MEMRISTIVE SPIKING NEURAL NETWORKS FOR SPEECH RECOGNITION

A Thesis

Submitted by

ANMOL S ZANWAR

EE16B160

In partial fulfilment of the requirements

For the award of the degree of

BACHELOR OF TECHNOLOGY &

MASTER OF TECHNOLOGY

(DUAL DEGREE)

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

June 2021

QUOTATIONS

कर्मण्येवाधिकारस्ते मा फलेषु कदाचन।
मा कर्मफलहेतुर्भूर्मा ते सङ्गोऽस्त्वकर्मणि॥ २-४७

Translation: Your right is for action alone, never for the results. Let not the fruits of action be your motive, nor let your attachment be to inaction.

DEDICATION

to my Parents, Teachers & the Eternal Energy

THESIS CERTIFICATE

This is to certify that the thesis entitled **MEMRISTIVE SPIKING NEURAL NETWORKS FOR SPEECH RECOGNITION** submitted to the **Indian Institute of Technology Madras** by **Anmol S Zanwar** for the award of the degree of **Bachelor of Technology and Master of Technology (Dual Degree)** is a bonafide record of the project work done by him under our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.Bhaswar Chakrabarti
Research Guide
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Madras
Chennai – 600 036.

Date: 29 June, 2021

Place: Chennai

ACKNOWLEDGEMENTS

I'm deeply indebted to Prof. Bhaswar Chakrabarti, my project guide, for his continuous guidance and encouragement. I'm extremely grateful for his valuable advice and insights throughout the project. Thank you for introducing me to the exciting field of neuromorphic computing. Working with you has been a wonderful experience and I shall cherish these memories forever.

I would also like to express my deepest gratitude to Prof. Anjan Chakravorty for his insightful suggestions and unwavering support. His constructive feedback has been immensely helpful in completing my project.

I'm extremely thankful to Prof. Suresh Balanethiram for his valuable suggestions and guidance throughout the course of my project. I'd like thank all of them for devoting their valuable time and holding regular meetings to guide me and motivate me at all times.

Special thanks to my friend Shubham for his wonderful insights and constant motivation. Many interesting discussions with him have been very helpful in my project. He has been immensely helpful in numerous possible ways and I can't thank him enough.

I would also like to acknowledge the kind extension of the Samsung-IITM Pravartak fellowship for the project.

Finally, I would like to extend my deepest gratitude to my parents whose endless support, constant motivation and unconditional love has always been the cornerstone behind all my achievements. I'm forever grateful to them for everything.

List of Tables

List of Figures

ABBREVIATIONS

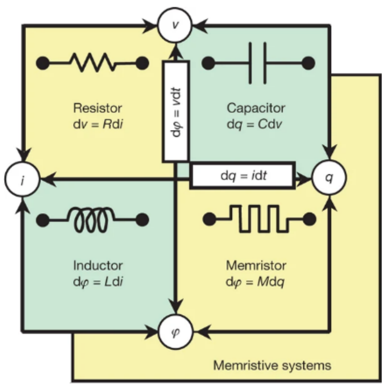
SNN	Spiking Neural Network
ANN	Artificial Neural Network
RRAM	Resistive Random Access Memory
STDP	Spike-timing Dependent Plasticity
HRS	High Resistance State
LRS	Low Resistance State
LTP	Long term Potentiation
LTD	Long term Depression
LIF	Leaky-Integrate-Fire
ASR	Automatic Speech Recognition
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network

Contents

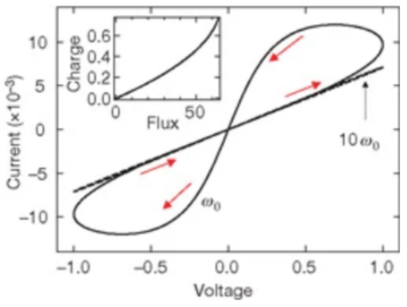
INTRODUCTION

1.1 Memristor

The memristor is a fundamental circuit element that was postulated by Chua[1] in 1971. In classical circuit theory, there are four fundamental quantities: the current (i) , the voltage(v), the charge(q) and the flux linkage(ϕ) . These quantities can be combined into six pairwise relations out of which two are the relations between a quantity and its derivative. They are : $i = \frac{dq}{dt}$, i.e. the current as the rate of change of charge and $v = \frac{d\phi}{dt}$, i.e. the voltage as the rate of change of flux. The other three relations are the defined by the fundamental circuit elements. The resistor relates the voltage(v) and the current(i) as $dv = Rdi$. The capacitor relates the charge(q) to the voltage(v) as $dq = Cdv$ and the inductor relates the flux(ϕ) to the current(i) as $d\phi = Ldi$. Out of the six possible pairwise combinations, only five had been identified. Chua realized that the relation between the flux(ϕ) and the charge(q) had been missing. Based on his famous symmetry argument, as shown in figure 1.1, he postulated



(a)



(b)

Figure 1.1: (a)The four fundamental circuit elements (b) I-V curve of the memristor displaying hysterisis behavior[2]

the existence of the fourth fundamental circuit element and called it as the memristor. The memristor provides a static relationship between the charge and the flux as $d\phi = Mdq$. M denotes the memristance of the memristor. It is a two-terminal passive element but unlike the other three elements, it is non-linear. The term memristor originates from the abbreviation of 'memory+resistor'. The memristor behaves like a non-linear resistor that exhibits memory, i.e. the resistance of the memristor depends on the amount of charge that has flown through it. Thus, its resistance can be changed by applying an appropriate input voltage/current. It also has a non-volatile nature because it can maintain its resistance indefinitely in the absence of any external inputs. All of these properties make it a very promising candidate for implementing cognitive artificial neural systems.

1.1.1 Physical realization of the memristor

Although the concept of a memristor was postulated way back in 1971, it was only in 2008 that Strukov et al[2] successfully demonstrated the correlation of memristance with the non-volatile resistance switching phenomenon observed in two-terminal metal-insulator-metal structures. Since then, memristors have been physically realized using various nanoscale two-terminal devices such as resistive-switching devices, phase-change devices, spintronic memories and ferroelectric tunnel junctions. Observation of memristance became possible with the ability to scale such devices in the nanometer regime. This is because memristance often strongly depends on the applied electric field which can be significant in devices with ultra-thin dielectric.

Our work focuses on Resistive Random-Access Memory or RRAM devices. The RRAM is a two-terminal nanoscale device with a metal-insulator-metal structure. While there are a variety of materials that exhibit resistive switching, we shall focus on bipolar metal-oxide RRAM devices. In such devices, metals such as Pt,Ti,Al etc are used as the top and bottom electrodes whereas the insulator is an oxide such as TiOx, HfOx, NiOx, AlOx etc. The resistance of the device is governed by the growth and rupture of a conductive filament in the oxide layer. The working principle of a 'bipolar' metal-oxide RRAM is explained in figure 1.2. The metal oxide has an initial concentration of oxygen vacancies, primarily at the grain-boundaries for a polycrystalline thin film. Application of a positive voltage at the top electrode (anode) results in the following phenomena: (i) migration of oxygen ions towards the anode, (ii) flow of a leakage current through the trap states and (iii) Joule heating which generates additional oxygen vacancies. These processes result in a rapid increase of the local concentration of oxygen vacancies, forming a continuous conductive path or filament (CF) between the two electrodes. This process, which turns the device 'on' from the initial 'off' state, is known as the 'set' operation. A current compliance is typically used during the 'set' operation to limit the current flow through the device and protect it from a permanent dielectric breakdown. The procedure to turn the 'off' is known as the 'reset' operation. For a bipolar RRAM as shown in figure 1.2, the 'reset' operation involves application of an electric field in the opposite direction (in comparison to the 'set' operation). This partially re-oxidizes the CF and increases the overall resistance.

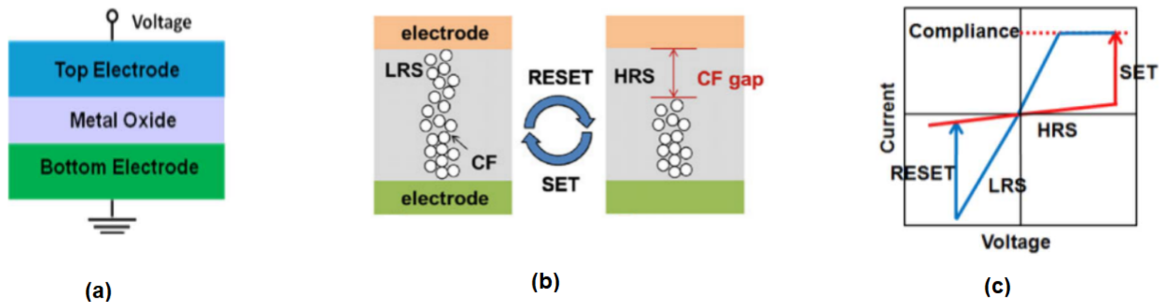


Figure 1.2: (a)Metal-insulator-metal structure of RRAM devices (b)Schematic of the LRS and HRS state depicting the growth and rupture of the conductive filament (c)Schematic of the I-V curve of RRAM devices[3]

1.2 Memristors and Artificial Neural Networks

Over the past decade, rapid advances in deep learning have led to a revolution in the field of artificial intelligence. The success of deep learning can be attributed to significant improvements in machine learning algorithms and continuous growth in computing power. At the core of deep learning lies the artificial neural network, which is a computational model loosely inspired by the human brain. Computation in ANNs is mainly comprised of massive matrix multiplications which can be broken down into multiple smaller tasks that can be executed in parallel. This has led to the usage of dedicated hardware such as GPUs, FPGAs and CMOS-ASICs for executing the massive computational requirements of deep learning. As the field continues to advance, the growing complexity of ANNs has caused an exponential increase in the demand for computing power. Since 2012, the amount of data used for training the largest deep learning models has roughly doubled every 3.4 months[4] which is much faster than hardware improvements due to Moore's law (which itself has been slowing down). Moreover, the implementation of ANNs on conventional CMOS digital computers is constrained by the Von Neumann bottleneck (figure 1.3). Due to the physical separation between the memory and the processing unit, vast amounts of data have to be shuffled back and forth which increases the execution time as well as the energy consumption while executing ANN workloads. Thus, there is a need to explore novel computational paradigms in which the memory and processing units are co-located. In-memory computing is one such approach, where data is not just stored in the memory but the computations are also performed at the same physical locations. It aims to eliminate the Von-Neumann bottleneck by implementing computation directly within the memory. The properties of a memristor make it a natural choice for implementing in-memory computing. The memristor crossbar array, as shown

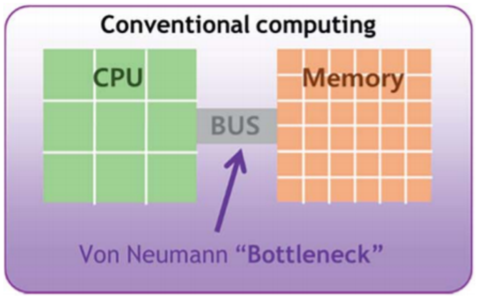


Figure 1.3: The von-Neumann bottleneck in conventional computers[5]

in figure 1.4, has shown promising results for implementing ANNs in an in-memory fashion. The essential idea is to store the synaptic weights of the ANN as the conductance of the memristors. The crossbar array is used to perform vector-matrix multiplications of the input data with the synaptic weights in a single step. A vector of voltage signals is applied at the input rows. The conductance of each memristor is multiplied with the input voltage to produce a current. Across each column, the current is summed up in accordance with the KCL(Kirchoff's current law). The output currents at each column are a result of the multiply-add operations.

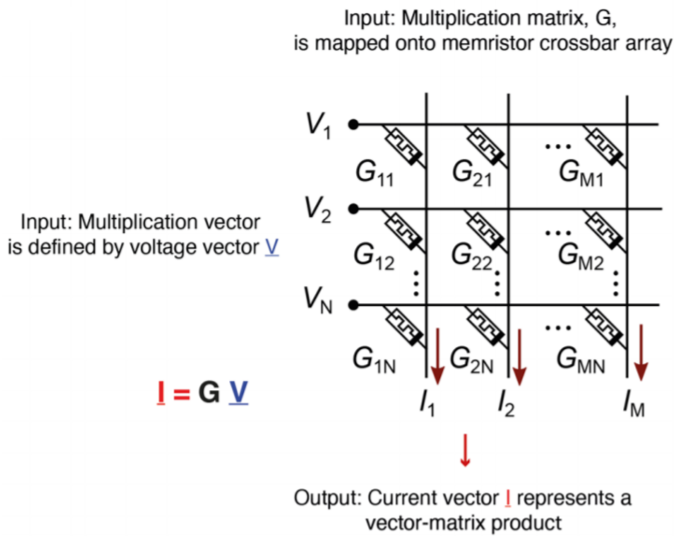

Input: Multiplication matrix, G, is mapped onto memristor crossbar array
V_1
V_2
V_N
G_{11}
G_{21}
G_{M1}
G_{12}
G_{22}
G_{M2}
G_{1N}
G_{2N}
G_{MN}
Input: Multiplication vector is defined by voltage vector V
I = G V
I_1
I_2
I_M
Output: Current vector I represents a vector-matrix product

Figure 1.4: Memristor crossbar array for vector-matrix multiplications in ANNs[6]

1.3 Memristors and Spiking Neural Networks

Although ANNs have been successful in a variety of tasks such as object recognition, machine translation and speech recognition, they require huge amounts of labeled data and consume significant amount of energy. Despite using customized hardware to implement ANNs, they are nowhere near the energy efficiency of the human brain. Moreover, the backpropagation algorithm used for training ANNs is unlikely to be the learning mechanism in the human brain[7]. Thus, there have been significant research efforts worldwide in developing a fundamentally different approach to computing inspired by the human brain.

Spiking neural networks (SNNs) have been proposed as a biologically-plausible computational model that resembles the learning mechanism of the human brain. The building blocks of a SNN are the integrate-and-fire neurons and the synapses connecting the neurons. The fundamental difference between SNNs and ANNs is that the information is encoded in the form of spikes in SNNs. The operation of a SNN can be stated as follows: Information about external stimuli (such as images or speech) is encoded in patterns of spike trains. The arrival of a spike at a synapse triggers a current flow into the downstream neuron, with the magnitude of the current weighted by the effective conductance of the synapse. The incoming currents are integrated by the neuron to determine its membrane potential, and a spike is fired when the potential exceeds a threshold. Once a spike is fired, the membrane potential is reset to a resting potential. Learning in SNNs occurs via the STDP (Spike-timing dependent plasticity) learning rule. Researchers in neuroscience have discovered that STDP learning is used in many parts of the human brain to modify synaptic strengths for learning real-world representations. Similarly, the synaptic weights in a SNN are modified according to the STDP learning rule. This learning rule can be stated as follows: if the post-synaptic neuron fires after the pre-synaptic neuron, the synapse connecting them is strengthened. Instead, if the pre-synaptic neuron fires after the post-synaptic neuron, it weakens the synapse connecting them. As evident, the information required to update the synaptic weight is both local to the synapse and is local in time. Thus, STDP is a local learning rule unlike the backprop algorithm used in ANNs which requires global information.

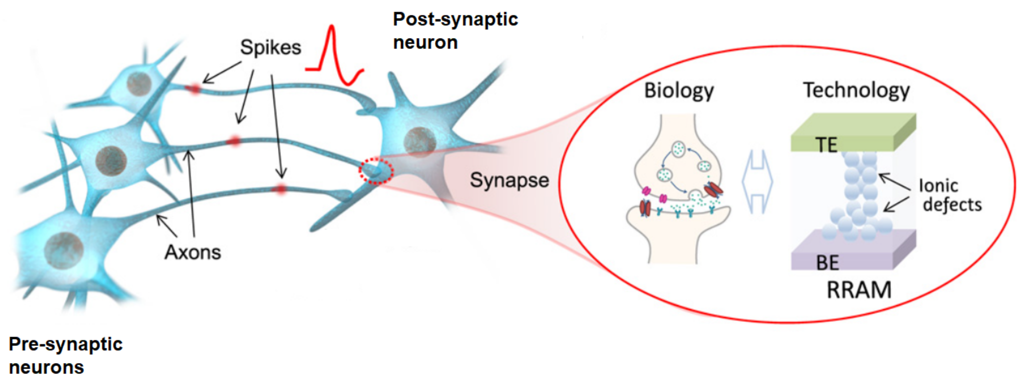


Figure 1.5: Diagram of biological neural subsystem depicting the neurons and synapses. The RRAM is the electronic equivalent of the biological synapse[8]

It turns out that the properties of a memristor closely resemble the STDP behavior of biological synapses. Memristors are the natural choice for implementing synapses in SNNs. The conductance of a memristor is analogous to the synaptic strength of a biological synapse. Its conductance can be changed by applying appropriate voltage pulses at its top and bottom electrodes similar to the pre-synaptic and post-synaptic spikes. Moreover, the extent of change depends on the relative timing of the pre and post-synaptic pulses. STDP has been intensively investigated in the memristors realized using bipolar RRAM devices. Many researchers[9][10] have experimentally demonstrated the STDP characteristics of such devices. Figure 1.6 shows the relative change of the conductance(synaptic weight) as a function of the relative spike timing for a 4-nm-thick HfO_2 device. This is consistent with the STDP behavior of biological synapses.

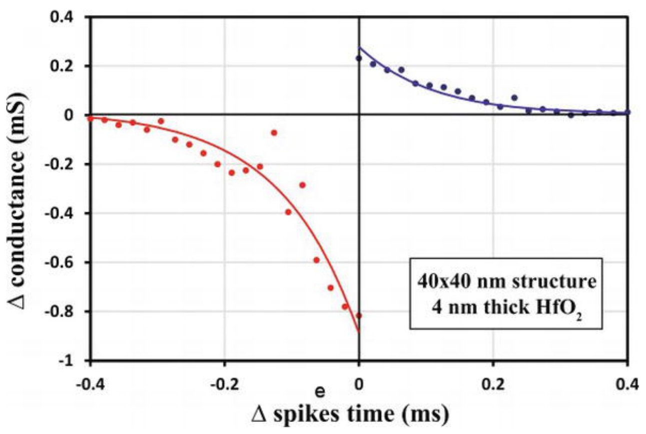


Figure 1.6: Experimental STDP function of an HfO_2 RRAM device[11]

Our work begins with exploring the RRAM device as a memristor and investigating some of its properties which enable its usage as an artificial synapse. Our work also incorporates variability in the RRAM compact model. We then use the RRAM model as a synapse to design spiking neural networks for the task of spoken digit recognition. We explore both the supervised as well as the unsupervised approaches for training the spiking neural network. We also take a detour to demonstrate the usage of the memristor crossbar array for implementing artificial neural networks.

The STDP Learning Rule for RRAM Devices and Variability in RRAM Devices

The human brain is perhaps the most sophisticated creation of nature. The brain's ability to perform complex tasks depends on specialized cells that make up the nervous system. These specialized cells are the neuronal cells, more commonly knows as the neurons. Neurons are connected to each other by synapses and they are essential to the transmission of information between neurons. Synapses play a major role in the learning process and in the formation of memory as they control the strength of the signals transmitted between neurons. The synapses have the ability to weaken or strengthen the signals over time according to the number of stimuli received during a learning process. This phenomenon is called synaptic plasticity[12] and it enables the brain to respond appropriately to changes in the external environment. While there are many mechanisms that can cause synaptic plasticity, a mechanism known as Spike-timing dependent plasticity (STDP)[13] has been observed in a wide variety of neural circuits. As the name suggests, the plasticity of the synapse depends on the relative timing of the pre-synaptic and post-synaptic spikes. If the post-synaptic neuron spikes after the pre-synaptic neuron, it implies a causal relation and the synaptic strength increases. This is called as long-term potentiation (LTP). Instead, if the post-synaptic neuron spikes before the pre-synaptic neuron, it implies an anti-causal relation and the synapse is weakened. This is known as long-term depression (LTD). The change in the synaptic strength as a function of the relative timings of the pre- and post-synaptic spikes is called the STDP window and is depicted in figure 2.1. The exponential nature of the STDP window with respect to the relative timing of spikes is quite evident. The exponential function has been the popular choice to fit experimental STDP data. It is mathematically expressed as

$$\Delta w = \begin{cases} A_{pre} e^{-\frac{\Delta t}{\tau_{pre}}} & \Delta t > 0 \\ A_{post} e^{-\frac{\Delta t}{\tau_{post}}} & \Delta t < 0 \end{cases}$$

where Δw denotes the change in the synaptic strength, Δt is the time delay between the pre and post-synaptic spikes, τ_{pre} and τ_{post} are the time constants. They are in the order of milliseconds. A_{pre} and A_{post} are fitting parameters.

