Memristor-based Neuromorphic Computing

Design and simulation of a Neural Network based on Memristor technology

María García Fernández

Master of Science Thesis

TU Delft — Delft University of Technology

Delft Center for Systems and Control

Memristor-based Neuromorphic Computing

Design and simulation of a Neural Network based on Memristor technology

Master of Science Thesis

For the degree of Master of Science in Embedded Systems at Delft University of Technology

María García Fernández

August 31, 2020

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) · Delft University of Technology

Copyright © Delft Center for Systems and Control (DCSC)
All rights reserved.

Table of Contents

Acknowledgements vii

1 Background 4
1-1 Machine Learning . . . 4
1-1-1 Neural Networks . . . 5
1-2 Research question . . . 11
1-3 History and Evolution of Neuromorphic Computing . . . 12
1-3-1 Hardware Implementations of Neural Networks . . . 14
1-3-2 Brief overview . . . 15

2 Introduction to memristors 17
2-1 Memristors in Neural Networks . . . 20

3 Memristor Characterization 22
3-1 Experiments and Goal . . . 22
3-1-1 Experiment Description . . . 22
3-1-2 Goal . . . 23
3-2 Brief Explanation of the Set-up . . . 24
3-2-1 Memristor's resistance calculation . . . 26
3-2-2 The memristor endurance problem . . . 29
3-3 First experiment . . . 29
3-3-1 Second experiment: From High Resistance State (HRS) to Low Resistance State (LRS) with trains of identical pulses . . . 31
3-3-2 The Mean Metastable Switch Memristor Model . . . 34
3-3-3 Model exploration via optimization runs . . . 38
3-3-4 Analysis of the data and the results . . . 41
3-4 Simulation Work . . . 44
3-5 Conclusion . . . 47

4 Memristors in Neural Networks: Introduction 48
4-1 Unsupervised Learning . . . 48
4-1-1 Unsupervised implementation of the Simplified Spiking-Time Dependant Plasticity Learning Rule . . . 49
4-1-2 Accuracy challenge . . . 52
4-2 Supervised Learning . . . 52
4-2-1 Supervised implementation of the Simplified Spiking-Time Dependant Plasticity Learning Rule . . . 52
4-2-2 Online Back-propagation Rule with Stochastic Gradient Descent . . . 53
4-2-3 Manhattan Rule-based Back-Propagation . . . 55
4-3 Summary and Design Choices . . . 55

5 Memristor Circuit Design 57
5-1 Main consideration when designing a memristor-based chip for Neural Networks . 57
5-2 Development . . . 58

6 Neural Network Simulation 63
6-1 Spiking Timing-Dependant Plasticity (STDP): Hebbian and anti-Hebbian learning 64
6-1-1 Supervised STDP . . . 64
6-1-2 Unsupervised STDP . . . 65
6-2 Classic Artificial Neural Network (ANN) Implementations . . . 66
6-2-1 Supervised ANN without hidden layers (1 input layer, 1 output layer) . . 67
6-2-2 Supervised with one hidden layer . . . 69
6-3 Conclusions . . . 72

7 Conclusions 73
7-1 Summary . . . 73
7-2 Main Contributions . . . 73
7-3 Future Work . . . 75
7-3-1 The dR/dE ratio . . . 76
7-3-2 Time series of the experiments . . . 79
7-3-3 Parameter Tuning - Matlab Code . . . 84
7-3-4 Memristor's stochasticity simulation . . . 86
7-3-5 Matrix Circuit Design . . . 87

Bibliography 88

Glossary 94
List of Acronyms . . . 94
List of Symbols . . . 95

List of Figures

1 A typical task that the human brain performs constantly. 2

1-1 Representation of a neuron [1] 6
1-2 Some examples of activation functions [2] 7
1-3 Representation of a neural network [3] 7
1-4 Spiking neuron network implementations [4] 9
1-5 Simplified version of STDP proposed in [5] 11
1-6 Simplified overview of the von Neumann Architecture 13
1-7 Percentage of papers in each field within neuromorphic computing [6] 13

2-1 The four fundamental two-terminal circuit elements[7] 17
2-2 Ideal behavior of a memristor's current and conductance with respect to voltage. 18
2-3 A series resistor can be used to measure the memristor's resistance. 19
2-4 I(V) curve. 19
2-5 Example of a crossbar architecture. 21

3-1 The experiment was added to the GUI provided by Knowm with the setup, which was open-source. 23
3-2 The experiments performed aim to help the design of a controller that works with the Self-Directed Channel (SDC) memristors given. 24
3-3 Initial Experiment Set-Up. Images provided with permission by Knowm Inc . . . 24
3-4 Circuit within the Knowm set-up. 25
3-5 How the voltage V_{READ} is selected. 27
3-6 Circuit simulated in LTSpice, as defined by Knowm. 27
3-7 Voltage and current across the memristor for a 10 μs pulse width. 28
3-8 Algorithm proposed in [8] 30

3-9 Circuit simulated in LTSpice, as defined by Knowm. 31
3-10 Histogram of the voltage applied directly on the memristor (measured via a scope) 32
3-11 Experiment 2 on one of the memristors. 33
3-12 34
3-13 Graphical explanation of the metastable switch memristor model 35
3-14 Mean Metastable Switch Memristor Model as explained in [9] and [10] 36
3-15 The predicted change is negligible for most points. 40
3-16 dX vs V_{MEM} for all pulses in which 0.5 V were applied to the circuit. 42
3-17 Real Data vs Optimization Runs Model Data 43
3-18 Model built in Simulink 44
3-19 Simulated X vs Voltage(V) and dX vs Voltage(V). 45
3-20 V_{MEM} over time. Resistance setpoints (blue) and actual read resistance (yellow) over time. 46

4-1 STDP as explained in vs the implemented version in [5]. 49
4-2 Visual explanation of the Leaky Integrate-And-Fire (LIF) neuron. 50
4-3 Behavior of the system in [5]. An output spike will only increase a synaptic weight its input is also active. Otherwise it will decrease. The input spikes are longer than the output spikes. Leakage is not explained by this plot, it is added to the simulation via external equations. 51
4-4 Scintillating Grid Illusion [11] 52
4-5 Architecture proposed in [12] 53

5-1 Inputs of the circuit designed. 58
5-2 Multiple inputs stacked together and connected an output neuron (the amplifying stage) 59
5-3 Transfer function of the amplifying stage clipped at 0.5 and -0.5 V, compared to scaled tanh and sigmoid with offset. 61
5-4 61
5-5 V_{OUT} during reading operation with 8 active inputs. 62
5-6 Voltage drop on several memristors while writing one of them. 62

6-1 Some examples of handwritten characters from the MNIST database [13] 64
6-2 Supervised Implementation of STDP. 64
6-3 Accuracy achieved by using STDP on 600 samples. 65
6-4 Unsupervised Implementation of STDP. 66
6-5 Supervised Implementation of an ANN without hidden layers. 67
6-6 Success of the experiment 68
6-7 Success of the experiment. 69
6-8 Effects of letting the network learn for too long (no early stopping) 69
6-9 Supervised implementation of an ANN with one hidden layer with 300 neurons. . 70
6-10 Success of the experiment 70

6-11 Success of the experiment 71
6-12 Weight Distribution. 72

7-1 76
7-2 76
7-3 77
7-4 77
7-5 77
7-6 78
7-7 78
7-8 78
7-9 79
7-10 79
7-11 79
7-12 80
7-13 80
7-14 81
7-15 81
7-16 82
7-17 82
7-18 82
7-19 82
7-20 83
7-21 83
7-22 83
7-23 83

List of Tables

1-1 Comparison of the usage of some neuromorphic devices 16

3-1 Numerical values thrown by both simulations 29
3-2 Memristor estimation error for Experiment 1. 38
3-3 Memristor estimation error for Experiment 2. 38
3-4 Lower and Upper bounds given to the parameters to be tuned 39
3-5 Comparison between three different optimization algorithms. They seem to give similar results. 40
3-6 Comparison between three different optimization algorithms. They give similar results. 41

Acknowledgements

I would like to thank my supervisor Joris Sijs for his assistance during the writing of this thesis, and for giving me the opportunity to learn about such an interesting topic.

I would also like to thank my family for being extremely supportive throughout my studies and for believing in me no matter what the circumstances were. Such unconditional love can only come from them, and it has given me strength on those not so easy days.

I would like to thank my friends in Spain for always receiving me with the same welcoming arms like not even a day has passed; and my friends and partner in the Netherlands for listening to my nonsensical ramblings and complaints after a long day of work, and for having the patience to wait until I become a normal person again that one can enjoy spending time with.

Finally, but not least important, I would like to thank my two current bosses at work and also one that already left: Willem, Jai and Edward. Working during my studies has been challenging, but also rewarding and heart-warming. They were always kind and understanding, and they always encouraged me to prioritize my studies.

Thank you.

Delft, University of Technology
August 31, 2020

María García Fernández

Introduction

During the whole history of Computer Science as we know it, the end goal has been to solve problems faster and more efficiently than us humans. Computers came to be to perform repetitive tasks we do not want to or do not have time to perform ourselves. At some of them, such as simple mathematical calculations, they were and they are indeed better and faster than a human being. With the appearance of computers, some effort was also put in the study of the human brain and the idea of building machines with similar principles of operation: The idea of emulating conscience has always been attractive to humankind.

The goal of creating bio-inspired solutions and ultimately a piece of hardware that could reason as a human being or even a mammal can, naturally arose within the scientific community from a very early stage of Computer Science, bringing Neuromorphic Computing into existence. Already in the 70s, Calver Mead [14] explored this idea. In 1992, his student Misha Mahowald developed a silicon-based retina [15] which modelled one of the many processes that happen during visual processing in human beings, in their retina. This chip implemented what in our retina is called “cones”, which act as light detectors, using photo-transistors and current to voltage converters. This was one of the first pieces of Neuromorphic hardware.

In those years, the low maturity level of the available silicon technology and computers in general hindered the progress and development of these efforts. Furthermore, the way in which computers work is very different to that in which the human mind functions and solves problems. The most evident example of this is the so called von Neumann bottleneck: in a computer, memory storage and arithmetical calculations happen in different hardware blocks. In other words, in order to perform a calculation the computer must first retrieve the operands from its memory. Then, another access to the memory is necessary to store the results of the calculation. Such division is not present in the human brain.

With computers becoming faster and more powerful, Neuromorphic Computing is again an active field. One of the reasons for this to happen was the appearance of Machine Learning and Neural Networks in particular, that also conceived set of bio-inspired algorithms and programming methods. Such bio-inspired algorithms are especially interesting when facing problems with a high space or time complexity or that simply cannot be worked out mathematically in a straight-forward manner. They are also relevant in cases where the problem is not explicitly defined, or it makes more sense not to go into detail about what the intrinsic

equations of the problem are.

While Machine Learning and Neural Networks have turned out to be useful for a variety of scenarios, such as image recognition or motion learning, normal computers are not the most efficient hardware for them to run on, mainly due to their highly parallel nature. The efforts that were initially put during the 70s in trying to create a human-like mind in hardware have been expanded: Neuromorphic Computing nowadays also entails designing more adequate supporting hardware for these algorithms.

This Thesis lays within the field of Neuromorphic Computing, a field which focuses on developing "brain-inspired computers, devices, and models" [6], and which also includes the search for hardware that works better with Machine Learning solutions. Even though biological beings cannot be emulated by the current Machine Learning algorithms, as many of their underlying principles are yet to be discovered, these algorithms perform well with some problems at which the human brain undoubtedly outperforms a computer. For instance, the task of recognising different objects just by looking at them is something that the human brain does continuously. However, no mathematical equation helps recognise objects.

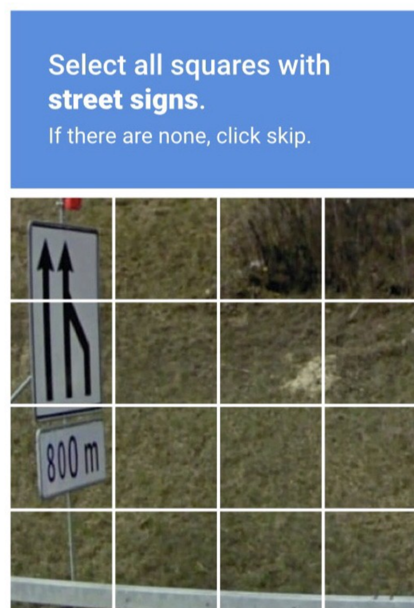


Figure 1: A typical task that the human brain performs constantly.

In biological beings, these tasks are completed with a minimum power consumption, often through a "learning process" or by extracting a pattern in the data. The human brain learns to process a large amount of data in a real-time manner and continuously extract features from it, effortlessly performing tasks such as motion learning or face recognition. The brain is capable of many things, while still occupying about 2 liters and consuming about 20 W of power [16].

In fact, the brain is not only interesting because of its learning capabilities; it also possesses the ability to reorganize itself and create new connections throughout its entire life cycle. This is known as *plasticity*: It is always learning (and forgetting), even though it ages and decays over time. Furthermore, it is also able to detect and correct errors, and even to "restore function after certain types of injury" [17]. It seems to be continuously putting efforts towards

being fully functional.

In the search for pieces of hardware that can help to achieve such functionality, memristors are often proposed. These are electrical components that were predicted in the 70s and behave like a variable resistance that holds its state (its electrical resistance) also in the absence of power. Their electrical resistance can be altered by applying trains of voltage pulses to them.

In this Thesis, a specific type of memristor is explored to understand its advantages over the current von-Neumann architectures and present neuromorphic solutions. Then, a neural network comprised by many memristors is simulated to test its usefulness, as a proof of concept.

Regardless of how promising memristor may be, it is important to keep in mind that they so far still need the aid of other processing blocks and complementary circuitry, and do not constitute a neuromorphic chip solution by themselves. The Neural Networks implemented with this work are not an exception, and they make use of additional processing blocks and Metal Oxide Semiconductor (MOS) circuitry. In any case, the usefulness of memristors as part of a hypothetical neuromorphic chip is studied. Then, it will be assessed whether memristors offer an improvement upon the existing Neuromorphic Computing frameworks.

This Thesis is structured as follows: Chapter 1 introduces the field of Machine Learning, narrowing it down to Neural Networks in relation with memristors. Then, the state-of-the-art of Neuromorphic Computing is covered. Finally, the Problem Statement is presented. Chapter 2 goes into detail about the behavior of memristors. Chapter 3 talks about an initial attempt of memristor characterization and the model used later on to simulate and predict memristor behavior. Chapter 4 explains how memristors were in a more complex simulation of different Neural Networks, and comments on results and what was necessary to achieve them. Chapter 6 provides a summary and gives some conclusions and ideas about possible future lines of work.

Chapter 1

Background

1-1 Machine Learning

Machine learning is a set of techniques and methods that in essence attempt to find patterns or structure in a set of data [3]. This is equivalent to saying that the goal is to turn a set of data into information [18].

This task of building a model for a given set of data can follow different methods depending on the nature of the data itself and on the desired output (and whether such desired output exists). Let us give a broad overview of some well-known categories.

Depending on the level of knowledge on what the output should be:

- **Unsupervised Learning:** In this category there are no input-output pairs, only a set of input data in which a structure is sought for. An example of Unsupervised Learning is **Clustering**, which refers to finding whether there are sets of so-called *clusters* in the data, or groups of points that share some similarities in one or more of their features. Some types of **Feature Extraction** algorithms are also Unsupervised. Feature extraction tries to simplify the data by finding one or more variables that can be used to describe it in a simplified manner, which also makes this data easier to store and retrieve.

  Unsupervised Learning can be used as a way of transforming and simplifying the data (*pre-processing*) before applying other types of Machine Learning on top of it.

- **Supervised Learning:** Here, a desired output exists. The data-set is normally divided in a training set, a validation set and a test set. The training set is used to find a hypothesis on how the input relates to the output and tune different parameters, mostly weights. A validation set can be used to tune the hyperparameters of the learning process. These are related, for example, to how fast it learns or how coarsely the parameters are updated in each step of the learning process. Finally, the test set is used to check the validity of the constructed hypothesis against a different part of the same data-set. Phenomena like over or under-fitting can be noticed in this phase. A properly trained neural network is able to generalise the data to such an extent where it succeeds

to fit the training set, but also different batches of the same data. The validation and test set are sometimes the same one.

A straight-forward example of Supervised Learning is **Linear Regression**, where the relation between the input and the output in the data must be modelled by a line. Linear Regression can be generalized within **Curve Fitting**, in which such input-output relation is fit by a curve of choice (polynomial, exponential, logarithmic or any other function)

- **Reinforcement Learning** This type of algorithms use a function called *reward function* that must be maximized. One way of explaining how they work is to think of them as a Markov Decision Process. Every action a makes the system go from state s to state s'. Then, a reward $R(s, s')$ can be calculated, which affects the subsequent decisions and the likelihood of every action given an initial state s.

There are of course algorithms that do not fall in these categories or can fall in more than one of them depending on how they are implemented. A hybrid between Supervised and Unsupervised Learning (Semi-supervised) exists. One example of this is **Anomaly detection**, based on the identification of outliers in data, or items that are different somehow to the rest of the data-set.

Depending on whether the data-set is static or dynamic, another classification is possible:

- **Online Learning**: The algorithm will learn as new data comes in, continuously in a stream. No division of the data is necessary. Typically an online learning algorithm will be Unsupervised, since it would otherwise need too much interaction with a human supervisor. But some mixed approaches exist [19].
- **Offline Learning**: The data-set is given and static. Dividing the data may be more appropriate.

As said before, Machine Learning creates a mathematical model of the data. In Curve Fitting, for example, the model would be the specific mathematical function that must fit the data. In some cases of Reinforcement Learning, the underlying model is a Markov Decision Process. But there are others that can be used, and they are not always purely a software abstraction. Some can be physically implemented as circuitry that acts on input signals and generates output signals. That is in fact the case of Neural Networks, which are of special interest for this Thesis.

1-1-1 Neural Networks

Neural Networks are a group of Machine Learning models that came to existence to mimic the human brain. Usually in a Neural Network (NN), the neuron would be a processing unit with one or more inputs (the **dendrites**) and one or more outputs (the **axons**). A synapse would connect every axon to the dendrite of the next neuron.

To this day there is not a unique manner of implementing the behaviour of the neurons, the synapses or for the networks. There are also different network topologies that can be used, with or without recurrence (that is, feeding an output signal as an input as well). There is

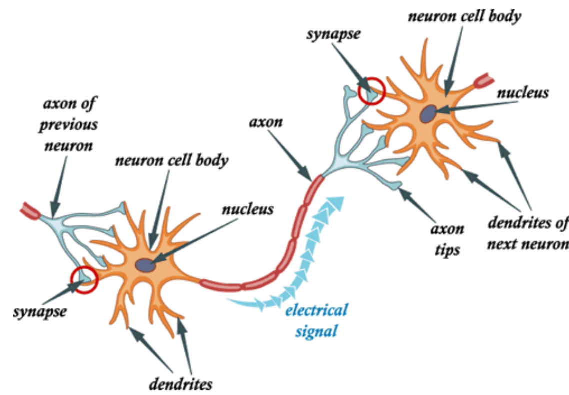


Figure 1-1: Representation of a neuron [1]

no consensus in the scientific community as to what Neural Network model is best. Rather than that, the choice is mostly related to the application at hand and the field of interest.

Although the initial intention in this field was indeed to emulate the human brain, the focus of these models soon began to diverge. Some later implementations and developments of this idea began to have a greater focus on Computer Science, some more on Neuroscience. The first ones grew apart from this original intent and would be better called "bio-inspired" [6], as in inspired by the brain but not necessarily realistic. Many of these NN are called Artificial Neural Networks, which may be more practical for hardware or software development. These efforts are all in all not so useful to Neuroscience, as they were developed with the need of a simplified, effective, more computer-oriented concept in mind. That is, the resulting solution must be able to generate the desired answers from the inputs, but it does not have to be bio-plausible.

On the other end of the spectrum (closer to Neuroscience research), there are models that attempt to copy certain features of the brain, at a level as realistic as possible. There are different types of neurons and many processes happening around them. Therefore, sometimes a specific neuron or process wants to be studied and replicated.

The focus of this thesis when it comes to Neural Networks is more on the Computer Science field, and not so much on Neuroscience. The Neural Network(s) at hand will be bio-inspired, but not bio-plausible. However, since it can be physically implemented on hardware, it will be more bio-plausible than a pure software implementation. Nevertheless, the main target of this work will still be to be able to carry out a task successfully and with a desired output, and not so much to mimic how the brain learns.

To understand the workings of Neural Networks, let us first look at a simple version of one. One of the simplest neural network models, which is also used with variations in many others, is the one designed by McCulloch-Pitts [20], where the output of a neuron follows the equation:

$$y = f(\sum_{i=0}^{n} w_{i,j} x_i) \tag{1-1}$$

Where f is what is called the activation function. Activation functions are used to simulate the firing of the neuron (whether it is activated or not, given a certain input). There is a

large amount of them with different complexities. In Fig. 1-2, a few of them are showed.

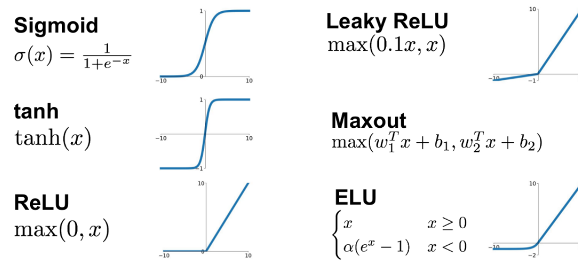


Figure 1-2: Some examples of activation functions [2]

The choice of activation function is related to the nature of the data. All of them have their weaknesses and strengths.

In general, in Artificial Neural Network (ANN) there are layers of nodes interconnected by links with a certain weight. Such nodes are called neurons, and they process input data with a certain activation function, that determines how activated that neuron is with that specific input. The weighted connections link nodes together. The weighted sum of several neuron outputs then becomes the input of another neuron in the next layer. Normally, the first layer is called “input layer”, the last one “output layer” and the ones in-between “hidden layers”. The neurons in a hidden layer can be also referred as units or hidden units, like in Fig. 1-3, and the topology of the Neural Network can be represented as a weighted graph. The output of a NN may be one or more neurons.

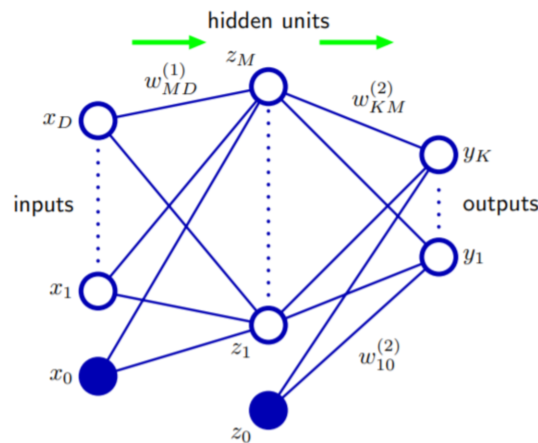


Figure 1-3: Representation of a neural network [3] with D inputs, M hidden units and K outputs. x_0 and z_0 are bias terms.

To explain how such a weighted graph representation would work, taking Figure 1-3 and ignoring for this example activation functions: For every neuron its output z_M would be:

$$z_M = x_0 \cdot w_{M0} + x_1 \cdot w_{M1} + ... + x_D \cdot w_{MD} \tag{1-2}$$

and thus effectively, increasing the number of layers or the number of neurons within one layer in the networks increases the level of complexity of the underlining model that will attempt to fit the data. For Fig. 1-3, the resulting output would be:

$$y_K = z_M \cdot w_{KM} + ... + z_0 \cdot w_{K0} = \\ = \sum_i (x_i \cdot w_{Mi}) \cdot w_{KM} + \sum_i (x_i \cdot w_{1i}) \cdot w_{K1} + ... + z_0 \cdot w_{K0} \tag{1-3}$$

Sometimes, **bias terms** are also added to the network (z_0 and x_0 in the picture). These are neurons with a constant value (there is no "firing" in them, or they can be seen as always firing) that can be added in any layer of the network. Their role is similar to that of the constant term used in a polynomial for curve fitting: it helps reduce the fitting error in cases where the shape of the function is fitted properly but there is a numerical offset. The bias term can also be seen as a way of ensuring that a neuron has a non-zero output when all of its inputs are zero.

It is also possible to add some type of **backpropagation** to the network, where the weights are readjusted based on the error of the output. Backpropagation involves calculating the gradient of a certain "loss function", which in practice is a measurement of how much error was induced into the output with the current weight configuration. A very popular choice for a loss function is the Mean Square Error (MSE).

$$MSE = \sum_i (y_i - t_i)^2 \tag{1-4}$$

Where y is the output of a neuron in the output layer, and t the desired target.

What happens then is that the gradients of this error or loss function, which will be noted here with δ, are added up following the structure of the network. For instance, a weight update for w_{MD} from 1-3 would be:

$$\Delta w_{KM} = \eta \cdot \delta_{y_k} \cdot z_M = \eta \cdot \frac{\partial MSE}{\partial w_{KM}} \tag{1-5}$$

Where η is a learning parameter.

Spiking Neural Networks

Since in ANNs the input is a digitally encoded signal or an analog signal (but in both cases a constant value), a new type of neural net was developed where the input is not a constant value but pulses or trains of them, which resembles more what happens in reality in a biological brain. They were called *Spiking Neural Network (SNN)s*, and they are sometimes referred to as "more bio-plausible" [21] than ANNs. While in regular ANNs the information is encoded in the magnitude of the signal itself, with SNNs it is encoded in the timing of the signal. Because of this characteristic, SNNs are also said to be more energy-efficient, or to use less energy for the same amount of information transfer. There is one feature that remains the same as with Artificial Neural Networks: Neurons fire only when their potential reaches a threshold, and their potential is a function of the input.

Despite of their popularity, the capabilities of Spiking Neural Networks seem to still be rather unexploited, because they work on a more complex way and it is not yet fully known how to use them and take advantage of their properties, or even how to train them [6].

There are many different types of neurons that use spikes in the brain. Similarly, a variety of bio-inspired spiking neuron models exist. There is not a preferred one in general: It is typically dependent on the application. In [4], a few of the features that a spiking neuron model can contain are explained. To summarize a few of the ones explained by Izhikevich:

- **Tonic spiking or Tonic Bursting**: The neuron fires a (train of) spike(s) when stimulated. As long as the stimulus remains active, it continues to fire (1-4).
- **Phasic spiking or Phasic Bursting**: The neuron fires a (train of) spike(s) only with rising edges. Thus, even if the stimulus is still active, it will not continue to cause firings on the neuron. That is the difference with tonic spiking.
- **Frequency adaptation**: When a high signal arrives, the neuron outputs a burst which frequency decreases over time while the signal persists.
- **Resonance**: The neuron only responds to certain input frequencies.
- **Accommodation**: The neuron is less excitable by wide pulses than by narrow ones.
- **Threshold variability**: The threshold of the neuron is dependent on the prior activity of the neuron.

Fig. 1-4 gives a more complete overview of these features.

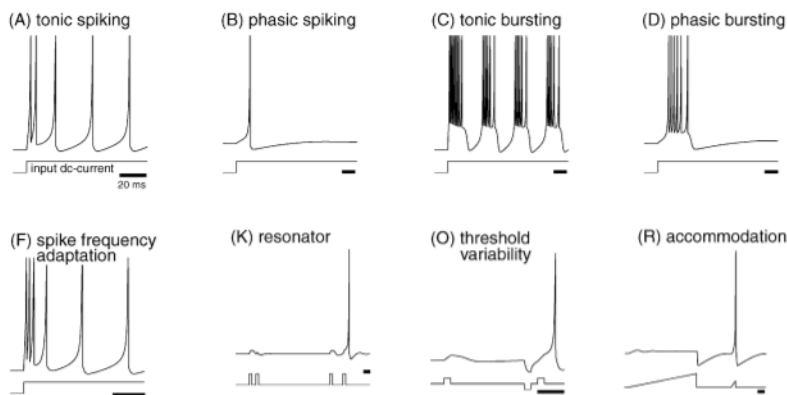


Figure 1-4: Spiking neuron network implementations [4]

Because the state of a memristor is also changed via trains of (voltage) pulses, SNN is often recommended when working with them. This will be revisited later on.

Different options that can be used for the encoding of the signal in the SNN have been presented, but it is also necessary to talk about learning rules. Even though memristors could be used just as weight storing units for a classic ANN, there is another idea which is broadly suggested in literature together with Spiking Neural Networks: Spiking Timing-Dependant Plasticity (STDP). STDP is actually a biological concept, which refers to how the strength of synaptic connections between neurons is adjusted in the brain. Linares et al. have gone as far as to say that “Memristance can explain STDP in Neural Synapses” [22]. To explain this, let us first introduce STDP.

First of all, when it comes to the way the synaptic weights change through the learning process, it makes use of two main techniques [23]. To explain this, a pre-synaptic and a post-synaptic neuron will be considered. That is, given a synapse, the neuron before it and the neuron after it, respectively.

- Anti-Hebbian Learning: This rule makes the strength of synaptic connections (their weight, in ANN terminology) decrease. The term *depression* is also used sometimes to refer to it. Given two neurons connected by a certain weight, if firings usually take place at the first neuron after they do at the second one, the strength of their connection is decreased. In other words, there is no relation of causality between the first at the second one, and thus the influence of the pre-synaptic neuron on the post-synaptic neuron is weakened.
- Hebbian Learning: This rule makes the weights grow. The term *potentiation* is also used sometimes to refer to it.Given two neurons connected by a weight, if a firing usually takes place at the first one before it does so the second one, the weight of the input is increased. This means that a causality relation between them (A *then* B) makes the weights grow.

The total weight change is then gathered by the following learning rule, presented by Kempter, Gerstner et al. and by Kempter et al. [24].

$$\Delta w_i = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_i^n - t_j^f) \tag{1-6}$$

Where t_i^n denotes the timing of the nth firing of the post-synaptic neuron of that weight and t_j^f denotes the timing of the fth firing of the pre-synaptic neuron.
$W(x)$ is the STDP function of choice. According to [25], a widely used W(x)

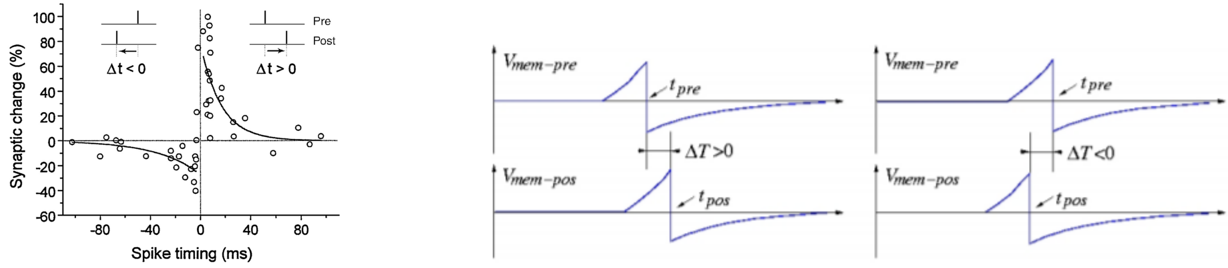
$$\begin{cases} W(x) = A_+ e^{\frac{-x}{\tau_+}} & x > 0 \\ W(x) = -A_- e^{\frac{x}{\tau_-}} & x < 0 \end{cases} \tag{1-7}$$

A_+ and A_- are learning parameters and not necessarily equal. Sometimes one may want to give Hebbian learning more weight than anti-Hebbian learning or vice cersa. Then, τ_+ and τ_- are time constants, in [25] within the order of tenths of milliseconds.

