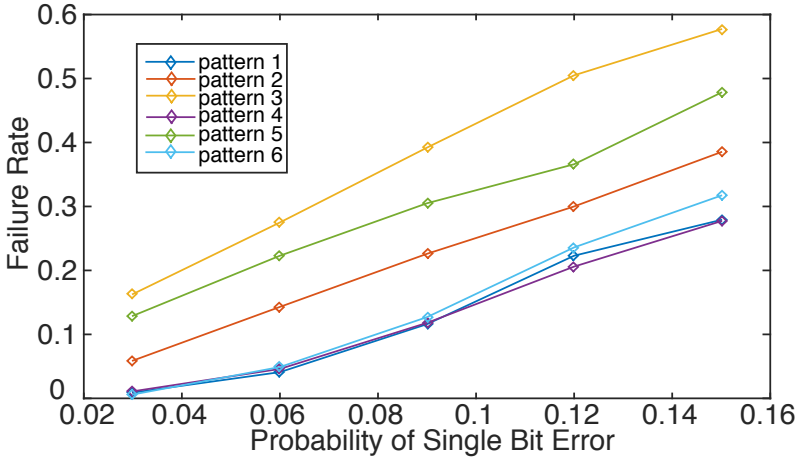



Figure 12: The pattern recognition failure rate of the neuromorphic system using 1S1M structure [4]. © 2016 IEEE.

compared to the same system using the 1T1M cell [5]. However, the 1S1M design offers almost 50% reduction in the area used, compared to the design in [5], offering a much better scalability.

This result was further analyzed by [4] and summarized in Fig. 13. Fig. 13(a) shows the $I$-$V$ and $G$-$V$ characteristics of the ideal and desired cell. While the applied voltage, say $V_i$, is larger than a threshold voltage $V_{th}$, the cell starts passing current with a constant conductance, regardless of the value of $V_i$. This behavior guarantees a stable vector-matrix multiplication. Fig. 13(b) shows the same characteristics for the 1T1M cell. $V_{th}$ here is the threshold voltage of the transistor, which is around 0.64V. As can be told, the behavior of the 1T1M is very similar to the behavior of the ideal cell in the region of $V_i > 1.1$ V (with only 8% variation in cell conductance) and $V_i < 0.3$ V. However, there is a big difference in both behaviors in the remaining region.

On the other hand, Fig. 13(b) depicts the same characteristics for the 1S1M cell. While this cell shows a great similarity to the ideal behavior in the OFF region and the switching region, a great discrepancy can be found in the ON region, with the conductance being almost linear and not constant. This discrepancy is the main reason why this 1S1M cell has poor performance compared to the 1T1M cell.

### 2.2.2 Cell Structure Adopted in This Dissertation

Since the main goal of this dissertation is to explore the potential and functionality of neuromorphic computing systems in implementing various neural networks, and as per our previous comparison, we will be adopting the 1T1M proposed in [5, 41].

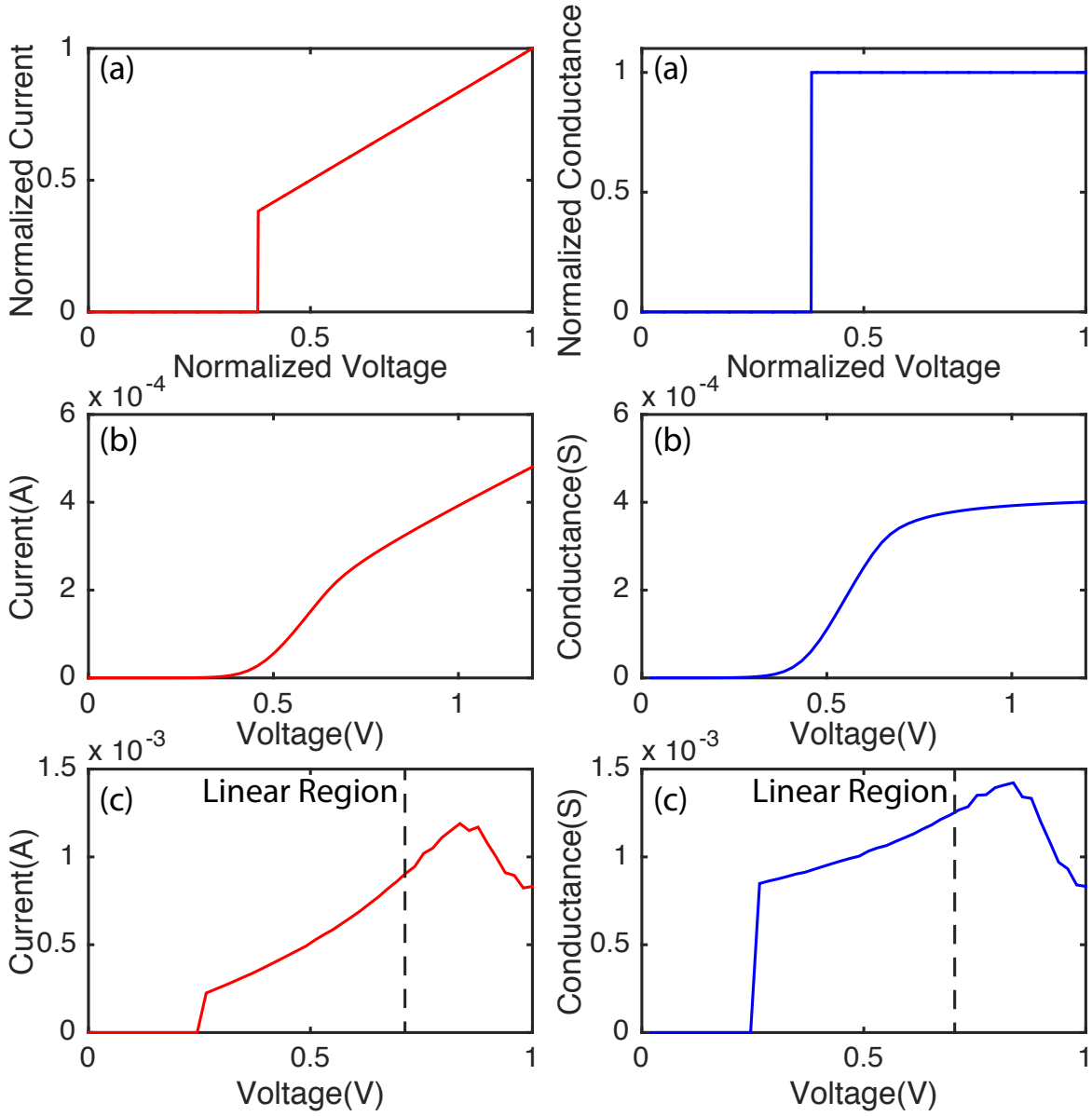



Figure 13: The $I$-$V$ and $G$-$V$ curves of (a) an ideal cell; (b) a 1T1M cell; and (c) 1S1M cell [4]. © 2016 IEEE.

# 3.0 Hybrid Spiking-based Multi-Layered Self-Learning Neuromorphic System Based on Memristor Crossbar Arrays

## 3.1 Introduction

With rapidly growing of online data, there is an need for new systems that can provide real-time data processing with high power efficiency. Traditional von Neumann systems are not able to cope up with such demands due to the well known "memory wall" phenomenon [25]. Neuromorphic computing systems, instead, offer great potentials by mimicking the working mechanism of human brains, where data are stored and processed at the same location [26]. Many research entities around the globe adopted those systems. For example, the IBM TrueNorth system [44] implements artificial synapses by using conventional CMOS circuitry [28, 29]. However, the high implementation cost could be a great challenge for further performance improvement and system scaling up.

Emerging memory devices [45], such as Resistive Random Access Memories (ReRAM), or Memristors, can offer a solution to this problem. Given their low feature size and synaptic like behavior [46, 47], memristors have been heavily investigated for possible usage in implementing large-scale neuromorphic systems. The two-terminal device is used to construct crossbar arrays, where memristor cells are located at each intersection of vertical and horizontal metal wires. Such a layout is of close resemblance to neural network models, in which the conductances of memristor cells correspond to the synaptic weights. In addition, the conductance value can be programmed to different discrete levels (or even continuous levels) by adjusting the amplitude (or pulse width) of the programming signal [48, 49].

Memristor crossbar arrays were used to implement matrix-vector multiplication in the neuromorphic computation process [50–52], in which voltage and currents values are used to represent the actual input and output data. These systems rely on Digital-to-Analog Converters (DACs) and Analog-to-Digital Converters (ADCs) as communication blocks. Consequently, further up scaling of these systems is hindered as the design complexity, cost, and area will rapidly increase. Alternatively, spiking-based neuromorphic systems were developed, where the inputs (outputs) take the form of voltage (current) spikes [5, 53, 54]. Such systems utilize simple CMOS circuitry to digitize the inputs and outputs to spikes, thus, offering adequate performance without the need for complex and expensive communication blocks.

The system in [5] is a single layer pattern recognition system based on what is so called off-chip (sometimes called offline or *ex-situ*) training. In this method, computer-based simulations are used to train the neural network and obtain the corresponding memristive weights. After that, those weights are programmed to the memristor cells in the crossbar array. Despite showing good compromise between performance and low hardware utilization, single-layered, off-chip trained systems cannot cope up with the overwhelming increase in the amount of data to be processed nowadays [51]. Moreover, the memristors resistance values drift from the programmed value during normal read operation, which requires regular weights updating for proper operation; a thing that off-chip training techniques cannot provide. That is why on-chip (sometimes called online or *in-situ*) training is widely adopted recently [51, 52], where extra hardware is added to the chip to allow online training (using training techniques which will be discussed later in Section 3.3.1) and constantly allow memristive weights updating.

To overcome the previous problems, we propose a hybrid spiking-based multi-layered self-learning neuromorphic system. The proposed system performance was evaluated from different aspects and following are the main contributions:

- Introducing a simple and effective method of implementing backpropagation in hardware, which reduces the time needed for circuit training to half compared to previously published work
- Offering design methodology and in-depth study for multi-layer spiking-based neuromorphic pattern recognition systems.
- Improving average failure error by 42% compared to previously published work for three different datasets.

To our best knowledge, such a hybrid system has never been proposed before.

The rest of the work will be organized as follows: Section 3.2 introduces the basic building blocks of the system and the implementation fundamentals. Section 3.3 explains the proposed multi-layer system, the design considerations, and the simple way of implementing backpropagation. Section 3.4 evaluates and discusses the use of our design in three different test cases. Finally, conclusion will be provided in Section 3.5.

## 3.2 Preliminary

### 3.2.1 Memristor Devices and Model Used

Memristors are nonlinear devices whose resistance values change when the voltage across the device exceeds a certain threshold [49]. Usually they are made of an oxide layer sand-

wiched between two conductive layers, where $TiO_{2-x}$ [55] and a layer composed of $HfO_x$ and $AlO_x$ [49] being the most promising oxides used. Since high energy efficient crossbar is needed, devices with higher resistances range become a must. The latter oxide shows a resistance ratio of ($R_{OFF}/R_{ON}$≈1000) and on-resistance ($R_{ON}$≈10K$\Omega$), which is adequate to be utilized in our system. For crossbar arrays, each cell is composed of a memristor and another device to enhance selectivity and eliminate *sneak path* problem [40, 56]. Among the different devices used as selectors [4, 5], the 1-Transistor 1-Resistor (1T1R) structure has been proven to be more reliable and has negligible effect on the overall conductance of the cell [5].

For this work, we are going to adopt the 1T1R structure. The memristor resistance ranges from [50K$\Omega$, 1M$\Omega$], which is divided into 8 discrete levels [5]. Those are the only allowable levels during training.

### 3.2.2 Neural Network Crossbar Arrays

Fig. 14 illustrates the resemblance between neural networks and crossbar arrays, where a typical one layer neural network can be implemented using memristor crossbar arrays and CMOS circuity. The synaptic weights, $w_{ij}$, which connect the inputs (blue circles) and output (gray circles) neurons in Fig. 14(a), are represented by the conductance, $G$, of the memristor cells at each cross point of the array in Fig. 14(b). The output neurons (and hidden neurons, if any) perform two functions (i) they evaluate the weighted sum of the inputs, $z_j = \sum_{i=1}^{N} x_i w_{ij}$, and (ii) they generate the output according to a nonlinear activation function, $y_j = f(z_j)$. The weighted summation function can be implemented directly using the crossbar array as follows: assuming that each vertical (bit) line is virtually connected to ground (which can

be done using the Integrate-and-Fire Circuit (IFC) [5]), the inputs, $x$, are encoded into a series of voltage pulses, $V$, and applied to the horizontal (word) lines. Those pulses will be multiplied by the conductance, $G$, resulting in current $I = GV$ injected in each bit line. The second function can be done using the IFC, which will be explained in Section 3.2.3.

Since the synaptic weights can be either positive or negative, two memristors will be used to represent one synapse, where $G_{ij} \equiv G_{ij}^{+}$- $G_{ij}^{-}$. In such a scheme, each output neuron is represented by two columns, representing $G^{+}$ and $G^{-}$, and currents from those columns are directed to IFC, where the corresponding pulses are generated. Those pulses are then used to increment (in case of $G^{+}$) or decrement (in case of $G^{-}$) the value of a counter which represent the final output of the neuron. In addition, this exponentially expands the number of allowable resistance levels to 57 levels instead of 8 only, which are all the possible combinations $8^2$ minus the number of repeated combinations of $G_{ij} = 0$ (7 combinations).

The major advantage of the spiking based systems is that it does not need expensive hardware, such as ADCs or summer amplifiers, unlike the other level based systems [51, 52].

### 3.2.3 Integrate-and-Fire Circuit (IFC)

Fig. 15(a) shows the schematic of the IFC as proposed in [5]. This circuit generates a number of voltage spikes at the output, $V_{out}$, which is proportional to the magnitude of the input current, $I_{in}$, that comes from the crossbar column $j$.

We characterize the IFC in the curves shown in Fig. 15(b). The curves show nonlinear dependency on the input current, $I_{in}$, with the ability of controlling the maximum number of spikes, $IFC_{max}$, by adjusting the reference voltage, $V_{ref}$ [5]. We will make use of these two properties to represent the activation function of hidden neurons and in designing the intermediate stages IFC, as will be further discussed in Section 3.3.2 and 3.4.1.

## 3.3 Design Methodology and Hardware Implementation

### 3.3.1 Proposed Training Scheme

Neural networks in general need to be trained to provide proper functionality. The most used training scheme for multi-layer neural networks is the back-propagation algorithm [57]. In the recent period, many variants of that algorithm have been proposed and used in training memristor crossbar arrays [52,58]. In this work, we will introduce, for the first time, a modified training algorithm for hybrid spiking-based multi-layered systems. This algorithm has reduced the time needed for training by half compared to previously published work, which shall be explained in Section 3.3.2.

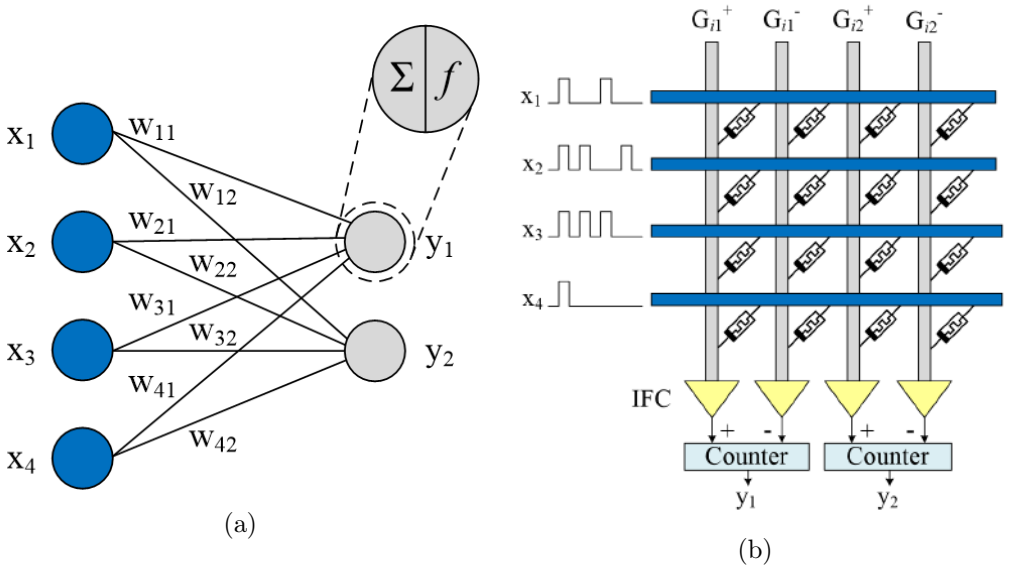



Figure 14: Neural network implementation using memristor crossbar arrays. (a) shows a conventional diagram for 4×2 neural network while (b) shows the crossbar implementation of that network.

In order to apply the back-propagation algorithm, input has to be fed forward through the whole network to calculate the weighted sums and activations for all the hidden and output neurons. Once that is done, weight update is calculated according to the following formula [57]:

$$\nabla w_{ij} \quad = \quad \eta \times \delta_j \times x_i, \tag{3.1}$$

where $\eta$ is the learning rate, $x_i$ is the $i^{\text{th}}$ output of the previous layer neuron (input or hidden), and $\delta_j$ is the $j^{\text{th}}$ error propagated form the next layer (hidden or output) and is expressed as [57]:

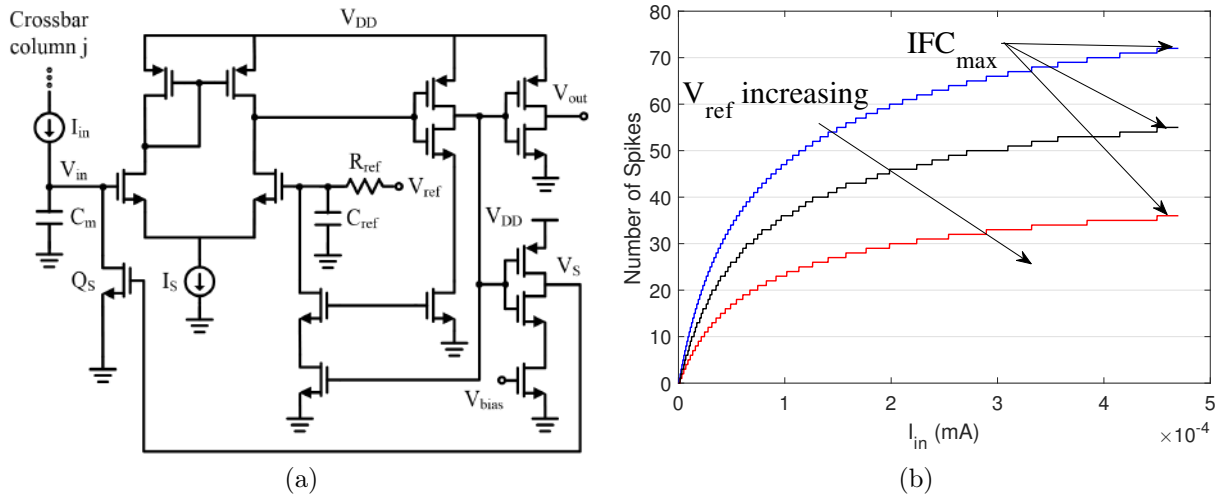



Figure 15: Integrate-and-Fire Circuit. (a) shows the circuit diagram of the IFC as proposed in [5], while (b) shows our simulation for that circuit for the number of spikes (y-axis) against current values (x-axis) for different $V_{\text{ref}}$.

$$\delta_j = \begin{cases} (t_j - y_j) & for\ the\ output\ layer \\ f'(z_j) \sum_{k=1}^{N} w_{kj}^n \delta_k^n & for\ any\ hidden\ layer, \end{cases} \tag{3.2}$$

where $t_j$ ($y_j$) is the $j^{\text{th}}$ desired (actual) output at the output layer, $f'$ is the derivative of the activation function of the hidden layer, $z_j$ is the $j^{\text{th}}$ weighted sum, and the superscript $n$ refers to the next layer. So, it can be seen that weights updates are obtained backwards starting from the output layer. Applying the exact version of back-propagation algorithm requires very expensive hardware, such as ADCs, DACs, lookup tables (to calculate the activation function derivative) , buffers, and multipliers.