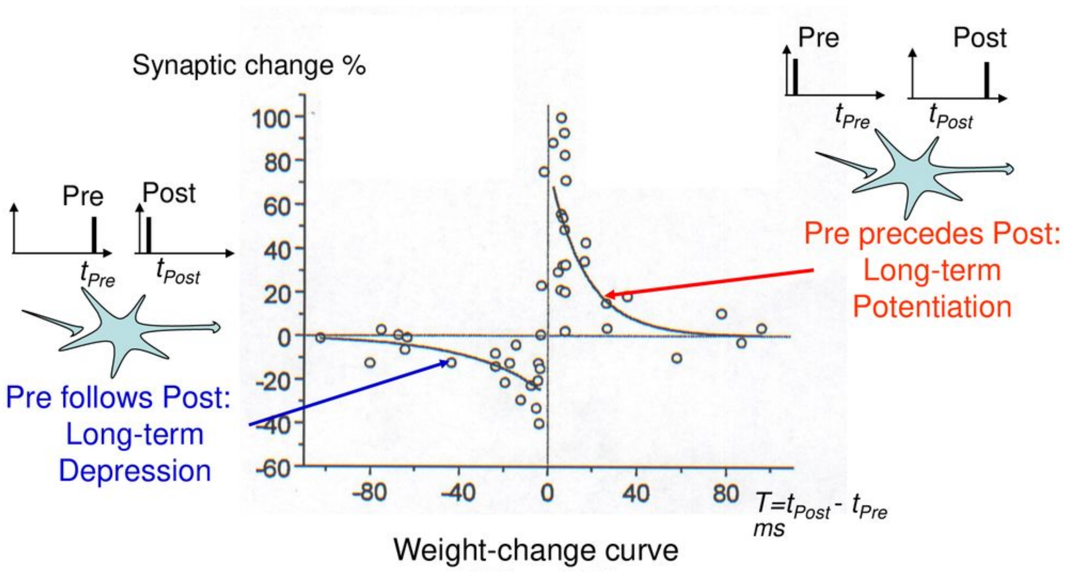


Figure 2.1: STDP window[14]

2.1 Memristors and STDP

The memristor can quite closely mimic the STDP behaviour of biological synapses. The conductance of the memristor can be changed by applying appropriate voltage pulses across its terminals. Basically, the conductance of the memristor is analogous to the synaptic strength of the biological synapse. The strength of a biological synapse is determined by its pre- and post-synaptic spiking history.

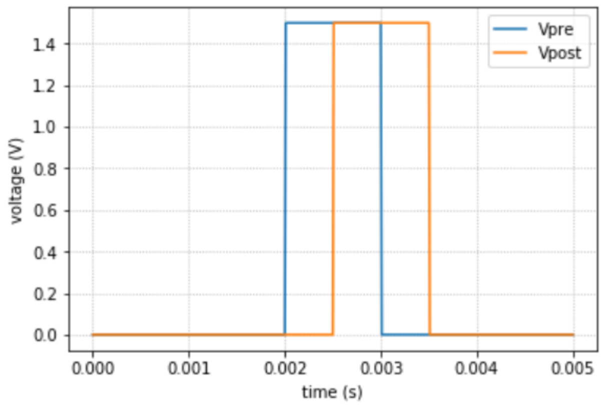
Similarly, the conductance of a memristor is a function of the amount of charge that has flown through it due to the voltage pulses applied to it earlier. There are certain conditions that need to satisfied to ensure that a RRAM device can properly function as a synapse:

1. If only one of the pre-synaptic or the post-synaptic neuron fires a spike within a given time interval, the conductance of the device should not change. To ensure this, the voltage pulses applied by the pre and post-synaptic neurons should be below the threshold voltage of the device.

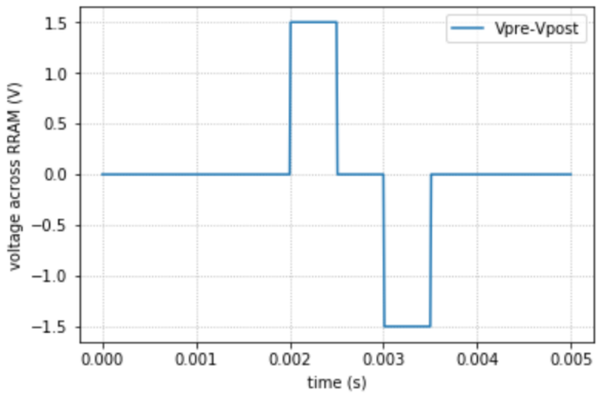
2. If both the pre and post-synaptic neurons fire in close proximity, the conductance should undergo a change. This places further restrictions on the shape and the amplitude of the voltage pulses. The voltage pulses cannot be unipolar. For example, if the device threshold is 2 V, the pulses must have an amplitude less than 2 V.

We apply unipolar voltage pulses of amplitude 1.5 V at both the terminals of the RRAM as shown in figure 2.2a (denoted by V_{pre} and V_{post}). The net voltage across the terminals of the device will be $V_{pre} - Vpost$ as shown in the figure 2.2b. The effective voltage does not cross the threshold voltage and thus the state of the device remains unchanged despite both the pre and post synaptic spikes firing in close proximity. Hence, unipolar pulses aren't suitable for STDP.

Instead, if bipolar pulses are applied at both the terminals of the device, the effective voltage across the device goes above the threshold voltage for a certain duration of overlap. This results in a change in the state of the device as expected according to the STDP learning rule.

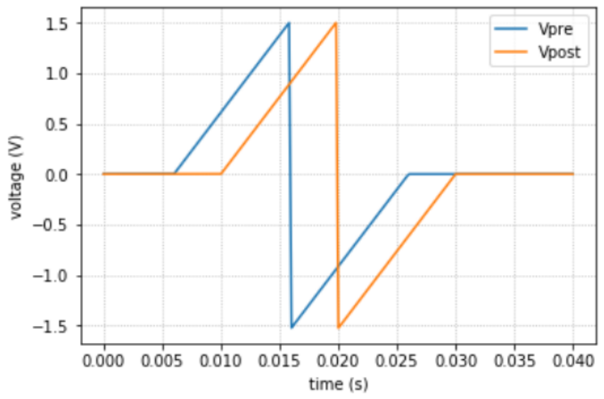


(a) Unipolar pre and post synaptic pulses

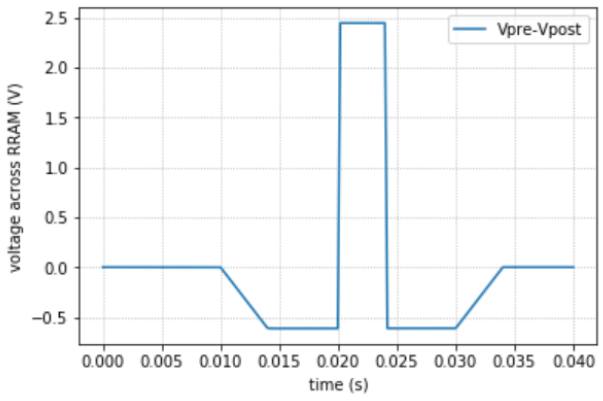


(b) Effective voltage across the RRAM

Figure 2.2: Unipolar pulses



(a) Bipolar pre and post synaptic pulses



(b) Effective voltage across the RRAM

Figure 2.3: Bipolar pulses

2.1.1 RRAM Compact models

Compact models of the RRAM device are essential for fast and accurate circuit simulation. Many compact models of RRAMs[15, 16, 17, 18] have been proposed by researchers. From an application viewpoint, we require a model that is able to demonstrate the STDP behaviour of synapses. As discussed in the previous section, the model must be able to handle bipolar voltage pulses. We briefly discuss a few compact models that are currently available.

Stanford RRAM model

The Stanford RRAM model[15] has become a popular open source model. It is a physics-based model that takes into account the effect of Joule heating on the switching characteristics of the device. The conductive filament is modeled as a cylindrical shaped filament. The gap between the tip of the conductive filament and the bottom electrode is the internal state variable of the compact model. This model assumes that the radius of the conductive filament is constant and thus the switching dynamics are governed only by the evolution of the gap. The conductance of the device increases as the gap is decreased (conductive filament grows) by applying a positive voltage across the top and the bottom electrodes. The rate of change of filament gap(g) is given as:

$$\frac{dg}{dt} = V_0 exp\left(\frac{-E_a}{k_b T}\right) sinh\left(\frac{\gamma a_0 q V}{t_{ox} k_b T}\right)$$

where V is the applied voltage across the cell, I is the current through the cell, g is the average tunneling gap distance, Ea is the activation energy, a_0 is the atomic spacing, t_{ox} is the oxide thickness, q is the elementary charge, k_b is the Boltzmann constant, $\gamma = \gamma_0 - \beta g^3$ and $T = T_0 + V.I.R_{th}$, where T_0 is the room temperature, R_{th} is the equivalent thermal resistance, and I_0, g_0, V_0, $_0$, β, $_0$ are fitting parameters.

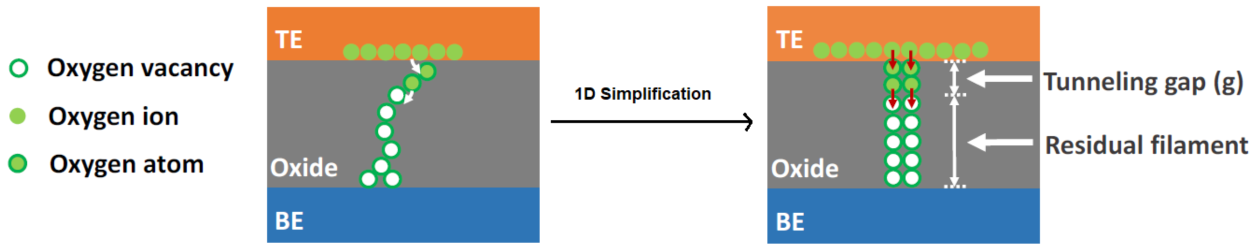


Figure 2.4: Stanford RRAM model[15]

Gonzalez RRAM model

Their model[16] assumes a truncated cone shaped filament whose radius is highest near the top electrode and decreases towards the bottom electrode. It uses the redox and diffusion processes to describe the physics behind the switching mechanism. Two different temperatures are used in the model: the first one approximates the main conductive filament body that is not destroyed in the reset process and the other temperature describes the narrowing of the conductive filament. This model assumes that there is no gap between the conductive filament and the electrode. Thus, the length of the conductive filament is constant.

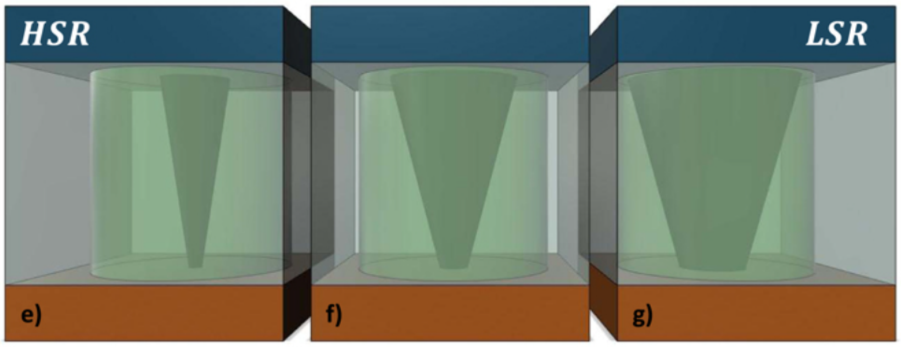
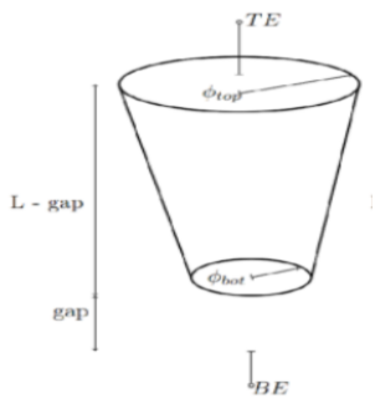
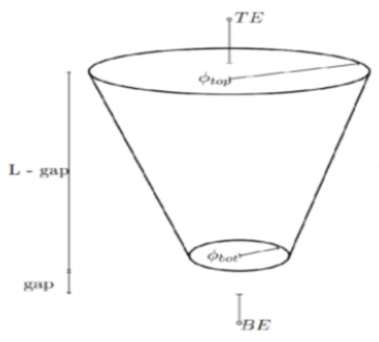


Figure 2.5: Gonzalez et al RRAM model[16]

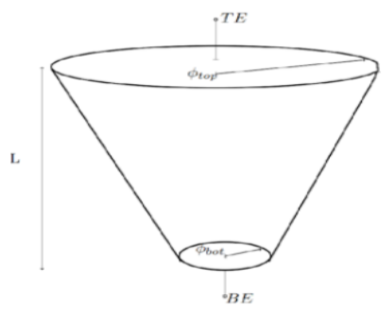
These models have certain shortcomings, for example, in the Stanford model, the radius of the conductive filament is fixed and the gap mainly determines the conductance of the device. In Gonzalez's model, the gap is assumed to be negligible and only the radii govern the conductance. A more realistic approach



(a) HRS(high resistance state)



(b) Intermediate state



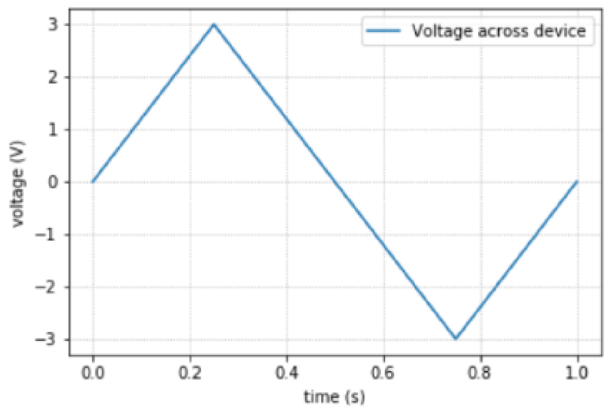
(c) LRS(low resistance state)

Figure 2.6: Unified RRAM model

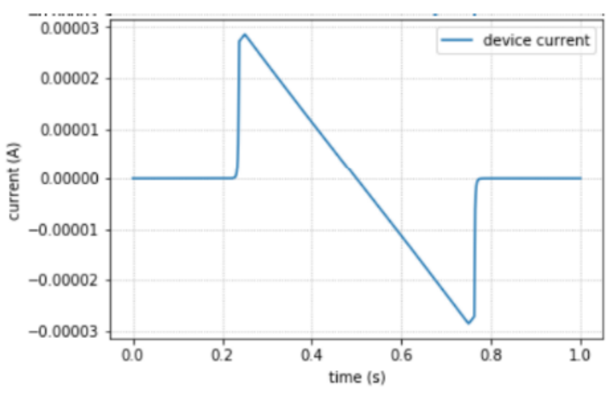
would be to have both the gap and the radius as the state variables that govern the switching dynamics of the device. Moreover, it turns out the these models cannot directly handle bipolar voltage pulses in the same simulation run. As we've explained above, bipolar pulses are crucial to demonstrate the STDP learning rule in RRAM devices. In order to improve upon the existing models, a unified RRAM model has been developed by our research group.

The unified RRAM model is more realistic as it assumes a truncated-cone shape for the conductive filament along with a variable gap between the filament and the bottom electrode. This model uses the gap as well as the radius as internal state variables and it also handles bipolar pulses in a robust manner. Figure 2.6 depicts the growth of the conductive filament as the device transitions from HRS(high resistance state) to LRS(low resistance state). We expect the both the top and bottom radius to increase and the gap to decrease as the device goes to the LRS state.

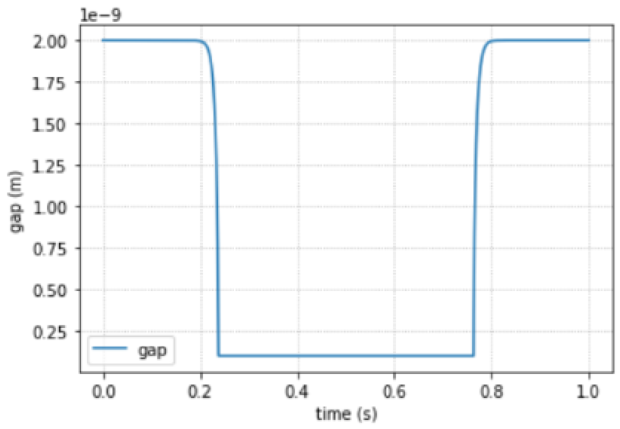
This verilog-A model was simulated in both Cadence Virtuoso and QUCS studio circuit simulator. In our simulations, we performed a quasi-DC sweep of the RRAM model to observe the SET-RESET behavior of the device (figure 2.9). The device is in the HRS at t=0. The gap is at its maximum value and the radii are at their minimum values. As the voltage across the top and



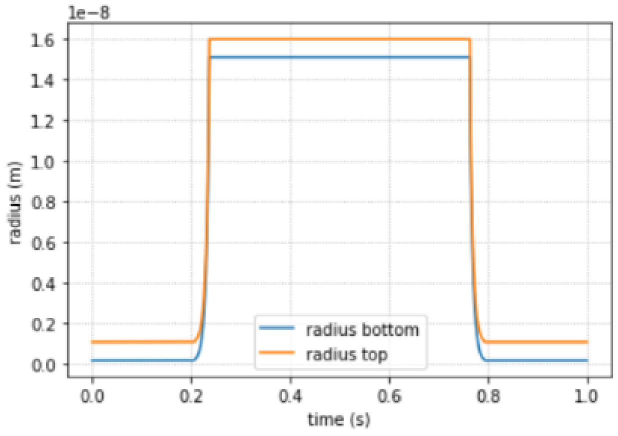
(a) Voltage



(b) Device Current



(c) Gap



(d) Radius

Figure 2.7: Unified RRAM model SET-RESET operation

the bottom electrode is increased gradually, there is a very small increase in the current of the device. Once the applied voltage crosses a certain threshold, there is an abrupt increase in the device current. The gap of the device also abruptly decreases to its minimum value and the radii increase to their maximum values. Thus, the model is able to incorporate the abrupt nature of the transition from HRS to LRS. Now, the applied voltage starts is decreased towards negative values and the device goes from the LRS to the HRS. Since the device is symmetric, the transition from the LRS to the HRS also occurs in an abrupt fashion. The gap and the radii return to their original state (same as t=0).

The I-V curve for the SET-RESET operation is shown in figure 2.8

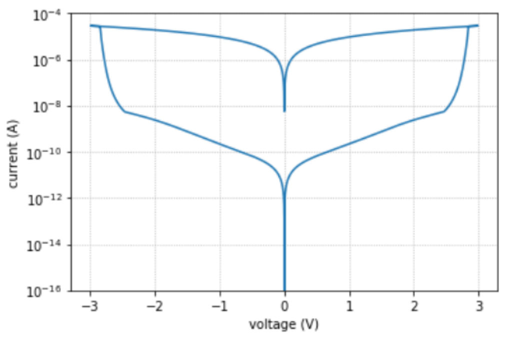


Figure 2.8: Current-Voltage curve of the unified RRAM model

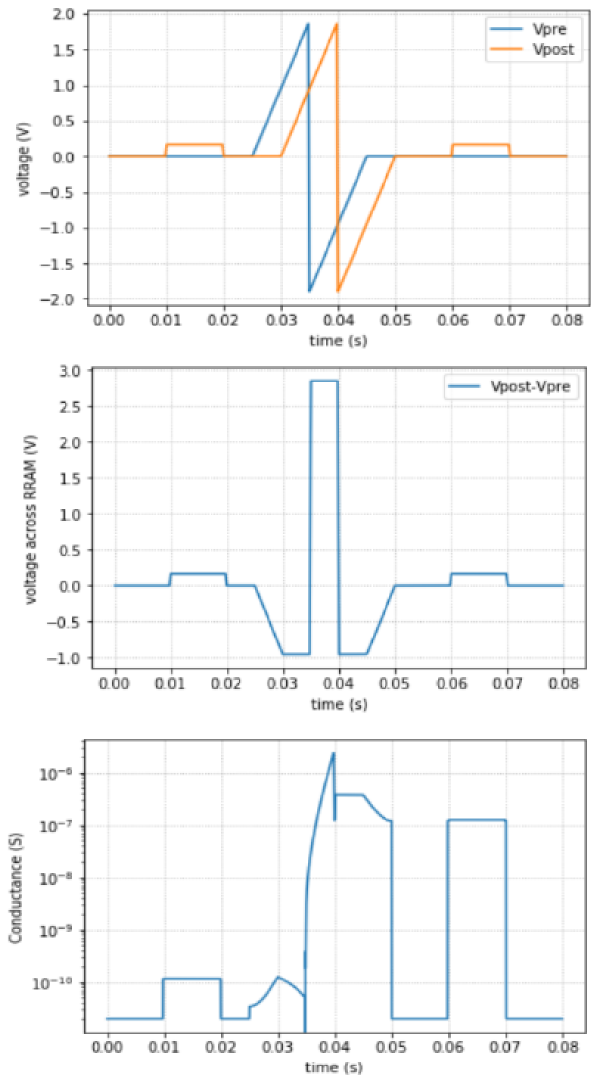
2.1.2 STDP demonstration using the unified RRAM model

The STDP behavior of the RRAM model is demonstrated by performing certain simulations to cover all possible scenarios. In the first set of simulations, we consider four possible scenarios:

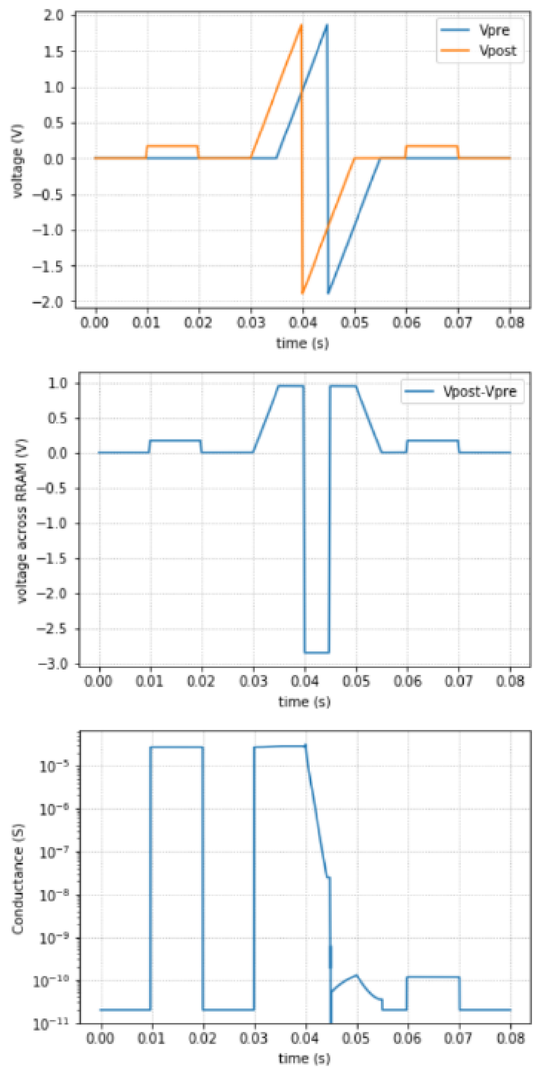
1. The pre-synaptic spike occurs before the post-synaptic spike
2. The pre-synaptic spike occurs after the post-synaptic spike
3. Only the post-synaptic synaptic spike occurs
4. Only the pre-synaptic spike occurs

The 1^{st} case corresponds to LTP, i.e the synaptic strength increases which in turn implies that the conductance of the device should increase. The 2^{nd} case corresponds to LTD, i.e the synaptic strength decreases which in turn implies that the conductance of the device should decrease. In the 3^{rd} and the 4^{th} case, the synaptic strength remains unchanged and thus the conductance should remain unchanged.