A representation of how the weights change as a function of the timing between the signals following the function in 1-7 can be seen in 1-5a.

Going back to the relationship between memristors and STDP, Linares et al. find a similarity between them by comparing the shape of electrical neural spikes to that of a change of state in an ideal memristor. The details of this will be discussed later on, after memristors are introduced. However it is true that STDP-like behavior can be observed when applying voltage on an ideal memristor, which can be seen in Fig. 1-5b.

STDP does not give any information on how much the weights are increased or decreased numerically or what the thresholds are. However there are still some who have managed to make a learning technique out of it after some small changes and additions. Labeling the outputs must still be done afterwards in these cases.



(a) Representation of the explained Hebbian and anti-Hebbian learning [26]. If $t_i - t_j > 0$, the synaptic change is positive. If $t_i - t_j < 0$, the synaptic change is negative.

(b) Hebbian learning or potentiation (on the left) and anti-Hebbian learning or depression (on the right) observed by applying *Vmem-pre* and *Vmem-Pos* between the terminals of a memristor [22].

Figure 1-5: Simplified version of STDP proposed in [5]

Now, enough information has been presented in order to formulate the research question. Therefore, it will be done in the following section.

1-2 Research question

The challenge to be tackled with this work is to test a new type of technology and approach in the field of Neuromorphic Computing: Memristor-based Neural Networks. The main detail that makes this type of technology attractive is their power efficiency, non-volatility and the similarities between their behavior and STDP processes in the human brain.

This novel type of technology will be tested on a typical Neural Network problem, handwritten character recognition.

This approach will not use an extremely complex neural network, but will rather attempt to create a minimal setup that works with as little complication of the design as possible. Power consumption will also be a key factor in the design, since it is interesting to try and proof that it is indeed an advantage present in neural networks designed with memristors.

The work carried out will attempt to answer the following questions:

- Are memristors a competitive component for Neural Networks, and in particular for Spiking Neural Networks? This will be addressed throughout the entire progress of this thesis.
- Can memristors solve or at least alleviate the von-Neumann bottleneck? Are they helpful when working with highly parallel problems, namely Neural Networks? This question is relevant for Chapters 3 and 4, by studying the memristors of choice and their capabilities when integrated in a Neural Network.
- What are Neural Network topologies or types, such as SNN or ANN, that are particularly suited for them? This is specifically addressed in Chapter 4.

- What are the challenges of circuit design with memristors? That is, are there non-idealities that appear typically when working with them? This will be discussed in Chapter 5.

- What is a suitable implementation of a circuit with memristors that takes advantage of its non-volatile qualities while still being robust? In other words, circuits which can be realistically implemented without a high degree of complexity, that will contain elements to protect the memristors from undesired currents and voltages. Ideally, the design not being too complex should also make it less prone to less predictable effects such as noise. This is also covered by Chapter 5.

In the next section, some background about neuromorphic computing and its state-of-the-art will be given.

1-3 History and Evolution of Neuromorphic Computing

In 1936, Alan Turing provided a formal proof that a machine could be capable of performing any possible mathematical computation if it could be represented as an algorithm [27]. In fact, Turing also anticipated connectionism and neuron-like computing through a machine which consisted of artificial neurons connected with modifying devices. Neurons were composed of NAND gates, since any other logic can be implemented from them [28]. In 1949, Hebb proposed synaptic plasticity as a mechanism for learning and memory retention [23]. Rosenblatt defined the theoretical basis of connectionism and simulated the perceptron, a supervised learning classifier [29].

The primary reason why research started in this direction was the desire to replicate biological neural systems in computers. Over the years since the late 1980s, several motivations led to the evolution of Neuromorphic Computing: The search for hardware that could present neurological features from biological beings.

However, the majority of computers invented back then and up to this day follow an architecture based on the one designed by the scientist John von Neumann in 1945 [30]. A very basic block diagram of it is shown in Fig. 1-6. In it, there are three main blocks: the Central Processing Unit (CPU), the Memory and one or more block(s) of Inputs and/or Outputs. Most of the operations that happen in personal desktop computers can be summarized as the interaction between these three blocks. This division is not only functional but also present at a hardware level. When realizing a calculation, retrieving the necessary values from the memory and storing the results to it will require two separate blocks to communicate. This limitation is known as “the von Neumann bottleneck”.

The field of Neuromorphic Computing continued evolving and in the last years, the possibility of getting rid of the von-Neumann bottleneck and getting closer to the brain performance has continued to be studied by the scientific community [16] [31]. Not only that: the fault-tolerance of neuromorphic solutions, their low power, their speed for highly parallel problems or their smaller footprint also were attractive for many applications. In Fig. 1-7, the main fields of interest within neuromorphic computing and their shifting through the years can be seen.

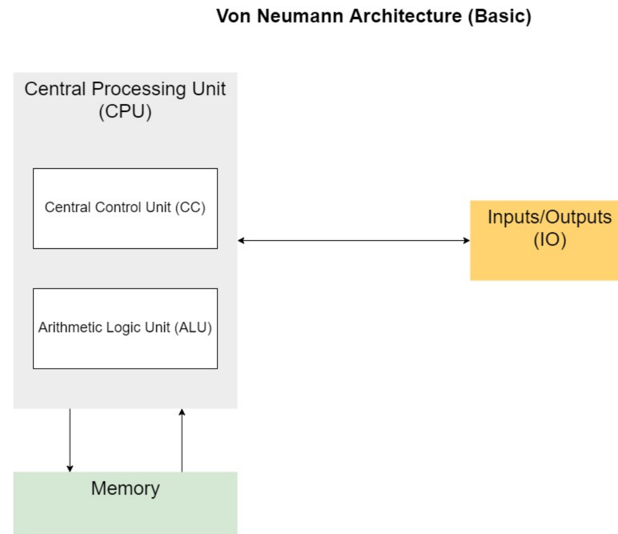


Figure 1-6: Simplified overview of the von Neumann Architecture

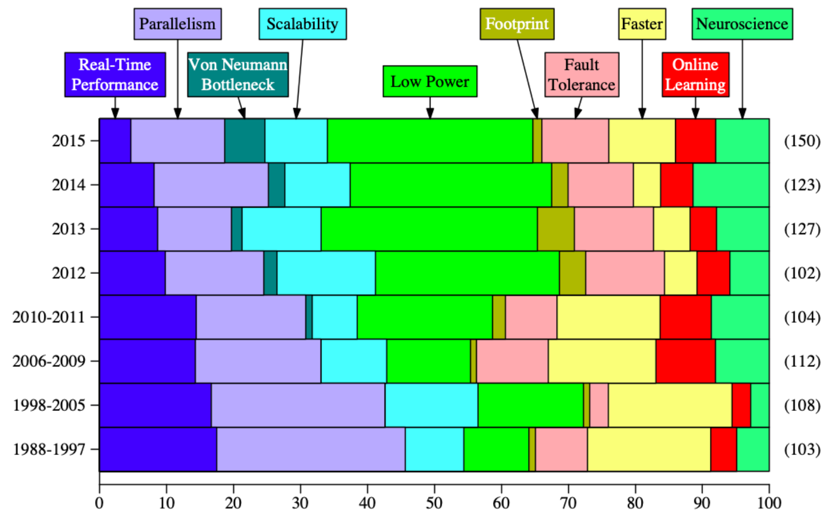


Figure 1-7: Percentage of papers in each field within neuromorphic computing [6]

But the field has also diverged, and to this day there are different lines of research that can be put together under this term. It includes for example material scientists who study how to copy particular properties of biological systems artificially but do not aim to integrate such design in a machine, neuroscientists who continue studying the brain to gain more insight on how it processes information, or engineers who study how to create machines that require less and less explicit programming to solve a problem and/or that become at some point as good or better than humans at tasks that that they would like to automate [6]. The focus of this work is on the third one, more from the hardware but to some extent also from the software perspective. More specifically, Neural Networks will be studied. Let us see what are the existing neuromorphic implementations of them.

1-3-1 Hardware Implementations of Neural Networks

In this section, several available neuromorphic implementations of Neural Networks are discussed. These are divided into three categories, namely analog, digital and mixed platforms.

Purely Analog or Digital Implementations

Analog systems are pieces of electronics where signals are processed as continuous variables. Solutions of this type are known to be suited better for neuromorphic applications with an emphasis on bio-plausibility because factors such as conservation of charge, amplification, thresholding and integration are all characteristics that are present in both analog circuitry and biological systems [14].

Digital neuromorphic implementations, in which signals are treated in a discrete manner, are made out of logic blocks. These blocks are either modular and interconnectable, as is the case of Field-Programmable Gate Arrays (FPGAs), that can be configured using a hardware description level; or custom-designed and with a mostly predefined structure, as is done in Application-Specific Integrated Circuits (ASICs).

Graphical Processing Units, or GPUs, are also a popular digital choice for Neuromorphic Computing in the last decade. Originally meant for Image Processing, a Graphics Processing Unit (GPU) has a high amount of cores and Arithmetic Logic Unit (ALU)s to make simple operations on images, and a higher memory bandwidth (the amount of data it can process at a time). This makes them suitable for Neuromorphic Computing. GPU cores cannot perform complex operations, but they offer a better performance with highly parallel operations. Nevertheless, they are quite power consuming (in the order of MW).

Mixed Digital/Analog

Due to vast similarity to biological systems, analog circuitry is used in these designs for those components where a certain bio-plausible behavior is desired, such as neurons and synapses. Then, robustness is added to the system by means of digital components, which help to cope with non-idealities and undesirable effects. These are more relevant for this thesis.

Conductive-Bridging RAM (CBRAM) The working of CBRAM is based on the electrochemical formation of conductive metallic filaments. It is easy to manufacture, Complementary Metal Oxide Semiconductor (CMOS) compatible and very low power consumption ($\sim nW$). It can be used to implement synapses [32], [33] or neurons [34].

Phase Change Memory (PCM) The resistance change in PCM relies on the reversible crystallization and amorphization of chalcogenide materials. There is a tremendous difference in the electrical resistivity between the amorphous phase and crystalline phase of the phase change material [35]. However, the drawback is that only the crystallization (going to high-conductance state) process can be made incremental, with repetitive pulses slowly crystallizing a high-resistance amorphous plug within the device. The amorphization (going to low-conductance state) tends to be an abrupt process, especially within an array of not quite homogeneous devices. PCM devices have been used as a synapse [36] [37] and also as a

synapse and neuron [38] [39] simultaneously. PCM devices are favourable because they have certain advantages like maturity, scaling capability, high endurance and good reliability over other resistive memory devices.

Spin Devices - MRAM Magnetic Random-Access-Memory (MRAM) is a type of device which stores data in magnetic domains. Magnetic devices have found vast applications in neuromorphic computing since they allow for a variety of tunable functionalities, are CMOS compatible, and are known to have nanoscale implementations for high density [6]. The most commonly used spintronic devices used in neuromorphic systems are spin-transfer torque devices, spin-wave devices and magnetic domain walls. Spintronic devices have been used as a synapse [40] [41], neurons [42] and also as full networks [43].

Floating Gate Transistor (FGT) Floating Gate Transistors have been most commonly used in storage elements like flash memory. They are essentially Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) devices where the gate is electrically isolated, making them capable of retain charge for long periods of time. With the increasing need for better Non-volatile Memory (NVM) devices for neuromorphic computing, several FGT-based analog and digital circuits have been implemented. The most frequent application of FGTs in neuromorphic systems have been either as parameter storage or analog memory cells for synaptic weights [44] [45]. There are also synapse implementations [46] [47], full neuron implementation [48] and dendrite models [49]. The difficulty of moving electrons through SiO_2 in FGTs resulted in their decreased use as memory elements.

Optical Devices Optical devices have found decent applications in the neuromorphic field. They have been used to generate neuromorphic properties like neurons [50] [51] and synapses [52] [53]. During a certain time they lost popularity due their difficulty to store weight, but they are now again relevant because they offer extremely fast operation and not too high complexity. However, efficient hardware chips which are not hard coded are yet to be realised and are in developmental phase. The existing chips and concepts have certain limitations to their complete usage, but it is certainly a growing field.

Memristors Memristors are the main focus of this thesis. These are devices with a variable resistance which can be configured via trains of low voltage pulses. Once configured, they retain their state (their resistance value) also in the absence of power. All of this makes them very suitable for Neuromorphic Computing. Memristors can be used to implement either neurons or synapses, but normally not both within the same circuit. Using memristors for both the neuron and the synapse would add a lot of complexity and make the firing thresholds fixed to the memristor's threshold. For this thesis, memristors will be used as synapses.

1-3-2 Brief overview

Table 1-1 shows a comparison of how the above mentioned non-volatile memory devices have been used to replicate neuromorphic properties. The notation high, moderate and low imply the number of hardware implementations that currently exist using these devices. The "-"

Table 1-1: Comparison of the usage of some neuromorphic devices

	Neuron Implementations	Synapse Implementations	STDP Learning
Memristors	High	High	High
CBRAM	High	High	-
PCM	Moderate	High	Moderate
FGT	Low	High	Low
Spin Devices	Moderate	Moderate	-
Optical Devices	Under research	Under research	-

notation doesn't mean there is no implementation but nothing significant was found while conducting this survey.

It can be seen that memristors are one of the most popular technologies within Neuromorphic computing at the moment, and the most widely used in combination with STDP learning. This is why a specific memristor will be studied with this thesis, and then integrated in different Neural Networks to assess their capabilities. In the next chapter, memristors will be discussed in detail.

Chapter 2

Introduction to memristors

This section introduces memristors and presents typical approaches that have been proposed in literature to work with them.

The memristor was theoretically presented for the first time in 1970 by Chua et al. [54] as the lost "missing circuit element" - the other ones being naturally the resistor, capacitor and inductor.

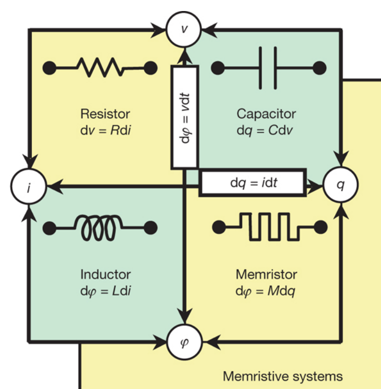


Figure 2-1: The four fundamental two-terminal circuit elements[7]

This -back then theoretical- component would relate flux and electrical charge, behaving similarly to a resistor but exhibiting a hysteresis behavior (and thus, a non-linear resistance also known as memristance). This is formalized in the equations below [55].

$$\frac{dw}{dt} = f(w, v) \tag{2-1}$$

$$i = g(w, v)v \tag{2-2}$$

Where v refers to voltage, w to the state of the device (it can be one or more variables) and g to its conductance.

A memristive device or memristor can also be described as an “analog memory” [55] with an analog state w that can be retrieved by reading out conductance g of the device.

(a) Ideal IV curve of a memristor [56]

(b) Ideal change in conductance versus voltage curve of a memristor without hysteresis [55]

Figure 2-2: Ideal behavior of a memristor’s current and conductance with respect to voltage.

In Fig. 2-2a, the ideal IV curve of a memristor is shown, as presented in [56]. The “pinched” characteristic of the memristor’s IV curve comes from hysteresis. This plot is obtained from memristive devices by doing a voltage sweep in the positive and the negative direction. In this case, it would be from -1 V to 1 V and then from 1 V to -1 V. The reason for the two loops is that when $V = 0$ (that is, in absence of power) the memristor retains the state (resistance) it had been driven to, but current is still zero. However, as soon as the voltage is not zero, there are two different possible currents, depending on the memristor’s resistance (hence the arrows in the plot). This state retention can be seen better in Fig. 2-2b, where the Conductivity Change ($\frac{dg}{dt}$) vs Voltage relation is shown. This curve is more intuitive because it is similar to a diode’s IV curve. This particular curve does not show hysteresis behavior (it would be the result of doing a voltage sweep in only one direction), but it will serve the purpose of illustrating the modes of operation of a memristor. The general idea is that if a voltage sweep between 0 and 1 Volts is applied between the two terminals of the device, its conductivity will barely increase until a certain threshold is reached, point from which the change will be abrupt. When a negative voltage sweep is applied to the device (0 towards $-\infty$), the opposite occurs: its conductivity will decrease negligibly for negative voltages closer to zero, and from a certain threshold on it will do so more abruptly.

There is no consensus about these thresholds, and according to [56] they are subjective. The reason why they are used is that it makes the memristor easier to compare to other electrical components, like diodes.

Real memristors do not have the exact shapes seen in 2-2. There are different types of memristors, but with this thesis only one of them is going to be looked into: Self-Directed Channel memristors. The details about the manufacturing of these devices are out of the scope of this work, but it is important to note that they are distinct to Conductive-Bridging RAM (CBRAM) or Phase Change Memory (PCM), even though they are also made out of chalcogenide materials.

Memristors are very sensitive devices, and Self-Directed Channel (SDC) memristors are not an exception. In order to measure their resistance the most straightforward method is to place a resistor in series of a known value, then apply voltage to both and measure the voltage drop on the series resistor to calculate the current. Since the voltage applied is also known, the memristor’s state can be derived from that.

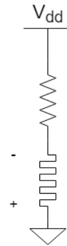


Figure 2-3: A series resistor can be used to measure the memristor's resistance. This resistor will also bound the current through the memristor.

Now, memristors are undoubtedly possible to operate as binary devices with two main states: Low Resistance State and High Resistance State. Anything below Low Resistance State or above High Resistance State is considered unstable and dangerous, since the memristor may not be able to leave that state anymore. These limits are design parameters. The intermediate levels are in theory achievable, but not as easy. This is called Multi-Level Operation and will be explained in more depth later on. Without it, memristors are just as useful as any other transistor logic.

The corresponding I(V) curve can be seen in 2-3. The almost flat lines marked as Low Resistance State (LRS) and High Resistance State (HRS) show that voltages lower than V_F or higher than V_R can be used for reading the resistance of the device without altering it.

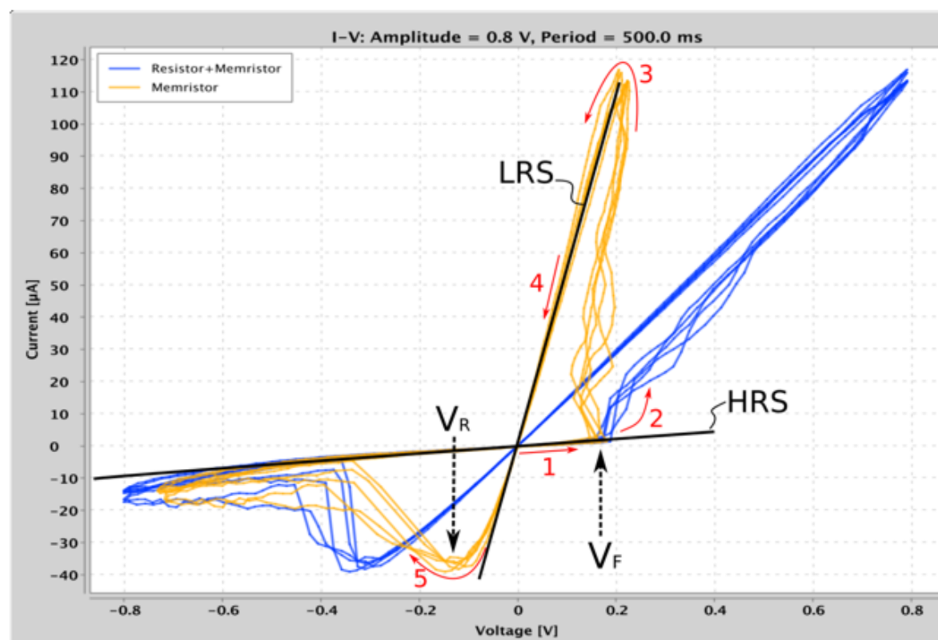


Figure 2-4: I(V) curve. Yellow: I(V) on Knowm's memristor. It can be seen that the changes are more unstable and abrupt than on the ideal curve. Also, the memristor does not always return to exactly the same states (hence the multiple lines that are almost overlapping) due to the device's stochasticity. The blue curve is the current measured on the overall circuit with a series resistor, it is used to calculate the memristor's current. It is not relevant for this explanation.

In Fig. 2-4, the different points that are relevant to the operation of the memristors are

displayed, together with the different modes of operation. Looking at the Memristor curve only, and assuming a voltage sweep starting at 0V [57]:

- It is assumed that the device starts at High Resistance State. The voltage is too low to change the conductivity of the device. This region is marked with the number 1 in the plot.
- V_F has been reached, now the increase in voltage applied does make the conductivity increase. As a result, the current increases substantially. This region is marked with a number 2 in the plot.
- At some point (around 0.8V for this plot), the device reaches its peak conductance. There is a small decrease in current, and a rather unstable resistance value (region 3 in the plot). It is advisable to work below that point. Now the voltage sweep goes in the opposite direction (towards 0V).
- The device is still in Low Resistance State and it will remain there until a second threshold is reached, V_R. This is marked with a 4 in the plot. From that point on (region 5) and until the negative limit of the device, the conductivity decreases with the voltage. As a consequence, so does the current. When the conductivity reaches its minimum possible value, the device is back to the High Resistance State.

V_R and V_F are sometimes called V_{OFF} and V_{ON}, respectively.

The interesting part of this behavior is that the memristor will keep the conductivity value it has been driven to, also when there is no voltage being applied to it. In addition, by applying a small enough voltage and placing a resistor in series, such value can be read without altering the value of the device substantially. This is further illustrated in 2-2.

The memristor presented can be reprogrammed via voltage pulses with a width between 20 μs and 100 μs (knowing, of course, the state that it is in and reading the resulting magnitude in order to fine tune the value). According to Knowm, the way the resistance changes after every pulse is not always the same and it is not predictable nor deterministic, but it can be continuously read and steered towards the right value.

Memristors have been presented briefly, together with their main characteristics. Next, a more specific explanation about their usefulness in Neural Networks is going to be given.

2-1 Memristors in Neural Networks

Memristors can "be scaled down to sub-10 nm feature sizes, retain memory states for years, and switch with nanosecond timescales" [58]. Such properties are interesting for implementing neural networks (where the power usage could be minimised by using spike encoding and event-driven computations [59]) or volatile RAM memories, for example. For all this reasons, there has been a lot of research around memristors for the past years because of their suitability for applications such as "non-volatile memory, neuromorphic and bio-inspired computing and threshold logic" [60].

Another interesting quality they have is that they can be manufactured arranged in what is called a crossbar array architecture, which consists in arranging them in a matrix structure. A

crossbar can be seen in Fig. 2-5. These crossbars can have a very high density. For example, in [12] a procedure is proposed for training crossbar-based multi-layer neural networks. A backpropagation function is applied to update to the weights.

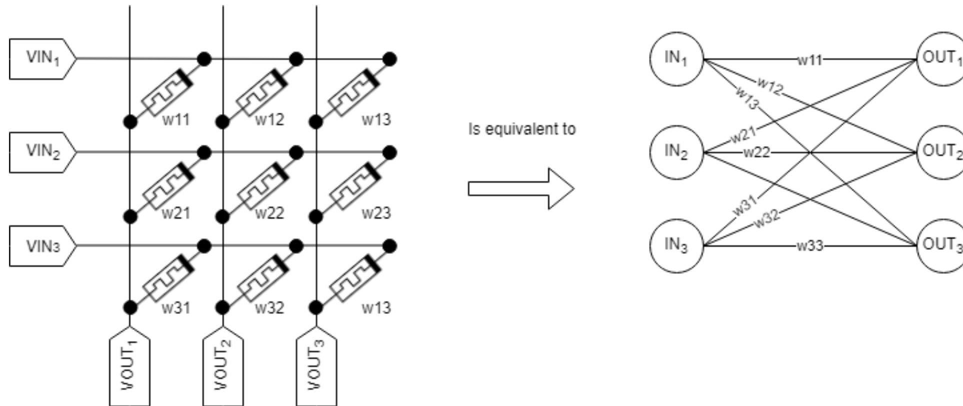


Figure 2-5: Example of a crossbar architecture [55]. On the left, the circuit with synapses but with the neurons displayed as black boxes. Dots mark connection points, the intersections between lines do not. Memristors behave as synapses between the input neurons (left labels) and the output neurons (bottom labels). Effectively, this is one layer of a neural network and can be interconnected with more of them. On the left, the circuit. On the right, the ANN equivalent.

The activation functions of the network, or any necessary conversion between the output of one layer and the input of the next one would still need to be added.

For example, if the output of one layer needs to be used as the input of another layer made out of memristors without reprogramming them, this signal needs to be encoded and adapted so that the voltages are between $-V_R$ and V_F.

Also, when writing to one memristor it is always advisable to isolate the other memristors as much as possible so that they are not reprogrammed accidentally. Complementary Metal Oxide Semiconductor (CMOS) circuitry is often suggested to be used in combination with memristors, since it can be integrated together with a crossbar in the same chip. Choosing what technology is best to be combined with memristors to manufacture a crossbar is out of the scope of this thesis, so during design standard ideal switches will be used here. These contain CMOS technology inside together with other elements that improve their functionality, but that will not be looked into.

Also, the calculation of a backpropagation function or any sort of update function for the weights (if necessary), or the encoding and decoding of the signals would still have to occur externally, via another processing unit as in [61], or via additional circuitry such as a look-up table. This will be explained in more depth in Chapter 4.

Before that, it is necessary to know how memristors can be used as synapses: which voltages may be applied to them, which range of operation they have, how they will behave when voltage is applied to them. The Knowm memristors are going to be studied and characterized in the next chapter.

Chapter 3

Memristor Characterization

This chapter gives some numbers and details about the set-up and signals used on it.

3-1 Experiments and Goal

3-1-1 Experiment Description

The experiments performed will be now briefly explained. A more detailed explanation will be given later.

- A first exploratory experiment was performed in one of the chips, aiming to have a better idea of how these devices work without wearing them out more than necessary. This experiment consisted on trying to drive the memristors to a set of resistance setpoints by applying incrementally stronger pulses until a change was achieved. It was often necessary to apply several set of pulses in the positive and negative direction to achieve the setpoint, despite of the pulses being incremental. This is due to the stochasticity of the Self-Directed Channel (SDC) memristors used. This experiment provided mostly insight about how the memristors work.
- A second one tried to gather more extensive information from them and their dependence on their previous history. Trains of identical pulses of different magnitudes but the same pulse width were applied to the device. This experiment helped find a model to predict the behavior of the memristors and to validate it.
- Every memristor was performed over a whole chip. Thus, 16 memristors.

In general, what is sought for is a model that provides us with the following knowledge:

$$\begin{aligned} R_{k+1} &= f(R_k, V_{MEM}) \\ Y_k &= f(R_k, V_{MEM}) + \xi_k \end{aligned} \tag{3-1}$$

Where Y_k is the measured resistance and ξ_k the prediction error. R_k is the resistance at time k and R_{k+1} is the predicted resistance.

When talking about these memristors, the word "state" will also be used often. This refers to its electrical resistance, or reciprocally to its conductance. High resistance can be seen as a closed switch or the memristor being off, since less electrons will go through that path for the same given voltage. Similarly, low resistance can be seen as an open switch or the memristor being on. Between the on and off states, which are specific resistances provided by the manufacturer, the intermediate states or levels occur. The existence of these is of special interest.

In order to run both experiments, the software tool provided by Knowm is going to be used as a base (a framework developed in Java) [62]. This tool allows for several simple experiments, such as applying individual configurable pulses or train of pulses to the board, or analysing the hysteresis of the memristors via a sine wave. Another two experiments are going to be added as explained above.

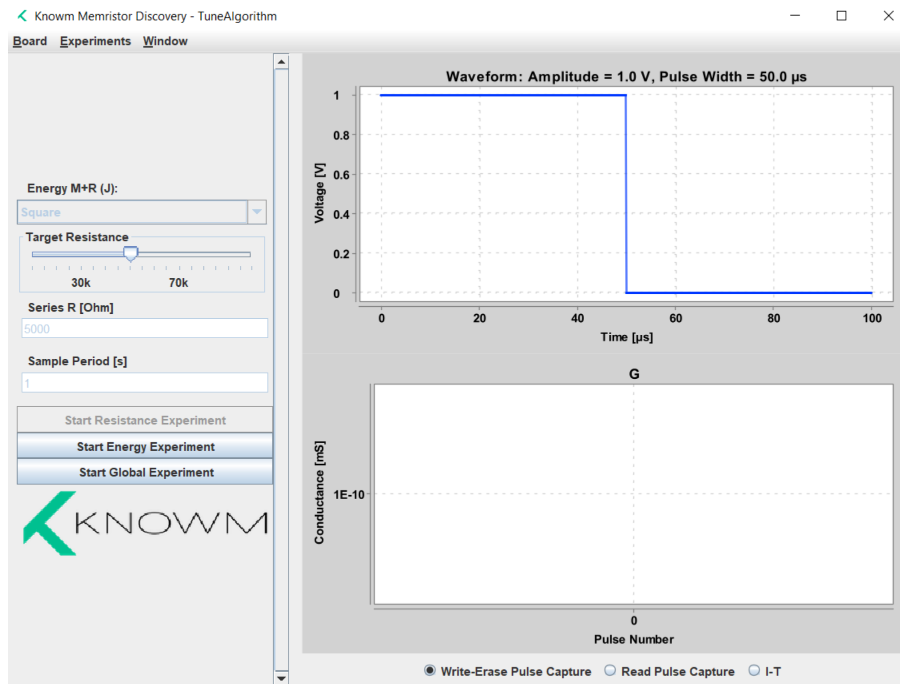


Figure 3-1: The experiment was added to the GUI provided by Knowm with the setup, which was open-source.

3-1-2 Goal

The experiments performed aim to characterize the memristors at hand and predict their resistance change, in order to create a predictive controller that can make use of the memristors in a neural network. The first two experiments performed provided experience and insight, the third one helped validate the chosen model.

The main questions would be:

- What model can be used to predict the behavior of the memristors?
- Does multilevel operation make sense for memristors? That is, is it realistic to expect that the memristors can be used not as binary devices with two stable resistance values, but as devices with several possible resistance values used to encode information?
- How many levels does it make sense to use in the memristor, for a neural network?

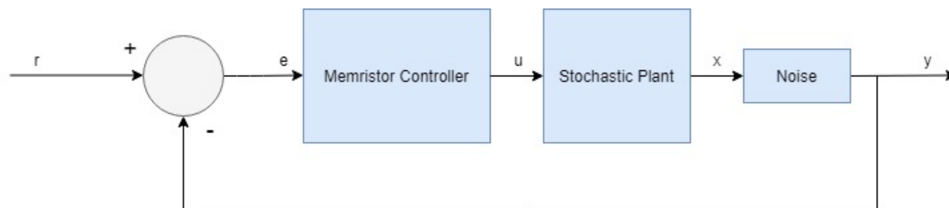


Figure 3-2: The experiments performed aim to help the design of a controller that works with the SDC memristors given.