In this work, we propose a simpler, yet accurate enough, version of back-propagation algorithm. For spiking systems, the inputs to any layer are always positive and in form of spikes. By using this property, we can, approximately, separate Eq. (3.1) to magnitude and sign, where the sign is determined by $sgn(\delta_j)$ and the magnitude by $\eta \times x_i$. In that way, detecting the sign information of $\delta_j$ doesn't require complex hardware, which significantly simplifies the design of the training circuits, as will be seen later. Algorithm 3.1 depicts the exact steps of the proposed training scheme.

### 3.3.2 Hardware Implementation of Two Layers Crossbar Array

In this part, we will discuss the actual implementation of the proposed algorithm in hardware. Shown in Fig. 16 is an example of 4×3×2 neural network (lower left corner) and its memristor crossbar array implementation. All the corresponding elements between the neural network diagram and the hardware implementation are color matched.

**Algorithm 3.1** Proposed Training Algorithm

1: Randomly initialize memristor weights

2: **while** $(E > E_{desired})$ **do**

3: **for** (each x in the training set) **do**

4: Feed the input x forward through the whole network

5: Calculate $z_j$ and $y_j$ for all hidden and output neurons

6: Calculate the error for each output layer neuron $j$ using:

$$\delta_j = sgn\left(t_j - y_j\right)$$

7: Back propagate the above error for hidden layers neurons

8: Calculate the error for each hidden layer neuron $j$ by using:

$$\delta_j = sgn\left(f'\left(z_j\right) \sum_{k=1}^{N} w_{kj}^{n} \delta_k^n\right)$$

9: Apply programming pulses to memristors, where the number of pulses for the $i^{th}$ and $j^{th}$ memristor in each crossbar are determined by $\eta \times x_i$, and the polarity by $\delta_j$

10: **end for**

11: **end while**

*1) The feed forward step for error vector calculation.*

During the feed forward step of the proposed algorithm, all the switches with *ctrl 1* (*ctrl* 2) are closed (opened). A randomly chosen input $x$ from the training dataset is then converted into pulses and applied to the word lines of the first crossbar layer. Positive and negative currents are then converted by IFC into spikes and then applied to the up and down control signals of the counter, respectively. We are making use of the nonlinearity in the IFC characteristics and counter, whose lower limit value is zero, to represent the nonlinear activation function of the hidden neuron. The value stored in the counters are then converted into

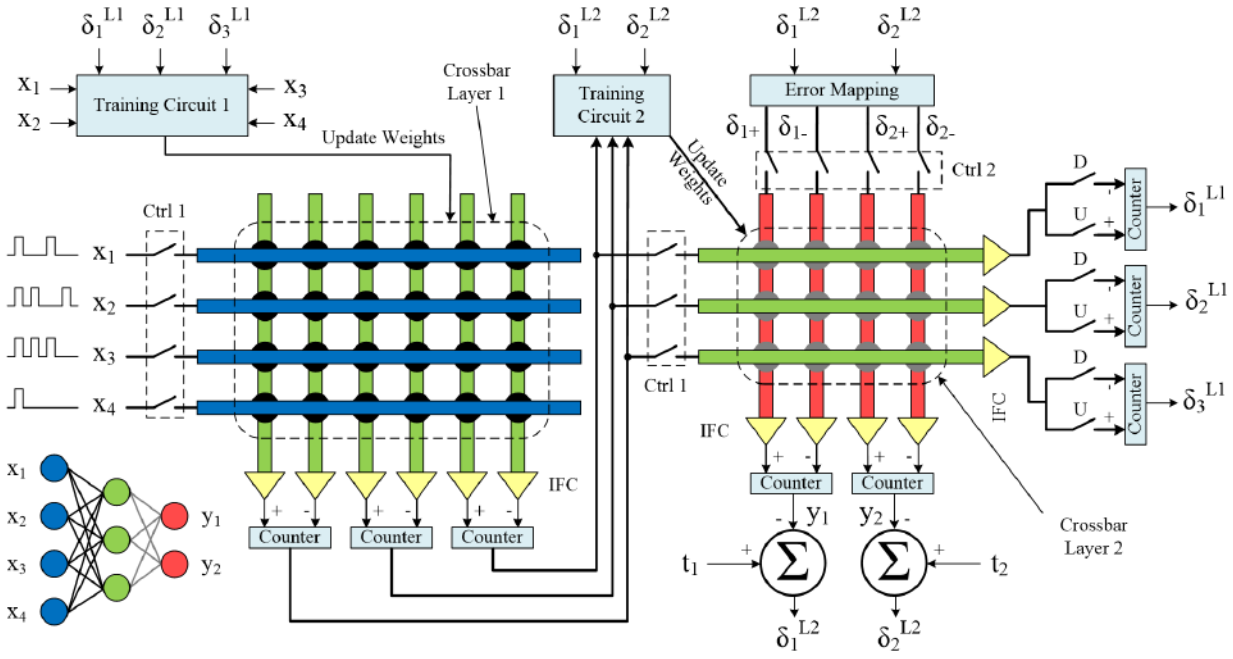



Figure 16: Hardware implementation of a two layer memristor crossbar array. A 4×3×2 neural network diagram is shown in the lower left corner, and its circuit implementation is shown in the rest of the figure.

pulses and applied to the word lines of the second crossbar layer as well. After evaluating the output vector $y$, the desired output vector $t$ is compared with the $y$ and the sign of the error vector $\delta^{L2}$ is obtained. The superscript in the error vector refers to the crossbar layer number.

*2) The backpropagation step for crossbar weight updating.*

According to (3.2), for any hidden layer, the summation part can be thought of as the error vector $\delta^{L2}$ multiplied by the transpose of the conductance matrix of the second crossbar layer $G_2^{'}$. So $\delta^{L2}$ will be applied vertically, from the top, on the bit lines and currents will be collected horizontally, on the left, from word lines. Since the crossbar array is originally designed with two memristors per synaptic weight and for feed forward propagation, $\delta^{L2}$ has to be modified before applying to the crossbar. The error mapping block generate two error vectors, which will be applied sequentially to second crossbar. Each one of those vectors is double the size of the original error vector and mapping is done according to Table 1. Each $\delta_j$ can take only three values: +1, 0, -1. Two vectors, and two corresponding values per vector $\delta_{j+}$ and $\delta_{j-}$, are generated for each value $\delta_j$. During the application of the two generated error vectors, the switches with *ctrl 2* (*ctrl 1*) are closed (opened). The top row corresponding to each of the three values in Table 1 is applied first and the switched marked as $U$ ($D$) are closed (opened), while the bottom row is applied second and the switches marked as $D$ ($U$) are closed (opened). After that, the sign of the error vector $\delta^{L1}$ is obtained, which is merely the MSB of the counter (for polarity) and a zero detector.

Each crossbar has its own weight updating block (training circuits in Fig. 16). Each training block takes the corresponding input vector $x$ and the error vector $\delta$, then, weight updating is done on two steps, as shown in Fig. 17. Assuming the error vector, $\delta$, and

the amount by which each weight is updated, $\eta \times x$, as shown in the Fig. 17(b), the effective conductance $G_{i1}$ ($G_{i2}$) needs to be increased (decreased). Since $G_{ij} = G_{ij}^{+} - G_{ij}^{-}$, the

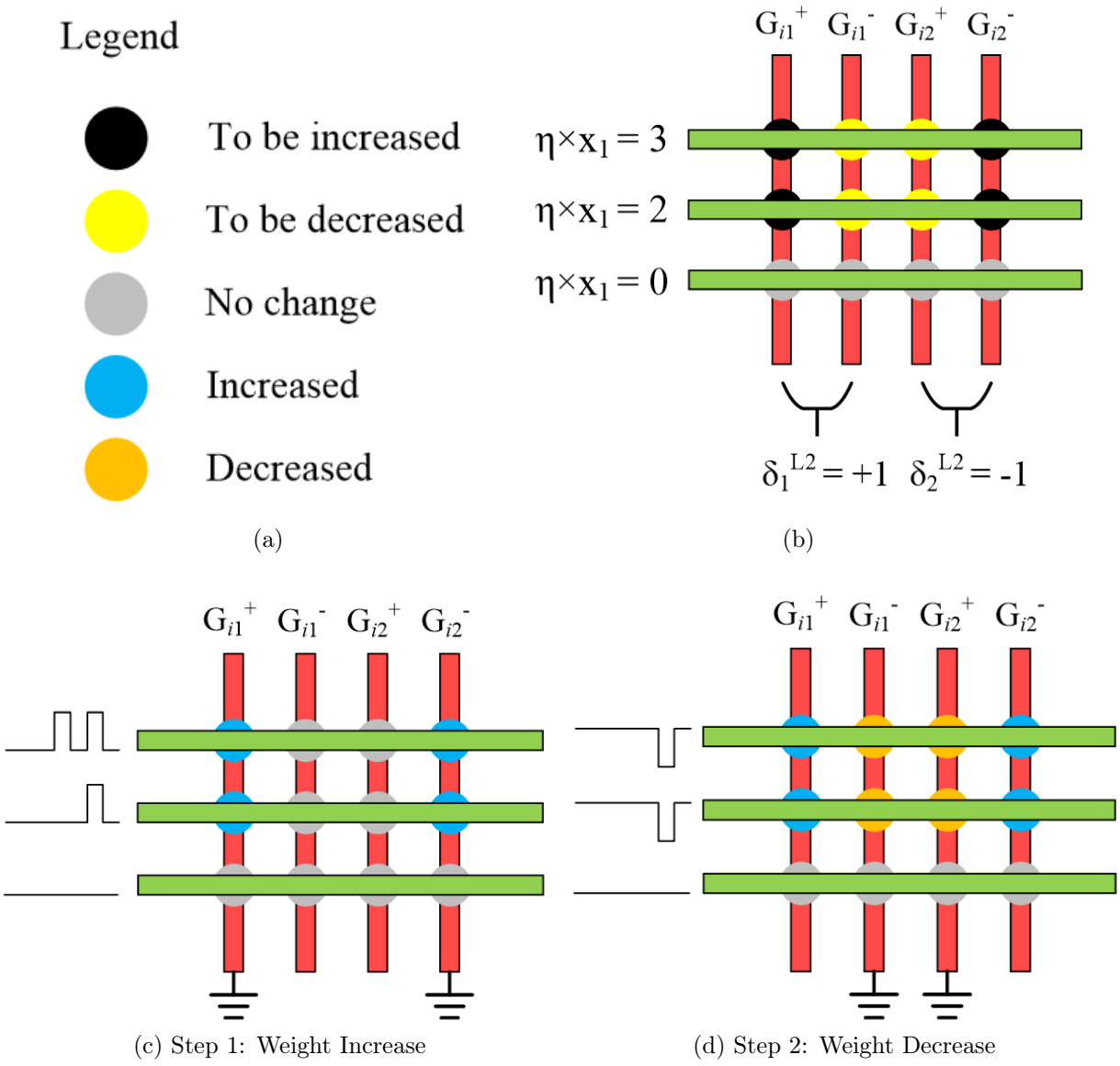



Figure 17: Training the second crossbar in Fig. 16. (a) shows the legend for different colors, while (b) depicts the assumed value for the error vector $\delta$ and the input vector $\eta \times x$. (c) and (d) show the only two required steps to program the whole crossbar.

Table 1: Error vector mapping for different values of $\delta_j^{L2}$

| $\delta_j^{L2}$ | $\delta_{j+}^{L2}$ | $\delta_{j-}^{L2}$ | $\delta_j^{L2}$ | $\delta_{j+}^{L2}$ | $\delta_{j-}^{L2}$ | $\delta_j^{L2}$ | $\delta_{j+}^{L2}$ | $\delta_{j-}^{L2}$ |
|---|---|---|---|---|---|---|---|---|
| +1 | 1<br>0 | 0<br>1 | 0 | 0<br>0 | 0<br>0 | -1 | 0<br>1 | 1<br>0 |

weight update is done deferentially with more priority for the conductance to be increased. For example, $\eta \times x_1 = 3$ which means both $G_{11}{}^{+}$ and $G_{12}{}^{-}$ ($G_{11}{}^{-}$ and $G_{12}{}^{+}$) will increase (decrease) by 2 (1) levels. If any value in the error vector (input vector) is zero, then the memristors in the corresponding columns (row) are not updated. Fig. 17(b) shows the required memristors to be increased or decreased. Training is done in two steps: (i) weights to be increased, in which positive programming pulses are applied to the crossbar and the corresponding columns are connected to ground (Fig. 17(c)), and (ii) weights to be decreased, in which negative programming pulses are applied to the crossbar and the corresponding columns are connected to ground (Fig. 17(d)).

To the authors best knowledge, no one has proposed a fast training scheme like this before. The only ones which are near to the proposed training scheme requires 4 steps to complete training [52, 58]. Since the activation function used in [52, 58] is a *tanh* function, the input $x$ of the hidden neurons could be negative or positive; thus, increasing the different combination required for training into 4 instead of 2 as proposed here.

Table 2: Properties of the used datasets and their corresponding crossbar systems.

| Dataset | Network Size | Training Set Size | Testing Set Size |
|---|---|---|---|
| Digits [5] | 32×10×6 | 600 | 200 |
| Cancer1 [6] | 9×10×2 | 594 | 105 |
| Thyroid1 [6] | 21×15×3 | 6120 | 1080 |

### 3.4 System Evaluation

Three different datasets were used to evaluate the proposed training scheme. The first *Digits* dataset is simple binary figures corresponding to digits 0-5 from [5]. Training and testing ensembles were generated from the basic figures by randomly injecting errors in them [5]. The other two datasets are Cancer1 and Thyroid1 from the Proben1 database [6]. All the datasets were implemented using a two-layer spiking-based system and weights were updated according to the proposed algorithm in Section 3.3. Table 2 summarizes the properties of the three datasets and the corresponding crossbar network configurations. For the *Digits* and *Cancer1* datasets, inputs are binary and 10 discrete levels, respectively, while inputs are analog for the *Thyroid1* dataset. In order to represent the inputs by spikes, each input of the Thyroid1 dataset is mapped between 0 ~ 1, then discretized into 10 levels, where each level corresponds to one spike.

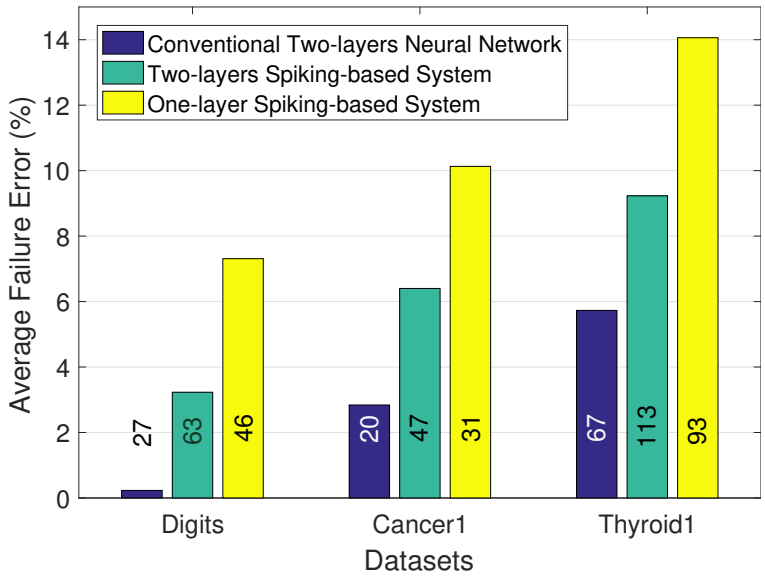
In the following subsections, we will evaluate how much improvement each of the three parts of the proposed systems achieved, namely, extending the system to two-layers, controlling $IFC_{\max}$, and finally training the system online.

### 3.4.1 Ideal System Performance

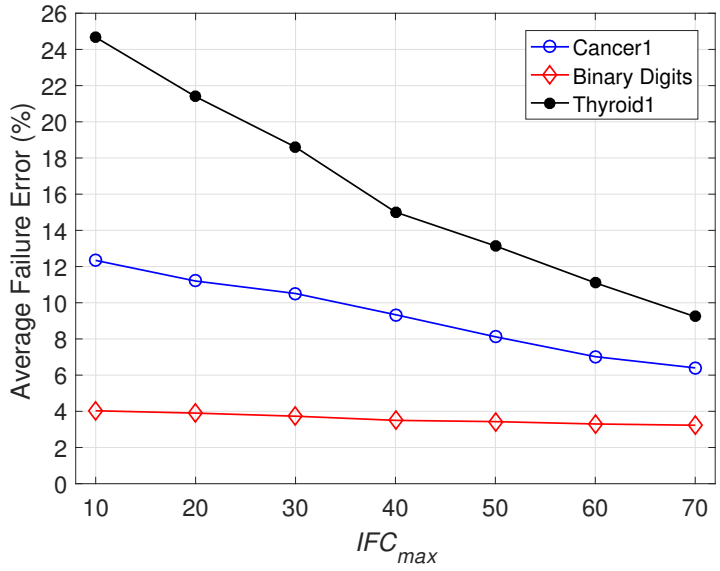
Fig. 18(a) shows the performance of the three networks with respect to average failure error. For each system, a comparison is made among that proposed system, a one-layer spiking-based system, and a conventional two-layer neural network system. All simulations were done using MATLAB with the overall failure error averaged over 100 runs. As you can see, the two-layer system has adequately improved the overall failure error by 42% (on average) over the one-layer system and still not far away from the conventional neural network system.

The average number of epochs, averaged over 100 runs, to achieve this accuracy is shown inside the bar for each system. Each epoch represents one iteration of the while loop in 3.1. As shown, the epochs needed by the two-layers system to converge are more than those needed by the one-layer system. This is something expected since we are updating two crossbars instead of one.

As aforementioned in Section 3.2.3 that controlling $IFC_{\max}$ is very crucial for intermediate stages IFC, we further investigate the effect of $IFC_{\max}$ on the system performance. The two-layers spiking-based systems were evaluated for different value of $IFC_{\max}$, as shown in Fig. 18(b). All simulations follow the same rules depicted before. It can be easily deduced that the performance of *Digits* system doesn't really change for different $IFC_{\max}$. On the other hand, the performance of Cancer1 and *Thyroid1* systems dramatically depend on the $IFC_{\max}$ value. Such a behavior can be explained as follows: for the *Digits* system, the input resolution is binary, so it does not need intermediate stages IFC with high resolution. On the other hand, both *Cancer1* and *Thyroid1* have higher input resolution (10 levels per input) which definitely needs high resolution for the intermediate stages IFC.