The spikes occur over few 10s of milliseconds and the duration of overlap is a few milliseconds. Bipolar voltage pulses with peak voltage of 1.5 V are used as the pre and post-synaptic spikes. The peak voltage of each individual pulse is less than the threshold voltage of the device. In order to measure the conductance of the device before and after the event, a test voltage of 100 mV



(a) Pre-synaptic before post-synaptic spike

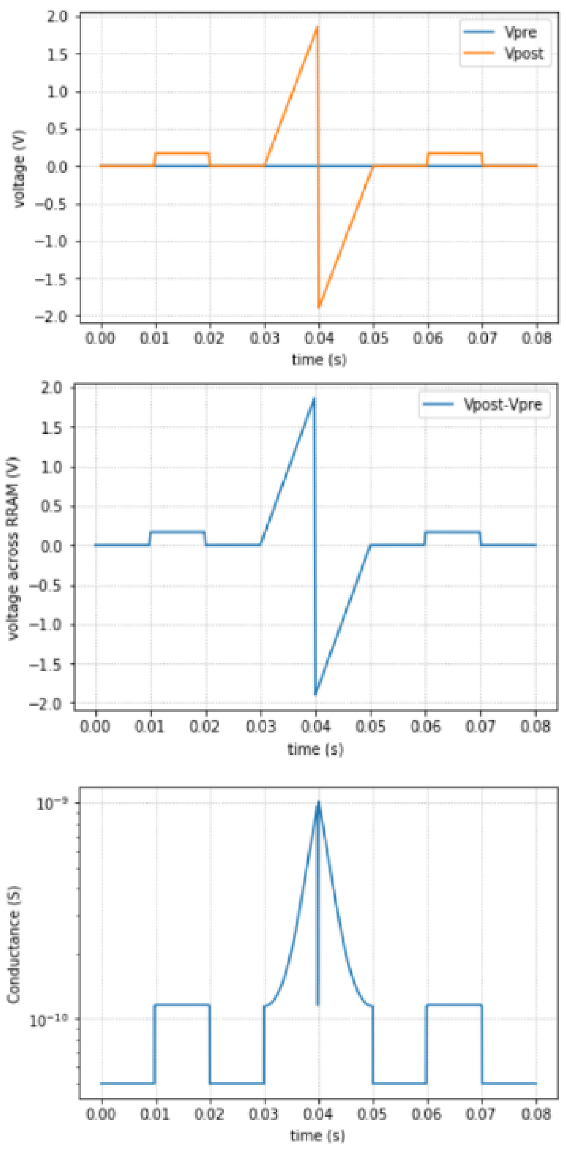


(b) Pre-synaptic after post-synaptic spike

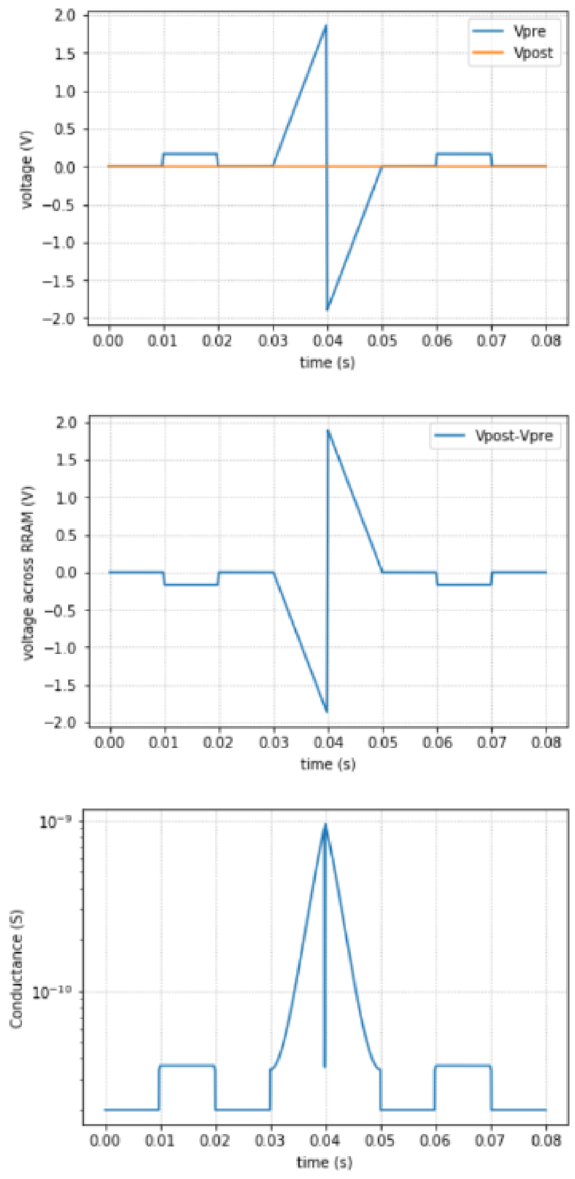
Figure 2.9: STDP demonstration using the compact model

is applied and the current is measured. In the below figure, the scenarios (a) to (d) are depicted. For each scenario, three graphs have been plotted. The topmost graph shows the pre-synaptic and the post-synaptic spikes. The graph in the middle shows the effective voltage across the terminals of the device. The graph at the bottom shows the measured conductance of the device before and after the spikes have occurred. Clearly, the change in the conductance for each scenario exhibits the expected STDP nature.

In the second set of simulations, the dependence of the change in the device

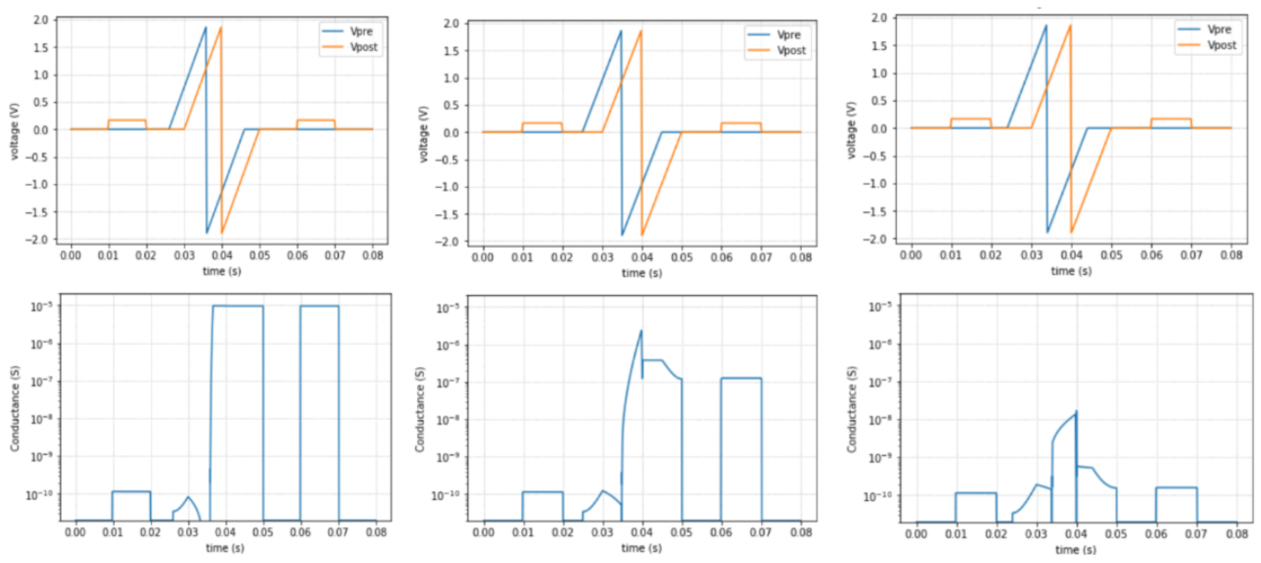


(c) Only pre-synaptic spike



(d) Only post-synaptic spike

Figure 2.9: STDP demonstration using the compact model



(a) Delay = 4 ms (b) Delay = 5 ms (c) Delay = 6 ms

Figure 2.10: STDP:Effect of time delay between pre and post-synaptic spikes on the conductance of the device

conductance as a function of the time delay between the pre-synaptic and post-synaptic spikes is demonstrated. In all the graphs shown below, the pre-synaptic spike occurs before the post-synaptic spike but the time delay between the spikes is different in each case. For the graph in fig 2.10a, the delay is 4ms. For the middle graph in fig 2.10b, it is 5 ms and for the graph in fig 2.10c, the delay is 6 ms. The pre and post-synaptic voltage pulses and the conductance of the device are plotted in each case. The initial state of the device is the same in all the cases. From the conductance graphs, it can be observed that as the time delay between the pre and post-synaptic spike increases, the change in conductance decreases in an exponential manner. This closely resembles the exponential dependence of STDP on the relative timing of the pre and post synaptic spikes. Thus, we have demonstrated that the unified RRAM model shows all the desired properties of the STDP learning rule and it closely resembles the behavior of biological synapses. The model can now be used to implement the synapses in spiking neural networks.

2.2 Variability in RRAM devices

The formation and rupture of the conductive filament in an RRAM is due to the generation and diffusion of oxygen vacancies. This process is inherently stochastic and it causes variations in the size (and shape) of the conductive filament. This leads to variability in the device current. In order to make the RRAM model more realistic, the model must be able to account for these variations. Variability in RRAM devices can be of two types : (i) cycle-to-cycle variations for a given device and (ii) device-to-device variations. Our work incorporates variability in the unified RRAM model.

As mentioned earlier, the model has two internal state variables, i.e the gap and the radius of the conductive filament. Both of these play an important role in determining the device current (and conductance). Variability in the device current can be modeled as a consequence of fluctuations in the gap and the radius of the conductive filament. We've considered multiple approaches for adding variations in the gap and the radius.

2.2.1 Direct addition of temperature-dependent gaussian noise

This is the simplest approach to add variations in the gap and the radius. The standard gaussian distribution is sampled at every time step of the simulation. The sampled value is scaled by $\delta(T)$ and then added to the original value of the gap. In mathematical terms,

$$g = g_{det} + \delta_g(T)N(0, 1)$$

where g_{det} is the gap obtained from the deterministic equations of the model and the temperature dependence of the variations is expressed as

$$\delta_g(T) = \frac{\delta_g^0(T)}{1 + exp\left(\frac{T_{crit} - T}{T_{smth}}\right)}$$

where T_{crit} is a threshold temperature above which variations become significant and δ_g^0 and T_{smth} are fitting parameters. Figure 2.11 shows the resultant current-voltage curve for three set-reset cycles. This approach introduces variations that appear as noise. The fluctuations occur much more rapidly than observed in experiments. Moreover, it doesn't quite capture the cycle to cycle variations as seen in experimental data.

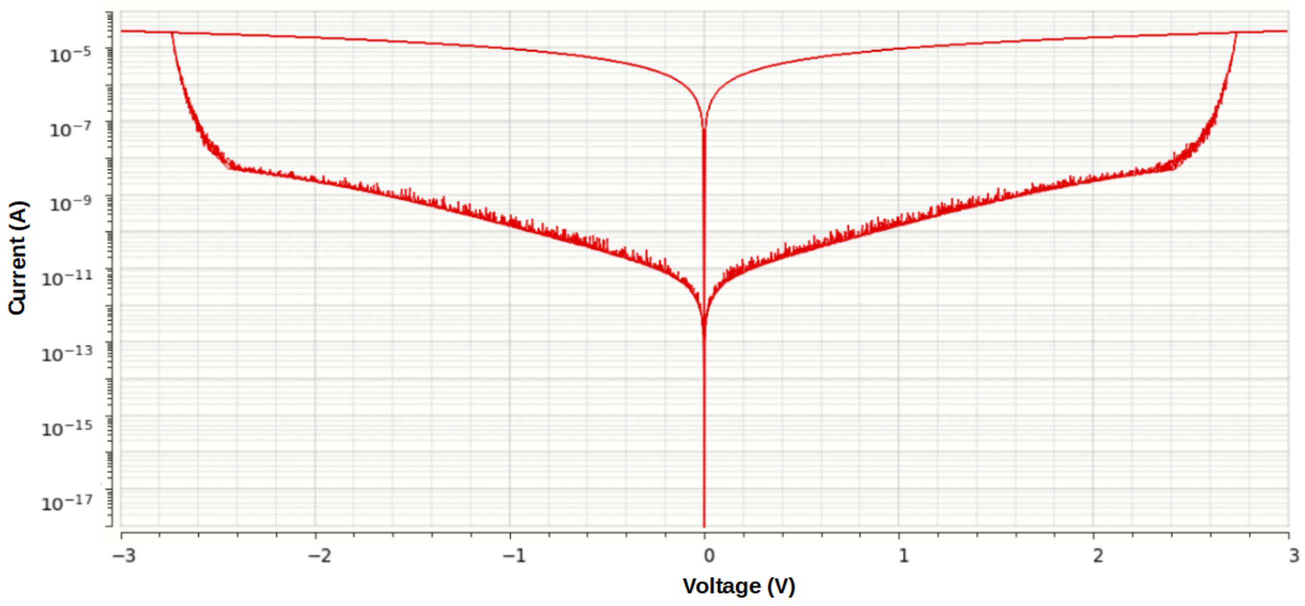


Figure 2.11: Current-Voltage curve for the direct addition variability model

The corresponding variations in the gap with respect to time are shown in figure 2.12. Variations appear only when the gap is close to its maximum value, i.e the device is in the HRS. Also, the variations are too rapid and appear unrealistic.

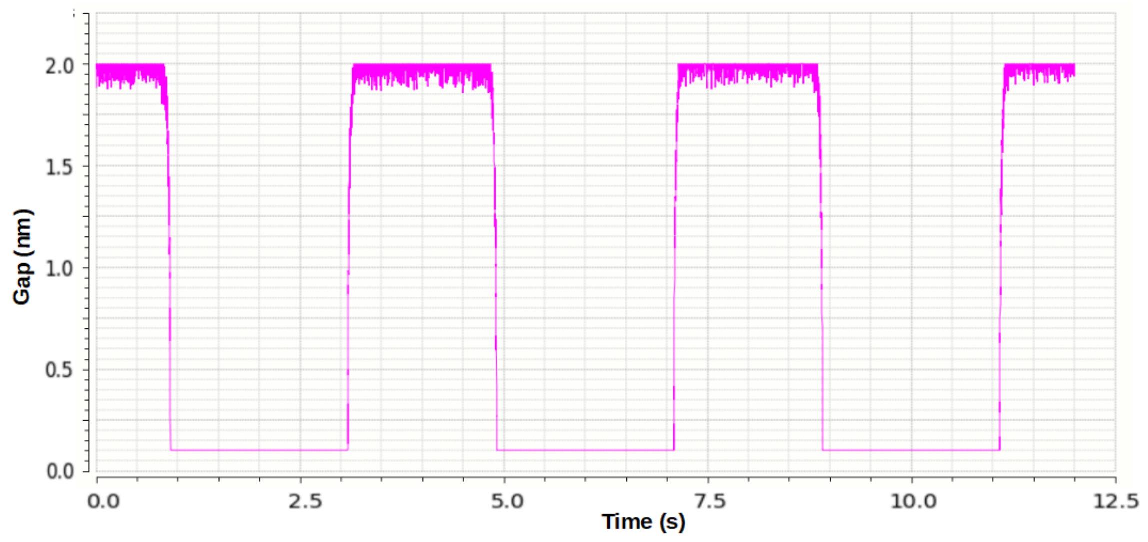


Figure 2.12: Variations of the gap for the direct addition variability model

2.2.2 Temperature-dependent Gaussian noise added to the derivative of the state variable

This method has been implemented in the Stanford model[15]. Their equations for the gap are :

$$\frac{dg}{dt} = V_0 exp\left(\frac{-E_a}{k_b T}\right) sinh\left(\frac{\gamma a_0 q V}{t_{ox} k_b T}\right)$$

$$g|_{t+\Delta t} = \int \left(\frac{dg}{dt} + \delta_g(T) N(0,1)\right) dt$$

where $\delta_g(T)$ plays the same role as discussed above. We use their approach to add variability in the unified RRAM model. The graphs below show the effect of adding variability in the gap. The simulation is performed for three set-reset cycles.

Observing the I-V plot (log-linear) in figure 2.13, we see that the variations in the HRS are depicted in a realistic manner by this approach but if we observe the I-V plot (linear-linear) in figure 2.14, it becomes evident that this approach is unable to display variability in the LRS. This is because the gap (figure 2.15) shows significant variation when it is close to its maximum value (device in

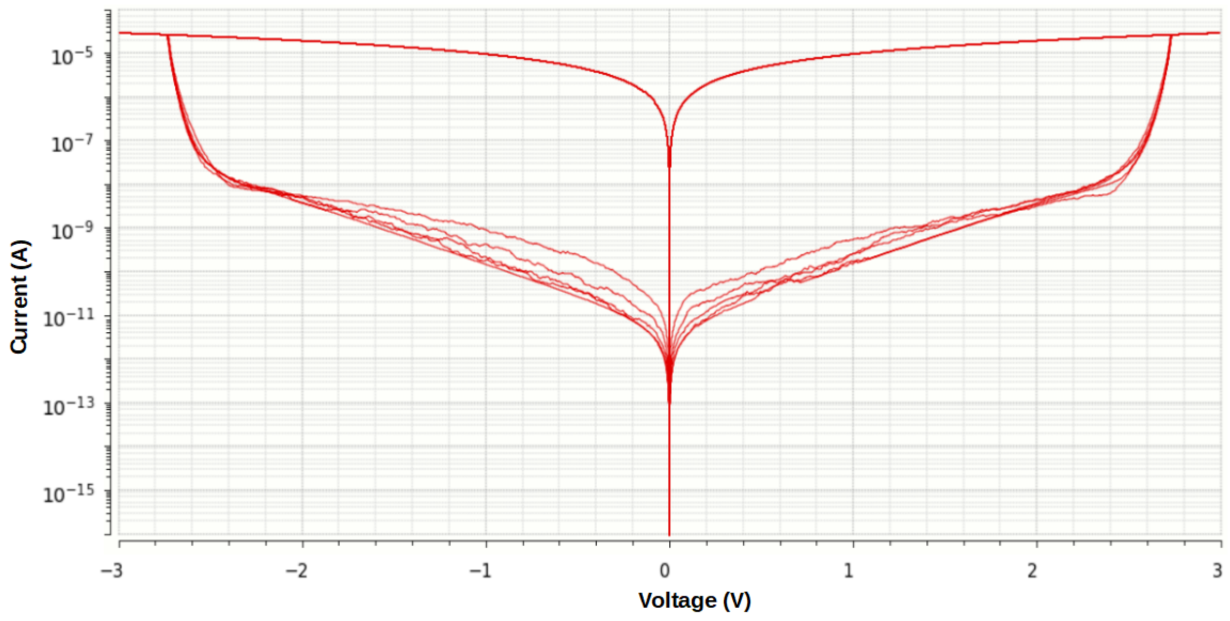


Figure 2.13: Current-Voltage (log-linear) curve for the 2^{nd} variability model depicting HRS variations

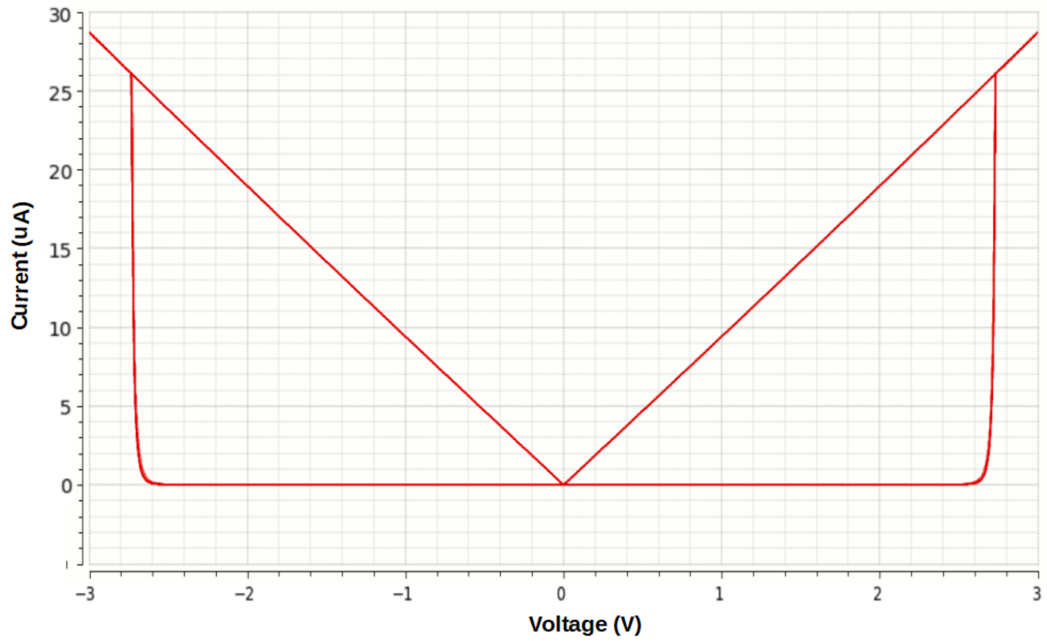


Figure 2.14: I-V (linear-linear) curve for the 2^{nd} variability model depicting absence of LRS variations

HRS) but no variations at its minimum value (device in LRS).