A block diagram of the memristor controller that will be used can be seen in 3-2. The memristor controller will be used to control a stochastic plant: The memristors. These do not have a deterministic behavior, and cannot be fully predicted. On top of that, there will be a certain amount of measurement noise. This noise will not be taken into account since the measurement device used has an accuracy of $\pm 10mV \pm 0.5\%$ according to its own specifications.

3-2 Brief Explanation of the Set-up

The memristors used are manufactured by Knowm Inc. For the initial experiments, their Memristor Discovery Kit is used both as oscilloscope and signal generator.

Figure 3-3: Initial Experiment Set-Up. Images provided with permission by Knowm Inc

Inside every memristor chip within this setup, 16 memristors can be accessed via a switch (one at a time) that also connects them to a series resistor, as explained in Chapter 2. The default series resistor is 5 $k\Omega$, which was kept throughout the experiments.

Three integrated circuits with 16 memristors were used for these experiments: Two made out of Carbon, and one made out of Tungsten. Carbon and Tungsten memristors have the same forward reverse voltage thresholds (the same V_F and V_R). According to the manufacturer, the switching response in both is very similar but the state retention, or how a state is held by the memristor over long periods of time, is lower on the Carbon ones. The manufacturer does not specify for how long specifically either type can retain their state, but this duration should in both cases be in the order of magnitude of years according to [60]. Thus, it has not been measured. Carbon memristors are supposed to have lower switching energy as well, which means that they will experience more abrupt changes in resistance in general.

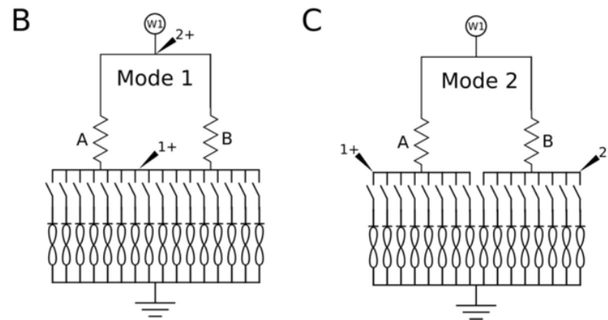


Figure 3-4: Circuit within the Knowm set-up. It has two modes to give also the possibility to test the usage of a synapse made out of two memristors. Images provided with permission by Knowm Inc

Let us first go through some basic ideas that will be used throughout the experiments.

The series resistor acts as a current limit, giving a worst case of 300 μA. This worst case would be reached if the memristor reached a state in which its resistance is so low that it is negligible in comparison to that of the series resistor, which is not desirable. In any case, having the series resistor prevents the current from spiking to extremely high values. It also means that the voltage that reaches the memristor is different to the voltage that is applied to the circuit. In the Knowm setup, neglecting the switches, the actual voltage applied to the memristor is:

$$V_M = \frac{-V_{dd}}{R_{SERIES} + R_M} \cdot R_M$$

The minus sign on the source voltage is due to the fact that the polarity of the memristor is inverted on the setup. The range of resistance that is recommended on these memristors is between 10 $k\Omega$ and 100 $k\Omega$, and the range of voltages is -1.5 V and 1.5 V. Also, V_R = -0.11 and V_F 0 0.27 V. A voltage of 0.1 V is recommended to be used as reading voltage. The range is not symmetric because V_R and V_F are different. Why they are different is not explained by the manufacturer.

Since some authors claim that energy is a defining magnitude for memristors, it will be calculated for some experiments to see if some pattern can be extracted from it. That is the energy generated given an applied voltage and the resulting change in resistance. In memristors, all energy that is generated when changing resistance is transformed into heat by the Joule effect.

The energy per resistance transition will be approximated taking into account three input parameters: Source voltage Vdd (V), pulse width Δt (s) and initial resistance R_{M_0} (Ω). The resistance added by the switches is known: $R_{SWITCHES} = 100\Omega$. Then, using the basic relation $E = I \cdot t \cdot V$:

$$E(Vdd, t, R_{M_0}) = \left(\frac{-V_{dd}}{R_{SERIES} + R_M + R_{SWITCHES}} \right)^2 \cdot R_{M_0} \cdot \Delta t$$

Which leads to a maximum energy per event of:

$$E(1.5V, 20\mu s, 10k\Omega) = 1.97nJ$$

$$E(1.5V, 20\mu s, 100k\Omega) = 0.41nJ$$

To have a preliminary estimation of the energy range in a memristor, one can use the fact that voltages below 0.1 V are used as “reading voltages”, since they don’t affect the device. Then, the energy per reading operation would be

$$E(0.1V, 20\mu s, 100k\Omega) = 1.81pJ$$

$$E(0.1V, 20\mu s, 20k\Omega) = 8.77pJ$$

This is very promising, since in a read operation throughout an entire neural network with a thousand weights, memristors would still not consume more than 2 μJ.

Since the behavior of the memristors is dependent on their history and on the energy that is being given to them, an experiment is going to be performed over 64 memristors to attempt and identify their behavior.

3-2-1 Memristor’s resistance calculation

Since the tools provided by Knowm are going to be used, it is important to explain some details about their calculations. As said before, a resistor of a known value in series with every memristor is used to indirectly measure the current on the memristor.

Since according to Knowm, their board presents capacitive parasitic effects, they do not calculate the current directly by looking at the reading pulse. Instead, they take the voltage of the reading pulse before the falling edge of the pulse. Then, this voltage to is compared against a simulation and a look-up table is generated for a list of possible voltages at that node given a list of resistance R_M the memristor could be in.

The simulation is run with a tool called *JSpice*, a “SPICE-inspired analog circuit simulator made in Java with an emphasis on simulating memristors and analog circuits containing memristors” [63], which allows to use it within their software framework. In any case, the simulation is simply of a RC circuit supplied by a DC source, it contains no memristors. It gives a static voltage versus resistance look-up table that is not dependant on any external parameter besides the chosen series resistor.

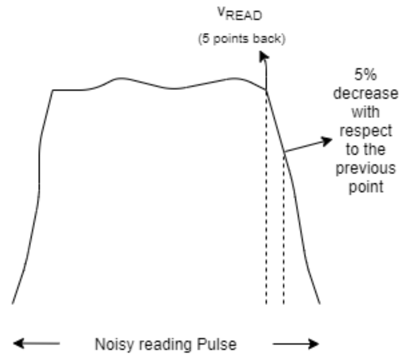
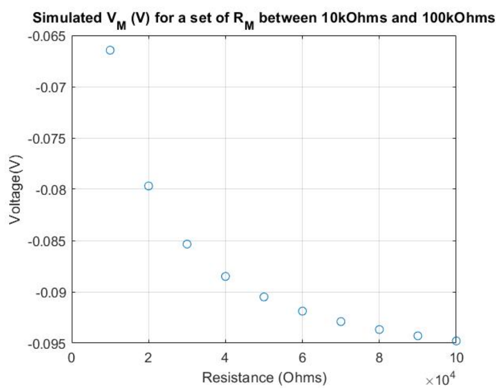


Figure 3-5: How the voltage V_{READ} is selected before using the look-up table. First, a 5 % drop in the voltage is searched for. Then, the point 5 timesteps before that is taken.

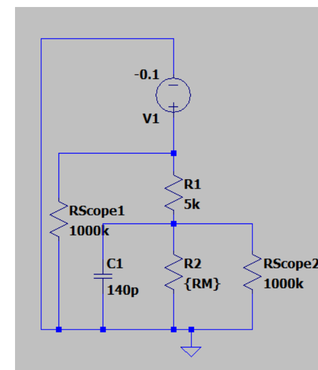
The schematic of the circuit simulated can be seen in Fig. 3-6b. First, the DC voltage source is defined. Then, the memristor and the series resistor. Then, the internal resistors of the oscilloscopes and finally the parasitic capacitance of the switches.

In order to see what type of values this simulation would provide, the same circuit was simulated in LTSpice, a "high performance SPICE simulation software, schematic capture and waveform viewer with enhancements and models for easing the simulation of analog circuits" [64]. Then, a parametric simulation was run for the range of resistance values of interest (between 10 kΩ and 100 kΩ, which is a range in which measurements are both reliable and safe for the memristors according to the manufacturer) and the voltage drop on the memristor was measured. The simulation was carried out with a DC source, without considering transient behavior of the circuit.

That is effectively the circuit in Fig. 3-6b.



(a) The same simulation done by Knowm over a Java library was done on LTSpice.



(b)

Figure 3-6: Circuit simulated in LTSpice, as defined by Knowm.

According to the board manufacturer, something called "polarity inversion" may occur on the board in the old version of the board: Because of a parasitic capacitance, when applying a square pulse the memristor can see a negative voltage as the supplied voltage approaches zero, resulting in a decrease in conductance when an increase is to be expected. This happens

because the parasitic capacitor is still charged and takes some time to discharge still, forcing a voltage drop that in turn makes the memristor see a negative voltage when the supply applies 0 V.

Since the circuit was changed slightly in the current version of the board, there should not be polarity inversion (it will occur across the series resistor in this case, which will have no effect on the memristor itself). However, the parasitic capacitance can alter the voltage that reaches the memristor because of the charging and discharging of the parasitic capacitance. This will get more apparent the shorter the pulses applied are. The pulses applied during these experiments are never below 20 μs, which is considerably larger than the time constant of the circuits and which should give it enough time to settle before the pulse ends. The time constant τ of the circuit given in 3-6b would be:

$$\tau = R_S \parallel (R_M \parallel R_{Scope2}) = \frac{\dfrac{R_M \cdot R_{Scope2}}{R_M + R_{Scope2}} \cdot R_S}{\dfrac{R_M \cdot R_{Scope2}}{R_M + R_{Scope2}} + R_S} \tag{3-2}$$

Which is 663.5 ns for the High Resistance State (HRS) and 557.77 ns for the Low Resistance State (LRS). The fact that the circuit has enough time to charge and discharge is shown in Fig 3-7.

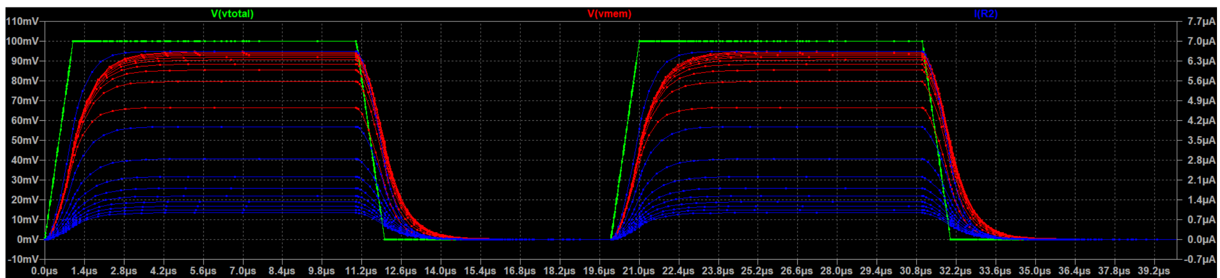


Figure 3-7: Voltage and current across the memristor for a 10 μs pulse width.

These figures show that the transient behavior of the circuit can be neglected for the chosen range of pulse widths (20 μs - 100 μs).

To not make the experiment more time consuming than it already was, it was further checked whether this simulation was all in all necessary. Assuming that the simulation threw perfect values, a second simulation was run ignoring the switch capacitance and oscilloscope probe resistances (as shown in Fig. 2-3, just two resistors in series) to calculate the corresponding voltage drop and what relative error would be.

Resistance Value (Ω)	V_{MEM} (LTSpice)	V_{MEM} (JSpice, Knowm)	V_{MEM} (LTSpice, simplified)	Relative error (Col2 vs Col4)
10k	-0.0664452	-0.0667172	-0.0666667	0.0033223
20k	-0.0796813	-0.0798489	-0.0800000	0.0039840
30k	-0.0853485	-0.0853511	-0.0857143	0.0042674
40k	-0.0884956	-0.0885818	-0.0888889	0.0044248
50k	-0.0904977	-0.0905625	-0.0909091	0.0045248
60k	-0.0918836	-0.0918727	-0.0923077	0.0045942
70k	-0.0928998	-0.0928531	-0.0933333	0.0046450
80k	-0.0936768	-0.0937156	-0.0941177	0.0046839
90k	-0.0942902	-0.0942309	-0.0947368	0.0047145
100k	-0.0947867	-0.0948144	-0.0952381	0.0047394

Table 3-1: Numerical values thrown by both simulations

First of all, the LTSpice and JSpice results with the parasitic capacitance and the probe resistances (column 2 and column 3) are practically identical. This gives validity to the simulation. Then, the difference between the JSpice simulation and the simplified LTSpice simulation was calculated, assuming the first one to give perfect results. The error turned out to never reach 1 %, as shown in Table 3-1. Because of this error being negligible their results were kept but no longer used. The memristor's resistance was calculated by using the average current per reading pulse. That is, given a reading pulse, all time steps where the supply voltage was roughly 0.1 V (the reading voltage) were used to calculate the average current and then the memristor's resistance.

3-2-2 The memristor endurance problem

A special mention should be made about the endurance of the used memristors before going into details about the experiments.

Although according to Knowm they are robust devices with over 9 Million cycles of endurance [60], they also state that when using one of their memristors they may change properties. This happens due to the glass melting after a certain amount of pulses, and then crystallizing again. As a result, the memristor may no longer be able to get out of its current resistance state, it may have a more reduced resistance range or it may present other changes that are more difficult to identify. That is, its properties will have degraded, and that memristor will be flagged as unusable since it cannot be used to help characterize a properly working memristor nor to train a Neural Network.

This happened many times while performing these experiments. It was not possible to find a way of using the memristors that completely avoided this situation. This information is used to decide how to perform the experiments as well.

3-3 First experiment

The idea behind this experiment is to use a very generic and simple procedure found in literature to tune memristors to different setpoints. Then, the information obtained by doing this can be analysed in search of a better tuning method.

This first experiment was based on the method proposed in [8] to control memristors. Alibart proposes a method to modify the resistance in a memristor by applying increasingly stronger pulses in the necessary direction (positive or negative). This method divides the pulses that can be applied to the memristor in two types (this naming will also be used later on for the sake of convenience):

- Set: Positive voltage pulses across the memristor are called *Set* pulses because they drive it towards its ON state gradually (about 10 $k\Omega$ for this set-up at the beginning, but a more conservative 20 $k\Omega$ later on).
- Reset: Negative voltage pulses across the memristor are called *Reset* pulses because they drive it towards its OFF state gradually (about 100 $k\Omega$ for this set-up)

The method described in [8] could be called a "lazy" controller, because it controls the Resistance State of the memristor via increasingly higher pulses (either positive or negative, depending on the desired direction) until the desired target resistance is achieved. Steps of $10k\Omega$ between 20 $k\Omega$ and 100 $k\Omega$ will be used as target resistance.

Figure 3-8: Algorithm proposed in [8]

This method will always work in the end, since by applying pulses that are increasingly high in magnitude on a working memristor, some change must occur eventually. Then, if the change is higher than desired, pulses of reversed polarity can be used to correct it. However, this method is time consuming. Having to use correcting pulses when aiming for a setpoint is almost inevitable, since the memristors are stochastic. But instead of starting from low voltages and slowly increasing them, it would be ideal to be able to generate an educated guess.

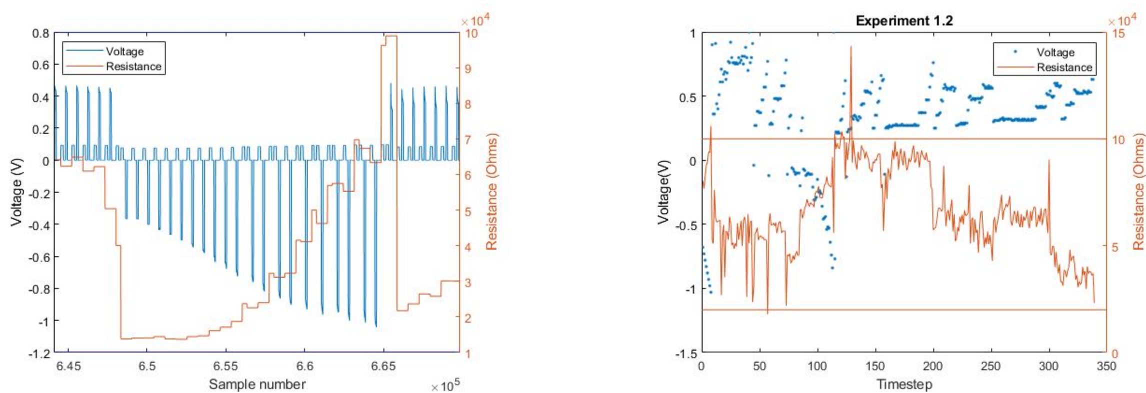
A Carbon memristor was used for this experiment. In it, for all **16 memristors** present in the chip,

- The lowest setpoint was initially 10 $k\Omega$. This turned out to not be a stable state for the memristor, since sometimes no pulse could drive it to other states anymore. Thus, 20 $k\Omega$ were used later on.
- Setpoints ranging from 20 $k\Omega$ (10 $k\Omega$ at the beginning) to 100 $k\Omega$ in 10 $k\Omega$ steps are used.
- Voltage pulses of 20 μs width and voltages between 0.2 and 1.5 V in magnitude are used, with steps of 0.05 V. That is, first $\pm 0.2V$ are applied, the sign depending on whether the required change is negative or positive. Then, the magnitude of the pulses is increased in steps of 0.05 V until the setpoint is either achieved or skipped.
  - If the setpoint is skipped, the pulse sequence starts over with the opposite polarity.

- If the setpoint is achieved, the next setpoint is aimed for.

- After every setting or resetting pulse, a reading pulse is used to measure the memristor's resistance.
- Setpoints must be achieved with 5 % accuracy.
- If more than 20 pulses are applied and the resistance change is negligible on the memristor, the memristor is flagged as broken.

A second version of this experiment was later on performed on another chip, made out of Tungsten. The goal was to see if an increasing and a decreasing setpoint (20 to 100 $k\Omega$ and 100 $k\Omega$ down to 20 $k\Omega$) to have more complete data. The principles were the same as explained before, only the list of setpoints changed. Timeseries from both experiment versions can be seen side to side in Fig. 3-9.



(a) Small part of Experiment 1 zoomed in. Resistance of a memristor being changed by applying trains of pulses. Negative pulses increase the resistance, positive pulses decrease it. A reading pulse can be seen next to every writing pulse (the 0.1 V pulses).

(b) Expanded version of Experiment 1 performed twice on a memristor. The resistance is first reset to HRS, then brought down to LRS. Then the cycle is repeated again. The horizontal lines represent HRS and LRS. In this plot, only writing pulses are displayed for the sake of clarity. It can be seen how they are increasingly higher in magnitude either in the set or reset direction, depending on where the setpoint lies.

Figure 3-9: Circuit simulated in LTSpice, as defined by Knowm.

3-3-1 Second experiment: From HRS to LRS with trains of identical pulses

The second experiment comes from a reasoning found in [65], arguing that the resistance change per energy unit (or "dR/dE") is a device constant that can be used to characterize it. This suggests that applying trains of pulses of identical amplitude should have the same effect on the memristor per pulse in average. It also suggests that the ratio between the change in resistance and the applied energy is the same for different initial states and different applied energies. Lastly, it suggests that a pulse that is not high enough in amplitude to drive the memristor high should have a stronger effect on it simply by increasing the width of the pulse.

for *Voltage j ranging from 0.2V to 1V in 0.05V steps* **do**
  for *Memristors 1 to 16 of every chip* **do**
    Attempt to reset memristor i to High Resistance State;
    if *Reset Fails* **then**
      Mark that memristor as faulty;
    end
    for *100 repetitions* **do**
      Apply voltage j;
      if *Memristor i in Low Resistance State* **then**
        Attempt to reset memristor i to High Resistance State;
        if *Reset Fails* **then**
          Mark that memristor as faulty;
        end
      end
    end
  end
end

Algorithm 1: Pseudo-code of the experiment performed

To quantify to what extent those statements are right or wrong, a hundred square pulses of voltages ranging between 0.2 and 0.95 V were applied to devices that had been reset to HRS beforehand (all 16 memristors of a Carbon chip). One hundred pulses were applied for every memristor of a Carbon chip, to obtain a statistically sound result while still preserving some chips. Since the memristors were in series with a resistor, the voltage that reached the memristor was always slightly different to the one applied to the circuit. This still made it possible to identify the dR/dE ratio and to see the effect of applying the same pulse repeatedly. Also, a varied range of voltages were applied to the memristors, as it can be seen in 3-10. This was good to obtain more complete data, from a statistical point of view. Thus, it was not accounted for or corrected. The pseudo-code of this experiment can be seen in 1.

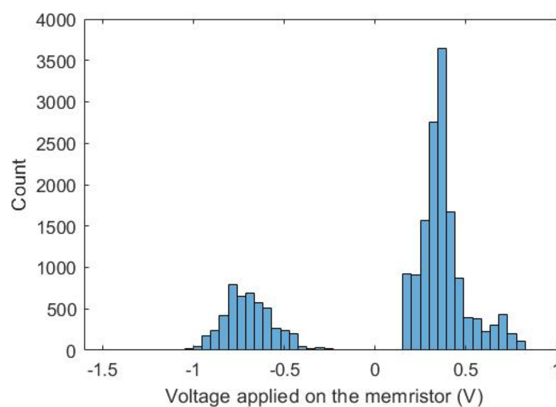


Figure 3-10: Histogram of the voltage applied directly on the memristor (measured via a scope)

Every time the device approached LRS, it was reset again. One could argue that the resetting

the memristor every time would give more reliable information. However, letting the memristor change towards LRS gives information on intermediate states, which was desirable, and makes use of less pulses, which also makes it less likely for the device to change properties and become unusable.

The reason behind applying a hundred repetitions of every pulse is that the behavior of the memristor has a stochastic component, making them not react the same in every situation to the exact same pulse being applied to them in the same state.

A plot of the voltage and resistance during the experiment in one of the memristors can be seen in 3-11.

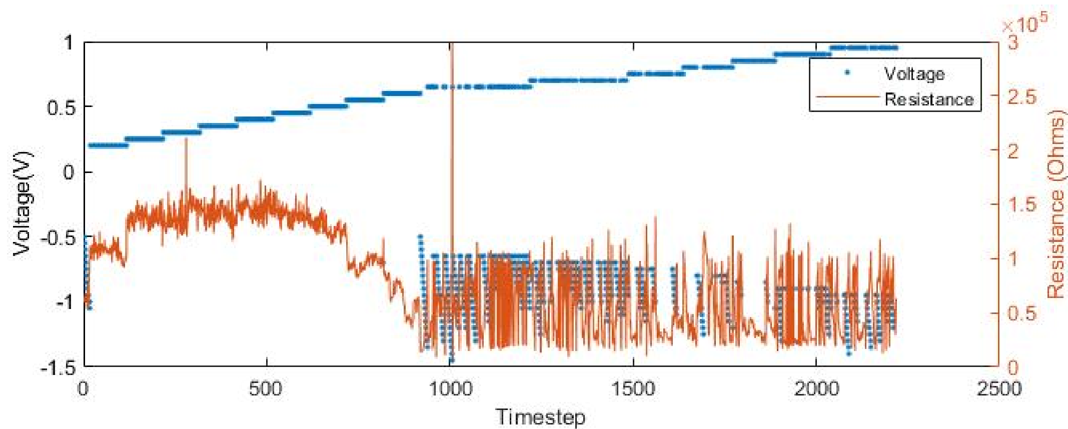
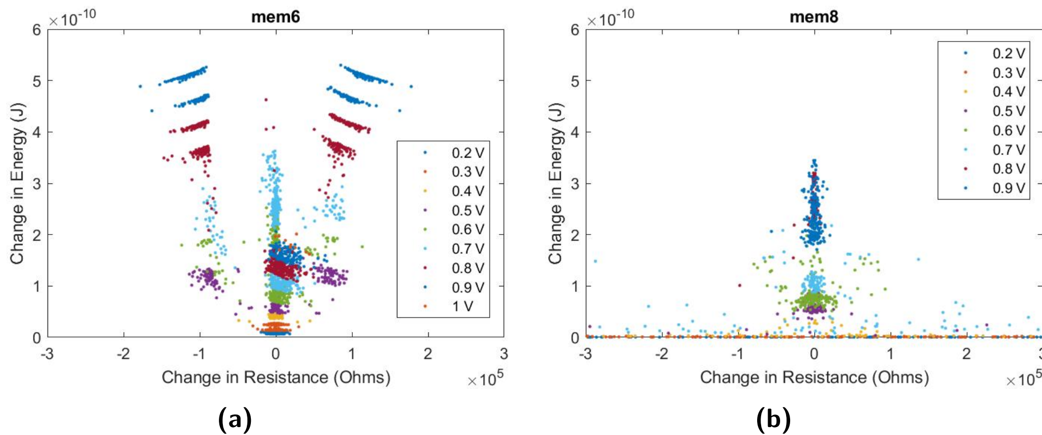


Figure 3-11: Experiment 2 on one of the memristors.

Since in [65] they propose that the dR/dE is a device constant for memristors, the Energy vs Resistance Change was plotted for all functional memristors in two chips independently (a Tungsten and a Carbon chip).

However, in the Knowm memristors not only has that relation between magnitudes not been found: It was also different for every memristor. That can be seen in figures 3-12a to 3-12b and in 7-1a to 7-9b (appendix).

Besides the plots not showing such a constant relation between change in resistance and applied energy, no general fit was found in any of the experiments that established a connection of any type between the two and was valid for all devices.



(a) **(b)**

Figure 3-12

3-3-2 The Mean Metastable Switch Memristor Model

This section describes how the efforts to interpret the data and turn it into a model have been carried out. It also explains a model proposal for the memristor behavior.

It is important to keep in mind that given a pulse applied on a memristor on a certain state R_{M0}, the subsequent state is not deterministic. In none of the tests did the memristors always present the same resistance change for a given pulse and a given initial state, although they did display a certain trend in average. It is known that negative pulses are likely to drive them towards OFF or HRS state, and positive pulses are likely to drive them closer to ON or LRS state. But that can happen immediately sometimes, and gradually some others (going through intermediate states). Sometimes, a pulse will produce no resistance change, and sometimes (because of relaxation effects) after a non-effective pulse the resistance will seem to go in the opposite direction of the one it should, slightly.

Also, a memristor having unusual behavior may have simply degraded its properties due to glass melting, as explained in subsection 3-2-2. It is important to be careful when modelling these devices, since using the wrong data will make the model less functional. Carefully choosing the “best” behaving memristors is necessary.

In any case and leaving unexpected behaviors aside, the model proposed in [10] has been used as a starting point for predicting the behavior of the memristors.

A few concepts need to be introduced in order to explain this model. First of all, the resistance R_M has been scaled between 0 and 1 into a new variable called X. When the memristor is at 20 $k\Omega$ or less, $X = 1$ (the device is said to be fully ON). When the memristor is at 100 $k\Omega$ or more, $X = 0$ (the device is said to be OFF). Any intermediate state is scaled linearly.

The reason to do that is that the model assumes that every memristor behaves in a way like a large number of switches in parallel. When all of them are closed, the resistance seen is the LRS and when all of them are open, the resistance seen in the device is the HRS.

This semi-empirical model assumes that the current in a memristor has two components: a memory component and a Schottky diode current component (present in many memristive devices), both in parallel. Without going in too much depth into device physics, it is important to explain that the memory component provides the switch-like behavior, since it comes from

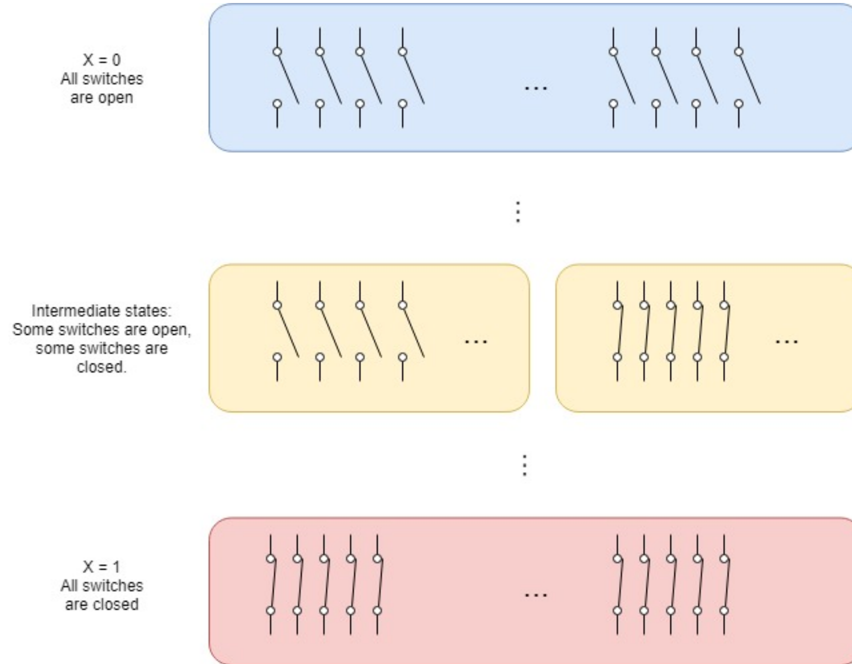


Figure 3-13: Graphical explanation of the metastable switch memristor model

the fact that in the device there are many conducting channels that switch between two states when excited with a voltage gradient. The Schottky component, on the other hand, gives the voltage thresholds, since it acts like a barrier that can only be overcome when a voltage threshold is exceeded. Both of these behaviors, switching channels and overcoming the Schottky barrier, have a probabilistic component as well. No voltage is guaranteed to do either, but it may have a high chance to do so. Based on that knowledge, an equation was put together for the probability for any of the conducting channels within the device to change state.

At any given time, when a pulse is applied, the probability for a conducting channel in the device to transition:

$$\begin{aligned} P_{ON} &= \alpha \frac{1}{1 + e^{\beta(V - V_{ON})}} \\ P_{OFF} &= \alpha \left(1 - \frac{1}{1 + e^{\beta(V - V_{OFF})}}\right) \\ \alpha &= \frac{\Delta t}{\tau} \end{aligned} \tag{3-3}$$

Where V_{ON} and V_{OFF} are the voltage thresholds of the device in the positive and the negative direction, also called V_F and V_R (0.26 V and -0.11 V according to the datasheet, for Carbon devices). Δt is the width of the voltage pulse applied. Then, every time a pulse is applied it may or may not have an effect in every one of the switches. According to [10], for a given initial state within the total amount of switches N (where N_{ON} are open and N_{OFF} are closed, for example), the probability of a voltage pulse closing k open switches follows a binomial distribution, and so does the probability of a voltage pulse opening k closed switches. When

the number of switches in the model is large, the binomials can be approximated by a normal distribution with

$$
\begin{aligned}
\sigma^2 &= N_{OFF} \cdot P_{ON} \cdot (1 - P_{ON}) \\
\mu_{OFF} &= N_{ON} \cdot P_{OFF} \\
\sigma^2 &= N_{ON} \cdot P_{OFF} \cdot (1 - P_{OFF}) \\
\mu_{ON} &= N_{OFF} \cdot P_{ON}
\end{aligned}
\tag{3-4}
$$

Therefore, the average change in relative resistance X for that initial state and voltage pulse will be:

$$
\begin{aligned}
dX &= \frac{N_{OFF} - N_{ON}}{N} = \\
&= \frac{\Delta t}{\tau} \left[\frac{N_{OFF}}{N} \frac{1}{1 + e^{-\beta(V - V_{ON})}} \cdot (1 - X) - \frac{N_{ON}}{N} \left(1 - \frac{1}{1 + e^{-\beta(V + V_{OFF})}} \right) \cdot X \right]
\end{aligned}
\tag{3-5}
$$

If averages are taken,

$$
\begin{aligned}
\overline{dX} &= \frac{N_{ON} - N_{OFF}}{N} = \frac{N_{ON}}{N} P_{OFF} \cdot (1 - X) - \frac{N_{OFF}}{N} P_{ON} \cdot (X) = \\
&= \frac{\Delta t}{\tau} \left[\mu_{ON} \cdot (1 - X) - \mu_{OFF} \cdot X \right]
\end{aligned}
\tag{3-6}
$$

This average resistance change is called the *Mean Metastable Switch Memristor Model*, and it can be used to predict what resistance change is produced by a voltage pulse in average. In other words, given the stochasticity of memristors, if the same pulse was applied to a large population of memristors, $\overline{dX}$ yields what the average resistance change over all of them would be.