(a)




(b)

Figure 18: Ideal system evaluation for the architecture shown in Fig. 16. (a) shows a comparison between average failure error for conventional two-layers neural network (blue), two-layers spiking-based system (green), and one-layer spiking-based system (yellow). Results are shown for three datasets, namely Digits [5], Cancer1, and Thyroid1 [6], and (b) shows the average failure error for Digits (red), Cancer1 (blue), and Thyroid1 (black) versus $\mathrm{IFC}_{\mathrm{max}}$ of the intermediate stages IFC.

We couldn't increase $IFC_{\max}$ beyond 70 as the simulation time increases dramatically. It is worth mentioning that increasing $IFC_{\max}$ for intermediate stages IFC requires bigger counters, so the overall system area and power increase, especially for bigger crossbar arrays. Moreover, the hardware implementation of the system gets slower, as more spikes per input have to be applied sequentially for the second crossbar.

### 3.4.2 Memristor Process Variations

For better and more realistic system evaluation, we are going to take into consideration memristor process variations. It is widely known that there is a deviation between the programmed and the desired memristor resistance due to process variations, in which most of those variations follow Gaussian distribution [59]. Those variations can dramatically degrade the system performance, especially for offline trained systems. Online training helps to significantly decrease the effect of process variations by constantly updating memristor weights on the fly. For our evaluation, we will assume that the memristor resistance is:

$$R_p = R_{desired} + \nabla R, \tag{3.3}$$

where $R_p$ is the final programmed resistance, $R_{\mathrm{d}}$ is the desired resistance value, and $\nabla R \sim N(0, \sigma)$ is a random variable that follows a Gaussian distribution of mean value 0 and standard deviation $\sigma$. The simulation results for the two-layered spike-based systems using online (solid lines) and offline (dashed lines), are shown in Fig. 19(a). The average failure error, averaged over 100 runs, is plotted against the standard deviation $\sigma$, whose maximum value is restricted to 5% which is widely accepted [59]. For offline training, MATLAB was used to train each memristor resistance value only from the pool of allowable resistance states.

Process variation modeling is applied when the trained resistance values are programmed to the memristor. On the other hand, online training takes into consideration process variation during each epoch of training. As shown in Fig. 19(a), online training has significantly improved the average failure error, where improvements at the worst process variation ($\sigma = 5$) are 21.97%, 17.95%, 16.18% for Thyroid1, Cancer1, and Binary Digits datasets, respectively. As the figure depicts, the rate of improvements increases as the process variations gets worse and is not the same for the three datasets. The latter can be explained as the sensitivity for process variations increase as the network size and input feature details increase.

The number of epochs, averaged over 100 runs, needed for training the online system versus $\sigma$ is shown in Fig. 19. It can be concluded from this figure that the rate of increase of the number of epochs is not constant, and changes, approximately, exponentially according to the complexity of the used dataset. This explains why the number of epochs needed for *Cancer1* for $\sigma = 1$ and 2 is lower than the rest of the values.

## 3.5 Conclusions

In this chapter, we demonstrated a hybrid spiking-based multi-layered self-learning neuromorphic computing system. We also proposed a simple and effective online training algorithm for spiking systems, which reduces the required steps for weights updating into half of the previously published training algorithms. System evaluation was carried out, using three different datasets, namely, Binary Digits [5], Cancer1 [6], Thyroid1 [6], which showed overall performance improvement of 42%. Moreover, the online training algorithm shows adequate immunity against memristor process variations. Since the results shown here are

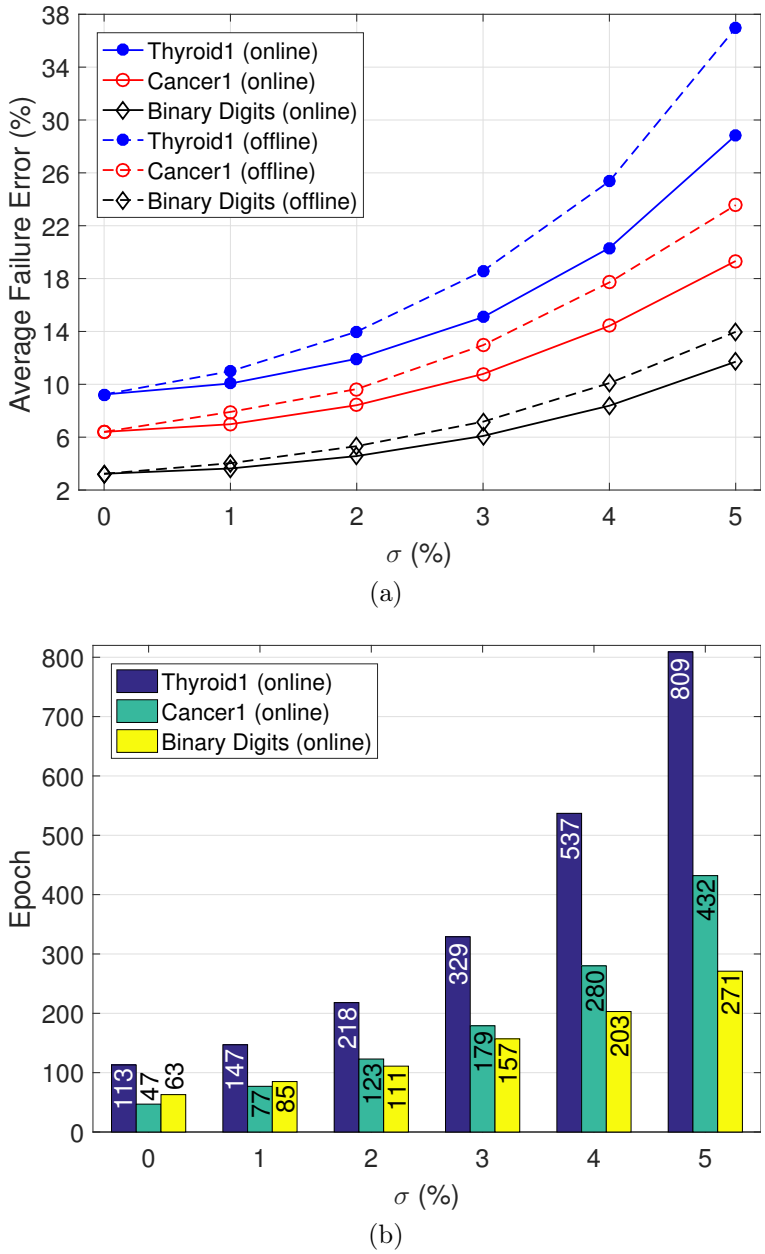



Figure 19: System evaluation, after taking into account process variations, for the architecture shown in Fig. 16. (a) shows a comparison between average failure error for the two-layers spiking-based system, using online (solid) and offline (dashed) training, versus standard deviation $\sigma$ for Thyroid1 (blue), Cancer1 (red), and Binary Digits (black). (b) shows the average number of epochs for online trained systems versus standard deviation $\sigma$ for the same datasets.

promising, potential future work will be investigating the effect of increasing the crossbar size and the number of layers on the overall system performance, as well as extending the concept of neuromorphic computing to different types of neural networks.

# 4.0 Hardware Implementation of Echo State Networks using Memristor Double Crossbar Arrays

## 4.1 Introduction

Recently, the amount of data need to be processed in real time is growing massively. Spatiotemporal data, such as weather forecasting and speech recognition data, require a huge amount of real-time processing to extract the essential information out of it. A common trait among spatiotemporal data is the need to extract the behavior of the data over different time windows to be able to classify or predict their behavior in the future. For example, meteorologists can predict the atmospheric behavior for an upcoming period of time depending on its past behavior. Utilizing general-purpose computing systems to process this huge amount of data in real-time is inefficient in terms of cost and power consumption, especially for application specific devices. In addition, conventional von Neumann systems cannot keep up with such huge amount of data due to the well known "memory wall" phenomenon [25].

Neuromorphic computing systems can provide an excellent trade-off between real-time processing, power consumption, and overall accuracy. Inspired by human brains, neuromorphic computing systems store and process the data at the same location, which overcome the "memory wall" phenomenon in von Neumann systems. Most of these systems rely on new emerging devices, such as Resistive Random Access Memories (ReRAM), or Memristors, as an important building block, because of their synaptic like behavior and low feature size [46,47]. Fig. 20 depicts a simple example on how to implement a typical fully connected 4x2 feed-forward neural network (shown in Fig. 20(a)) using crossbar arrays of memristors

(shown in Fig. 20(b)). The inputs to the neurons (blue circles) are represented by voltage signals at the horizontal (blue) wires, while the outputs of the neurons (gray circles) are represented by currents signals at the vertical (gray) wires. The conductance values of the memristors at each intersection in the crossbar array correspond to the weights connecting neurons of different layers. Since the weights can take any values, neuromorphic systems make use of the ability to program the memristors resistance to represent any weight value.

Significant work have been done investigating the potentials of using neuromorphic systems to represent different topologies of neural networks, especially feed-forward neural net-

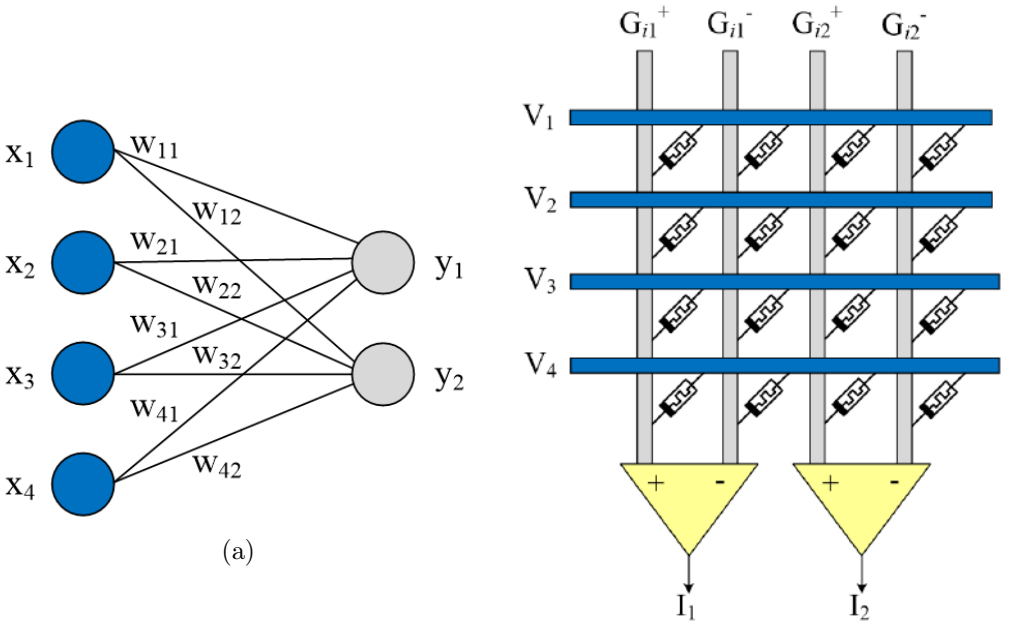



Figure 20: Neural network implementation using memristor crossbar arrays. (a) shows a conventional diagram for 4×2 neural network while (b) shows the crossbar implementation of that network.

works [5, 46, 52, 58, 60]. For instance, Liu *et al.* [5] investigated the use of memristor crossbar arrays in implementing a digital image recognition system using feed-forward neural networks. The system shows energy savings more than 50% with only 1.46% increase in the average failure error [5]. Hassan *et al.* [61] also implemented a multi-layered self-learning neuromorphic system, which implement a multi-layer feed-forward neural network using memristor crossbar arrays which are capable of online weight updating. The system has been evaluated on three different datasets showing promising results compared to single layered systems and good immunity against memristor process variations [61].

Although most of the investigated systems implement feed-forward neural networks, those types of networks are not suitable for spatiotemporal datasets. Echo State Networks (ESNs), which are a special type of partially-trained Recurrent Neural Network (RNNs), can represent spatiotemporal datasets very well [62]. ESNs have many attractive features (as will be discussed later on Section 4.2.2), but the feedback connections within its hidden layer neurons enable it to extract the spatial and temporal behavior, and correctly represent spatiotemporal datasets. Software implementations of ESNs showed promising results in numerous applications such as: forecasting of water inflow for a hydro-power plant [63], speech recognition [64], and many more [65]. On the other hand, hardware implementations of ESNs are scarce in literature. For example, Donahue *et al.* [66] (Zhang *et al.* [67]) implemented a hardware realization for RNN using a bi-stable memristor based synapses (spiking-based neurons), along with other CMOS circuitry. Both systems were evaluated for spoken digits recognition showing an average accuracy of 67% for [66] and 97% for [67]. Moreover, Kulkarni *et al.* [68] used memristors and genetic algorithms, in a graph-based approach, to implement a hardware-based ESN that distinguishes between triangular and square waveforms.

In this chapter, we propose a new hardware implementation design of ESN based on memristor double crossbar arrays. To the authors best knowledge, such a hardware realization of ESN has never been proposed before. Our main contributions are:

- Introducing a memristor double crossbar array to implement the recurrent connection within the ESN reservoir, as will be discussed later on Section *4.3.1.*
- Offering a detailed procedure for designing and simulating the proposed architecture.

The rest of the work is organized as follows: Section 4.2 introduces the theory of ESN and the fundamental building block of the system. Section 4.3 explains the details of the proposed hardware implementation of ESN and its design considerations. Section 4.4 evaluates and discusses the proposed system from different aspects. Finally, conclusion will be provided in Section 4.5.

## 4.2 Preliminary

### 4.2.1 Memristor Devices and Model Used

Memristors are usually made of two conductive layers with an oxide layered in between. There has been many researches trying to investigate the best oxide to be used, but the most promising ones are $TiO_{2-x}$ [55] and a layer composed of $HfO_x$ and $AlO_x$ [49]. For energy efficiency reasons, devices with higher resistances ranges are a must. Devices made of $HfO_x$ and $AlO_x$ [49] have a high resistance ratio of ($R_{OFF}/R_{ON}\approx1000$) and ($R_{ON}\approx10K\Omega$), which is good enough to be utilized in our system. In addition, another device is added on top of memristor to improve selectivity and suppress the sneak paths [40,56]. The most common

devices used as selectors are transistors [5] and devices made of Superlinear Threshold Layer (STL) [4]. Using transistors as selectors, or 1-Transistor-1-Resistor (1T1R) configuration, is more reliable and has negligible effect of on the overall conductance of the composed cell [5].

In this work, each cell of the crossbar array will be composed of 1T1R, where the resistance range of that structure is [50K$\Omega$, 1M$\Omega$] [5]. The whole range is divided into 8 discrete levels, which are the only allowable levels during training.

### 4.2.2 Echo State Networks and Reservoir Computing

In 2001, Jaeger [62] introduced a new network model based on recurrent neural networks, which is called Echo State Networks (ESNs). Nowadays, ESN model, and others that share the same basic structure, have been categorized under a unified model called Reservoir Com-

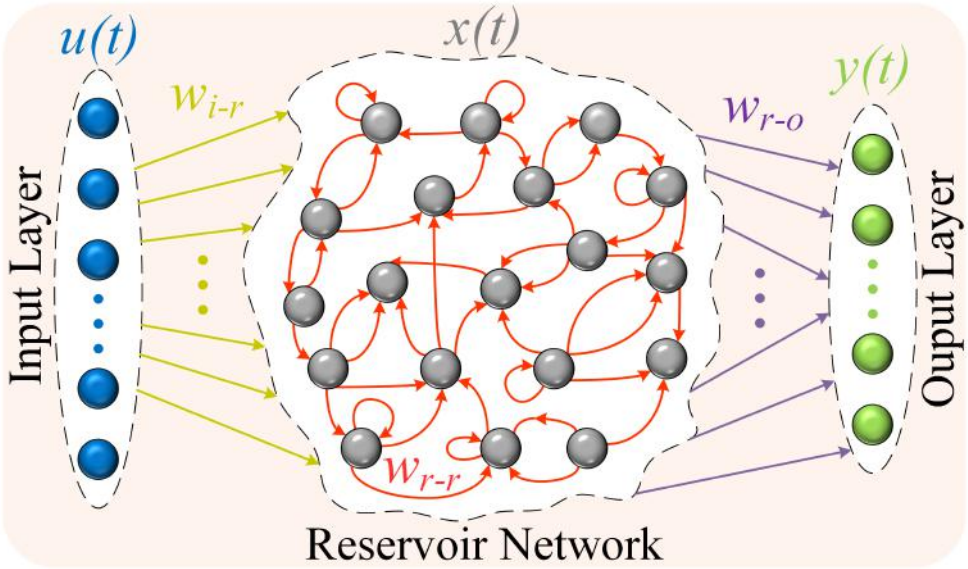



Figure 21: Generic Echo State Network (ESN) with an input layer, reservoir layer, and output layer.

puting (RC) model [65]. RC model has two major significant changes than the traditional fully-connected feed-forward neural networks, 1) network layers don't have to be fully connected anymore, and 2) recurrent connections are included among units of the same layer, as well as different layers. Fig. 21 depicts a generic ESN model with an input, a hidden (reservoir), and an output layer. Each unit in the input (hidden) layer is now connected to a random set of units in the hidden (output) layer. In addition, every unit in the hidden layer has random connections to different units in the same layer. These recurrent connections change the dynamics of the hidden layer. Now, at any given moment, the state of the hidden layer depends on the input applied at this moment and the previous state of the hidden layer, hence the name reservoir layer. Moreover, units of the reservoir layer can have different activation functions depending on the target application. Consequently, RC model is the best when it comes to representing spatiotemporal datasets, where datasets are timely or spatially linked over a certain window.

As depicted in Fig. 21, a generic ESN model has three layers, namely input, reservoir, and output layer. Due to the complicated dynamics in the system, the weights connecting the input and reservoir layer, $w_{\text{i-r}}$, and the weights within the reservoir layer, $w_{\text{r-r}}$, are randomly initialized at first and kept fixed during the training process. On the other hand, the weights connecting the reservoir and the output layer, $w_{\text{r-o}}$, are the only allowable weights to be trained using any of the known training techniques. The equation governing the reservoir dynamics is as follows: given an input dataset of $U = u_1, u_2, \ldots, u_{\text{N}}$ defined over discrete time $t \in \mathbb{Z}$ , where $u(t)$ is the input vector given at time $t$, $x(t)$, which is the reservoir state at time $t$, is defined as:

$$x(t) = f[w_{i-r} \times u(t) + w_{r-r} \times x(t-1)] \tag{4.1}$$

where *f()* is a nonlinear activation function for the reservoir units. The output *y(t)*, at time $t$, is defined as:

$$y(t) = g[w_{r-o} \times x(t)] \tag{4.2}$$

where *g()* is the activation function for the output layer units.

The ESN model can be thought of as if it projects the input space into a random higher-dimensional feature space, and then maps it down to a lower-dimensional output space. Some sophisticated feed-forward network has a similar approach [69], however, the recurrent connections make it easier for ESN model to capture spatiotemporal signals efficiently, in terms of accuracy and network size.