In order to obtain variability in the LRS, we introduced variations in the radius of the conductive filament using the same method as done for the gap. The

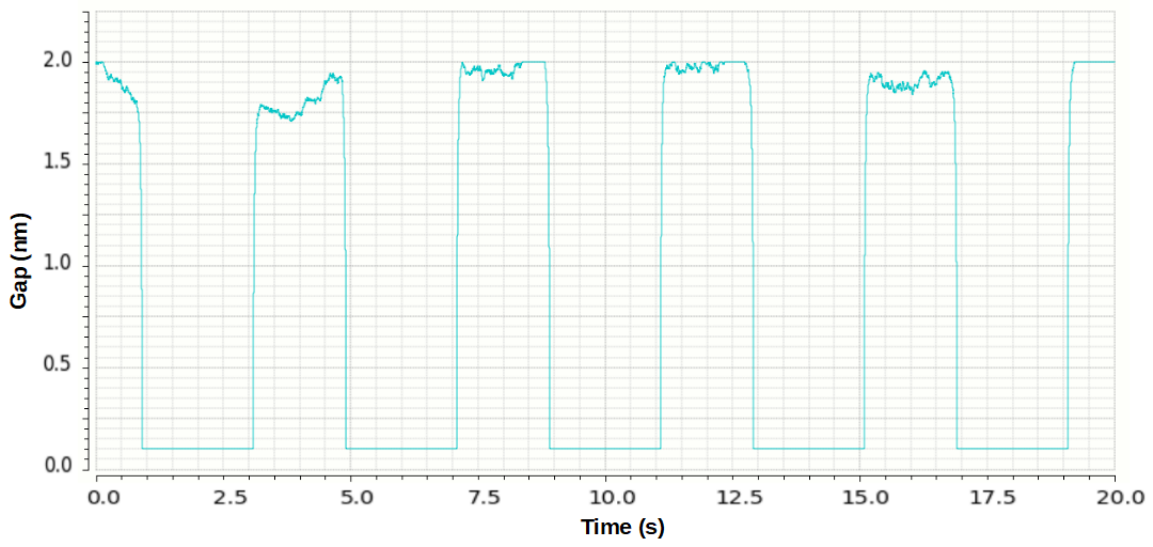


Figure 2.15: Evolution of the gap for multiple set-reset cycles

graphs below suggest that the model is still unable to display LRS variability. The graph of the radius (figure 2.17 indicates that the variations in the radius are prominent when the radius is close to its minimum value, i.e. when the device is in the HRS. There is hardly any variation in the radius when it is at its maximum value, i.e. when the device is in the LRS due to which there aren't any LRS variations in the device current. These observations suggest that a new model that accounts for both the HRS and LRS variations is required.

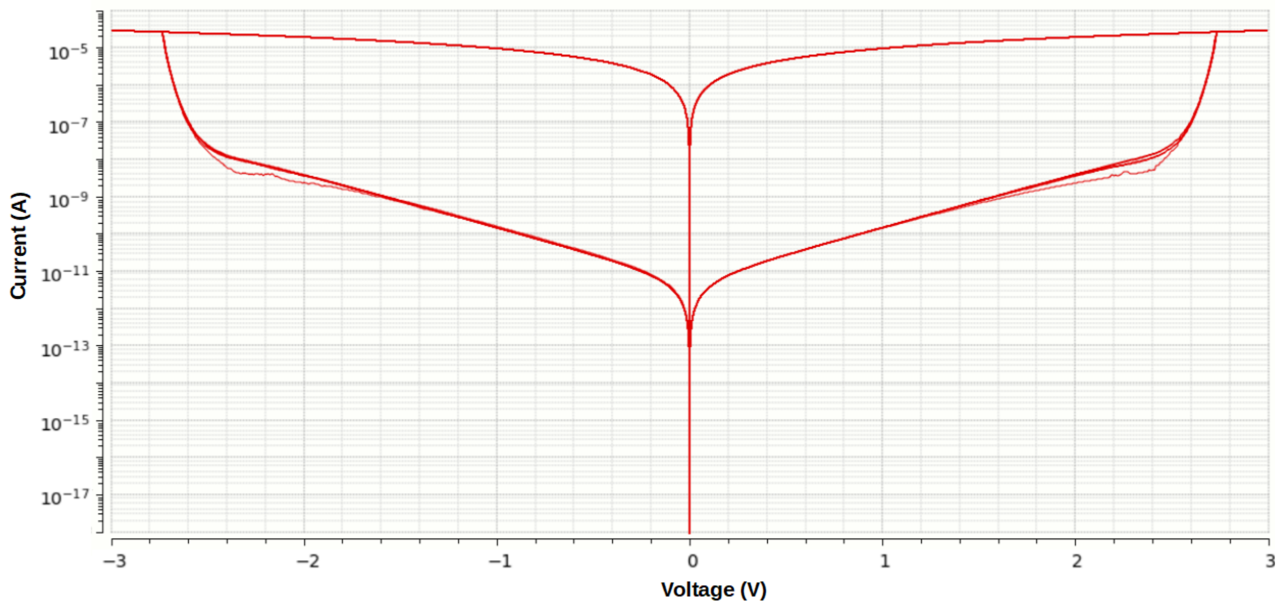


Figure 2.16: Current-voltage curve for the 2nd variability model upon adding fluctuations in the radius

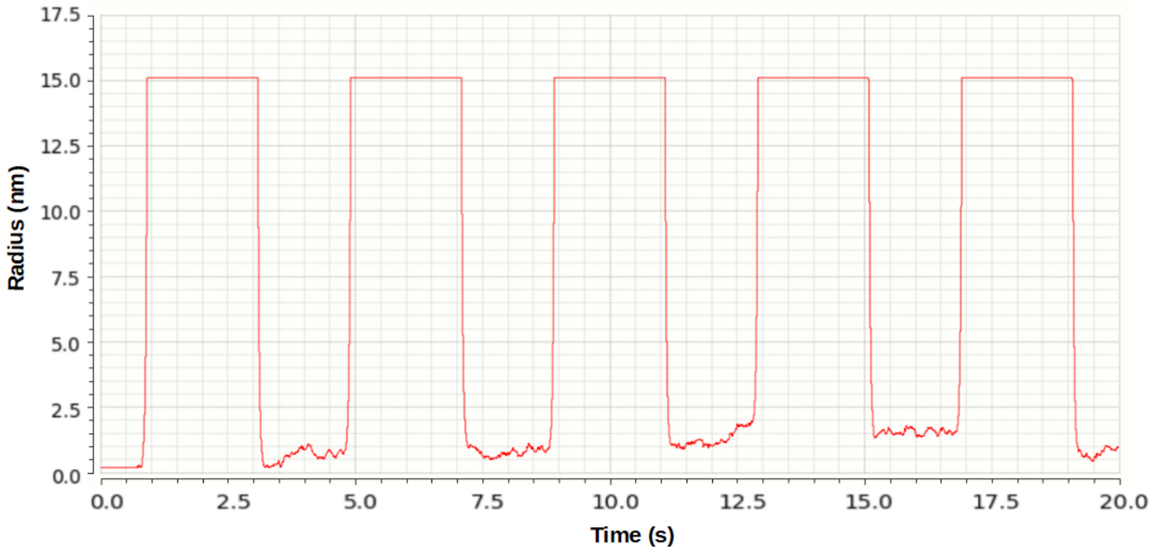


Figure 2.17: Evolution of the radius for multiple set-reset cycles

2.2.3 Variations as a function of the gap and the radius

We take a re-look at the physical process of the filament formation. When the device is in HRS, it is the gap that mainly determines the device current. So, fluctuations in the gap will show up as fluctuations in the current whereas the radius doesn't have much of a contribution in the HRS current. On the other hand, in the LRS, since the conductive filament has completely grown, the gap has decreased to its minimum value and minor variations in the gap would not have any influence on the device current. Instead, it is the radius that mainly determines the device current in the LRS. So, fluctuations in the radius account for the variability in the LRS. Also, note that the radius is close to its maximum value in the LRS and the gap is close to its maximum value in the HRS. This suggests that the amount of variation in the gap and the radius should itself be a function of the gap and the radius values. This can be done either in a direct or an indirect manner. In the unified RRAM model, the gap and radius values are being clipped to their maximum and minimum values if they are greater than the their maximum values or lesser than their minimum values. We can utilize this to check whether the gap is at its maximum value and if so, add variations in the gap. Similarly, if the radius is greater than its maximum value, variations are added to the radius. Another approach would be to express the variations as explicit functions of the gap and the radius values. We do that using the equations :

$$gap = gap_{deterministic} + V(noise_{gap}).\frac{(G1 - gap)}{gap_{max}}$$

$$V(noise_{gap}) = \int \delta_{gap}.N(0,1)dt$$

A similar equation is used for the determining the radius variations. The results are shown in the graphs below. It can be seen that this approach is able to account for both the HRS variations (log-linear graph in figure 2.18) and the LRS variations (linear-linear graph in figure 2.19).

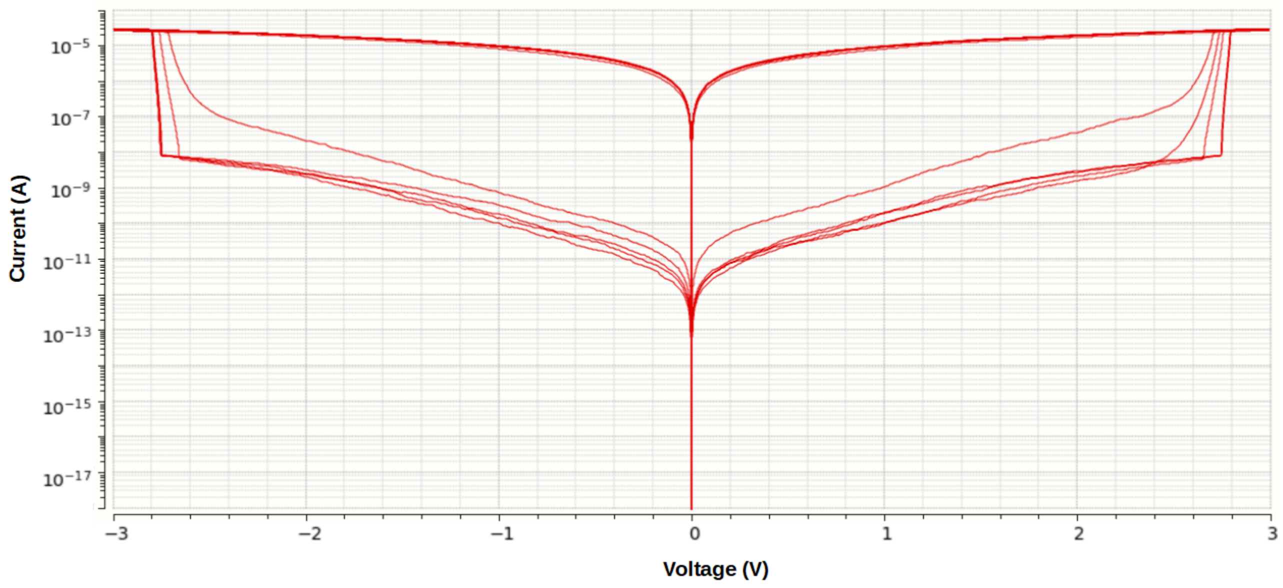


Figure 2.18: Current-voltage (log-linear) curve for the 3rd variability model

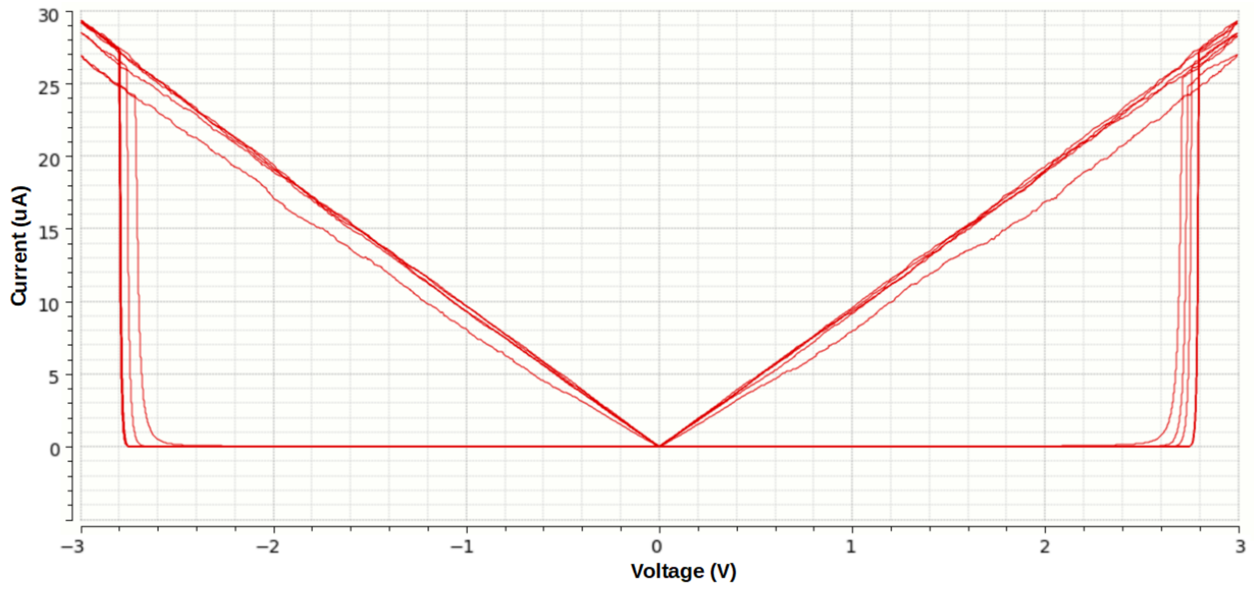


Figure 2.19: Current-voltage (linear-linear) curve for the 3rd variability model

Spiking Neural Networks for Spoken Digit Recognition

Speech is the most natural form of communication for humans. Rapid advances in technology have opened up the possibility to communicate with computers via speech. This has given rise to the field of Automatic Speech Recognition (ASR). The aim of ASR systems is to convert speech or audio signals into meaningful text that can be understood by the computer. Researchers have proposed a wide variety of approaches for speech recognition. Some of the popular methods are namely, the HMM (hidden Markov model)[19], SVM (support vector machines)[20], ANN (artificial neural network)[21] etc.

Until recent years, speech recognition systems were mainly restricted to research labs as they were sensitive to the surrounding conditions. For example, if the speech used for training the system is recorded under certain conditions, the system's accuracy would sharply decline under alternate testing conditions like noisy environments. The success of deep learning in recent years has enabled highly accurate speech recognition outside of carefully-controlled environments. It has thus become a mainstream technology in the current era and has been integrated into various real-life applications and devices.

We shall focus on a particular application of speech recognition, namely Spoken Digit Identification. The automated recognition of isolated spoken digits is crucial to many applications such as telephone based services, bank transactions, voice dialing etc. It is especially useful for the elderly and differently-abled people. Spoken digit recognition is generally a challenging task due to two reasons: I) the signals have a very short duration and ii) some digits are acoustically similar to each other. We focus on digits spoken in English but our techniques are applicable to any language.

3.1 Basic principles of automatic speech recognition

The building blocks of any speech recognition system consist of voice signal preprocessing, feature extraction and finally a pattern recognition system.

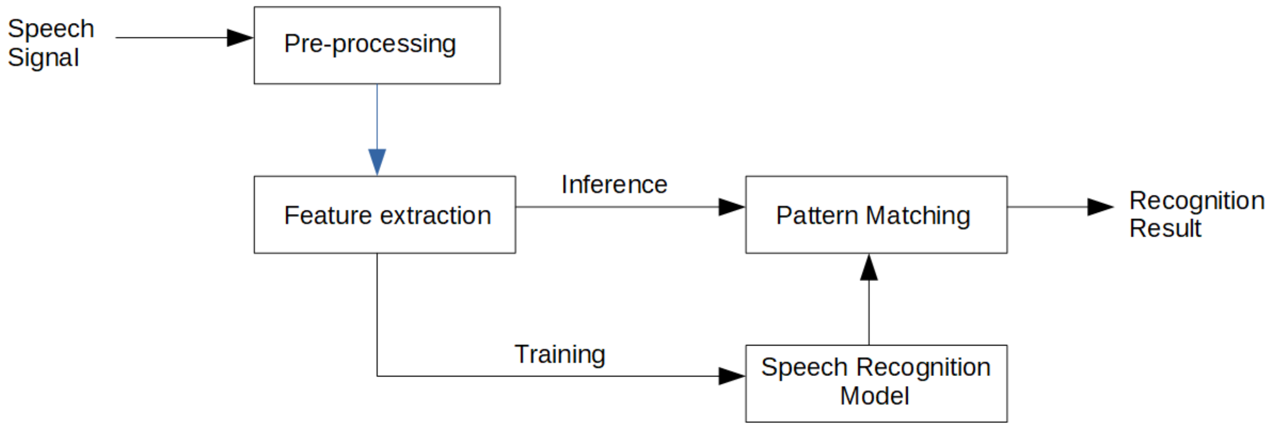


Figure 3.1: Speech recognition system

The speech spoken into the microphone is converted to an electric signal. This signal is a time-domain waveform as illustrated in figure 3.2. The time-domain waveform needs to be pre-processed appropriately before it can be used for speech recognition.

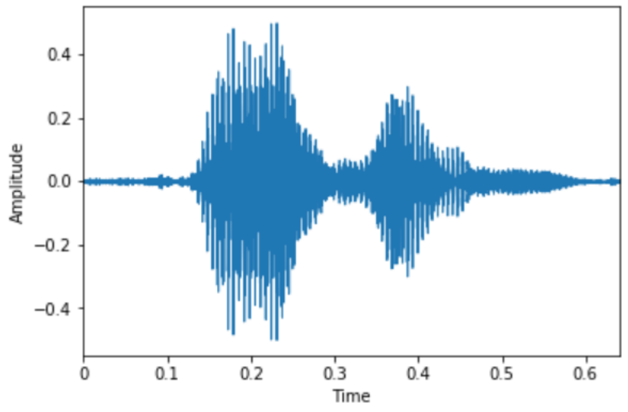


Figure 3.2: Time-domain waveform of the spoken digit "7"

3.2 Feature Extraction

A spectrogram is a powerful way of representing the signal strength, or “loudness”, of a signal over time at various frequencies present in a particular waveform. It allows us to explicitly depict the frequency content of the signal as it varies with time. Spectrograms are commonly used to display frequencies of sound waves produced by humans, birds, machines etc., as recorded by microphones.

3.2.1 Interpreting a spectrogram

Spectrograms are actually 3D plots but they are visualized in 2D using a heatmap. Time runs from left to right along the horizontal axis . The vertical axis represents the frequency with the lowest frequencies at the bottom and the highest frequencies at the top. The energy of a particular frequency at a particular time is represented by the third dimension, color, with dark blue corresponding to low amplitudes and brighter colors up through red corresponding to progressively higher energies.