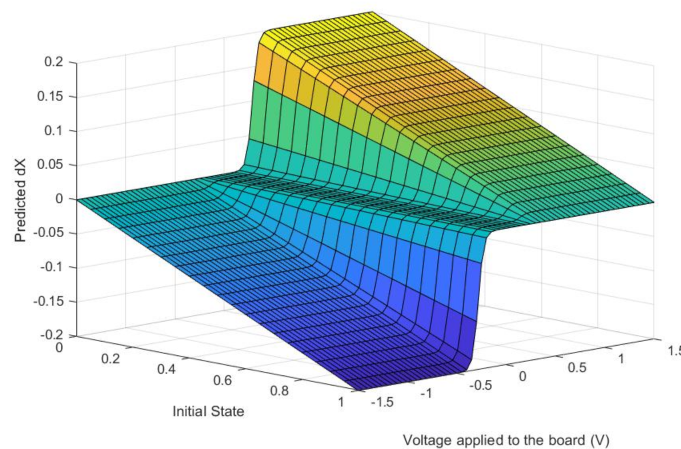


Figure 3-14: Mean Metastable Switch Memristor Model as explained in [9] and [10]

This model has been fine tuned to try and fit the memristors data. Six parameters have been identified in the model: Two related to the time constant τ (depending on whether the voltage applied is negative or positive), two related to the threshold voltage for Set and Reset operations and two related to how quickly the change in resistance grows as the voltage applied increases. The parameters that were used during optimization are shown in 3-7.

$$\begin{aligned} P_{OFF} &= \frac{\Delta t}{\theta_1} \cdot \frac{1}{1 + e^{-\theta_5 (V_{MEM} - \theta_3)})} \\ P_{ON} &= \frac{\Delta t}{\theta_2} \cdot \left(1 - \frac{1}{1 + e^{-\theta_6 (V_{MEM} + \theta_4)}}\right) \end{aligned} \tag{3-7}$$

Where, by default:

- $\theta_1 = \theta_2 = \tau = 100\mu$s
- $\theta_3 = V_{ON} = 0.27$ V
- $\theta_4 = V_{OFF} = 0.11$ V
- $\theta_5 = \theta_6 = \beta_1 = \beta_2 = 1/0.026$

The cost function to optimize then is the difference between the predicted resistance and the actual one. For a particular timestep, that is:

$$Err_k = Abs(X_k - (X_{k-1} + \widehat{dX_k})) = dX_k - \widehat{dX_k} \tag{3-8}$$

The error is then at the end of every time series added together.
Where $\widehat{dX_k}$ is the predicted change in normalized resistance for that particular time-step.
The code used for the optimization can be seen in the Appendix.

Results before optimization

The model is going to be applied to the data from the experiments performed with the default parameter values, to see how good its predictions are. For every time-step, the resistance will be predicted using the model and the information from the previous time-step.

The time series of Experiment 1 results and the predictions (with error bars using the variance of the distribution for that particular prediction) is in Figs.7-10a to 7-14b. An average absolute error per sample in terms of dX has also been calculated for every memristor, gathered in Table 3-2.

The time series of Experiment 2 is not shown here because this experiment has many more data points. Only one chip was used, to damage the other one as little as possible. It can be seen that less memristors are functional within Chip 1 than in the previous experiment. The results are put together in Table 3-3.

Table 3-2: Memristor estimation error for Experiment 1. Note that there are 16 memristors per chip, but the ones that were flagged as broken are not shown. For the second chip, the number of pulses is also shown. This is because the experiment had two parts and some memristors worked on the first part but no longer on the second one, and in that case the data from the first part was kept. For that reason, the amount of pulses can be quite different for the second version of the first experiment.

Chip 1 (Carbon)		Chip 2 (Tungsten)		
Memristor Number	Average error per sample (dX)	Memristor Number	Average error per sample (dX)	Number of Applied Pulses
0	0.1460	0	0.1257	561
1	0.1293	1	0.1060	339
2	0.2201	4	0.3565	279
3	0.1445	5	0.1106	302
4	0.3370	6	0.1104	303
5	0.0959	8	0.1759	182
6	0.1076	11	0.1200	68
7	0.1424	13	0.1218	334
8	0.1440	14	0.1091	146
9	0.0888	15	0.1411	126
10	0.2003	-	-	-
11	0.1239	-	-	-
Average error per sample per memristor	0.1566	Average error per sample per memristor	0.1477	-

Table 3-3: Memristor estimation error for Experiment 2.

Chip 1 (Carbon)		
Memristor Number	Average error per sample (dX)	Number of Applied Pulses
1	0.1366	2217
2	0.1986	1731
3	0.0903	2009
4	0.2110	1855
5	0.2870	2141
6	0.3117	3110
8	0.1427	1604
10	0.1244	1664
Average error per sample per memristor	0.1878	-

Despite of some outliers being present, it can be said quite generally that overall the error is rather constant.

3-3-3 Model exploration via optimization runs

Sequential Quadratic Programming has been the main method used to optimize the parameters. This gave a lot of results that did not make physical sense (negative coefficients that should be positive, too high or too low values and so on), and even when giving boundaries

to all parameters it still caused over-fitting. The optimization also became really time consuming. A more careful set of optimization runs has been carried out, optimizing over less parameters,not adding any scaling to the model or not changing β, which is a very standard parameter from device physics. Two other algorithms have been tested alongside SQP as well: Simulated Annealing and Genetic Algorithm, to assess whether one of them performed significantly better than the others. It was not the case. The details about the optimization runs can be seen in the next paragraphs.

Run 1: Optimizing V_{ON}, V_{OFF} θ_5 and θ_6

Some parameters described in 3-7 have been given fixed values to simplify the model:

- $\theta_1 = \tau_1 = 100\mu$s
- $\theta_2 = \tau_2 = 100\mu$s

The lower and upper boundaries for the parameters can be seen in table 3-4. Their voltage thresholds (V_{ON} and V_{OFF}, respectively) are based on the datasheet of the memristors provided by the manufacturer but with slightly wider ranges. According to Knowm, for Carbon and Tungsten memristors V_{ON} should be between 0.15 and 0.35 V and V_{OFF} should be between 0.05 and 0.27 V. However, these ranges were too narrow to provide any improvement. To see if there was any value slightly out of them that could improve the predictions, they were widened.

Table 3-4: Lower and Upper bounds given to the parameters to be tuned

Parameter	θ_3	θ_4	θ_5	θ_6
Lower Bound	0.15	0.05	0.01	0.01
Upper Bound	0.8	0.8	5	5

The results of the optimization runs are displayed in table 3-5

Looking at these results, Sequential Quadratic Programming yields the best cost function values. However, after generating these results, they were all checked for over-fitting and they all over fit the data. Thus, a general parameter cannot be extracted. Also, that the behavior in the setting and resetting pulses is asymmetrical (more so than the initial values of the parameters indicate, since the optimized V_{ON} is close to 1 and the optimized V_{OFF} is nearly zero.).

Going back to the overfitting, when looking at the time series for the different experiments closely, as displayed in 3-15, in many occasions the predicted change in resistance is negligible (every predicted value is almost identical to the previous resistance value). This results in a lower error in average, but it generates a model that does not succeed in predicting the behavior of the memristors.

Table 3-5: Comparison between three different optimization algorithms. They seem to give similar results.

Sequential Quadratic Programming			
Parameter	Experiment 1 Carbon Chip	Experiment 1 Tungsten Chip	Experiment 2 Carbon Chip
V_{ON}	0.8000	0.7157	0.5445
V_{OFF}	0.0500	0.0616	0.0500
θ_5	0.0100	4.7057	1.6503
θ_6	1.6669	2.9403	4.3097
Cost Function	151.3810	546.2903	3.7124e03
Genetic Algorithm			
Parameter	Experiment 1 Carbon Chip	Experiment 1 Tungsten Chip	Experiment 2 Carbon Chip
V_{ON}	0.8000	0.7162	0.5445
V_{OFF}	0.0501	0.0616	0.0501
θ_5	0.0100	4.7074	1.6504
θ_6	1.6669	2.9404	4.3097
Cost Function	164.6433	698.3755	5.7096e03
Simulated Annealing			
Parameter	Experiment 1 Carbon Chip	Experiment 1 Tungsten Chip	Experiment 2 Carbon Chip
V_{ON}	0.7989	0.7035	0.5446
V_{OFF}	0.0500	0.0627	0.0500
θ_5	0.0502	4.0368	1.7135
θ_6	0.5446	0.0500	4.3141
Cost function	151.6433	546.9406	3.7125e03

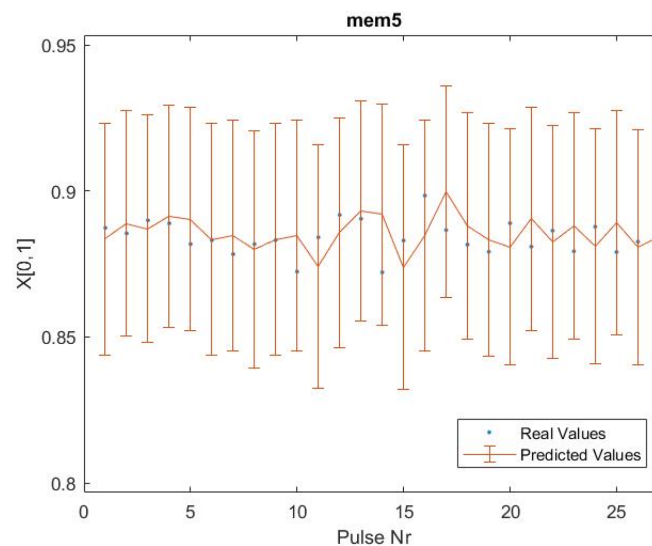


Figure 3-15: Proof that the results thrown by the optimization runs do not fit the data properly. The predicted change is negligible for most points, which can be seen in how similar the predicted value is to the previous value.

The explanation for this is that there are in fact many pulses that have no effect on the memristors. This is not ideal when it comes to having more accurate predictions or a better

controller. For that reason, a final attempt to optimize the model has been made by only tuning V_{ON} and V_{OFF}.

Run 2: Optimizing only V_{ON} and V_{OFF}

The optimization algorithm was altered so to search for optimal values on V_{ON} and V_{OFF} only (θ_3 and θ_4 in the model presented in 3-7). The other parameters described in the model have been given the following values:

- $\theta_1 = \tau_1 = 100\mu$s
- $\theta_2 = \tau_2 = 100\mu$s
- $\theta_5 = \beta_1 = 1/0.026$
- $\theta_6 = \beta_2 = 1/0.026$

Upper and lower bounds have been set for both V_{ON} and V_{OFF}: They both must be between 0 and 1.5 V. The upper bound is rather large but no results above 0.8 were obtained, so it was kept.

Optimizing converged to similar values for Experiment 1 and Experiment 2, as it can be seen in 3-6

Table 3-6: Comparison between three different optimization algorithms. They give similar results.

Sequential Quadratic Programming			
Parameter	Experiment 1 Carbon Chip	Experiment 1 Tungsten Chip	Experiment 2
V_{ON}	0.8000	0.6525	0.5096
V_{OFF}	0.0500	0.0500	0.0500
Cost Function	164.6165	698.3738	5.7096e03
Genetic Algorithm			
Parameter	Experiment 1 Carbon Chip	Experiment 1 Tungsten Chip	Experiment 2
V_{ON}	0.8000	0.6525	0.5096
V_{OFF}	0.0500	0.0500	0.0500
Cost Function	164.6166	698.3755	5.7096e03
Simulated Annealing			
Parameter	Experiment 1 Carbon Chip	Experiment 1 Tungsten Chip	Experiment 2
V_{ON}	0.8000	0.6524	0.5096
V_{OFF}	0.0500	0.0500	0.0500
Cost Function	164.6165	698.3739	5.7096e03

These results also produce over-fitting, which is also easily seen in how similar these optimized parameters are to the ones obtained in the previous optimization run.

3-3-4 Analysis of the data and the results

The optimization showed no improvement upon the parameters given by the manufacturer. This is not per se a negative result, since it means the model is not so easily improvable and it

yields relatively valid results already. Combining this models with other models, or improving the Metastable Switch Memristor Model (MSMM) model in more complex ways could have been possible. However, this seems to be slightly out of scope. A materials scientist or device physicist with deeper knowledge about where these equations come from and where they allow for some changes could possibly generate a much richer model, but purely from the Optimization perspective it is not so easy to devise better models without employing more time than what is available for this task. Thus, the MSMM model is going to be used as it is without changing its parameters.

The data extracted from the experiments has been found to have at least some structure to it. If all the data points from all experiments are put together and the dX is plotted vs the memristor voltage, as is done in Fig. 3-16, various exponential curve-looking patterns can be seen. There seem to be a variety of exponential curves on the plots.

Then, predicted average dX values are generated by the model with the default parameter values and also plotted to show that this exponential trend is present on the model as well. This can be seen as part of validating the model.

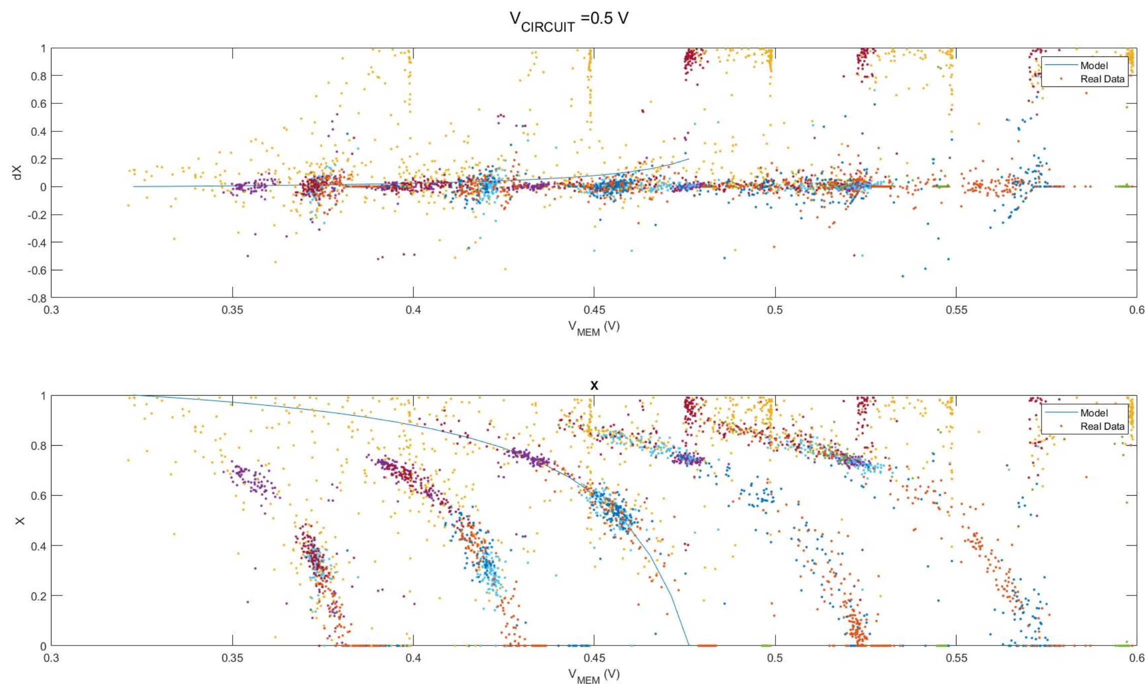


Figure 3-16: dX vs V_{MEM} for all pulses in which 0.5 V were applied to the circuit. Memristor Data (dots) vs Model Data (line). Every color represents a different memristor.

Due to the stochasticity of memristors and the fact that the predicted values only show behavior resistance change, only one of the exponential curves is fitted. This exponential behavior can be explained by the Schottky diode in parallel with a memory element behavior, as said in Subsection 3-3-2. Every curve corresponds to different states, different conductive channels being ON and OFF.

However,no particular memristor contributes to a specific curve. Rather than that, all of

them seem to contribute to all the curves together. The origin of this different clusters of data seems to be the stochastic nature of the memristive devices, that makes them take different resistance values.

When using the parameters obtained in the optimization runs in the model, the result does no longer fit any of the curves. This is easily noted in Fig. 3-17, and will be used as a final argument to discard these optimization results. The average error does improve. Nevertheless, as seen before in the optimization results, these parameters will vary depending on the experiment.

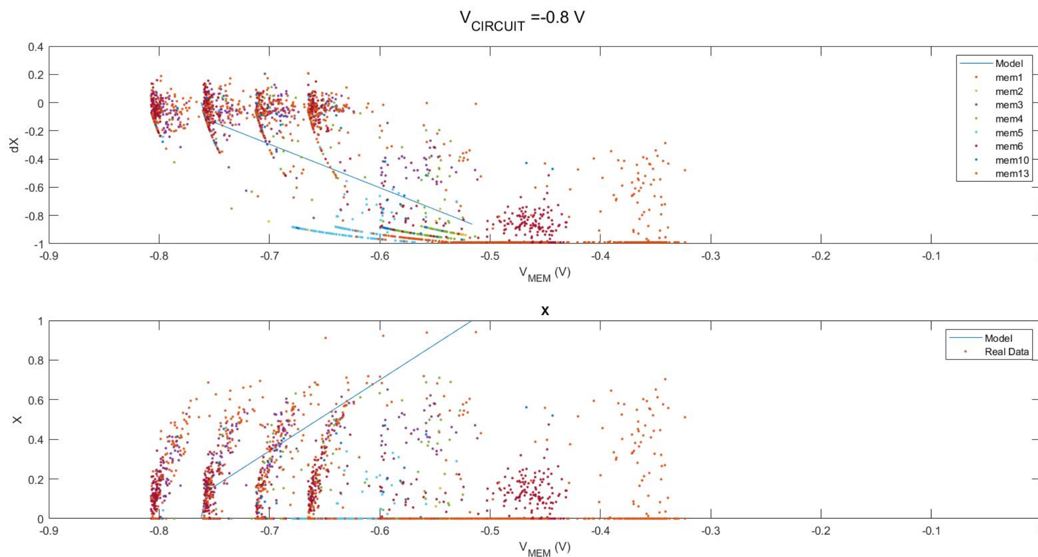


Figure 3-17: dX vs V_{MEM}. Memristor Data (dots) vs Model Data (line). Here, the model data was calculated using the parameter values from the optimization runs, which further proves they are not valid. The model does not longer fit any of the curves. Every color represents a different memristor.

A general measure that would be compliant with the optimization results would be to use the smallest possible V_{ON} and largest possible V_{OFF} from the memristor's datasheet, but in general it is not possible to make the model improve significantly. This is somewhat logical, since these constants are used during manufacturing as a guideline. It is always attempted when manufacturing a memristor to have almost exactly the thresholds in the datasheet, so there should not be mayor deviations. Thus, the default parameters will be used on the MSMM model throughout the rest of this thesis.

3-4 Simulation Work

In order to see if the model chosen appropriately captures the main features of the behavior of the memristors, a Simulink model has been built.

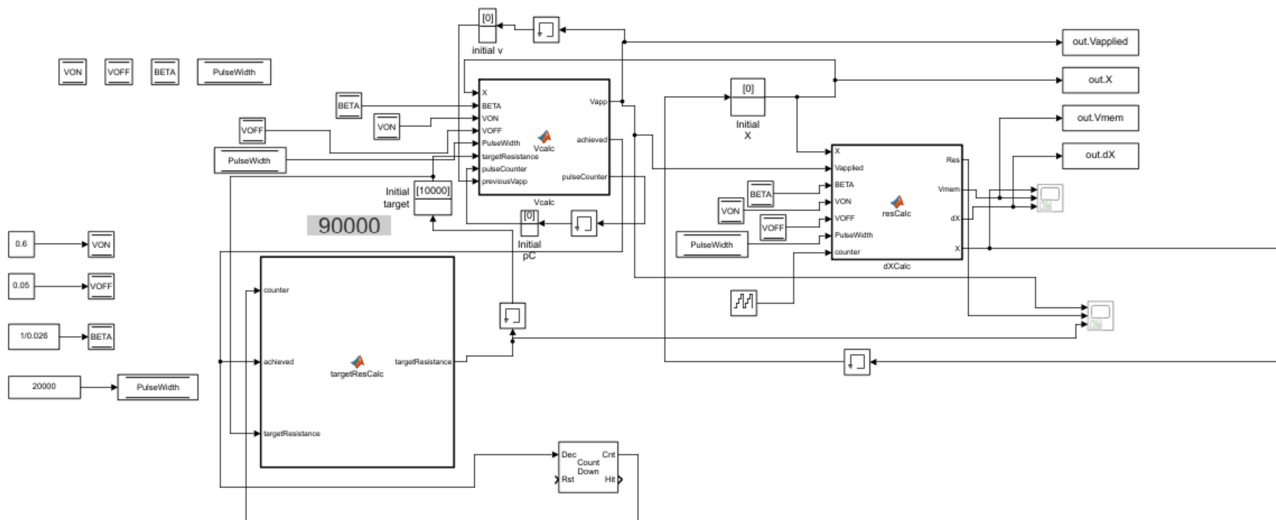


Figure 3-18: Model built in Simulink

The Simulink model is made out of three main blocks: A block that generates setpoints (*targetResCalc*) in ohms, a block that calculates the appropriate applied voltage (*Vcalc*, the controller) and a block that generates normal distributed changes in resistance according to the MSMM (*resCalc*). This framework will be used to explore controller approaches.

In Fig. 3-19, it can be seen that for the Simulated data, the applied control action did not have to reach values as extreme as -$\pm 1.5V$ to achieved the desired state. There are several reasons for this:

- In this simulation, there are no pulses that result in no resistance change at all, since that behavior is not reproduced by the MSMM. Situations where voltages as high as 0.7 V are applied to a memristor and no change occurs are not contemplated by the model. Also, something that is not seen in the plot is that the desired state is achieved faster.
- The model does not replicate all the non-idealities of the memristors, although it serves the purpose of giving a rough estimate. One of such non-ideal behaviors is their tendency to "get stuck" in certain states (mostly extreme states, HRS and LRS), which requires stronger control actions to get out of them.
- The real data seems more stochastic than the model-generated data, which makes the predictions from the model inaccurate more often on real data than in the simulation.

Despite all of this, the general trend of memristor's behavior seems to be captured by the model.

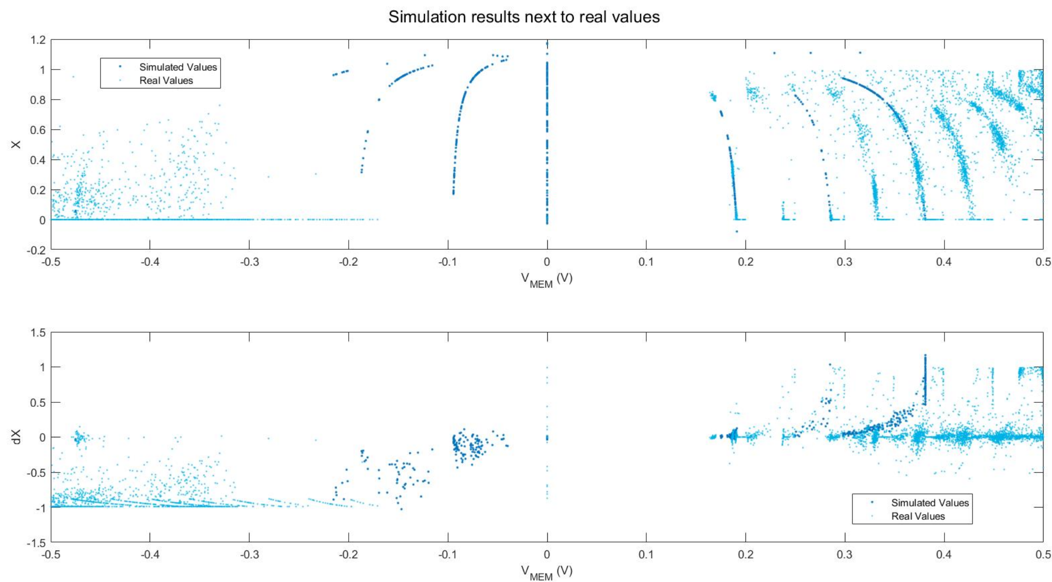


Figure 3-19: Simulated X vs Voltage(V) and dX vs Voltage(V). Real values from Experiment 2 have been added as a reference. This further validates the functions written based on the Metastable Switch Memristor Model.

Finally, the exponential curves that were shown in the previous section are successfully mimicked by the simulation. In this case, more of the curves from the real data are reproduced because of the simulation also adding some stochasticity to the predictions via the block *ResCalc*.This is a point in favor of the model of choice.

In Fig. 3-20 a timeseries of the Simulink simulation is shown.

The function blocks just explained will be reused outside of the Simulink environment to simulate different Neural Networks made out of memristors, where the behavior of the memristors will be calculated with these functions.

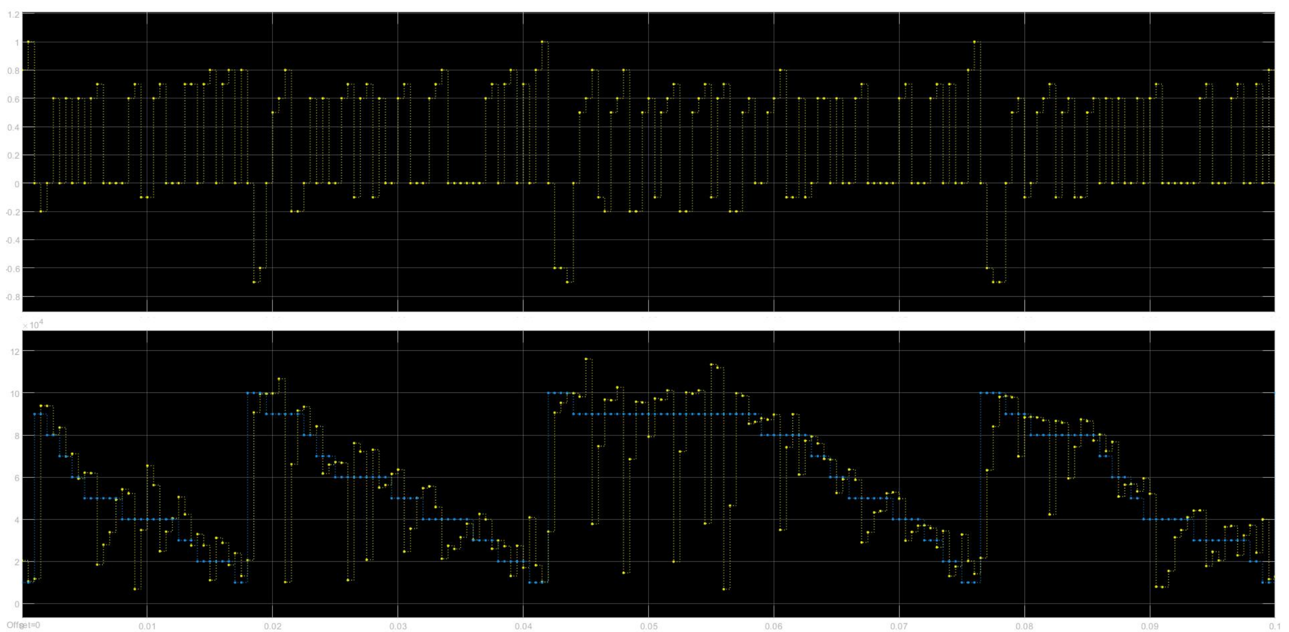


Figure 3-20: On the plot above: V_{MEM} over time. On the plot below: Resistance setpoints (blue) and actual read resistance (yellow) over time.

3-5 Conclusion

This chapter aimed to model the memristors so that they can be controlled faster. In these experiments, either a list of setpoints ranging from LRS to HRS was achieved by means of a lazy controller or a set of pulses of different magnitudes were applied to the memristors. Let us now answer the questions presented at the beginning of the chapter and assess whether they have been answered.

- What model can be used to predict the behavior of the memristors? Many models can be used, but the MSMM, proposed by Knowm, will be used. It yields valid results and it succeeds to capture the exponential patterns found in the data.
- **Does multilevel operation make sense for memristors?** Yes, but it comes at the cost of stochasticity. All setpoints given in Experiment 1 were achieved with 5% accuracy, but sometimes it took many pulses to get there.
- **How many levels does it make sense to use in the memristor, for a neural network?** Steps of 10 $k\Omega$ seem to be possible, according to the experiments. That means that for the safe range of operation of the memristors, there would be about 8 states. However, this is a tricky question due to the fact that to achieve those states, sometimes it is necessary to oscillate around the desired state a few times by applying positive and negative pulses until the desired resistance is achieved (overshooting). But by that same reasoning, in theory any resistance value could be achieved if enough pulses are applied. Since it is not necessary to know how many states are possible to apply a writing pulse to a memristor, and no pulse can always cause the same change in resistance, the Neural Network will just be implemented without supervising if the exact necessary resistance change took place. Only the evolution in accuracy of the network will be looked at.

Chapter 4

Memristors in Neural Networks: Introduction

The next steps will be to design and simulate a crossbar-like architecture that can be tuned and used as a small neural network. First, some findings will be introduced from the current State-of-the-art. Then, the design choices will be explained and the implementation will be presented. Finally, some results will be shown.

This section describes different learning rules that can be used in a Neural Network (NN) implemented with memristor-based circuits. The focus of this section will be on depicting the state of the art of this question, and on explaining the necessary concepts that will be used further on simulation. In order to tackle this, the research is going to be divided into two parts: Unsupervised Learning and Supervised Learning.

4-1 Unsupervised Learning

Unsupervised learning algorithms for memristors are attractive because they are far more biologically plausible than backpropagation, which is merely a mathematical concept where a loss function seeks to be minimized. As a matter of fact, the way the human retina works and recognizes edges and objects in the vision field can be easily compared to how an unsupervised algorithm works. The idea of having an algorithm that can find structure in a set of data by itself, or can learn to perform a task without having to provide it with a loss function is at the very core of neuromorphic computing. Every biological creature seems to present some form of supervision during its life while learning, but its workings have not been unveiled yet.