## 4.3 Proposed Architecture and Design Procedure

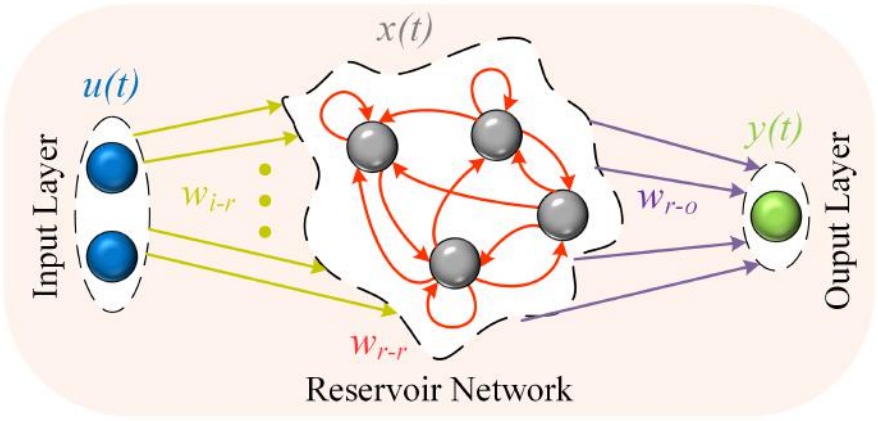
This section is composed of two parts: first, we will discuss the hardware implementation of a generic ESN using memristor double crossbar arrays. Second, the design procedure used to train and test the system will be explained. It's worth mentioning that the proposed system provides the basic foundation for the evaluation carried out in Section 4.4.

### 4.3.1 Generic ESN Hardware Implementation

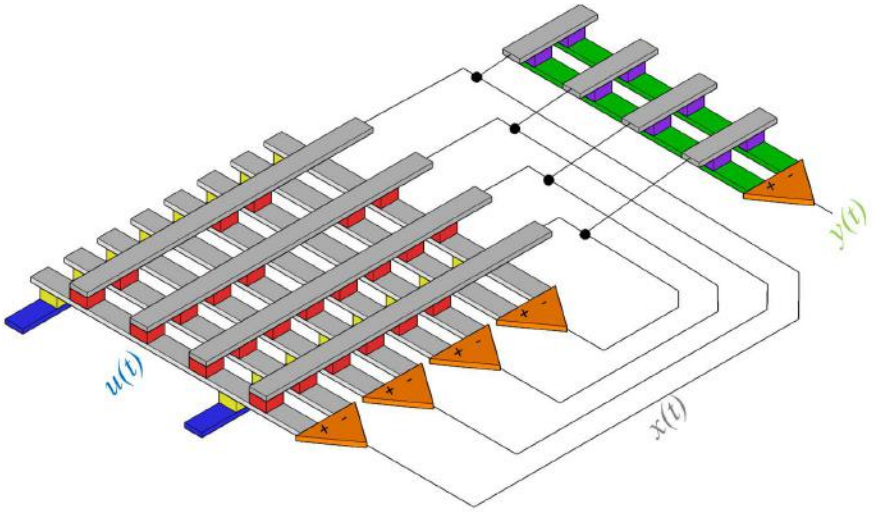
It is better to illustrate how to implement ESN using memristor crossbar arrays by a generic example. Fig. 22(a) shows a simple 2×4×1 ESN, which has 2 input neurons (blue circles), 4 reservoir neurons (gray circles), and 1 output neuron (green circle). As can be seen in Fig. 22(a), the input layer is fully connected (yellow arrows) to the reservoir, the reservoir units are randomly connected (red arrows), and the output layer is fully connected to the reservoir (purple arrows). The hardware implementation of this network is depicted in Fig. 22(b), which is color matched to the network in Fig. 22(a). The weight matrices $w_{\text{i-r}}$, $w_{\text{r-r}}$, and $w_{\text{r-o}}$, in eq (4.1) and (4.2), are represented by the conductances of the yellow, red, purple memristors (cubes in Fig. (22(b))), respectively. Since each weight can be either negative or positive, two memristors are used to represent one weight, as shown in Fig. 20(b). For instance, $G_{11}$, which represent $W_{11}$ in Fig. 20(a), is $G_{11} \equiv G_{11}^{+} - G_{11}^{-}$.

Input signals, *u(t)*, are represented by voltage signals applied to horizontal blue lines. This voltage signal is multiplied by the memristor conductances (yellow memristors) and corresponding currents are collected at the bottom of the vertical gray lines. Currents are then converted by means of the summation amplifier (orange blocks) to voltage signal *x(t)*, the operation of the summation amplifier will be discussed later on this section. The signal *x(t)* is then fed back to the crossbar array by means of the top horizontal gray lines and the red memristors, which represent the recurrent connections of the reservoir. In addition, the signal *x(t)* is fed to the second crossbar, and the currents are collected at the bottom of the green vertical lines, then converted to a voltage representing *y(t)*. It can be said that, the double crossbar array implements eq. (4.1), while the single crossbar array represent eq. (4.2). It's worth mentioning here that sneak paths [40] cannot be formed between the lower

and upper parts of the double crossbar arrays because of the following: 1) the applied voltage signal range is always kept below the programming threshold voltage of the memristors used and 2) the transistor used with each memristor enhance selectivity and prevent sneak paths.




(a)



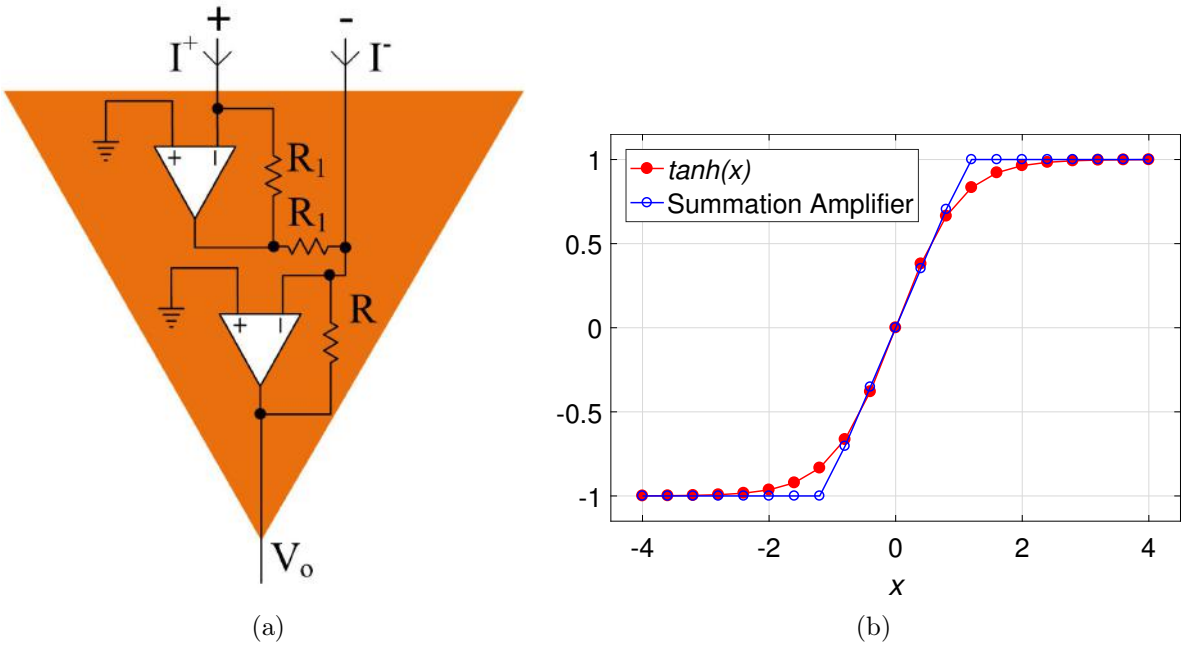

(b)

Figure 22: Hardware implementation of ESN. (a) shows a 2x4x1 ESN with random connections inside the reservoir and (b) depicts the hardware implementation of this network model. both figures are color matched.

The circuit diagram of the summation amplifier (the orange block in Fig. 22(b)) is shown in Fig. 23(a). This block is composed of two operational amplifiers, in which each one is connected in a negative feedback configuration. The summation amplifier represents the equation:

$$V_o(t) \;=\; R\left[I^{+}(t) - I^{-}(t)\right] \tag{4.3}$$

where $V_o(t)$ is the output voltage signal, $I^{+}(t)$ and $I^{-}(t)$ are the +ve and -ve current signal from the crossbar array, and $R$ is the resistance value. $R$ is adjusted according to the application to use the full range of the amplifier. The summation amplifier IV characteristics (blue curve) is shown in Fig. 23(b). It is merely a linear relation between the +ve and -




(a) (b)

Figure 23: Summation amplifier (orange block) in Fig. 22(b) (a) shows the circuit implementation of the amplifier and (b) depicts the amplifier IV characteristics (blue curve) versus *tanh(x)* characteristics (red curve).

ve supplies of the operation amplifier. It can be seen that the summation amplifier IV characteristics is of a very close resemblance to the IO relation of *tanh(x)* function, under the normal range of operation.

Two things worth mentioning here, first, the summation amplifier virtually connect the horizontal lines to ground, which ensures the correct operation of the crossbar array. Second, the non-linear IV characteristics of the summation amplifier is used to implement the activation functions *f()* and *g()* in eq. (4.1) and (4.2).

### 4.3.2 Design Procedure

Designing memristor crossbar arrays for ESNs requires composite design procedure that incorporates utilizing a software tool, such as MATLAB. Fig. 24 shows the required steps to design memristor crossbar arrays for ESN. First of all, the input-reservoir weights, $w_{\text{i-r}}$, and the reservoir recurrent weights, $w_{\text{r-r}}$, are randomly initialized (step 1), then, those weights are mapped to the allowable conductance levels and programmed to their corresponding memristors in the double crossbar array (step 2). After that, input signal *u(t)* is pre-processed, using scaling and normalization, to span the valid input range of the crossbar array (step 3). The double crossbar array is now ready for operation. The first few samples of the input signal are applied to the reservoir in a process called "reservoir warm-up", which initialize the dynamics of the reservoir. The number of the samples used depends on the nature of the dataset. Next, the training signal is applied to the double crossbar array (step 4), and the reservoir states, *x(t)*, are collected (step 5). Those reservoir states, collected at step 5, are used to train the reservoir-output weights, $w_{\text{r-o}}$ (step 6). The algorithm used to train $w_{\text{r-o}}$ depends directly on the dataset under investigation, however for the dataset used in this

work, linear regression [62] will be used. The trained weights are mapped to the allowable conductance levels and programmed to their corresponding memristors in the single crossbar array (step 7). Now, the crossbar array system is ready for testing, where the testing signal is applied to the double crossbar, then the reservoir states are applied to the single crossbar array (step 8). Last but not least, the output signal, *y(t)*, is collected by the software (step 9), and finally system evaluation and validation are carried out (step 10).

The previously illustrated design procedure will be exactly used in the evaluation part in Section 4.4.

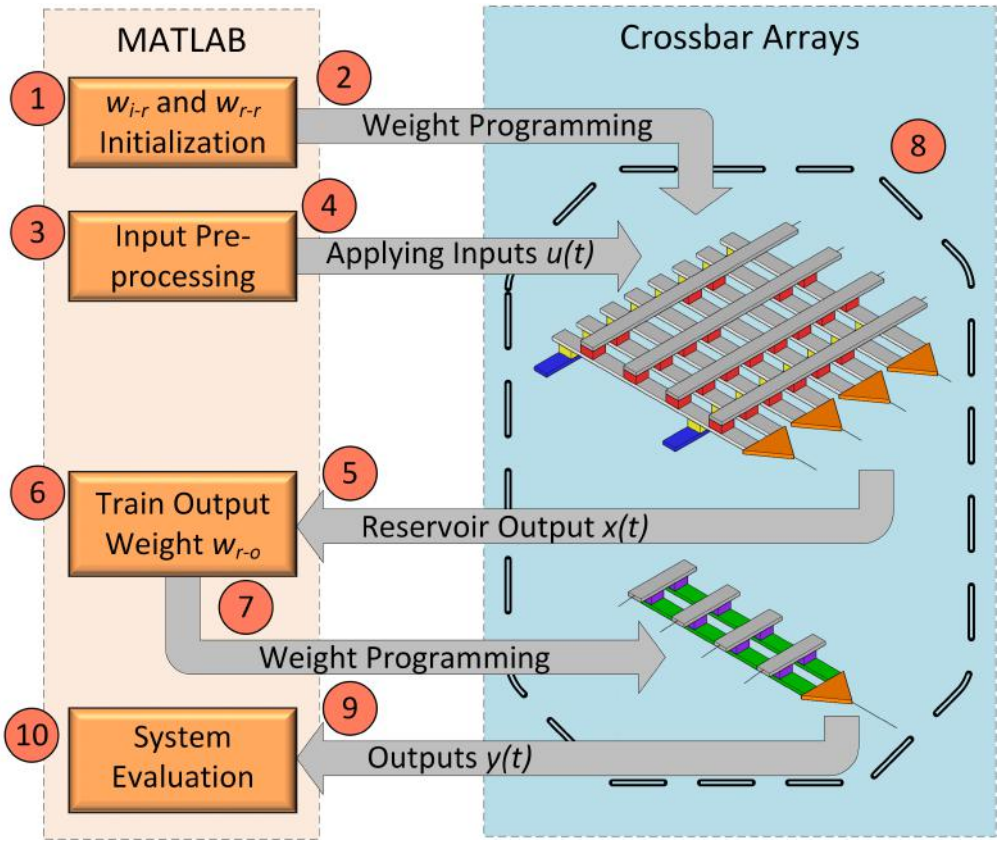



Figure 24: Design procedure steps to implement memristor crossbar arrays for ESNs.

## 4.4 System Evaluation

The dataset used to evaluate the performance of the proposed system is generated from the famous Mackey-Glass equation [70]. It is a nonlinear time-delayed differential equation, defined as:

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1+(x(t-\tau))^n} - \gamma x(t) \tag{4.4}$$

where $\tau$ is a non-negative time delay used in evaluating a signal. $n$ is a non-negative shape parameter that helps to describe the time-delayed feedback/response. $\beta$ and $\gamma$ are non-negative parameters.

The importance of this equation arises from the fact that it can display a wide variety of extremely chaotic behavior, depending on the adjustable delay term $\tau$, because its value at any time may depend on its entire previous history. Mackey *et al.* [70] originally presented eq. (4.4) to explain the complex dynamics in physiological control systems by way of bifurcations in the dynamics. He suggested that many dynamical diseases, which are physiological disorders, can be characterized by changes in qualitative features of dynamics, that corresponds mathematically to bifurcations in the dynamics of the system. Those bifurcations could be induced by changes in the parameters of the system, which can be caused by disease or environmental factors, such as drugs. Consequently, many researchers extensively studied bifurcations in mathematical models of physiological systems and related these with the abnormal (pathological) dynamics of disease [71].

Table 3: Different parameters for the dataset and system under evaluation.

| $\beta$ | $\gamma$ | $\tau$ | $n$ | ESN size | Training Set Size | Testing Set Size |
|---|---|---|---|---|---|---|
| 0.25 | 0.1 | 17 | 10 | 1×M×1 | 2000 | 1500 |

Table 3 summarizes different parameters of the dataset used for evaluation. We have only one input unit, reservoir of size M, and one output unit. The reservoir is fully connected and its size is kept variable because it will be one of the parameters used in evaluating the system. The input is pre-processed and the input signal range is [-0.6, 0.6]. The proposed system is trained to do autonomous signal generation of the input, which is done by making the target signal one-sample-ahead of the input training signal. After training, the system is fed its own prediction, which autonomously generates a signal, for a certain period of time, that closely matches the testing signal. Our metric here in evaluating the system is the Mean Square Error (MSE) between the generated signal and the testing signal. All simulations were done using MATLAB on a machine having Intel Core i7 Processor running at 2.9 GHz and 8 GB of RAM.

In the following subsections, we will evaluate the system performance for different parameters, such as: reservoir size, M, and the memristor resistance shift due to process variations.

### 4.4.1 Ideal System Performance

Fig. 25 depicts the MSE of the proposed hardware implementation (red) and the software implementation (blue) of ESN for Mackey-Glass dataset. As expected, the proposed

hardware implementation dropped the accuracy by 1%, on average, over the software implementation. This is due mapping the real valued weights to the discretized memristor weights and approximation of the activation function to the IV characteristics of the summation amplifier. The worst case accuracy drop was 1.8% at M=200, and the least accuracy drop was 0.53% at M = 800. It can be told from the curve that as M increases, the proposed hardware accuracy improves and the accuracy drop, compared to the software implementation, improves also. This can be explained as follows: increasing the reservoir size increases the reservoir dynamics non-linearity. As the non-linearity increases, the overall system becomes less susceptible to weights discretization error.

In order to have a better understanding of the impact of using the proposed system on the overall accuracy, a comparison between the actual testing dataset (green) versus the software (blue) and hardware implementation (red) outputs, are shown in Fig. 26, where the x-axis represent the time samples of the testing dataset and the reservoir size is 1000. As can be seen, there is a negligible slight difference between how the output of each system follows the testing dataset. It is worth mentioning that, since the system is trained for autonomous signal generation and given the size of the training and testing dataset, the proposed hardware implementation can accurately predict the samples following the training dataset with a window size of 75% of the training dataset size.