3.2.2 Computing the spectrogram of a signal

In order to obtain the spectrogram, we divide the speech signal into many small overlapping time sections of equal duration. These sections are called speech frames. The total number of frames are fixed and that is determined by the number of neurons in the input layer of the spiking neural network. In order to determine the duration of each frame, we need to note that longer the duration, the better is the frequency resolution, but the time localization is poor. Similarly, the shorter the duration, the better localization in time, but the frequency resolution would be poorer. The percentage of overlap between adjacent frames is usually chosen to be 50% .

Fast fourier transform is applied to each of the speech frames in order to obtain their respective frequency spectrums. The Hamming window[22] is applied to each frame in order to smoothen it and it also has some useful frequency features. The figures below illustrates the procedure.

The frequency spectrum of a signal is typically visualized with the frequency on the x axis and the corresponding energies on the y axis. For obtaining the spectrogram, we've to rotate the frequency spectrum of each frame so that the frequency lies on the y axis (see figure 3.4). The frequency spectrum of each speech frame is arranged side by side in a chronological order to obtain the spectrogram of the signal.

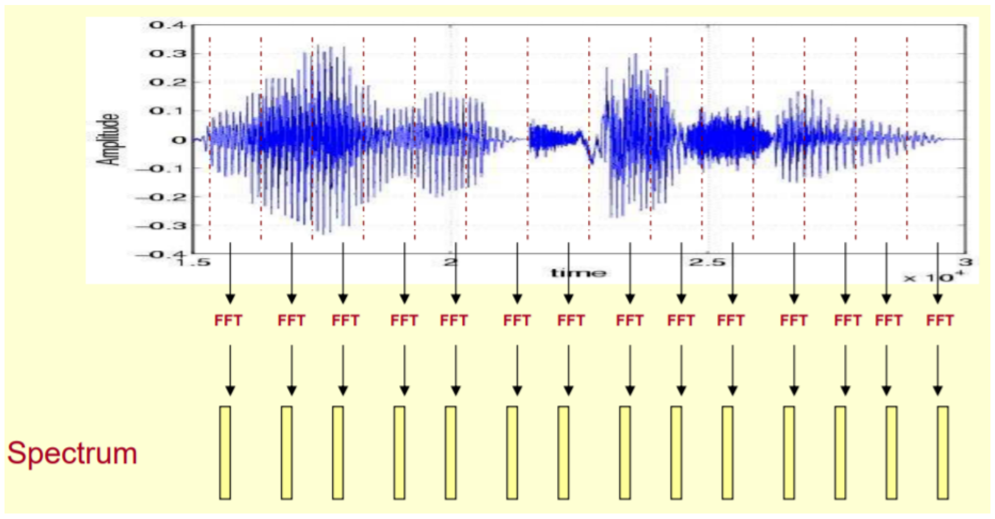


Figure 3.3: Illustration of the procedure to obtain the spectrogram of a speech signal[23]

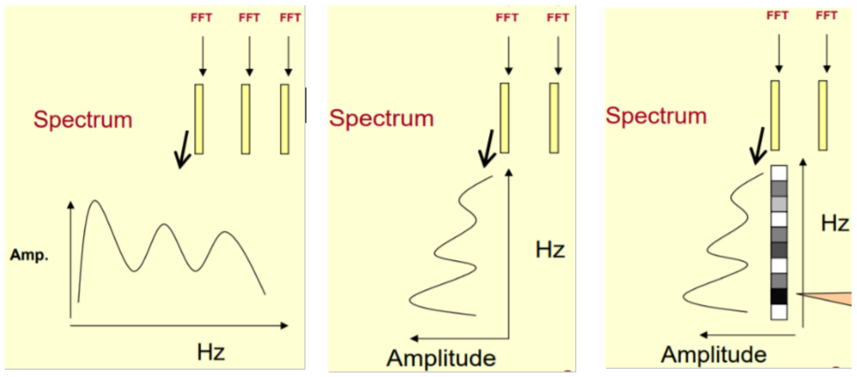


Figure 3.4: Rotating the spectrum of each frame by 90° [23]

The duration of spoken digits varies between 400 ms to 900 ms. As stated above, we've to divide it into a certain number of speech frames. We've chosen to use the following parameters for obtaining the spectrograms of spoken digits:
No. of speech frames = 40
Overlap between adjacent frames = 50
We take the logarithm of the spectrogram since the energy values in the spectrograms vary from as low as 1e-6 to as high as 1e4. The next step is to use a band of filters on the spectrogram due to two reasons, the first being that the lower frequencies contain more information and the second being that the human ear responds to frequencies in a non-linear fashion. (The difference between 9.9kHz and 10kHz is hardly perceived whereas the difference between 900 Hz and 1000 Hz is quite evident). The range of frequencies of our spectrograms is from 0 Hz to 4000 Hz. We divide it into five bands of frequencies ,i.e 333.3 Hz, 333.3 Hz, 666.6 Hz, 1000 Hz, 1666.6 Hz. The energy of each band is the average energy of all the frequencies contained in that band. We can increase the number of frequency bands for finer resolutions but it would lead to an increase in the dimension of the input vector thereby increasing the size of the input layer of the spiking neural network. The final spectrogram has five frequency bands and 40 speech frames as shown in figure 3.5. It can be flattened into a 200 dimensional vector (50x4=200). This is the input vector to the SNN.

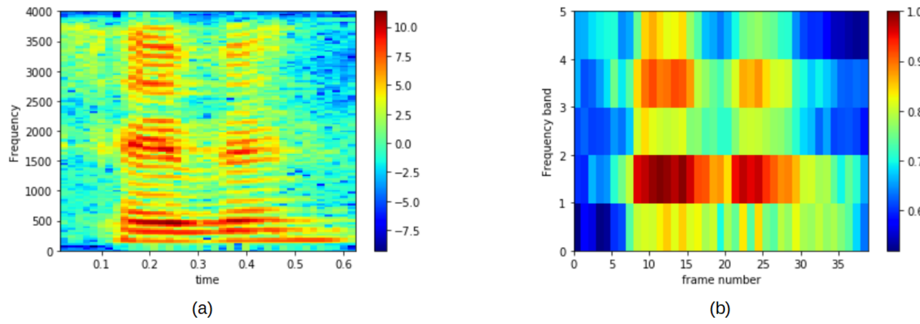


Figure 3.5: (a) Sample spectrogram of the digit 0 (b)Input feature matrix after applying the band of filters on the spectrogram

3.3 Spiking Neural Networks

Researchers have proposed many interesting approaches for performing speech recognition using SNNs. (Hu et al,2020)[24] have used a SNN cascaded by a CNN. While this achieves good accuracy, it doesn't operate entirely in the spiking domain. (Wu et al,2018)[25] make use of the self-organizing map followed by a simple SNN. This approach requires the usage of self-organizing maps which is a computation overhead. Since speech is temporal as opposed to say, images (spatial data), recurrent architectures are generally favored for this task. The main challenge in using spiking RNNs lies in training them effectively. The usage of local plasticity rules for training spiking RNNs still remains unclear. (Bellec et al,2020)[26] have recently proposed an alternative called the e-prop learning rule as an approximation to STDP. Others (Skowronski et al, 2007)[27], (Zhang et al, 2015)[28] have proposed reservoir computing based SNNs that overcome the convergence issues in RNNs. This approach requires a very large number of synapses and neurons and it is also computationally expensive.

Since our aim is to eventually implement the SNN at the circuit level, we have used used a circuit-friendly approach that tries to minimize the number of synapses and neurons without significantly affecting classification accuracy. We propose a small-footprint fully-connected feedforward SNN that is trained solely using the local STDP learning rule.

3.3.1 SNN Architecture

We have used a SNN with having an input layer consisting of two hundred LIF neurons and an output layer consisting of M LIF neurons. The neurons in the input layer are referred to as the pre-synaptic neurons and the neurons in the output layer are called the post-synaptic neurons. The number of neurons in the output layer depends on the approach that we take towards training the network. For the case of supervised learning, since we have ten distinct classes (the digits 0 to 9), we can set $M = 10$ and assign each output neuron to a particular class. In case of unsupervised learning, experiments indicate that the number of neurons in the output layer usually has to be greater than the total number of distinct classes. If we use just as many output neurons as the number of classes, it turns out that not every class would be represented by an output neuron. We need to ensure a sufficient number of output neurons so that each class gets is represented by least one output neuron. Each neuron in the input layer is connected to all neurons in the output layer using synapses thereby making it a fully-connected SNN. The properties of these synapses closely resemble that of the memristor.

3.3.2 Input to SNN

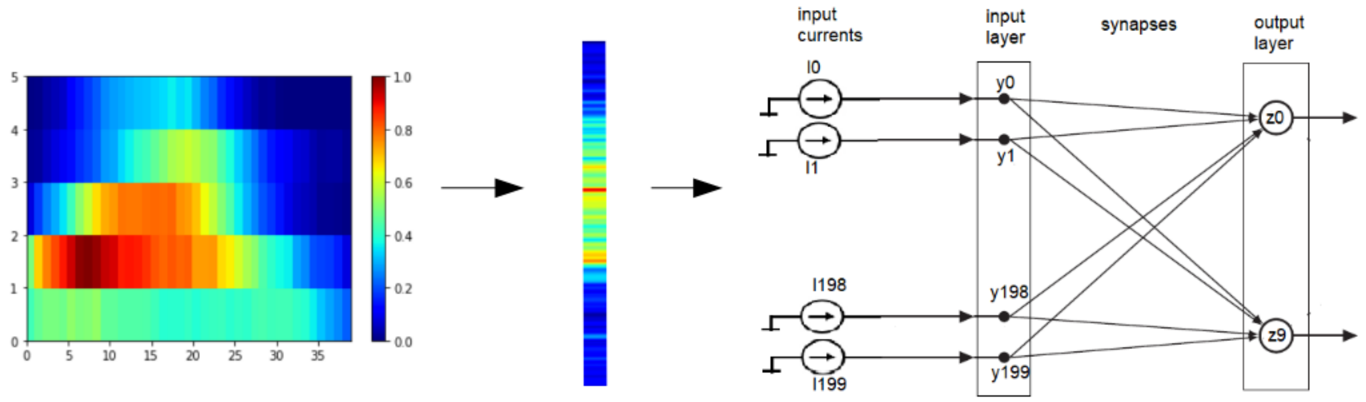


Figure 3.6: Input to the SNN

The input to the SNN is the spectrogram of the spoken digit. The spectrogram is flattened into a vector of size 200x1. The entries of this vector are real valued numbers that denote the energy of the signal. Since SNNs use spikes to transmit information, we need to convert the each entry of the input vector into appropriate spike trains. This role is played by the LIF neurons in the input layer. We inject a constant current into each input neuron for a fixed duration

(100 ms usually). The input neuron converts the input current into a train of voltage spikes. The spike-rate is determined the magnitude of the input current. As the magnitude increases, the spike-rate also increases. The magnitude of the injected current is directly proportional to corresponding entry in the input vector. If the energy denoted by a particular entry in the input vector is close to zero, it's corresponding input neuron will generate very few spikes. Whereas for a higher energy, the input neuron will generate a large number of spikes.

3.3.3 Parameters of the SNN

Note: We shall use the terms input neuron and pre-synaptic neuron interchangeably. Similarly for the terms output neuron and post-synaptic neuron. Firing rate and spiking rate are also used interchangeably.

There are many parameters of the SNN that need to be determined in order to ensure proper operation and efficient training of the SNN. There are two important factors that guide us towards the right parameters. Firstly, the very purpose of using an SNN is to accomplish learning in a biologically-plausible way. Thus, the parameters should be set such that they resemble their biological counterparts. Secondly, from the perspective of a circuit level implementation, the values should be realizable. This implies that the voltages, currents and RC time constants should not take on extreme values. We briefly discuss some of the important parameters below:

Time constant of pre-synaptic (input) neurons τ_{in} : τ_{in} determines the number of spikes fired by the pre-synaptic neuron for a given input current. Similar to an RC circuit, as τ_{in} increases, it takes longer for the membrane potential to reach the threshold voltage which in turn reduces the firing rate of the neuron(see figure 3.7. τ_{in} has to be set in tandem with the input current values. It is chosen to ensure that the neuron fires close to zero spikes when the input current corresponds to minimum energy and a reasonable number of spikes for the input current corresponding to maximum energy (refer 3.1). The number of spikes should not be too high either as that would unnecessarily burn more power. We set τ_{in} to be 200 ms.

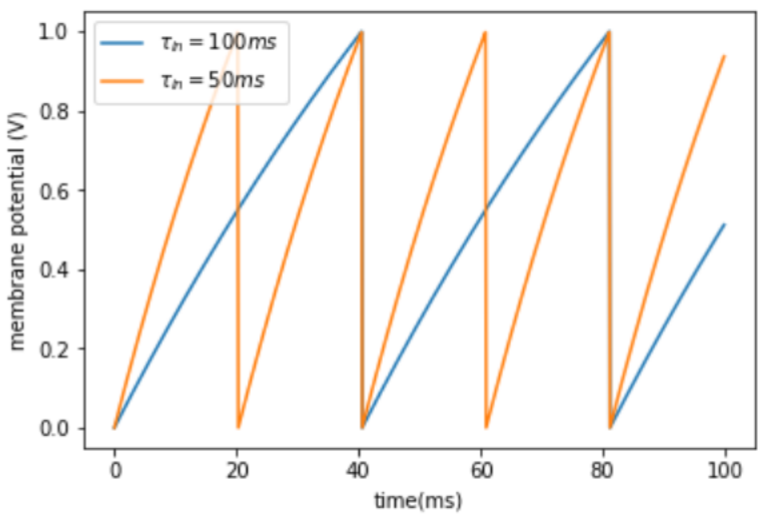


Figure 3.7: Effect of τ_{in} on the firing rate

Input Current	**Number of Spikes**
20	9
18	8
16	7
14	6
12	5
10	4
8	3
6	2
4	1
2	0
0	0

Table 3.1: Number of spikes fired by an input LIF neuron for various values of input currents

Threshold voltage of pre-synaptic and post-synaptic neurons V_{thresh}: The threshold voltage also plays a key role in determining the firing rate of a neuron. Decreasing V_{thresh} leads to an increase in the firing rate (figure 3.8). V_{thresh} is set to 1V for both the pre and post-synaptic neurons.

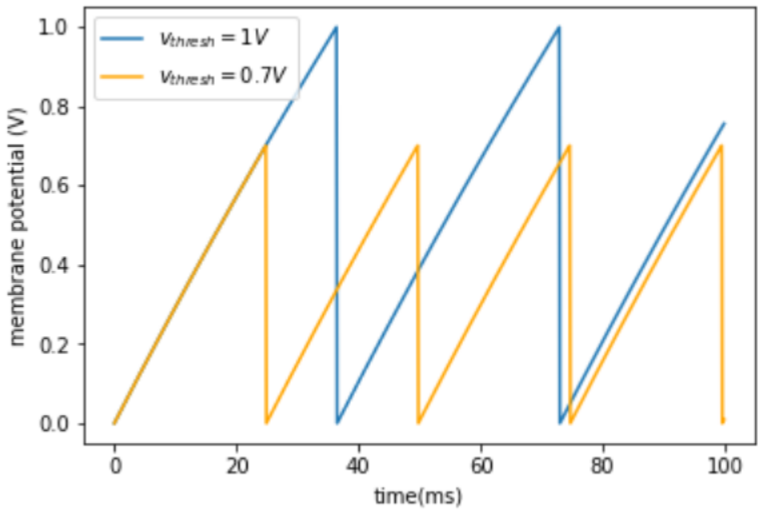


Figure 3.8: Effect of threshold voltage (both pre and post-synaptic neurons

Time constant of post-synaptic (output) neurons τ_{out}**:** Time constant of an output neuron serves a slightly different purpose than that of the input neuron. The membrane potential of the output neuron increases in a stepwise fashion as it receives current in the form of spikes via the synapses connected to it. Approximating the incoming current as a delta function $i(t) = I_{syn}\delta(t)$, the membrane potential just after the incoming spike would be :

$$v_{post}(t + dt) = v_{post}(t) + \frac{1}{C}\int I_{syn}\delta(t)dt$$

Between two incoming spikes, the membrane potential tends to discharge towards the reset voltage. τ_{out} mainly controls this discharge rate. Increasing τ_{out} reduces the discharge rate. If the membrane potential discharges quickly, it would take a larger number of incoming spikes for the post-synaptic neuron to reach the threshold voltage, thereby reducing the spiking rate of the output neuron (refer fig 3.9).

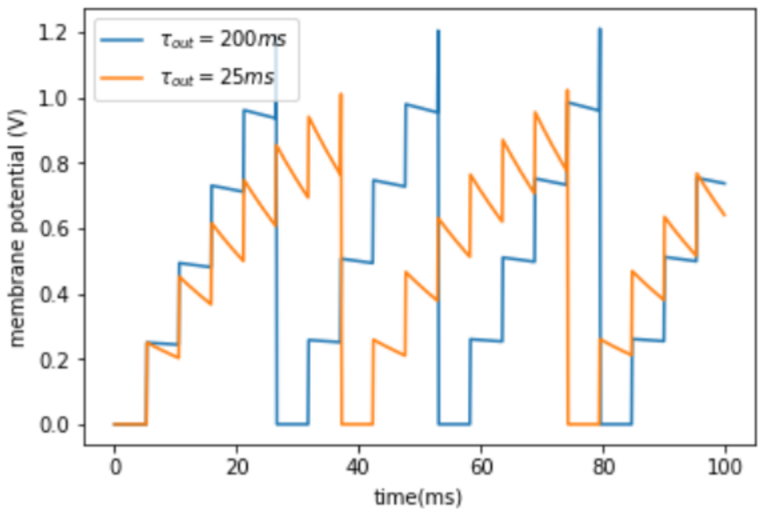
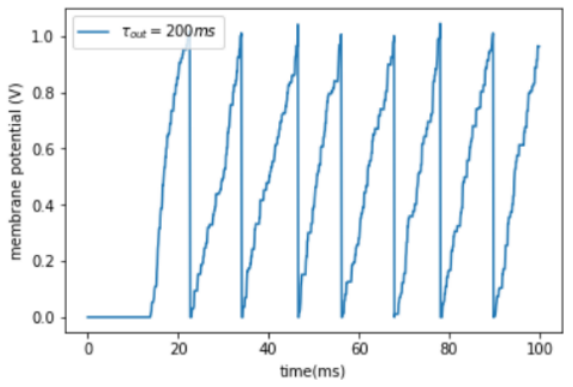


Figure 3.9: Effect of post-synaptic neuron time constant τ_{out}

In our SNN, each output neuron receives spikes from 200 input neurons. Thus, for one training sample, the number of incoming spikes received by an output neuron is much higher than the spiking rate of the input neurons. This causes the membrane potential of the output neuron to rise in an almost continuous fashion as there is a very small time gap between two incoming spikes, due to which the membrane potential doesn't get sufficient time to discharge. Figure 3.10 depicts the evolution of an output neuron's membrane potential for our SNN upon presenting a train/test sample. It can be observed that the membrane potential hardly discharges because there the incoming spikes from 200 input neurons.

Conductance scaling factor (g) : This is a critical parameters that allows us to control the spiking rate of the post-synaptic neurons. As mentioned previously, the number of incoming spikes received by a post-synaptic neuron is much higher than the spiking rate of the input neurons. Since each incoming spike increases the membrane potential of the post-synaptic neuron as $v_{post}(t + dt) = v_{post}(t) + w$ (where w is the synaptic weight), it causes the



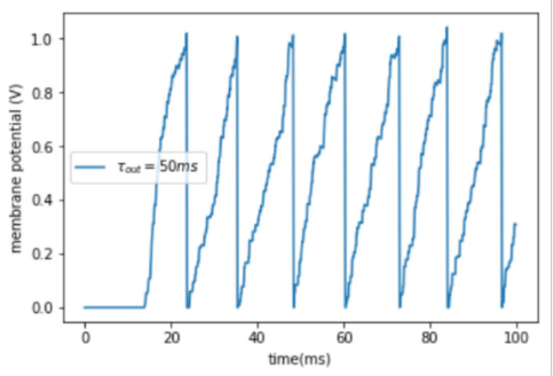


Figure 3.10: Evolution of the membrane potential of an output neuron upon presenting one training sample.

post-synaptic neuron to reach the threshold voltage very quickly in comparison to the pre-synaptic neurons. An undesired outcome of this being that the firing rate of post-synaptic neurons is much higher than pre-synaptic neurons. In biological systems, the complex neuronal dynamics ensures that the pre-synaptic and post-synaptic neurons fire at similar rates. So, we would also like to enforce similar spiking rates for pre and post-synaptic neurons. This is done by introducing the conductance scaling factor (g). The modified membrane potential equation would be $v_{post}(t + dt) = v_{post}(t) + g.w$. Setting g in the range of 1e-2 to 1e-3 allows us to ensure similar spiking rates. A more biologically-plausible approach would be to use adaptive threshold neurons but their circuit implementation is much complex in comparison to LIF neurons.

Synaptic pre-exponential constants (A_{pre} and A_{post}): These parameters determine the maximum amount by which a synaptic weight can increase on decrease. A_{pre} determines to the increase in the synaptic weight when the pre-synaptic neuron fires just before the post-synaptic neuron. A_{post} determines to the decrease in the synaptic weight when the post-synaptic neuron fires just before the pre-synaptic neuron The values of A_{pre} and A_{post} are analogous to the learning rate in ANNs. They should neither be too high nor too low. If it is too low, the training takes too many epochs whereas if it's too high, it may cause convergence issues. We set A_{pre} in the range of 1e-2 and A_{post} in the range of $-1e^{-2}$.