The main application of these learning techniques are Principal Component Analysis and Clustering. Among them, Unsupervised implementations of Spiking Timing-Dependant Plasticity (STDP) are often suggested as suitable for memristor-based circuits. They require less additional hardware for their implementation, although they seem to achieve a lower accuracy. For example, in [5] 235200 neurons have to be used just to achieve 93.5 % accuracy on handwritten character recognition with STDP, which can be achieved with 310 neurons in an Artificial Neural Network (ANN). However, the algorithm requires no supervision, which

is quite remarkable. Because the aim of this thesis is to achieve a minimal, simple working concept, other unsupervised algorithms than STDP will not be studied.

Let us see how implementing unsupervised STDP is possible.

4-1-1 Unsupervised implementation of the Simplified Spiking-Time Dependant Plasticity Learning Rule

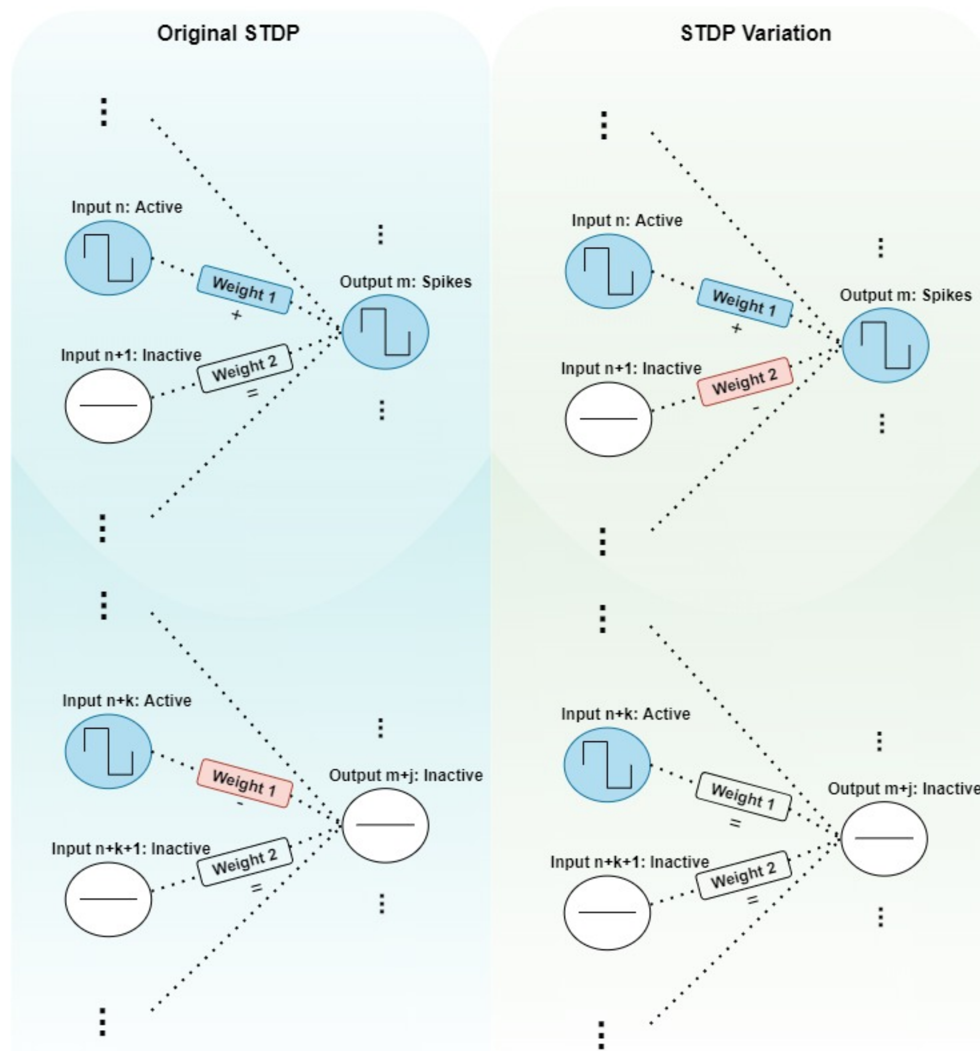


Figure 4-1: STDP Concept vs the implemented version in [5]. The plus sign (+) under a weight means that the weight increases in that situation, the minus sign (-) that it decreases and the equal sign (=) that it remains the same.

Even though STDP is not a learning rule per se, there are ways to use it as such. Examples of this can be found in [5], [66] and [67], although only a simulation of it is run (no physical implementation) and the hardware is not fully specified in all three cases. The circuit

uses Complementary Metal Oxide Semiconductor (CMOS) technology as inputs and output neurons and memristors as synaptic connections between them. The particular variation of STDP that was implemented can be seen in 4-1.

The inputs in all three mentioned papers are coded as square waves of different frequencies. The information is in this case encoded in the signal's frequency, and the output neurons implemented follow the *leaky integrate and fire* model, a variation of 1-4 (L). Let us explain more in detail what a Leaky Integrate-And-Fire (LIF) neuron does.

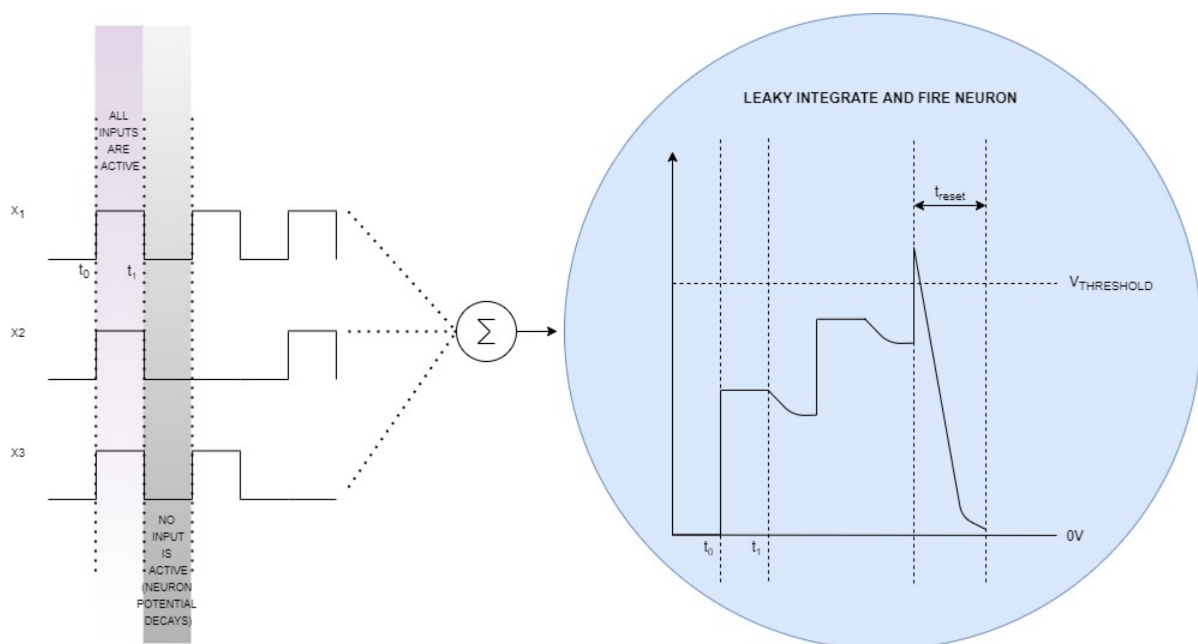


Figure 4-2: Visual explanation of the LIF neuron. When being excited by its inputs, the neuron increases its potential. If it reaches its own firing threshold, it resets its potential to zero. While the neuron is resetting its potential, it will not respond to other excitatory inputs.

Because the input signal is presented by means of a train of pulses and not one unique voltage peak, the output accumulates potential by adding up all the inputs related to one sample. Then, it will fire provided that a certain threshold is reached. Because it is leaky, the potential of the output neurons will slowly decay over time. This makes it easy to reset the potential of the neurons between different samples of a data-set, just by waiting. During that delay, the potential of all neurons will decay until it reaches zero or close to zero.

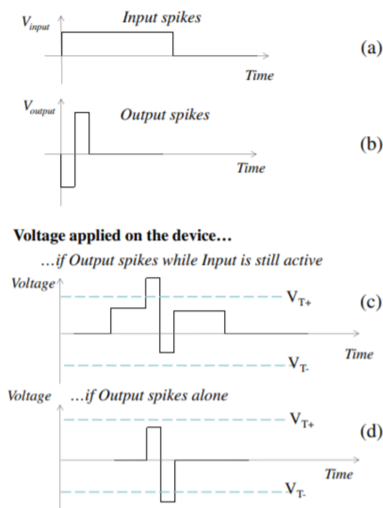


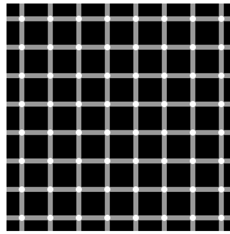
Figure 4-3: Behavior of the system in [5]. An output spike will only increase a synaptic weight its input is also active. Otherwise it will decrease. The input spikes are longer than the output spikes. Leakage is not explained by this plot, it is added to the simulation via external equations.

The way STDP is achieved in [5] can be seen in 4-3. Output neurons produce a slightly negative pulse when spiking, so that for any input that is not active at that time the weight connecting them decreases. However, if the input is also active when this firing happens, the resulting output pulse will be high enough to increase the synapse strength between those two particular neurons. The leaky behavior in this case is added by decaying all synaptic weights slowly over time, so that if they have not been strengthened for a long time they will gradually lose relevance.

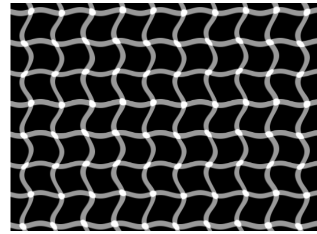
A few more concepts are included in [5] and [66]:

- **Homeostasis**: STDP by itself does not ensure that the threshold for the neurons to fire is the appropriate one. Different neurons may need different firing thresholds within the same network. A way to tackle this problem is to have a varying threshold that is dependent on the firings per time unit. Also mentioned in [22]. This would add complexity to the hardware, so it was initially ruled out for this thesis.

- **Lateral inhibition or *Winner-Takes-All* topology:** Also used in the human brain. When an output neuron fires, it prevents the neighboring output neurons from firing, which is useful in classification problems where the classifying label must be unique. This is a complex concept to apply and the human brains seems to not always use it. A well-known example of this can be seen in 4-4. More will be said about how this was used in this thesis later on.



(a) When looking at this image, some dots at the intersections of straight lines seem darker than others. According to Baumgartner, this happens due to adjacent retinal ganglion cells not being allowed to fire at the same time.



(b) Lateral inhibition seems to not happen so strongly when using curved instead of straight lines for the grid, which suggests lateral inhibition happens in a selective manner: it is one of many possible filters our brain may apply to visual data

Figure 4-4: Scintillating Grid Illusion [11]

4-1-2 Accuracy challenge

The main disadvantage of using Unsupervised Learning is that it is slower to converge than Supervised Learning (which is logical) and that the final recognition rate will most likely be lower. In [5], when using 10 output neurons to identify 10 different handwritten digits, the final recognition rate was 60%. Only with hundreds of output neurons did the recognition rate rise above 90%. In [66], the achieved accuracy is above 90% only for a network with thousands of output neurons. That means that there would be different versions of every number, which is true to a certain extent, although it makes it harder to interpret the results that the network converges to. Maybe using a network with multiple layers could improve the results for a network with 10 output neurons.

4-2 Supervised Learning

One step more complex than supervised learning in terms of implementation, supervised learning would require some external supervisory signal. Three types of back-propagation have been found as relevant to this thesis in Literature: Supervised STDP, Gradient Descent and the Manhattan Rule.

4-2-1 Supervised implementation of the Simplified Spiking-Time Dependant Plasticity Learning Rule

Implementing a Supervised version of STDP can be as simple as, given a neuron, instead of adding up all its input voltages and checking for a firing, providing the necessary firing voltage on the output if that neuron is supposed to fire. This may take a lot of time or not converge to a solution due to stochasticity, the weight updates being coarse in memristors and the difficulty to interpret when the network is stuck in a local minimum that is still far from a desirable optimum.

In [68], a combination of Supervised and Unsupervised STDP is used for handwritten character recognition. The supervised version of STDP is implemented as follows:

- If a neuron fires and it is supposed to fire, the pulse at the outputs increases the weight of the active inputs.

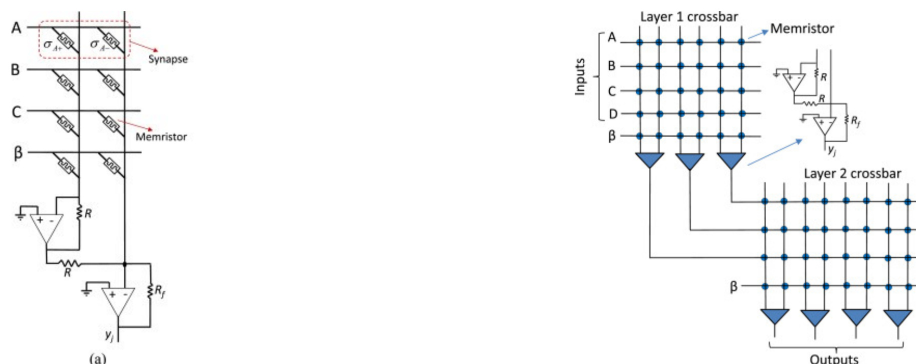
- If a neuron is not supposed to fire:

  - And it fires, the weights of the active inputs are decreased.
  - And it does not fire, the voltage at the output has no effect on the synapses connected to it.

This is used in combination with one layer of the Unsupervised version of STDP.

4-2-2 Online Back-propagation Rule with Stochastic Gradient Descent

A supervised learning circuit was proposed in [12], which implements stochastic gradient descent back-propagation in a more classic ANN manner. The term stochastic comes from the fact that the weight update happens after every sample is presented to the network, so every error is calculated from a random sample of the entire data-set, and then the derivative with respect to the weights is estimated as explained in 1. A possible alternative to stochastic back-propagation is batch back-propagation, where the error is calculated over a larger subset of the data (a batch) and the weight update happens after the whole batch has been presented to the network.



(a) Memristor based circuit with four inputs and one output. On second circuit, wire resistance has been added.

(b) The circuit proposed in [12] is meant to be modular.

Figure 4-5: Architecture proposed in [12]

Fig. 4-5a explains a simplified overview of their proposed architecture. A key element of their proposal is a differential synapse based on two memristors. This can be seen in every row (A, B, C, D and β). As explained in [12]:

$$
\begin{aligned}
y_j =& R_f(I_A + I_B + I_C + I_\beta) \\
I_A =& A(\sigma_{A+} - \sigma_{A-}) \\
I_B =& B(\sigma_{B+} - \sigma_{B-}) \qquad \implies \qquad y_j = R_f(A(\sigma_{A+} - \sigma_{A-}) + ... + \beta(\sigma_{\beta+} - \sigma_{\beta-})) \\
& ... \\
I_\beta =& B(\sigma_{\beta+} - \sigma_{\beta-})
\end{aligned}
\tag{4-1}
$$

Where σ refers to conductivity. A,B and C are inputs and β is a bias input. This array structure implements a physical neural network where the sum of weighted inputs happens at a circuit level via sums of currents. Every row has a total current of $V \cdot (\sigma_+ - \sigma_-)$. This makes it possible for the weight to be either positive or negative, which doubles the range of possible synaptic weights with respect to the case in which one memristor per synapse is used. The physical meaning of having a negative weight is giving less and less statistical probability for an input to be relevant in the firing of a neuron, but also reducing the chances of firing in that particular neuron. More importantly, the output voltage range is also doubled. This is somehow similar to the architecture proposed by Knowm for simple experiments, although their circuit does not add up weights of different synapses together. Only one synapse can be chosen at a time, which reduces the amount of possibilities but could be improved with external circuitry. This circuit presents a more powerful proposal while still being realistic to implement, so its design will be used with minor alterations.

Another relevant element of the design in 4-5a is the presence of two operational amplifiers, that effectively behave as an activation function. Knowing the input voltage range, the operational amplifier's supply voltages V_{DD} and V_{SS} can be chosen accordingly (0.5 V and -0.5 V, for example). This activation function can be approximated by a sigmoid or tanh activation function (1), which makes it easier to calculate the weight updates during backpropagation. Then, not only are the firings of the neuron easily identified and processed, but also the output is within a range that can be used as an input for another neuron. This is illustrated in Fig. 4-5b. Here, β is a bias term.

Then, the learning rule proposed by Hasan et al. is simply backpropagation, with

$$
\delta_j = (t_i - f(y_j))f'(y_j) \tag{4-2}
$$

Where δ_j refers to the calculated error function for gradient descent, t_i to the output neuron's target, y_j to the output before the operational amplifier($f(y_j)$ would be the voltage after the operational amplifier, that is, after being passed to the activation function) and f' to the derivative of the activation function. This error is then back propagated, and the weights are updated. The specific circuitry used is not specified, only the main necessary blocks: An adder, a multiplier and a buffer to store a look-up table for the derivatives of the activation function. The learning takes place in two different phases: one for a forward pass (where the errors would be computed) and one for a backwards pass (this is the back propagation phase). In the back propagation phase, the weight update occurs, by applying voltage to the columns of the circuit instead of the rows.

The weight update has to be divided by two so that the first memristor of the differential neuron is increased half the total update, and the second memristor decreased by the same amount (σ_+ and σ_- from Fig. 4-5a). That way, the synapse has been modified properly. This

proposal is in general interesting because it does not make use of a too complex or intricated circuitry (besides the backpropagation, but that is to some extent solved by a look-up table), and the crossbar consumes not more than 192.8 μW for any of the experiments it has been tested with.

Finally, it is important to note that the work of Hasan et al. timing encoding is not used, which drastically reduces the range of input values, and turns the network into a Binary Neural Network. However, it would be possible and maybe desirable to merge this idea with the concept of Spiking Neural Network.

4-2-3 Manhattan Rule-based Back-Propagation

Some may argue that applying a precise analog weight update is too complex, and as an alternative to that, there is the Manhattan Rule: applying always the same update to the weights no matter how large the error is. This solution would take longer to converge, since the specific error is not calculated. Only the sign of the error would be used: if the error is negative, then the update on the total synapse is negative; if the error is positive, the update on the total synapse is also positive. This forces the weight updates (or optimization steps) to be sufficiently small so that a satisfying solution can be found.

This type of approach happens to align more with the philosophy of Knowm, since it is already implemented in their testing framework: having a small set of instructions that can easily be applied on the differential neurons and over time converge to the desired result. It was suggested in [69]. Unfortunately, no circuitry or hardware architecture is proposed with it, only the concept itself is presented.

An On-Chip Supervised Learning approach with the corresponding supporting hardware is proposed in [70], with what they call a “binary” weight update after presenting every pattern. This can also be seen as applying the Manhattan stochastic learning rule. This work is more specific about architecture and also works with differential (signed) synaptic weights.

In [70] the design presented is meant for Ultrahigh Density crossbars, which use thousands of memristors, and it uses a quite complex procedure for the learning process, in which quite some more additional circuitry than in [12] is necessary. It is not realistic to attempt to build anything similar to an Ultrahigh Density Crossbar like the one proposed by Chabi et al, so this design will not be considered.

Finally, the Manhattan rule having a fixed step size makes it easier for the problem to oscillate around optima without ever achieving them.

4-3 Summary and Design Choices

Some important takeaways from this section:

- Using two memristors per synapse is a widely used workaround to compensate for the lack of resolution of memristors.
- The frequency timing encoding used in [66] and [5] is complex to implement and to debug in a real implementation. It requires very precise clocks.

- Unsupervised STDP algorithms are not easy to make work, but they require less interaction (and thus, less additional hardware) than Supervised Learning, once they are tuned properly. Before that happens, the firing thresholds must be adjusted and some type of lateral inhibition may be necessary.
  - The simplest form of lateral inhibition would be to simply take the maximum output value from the output layer.
  - A LIF neuron is often used for the outputs, which adds hardware complexity.
  - These algorithms often require many neurons, which in turns results in many synapses and therefore many memristors.
- Supervised Learning algorithms with either Stochastic or Batch Gradient Descent are easier to make work in simulation because they have been used profusely throughout literature. However, they are less bio-plausible and adding back-propagation adds complexity to a hypothetical circuit. The simplest solution for this is a look-up table.

Therefore, a simulation is going to be implemented on Matlab with the following choices:

- Two memristors per synapse will be used.
- A circuit similar to the one proposed in [12] will be used, with minor variations. This allows for implementing Unsupervised STDP, Supervised STDP and a Supervised algorithm with backpropagation.
- A simplified version of STDP will be attempted without frequency timing encoding, in both Supervised and Unsupervised variants.
- Supervised learning will also be implemented, to have a better idea about how the low resolution of memristors has an impact on the convergence of the network.
- Lateral inhibition will only be present in its simplest form: By taking the maximum of the output layer as the only firing.

Next, the circuit that is going to be used is explained.

Chapter 5

Memristor Circuit Design

A circuit has been designed and simulated on the software LTSpice for the Neural Network implementation. This circuit is based on the one in Fig. 4-5 [12], with the difference that no time encoding of the input signals is used, the output levels are slightly different and the distribution of the switches is not the same. This circuit (the one in [12]) was chosen as a basis because it was the most feasible one found in literature that could be physically implemented on a Printed Circuit Board (PCB).

5-1 Main consideration when designing a memristor-based chip for Neural Networks

The key features of the circuit simulated on LTSpice are:

- Memristors are represented by resistors, their behavior is not simulated by LTSpice. The focus was put on the elements around them and on getting rid of voltages or currents that might change the resistance of the wrong memristor.
- The model used for the switches on LTSpice is an ideal model. However, since the type of switches used by Knowm was likely to be used in case of a physical implementation of the circuit, their instructions to model the capacitance of the switch by placing a 140 pC capacitor between one of its terminals and ground was followed.
- Each weight is implemented by two memristors, which can be accessed individually via multiplexing during the writing phase.
- Multiplexers and switches are used to determine which signals are necessary depending on whether a writing or a reading operation needs to be performed.
- As suggested by the memristor manufacturer, a series resistor of 5 $k\Omega$ was used as a simple current limiter after the input.

- It was kept on mind at all times that the amount of necessary control signals in the designed circuit should be kept around 16, since the setup provided by Knowm (containing an oscilloscope, switches and memristors) has 16 I/O signals.

5-2 Development

The inputs on rows and columns of the memristor matrix have been simplified so that every memristor can be accessed directly via the rows. In the circuit presented in [12], voltage is applied on either the columns or the rows depending on whether a reading or a writing phase is happening, respectively. Also, they used no switches since they claim that during a writing operation, the voltage that reaches the memristors that are not being written to is too low to have an effect on them. Their threshold is quite high (1.3V) compared to the one in the Knowm devices (0.26 V), so their reasoning surely was not valid for this thesis and switches were added to the design.

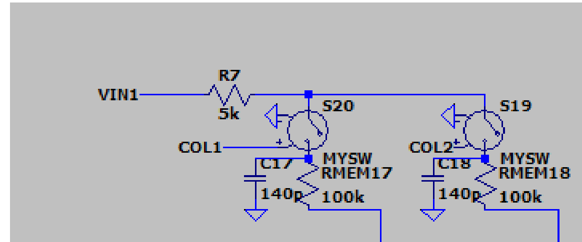


Figure 5-1: Every input in the circuit designed (V_{IN1}, in the image) is connected to two memristors (R_{MEM17} and R_{MEM18}). The SPST switches are used to choose which one will be modified during the writing phase. During the reading phase, both signals COL_1 and COL_2 are connected to 5V, closing the switch so that the reading voltage reaches both memristors.

The block shown in 5-1 can be used as many times as inputs are necessary. Then, an amplifying stage sums up all the inputs multiplied by the differential weights similarly as it was explained in 4-1.

Let us now focus on the amplifying stage, seen at the end of an input line. Following the naming convention used on Fig. 5-2 and taking the left circuit, its output will be derived. For every differential memristor circuit, the left side is considered the positive side of the weight, and therefore the magnitudes from the resistors on that side will have a minus (-) sign. Likewise, conductances/resistances from resistors on the right side make up the positive side of the weight and will have a plus (+) sign.

$$\begin{aligned} \frac{0-(-R_{13}\cdot\sum_i(VIN_i\cdot\sigma_{-i}))}{R_{15}} &= \sum_i(VIN_i\cdot\sigma_{+i})+\frac{V_{OUT}}{R_{14}} \\ \frac{R_{13}}{R_{15}}\sum_i(VIN_i\cdot\sigma_{+i})-\sum_i(VIN_i\cdot\sigma_{-i}) &= \frac{V_{OUT}}{R_{14}} \end{aligned} \tag{5-1}$$

For the left and the right side of the amplifying stage to have the same importance, R_{13} and R_{15} must be equal.

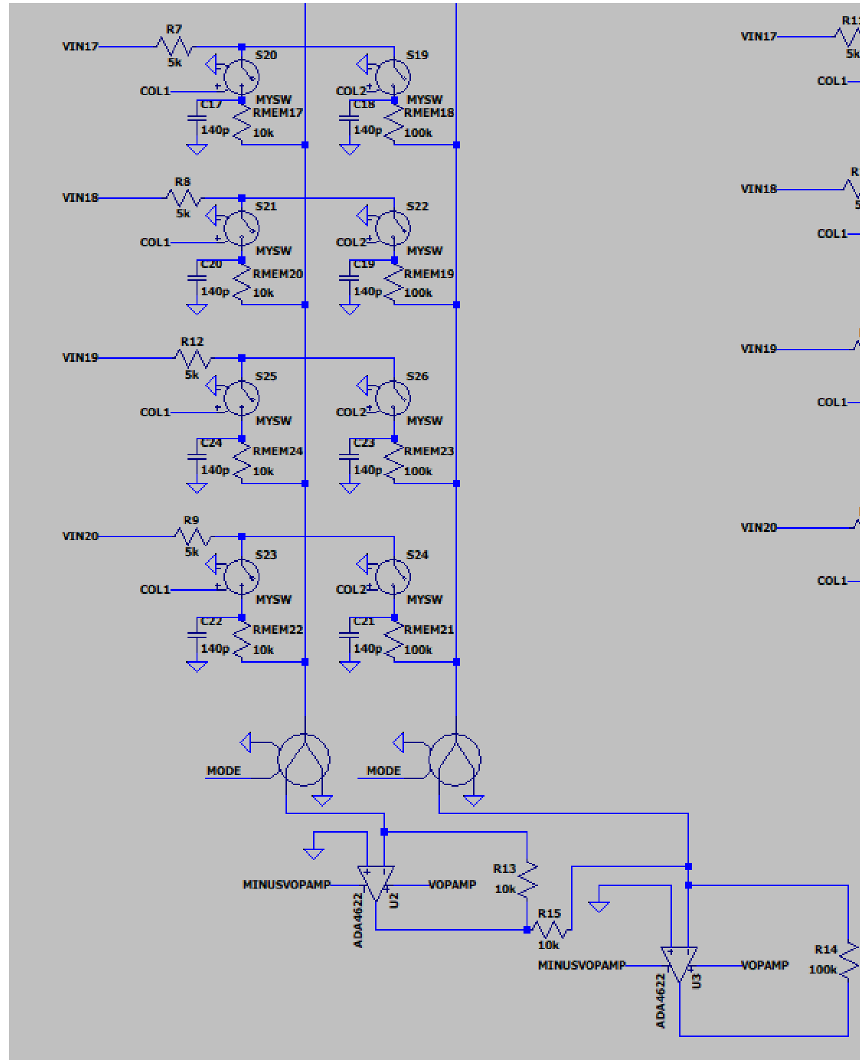


Figure 5-2: Multiple inputs stacked together and connected an output neuron (the amplifying stage)

$$V_{OUT} = R_{14} \cdot \sum_i (VIN_i \cdot (\sigma_{i+} - \sigma_{i-})) = R_{14} \cdot \sum_i \left(\cdot VIN_i \cdot \left(\frac{1}{R_{i+}} - \frac{1}{R_{i-}} \right) \right) \tag{5-2}$$
$$R_{i+}, R_{i-} \in [20, 100] k\Omega.$$

Then, R_{14} can be used as the gain of the circuit. If all inputs were active and all weights were maximum, the sum of all their voltages multiplied by the differential weights would be $n \cdot 0.1/90000$, where n is the number of inputs. Increasing that by a factor of 10000 makes the voltage output easier to work with, so a resistor of 10 kΩ will be used.

Then, as a result,

$$
\begin{bmatrix} VOUT_1 \\ VOUT_2 \\ \vdots \\ VOUT_i \end{bmatrix} = \begin{bmatrix} \sigma_{11+} - \sigma_{11-} & \dots & \sigma_{1M+} - \sigma_{1M-} \\ \sigma_{21+} - \sigma_{21-} & \dots & \sigma_{2M+} - \sigma_{2M-} \\ \vdots & \vdots & \vdots \\ \sigma_{N1+} - \sigma_{N1-} & \dots & \sigma_{NM+} - \sigma_{NM-} \end{bmatrix} \begin{bmatrix} VIN_1 \\ VIN_2 \\ \vdots \\ VIN_M \end{bmatrix} = \\
= \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{N1} & w_{N2} & \dots & w_{NM} \end{bmatrix} \begin{bmatrix} VIN_1 \\ VIN_2 \\ \vdots \\ VIN_M \end{bmatrix} \quad (5\text{-}3)
$$

Where V_{OUT} would be the voltage at the output of the operational amplifier if no clipping was applied to the signal (and therefore it is not a voltage that actually happens at any node of the circuit).

The activation function is defined by this piece-wise equation, realized by the operational amplifiers (which clip their output signal between the levels provided as their voltage supplies).

$$
F(V_{OUT}) = \begin{cases} -V_{SS} & \text{if } V_{OUT} < -VSS \\ V_{OUT}/\alpha & \text{if } -V_{SS} <= V_{OUT} <= V_{DD} \\ V_{DD} & \text{if } V_{OUT} > VDD \end{cases}
$$

Where V_{OUT} would be strictly the result of adding all the inputs together multiplied by their weight, V_{SS} is the negative supply fed to the amplifiers (MINUSVOPAMP in Fig 5-2) and V_{DD} the negative supply (VOPAMP in Fig 5-2). The factor of α comes from the resistor used in the amplifying stage. This activation function, as explained before, can be approximated by both a tanh or a sigmoid, and then their derivatives can be used during the calculation of weight updates during backpropagation (Fig. 5-3). The accuracy of both approximations is not perfect, but since the memristors do not have a too high resolution it should not have an impact on the overall accuracy. It is possible to tune the slope and the limit levels of the activation function by adjusting the supply voltage of the operational amplifier or the gain (via R_{14} or R_6, in Fig 5-2) By connecting the negative supply of the operational amplifier to ground, the output signal can be also clipped at 0, which also makes it possible to implement a ReLU activation function.