### 4.4.2 Memristor Process Variations

For a more realistic system evaluation, memristor process variations will be taken into consideration. It is widely known that there is a deviation between the programmed and

the desired memristor conductance due to process variations, in which most of those variations follow a Gaussian distribution [59]. These variations could drastically degrade the system performance. For our evaluation, we will assume that the programmed memristor conductance is:

$$G_p = G_d + \nabla G, \tag{4.5}$$

where $G_p$ is the final programmed conductance, $G_\mathrm{d}$ is the desired resistance value, and $\nabla G \sim N(0, \sigma)$ is a random variable that follows a Gaussian distribution of mean value 0

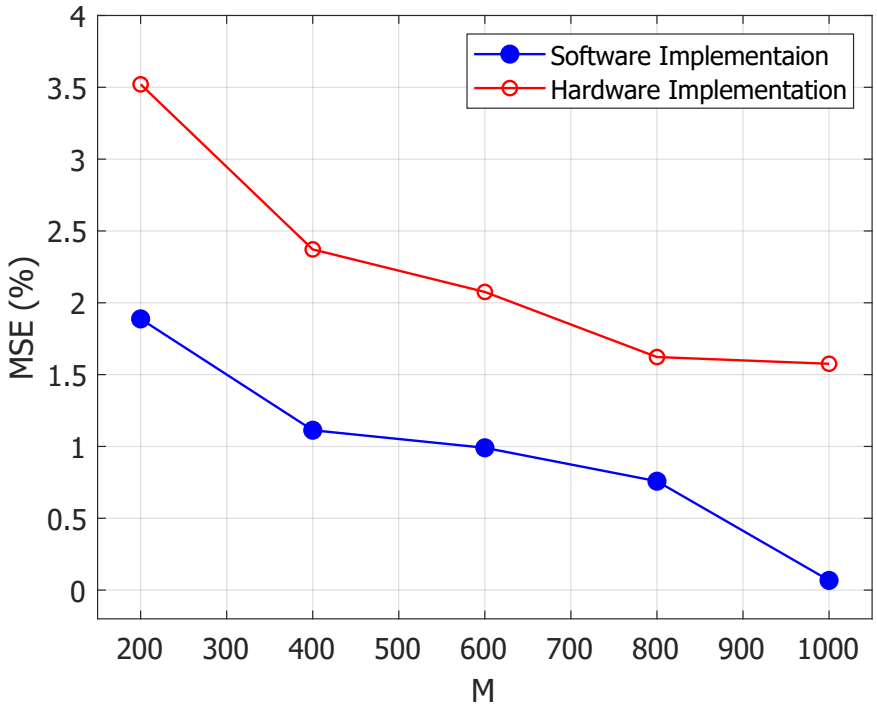
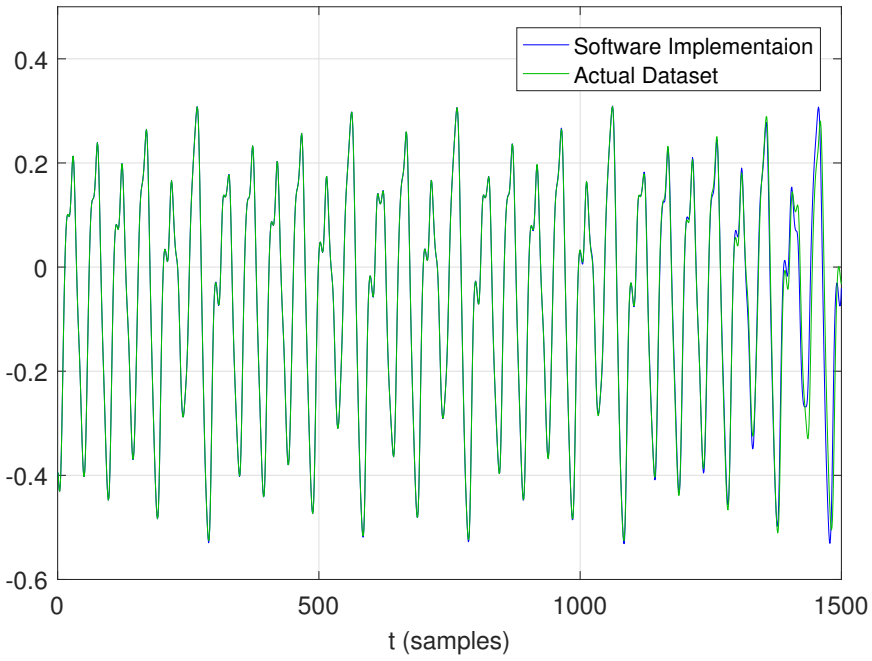
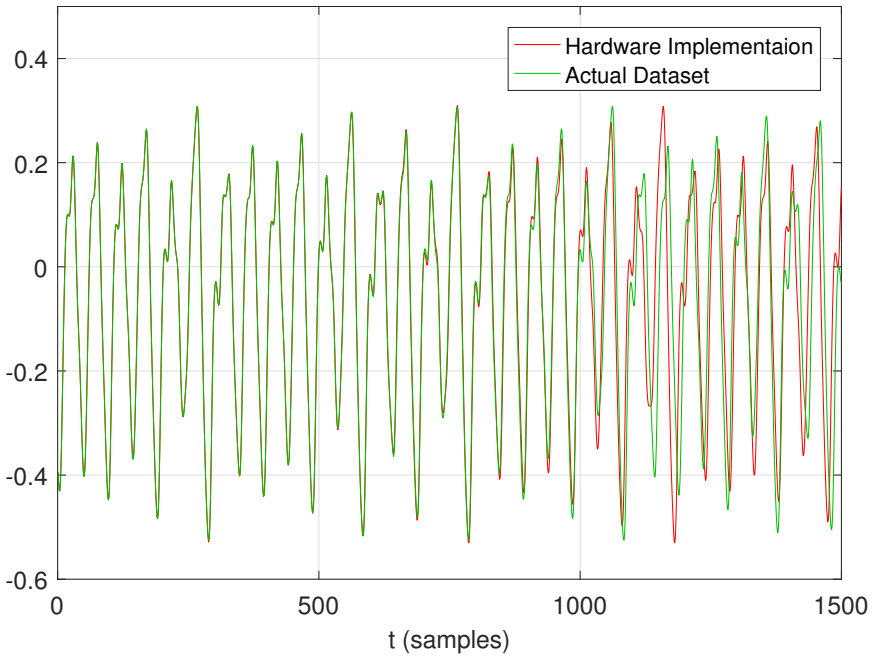



Figure 25: Performance comparison between the proposed hardware implementation (red) and the software implementation (blue) of ESN for Mackey-Glass dataset.




(a)




(b)

Figure 26: Comparison between the actual testing dataset (green) versus the ESN output, *y(t)*, for (a) software implementation (blue) and (b) hardware implementation (red). The results shown are for M=1000.

and standard deviation $\sigma$. Fig. 27 shows the MSE versus the reservoir size M for different values of $\sigma$. The maximum value of $\sigma$ is restricted to 5% which is widely accepted [59]. All the MSE values were calculated and averaged over a 100 runs for each (M,$\sigma$) pair. As can be told from the curves, for lower reservoir sizes (M = 200), the system accuracy is the worst. However, increasing the reservoir size improves the system accuracy dramatically. It is also noticeable that for larger reservoir size (M = 800, 1000), There is almost no difference in the system accuracy as $\sigma$ changes. Such a result implies that hardware implementation

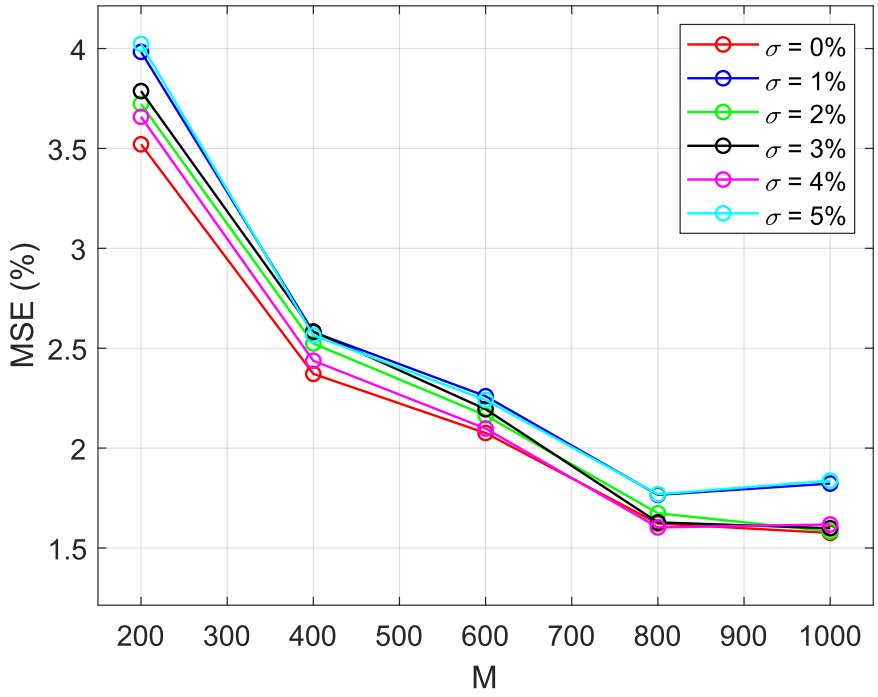



Figure 27: System evaluation, after taking into account process variations, for the proposed hardware implementation. It depicts the MSE versus the reservoir size M, for different values of $\sigma$.

of ESN is immune to memristor process variation, for larger reservoir sizes. A qualitative explanation of this phenomenon might be as follows: increasing the reservoir size increase the system dynamics non-linearity to the level where it can accommodate small changes in the memristor variations without affecting the overall performance of the system. In other words, the more non-linearly dynamical the system is, the less susceptible it is to process variations.

## 4.5 Conclusions

ESN model represents a special type of RNNs, which has the ability to regenerate spatiotemporal signals. For that reason, having a compact, fast and accurate hardware implementation of ESN would be a great achievement for application-specific devices. In this work, we proposed a hardware implementation for ESN model using memristor double crossbar array structure. In addition, a design framework for implementing the proposed system has been depicted. System evaluation, using the Mackey-Glass nonlinear time-delayed differential equation, showed very promising results compared to the conventional software implementation of the ESN. In addition, the hardware implementation demonstrated excellent immunity against process variations, for larger reservoir sizes.

Future work will be to investigate the performance of the proposed system for different reservoir topologies, such as Ring topology. In addition, different types of memristor crossbar arrays, such as spiking-based arrays [61] , will be investigated as a potential candidate for implementing ESNs.

# 5.0 Real-time Cardiac Arrhythmia Classification using Memristor Neuromorphic Computing System

## 5.1 Introduction

Electrocardiogram (ECG) is a diagnostic tool used to monitor electrical activity of the heart. Therefore, it can be used to detect irregularities in the heart rhythm due to the presence of certain arrhythmias [72]. Inspired by the fact that early detection of specific cardiac arrhythmias is essential to reduce mortality rate worldwide, Computer Aided Diagnosis (CAD) systems [73–75] were introduced as a promising approach that can aid the physicians to provide early arrhythmia detection and assessment.

Recently, several methodologies were proposed to improve the performance of arrhythmia CAD systems. Sayed *et al.* [73] proposed a distance series transform for phase space trajectories of 5 different beat types. In another study, features derived from spectral correlation analysis were classified using Support Vector Machines (SVM) [74, 76]. Additionally, methods including time-domain features which characterize the ECG signal morphology along with Linear Discriminant Analysis (LDA) and Fuzzy Logic based classifiers have been introduced in [77, 78] to provide robust arrhythmia classification systems. Furthermore, transform-domain features including discrete wavelets and higher-order spectra were proposed to train Probabilistic Neural Network (PNN) and SVM classifiers to obtain higher classification accuracy [74, 79].

In order to build fast and compact arrhythmia classification systems, researchers have utilized hardware platforms such as Digital Signal Processors (DSP) to implement their

classification and detection algorithms [75,76]. In [75], discrete wavelet transform, as well as PNN classifier, were used to classify 8 beat types on a DSP platform. On the other hand, Jeon et. al. [80] proposed an implantable System-On-Chip (SoC) that exploits both time and frequency domain techniques to detect cardiac arrhythmias. Due to space limitations, please refer to the references for further information [75, 76, 80].

Most of the previously introduced systems either require a lot of pre-processing, hard to be implemented on hardware platforms, or take a long time to detect an arrhythmia in ECG signals. Recently, neuromorphic computing systems have been proposed, implementing different types of Neural Networks (NNs), such as feed-forward NNs [61] and Recurrent NNs [81]. These systems are inspired by the working mechanism of the human brain and utilize new emerging devices, such as memristors, as an essential building block. Memristors have a few nanometers feature size and synaptic like behavior, therefore, neuromorphic computing systems provide excellent trade-off between real-time processing, power consumption, and overall accuracy.

In this chapter, we propose a real-time cardiac arrhythmia classification using memristor neuromorphic computing system for classifying five different beat types including: normal (N), Premature Atrial Contraction (APC), Premature Ventricular Contraction (PVC), Right Bundle Branch Block (RBBB), and Left Bundle Branch Block (LBBB). Raw ECG data is directly used as inputs to the proposed system which minimizes the amount of computations needed for feature extraction and hence reducing classification time and power consumption. The proposed system achieves an overall accuracy of 96.17% and requires 34 $ms$ to test one ECG beat, which outperforms most of its counterparts. To our best knowledge, such a system has never been proposed before for cardiac arrhythmia classification.

The rest of the chapter is organized as follows: Section 5.2 introduces the theory behind neuromorphic computing systems. Section 5.3 explains the details of the proposed system and the training procedure. Section 5.4 evaluates and discusses the proposed system from different aspects. finally, conclusion will be provided in Section 5.5.

## 5.2 Preliminary

### 5.2.1 Memristor Devices and Model Used

Memristors are considered as the fourth circuit element along with resistors, capacitors, and inductors. They were first postulated by Leon Chua in 1971 [35]. In 2008, a team in HP labs fabricated a two-terminal devices with the same behavior that Chua predicted before [21]. Nowadays, memristors are fabricated as an oxide layer sandwiched between two conductive terminals. Among all the oxide materials used to fabricate the memristors, the ones made using $TiO_{2-x}$ [55] or a layer composed of $HfO_x$ and $AlO_x$ [49] appear to be the most promising. In this work, we will adopt the device model made with $HfO_x$ and $AlO_x$ [49], as they have a high resistance ratio of ($R_{OFF}/R_{ON}\approx1000$) and ($R_{ON}\approx10K\Omega$), which is crucial for an energy efficient system, as the one proposed here. Moreover, each memristor cell is combined with another device, called selector, to enhance selectivity. From the different devices used as selectors [4, 5], we will be using the 1-Transistor 1-Memristor (1T1M) structure, as it is reliable, easy to fabricate, and has a negligible effect on the overall conductance of the cell [5]. Each cell of the crossbar array is composed of 1T1M structure, where the conductance range of that structure is [1μS, 20μS] [5]. Only 8 pre-chosen discrete conductance levels in this range are permissible during training.

### 5.2.2 Feed-Forward Neural Network Crossbar Arrays

Fig. 28 depicts the resemblance between feed-forward neural networks and crossbar arrays. A simple feed-forward neural network is mapped to a memristor crossbar array as follows: The synaptic weights, $w_{ij}$, connecting the input (purple circles) and output (red circles) neurons in Fig. 28(a), are mapped to the conductance, $G$, of the 1T1M cells at each cross point of the array in Fig. 28(b). The output neurons (and hidden neurons, if any) perform two important functions (i) evaluating the weighted sum of the inputs, $z_j = \sum_{i=1}^{N} x_i w_{ij}$, and (ii) generating the output according to a nonlinear activation function, $y_j = f(z_j)$. The weighted summation function is implemented directly using the crossbar array as follows: assuming that each vertical red (bit) line is connected virtually to ground (which is done using the summation amplifier circuit , yellow block in Fig. 28(b)), the inputs, $x$, are encoded into voltage signals, $V$, and applied to the horizontal purple (word) lines. These voltage signals are multiplied by the conductance, $G$, resulting in current $I = GV$ injected in each bit line. The second function is implemented using the summation amplifier I-V characteristics, as will be shown in Section 5.3.1.

Synaptic weights can be any positive or negative value, therefore, one synapse is represented by two 1T1M cells, where $G_{ij} \equiv G_{ij}{}^{+}$- $G_{ij}{}^{-}$, as shown in Fig. 28(b). In such a scheme, each output neuron is composed of two columns, one for $G^{+}$ and one for $G^{-}$, and currents from these columns, $I^{+}$ and $I^{-}$ are directed to the positive and negative ports of the summation amplifier, respectively. Moreover, the number of permissible conductance levels now are exponentially expanded to 57 levels (all the $8^2$ possible combinations of two 1T1M cells, dropping out the 7 repeated combinations of $G_{ij} = 0$).

## 5.3 Proposed Architecture and Design Procedure

In this section, we will illustrate the proposed system architecture and the training and programming procedure utilized to prepare the system. The information in this section provide the basic foundation for the evaluation carried out in Section 5.4.

### 5.3.1 Two-Layers Memristor Crossbar Arrays

Shown in Fig. 29 is the proposed system and its memristor crossbar array implementation. All the corresponding elements between the feed-forward neural network diagram and

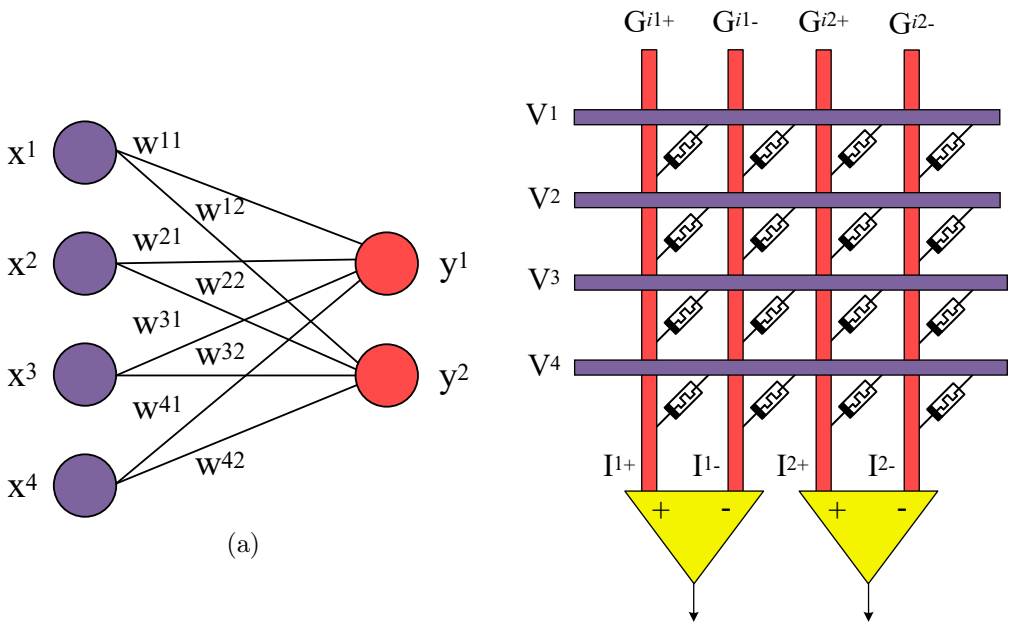



Figure 28: Neural network implementation using memristor crossbar arrays. (a) shows a conventional diagram for 4×2 feed-forward neural network while (b) shows the memristor crossbar array implementation of that network.

the hardware implementation are color matched. Crossbar layer 1 (2) corresponds to the synaptic weights connecting the input and hidden neurons (hidden and output neurons). The black and gray dots at each intersections are the 1T1M cells holding the synaptic weights, as their conductance values, of the neural network.

After the system is trained and the cells are programmed with the desired conductances (as will be described in Section 5.3.2), it is ready for testing. ECG test signals, corresponding to raw data, are normalized and mapped to the allowable crossbar voltage range [-1V, 1V]. Signals are applied horizontally, via the purple lines, multiplied by the corresponding conductances, and then currents are collected vertically, at the inputs of the summation amplifiers through the green lines.