Synaptic time constants (τ_{pre} and τ_{post}): As the time delay between the pre and post synaptic spike increases, the change in the synaptic weight decreases exponentially. This is the defining characteristic of STDP. τ_{pre} and τ_{post} control this exponential nature of synaptic weight change.

The following equation and its graph in figure 3.11 depict the usage of the synapse parameters:

$$\Delta\text{w}=\begin{cases} wA_{pre}e^{-\frac{\Delta t}{\tau_{pre}}} & \Delta t > 0 \\ wA_{post}e^{-\frac{\Delta t}{\tau_{post}}} & \Delta t < 0 \end{cases}$$

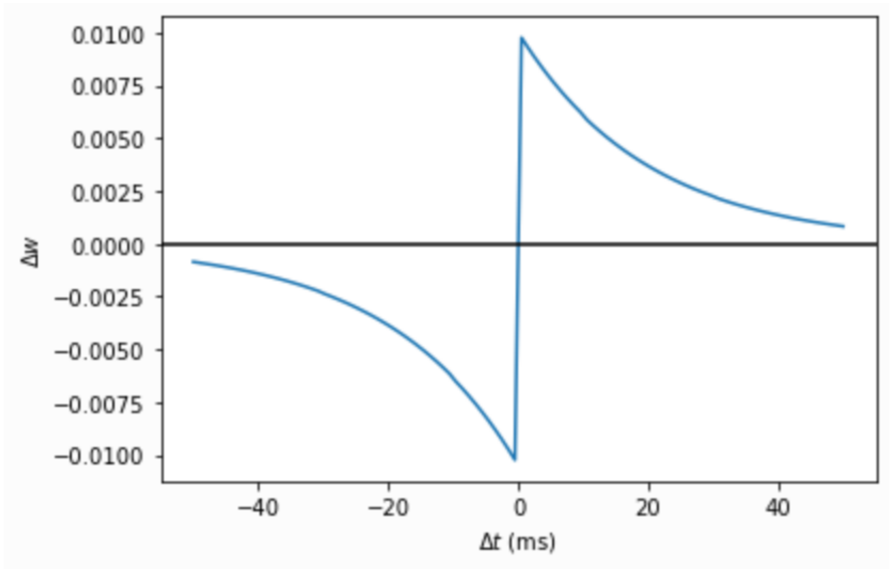


Figure 3.11: STDP window of the synapse model. The change in synaptic weight as a function of time is depicted in this graph

3.3.4 SNN training approaches

There are two distinct approaches to train a SNN using STDP. They are unsupervised learning and supervised learning. In supervised learning, the input to the network consists of both the input sample (i.e the spectrogram in our case) and its corresponding label (indicates the spoken digit). In unsupervised learning, just the input samples are used to train the network.

From a biological viewpoint, the learning process is very complex and is still not clearly understood by researchers. However, experts in neuroscience have suggested that unsupervised learning is much more likely to be employed by the brain as humans generally do not use explicit labels to know their surroundings. But even the most sophisticated unsupervised algorithms known to us cannot fully explain learning in nature. The reader can refer to (Zador et al, 2019)[29] for a detailed discussion of learning in nature vs learning in neural networks.

We have explored both supervised and unsupervised approaches to train the SNN for spoken digit recognition. We shall now discuss their implementation and results.

Unsupervised Learning

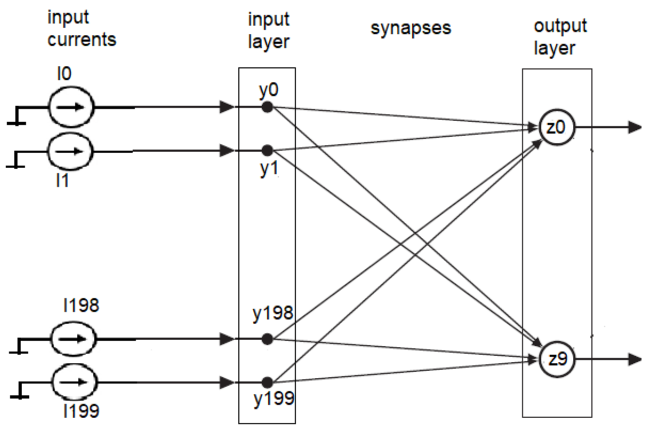


Figure 3.12: SNN architecture for unsupervised learning

There are three stages in unsupervised learning: the first being the train stage followed by the assign stage and finally the test stage. Before training, the synaptic weights of the SNN are initialized to values between 0 and 1 sampled from the uniform distribution.

Train stage: During the training stage, for each training sample, the input to the SNN are constant current sources with the value of each current source being proportional to the energies in the sample's spectrogram. These current sources are injected into the pre-synaptic neurons (input layer) for a fixed duration of 100 ms. Depending on the magnitude of the injected currents, the pre-synaptic neurons convert the injected currents into spike trains. These spike trains are transmitted to the post-synaptic neurons via the synapses. The membrane potential of the post-synaptic neurons builds up due to the incoming spikes and once the threshold voltage is reached, it emits a voltage spike. Depending on the relative timing of the pre and post-synaptic spikes, the synaptic weights undergo changes according to the STDP learning rule as discussed earlier. All the training samples are used to train the SNN in this manner. Multiple rounds of training (called as epochs) are performed. As the training progresses, we would expect the synaptic weights to undergo learning so that they eventually learn the desired patterns. Note that the class labels are not used during the training stage. The synaptic weights are allowed to evolve without any external signal in an unsupervised manner.

Assign stage: Once the training stage is completed, the SNN has presumably learnt some meaningful patterns. We now have to assign each output neuron to a class. This is the only stage where the labels of training samples are used. No learning takes place in this stage. The synaptic weights that have been learnt in the train stage are used here. We present the training samples one by one to the SNN and keep track of the spiking activity of the output neurons. For an output neuron, we count the total number of spikes that it fires for each class. The output neuron is assigned to that class for which it spikes the highest number of times. For eg., let's assume that there are only 3 distinct classes. If a particular output neuron fires a total of 15 spikes for all input samples belonging to class 1, 34 spikes for class 2 and 8 spikes for class 3, this output neuron is assigned to class 2. Similarly, all output neurons are assigned to some class. This obviously means that we need at least as many output neurons as the total number of classes. So, we have at least 10 neurons in the output layer. It might happen that multiple output neurons are assigned to the same class whereas some classes are not represented by any output neuron. As we shall see later, the number of output neurons have to be increased in such cases so that each class gets represented by at least one output neuron.

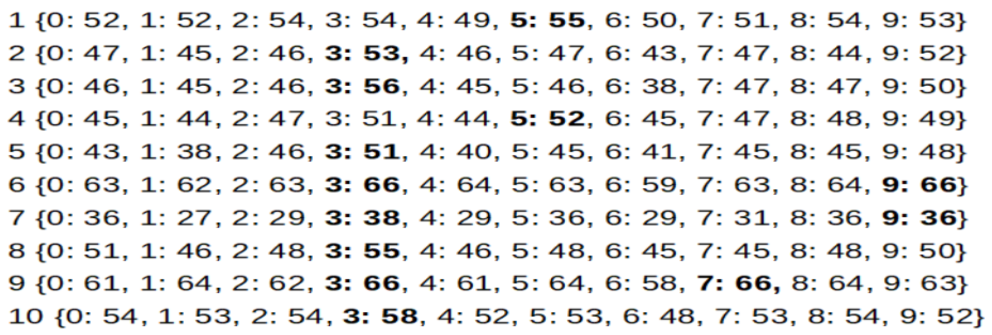
Test Stage: Having assigned output neurons to classes, the SNN can now be used for classification. The unknown sample is presented to the SNN for a duration of 100 ms and the number of spikes at each output neuron are recorded. The unknown sample is assigned to that class for which it generates the highest number of spikes. For eg., as shown in the figure below...(complete it)

Results and discussion of unsupervised learning

During the assign stage, we discover that certain classes (the digit 3 and 5) produce a higher number of spikes at the output neurons. Due to this, most of the output neurons get assigned to these classes which and other classes (the remaining digits) aren't assigned to any output neuron. So, during the test stage, when an unknown sample is presented to the network, it would classify it either as 3 or 5 most of the times. Moreover, during the assign stage, it is also observed that each output neuron has more or less similar number of spikes for each class. This implies that the SNN is unable to confidently assign an output neuron to a class. Figure 3.13 shows the number of spikes per class at each output neuron in the assign stage. The format to read it is :

Output_neuron_number {Class_0:# spikes, Class_1:# spikes, ... }

A possible solution would be to increase the number of output neurons so that



```
1 {0: 52, 1: 52, 2: 54, 3: 54, 4: 49, 5: 55, 6: 50, 7: 51, 8: 54, 9: 53}
2 {0: 47, 1: 45, 2: 46, 3: 53, 4: 46, 5: 47, 6: 43, 7: 47, 8: 44, 9: 52}
3 {0: 46, 1: 45, 2: 46, 3: 56, 4: 45, 5: 46, 6: 38, 7: 47, 8: 47, 9: 50}
4 {0: 45, 1: 44, 2: 47, 3: 51, 4: 44, 5: 52, 6: 45, 7: 47, 8: 48, 9: 49}
5 {0: 43, 1: 38, 2: 46, 3: 51, 4: 40, 5: 45, 6: 41, 7: 45, 8: 45, 9: 48}
6 {0: 63, 1: 62, 2: 63, 3: 66, 4: 64, 5: 63, 6: 59, 7: 63, 8: 64, 9: 66}
7 {0: 36, 1: 27, 2: 29, 3: 38, 4: 29, 5: 36, 6: 29, 7: 31, 8: 36, 9: 36}
8 {0: 51, 1: 46, 2: 48, 3: 55, 4: 46, 5: 48, 6: 45, 7: 45, 8: 48, 9: 50}
9 {0: 61, 1: 64, 2: 62, 3: 66, 4: 61, 5: 64, 6: 58, 7: 66, 8: 64, 9: 63}
10 {0: 54, 1: 53, 2: 54, 3: 58, 4: 52, 5: 53, 6: 48, 7: 53, 8: 54, 9: 52}
```

Figure 3.13: Number of spikes per class at each output neuron.

all classes can be represented by the output neurons. We used as many as 50 output neurons in our experiments but faced similar issues during the assign stage. The results indicate that the SNN responds to not only the input pattern but also the total energy contained in the input spectrogram. Certain digits are spoken for a longer duration than other digits due to which the total energy in the signal is higher. These digits tends to produce a higher number of spikes at each output neuron. To fix this issue, we calculate the average energy for each class and normalize the spectrograms. Still, the SNN is unable to confidently assign output neurons to classes. Due to the lack of a supervision signal, it might be challenging to learn the desired patterns in a fully unsupervised manner.

We also take a look at the evolution of the distribution of synaptic weights as training progresses. A lot of the synaptic weights tend to either decay towards the minimum value 0 or reach the maximum value 1. This behaviour can be explained as a consequence of the hebbian (STDP) learning rule (fully unsupervised learning). Researchers in neuroscience[30] suggest that hebbian learning on its own is not sufficient to learn complex patterns. There needs to be some form of assistance for the synaptic weights to learn properly.

We use a simple example to explain the possible reason behind the dismal performance of fully unsupervised learning. Consider a SNN (as shown in figure 3.14 with 3 input neurons and 2 output neurons. Without loss of generality, assume that all the synaptic weights are initialized to the same value. We need to consider groups of weights in order to clearly understand what's going on over here. All the weights connecting all the input neurons to the first output neuron belong to the red group (the weights r1, r2 and r3). Similarly, all the weights connecting all the input neurons to the second output neuron belong to the blue group (the weights b1, b2 and b3).

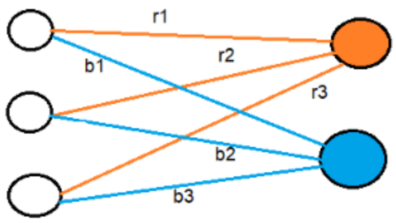


Figure 3.14: An SNN depicting two groups of weights

As the network is trained using input samples, each group of weights sees the same input and they would evolve in the exact same manner. No matter how many epochs we train the network for, the weight r1 and b1, r2 and b2, r3 and b3 change in the same manner. Due to this, during the assign stage, both the output neurons would show the same behaviour in terms of the number of spikes for each class. Both the neurons will be assigned to the same class and thus the SNN is unable to classify input samples properly. Clearly, the network needs some form of assistance, either direct or indirect, to ensure that the output neurons and the corresponding synaptic weight learn distinct patterns to ensure proper classification.

Supervised Learning

Till now, we have been using the hebbian STDP learning rule. For supervised learning, we have to use the anti-hebbian learning rule along with the hebbian learning rule. Anti-hebbian learning is the opposite of the hebbian learning rule. From a biological perspective, (Choe, 2013)[31] defines it as "...a form of synaptic plasticity where correlated activation in the pre- and postsynaptic neurons leads to the reduction in the efficiency of the presynaptic neuron's ability to elicit activation of the postsynaptic neuron". As far as training an SNN is concerned, the anti-hebbian learning rule is as follows : if the post-synaptic neuron fires just after the pre-synaptic neuron, the synaptic weight decreases whereas if the post-synaptic neuron fires just before the pre-synaptic neuron, the synaptic weight increases.

The architecture of the SNN is modified by introducing a "teacher" that is used for training the network in a supervised manner. The role of the teacher is to use the class labels and determine which output neurons should undergo hebbian STDP and which other output neurons should undergo anti-hebbian STDP. Since we are performing supervised learning, we require only one output neuron for each class (so total 10 output neurons). The output neurons are denoted as z_i, where $0 \leq i \leq 9$. Output neuron z_i is assigned to the i^{th} class. Supervised learning consists of two stages, i.e the train and the test stage.

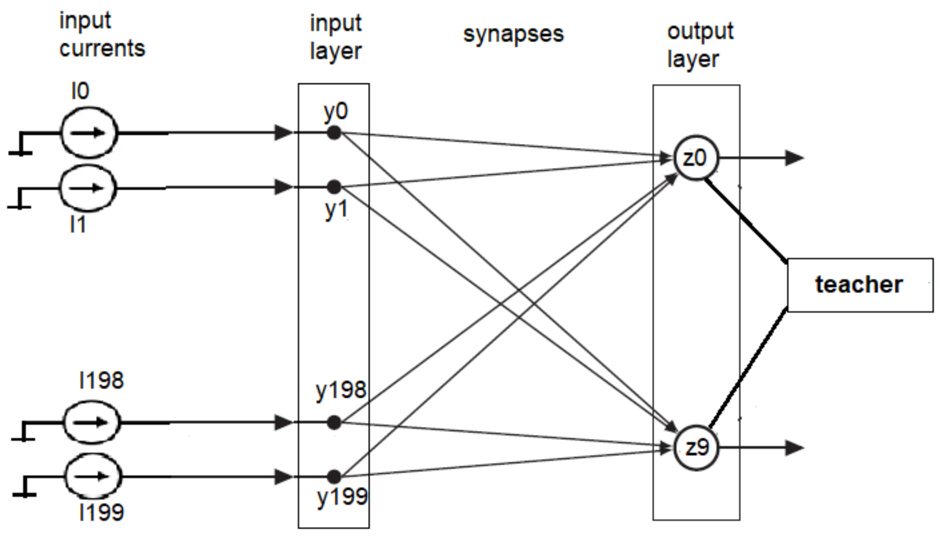


Figure 3.15: Supervised SNN architecture

Train Stage: The main difference between the unsupervised and supervised train stage is the usage of the class labels by the teacher to decide the type of STDP that each output neuron undergoes. If the input sample presented to the network belongs to the k^{th} class, the teacher signal ensures that the output neuron z_k and its synapses connecting z_k to all the input neurons undergo hebbian-STDP whereas all the remaining output neurons (other than z_k) undergo anti-hebbian STDP. The teacher plays a key role in the training procedure by assisting the synaptic weights to learn the desired patterns.

Test Stage: We experiment with two different methods in order to classify an unknown input sample during the test stage. The input sample is presented to the SNN for a duration of 100 ms. The first method is the time-based method. Whichever output neuron fires earliest, the input sample is assigned to that class. The second method is the rate-based method. We count the number of spikes at each output neuron. Whichever output neuron fires the highest number of spikes, the input sample is assigned to that class.

It turns out that the time-based method is able to classify the unknown test samples more confidently than the rate-based method. The rate-based method is a bit ambiguous because some of the other output neurons also fire an equal number of spikes as the correct output neuron (the class to which the test sample actually belongs). This is a consequence of the SNN responding to the energy of the sample's spectrogram.

The synaptic weights connecting the input neurons to each of the output neurons learn some meaningful patterns as the training progresses. The synaptic weights are grouped into ten groups. Each group consists of those weights connecting all the input neurons to an output neuron. As seen in figure 3.16, the weights in red connect all the input neurons to the output neuron z_0. This is group 0. Similarly the weights colored in blue are group 1 and so on. Since

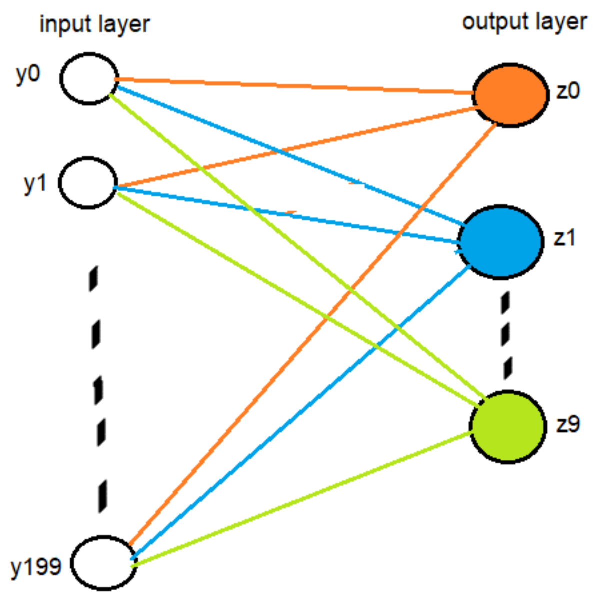


Figure 3.16: Depicting groups of synaptic weights of the SNN

there are two hundred input neurons, there are two hundred weights in each group. We take a vector of these weights and reshape it into a 50x4 matrix. Recall that the input spectrograms were also of the same dimensions. We can now visualize these weights as a 2D matrix and compare the emerging pattern with the input spectrograms of each digit. Before training, the weights do not depict any particular pattern. As the network is trained, we see that the a pattern begins to emerge. Upon completion of training, the visualized weights resemble the spectrograms of their corresponding digits. Figure 3.17 shows the evolution of the weights of group 0. It is clearly visible that the weights learn the desired pattern as the training progresses. The final values of the weights quite closely resemble the average spectrogram of the training samples of the spoken digit 0 as seen in figure 3.18. All other groups of weights are also visualized and compared with the corresponding digit's average spectrogram.

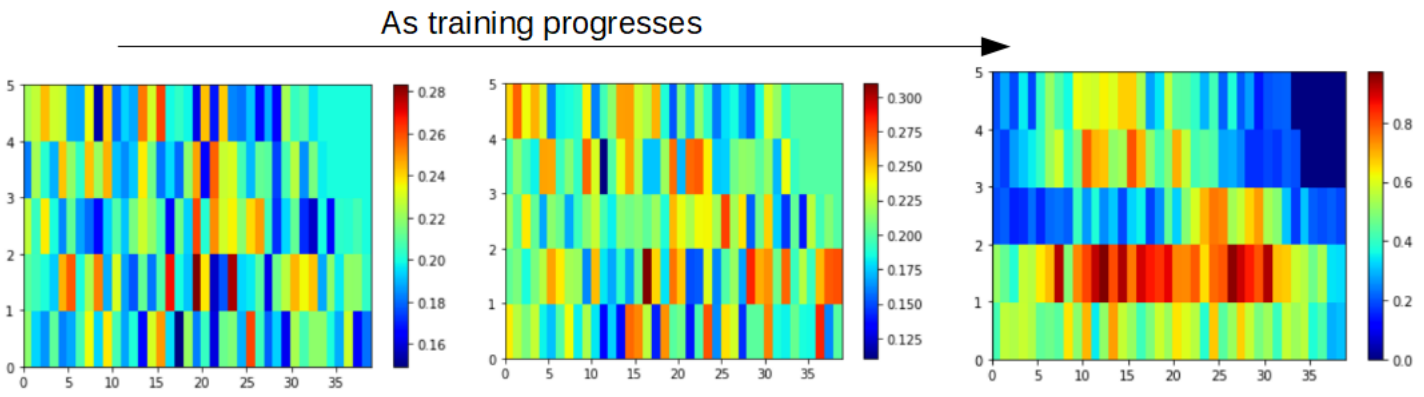


Figure 3.17: Evolution of weights of group 0 as training progresses

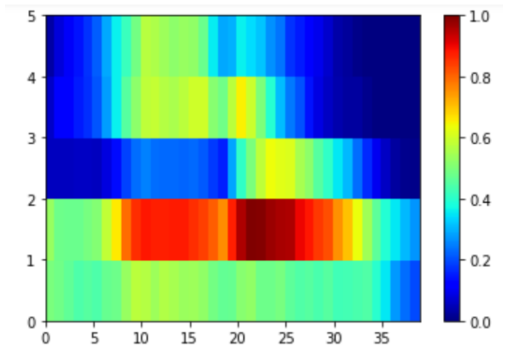


Figure 3.18: Average spectrogram of the digit 0

The learnt weights corresponding to the groups for each digit have been visualized as shown in the figures below. We compare the visualized weights with the average spectrogram of the the corresponding digit. There are two figures for each digit. The figure at the top is a visualization of the learnt weights and the figure at the bottom is the average spectrogram. It can be seen that the learnt patterns are similar to the spectrograms. Thus, the SNN has learnt meaningful and interpretable patterns.