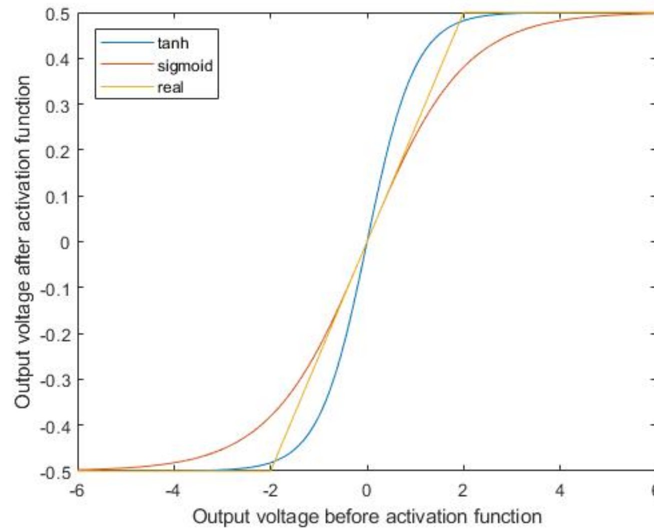
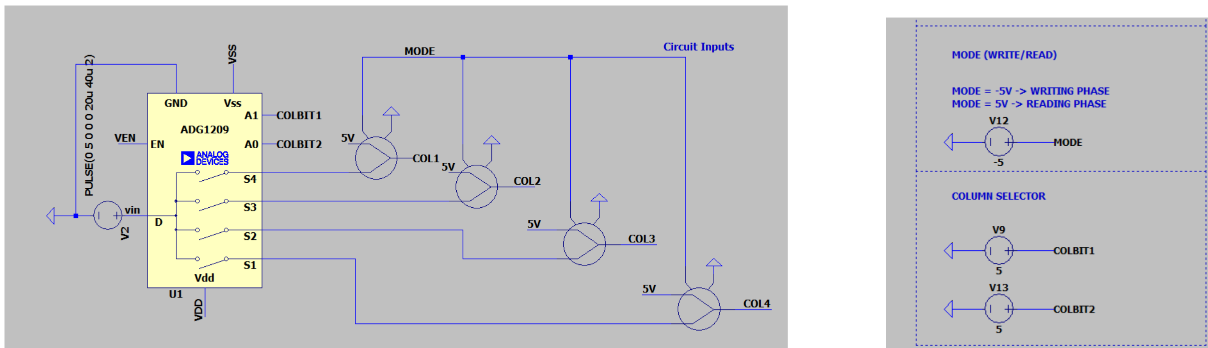


Figure 5-3: Transfer function of the amplifying stage clipped at 0.5 and -0.5 V, compared to scaled tanh and sigmoid with offset.

Some signals were multiplexed, which can be seen in 5-4.



(a) Multiplexing of the switch signals. During a reading operation, the MODE node is at 5V and the SPDT switches connect all the COL signals to 5V. This closes all the switches so that all memristors can be read at the same time. During a writing operation, only one column of switches a time is closed. In this case, two different groups of inputs connected to two different neurons are multiplexed to reduce the number of control signals.

(b) Signals used to choose which one of the two memristors that form a synapse is being written to during the writing phase. $COL1$ to $COL4$ can be seen in 5-2. COLBIT1 and COLBIT2 both have a high value in this case (5V) so switch S4 from the multiplexer is closed.

Figure 5-4

An example of a typical output of a reading operation (the voltage after the second amplifier stage) can be seen in Fig. 5-5. In this case there were 12 active inputs. For this particular simulation, the differential weight was maximum for every memristor pair ($R_+ = 10k\Omega$,

$R_{-} = 100k\Omega$).

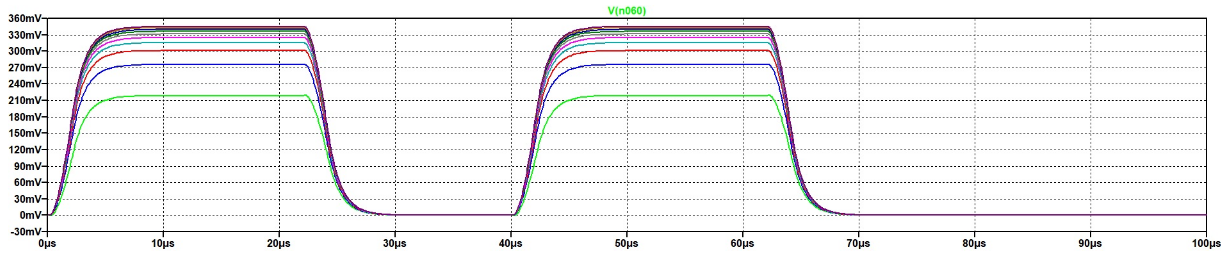


Figure 5-5: V_{OUT} during reading operation with 12 active inputs. All left weights were set to 10 $k\Omega$. All right weights were set to 100 $k\Omega$ (setting the weights to their maximum value), except for two of them. A parametric sweep was done over those two memristors, changing their value from 10 $k\Omega$ to 100 $k\Omega$. Hence the multiple colored lines.

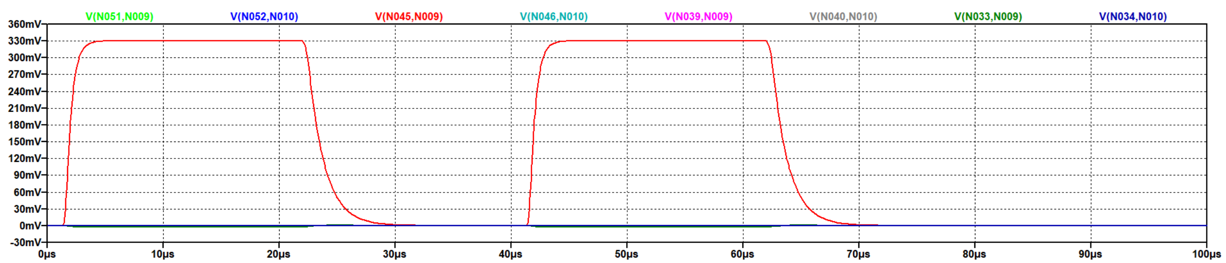


Figure 5-6: Voltage drop on several memristors while writing one of them. All weights were set to maximum. The red line indicates the voltage drop on the memristor that is being accessed, all the other voltages (negligible) correspond to other memristors in the circuit and they indicate that the other memristors will not be accidentally rewritten during this operation.

To make this circuit work, the number of control signals depends on the number of inputs, number of rows and modes of operation. With 12 inputs, 2 rows (1 layer, 1 output neuron) and two modes of operation (reading and writing), the number of control signals is 15. A way to improve the circuit would be to have a separate signal for every switch in the array, which was not done to have less control signals. This would isolate the memristors from sneak path currents when writing to any of them. Sneak path current is any current in a crossbar array that reaches a memristor that is not being programmed. It can be a problem if such current creates a voltage drop across the memristor that is enough to change its resistance. In this simulation, no voltage greater than 0.1 V arrived to other memristors while one of them was being written to. This is sufficient for these memristors, since 0.1 V is a reading voltage.

Chapter 6

Neural Network Simulation

A very standard application will be used to create a proof of concept based on how memristors (specifically the Knowm ones) work, but also to show that such a Neural Network can learn and converge towards an acceptable amount of accuracy as well. Using the Metastable Memristor Model to predict the behavior of the memristors, and the principles of operation of the memristor circuit explained in the previous section (which we will call here "Memristor Matrix"), two different neural network models and a few topologies will be simulated on Matlab.

It is important to note that the source of interest of this approach is not the accuracy itself, but the novelty of memristors, the fact that they can keep their states, the fact that calculation and memory retrieval happens simultaneously in memristor-based circuitry and their ability to be put in matrix structures to perform highly parallel computations. As a matter of fact, these simulations have made use of the GPU of a computer when performing operations such as the calculations of the outputs. This is somewhat comparable to using memristors, in terms of speed and parallelization.

The data-set of choice is the MNIST database, which contains a training set with 60000 handwritten characters and a testing set with 10000 handwritten characters. Every character is in an image made out of 20 by 20 pixels, coded in gray-scale. The goal of the network is to recognize each character (a number from 0 to 9) from that gray-scale image.

The MNIST database is oftentimes used as a benchmark for a learning algorithm, and there is extensive documentation about the performance it can achieve with different types of Artificial Neural Network (ANN). The goal of this chapter is to find out whether the Network will be able to learn given the stochasticity of the used memristors, which will make the weight updates coarser.

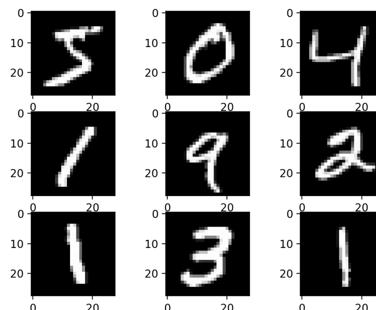


Figure 6-1: Some examples of handwritten characters from the MNIST database [13]

6-1 Spiking Timing-Dependant Plasticity (STDP): Hebbian and anti-Hebbian learning

The first type of Neural Network that was attempted was based on STDP, in both the Supervised and Unsupervised versions as mentioned in Chapter 2. Since STDP is often mentioned in literature to be specially suited for memristors, his subsection aims to find out whether such a learning approach in a neuromorphic network would be able to learn the recognition task as addressed in this chapter.

The main principle of STDP in memristor-based circuits is that a Network can learn the patterns in a set of data just by presenting the data to the Network. From this principle, Supervised and Unsupervised STDP can be derived.

6-1-1 Supervised STDP

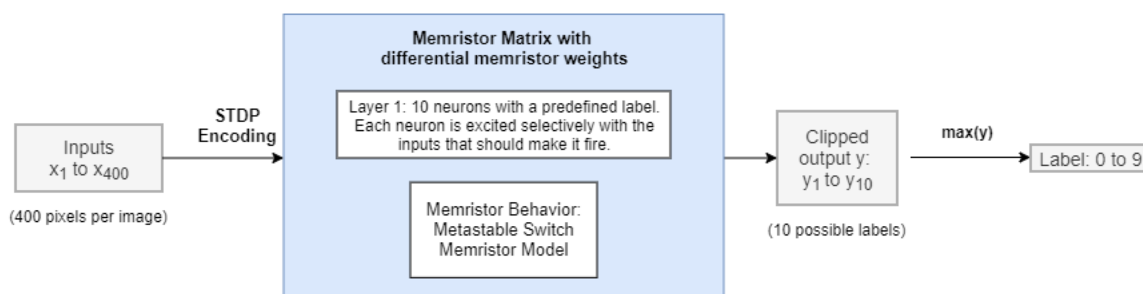


Figure 6-2: Supervised Implementation of STDP. Each neuron is only excited with the inputs it should fire with. For example, a neuron that should recognize number 6 is always excited with representations of number 6 encoded between -0.11 and 0.26 V, which are voltages that can change the memristor's state.

- The inputs were encoded as voltage pulses within a range which is safe to apply to the memristors but that can change their state. In this case, both the negative and the positive threshold voltages were used as limits for the range. The inputs were then coded between -0.11 V and 0.26 V. A negative input makes the weight connected to it decrease slightly, and a positive one makes the weight connected to it grow.

- The inputs were applied to the network selectively: Since the labels were known beforehand, each neuron was stimulated with one type of digit. That is equivalent to making that "path" or input combination more likely to cause a firing on that particular neuron.
- The voltage at the output was calculated as the neuron's potential, as seen in 5-3.
- If a neuron fired (the firing level was determined empirically) and was not supposed to, the sign of the encoded input voltage pulses was reversed and then applied to that neuron. That way, the strength of that input combination was weakened.

When trained for only 600 examples, this network managed to achieve about 70 % accuracy, as it can be seen in 6-3.

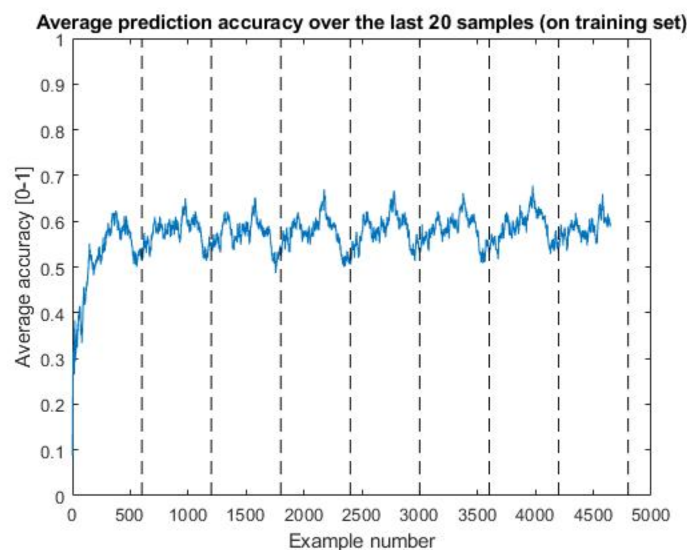


Figure 6-3: Accuracy achieved by using STDP on 600 samples (measured on the training set). Since the accuracy achieved was not high and the sample size was small, these results were not tested on the validation set. The vertical dashed lines indicate epochs.

The performance seems to get stuck. A way of overcoming this problem could be to use varying firing thresholds or lateral inhibition, as done in [68] and [5].

6-1-2 Unsupervised STDP

All neurons are excited with the encoded inputs, using voltage levels that can have an effect on the memristors. Then, a second layer is used to recognize which digit can be associated with every neuron. The same digit may have very different representations or some neurons may be sensitive to particular features of a digit. The second layer is a classic supervised layer with backpropagation.

Both the Unsupervised and Supervised approach are used in [68]. No remarkable performance was achieved with the unsupervised version, but no additional methods such as lateral inhibition or variable firing threshold were used, whereas they were used by Querlioz et al. Both these workarounds are present in the human brain, but establishing a coherent firing rate

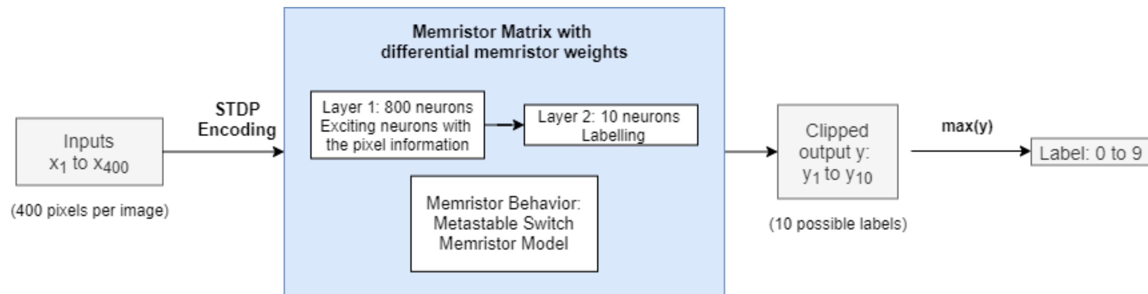


Figure 6-4: Unsupervised Implementation of STDP. Each neuron is excited with all inputs encoded between -0.11 and 0.26 V. Then, a second layer on top of that attempts to classify the obtained data and extract labelling information out of it.

or putting boundaries to lateral inhibition is not straightforward. These methods are always tuned in such a way that a particular problem can be solved, but not necessarily in the same way they would be applied in a biological brain. However, it would be an interesting line of future work to put more time on making an STDP Neural Network work.

6-2 Classic ANN Implementations

There are two distinct phases in this type of Network:

- Reading Phase: The inputs are the pixels of the image, they are encoded between -0.1 and 0.1V, which are acceptable reading voltages according to the manufacturer (they will not change the memristor's state, with high probability).

- Writing Phase: The update of the weights is calculated via backpropagation, knowing that the resistors cannot go below 20 kΩ or 100 kΩ. Then, the voltage that is necessary to apply in order to achieve that weight update is calculated. Since the memristors are stochastic, a second calculation needs to be done afterwards to simulate that stochasticity by generating a normal random number, using μ and σ from the Memristor Metastable Model. The actual weight change is never corrected, since it is assumed that it will always deviate slightly from the estimation. This is also done to not over-complicate the design.

The functions that calculate the voltage that has to be applied and the actual resistance change can be seen in the Appendix. They are extracted from the Simulink Model built in 3-4

Baseline Version If memristors are used as weight storage units, a fine resolution is not possible when it comes to updating the weights, nor so much knowledge about how much exactly that update will be. The simulations make use of the the Mean Meta-stable Switch Memristor Model to calculate the voltage that needed to be applied and the resulting (stochastic) change in resistance. For that reason, a baseline version is calculated for each one of the implemented networks, in which instead of calculating the necessary voltage to be applied

and the subsequent resistance change, it is assumed that the required weight change can be achieved exactly. This version helps set a limit for the performance of a given topology.

In other words, in the memristor simulation:

$$\begin{aligned}\widehat{\Delta W_+} &= \widehat{\Delta\sigma}\left(V_{MEM}\left(\frac{-\alpha\cdot\delta}{2}\right)\right)\\ \widehat{\Delta W_-} &= \widehat{\Delta\sigma}\left(V_{MEM}\left(\frac{\alpha\cdot\delta}{2}\right)\right)\end{aligned} \tag{6-1}$$

Where α is the Leaning Rate, δ is the backpropagation error, $\widehat{\Delta\sigma(V)}$ is the function that estimates the change in state for a given voltage and $V(\delta)$ is a calculation of the voltage that yields the desired state change in average.

On the other hand, on the baseline version:

$$\begin{aligned}\Delta W_+ &= -\frac{\alpha\cdot\delta}{2}\\ \Delta W_- &= \frac{\alpha\cdot\delta}{2}\end{aligned} \tag{6-2}$$

The reason for the plus and minus signs on the baseline version as well are that the weights are still implemented by two memristors, but their behavior is ignored and total control over them is assumed.

6-2-1 Supervised ANN without hidden layers (1 input layer, 1 output layer)

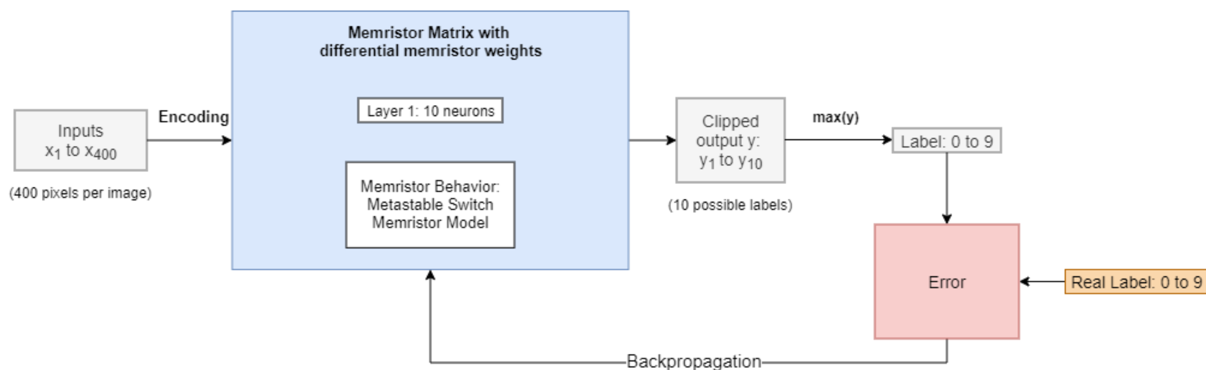


Figure 6-5: Supervised Implementation of an ANN without hidden layers.

Here, a classic Supervised Neural Network with 10 neurons is simulated. This configuration is interesting because of its simplicity. According to LeCunn et al. [71], 88 % accuracy has been achieved before by using one layer in an ANN. In order to test if that can be used as a theoretical limit for the type of circuit explained in the previous section, a baseline example has been coded first that does not take the stochasticity of the memristors into account.

The weights of this circuit are implemented via two resistors in parallel in the circuit. Every weight is made out of two parts, and the difference between them quantifies how strongly the input signal contributes to the output of a particular neuron. They are initialized with

random values between 20 kΩ and 100 kΩ, without making them symmetrical with respect to each other.

For this configuration, the outputs have been clipped between 0 and 40 V, generating a ReLU layer instead of a sigmoid. Its derivative can be easily calculated as:

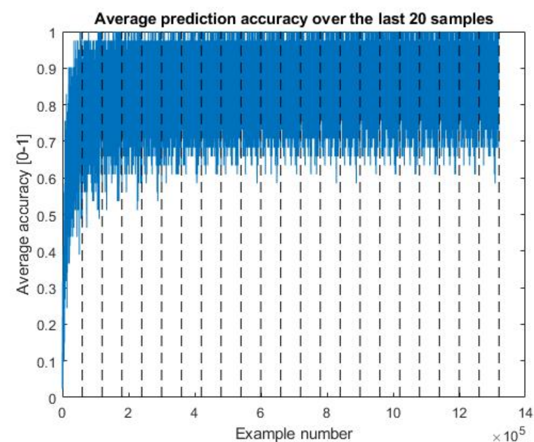
$$f'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{6-3}$$

Baseline

On the baseline version, 88.28 % accuracy is reached on the test set after 16 epochs. A moving average of the accuracy of the network was used (taking into account the last twenty predictions) to find an early stopping point. In the baseline, stopping before an epoch finishes is not so critical, but it is necessary for the simulation with the memristor behavior because their incapability for fine state changes. Since the step is so coarse, after reaching a local optimum the accuracy normally decreases.

True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10
1	899		13	6	1	45	10	1	4	1
2		1068	8	9		12	2		36	
3	7	8	882	29	21	8	14	12	45	6
4	3		16	898	1	45	2	11	27	7
5	1	4	7	1	874	1	11	1	14	68
6	7	7	7	70	14	737	10	4	24	12
7	11	3	19	1	21	36	857	3	7	
8		5	47	4	5	2		892	10	63
9	8	8	10	38	8	36	11	6	828	21
10	7	3	5	19	39	10	1	12	20	893

(a) Confusion Chart for the Supervised baseline Network with 1 layer and Learning Rate of 0.00001. 88.28% accuracy.



(b) Moving Average of the Prediction Accuracy. Accuracy was calculated as a binary value (0 for a correct prediction, 1 for a wrong one) when feeding the images to the network.

Figure 6-6: Success of the experiment

Memristor-based

In this simulation, 82 % accuracy is achieved within one epoch (presenting the whole training set). This may be of course not ideal for character recognition, but it is nonetheless remarkable for a network made out of stochastic devices. Early stopping was crucial to achieving this accuracy, since due to the stochasticity and the coarse update the accuracy decreased after reaching that local optimum.

(a) Confusion Chart for the supervised memristor network with one layer.

(b) Moving Average of the Prediction Accuracy, zoomed in over the area where the local optimum is achieved. The accuracy of every example was calculated as a binary value (0 for a correct prediction, 1 for a wrong one).

Figure 6-7: Success of the experiment.

(a) Effects of letting the network learn for too long (no early stopping). Every vertical dashed line is an epoch. Learning Rate: 0.1

(b) Moving Average of the Prediction Accuracy over one epoch. The accuracy of every example was calculated as a binary value (0 for a correct prediction, 1 for a wrong one).

Figure 6-8: Effects of letting the network learn for too long (no early stopping)

6-2-2 Supervised with one hidden layer

This Neural Network has one hidden layer with 300 neurons, and it is fully connected.

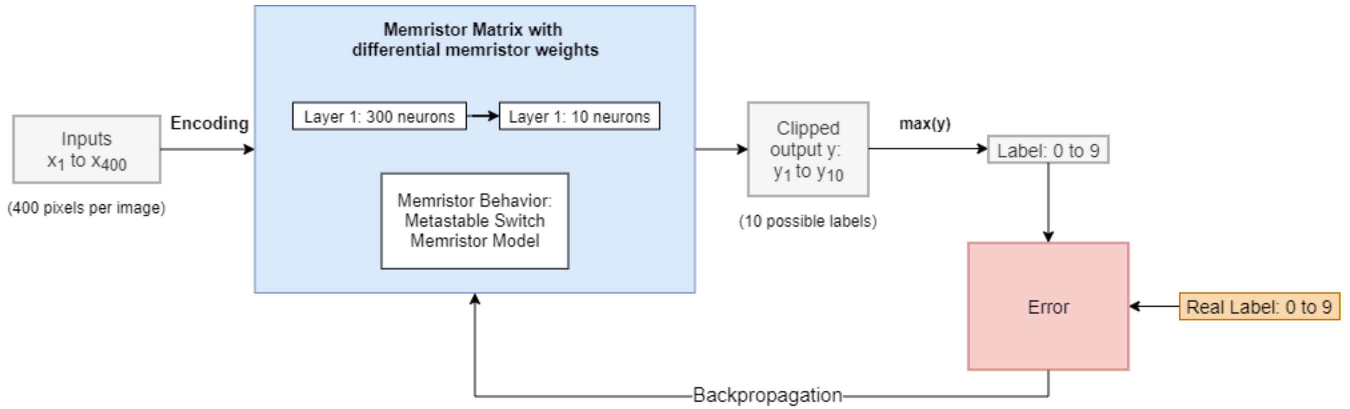
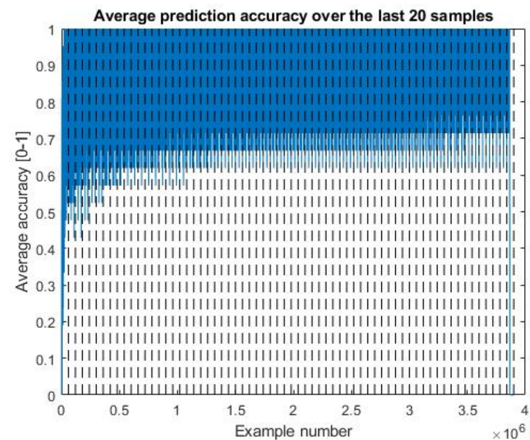


Figure 6-9: Supervised implementation of an ANN with one hidden layer with 300 neurons.

Baseline

True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10
1	965		3	1		2	5	1	3	
2		1116	2	4	1	1	4	2	5	
3	7	1	964	9	12	4	8	10	16	1
4			13	960	1	11	2	7	10	6
5	1	2	4	1	941		9	2	4	18
6	8	1	1	23	2	820	14	1	16	6
7	9	3	2	3	6	10	922		3	
8	2	8	21	4	8	1		966	2	16
9	5	2	3	11	8	7	9	9	919	1
10	9	7	1	12	29	7	1	8	7	928

(a) Test set's confusion Chart for the supervised memristor network with one layer. 95 % accuracy on the test set



(b) Moving Average of the Prediction Accuracy. The accuracy of every example was calculated as a binary value (0 for a correct prediction, 1 for a wrong one).

Figure 6-10: Success of the experiment

Memristor-based

Again, with this configuration a lower accuracy on the test set has been achieved: 90%. According to LeCun et Al [71], the maximum accuracy ever achieved for this particular architecture (classic ANN with 1 hidden layer, 300 hidden units, no memristors) is 95.3 %. 90%, taking into consideration the limitations of memristors in terms of resolution, can be considered a success. However, a few things must be indicated about how this result was achieved:

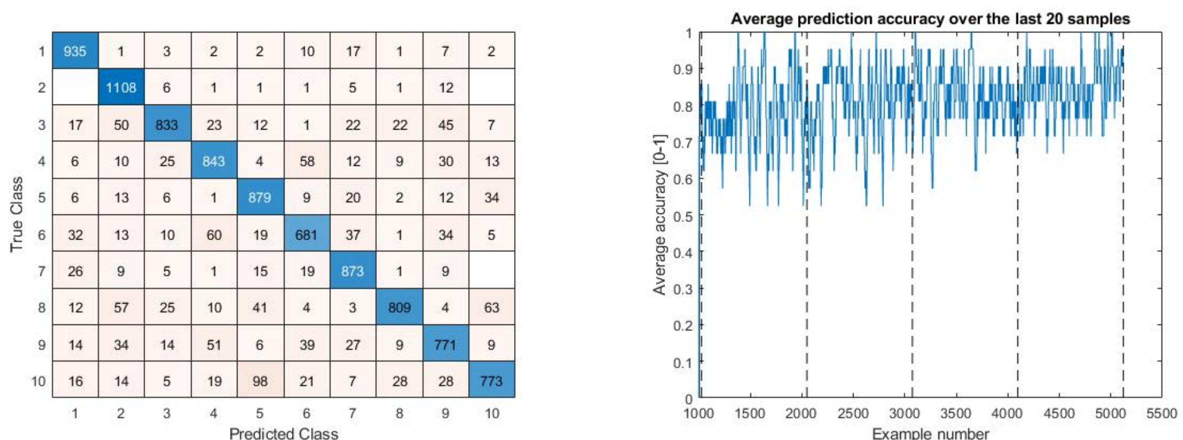
- For every epoch, only 1024 samples out of the entire data-set were taken. The images were shuffled at the beginning of every epoch so that these 1024 samples would never be

the same ones. This was done mostly to speed up the process and due to the fact that the weight update is quite coarse on memristors. Presenting the entire data-set to such a network only lead to having oscillations around an insufficient accuracy measurement (60 % for example).

- Not all layers were updated when going through every sample of the batch, but one at a time. This was done to compensate for the coarse step sizes that had to be used, so that the effect of one sample on the entire network would be reduced. It was also beneficial in terms of computational cost. This is similar to the principle of using dropouts.

- Early stopping was used to stop learning when a sufficiently good result was achieved, just like in the previous case.

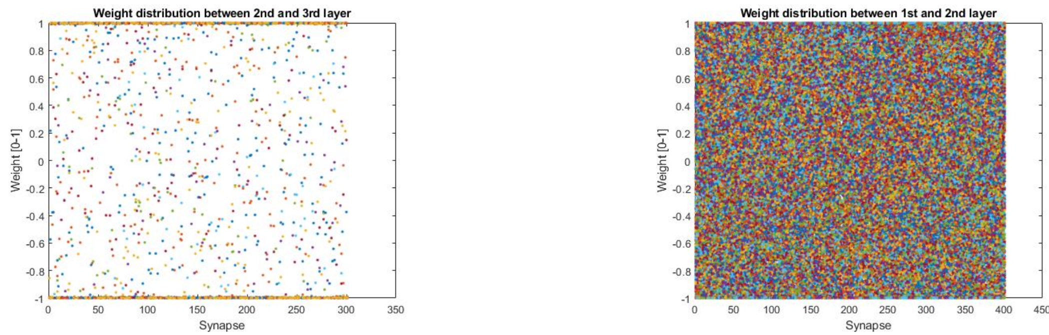
As a side note: One may point out that these same methods could have been then reapplied to the baseline. However, the goal of this work is not to discern which architectures or methods work best on a neural network, but first to figure out whether a certain configuration *works* in principle and then to make it work with the added difficulty of memristors. This is a proof of concept, and not a discussion about building the best possible neural network for handwritten character recognition.



(a) Test set's confusion Chart for the supervised memristor network with one layer. 90 % accuracy on the test set

(b) Moving Average of the Prediction Accuracy. The accuracy of every example was calculated as a binary value (0 for a correct prediction, 1 for a wrong one).

Figure 6-11: Success of the experiment



(a) Distribution of the first group of weights

(b) Distribution of the second group of weights

Figure 6-12: Weight Distribution. It seems that the first group of weights (on the right) is more equally spread out, whereas the second group of weights (on the left) has many memristors in extreme states. It seems that the weight update is stronger between the second and the third layer, which makes sense but also may prevent the network from achieving better results.

These two topologies of ANN show that memristors are a suitable element for storing the weights, and that the proposed circuit is a valid way of putting them together in a differential manner.

6-3 Conclusions

Several Neural Networks have been implemented: an Unsupervised STDP based one, a Supervised STDP based on and two Artificial Neural Networks with backpropagation. In the case of STDP, no weight update is calculated: the weights are updated according to a computational model acting on the relations between input and output signals. In the Artificial Neural Networks, even though a specific change in resistance is calculated, no feedback was used to make that change accurate. That is, if a pulse did not exactly cause the expected resistance change, no correcting pulses were applied. Instead, subsequent steps of the learning process were expected to adjust those errors.