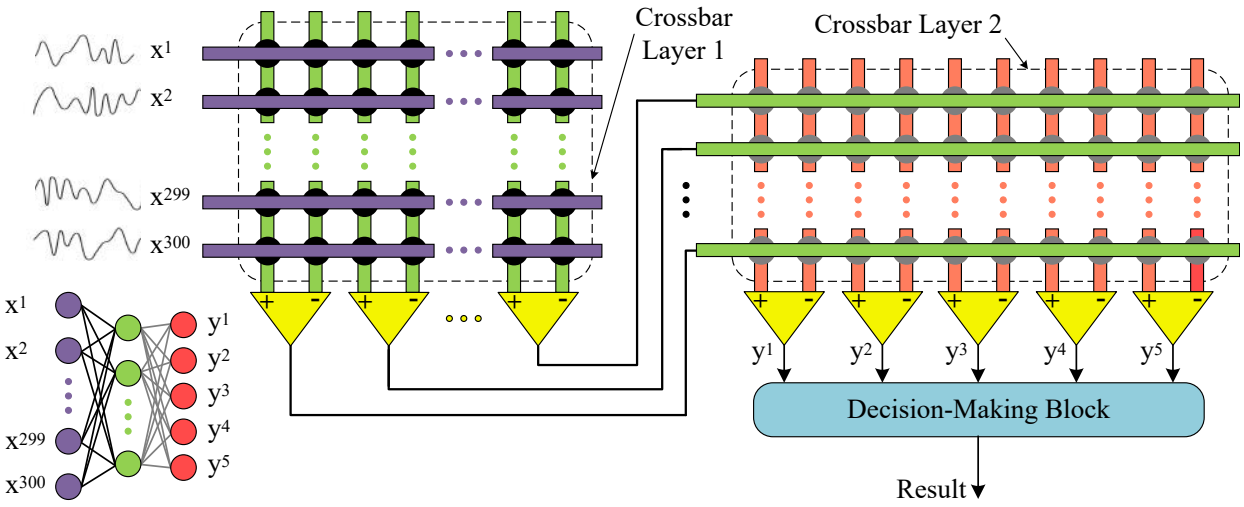


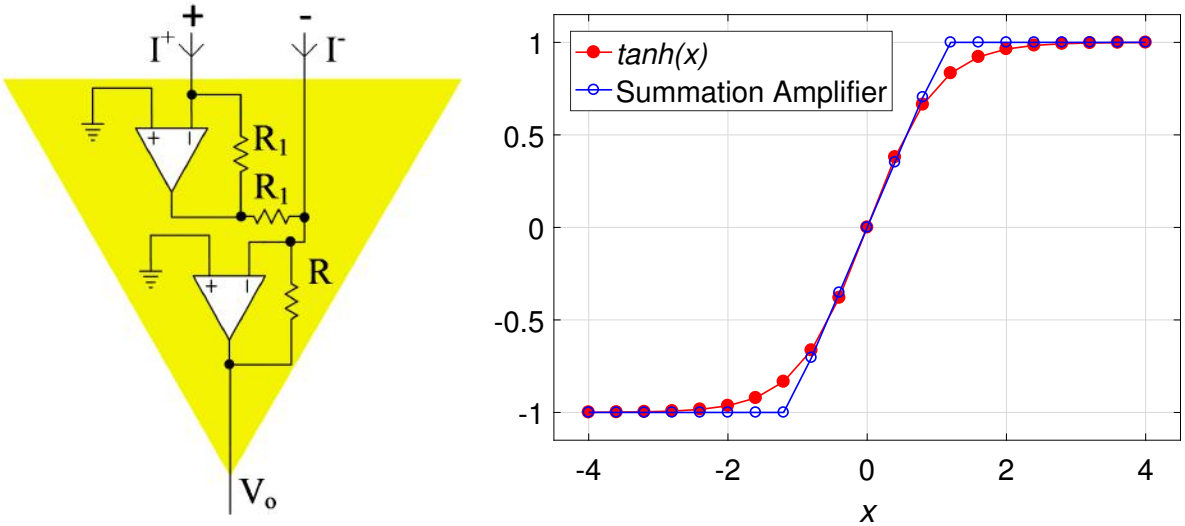

Figure 29: Hardware implementation of the proposed two-layers memristor crossbar array. A 300×210×5 feed-forward neural network diagram is shown in the lower left corner, and its circuit implementation is shown in the rest of the figure.

The circuit diagram of the summation amplifier is shown in Fig. 30(a). Two operational amplifiers, connected in a negative feedback configuration, are used. The summation amplifier implements the following equation:

$$V_o(t) = R\left[I^{+}(t) - I^{-}(t)\right] \tag{5.1}$$

where $I^{+}(t)$ and $I^{-}(t)$ are the +ve and -ve current signal from the crossbar array, $R$ is the resistance value, and $V_o(t)$ is the output voltage signal.

As mentioned before, the summation amplifier I-V characteristics is used to implement the neuron activation function. Fig. 30(b) depicts the amplifier I-V characteristics (blue circled curve) versus the I-O relation of *tanh(x)* function (red dotted curve). As can be told,




(a) (b)

Figure 30: Summation amplifier (yellow block) in Fig. 29 (a) shows the circuit implementation of the amplifier and (b) depicts the amplifier IV characteristics (blue curve) versus *tanh(x)* characteristics (red curve).

the summation amplifier I-V characteristics closely resembles the I-O relation of *tanh(x)* function, under the normal range of operation, hence it is safe to use the amplifier characteristics as an activation function. It should be mentioned here that the summation amplifier virtually connect the horizontal lines to ground, which ensures the correct operation of the crossbar array.

After the signals have been collected from crossbar layer 1, they are applied to crossbar layer 2 (in the same way as illustrated in crossbar layer 1). Afterwards, the 5 output signals representing the 5 possible beat types are applied to the decision-making block to give system prediction of the corresponding class.

### 5.3.2 Training Procedure

To be able to utilize the previously discussed system, training on the desired dataset and programming of the 1T1M cells have to be done first. The system training part can be carried out on any machine learning software tool. In our case, a custom version of the MATLAB Neural Network toolbox, specifically modified to train the system using the permissible conductance levels only, is used. Training is done using gradient descent momentum with adaptive learning rate.

After training, the obtained synaptic weights values are programmed to their corresponding 1T1M cells. Now the system is ready for testing, which will be discussed in Section 5.4.

Table 4: Records used to test the proposed system.

| Type | Count | Records |
|---|---|---|
| N | 1500 | 100, 101, 105, 106, 114, 116, 200, 209, 233, 234 |
| APC | 1317 | 100, 118, 202, 209, 220, 222, 232 |
| PVC | 1274 | 106, 116, 119, 200, 203, 208, 213, 221, 223, 233 |
| LBBB | 1131 | 109, 111, 207, 214 |
| RBBB | 1037 | 118, 124, 212, 231, 232 |

### 5.4 System Evaluation

The dataset used is composed of 6258 samples, collected form MIT-BIH arrhythmia database [82] and shown in Table 4. It is randomly divided into 4380 (70%) training samples, 939 (15%) validation samples, and 939 (15%) testing samples. The input signals are normalized and mapped to the voltage range [-1V, 1V]. The metric used in evaluating the system is the misclassification error of the 5 beat types. All simulations were done using MATLAB on a machine having Intel Core i7 Processor running at 2.9 GHz and 8 GB of RAM.

The proposed system achieved average PVC, LBBB, APC, RBBB sensitivities of 98.98%, 98.68%, 97.09%, 91.62% respectively, as well as 97.70% specificity. Compared to most of the methods listed in Table 5, the proposed system obtained higher accuracy of 96.17%, although the raw data is used directly in the classification process without the computation of any features. In terms of testing time, our proposed system requires 34 ms to process one ECG

beat, based on memristor access time data in [83], compared to 0.32, 0.37, and 0.43 s required by Gutiérrez-Gnecchi *et al.* [75], Khalaf *et al.* [74], and Sayed *et al.* [73] respectively. Note that higher accuracy can be obtained by applying any of the feature extraction methods, in Table 5. However, such pre-processing will increase testing time and hardware overhead. Inspired by these results, we introduce our system as an efficient hardware solution for designing a real-time arrhythmia classification module that can be implemented in patient monitors and wearable ECG devices.

Table 5: Proposed method in comparison with other studies in literature.

| Method | Features | Classification | Classes | Accuracy |
|---|---|---|---|---|
| 2009 [78] | Qualitative Features | Fuzzy Logic | 5* | 93.78% |
| 2012 [84] | Qualitative Features | Cluster Analysis | 5* | 94.30% |
| 2013 [79] | Higher Order Spectra | SVM | 5* | 93.48% |
| 2013 [85] | Qualitative Features | Hidden Markov Model | 5* | 89.25% |
| 2013 [86] | Higher Order Cumulants | Neural Networks | 5* | 94.52% |
| 2013 [87] | Linear Predictive Coefficients | Probabilistic Neural Networks | 4 | 92.90% |
| 2015 [73] | Nonlinear Modeling Features | KNN | 5* | 98.70% |
| 2015 [74] | Spectral Correlaion | SVM | 5* | 98.60% |
| 2017 [75] | Wavelet Transform | Probabilistic Neural Networks | 8 | 92.75% |
| **Proposed** | **Raw Data** | **Neuromorphic Computing** | **5*** | **96.17%** |

* The 5-class studies [73, 74, 78, 79, 84–86] include the same beat types used in this paper.

### 5.4.1 Memristor Process Variations

During the programming process, the programmed conductance values differ a little bit from the desired conductance values. This problem happens due to process variations during the fabrication of memristors [59]. It is widely accepted to assume that these variations follow a Gaussian distribution [59]. In order to evaluate the proposed system against these variations, the programmed conductance value will be assumed as:

$$G_p = G_d + \nabla G, \tag{5.2}$$

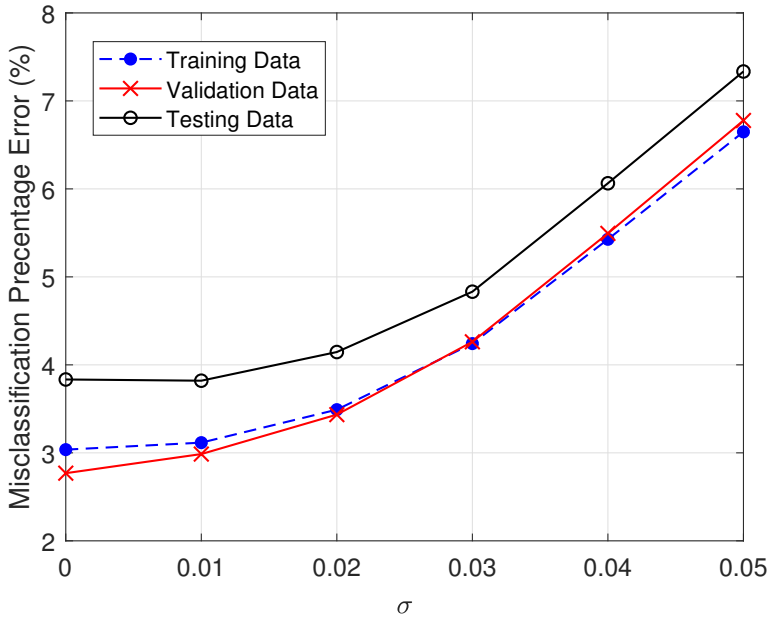



Figure 31: Misclassification error, after taking into account memristor process variations, for the proposed system architecture.

where $G_p$ is the final programmed conductance, $G_{\mathrm{d}}$ is the desired conductance value, and $\nabla G \sim N(0, \sigma)$ is a random variable that follows a Gaussian distribution of mean value 0 and standard deviation $\sigma$. Fig. 31 shows the misclassification error versus different value of $\sigma$. The maximum value of $\sigma$ is restricted to 5% which is widely accepted [59]. For each $\sigma$, the misclassification error is calculated and averaged over 100 runs. As can be told from the curves, the worst case scenario, $\sigma = 5$, only worsen the misclassification error by 3.5%. Such results are promising since they show that the proposed arrhythmia classification system is robust and can perform efficiently in non-ideal conditions.

## 5.5 Conclusions

Early and accurate detection of cardiac arrhythmias could help reduce mortality rate worldwide. Therefore, having a real-time, low power, accurate, and compact devices is a must. In this work, we proposed real-time cardiac arrhythmia classification using memristor neuromorphic computing system for classifying five different beat types. The system shows excellent performance, where it achieved overall accuracy of 96.17%, although raw ECG data were used directly without any feature calculation. Moreover, the system only needs 34 ms to test of ECG signal, which is about 10 times less compared to the fastest DSP classification systems. such a result makes it perfect for patient monitors and wearable ECG devices. In addition, the system showed good immunity against memristor process variations.

Future work will be to investigate spiking-based self-trained neuromorphic computing systems [61] as a more generic real-time cardiac arrhythmia classification systems.

# 6.0 Conclusion and Future Work

As can be concluded from the results discussed in the previous chapters, neuromorphic systems have proven a great potential for implementing low power, small size, and high accuracy systems, representing different types of neural networks. The main focus of this dissertation is to enhance neuromorphic computing systems implementation, targeting classification and generation applications. Hence, perhaps the next step is to investigate the candidacy of neuromorphic systems in implementing Deep Neural Networks (DNNs) architectures [88]. Compared to early attempts on neural networks [89, 90], modern deep learning architectures introduce more hidden layers with complex structures and nonlinear transformations to model a high-level of data abstraction [91]. In fact, many researchers have already attempted to implement DNNs using neuromorphic systems for various applications [92,93]. Moreover, deep learning systems have demonstrated fascinating performance in some real-world applications and achieved near, or even beyond, human-level accuracy in speech recognition [94], computer vision [95, 96], and a variety of other applications. Recently, natural video prediction has become one of the hottest topics in deep learning, for its wide applicability in the area of autonomous driving. However, the generation of a real-world temporal consistent video frame is still a big challenge due to the higher dimensions involved.

In the following sections, we will give an overview of a recent trend in deep learning, called Generative Adversarial Networks (GANs) [97, 98], and how it is applied to the problem of video prediction [1]. In addition, we propose a new GAN based on the previously demonstrated ESN, we call it ESN-GAN, which we believe can be easily and effectively im-

plemented using neuromorphic systems. It's worth mentioning here, that this system is more of a promising idea and the results presented here are inconclusive, yet.

## 6.1 Video Prediction using GAN

In 2014, Ian Goodfellow [98] proposed a new method of training and generating examples called Generative Adversarial Networks. GANs tasks are framed in some sort of a competition between two network models, where the first network, the *Generator*, tries to create synthetic samples of a certain dataset, while the second network, the *Discriminator*, tries to classify if the given sample is real (i.e. from the real dataset), or synthetic. As can be told from this sort of rivalry, the discriminator is always looking for differences that allow it to pin down a sample as synthetic. Consequently, the generator modifies its parameters in order to minimize those differences as much as possible for the upcoming generated samples. That cycle keeps going between the generator and the discriminator, in the training phase, so that both of them get better in their task. Theoretically, after enough reciprocating between the two networks, the generator will be able to make synthetic samples good enough to fool the discriminator into classifying them as real samples. This mechanism is illustrated in Fig. 32.

Recently, Mathieu *et al.* [1] proposed a novel GAN, based on Convolutional Neural Networks (CNNs), combined with a novel loss function that dramatically improves the sharpness of the predicted frames. They also combined multiple scales of each frame linearly as in the Laplacian pyramid reconstruction process [99, 100]. In addition, the use of multiple scales help in keeping long-range dependencies and avoid loss of resolution, which is a common problem of having pooling/sampling layer in CNNs [1].

Consider a sequence of consecutive video frames $X^1$, $X^2$, ..., $X^n$, now we want to accurately predict the subsequent video frames $Y^1$, $Y^2$, ..., $Y^m$. As proposed in [1], the GAN would consist of a generator, *Gen*, and a discriminator, *Disc*, where both are based on a CNN. Both *Gen* and *Disc* are trained alternately in parallel, where *Gen* is trained to predict one or multiple concatenated frames $Y$, while *Disc* is trained to tell the source of those frames. Such a training method assures that *Gen* can eventually generate temporally coherent sequence. Training is done using Stochastic Gradient Descent (SGD) on one network, while keeping the weights of the other network fixed.

Further more, the multiple scales architecture based on the Laplacian pyramid mentioned before is shown in Fig. 33. Let $s_1$, ..., $s_{N_{scales}}$ define the sizes of the frame inputs to the network. The studied network is composed of four scales defined as follow: $s_1 = 4\times4$, $s_2 = 8\times8$, $s_3 = 16\times16$, and $s_4 = 32\times32$. $u_{k+1}$ is an upscaling operator to size $s_{k+1}$. Finally, $X^i_{k+1}$ and $Y^i_{k+1}$ are the downscaled version of $X^i$ and $Y^i$ to size $s_{k+1}$, and $Gen'_{k+1}$ is a network

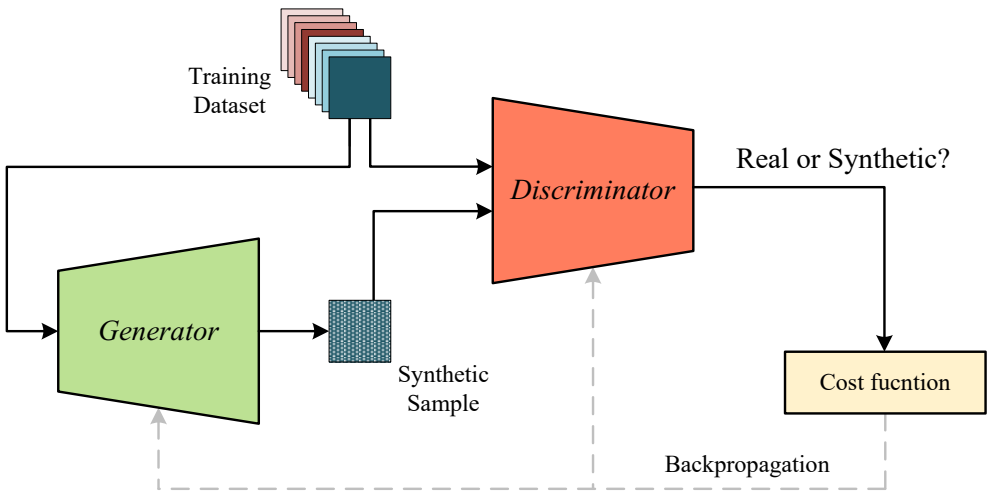



Figure 32: Illustration of the working mechanism of GAN.

that learns to predict $Y_{k+1} - u_{k+1}(Y_k)$ from $X_{k+1}$ and a rough guess of $Y_{k+1}$. The network $Gen_{k+1}$, which makes a prediction $\hat{Y}_{k+1}$ is defined as:

$$\hat{Y}_{k+1} \equiv Gen_{k+1}(X) = u_{k+1}\left(\hat{Y}_k\right) + Gen'_{k+1}\left(X_{k+1}, u_{k+1}\left(\hat{Y}_k\right)\right) \tag{6.1}$$

So the generator network starts to make a series of frame predictions, starting with the lowest resolution, $s_1$, all the way to up to highest resolution, $s_4$, which is now ready to be tested by the discriminator network.

The discriminator, *Disc*, is also a multiple scales network (same as above) with one scalar output [1]. *Disc* takes a sequence of consecutive frames and is trained to tell if the last frames are real or synthetic. It's worth mentioning here that only the last frames can either be real or synthetic, the first frames are always real.

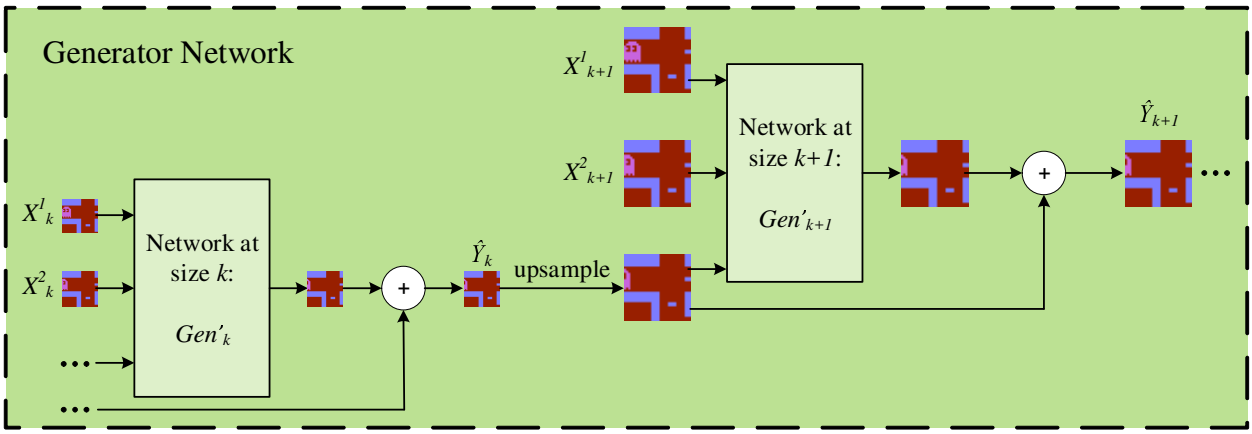



Figure 33: Multiple scales architecture proposed in [1].