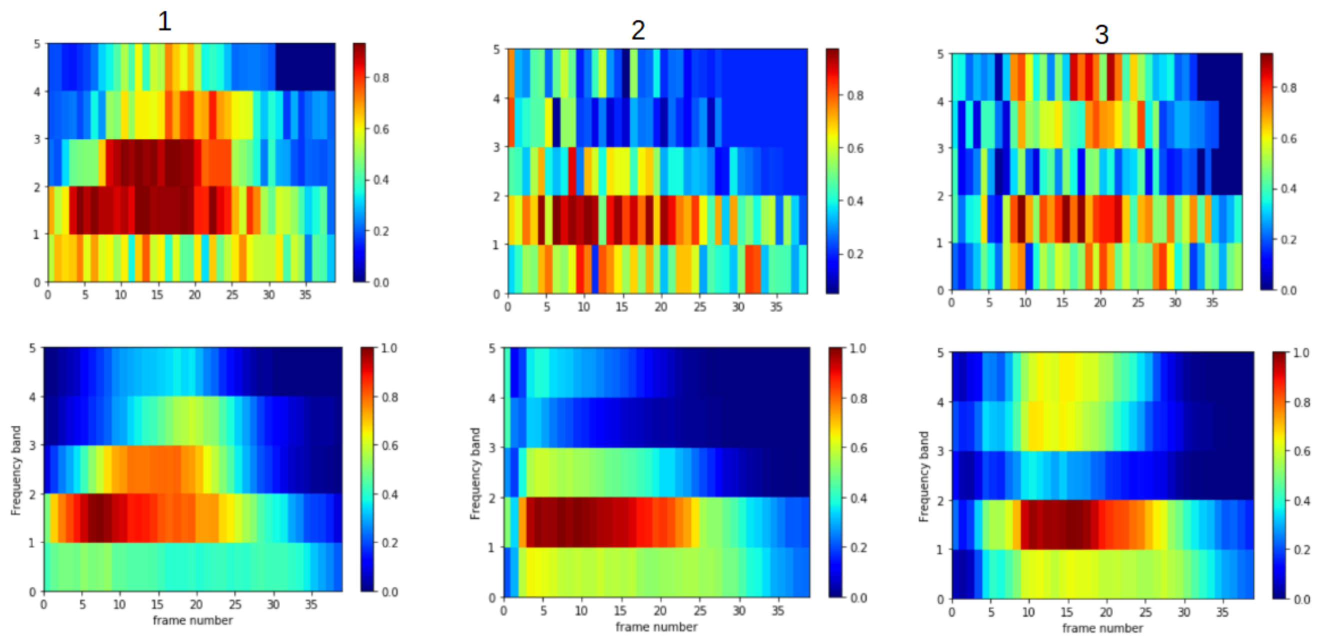


Figure 3.19: Visualizing the learnt weights for the digits one, two and three. The respective spectrograms are shown for comparison

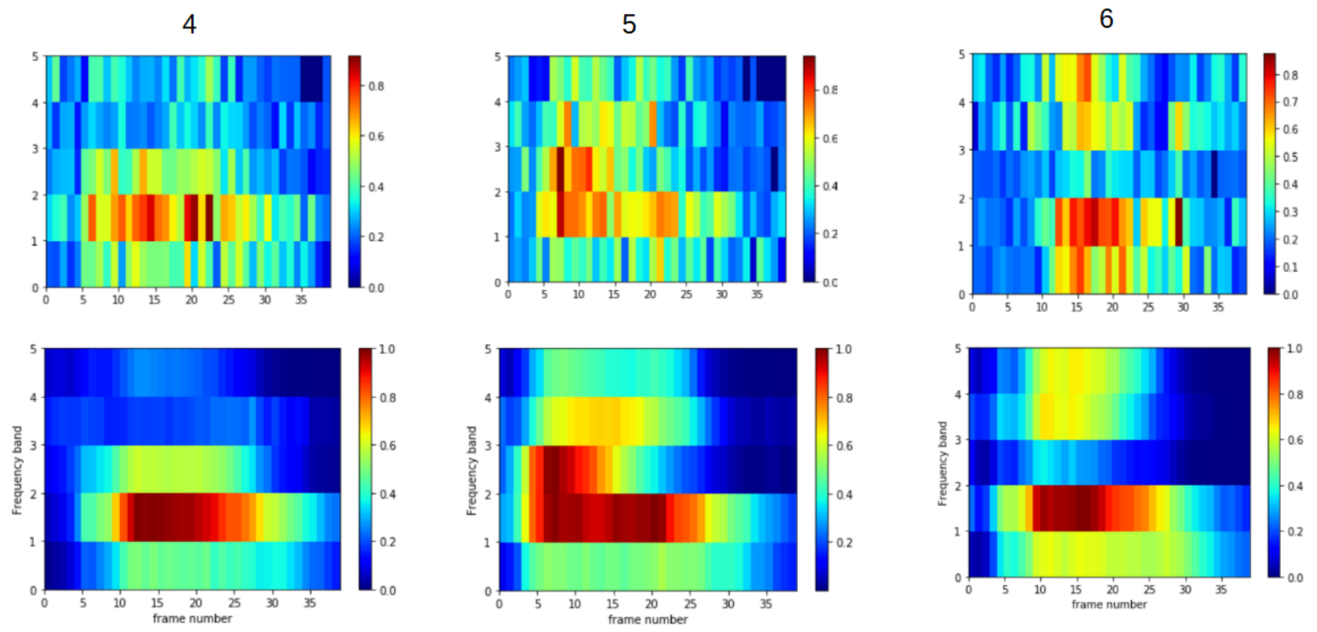


Figure 3.20: Visualizing the learnt weights for the digits four, five and six

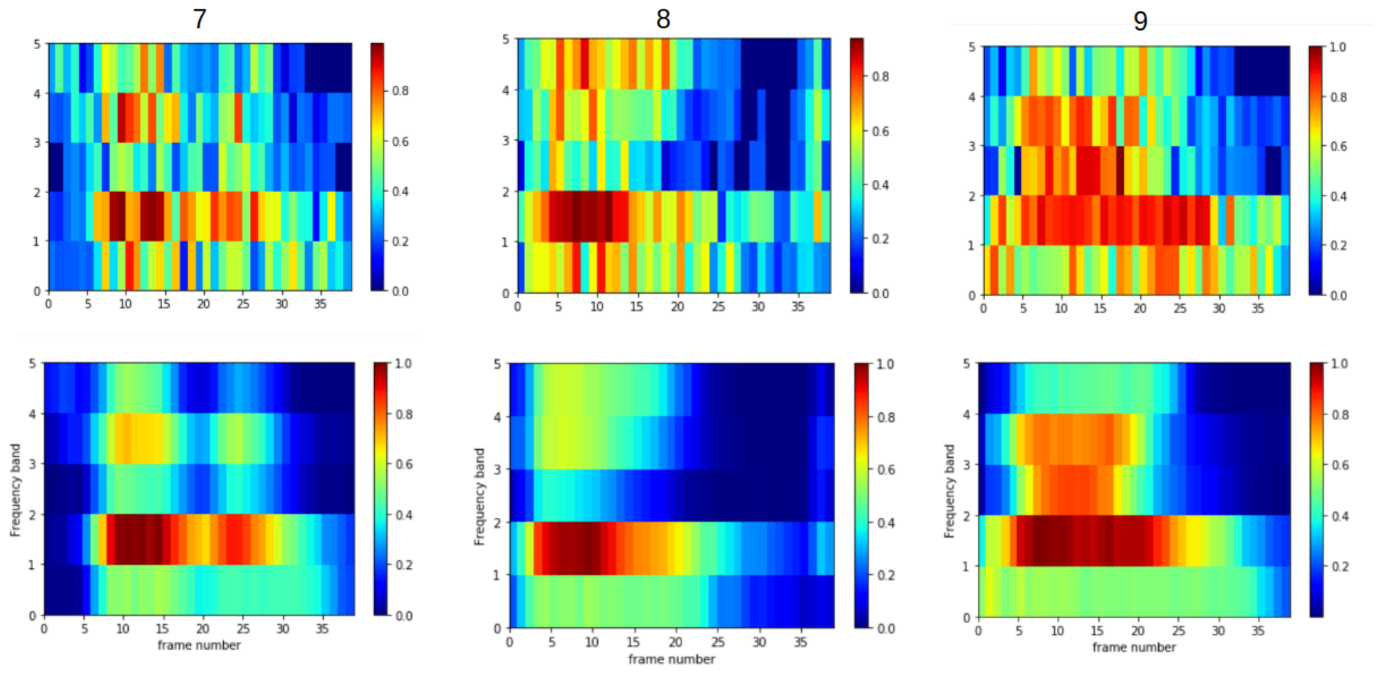


Figure 3.21: Visualizing the learnt weights for the digits seven, eight and nine

Memristor crossbar array for ANN

In this chapter, we use the unified RRAM compact model developed by our group to implement an Artificial Neural Network in SPICE for performing image classification. In an ANN, the vector-matrix multiplications are the most computationally expensive operation. The idea behind using a memristor crossbar array is to perform the vector-matrix multiplications directly in-memory. This would be much fast and efficient compared to the CMOS-based digital computers being used today. There are various aspects that need to be considered in order to implement a crossbar array of RRAM devices.

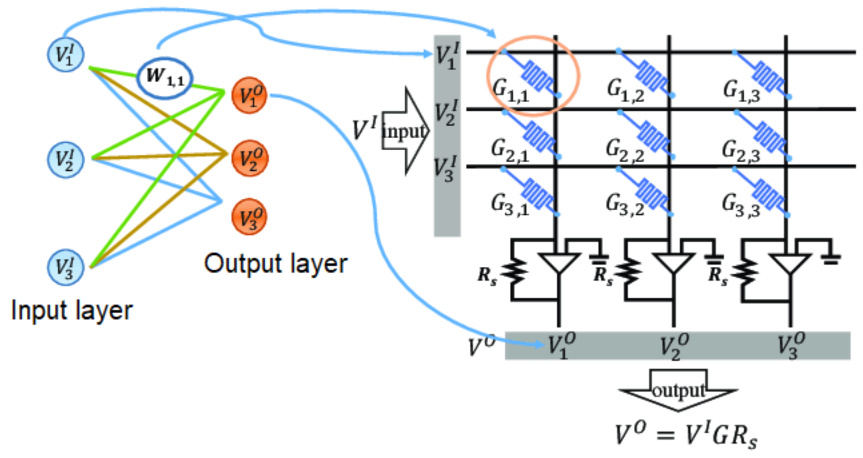


Figure 4.1: An ANN and its implementation using a memristor crossbar array[32]

The one-to-one correspondence between an abstract ANN and its corresponding implementation using a memristor crossbar array is show figure 4.1. The input to the ANN is a real-valued vector of dimension D. The equivalent input to the crossbar array would be D distinct voltage sources proportional to the input vector. The weight $w_{k,j}$ connects the k^{th} input neuron to the j^{th} output neuron. These are called the synapses of the ANN. These weights are implemented using memristors in the crossbar array. The conductance of the memristors is

proportional to the corresponding weights of the ANN.
The output at the j^{th} output neuron is a result of multiplying the weights with the input followed by a non-linear activation function.The following equations explain the operation of the ANN :

$$z_j = \sum_{k=0}^{D-1} w_{kj}.v_k^I$$

$$v_j^O = tanh(z_j)$$

where v_k^I is the k^{th} entry of the input vector and v_j^O is the output at the j^{th} output neuron.
The same operation can be expressed in terms of the voltages, currents and conductances of the memristor crossbar array.

$$I_j^O = \sum_{k=0}^{D-1} g_{kj}.V_k^I$$

$$V_j^O = tanh(I_j^O)$$

where V_k^I is the k^{th} input voltage, g_{kj} is the conductance of the memristor connecting the k^{th} input neuron to the j^{th} output neuron, I_j is the resultant output current at the j^{th} index and V_j^O is the corresponding output voltage.

4.1 Dataset

We've created a dataset consisting of three alphabets (n, v and z) represented as an image of 3 pixels by 3 pixels. These are greyscale images with each pixel either taking the value 0 or 1. In order to expand the dataset, noisy versions of the images are obtained by flipping one pixel at a time of the original image. In this manner, we get a total of 30 images in our dataset. These are shown in figure 4.2. The size of the image is restricted to 3x3 pixels because as the number of pixels increase, the size of the crossbar array increases rapidly.

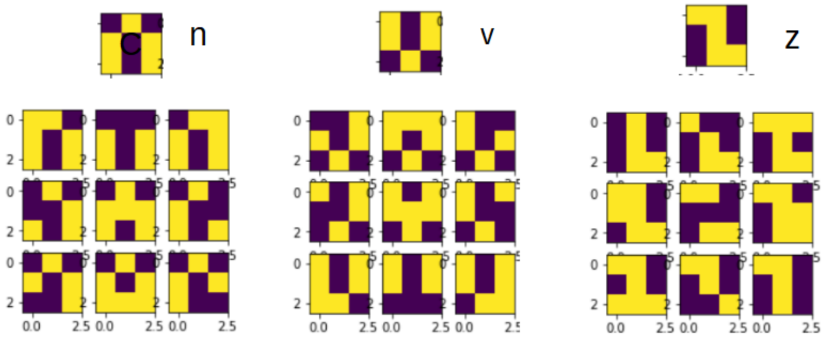


Figure 4.2: Dataset consisting of 30 images

4.2 ANN Architecture

We use an ANN with having an input layer of nine neurons and an output layer of three neurons since there are 3 distinct classes. There are no hidden layers in our ANN. It is a fully-connected feedforward network with 27 synapses connecting the input layer to the output layer. The non-linear activation at the output neurons can either be the sigmoid, the tanh or the ReLu activation. The output neuron y_1 is assigned to the alphabet ‘n’, y_2 is assigned to the alphabet ‘v’ and y_3 is assigned to the alphabet ‘z’.

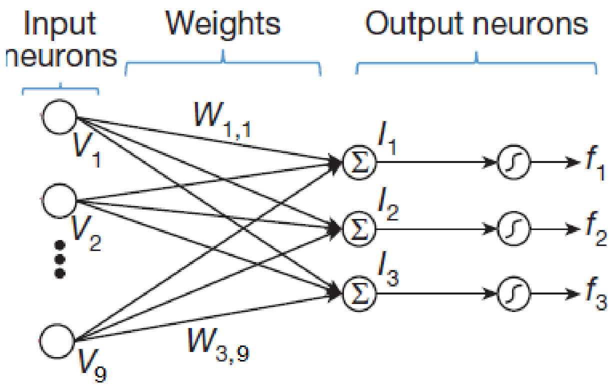


Figure 4.3: ANN architecture

4.2.1 Training the ANN

The ANN can be trained in a supervised manner using the classic backpropagation algorithm but a modified version of it known as the Manhattan update rule[33] is more realistic from a hardware perspective. We implement this learning algorithm in python. The training procedure is as follows: The weights are initialized to values sampled from a gaussian distribution N(0,1). Each image from the training set is flattened into a vector and is given as the input to the network. After the feedforward operation, the output values are obtained at each output neuron. The error $\delta_j(n)$ incurred at the j^{th} output neuron for the n_{th} training sample is calculated as :

$$\delta_j(n) = [v_{target} - v_j^O].\frac{df}{dz}|_{z=z_j(n)}$$

where v_{target} is the target value of the training sample. For eg., if the input sample is an image of the alphabet 'v', the target values at the output neuron y_1 and y_3 would be -1, whereas the target value at the output neuron y_2 would be +1. v_j^O is the output of the network. $\frac{df}{dz}$ is the derivative of the activation function with respect to z_j.

After presenting all the training samples to the network and calculating their δ_j values, the weights are updated using the Manhattan update rule as follows:

$$\varDelta w_{kj} = \eta.sgn(\sum_{n=1}^{N} \delta_j(n) v_j(n))$$

where η controls the learning rate and sgn is the sign function

$$\text{sgn(x)} = \begin{cases} +1 & x > 0 \\ -1 & x \leq 0 \end{cases} \tag{4.1}$$

The network is trained for multiple epochs to achieve good training accuracy.

We are able to achieve an accuracy of approximately 80%.

4.3 Weight Mapping

The desired weights have been obtained after offline-training of the ANN as discussed above. The next step involves mapping the learnt weights onto the RRAM crossbar array. But before doing that, one needs to decide the region of operation of the RRAM. As can be seen from the I-V curve of the device, there are two distinct resistance states namely the low resistance state (LRS) and the high resistance state (HRS). Since we want to use the conductance of the RRAM as a synaptic weight, we'd like the device have as many conductance states as possible between the LRS and the HRS. Observing the set-reset characteristic of typical RRAM devices, it is evident that the set process is abrupt whereas the reset process is gradual. So, it is usually preferred to operate the device in the reset region.

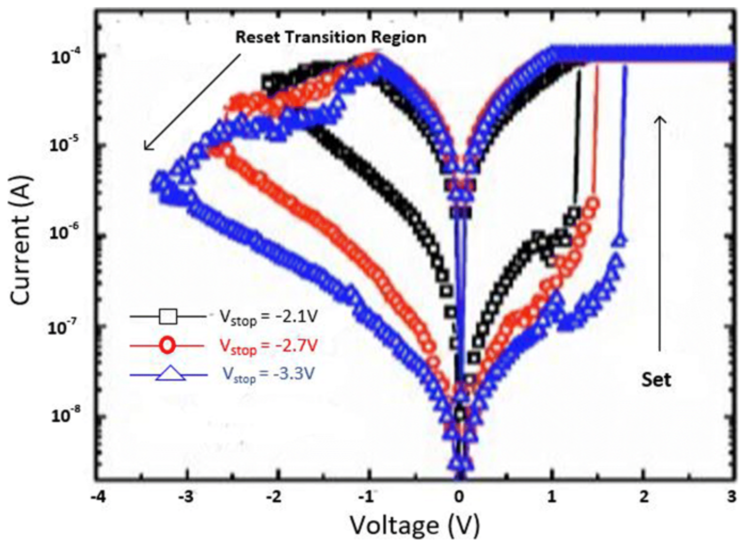


Figure 4.4: I-V curve of a RRAM device depicting the abrupt nature of the SET process and the gradual nature of the reset process[34]

4.3.1 Characterizing the RRAM device

The synaptic weights have to be mapped to discrete conductance states of the RRAM device. In order to find the possible conductance states, we have to characterize the device. Due to the lack of experimental data for our RRAM model, we make some reasonable assumptions to arrive at the possible conductance states. There are two state variables that determine the conductance of the device. They are i) the gap between the conductive filament and the bottom electrode and ii) the radius of the conductive filament. The maximum and the minimum gap of the filament is $gap_{max} = 2$ nm and $gap_{min} = 0.1$ nm. When the gap is at its maximum value, the conductance is exponentially dependent on the gap. So, we decrease the gap from its maximum value in steps of 0.05 nm and measure the conductance at each step. As the gap decreases, the dependence of the conductance on the gap is no longer of exponential nature. Beyond a certain point (gap $<$ 1nm), varying the gap causes very little change in the conductance. Figure 4.5 shows that conductance as a function of the gap. The minimum conductance of the device is around 35 pS and the maximum is around 10 nS.

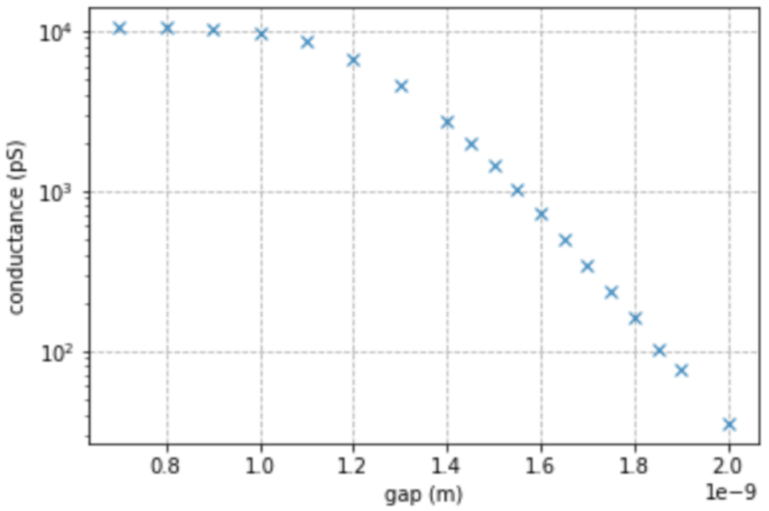


Figure 4.5: Conductance of the RRAM device as the gap is varied

The synaptic weights are mapped to the device's conductance values in a proportional manner. The minimum weight w_{min} (magnitude) is scaled by a factor $\frac{g_{min}}{w_{min}}$ to map it to the minimum conductance g_{min}. We shall refer to the quantity $\frac{g_{min}}{w_{min}}$ as the weight mapping factor. Now, the maximum weight is scaled by this weight mapping factor to obtain the corresponding conductance. If the obtained conductance is close to the maximum conductance of the device, the weight mapping factor is appropriate. If not, there are two possible scenarios :

1. If the obtained conductance is much lesser than the maximum conductance g_{max}, the weight mapping factor is recalculated by mapping w_{min} to the 2^{nd} smallest value of conductance. The new weight mapping factor is used to map w_{max} and its corresponding conductance is calculated.