Despite of STDP being an interesting approach that seems to take advantage of the properties of memristors, the best performing approaches were the two Artificial Neural Networks, from which the multi-layer one performed best. The non-idealities of memristors seem to be compensated by using a large amount of them.

Chapter 7

Conclusions

7-1 Summary

Neuromorphic Computing was originally intended to create devices that could mimic biological brains. However, with the appearance of Machine Learning and more specifically Neural Networks, it expanded its field of application in the search of hardware that could be more suited for these algorithms. Biologically inspired concepts such as Spiking Neural Networks or Spiking Timing-Dependant Plasticity (STDP) have been revisited ever since, with the idea of not only mimicking the processing happening in a human brain, but also to solve problems with high levels of parallelism better and faster.

Within neuromorphic devices, memristors are an increasingly popular one. They were predicted in the 70s, said to be manufactured for the first time in 2008 and made commercially available to the public a few years ago. Their properties, such as non-volatility, are appealing to the scientific community. However, their behavior is not ideal nor straightforward.

The memristors of choice, Self-Directed Channel (SDC) memristors manufactured by Knowm, are highly stochastic and do not possess a too high endurance. If the conditions under which they need to be worked with are further studied, and if their behavior is improved, they have an enormous potential for neuromorphic applications.

7-2 Main Contributions

First of all, the research question will be answered.

- Memristors are a competitive component for Neural Networks.
  - They should be in theory also competitive for Spiking Neural Networks, or at least the Literature suggests so. However, no time-encoding approaches were in the end tested with this thesis. The approaches tested without time encoding are in a way spiking, since the resistance of memristors is changed by short pulses and not continuous DC signals.

  - When it comes to Artificial Neural Networks, the topologies that work best with them seem to be those in which there is a high number of memristors, which compensates for their non-idealities, such as stochasticity and coarse resistance change (weight update).
  - Another promising Neural Network approach for memristors is STDP, although no success was achieved with it in this thesis. Better results have been achieved in Literature and the ones in this thesis should be possible to overcome and improve by adding mechanisms such as lateral inhibition, decaying weights and more output neurons. STDP reduces the complexity of the network's learning rule by removing the need to calculate backpropagation updates.

- The main challenges of circuit design with memristors are their stochasticity and their sensitivity to low voltages. Some circuits proposed in Literature assume high threshold voltages, which allows them to use simpler electronics with less switches and other protection elements. However, the memristors used in this thesis respond to voltages as low as 0.26 V or -0.11 V. Therefore, when writing a memristor in a matrix configuration it is important to ensure that no current accidentally reaches the other memristors and reprograms them.

- The circuit designed in Chapter 5 seems to be, at least in simulation, a valid and realistically implementable proposal for working with memristors. It would need other processing units, at least for backpropagation (or a look-up table), but it can be used for magnitude encoding and timing encoding, and if scaled adequately, the outputs of the circuit can be used as the inputs of another one with the same design. If timing encoding was implemented, it would also need suitable Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC) converters.

Then, the main contributions in this Master Thesis will be outlined:

- An existing software framework for the Knowm memristor set-up (Knowm Memristor Discovery) was extended to include a set of experiments that aimed to characterize the behavior of the memristors.

- The results from the experiments were used to find a suitable model for the behavior of the memristors: The Metastable Switch Memristor Model was the one that gave the best results. Optimization was attempted on the parameters of the Metastable Switch Memristor Model as well, but no improvements were achieved. The optimization could have been more thorough, but it was decided to move on: the achieved results provided useful predictions with an acceptable error, and the research of a much more better model went too much into the direction of device physics.

- The Metastable Switch Memristor Model was used to build a simulation on Simulink that could help develop a memristor controller. Functions were written to calculate the necessary voltage to apply given a wanted change in resistance, and to estimate the actual change in resistance with normal behavior.

- A circuit was designed and tested on software with the help of LTSpice, with the purpose of creating a Neural Network in hardware that could take different approaches (Unsu-

pervised and Supervised STDP; a classic supervised Artificial Neural Network (ANN) with backropagation).

- The functions built and tested on Simulink to model memristor's behavior were put together with the principles of operation of the circuit designed on LTSpice, and a simulation was designed on Matlab of different Neural Networks: Two classic supervised ANN topologies with backpropagation and Supervised and Unsupervised STDP-based Networks. The best results were achieved on the ANN runs.

7-3 Future Work

This work can be further improved in different directions:

- Manufacturing the designed circuit with several layers and a large amount of neurons per layer, so that different topologies can be tested on it by having a selectable amount of layers and neurons per layer.
- Exploring how to improve the simulated Neural Network with STDP so that it yields better results. STDP is present in the human brain and it succeeds to help us classify elements of reality in a way which is not yet fully understood. Hence, it should be possible to make a simpler version of it work with memristors.
- Exploring how to add convolutional layers to this design, to be able to do more intricate image processing.
- Exploring in depth the manufacturing process of memristors in order to see how to protect them from undesired effects, such as glass melting.

Appendix

7-3-1 The dR/dE ratio

Experiment 1

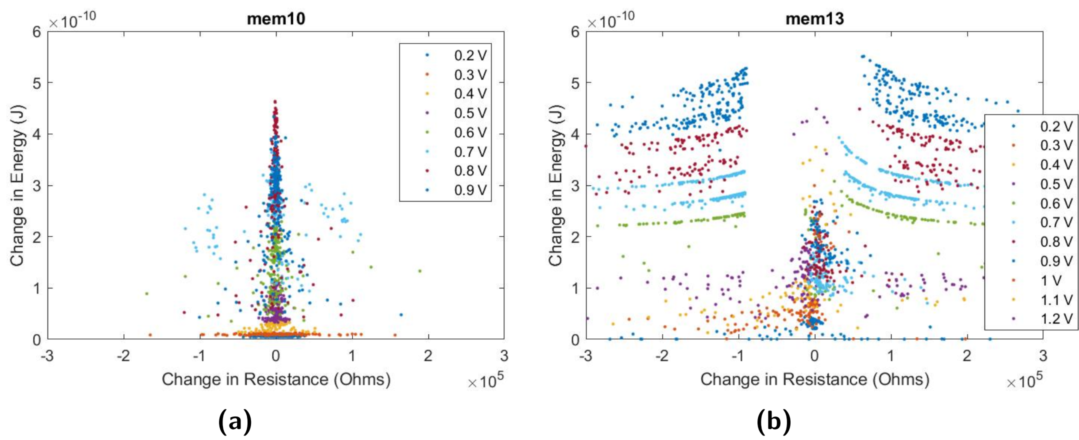


Figure 7-1

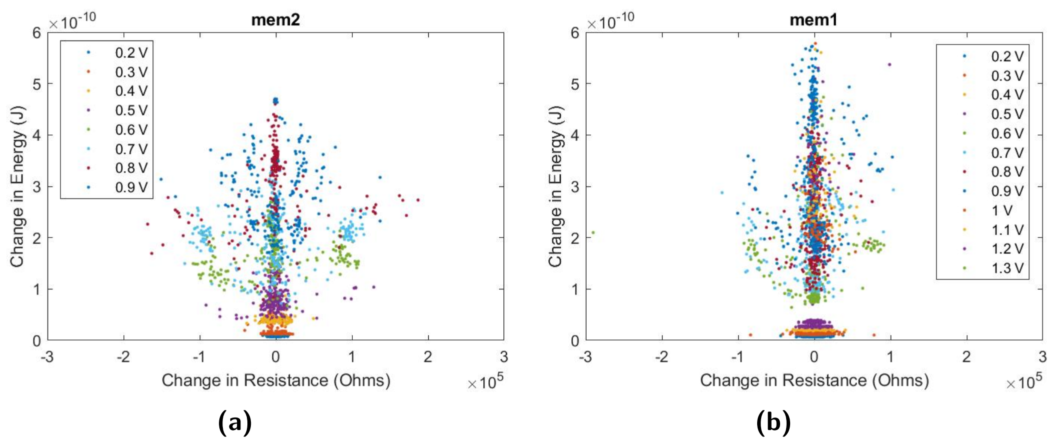
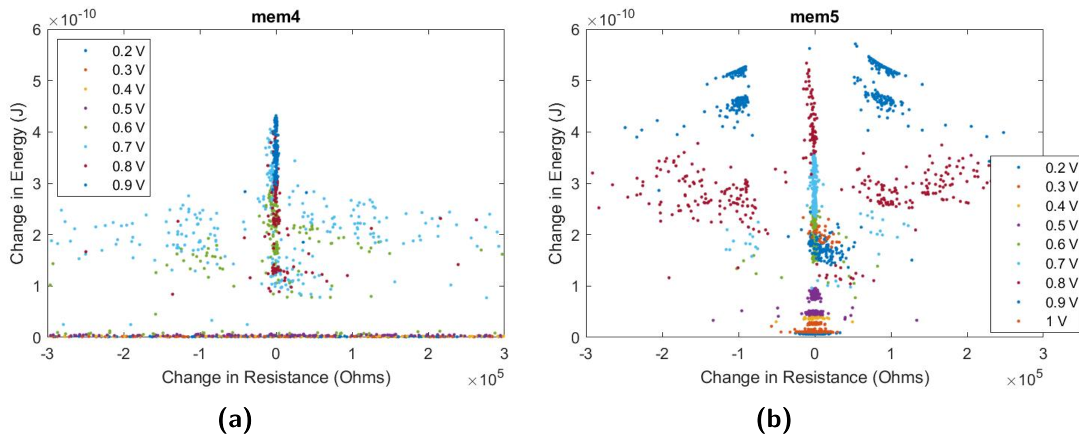


Figure 7-2



(a) **(b)**

Figure 7-3

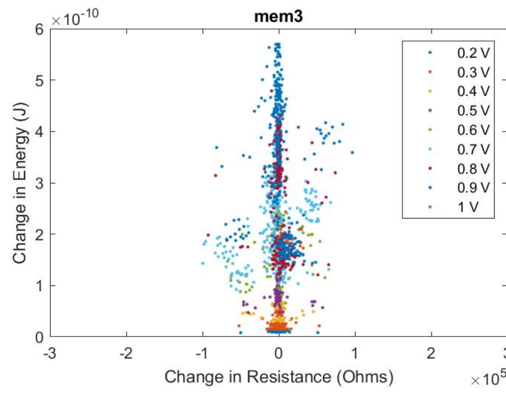
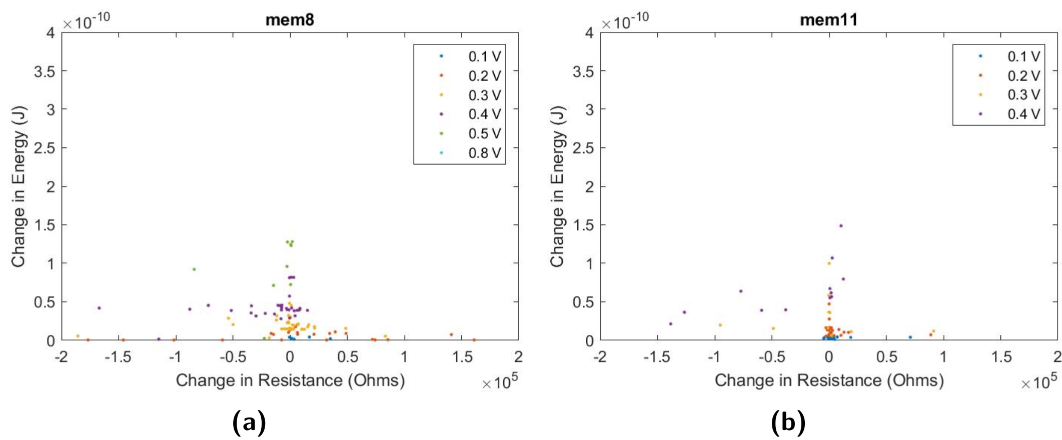


Figure 7-4

Experiment 2



(a) **(b)**

Figure 7-5

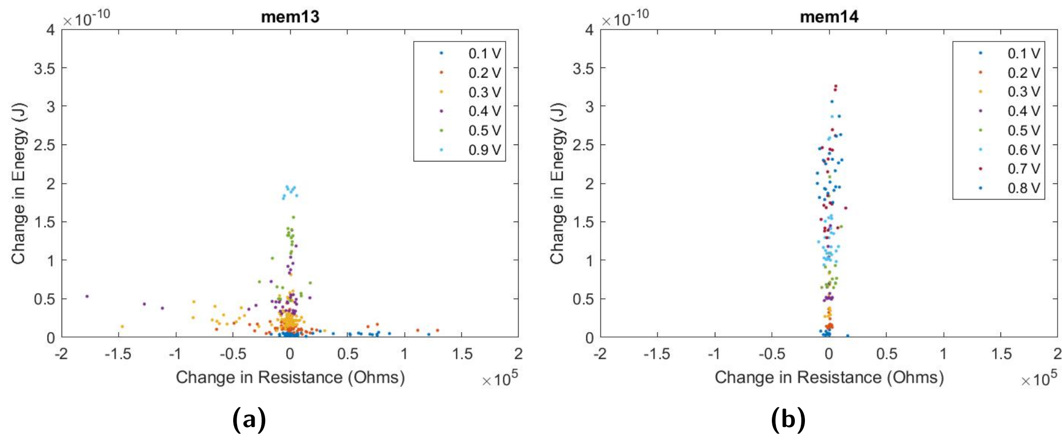


Figure 7-6

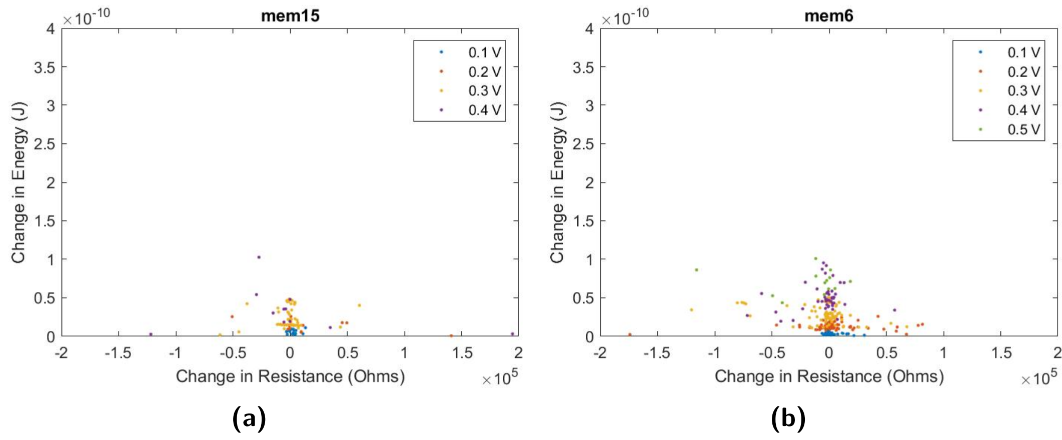


Figure 7-7

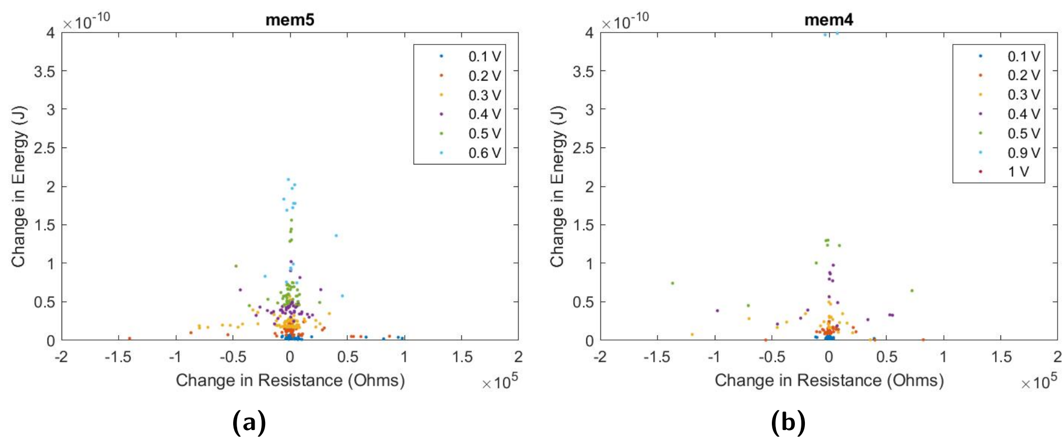
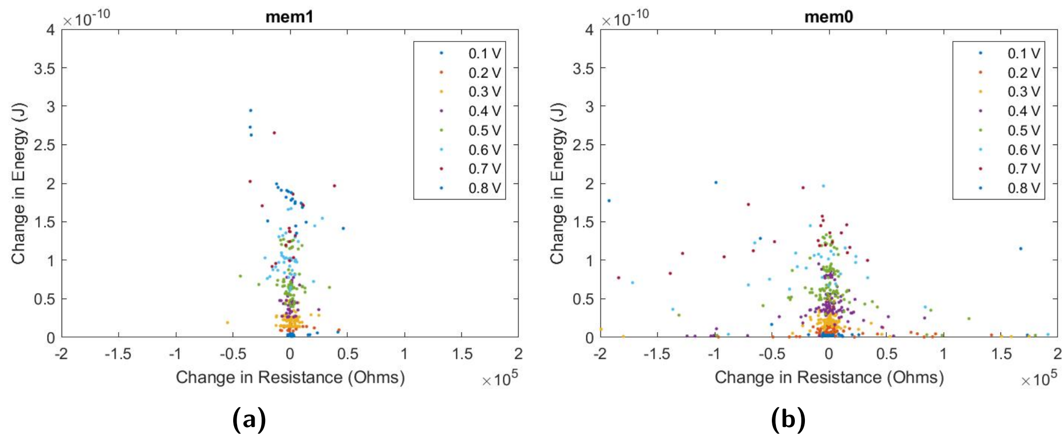


Figure 7-8

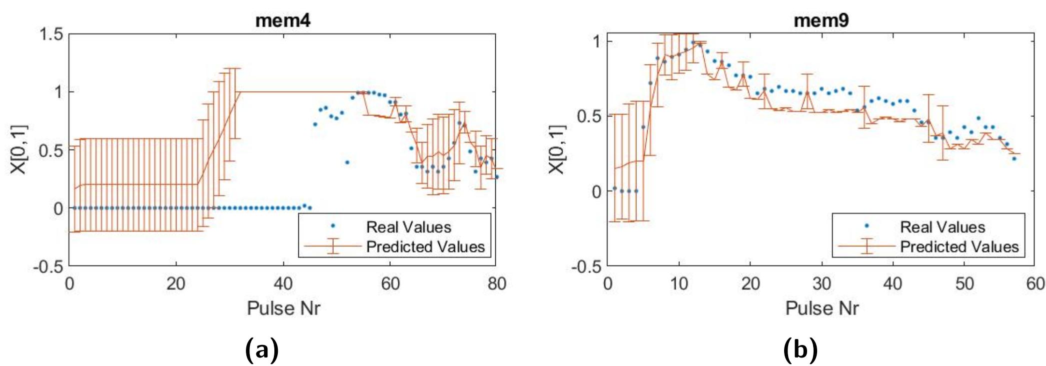


(a) **(b)**

Figure 7-9

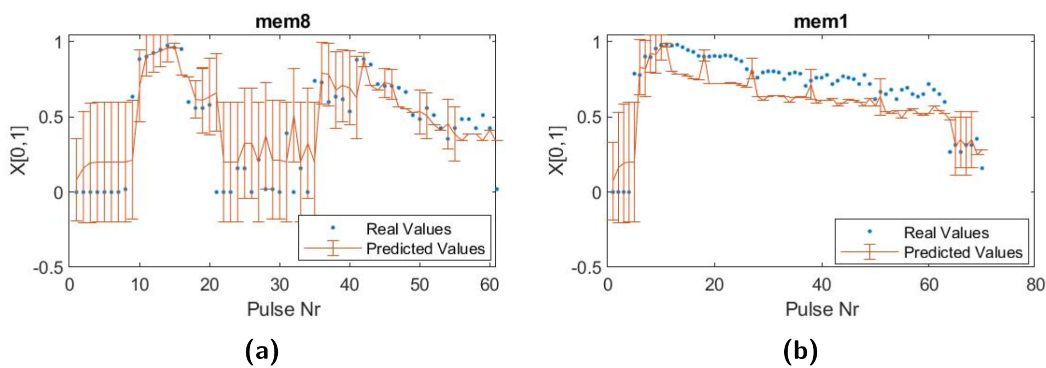
7-3-2 Time series of the experiments

Experiment 1: Chip 1 (Carbon) - Before Fine Tuning



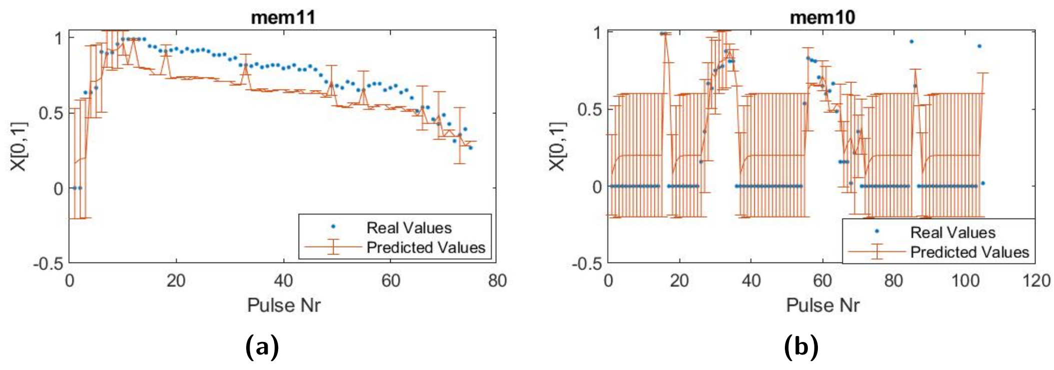
(a) **(b)**

Figure 7-10



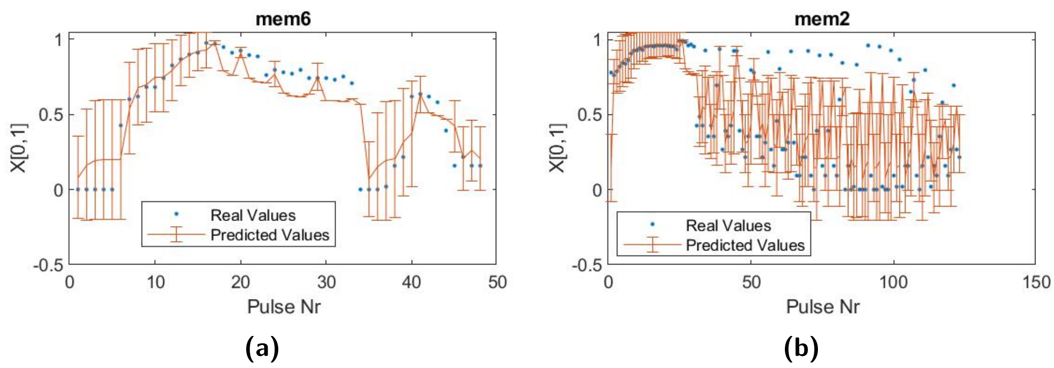
(a) **(b)**

Figure 7-11



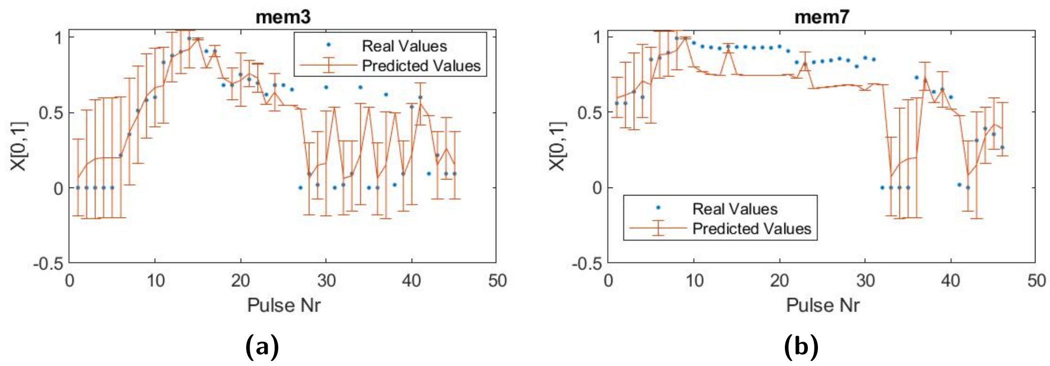
(a) (b)

Figure 7-12



(a) (b)

Figure 7-13



(a) (b)