### 6.1.1 Training *Disc*

For a sample $(X_k,\ Y_k)$ from the dataset, *Disc* is trained to classify $(X_k,\ Y_k)$ into class 1 and $(X_k,\ Gen_k(X))$ into class 0, at each scale, $s_k$. This is done by performing one SGD step while keeping all the weights of *Gen* fixed. The loss function for training *Disc* is:

$$\mathcal{L}_{adv}^{Disc}(X,Y) = \sum_{k=1}^{N_{scales}} L_{bce}\left(Disc_k\left(X_k,Y_k\right),1\right) + L_{bce}\left(Disc_k\left(X_k,\hat{Y}_k\right),0\right) \tag{6.2}$$

where $L_{bce}$ is the binary cross entropy loss, such that:

$$L_{bce}\left(\hat{Y},Y\right) = -\sum_i Y_i log\left(\hat{Y}_i\right) + \left(1-Y_i\right) log\left(1-\hat{Y}_i\right) \tag{6.3}$$

where $\hat{Y}_i \in [0,1]$ and $Y_i \in \{0,1\}$.

### 6.1.2 Training *Gen*

For a different sample $(X_k,\ Y_k)$ from the dataset, *Gen* is trained to predict the a set of frames $\hat{Y}_k$ as close as possible to $Y_k$. This is done by performing one SGD step while keeping all the weights of *Disc* fixed. The loss function for training *Gen* is a little bit complicated than the one used in Disc and is combined of three different loss functions, as follows:

1. *Adversarial Loss:* which means that *Gen* is trying to confuse *Disc* as much as possible by minimizing the following function:

$$\mathcal{L}_{adv}^{Gen}(X,Y) = \sum_{k=1}^{N_{scales}} L_{bce}\left(Disc_k\left(X_k,\hat{Y}_k\right),1\right) \tag{6.4}$$

The reason for not using this loss function only is that it might lead to system instability [1]. Hence, Mathieu *et al.* introduced the following loss functions.

2. *Image Gradient Difference Loss: (GDL)*: is another way to increase the sharpness of the predicted image by penalizing the difference of the gradient predictions between the real image, $Y$, and the synthetic prediction, $\hat{Y}$. This loss function is defined as:

$$\begin{aligned} \mathcal{L}_{gdl}\left(X,Y\right) &= L_{gdl}\left(Y,\hat{Y}\right) = \sum_{i,j}\left|\left|Y_{i,j}-Y_{i,j-1}\right|-\left|\hat{Y}_{i,j}-\hat{Y}_{i,j-1}\right|\right|^{\beta}+ \\ &\quad \left|\left|Y_{i-1,j}-Y_{i-1,j}\right|-\left|\hat{Y}_{i-1,j}-\hat{Y}_{i-1,j}\right|\right|^{\beta} \end{aligned} \tag{6.5}$$

where $\beta \geq 1$. It can be though of as the difference between the change in intensity from pixel to pixel in the real and predicted frames.

3. *Normal Loss:* which is the first or second norm between $X$ and $\hat{Y}$, as follows:

$$\mathcal{L}_{p}\left(X,Y\right) = L_{p}\left(Y,\hat{Y}\right) = \left\|Y-\hat{Y}\right\|^{p} \tag{6.6}$$

where $p \in \{1,2\}$. This is like minimizing the distance between the predicted and the real frames.

Now, the combined *Gen* loss is:

$$\mathcal{L}^{Gen}\left(X,Y\right) = \lambda_{adv}\mathcal{L}_{adv}^{Gen}\left(X,Y\right)+\lambda_{p}\mathcal{L}_{p}\left(X,Y\right)+\lambda_{gdl}\mathcal{L}_{gdl}\left(X,Y\right) \tag{6.7}$$

where $\lambda_{adv}$, $\lambda_{p}$, and $\lambda_{gdl}$ are fine tuning parameters.

## 6.2 Video Prediction using ESN-GAN

ESN-GAN is based on the GAN system we discussed above, however, instead of using a CNN generator, we use an ESN generator. The motivation behind this was the excellent results mentioned in Chapter 4, where the ESN implemented successfully captured the temporal information in the dataset used. In addition, we only need to train the memristor crossbar layer connecting the reservoir and the output nodes, which, when compared to complex and time consuming training process of CNN, makes it much feasible to realize ESN-GAN using neuromorphic computing system.

Fig. 34 depicts the proposed ESN-GAN. The *Disc* is the same as the one proposed in [1], while Gen is replaced by a multiple scale ESNs. All the previously discussed details about training both the *Disc* and *Gen* are applied for the proposed system too.

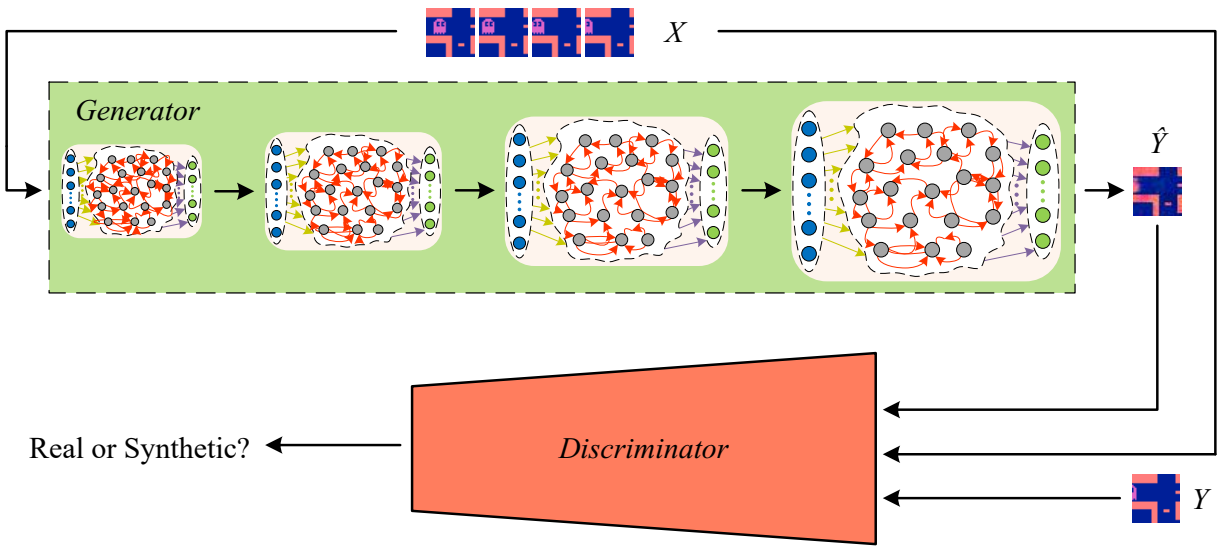



Figure 34: ESN-GAN architecture.

## 6.3 Experiments and Results

We provide here a comparison quantitative and qualitative comparison between results obtained from the CNN-GAN proposed in [1] and ESN-GAN.

### 6.3.1 Training Details

Both systems were trained used a four concatenated sequential frames and predict the next frame (as depicted in Fig. 34). Table 6 shows the values of the parameters used in training the systems.

Fig. 35 depicts a comparison between the training loss for both the generator and the discriminator of ESN-GAN and CNN-GAN. For *Disc* loss, the higher the value the better result, because it means the discriminator is confused and cannot differentiate between real and synthetic frames. On the contrary, the lesser the value of *Gen* loss the better the result, because it means that the synthetic frames are very close to the real ones. As can be told from Fig. 35, CNN-GAN has an advantage over ESN-GAN, however, ESN-GAN converges faster than CNN-GAN, as can be deduced from Fig. 35(a).

Table 6: Different parameters used in training both systems.

| $\beta$ | $p$ | $\lambda_{adv}$ | $\lambda_p$ | $\lambda_{gdl}$ | *Gen* Learning Rate | *Disc* Learning Rate | Epochs | Batch Size |
|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | 1 | 1 | 0.00004 | 0.02 | $\sim$ 450K | 8 |

### 6.3.2 Quantitative Metrics

We use two metrics, as defined in [1], to evaluate the similarity between the synthetic frames, $\hat{Y}$, and real ones, $Y$. The first metric is the Peak Signal to Noise Ratio (PSNR) defined as:

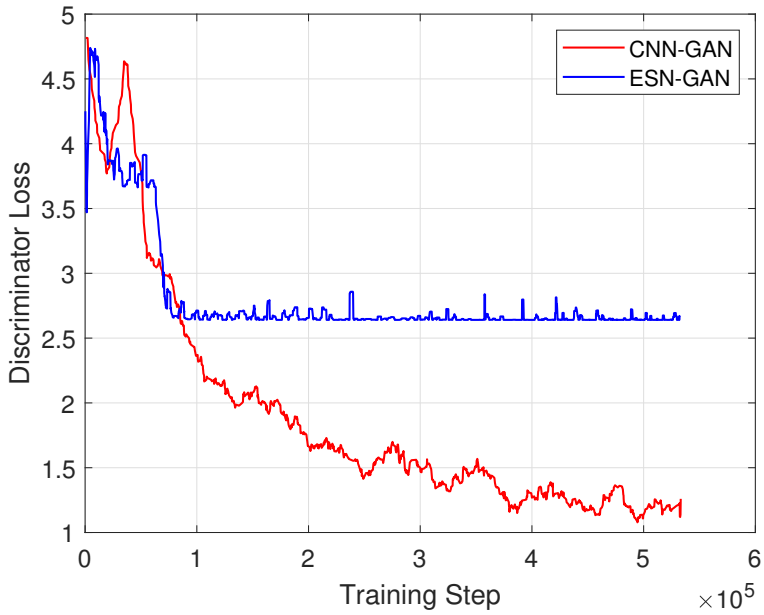
$$PSNR\left(Y,\hat{Y}\right) = 10log_{10}\frac{max_{\hat{Y}}^2}{\frac{1}{N}\sum_{i=0}^{N}(Y-\hat{Y})^2} \tag{6.8}$$

where $max_{\hat{Y}}$ is the maximum possible value of $\hat{Y}$ intensities.

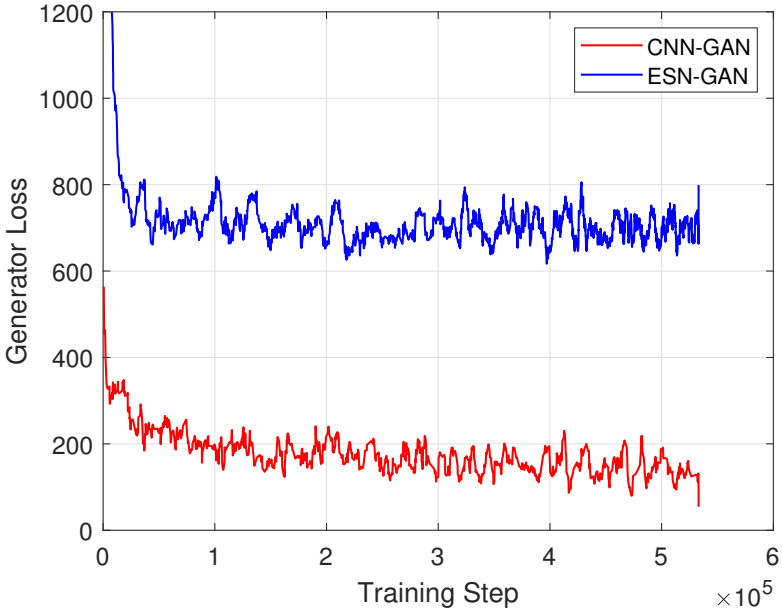
The second metric is the Sharpness Difference (SD), which measure the loss of sharpness between $Y$ and $\hat{Y}$. This metric is defined as:

Table 7: Comparison between GAN [1] and ESN-GAN architecture.

| | | | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|---|---|
| GAN | *Gen* | Feat. maps | 128, 256, 128 | 128, 256, 128 | 128, 256, 512, 256, 128 | 128, 256, 512, 256, 128 |
| | | Kernel sizes | 3, 3, 3, 3 | 5, 3, ,3 ,5 | 5, 3, 3, 3, 3, 5 | 7, 5, 5, 5, 5, 7 |
| ESN-GAN | | Res. sizes | 256 | 512 | 1024 | 2048 |
| Both | *Disc* | Feat. maps | 64 | 64, 128, 128 | 128, 256, 256 | 128, 256, 512, 256 |
| | | Kernel sizes | 3 | 3, 3, 3 | 5, 5, 5 | 7, 7, 5, 5 |
| | | FC sizes | 512, 256 | 1024, 512 | 1024, 512 | 1024, 512 |




(a)




(b)

Figure 35: Comparison between the training loss functions of (a) *Disc* and (b) *Gen* networks for both ESN-GAN (blue) and CNN-GAN (red).

$$SD\left(Y,\hat{Y}\right) = 10log_{10}\frac{max_{\hat{Y}}^2}{\frac{1}{N}\left[\sum_i\sum_j\left|(\nabla_i Y + \nabla_j Y) - (\nabla_i \hat{Y} + \nabla_j \hat{Y})\right|\right]} \tag{6.9}$$

where $\nabla_j Y = |Y_{i,j} - Y_{i,j-1}|$ and $\nabla_i Y = |Y_{i,j} - Y_{i-1,j}|$.

For both these metric the higher value the better. Fig. 36 depicts a comparison between PSNR and SD for both ESN-GAN and CNN-GAN. Although at first glance, these results are not in our favor, our ESN-GAN converges faster than CNN-GAN.

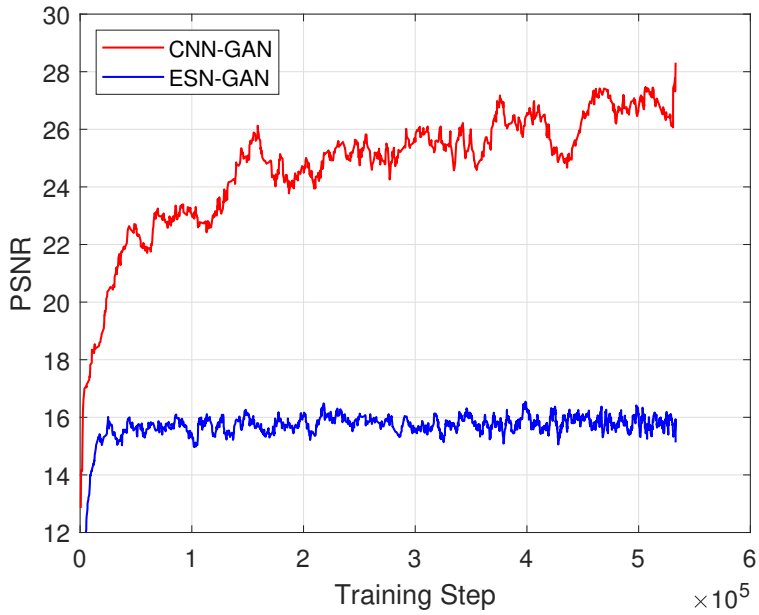
### 6.3.3 Qualitative Evaluation

Fig. 37 depicts a sample batch of 8 different *(X, Y)* pairs used in training ESN-GAN. As can be seen, the predicted frame, $\hat{Y}$, is not visually completely far away from the real frame, $Y$, but it can certainly be improved.

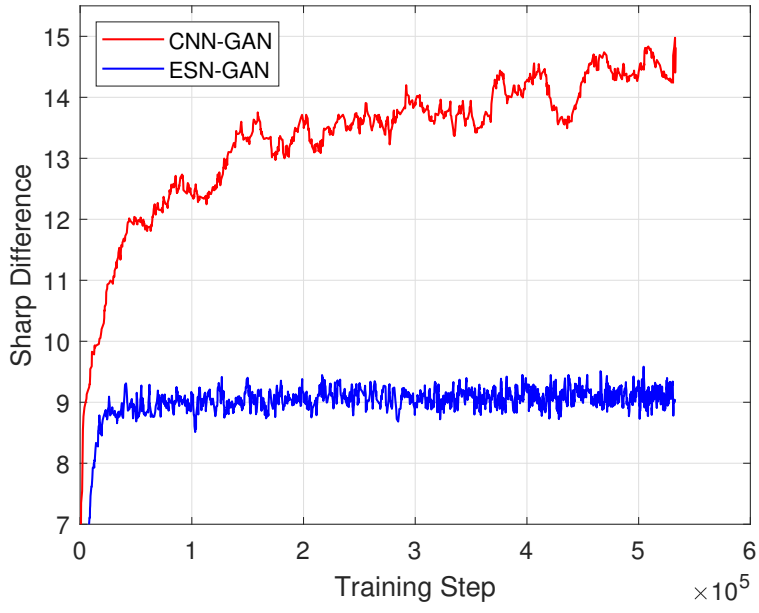
## 6.4 Future Work

In this chapter, we have proposed an ESN-GAN system for the purpose of natural video prediction. As we mentioned at the beginning of this chapter, the results here are inconclusive, as it can't be compared to the state-of-the-art results [1], yet.

However, the system can certainly be dramatically improved by exploring different loss functions, fine tuning the loss parameters, and trying different ESN reservoir sizes. The ultimate goal here is to finalize the design of ESN-GAN to achieve results as close as possible to the state-of-the-art, then start migrating this system into a neuromorphic computing one.




(a)




(b)

Figure 36: Comparison between (a) PSNR and (b) Sharp Difference for both ESN-GAN (blue) and CNN-GAN (red).

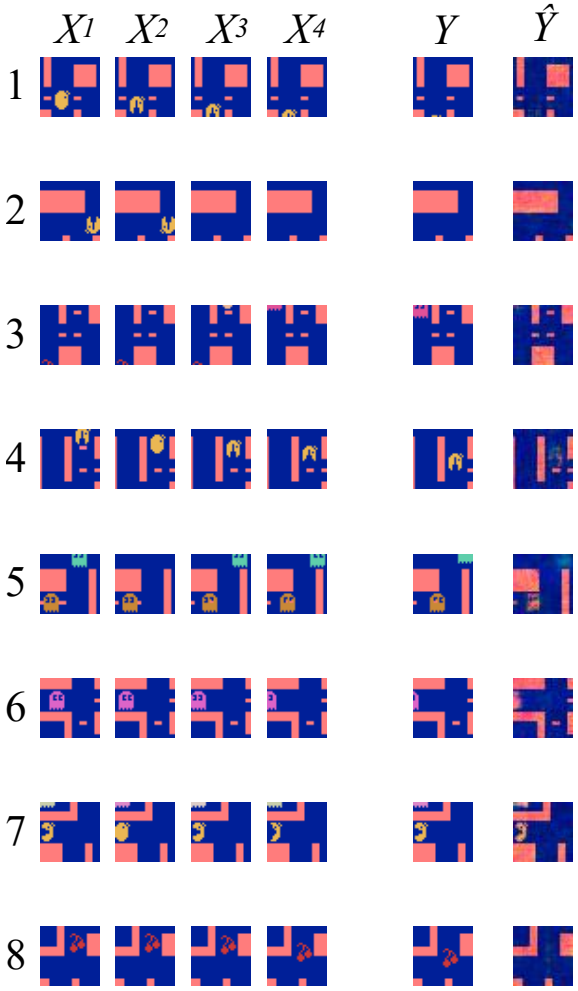



Figure 37: Sample batch showing the sequence of frames used in training, the next generated and real frame. This batch was recorded at the last training epoch.