Figure 7-14

Experiment 1: Chip 2 (Tungsten) - Before Fine Tuning

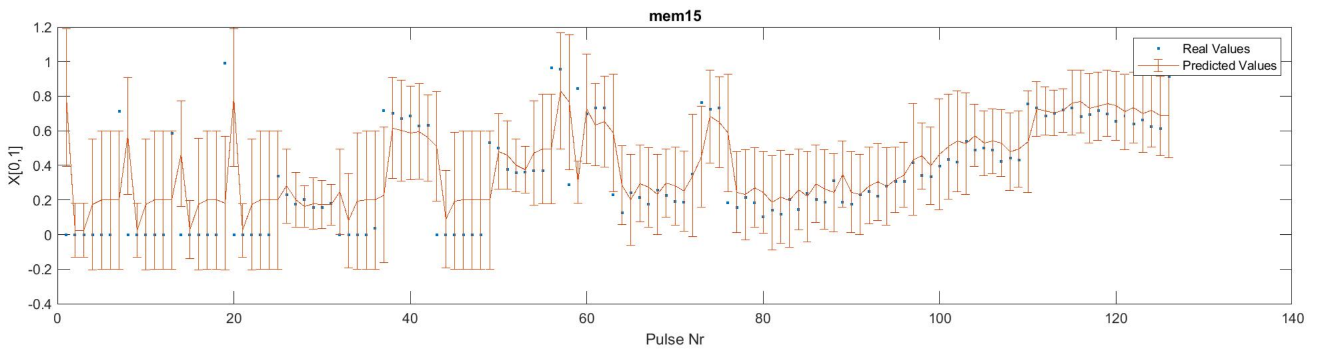


Figure 7-15

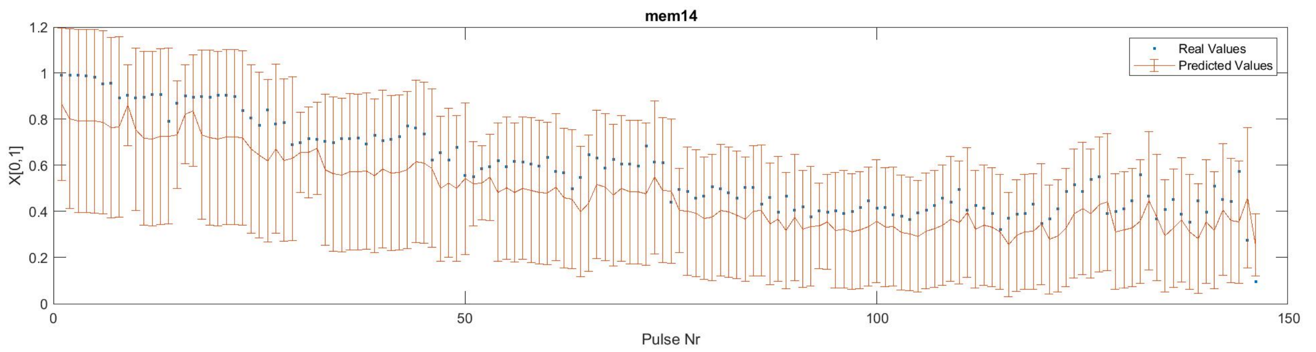


Figure 7-16

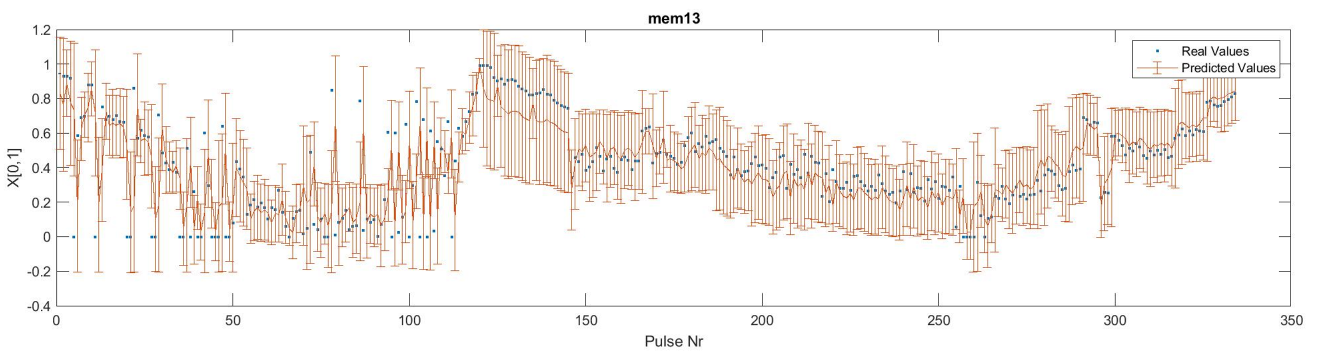


Figure 7-17

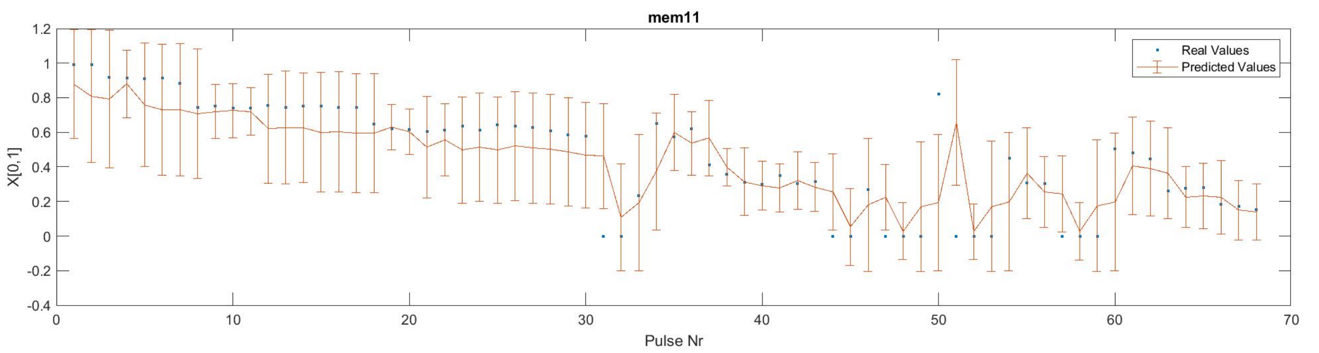


Figure 7-18

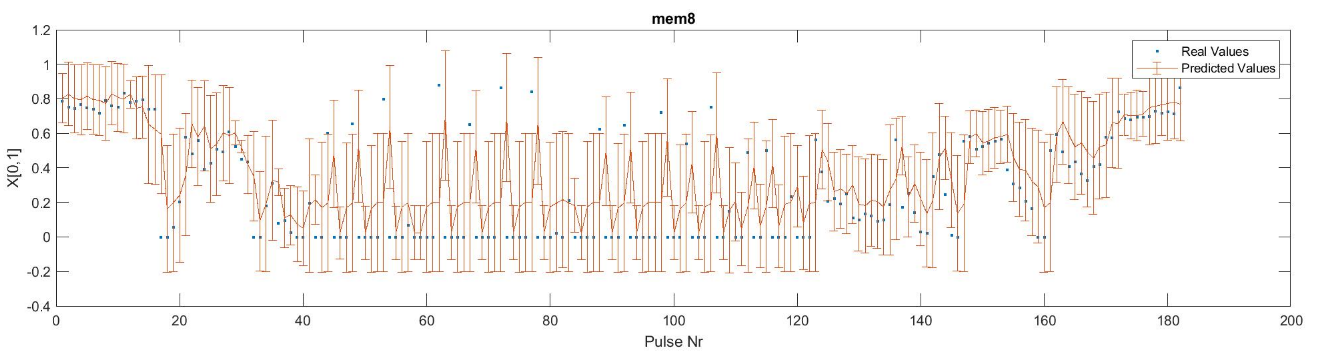


Figure 7-19

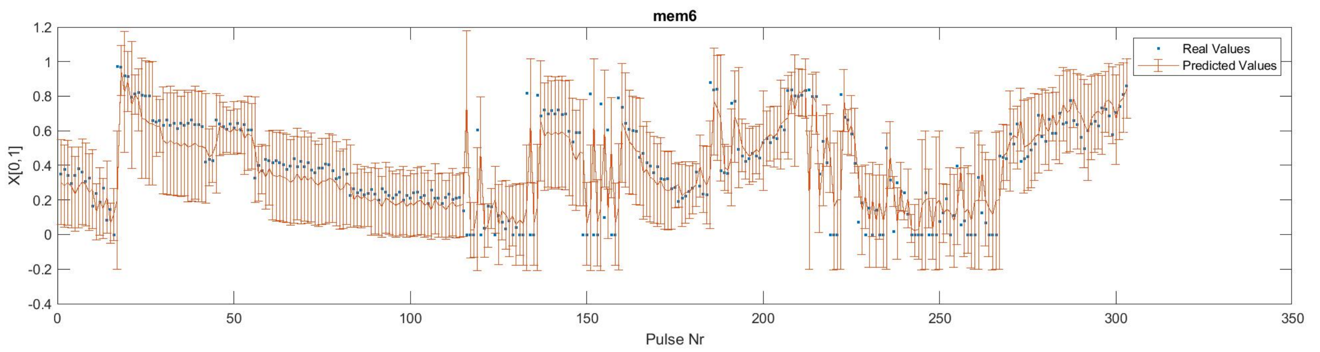


Figure 7-20

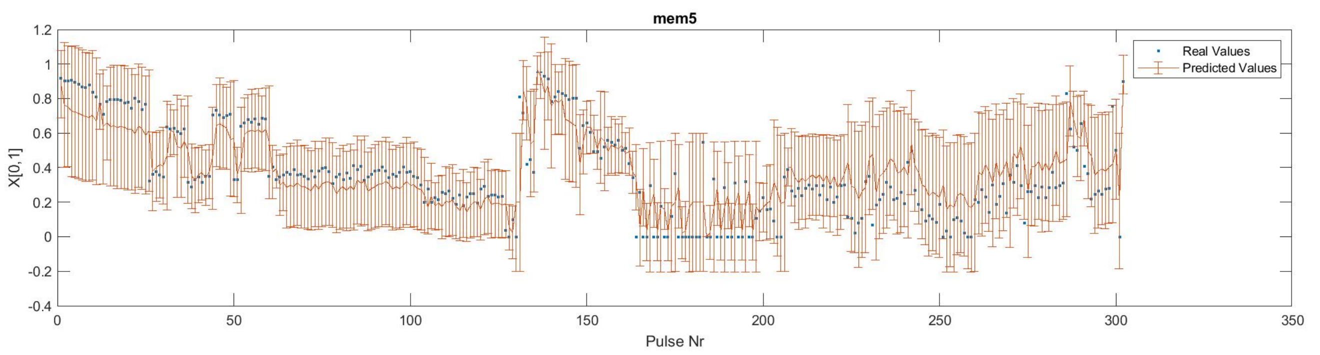


Figure 7-21

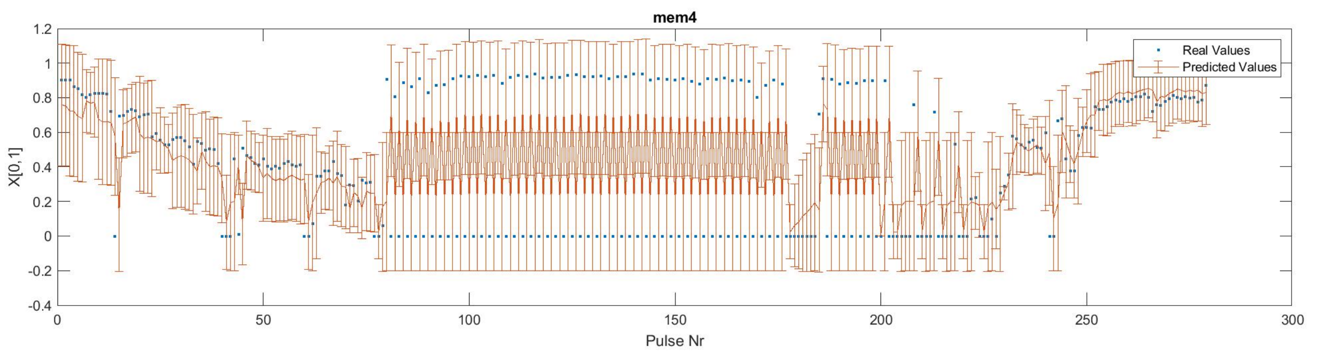


Figure 7-22

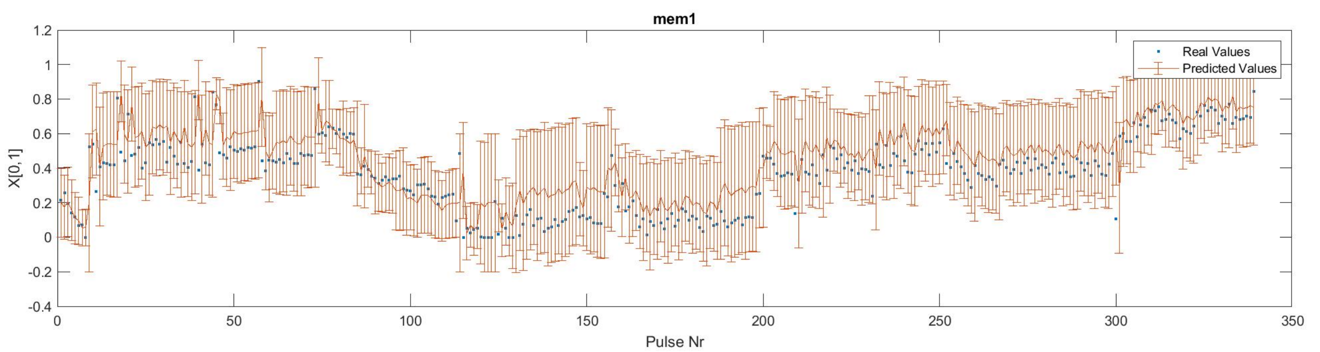


Figure 7-23

7-3-3 Parameter Tuning - Matlab Code

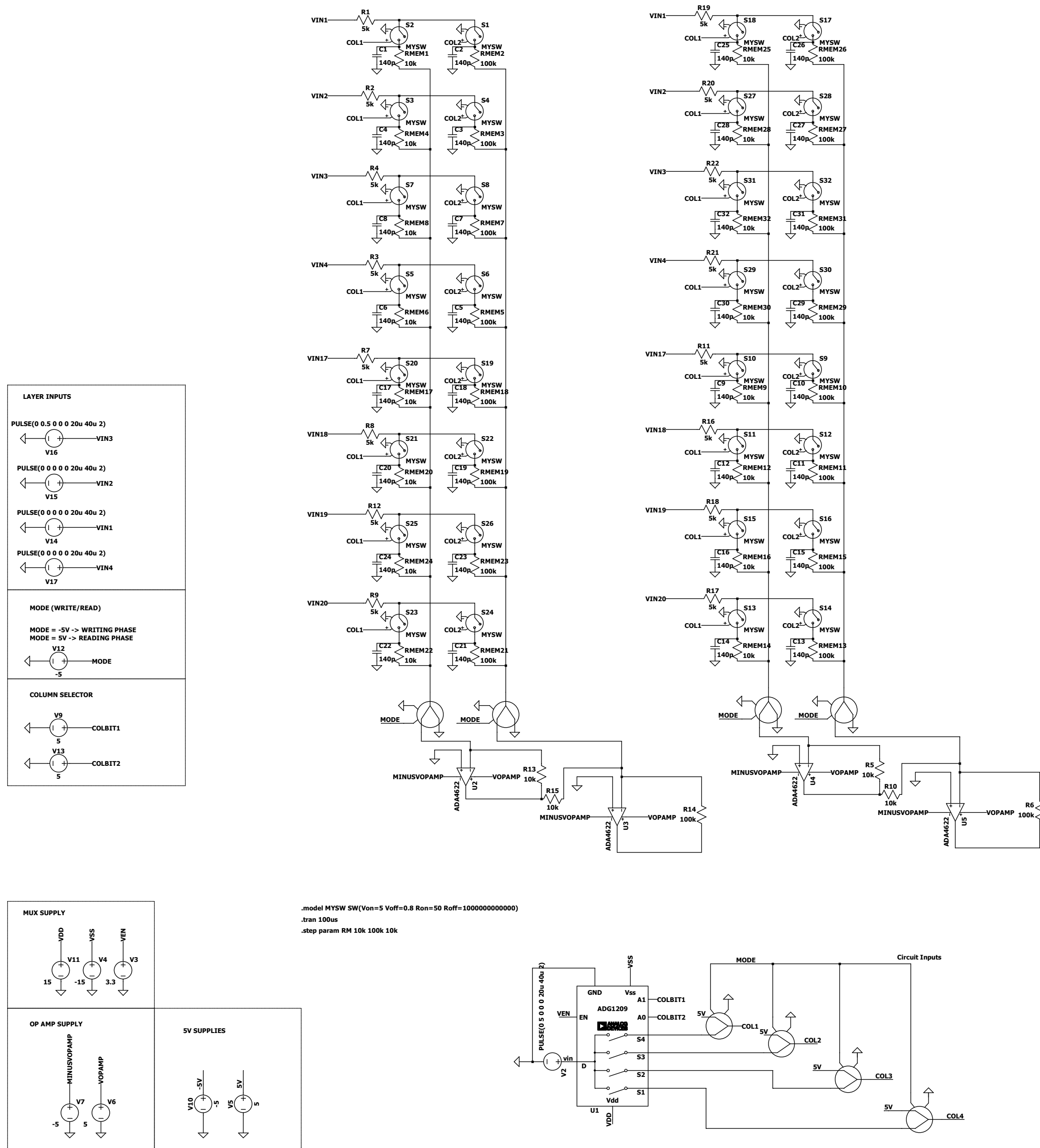
```
1  %% Optimizing 4 vars
2
3  rng default
4  options=optimoptions(@fmincon,'MaxIterations', 10000,'Algorithm','sqp',
       ...
5  'UseParallel',true,'FiniteDifferenceStepSize',1e-8,'StepTolerance',1e-8,'
       FunctionTolerance',1e-4);
6  options.MaxFunctionEvaluations=100000;
7
8  clear lb ub x0
9  lb=[0.15 0.05 0.01 0.01];
10 ub=[0.8 0.8 5 5];
11 % lb=[0.15 0.05];
12 % ub=[0.8 0.8];
13 x0(1)=0.26;
14 x0(2)=0.11;
15 x0(3)=1;
16 x0(4)=1;
17
18 problem = createOptimProblem('fmincon','objective',@(x) errCalc(x,
       memDataReduced),...
19     'x0',x0,'lb',lb,'ub',ub,'options',options);
20 % xcalc = fmincon(problem);
21
22 ms = MultiStart;
23 ms.UseParallel = true;
24 ms = MultiStart(ms,'StartPointsToRun','bounds');
25 [xsqp,Ysqp] = run(ms,problem,10000);
26
27 function Y = errCalc(x,memDataReduced)
28
29     VmemAccumulated = [];
30     VappliedAccumulated = [];
31     XoldAccumulated = [];
32     Xaccumulated = [];
33     dXaccumulated = [];
34     ResAccumulated = [];
35     ResOldAccumulated = [];
36     pulseWidthAccumulated = [];
37
38     for i = 0:15
39         str = strcat('mem', num2str(i));
40         if isfield(memDataReduced,str)
41             if (memDataReduced.(str).state==1) %Don't waste time on
                   broken ones
42                 VmemAccumulated = [VmemAccumulated; memDataReduced.(str).
                       Vmem];
43                 VappliedAccumulated = [VappliedAccumulated;
                       memDataReduced.(str).Vapplied];
44                 XoldAccumulated = [XoldAccumulated; -1e-6*max(min(
                       memDataReduced.(str).oldRes,1000000),0)+1];
```

```
45                  dXaccumulated = [dXaccumulated; memDataReduced.(str).
                        realdX];
46                  ResAccumulated = [ResAccumulated; memDataReduced.(str).
                        Resistance];
47                  ResOldAccumulated = [ResOldAccumulated; memDataReduced.(
                        str).oldRes];
48                  pulseWidthAccumulated = [pulseWidthAccumulated;
                        memDataReduced.(str).PulseWidth];
49                  Xaccumulated = [Xaccumulated; memDataReduced.(str).X];
50              end
51          end
52      end
53
54      Y = [];
55
56      BETA = 1/0.026;
57      P_OFF_ON = zeros(size(VmemAccumulated));
58      P_ON_OFF = zeros(size(VmemAccumulated));
59      N_OFF_ON = zeros(size(VmemAccumulated));
60      N_ON_OFF = zeros(size(VmemAccumulated));
61      preddX = zeros(size(VmemAccumulated));
62      realdX = zeros(size(VmemAccumulated));
63
64      for j = 1:length(VmemAccumulated)
65          P_OFF_ON(j,1) = min(((pulseWidthAccumulated(j)*x(3))/100000),1)
                *(1./(1+exp(-BETA*(VmemAccumulated(j) - x(1)))));
66          P_ON_OFF(j,1) = min(((pulseWidthAccumulated(j)*x(4))/100000),1)
                *(1-1./(1+exp(-BETA*(VmemAccumulated(j) + x(2)))));
67          N_OFF_ON(j,1) = P_OFF_ON(j)*(1-XoldAccumulated(j)); %take
                previous X state
68          N_ON_OFF(j,1) = P_ON_OFF(j)*XoldAccumulated(j);
69          preddX(j,1) = N_OFF_ON(j,1)-N_ON_OFF(j,1);
70          realdX(j,1) = Xaccumulated(j)-XoldAccumulated(j);
71      end
72
73      Y = [Y (preddX - realdX)];
74
75  Y = sum(Y.^2);
76  end
77
78  function [c,ceq] = ourContr(x)
79  ceq = [];
80  c = [];
81  end
```

7-3-4 Memristor's stochasticity simulation

```
1  % X - Current State [0,1]
2  % relError - Relative error as calculated during backpropagation
3  % VON - Positive threshold voltage, parameter from the Metastable Switch
       Memristor Model
4  % VOFF - Negative threshold voltage, parameter from the Metastable Switch
       Memristor Model
5
6  function [Vmem] = vCalc(X, relError ,BETA,VON,VOFF)
7      relErrorPos = gpuArray(zeros(size(relError)));
8      indices = (relError<0.05 & X>0.95) | (relError>-0.05 & X<0.05);
9      relErrorPos(~indices) = relError(~indices);
10
11     VmemCalc = -1.5:0.05:1.5; %X has to be a column vector aight
12     P_OFF_ON = (1./(1+exp(-BETA*(VmemCalc - VON))));
13     P_ON_OFF = (1-1./(1+exp(-BETA*(VmemCalc + VOFF))));
14     N_OFF_ON = P_OFF_ON.*(1-X);
15     N_ON_OFF = P_ON_OFF.*X;
16     dX = gpuArray(N_OFF_ON - N_ON_OFF);
17
18     [~,closestIndex] = min(round(abs(dX - relErrorPos)'.*10000000)
           ./10000000);
19     closestValue = VmemCalc(closestIndex);
20     Vmem = closestValue;
21     Vmem(relErrorPos==0) = 0;
22
23 end
24
25 % X - Current State [0,1]
26 % Vmem - Voltage applied on the memristor directly
27 % BETA - Parameter from the Metastable Switch Memristor Model
28 % VON - Positive threshold voltage, parameter from the Metastable Switch
       Memristor Model
29 % VOFF - Negative threshold voltage, parameter from the Metastable Switch
       Memristor Model
30 function [Res,dX,X]= resCalc(X,Vmem,BETA,VON,VOFF)
31     P_OFF_ON = 1.*(1./(1+exp(-BETA.*(Vmem - VON))));
32     P_ON_OFF = 1.*(1-1./(1+exp(-BETA.*(Vmem + VOFF))));
33     N_OFF_ON = P_OFF_ON.*(1-X);
34     N_ON_OFF = P_ON_OFF.*X;
35     NA = normrnd(N_OFF_ON,N_OFF_ON.*(1-P_OFF_ON));
36     NB = normrnd(N_ON_OFF,N_OFF_ON.*(1-P_ON_OFF));
37     dX = NA - NB;
38
39     dX(abs(Vmem) < 0.01) = 0;
40
41     X = X + dX;
42     X(X < 0) = 0;
43     X(X > 1) = 1;
44     Res = (X -1.1)./(-1.1e-5);
45 end
```

7-3-5 Matrix Circuit Design



--- C:\Users\Steve\Documents\LTspiceXVII\NN1layer20in1out.asc ---

Bibliography

[1] G. Stovall, "Neurons synapse pictures — biological science picture directory." https://pulpbits.net/6-images-of-brains-synapse-neurons-structures/neurons-synapse-pictures/. Accessed: 20-09-2019.

[2] P. Jain, "Complete guide of activation functions." https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044, Jun 2019.

[3] C. M. Bishop, *Pattern recognition and machine learning.* springer, 2006.

[4] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *Trans. Neur. Netw.*, vol. 15, pp. 1063–1070, Sept. 2004.

[5] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, 2013.

[6] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

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

[8] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, p. 075201, jan 2012.

[9] Knowm.org, "The mean metastable switch memristor model in xyce." https://knowm.org/the-mean-metastable-switch-memristor-model-in-xyce/.

[10] T. W. Molter and M. A. Nugent, "The generalized metastable switch memristor model," 2016.

[11] G. Thomson and F. Macpherson, "Scintillating grid." https://www.illusionsindex.org/i/scintillating-grid, 2018. Accessed: 02-12-2019.

[12] R. Hasan, T. M. Taha, and C. Yakopcic, “On-chip training of memristor crossbar based multi-layer neural networks,” *Microelectron. J.*, vol. 66, pp. 31–40, Aug. 2017.

[13] J. Brownlee, “How to develop a cnn for mnist handwritten digit classification — machine learning mastery.” https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/, Jan 2020. Accessed: 17-06-2020.

[14] C. Mead, “Neuromorphic electronic systems,” in *Proc. IEEE, 78:16291636*, 1990.

[15] C. A. Mead and M. Mahowald in *Computational Neuroscience* (E. L. Schwartz, ed.), ch. A Silicon Model of Early Visual Processing, pp. 331–339, Cambridge, MA, USA: MIT Press, 1993.

[16] A. Pantazi, A. Sebastian, E. S. Eleftheriou, and T. Tuma, “Neuromorphic synapses,” May 7 2019. US Patent App. 10/282,657.

[17] B. Yu, “Neuroscience: Fault tolerance in the brain,” *Nature*, vol. 532, 04 2016.

[18] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016.

[19] C. Tan and G. Ji, “Semi-supervised incremental feature extraction algorithm for large-scale data stream,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 6, p. e3914, 2017.

[20] W. S. McCulloch and W. Pitts in *Neurocomputing: Foundations of Research* (J. A. Anderson and E. Rosenfeld, eds.), ch. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27, Cambridge, MA, USA: MIT Press, 1988.

[21] H. Jeong and L. Shi, “Memristor devices for neural networks,” *Journal of Physics D: Applied Physics*, vol. 52, p. 023003, oct 2018.

[22] B. Linares-Barranco and T. Serrano-Gotarredona, “Memristance can explain spike-time-dependent-plasticity in neural synapses,” *Nature precedings*, pp. 1–1, 2009.

[23] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory.* New York: Wiley, 1949.

[24] R. Kempter, W. Gerstner, and L. van Hemmen, “Hebbian learning and spiking neurons,” *Phys. Rev. E*, vol. 59, 04 1999.

[25] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, “Stdp and stdp variations with memristors for spiking neuromorphic learning systems,” *Frontiers in Neuroscience*, vol. 7, p. 2, 2013.

[26] M. Madadi Asl, A. Valizadeh, and P. Tass, *Propagation Delays Determine the Effects of Synaptic Plasticity on the Structure and Dynamics of Neuronal Networks.* PhD thesis, 03 2018.

[27] A. M. Turing, “On computable numbers, with an application to the Entscheidungsproblem,” *Proceedings of the London Mathematical Society*, vol. 2, no. 42, pp. 230–265, 1936.

[28] A. M. Turing, "Intelligent machinery," report, inst-NPL, inst-NPL:adr, 1948.

[29] C. Van Der Malsburg, "Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," in *Brain Theory* (G. Palm and A. Aertsen, eds.), (Berlin, Heidelberg), pp. 245–248, Springer Berlin Heidelberg, 1986.

[30] M. D. Godfrey and D. F. Hendry, "The computer as von neumann planned it," *IEEE Ann. Hist. Comput.*, vol. 15, pp. 11–21, Jan. 1993.

[31] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 1117–1125, Curran Associates, Inc., 2015.

[32] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017.

[33] D. Mahalanabis, M. Sivaraj, W. Chen, S. Shah, H. J. Barnaby, M. N. Kozicki, J. B. Christen, and S. Vrudhula, "Demonstration of spike timing dependent plasticity in cbram devices with silicon neurons," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2314–2317, IEEE, 2016.

[34] J. W. Jang, B. Attarimashalkoubeh, A. Prakash, H. Hwang, , and Y. H. Jeong, "Scalable neuron circuit using conductive-bridge ram for pattern reconstructions," *Transactions on Electron Devices, vol. PP, no. 99*, pp. 1—-4, 2016.

[35] H. P. W. et al, "Phase change memory," *Proc. IEEE, vol. 98, no. 12*, pp. 2201—-2227, 2010.

[36] B. L. Jackson, B. Rajendran, G. S. Corrado, M. Breitwisch, G. W. Burr, R. Cheek, K. Gopalakrishnan, S. Raoux, C. T. Rettner, and A. P. et al., "Nanoscale electronic synapses using phase change devices," *ACM J. Emerg. Technol. Comput. Syst. 9, 2, Article 12 (May 2013), 20 pages. DOI: http://dx.doi.org/10.1145/2463585.2463588*, 2013.

[37] D.-H. Kang, H.-G. Jun, K.-C. Ryoo, H. Jeong, and H. Sohn, "Emulation of spike-timing dependent plasticity in nano-scale phase change memory," *Neurocomputing, vol. 155 DOI: http://dx.doi.org/10.1145/2463585.2463588*, pp. 153—-158, 2015.

[38] T. Tuma, M. L. Gallo, A. Sebastian, , and E. Eleftheriou, "Detecting correlations using phase-change neurons and synapses," *IEEE Electron Device Letters, vol. 37, no. 9*, pp. 1238—-1241, 2016.

[39] T. Tuma, A. Pantazi, M. L. Gallo, A. Sebastian, , and E. Eleftheriou, "Stochastic phase-change neurons," *Nature nanotechnology, vol. 11, no. 8*, pp. 693—-699, 2016.

[40] A. Sengupta, Z. A. Azim, X. Fong, and K. Roy, "pin-orbit torque induced spike-timing dependent plasticity," *Applied Physics Letters, vol. 106, no. 9, p. 093704*, 2015.

[41] D. Zhang, L. Zeng, Y. Qu, Z. M. Wang, W. Zhao, T. Tang, and Y. W. et al., "Energy-efficient neuromorphic computation based on compound spin synapse with stochastic

learning," *IEEE International Symposium on Circuits and Systems*, pp. 1538—-1541, 2015.

[42] A. Sengupta, S. H. Choday, Y. Kim, and K. Roy, "Spin orbit torque based electronic neuron," *Applied Physics Letters, vol. 106, no. 14, p. 143701*, 2015.

[43] M. Sharad, C. Augustine, G. Panagopoulos, and K. Roy, "Spin-based neuron model with domain-wall magnets as synapse," *IEEE Transactions on, vol. 11, no. 4*, pp. 843—-853, 2012.

[44] R. R. Harrison, J. A. Bragg, P. Hasler, B. A. Minch, and S. P. Deweerth, "A cmos programmable analog memory-cell array using floating-gate circuits," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, vol. 48, no. 1*, pp. 4—-11, 2001.

[45] T. Morie, T. Matsuura, M. Nagata, and A. Iwata, "A multinanodot floating-gate mosfet circuit for spiking neuron models," *Nanotechnol- ogy, IEEE Transactions on, vol. 2, no. 3*, pp. 158—-164, 2003.

[46] S. Brink, S. Nease, and P. Hasler, "Computing with networks of spiking neurons on a biophysically motivated floating-gate based neuromorphic integrated circuit," *Neural Networks, vol. 45*, pp. 39—-49, 2013.

[47] M. Pankaala, M. Laiho, and P. Hasler, "Compact floating-gate learning array with stdp," *Neural Networks. IJCNN 2009. International Joint Conference on. IEEE*, pp. 2409—-2415, 2009.

[48] Y. L. Wong, P. Xu, and P. Abshire, "Ultra-low spike rate silicon neuron," *Biomedical Circuits and Systems Conference, BIOCAS. IEEE*, pp. 95—-98, 2007.

[49] S. Brink, S. Koziol, S. Ramakrishnan, and P. Hasler, "A biophysically based dendrite model using programmable floating-gate devices," *IEEE International Symposium on Circuits and Systems*, 2008.

[50] M. P. Fok, Y. Tian, D. Rosenbluth, and P. R. Prucnal, "Asynchronous spiking photonic neuron for lightwave neuromorphic signal processing," *Optics letters, vol. 37, no. 16*, pp. 3309—-3311, 2012.

[51] M. Nahmias, B. J. Shastri, A. N. Tait, and P. R. P. et al., "A leaky integrate-and-fire laser neuron for ultrafast cognitive computing," *Selected Topics in Quantum Electronics, IEEE Journal of, vol. 19, no. 5*, pp. 1—-12, 2013.

[52] B. Gholipour, P. Bastock, C. Craig, K. Khan, D. Hewak, and C. Soci, "Amorphous metal-sulphide microfibers enable photonic synapses for brain-like computing," *Advanced Optical Materials, vol. 3, no. 5*, pp. 635—-641, 2015.

[53] Q. Ren, Y. Zhang, R. Wang, and J. Zhao, "Optical spike-timing- dependent plasticity with weight-dependent learning window and reward modulation," *Optics Express, vol. 23, no. 19*, pp. 25247—-25258, 2015.

[54] L. Chua, "Memristor – the missing circuit element," *IEEE TRANS. ON CIRCUIT THEORY*, vol. 18, no. 5, pp. 507–519, 1971.

[55] G. S. Snider, "Spike-timing-dependent learning in memristive nanodevices," in *2008 IEEE International Symposium on Nanoscale Architectures*, pp. 85–92, June 2008.

[56] Z. Hui, P. Haipeng, J. Kurths, J. Xiao, and Y. Yang, "Anti-synchronization for stochastic memristor-based neural networks with non-modeled dynamics via adaptive control approach," *The European Physical Journal B*, vol. 88, 05 2015.

[57] A. Nugent, "Memristor discovery manual." Accessed: 2019-12-03.

[58] "The memristor revisited." https://www.nature.com/articles/s41928-018-0083-3, May 2018. (accessed: 10.01.2020).

[59] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, pp. 1117–1125, 2015.

[60] K. Campbell, "Self-directed channel memristor for high temperature operation," *Microelectronics Journal*, vol. 59, 08 2016.

[61] F. Alibart, E. Zamanidoost, and D. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nature communications*, vol. 4, p. 2072, 06 2013.

[62] A. N. Timm Molter, "Memristor discovery." `https://github.com/knowm/memristor-discovery`, 2020.

[63] T. Molter, "Knowm/jspice github repository." `https://github.com/knowm/jspice`, Feb 2020.

[64] A. Devices, "Ltspice (design center - analog devices)." `https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html`. (Accessed: 01.04.2020).

[65] J. Gomez, I. Vourkas, A. Abusleme, R. Rodriguez, J. Martin-Martinez, M. Nafria, and A. Rubio, "Exploring the "resistance change per energy unit" as universal performance parameter for resistive switching devices," *Solid-State Electronics*, vol. 165, p. 107748, 2020.

[66] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

[67] B. Linares-Barranco, T. Serrano-Gotarredona, L. A. Camuñas-Mesa, J. A. Perez-Carrasco, C. Zamarreño-Ramos, and T. Masquelier, "On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex," *Frontiers in neuroscience*, vol. 5, p. 26, 2011.

[68] D. Querlioz, W. Zhao, P. Dollfus, J.-O. Klein, O. Bichler, and C. Gamrat, "Bioinspired networks with nanoscale memristive devices that combine the unsupervised and supervised learning approaches," in *2012 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 203–210, IEEE, 2012.

[69] E. Zamanidoost, F. M. Bayat, D. Strukov, and I. Kataeva, "Manhattan rule training for memristive crossbar circuit pattern classifiers," in *2015 IEEE 9th International Symposium on Intelligent Signal Processing (WISP) Proceedings*, pp. 1–6, IEEE, 2015.

[70] D. Chabi, Z. Wang, C. Bennett, J.-O. Klein, and W. Zhao, "Ultrahigh density memristor neural crossbar for on-chip supervised learning," *IEEE Transactions on Nanotechnology*, vol. 14, no. 6, pp. 954–962, 2015.

[71] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *International conference on artificial neural networks*, vol. 60, pp. 53–60, Perth, Australia, 1995.

Glossary

List of Acronyms

ANN	Artificial Neural Network
SNN	Spiking Neural Network
NN	Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
ALU	Arithmetic Logic Unit
MOS	Metal Oxide Semiconductor
CMOS	Complementary Metal Oxide Semiconductor
STDP	Spiking Timing-Dependant Plasticity
CBRAM	Conductive-Bridging RAM
FGT	Floating Gate Transistors
LIF	Leaky Integrate-And-Fire
LRS	Low Resistance State
HRS	High Resistance State
NVM	Non-volatile Memory
PCM	Phase Change Memory
SQP	Sequential Quadratic Programming
MSE	Mean Square Error
MRAM	Magnetic Random-Access-Memory

FGT Floating Gate Transistor

SDC Self-Directed Channel

SPDT Single Pole Double Throw

SPST Single Pole Single Throw

ADC Analog to Digital Converter

DAC Digital to Analog Converter

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor

MSMM Metastable Switch Memristor Model

PCB Printed Circuit Board