## Bibliography


[1] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[2] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A Million Spiking-Neuron Integrated Circuit with A Scalable Communication Network and Interface. *Science*, 345:668–673, 2014.

[3] A Brain-Inspired Chip Takes to the Sky. https://www.technologyreview.com/s/532176/a-brain-inspired-chip-takes-to-the-sky/. [Online; accessed 17-May-2019].

[4] B. Yan, A. M. Mahmoud, J. J. Yang, Q. Wu, Y. Chen, and H. H. Li. A neuromorphic ASIC Design Using One-Selector-One-Memristor Crossbar. In *IEEE Int. Symp. Circuits and Syst. (ISCAS)*, pages 1390–1393, 2016.

[5] Chenchen Liu et al. A Spiking Neuromorphic Design with Resistive Crossbar. In *Design Automation Conference (DAC)*, pages 14:1–14:6, 2015.

[6] Lutz Prechelt. *"PROBEN 1: A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms"*. Univ., Fak. für Informatik, 1994.

[7] F. Masuoka, M. Asano, H. Iwahashi, T. Komuro, and S. Tanaka. A new flash e2prom cell using triple polysilicon technology. In *1984 International Electron Devices Meeting*, volume 30, pages 464–467, 1984.

[8] V. N. Kynett, A. Baker, M. L. Fandrich, G. P. Hoekstra, O. Jungroth, J. A. Kreifels, S. Wells, and M. D. Winston. An in-system reprogrammable 32 k times;8 cmos flash memory. *IEEE Journal of Solid-State Circuits*, 23(5):1157–1163, Oct 1988.

[9] H. Akinaga and H. Shima. Resistive random access memory (reram) based on metal oxides. *Proceedings of the IEEE*, 98(12):2237–2251, Dec 2010.

[10] Rainer Waser, Regina Dittmann, Georgi Staikov, and Kristof Szot. Redox-based resistive switching memories - nanoionic mechanisms, prospects, and challenges. *Advanced Materials*, 21(25-26):2632–2663, 2009.

[11] S. S. Sheu, M. F. Chang, K. F. Lin, C. W. Wu, Y. S. Chen, P. F. Chiu, C. C. Kuo, Y. S. Yang, P. C. Chiang, W. P. Lin, C. H. Lin, H. Y. Lee, P. Y. Gu, S. M. Wang, F. T. Chen, K. L. Su, C. H. Lien, K. H. Cheng, H. T. Wu, T. K. Ku, M. J. Kao, and M. J. Tsai. A 4mb embedded slc resistive-ram macro with 7.2ns read-write random-access time and 160ns mlc-access capability. In *2011 IEEE International Solid-State Circuits Conference*, pages 200–202, Feb 2011.

[12] J. Y. Wu, M. Breitwisch, S. Kim, T. H. Hsu, R. Cheek, P. Y. Du, J. Li, E. K. Lai, Y. Zhu, T. Y. Wang, H. Y. Cheng, A. Schrott, E. A. Joseph, R. Dasaka, S. Raoux, M. H. Lee, H. L. Lung, and C. Lam. A low power phase change memory using thermally confined tan/tin bottom electrode. In *2011 International Electron Devices Meeting*, pages 3.2.1–3.2.4, Dec 2011.

[13] M. Qazi, M. Clinton, S. Bartling, and A. P. Chandrakasan. A low-voltage 1mb feram in 0.13 μm cmos featuring time-to-digital sensing for expanded operating margin in scaled cmos. In *2011 IEEE International Solid-State Circuits Conference*, pages 208–210, Feb 2011.

[14] H. Shiga, D. Takashima, et al. A 1.6 gb/s ddr2 128 mb chain feram with scalable octal bitline and sensing schemes. *IEEE Journal of Solid-State Circuits*, 45(1):142–152, Jan 2010.

[15] Y. Chen, X. Wang, H. Li, H. Xi, Y. Yan, and W. Zhu. Design margin exploration of spin-transfer torque ram (stt-ram) in scaled technologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(12):1724–1734, Dec 2010.

[16] Y. Zhang, L. Zhang, W. Wen, G. Sun, and Y. Chen. Multi-level cell stt-ram: Is it realistic or just a dream? In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 526–532, Nov 2012.

[17] S. Yu and H. S. P. Wong. A phenomenological model for the reset mechanism of metal oxide rram. *IEEE Electron Device Letters*, 31(12):1455–1457, Dec 2010.

[18] Crossbar inc. ReRAM Advantages. https://www.crossbar-inc.com/en/technology/reram-advantages/. [Online; accessed 17-May-2019].

[19] W. W. Zhuang, W. Pan, B. D. Ulrich, et al. Novel colossal magnetoresistive thin film nonvolatile resistance random access memory (rram). In *Digest. International Electron Devices Meeting,*, pages 193–196, Dec 2002.

[20] I. G. Baek, M. S. Lee, S. Seo, M. J. Lee, D. H. Seo, D. S. Suh, J. C. Park, S. O. Park, H. S. Kim, I. K. Yoo, U. I. Chung, and J. T. Moon. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses. In *IEDM Technical Digest. IEEE International Electron Devices Meeting, 2004.*, pages 587–590, Dec 2004.

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

[22] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, September 1971.

[23] Y. C. Chen, H. Li, and W. Zhang. A novel peripheral circuit for rram-based lut. In *2012 IEEE International Symposium on Circuits and Systems*, pages 1811–1814, May 2012.

[24] Miao Hu, Hai Li, Qing Wu, and Garrett S. Rose. Hardware realization of bsb recall function using memristor crossbar arrays. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 498–503, New York, NY, USA, 2012. ACM.

[25] Wm A Wulf and Sally A McKee. Hitting The Memory Wall: Implications of The Obvious. *ACM SIGARCH Computer Architecture News*, 23:20–24, 1995.

[26] C. Gamrat. Challenges and Perspectives of Computer Architecture at the Nano Scale. In *IEEE Comput. Soc. Annu. Symp. VLSI*, pages 8–10, 2010.

[27] Carver Mead and Mohammed Ismail. *Analog VLSI implementation of neural systems*, volume 80. Springer Science & Business Media, 2012.

[28] Johannes Schemmel, Daniel Briiderle, Andreas Griibl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling. In *IEEE Int. Symp. Circuits and Syst. (ISCAS)*, pages 1947–1950, 2010.

[29] Srinjoy Mitra, Stefano Fusi, and Giacomo Indiveri. Real-Time Classification of Complex Patterns Using Spike-Based Learning in Neuromorphic VLSI. *IEEE Trans. Biomed. Circuits Syst.*, 3:32–42, 2009.

[30] Trafton. Mimicking the brain, in silicon, 2011.

[31] Syed M Qasim, Ahmed A Telba, and Abdulhameed Y AlMazroo. Fpga design and implementation of dense matrix-vector multiplication for image processing application. *IJCSNS*, 2010.

[32] C. R. Schlottmann, S. Shapero, S. Nease, and P. Hasler. A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing. *IEEE Journal of Solid-State Circuits*, 47(9):2174–2184, Sept 2012.

[33] Mrigank Sharad, Charles Augustine, Georgios Panagopoulos, and Kaushik Roy. Proposal for neuromorphic hardware using spin devices. *arXiv preprint arXiv:1206.3227*, 2012.

[34] J. Joshua Yang, M.-X. Zhang, John Paul Strachan, Feng Miao, Matthew D. Pickett, Ronald D. Kelley, G. Medeiros-Ribeiro, and R. Stanley Williams. High switching endurance in taox memristive devices. *Applied Physics Letters*, 97(23):232102, 2010.

[35] Leon Chua. Memristor-The Missing Circuit Element. *IEEE Trans. Circuit Theory*, 18(5):507–519, 1971.

[36] L.O. Chua and Sung Mo Kang. Memristive Devices and Systems. *Proceedings of the IEEE*, 64(2):209–223, Feb 1976.

[37] M. Di Ventra, Y.V. Pershin, and L.O. Chua. Circuit Elements With Memory: Memristors, Memcapacitors, and Meminductors. *Proceedings of the IEEE*, 97(10):1717–1724, Oct 2009.

[38] Leon O Chua. The Fourth Element. *Proceedings of the IEEE*, 100(6):1920–1927, June 2012.

[39] Mohammed Affan Zidan, Hossam Aly Hassan Fahmy, Muhammad Mustafa Hussain, and Khaled Nabil Salama. Memristor-Based Memory: The Sneak Paths Problem and Solutions. *Microelectronics Journal*, 44(2):176–183, 2013.

[40] Mohammed Affan Zidan, Ahmed M Eltawil, Fadi Kurdahi, Hossam AH Fahmy, and Khaled N Salama. Memristor Multiport Readout: A Closed-Form Solution for Sneak Paths. *IEEE Trans. Nanotechnol.*, 13:274–282, 2014.

[41] Shyh-Shyuan Sheu, Pei-Chia Chiang, Wen-Pin Lin, Heng-Yuan Lee, Pang-Shiu Chen, Yu-Sheng Chen, Tai-Yuan Wu, Frederick T Chen, Keng-Li Su, Ming-Jer Kao, et al. A 5ns fast write multi-level non-volatile 1 k bits rram memory with advance write scheme. In *2009 Symposium on VLSI Circuits*, pages 82–83. IEEE, 2009.

[42] Jiun-Jia Huang, Yi-Ming Tseng, Wun-Cheng Luo, Chung-Wei Hsu, and Tuo-Hung Hou. One selector-one resistor (1s1r) crossbar array for high-density flexible memory applications. In *2011 International Electron Devices Meeting*, pages 31–7. IEEE, 2011.

[43] Sung Hyun Jo, Tanmay Kumar, Sundar Narayanan, Wei D Lu, and Hagop Nazarian. 3d-stackable crossbar resistive memory based on field assisted superlinear threshold (fast) selector. In *2014 IEEE International Electron Devices Meeting (IEDM)*, pages 6–7. IEEE, 2014.

[44] F. Akopyan et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 34:1537–1557, 2015.

[45] ITRS. International Technology Roadmap of Semiconductors, 2013 Edition, Emerging Research Devices.

[46] Andy Thomas. Memristor-Based Neural Networks. *J. Physics D Appl. Physics*, 46:093001, 2013.

[47] 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*, 10:1297–1301, 2010.

[48] Tsung-Wen Lee and Janice H Nickel. Memristor Resistance Modulation for Analog Applications. *IEEE Electron Device Lett.*, 33:1456–1458, 2012.

[49] Shimeng Yu, Yi Wu, and H-S Philip Wong. Investigating The Switching Dynamics and Multilevel Capability of Bipolar Metal Oxide Resistive Switching Memory. *Appl. Physics Lett.*, 98:103514, 2011.

[50] Miao Hu, Hai Li, Yiran Chen, Qing Wu, Garrett S Rose, and Richard W Linderman. Memristor Crossbar-Based Neuromorphic Computing System: A Case Study. *IEEE Trans. Neural Netw. Learn. Syst.*, 25:1864–1878, 2014.

[51] Boxun Li, Yuzhi Wang, Yu Wang, Yiran Chen, and Huazhong Yang. Training Itself: Mixed-Signal Training Acceleration for Memristor-Based Neural Network. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 361–366, 2014.

[52] Raqibul Hasan and Tarek M Taha. Enabling Back Propagation Training of Memristor Crossbar Neuromorphic Processors. In *Int. Joint Conf. Neural Networks (IJCNN)*, pages 21–28, 2014.

[53] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The spinnaker project. *Proc. IEEE*, 102:652–665, 2014.

[54] M. Chu et al. Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron. *IEEE Trans. Ind. Electron.*, 62:2410–2419, 2015.

[55] Amr Mahmoud Hassan, Hossam AH Fahmy, and Nadia Hussein Rafat. Enhanced Model of Conductive Filament-Based Memristor via Including Trapezoidal Electron Tunneling Barrier Effect. *IEEE Trans. Nanotechnol.*, 15:484–491, 2016.

[56] Chenchen Liu and Hai Li. A Weighted Sensing Scheme for ReRAM-Based Cross-Point Memory Array. In *IEEE Comput. Soc. Annu. Symp. VLSI*, pages 65–70, 2014.

[57] CM Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2001.

[58] Elham Zamanidoost, Farnood M Bayat, Dmitri Strukov, and Irina Kataeva. Manhattan Rule Training for Memristive Crossbar Circuit Pattern Classifiers. In *IEEE Int. Symp. Intell. Signal Process. (WISP)*, pages 1–6, 2015.

[59] Beiye Liu et al. Digital-Assisted Noise-Eliminating Training for Memristor Crossbar-Based Analog Neuromorphic Computing Engine. In *Design Automation Conference (DAC)*, pages 7:1–7:6, 2013.

[60] Yongtae Kim, Yong Zhang, and Peng Li. A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing. *J. Emerg. Technol. Comput. Syst.*, 11(4):38:1–38:25, April 2015.

[61] © 2017 IEEE. Reprinted, with permission, fromA. M. Hassan, C. Yang, C. Liu, H. H. Li, and Y. Chen. Hybrid spiking-based multi-layered self-learning neuromorphic system based on memristor crossbar arrays. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 776–781, March 2017.

[62] Herbert Jaeger. The Echo State Approach to Analysing and Training Recurrent Neural Networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001.

[63] R. Sacchi, M. C. Ozturk, J. C. Principe, A. A. F. M. Carneiro, and I. N. da Silva. Water Inflow Forecasting using the Echo State Network: a Brazilian Case Study. In *Int. Joint Conf. Neural Networks (IJCNN)*, pages 2403–2408, 2007.

[64] Mark D. Skowronski and John G. Harris. Automatic Speech Recognition using A Predictive Echo State Network Classifier. *Neural Networks*, 20(3):414 – 423, 2007.

[65] Mantas Lukoševičius, Herbert Jaeger, and Benjamin Schrauwen. Reservoir Computing Trends. *KI-Künstliche Intelligenz*, 26(4):365–371, 2012.

[66] C. Donahue et al. Design and Analysis of Neuromemristive Echo State Networks with Limited-Precision Synapses. In *IEEE Symp. Computational Intelligence for Security and Defense Applications (CISDA)*, pages 1–6, 2015.

[67] Y. Zhang, P. Li, Y. Jin, and Y. Choe. A digital liquid state machine with biologically inspired learning and its application to speech recognition. *IEEE Transactions on Neural Networks and Learning Systems*, 26(11):2635–2649, Nov 2015.

[68] M. S. Kulkarni and C. Teuscher. Memristor-Based Reservoir Computing. In *Int. Symp. Nanoscale Arch. (NANOARCH)*, pages 226–232, 2012.

[69] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks. In *Int. Joint Conf. Neural Networks*, volume 2, pages 985–990, 2004.

[70] Michael C Mackey, Leon Glass, et al. Oscillation and Chaos in Physiological Control Systems. *Science*, 197(4300):287–289, 1977.

[71] John D Hunter and John G Milton. Amplitude and Frequency Dependence of Spike Timing: Implications for Dynamic Regulation. *Journal of Neurophysiology*, 90(1):387–394, 2003.

[72] Malcolm Thaler. *The only EKG book you'll ever need.* Lippincott Williams & Wilkins, 2017.

[73] Khaled S Sayed, Aya F Khalaf, and Yasser M Kadah. Arrhythmia classification based on novel distance series transform of phase space trajectories. In *Int. Conf. Eng. Med. Biol. Soc. (EMBC)*, pages 5195–5198. IEEE, 2015.

[74] Aya F Khalaf, Mohamed I Owis, and Inas A Yassine. A novel technique for cardiac arrhythmia classification using spectral correlation and support vector machines. *Expert Syst. Appl.*, 42(21):8361–8368, 2015.

[75] Gutiérrez-Gnecchi et al. Dsp-based arrhythmia classification using wavelet transform and probabilistic neural network. *Biomed. Signal Process. Control*, 32:44–56, 2017.

[76] Aya F Khalaf, Mohammed I Owis, and Inas A Yassine. Image features of spectral correlation function for arrhythmia classification. In *Int. Conf. Eng. Med. Biol. Soc. (EMBC)*, pages 5199–5202. IEEE, 2015.

[77] Yun-Chi Yeh, Wen-June Wang, and Che Wun Chiou. Cardiac arrhythmia diagnosis method using linear discriminant analysis on ecg signals. *Measurement*, 42(5):778–789, 2009.

[78] Yun-Chi Yeh, Wen-June Wang, and Che Wun Chiou. Heartbeat case determination using fuzzy logic method on ecg signals. *Int. J. Fuzzy Syst.*, 11(4), 2009.

[79] Roshan Joy Martis, U Rajendra Acharya, KM Mandana, Ajoy Kumar Ray, and Chandan Chakraborty. Cardiac decision making using higher order spectra. *Biomed. Signal Process. Control*, 8(2):193–203, 2013.

[80] Dongsuk Jeon et al. 24.3 an implantable 64nw ecg-monitoring mixed-signal soc for arrhythmia diagnosis. In *Int. Solid-State Circuits Conf. (ISSCC)*, pages 416–417. IEEE, 2014.

[81] © 2017 IEEE. Reprinted, with permission, fromA. M. Hassan, H. H. Li, and Y. Chen. Hardware implementation of echo state networks using memristor double crossbar arrays. In *Int. Joint Conf. Neural Netw. (IJCNN)*, pages 2171–2177. IEEE, 2017.

[82] George B Moody and Roger G Mark. The impact of the mit-bih arrhythmia database. *IEEE Eng. Med. Biol. Mag.*, 20(3):45–50, 2001.

[83] Dimin Niu, Cong Xu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. Design trade-offs for high density cross-point resistive memory. In *Int. Symposium Low Power Electron. Design*, pages 209–214. ACM/IEEE, 2012.

[84] Yun-Chi Yeh, Che Wun Chiou, and Hong-Jhih Lin. Analyzing ecg for cardiac arrhythmia using cluster analysis. *Expert Syst. Appl.*, 39(1):1000–1010, 2012.

[85] Shing-Tai Pan et al. Heartbeat recognition from ecg signals using hidden markov model with adaptive features. In *Int. Conf. Softw. Eng. Artificial Intell., Netw Parallel/Distrib. Comput. (SNPD)*, pages 586–591. IEEE, 2013.

[86] Roshan Joy Martis et al. Application of higher order cumulant features for cardiac health diagnosis using ecg signals. *Int. J. of Neural Syst.*, 23(04):1350014, 2013.

[87] Shiva Khoshnoud and Hossein Ebrahimnezhad. Classification of arrhythmias using linear predictive coefficients and probabilistic neural network. *Appl. Med. Inform.*, 33(3):55, 2013.

[88] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[89] DRGHR Williams and Geoffrey Hinton. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.

[90] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[91] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.

[92] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552. IEEE, 2017.

[93] Fan Chen, Linghao Song, and Yiran Chen. Regan: A pipelined reram-based accelerator for generative adversarial networks. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 178–183. IEEE, 2018.

[94] George Edward Dahl. *Deep learning approaches to problems in speech recognition, computational chemistry, and natural language text processing*. University of Toronto (Canada), 2015.

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

[96] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[97] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[98] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[99] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[100] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[101] © 2018 IEEE. Reprinted, with permission, from A. M. Hassan, A. F. Khalaf, K. S. Sayed, H. H. Li, and Y. Chen. Real-time cardiac arrhythmia classification using memristor neuromorphic computing system. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2567–2570. IEEE, 2018.