2. If the obtained conductance is much greater than g_{max}, we have to drop w_{max} as it is exceeding the range of conductance. We then map the 2^{nd} highest weight to obtain its corresponding conductance value.

The above procedure is repeated a few times till a satisfactory mapping is arrived at. Now, all the other weights are scaled by the weight mapping factor to obtain the corresponding conductances. The conductance values thus obtained are mapped to the nearest discrete conductance state of the device.

4.3.2 Implementing Negative Weights

The conductance of a RRAM device cannot be less than zero. Negative weights are a necessity in ANNs. So, it is important to obtain an equivalent of negative conductance values. In order to emulate the effect of negative conductances, the sign of the input voltage signal is inverted so that the corresponding device current flows in the opposite direction. This would have an unintended effect of reversing all the other currents in the given row. A possible solution would be using two input rows for each input neuron. The actual input voltage is applied at one row and it's inverted version is applied at the other. This approach allows the mapping of both positive and negative weights to appropriate conductance values.

4.4 Crossbar Array Implementation

The 9x3 crossbar array is implemented in the Cadence Virtuoso circuit simulator. The circuit schematic is shown in figure 4.6. Prior to the simulation, the conductance of each RRAM device is set by fixing its gap and radius as determined by weight mapping. There are nine voltage inputs (and the corresponding inverted signals for negative weights) since the ANN has nine inputs neurons. The input voltage should be small enough to ensure that the state of the RRAM device doesn't get altered. Since the pixel values of the input images are binary, i.e. 0 for black pixels and 1 for white pixels, the input voltages are +0.1 V for white pixels and -0.1 V for black pixels. The input voltages are multiplied by the conductance of the RRAM devices and the resultant currents are summed in each column. The first column (from left) corresponds to the alphabet 'n', the second column to 'v' and the third column to 'z'. The output current at each column is measured and whichever column has the highest current, the input image is classified as the alphabet corresponding to that column.

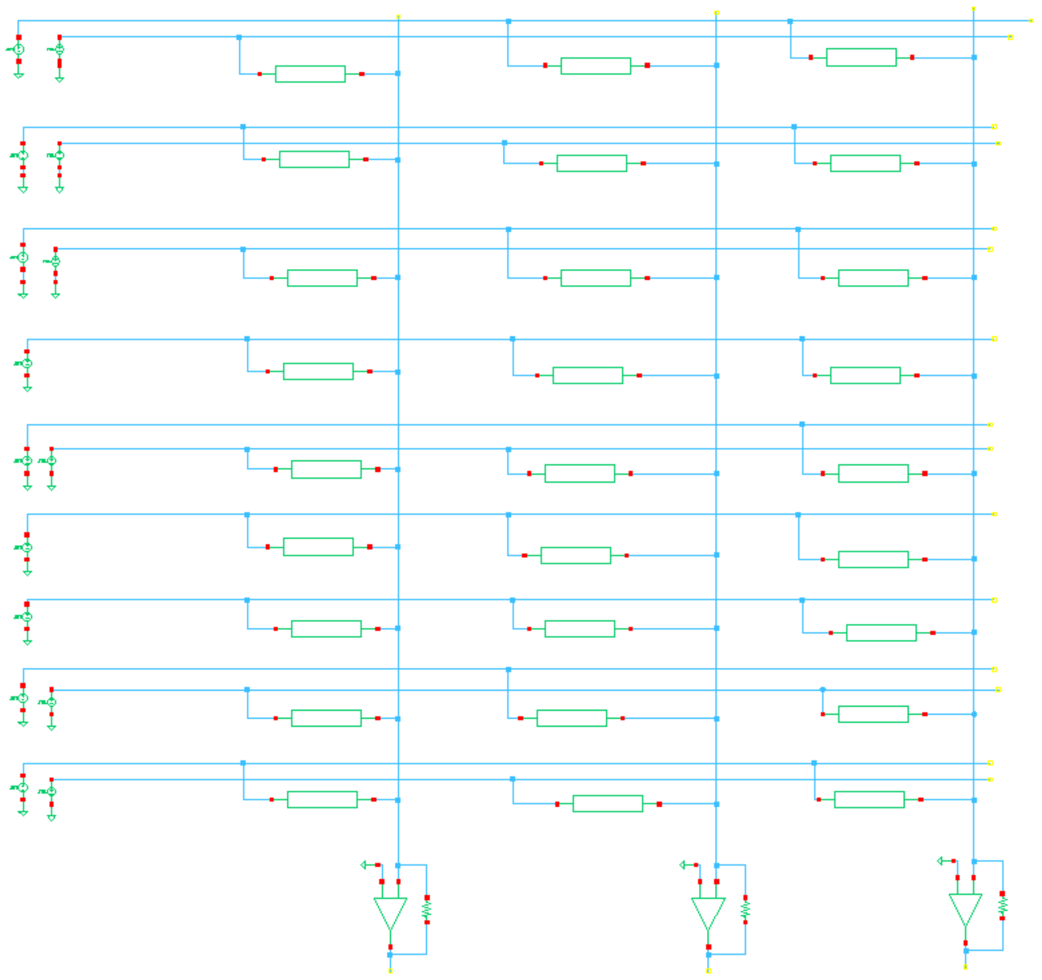


Figure 4.6: Schematic of the crossbar array

The input voltages and output currents of the circuit when an image of the alphabet 'v' is given as an input are shown in 4.7 and 4.8 respectively. Observe that the output current is highest in the 2^{nd} column which implies that the input image has been classified correctly.

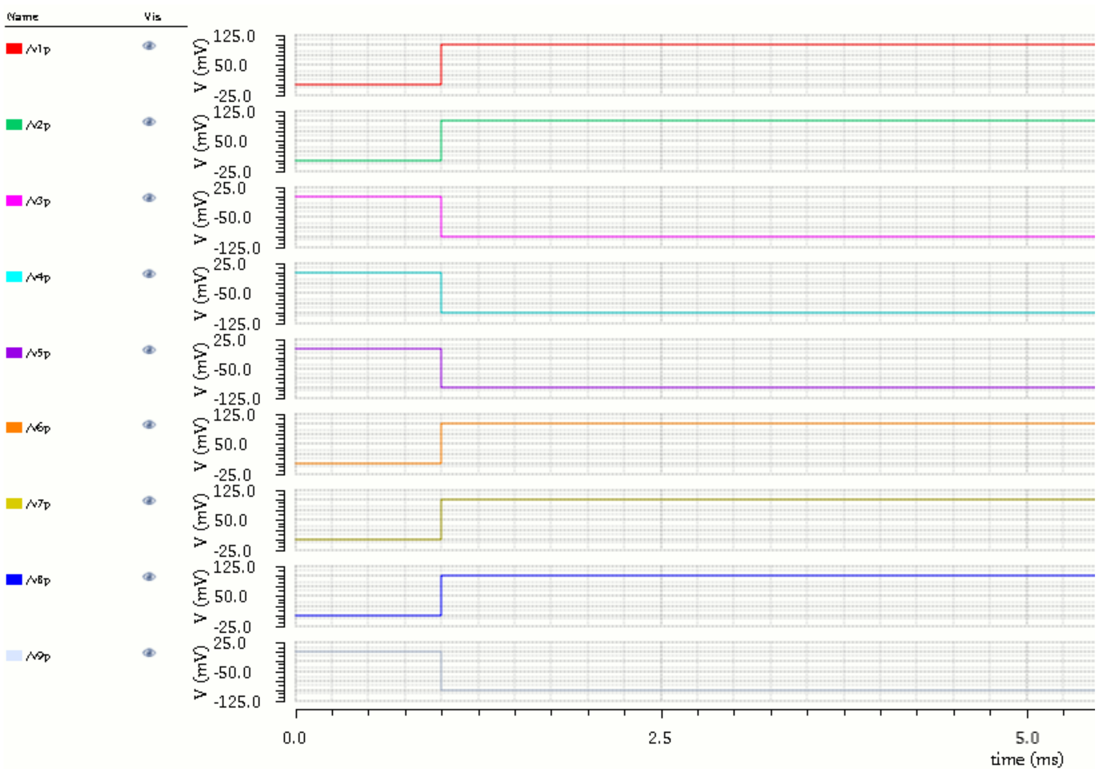


Figure 4.7: Input voltage waveforms applied to the crossbar array for the alphabet ’v’. Inputs applied at t = 1 ms

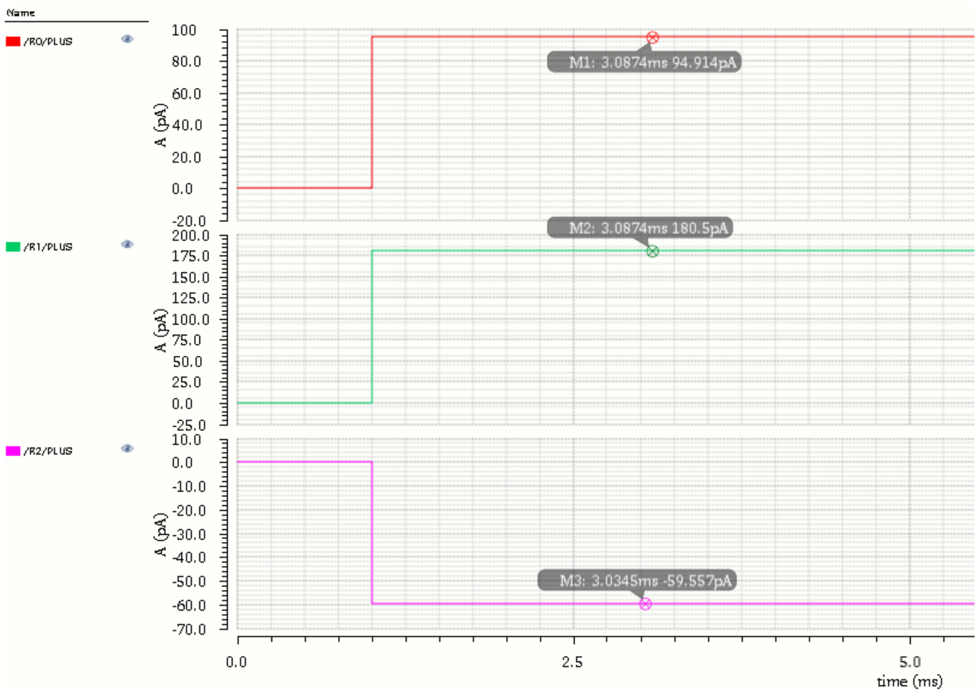


Figure 4.8: Output currents of the crossbar array for the alphabet ’v’ applied as the input

The crossbar array implemented above achieves a classification accuracy of around 68% for the 30 training images. This is lesser than the software implementation which had an accuracy of 80%. The main reason behind the drop in accuracy is due to the discrete conductance states of the RRAM. The weights obtained from training in software have to be mapped to the nearest conductance state. Since the device has discrete conductance states, the weights can't be mapped to the exact desired values. This causes a slight decrease in the classification accuracy.

Future Work

The spiking neural network for spoken digit recognition has shown good results as evident by the patterns that it has learnt. However, the neuron model used in the SNN was the leaky-integrate-and-fire neuron which is a simplified version of biological neurons. More sophisticated models such as the Izhikevich neurons and adaptive threshold neurons can improve the performance by automatically adjusting the spiking rate of the neurons similar to biological neurons. While they are more biologically-realistic, the circuit implementation of these models is more complicated than the LIF neurons. The tradeoff between model complexity and accuracy needs to be analyzed for the specific task at hand.

As discussed in the SNN chapter, fully unsupervised learning struggles to learn the patterns whereas supervised learning is able to learn the desired patterns. From a biological viewpoint, a combination of supervised and unsupervised learning is more plausible. There is a need to explore partially unsupervised learning by incorporating approaches such as lateral inhibition in the output layer. Lateral inhibition is the capacity of an excited neuron to reduce the activity of its neighboring neurons. This generates competition between the neurons and enables the connected synapses to learn different patterns. Whenever an output neuron fires, the membrane potential of all other output neurons is reset. For implementing lateral inhibition, each output neuron needs to be connected to all other output neuron which would increase the complexity and control signals of the SNN.

An important task would be the implementation of the SNN at the circuit level using the RRAM model. Starting with inference, the eventual goal would be to perform the training as well as inference at the circuit level.

Another interesting line of work would be to explore the use of variability as a regularizer during training larger SNNs. This would also account for noisy input signals and help train the SNN in a robust manner.

Finally, as the RRAM model is improved to reflect the gradual nature of the reset process, it would be interesting to study its effect on STDP and on the overall training of the spiking neural network.

References

[1] L. Chua. “Memristor-The missing circuit element”. In: *IEEE Transactions on Circuit Theory* 18.5 (1971), pp. 507–519. DOI: 10.1109/TCT.1971.1083337.

[2] Dmitri B. Strukov et al. “The missing memristor found”. In: *Nature* 453.7191 (May 2008), pp. 80–83. DOI: 10.1038/nature06932. URL: http://dx.doi.org/10.1038/nature06932.

[3] H.-S. Philip Wong et al. “Metal–Oxide RRAM”. In: *Proceedings of the IEEE* 100.6 (2012), pp. 1951–1970. DOI: 10.1109/JPROC.2012.2190369.

[4] Dario Amodei. *AI and Compute*. June 2021. URL: https://openai.com/blog/ai-and-compute/.

[5] K. Moon et al. “RRAM-based synapse devices for neuromorphic systems”. In: 213 (2019), pp. 421–451. DOI: 10.1039/c8fd00127h. URL: http://dx.doi.org/10.1039/c8fd00127h.

[6] Adnan Mehonic et al. “Memristors—From In-Memory Computing, Deep Learning Acceleration, and Spiking Neural Networks to the Future of Neuromorphic and Bio-Inspired Computing”. In: *Advanced Intelligent Systems* 2.11 (Aug. 2020), p. 2000085. DOI: 10.1002/aisy.202000085. URL: http://dx.doi.org/10.1002/aisy.202000085.

[7] Francis Crick. “The recent excitement about neural networks”. In: *Nature* 337.6203 (Jan. 1989), pp. 129–132. DOI: 10.1038/337129a0. URL: http://dx.doi.org/10.1038/337129a0.

[8] Wei Wang et al. “Learning of spatiotemporal patterns in a spiking neural network with resistive switching synapses”. In: *Science Advances* 4.9 (2018). DOI: 10.1126/sciadv.aat4752. eprint: https://advances.sciencemag.org/content/4/9/eaat4752.full.pdf. URL: https://advances.sciencemag.org/content/4/9/eaat4752.

[9] Kyungah Seo et al. “Analog memory and spike-timing-dependent plasticity characteristics of a nanoscale titanium oxide bilayer resistive switching device”. In: *Nanotechnology* 22.25 (May 2011), p. 254023. DOI: 10.1088/0957-4484/22/25/254023. URL: https://doi.org/10.1088/0957-4484/22/25/254023.

[10] Sungho Kim, ShinHyun Choi, and Wei Lu. “Comprehensive physical model of dynamic resistive switching in an oxide memristor”. In: *ACS nano* 8.3 (Mar. 2014), pp. 2369–2376. ISSN: 1936-0851. DOI: 10.1021/nn405827t. URL: https://doi.org/10.1021/nn405827t.

[11] Yury Matveyev et al. “Crossbar nanoscale HfO 2-based electronic synapses”. In: *Nanoscale research letters* 11.1 (2016), pp. 1–6.

[12] Larry F Abbott and Sacha B Nelson. “Synaptic plasticity: taming the beast”. In: *Nature neuroscience* 3.11 (2000), pp. 1178–1183.

[13] Natalia Caporale and Yang Dan. "Spike Timing–Dependent Plasticity: A Hebbian Learning Rule". In: *Annual Review of Neuroscience* 31.1 (July 2008), pp. 25–46. DOI: 10.1146/annurev.neuro.31.060407.125639. URL: http://dx.doi.org/10.1146/annurev.neuro.31.060407.125639.

[14] Guo-qiang Bi and Mu-ming Poo. "Synaptic modification by correlated activity: Hebb's postulate revisited". In: *Annual review of neuroscience* 24.1 (2001), pp. 139–166.

[15] Zizhen Jiang et al. "Verilog-A compact model for oxide-based resistive random access memory (RRAM)". In: *2014 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*. IEEE, Sept. 2014. DOI: 10.1109/sispad.2014.6931558. URL: http://dx.doi.org/10.1109/SISPAD.2014.6931558.

[16] G González-Cordero et al. "A new compact model for bipolar RRAMs based on truncated-cone conductive filaments—a Verilog-A approach". In: *Semiconductor Science and Technology* 31.11 (2016), p. 115013.

[17] Marc Bocquet et al. "Compact modeling solutions for oxide-based resistive switching memories (OxRAM)". In: *Journal of Low Power Electronics and Applications* 4.1 (2014), pp. 1–14.

[18] D. Ielmini and V. Milo. "Physics-based modeling approaches of resistive switching devices for memory and in-memory computing applications". In: *Journal of Computational Electronics* 16.4 (Nov. 2017), pp. 1121–1143. DOI: 10.1007/s10825-017-1101-9. URL: http://dx.doi.org/10.1007/s10825-017-1101-9.

[19] Mark Gales and Steve Young. "The Application of Hidden Markov Models in Speech Recognition". In: *Foundations and Trends® in Signal Processing* 1.3 (2007), pp. 195–304. DOI: 10.1561/2000000004. URL: http://dx.doi.org/10.1561/2000000004.

[20] Nathan Smith and Mark Gales. "Speech recognition using SVMs". In: *Advances in neural information processing systems*. 2001, pp. 1197–1204.

[21] Ali Bou Nassif et al. "Speech recognition using deep neural networks: A systematic review". In: *IEEE access* 7 (2019), pp. 19143–19165.

[22] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1999.

[23] Kishore Prahlad. *Speech technology:A practical Introduction*. University Lecture. 2005.

[24] Jinhai Hu et al. "Voice Keyword Recognition Based on Spiking Convolutional Neural Network for Human-Machine Interface". In: *2020 3rd International Conference on Intelligent Autonomous Systems (ICoIAS)*. 2020, pp. 77–82. DOI: 10.1109/ICoIAS49312.2020.9081859.

[25] Jibin Wu, Yansong Chua, and Haizhou Li. "A Biologically Plausible Speech Recognition Framework Based on Spiking Neural Networks". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489535.

[26] Guillaume Bellec et al. "A solution to the learning dilemma for recurrent networks of spiking neurons". In: *Nature communications* 11.1 (2020), pp. 1–15.

[27] Mark D Skowronski and John G Harris. "Automatic speech recognition using a predictive echo state network classifier". In: *Neural networks* 20.3 (2007), pp. 414–423.

[28] Yong Zhang et al. "A digital liquid state machine with biologically inspired learning and its application to speech recognition". In: *IEEE transactions on neural networks and learning systems* 26.11 (2015), pp. 2635–2649.

[29] Anthony M Zador. "A critique of pure learning and what artificial neural networks can learn from animal brains". In: *Nature communications* 10.1 (2019), pp. 1–7.

[30] James L McClelland. "How far can you go with Hebbian learning, and when does it lead you astray". In: *Processes of change in brain and cognitive development: Attention and performance xxi* 21 (2006), pp. 33–69.

[31] Yoonsuck Choe. "Anti-Hebbian Learning". In: *Encyclopedia of Computational Neuroscience*. Ed. by Dieter Jaeger and Ranu Jung. New York, NY: Springer New York, 2013, pp. 1–4. ISBN: 978-1-4614-7320-6. DOI: 10.1007/978-1-4614-7320-6_675-1.

[32] Sheng-Yang Sun et al. "Cases Study of Inputs Split Based Calibration Method for RRAM Crossbar". In: *IEEE Access* 7 (2019), pp. 141792–141800. DOI: 10.1109/access.2019.2944417. URL: http://dx.doi.org/10.1109/ACCESS.2019.2944417.

[33] W. Schiffmann, M. Joost, and R. Werner. "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons". In: 1994.

[34] Furqan Zahoor, Tun Zainal Azni Zulkifli, and Farooq Ahmad Khanday. "Resistive Random Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (mlc) Storage, Modeling, and Applications". In: *Nanoscale Research Letters* 15.1 (Apr. 2020), p. 90. ISSN: 1556-276X. DOI: 10.1186/s11671-020-03299-9. URL: https://doi.org/10.1186/s11671-020-03299-9.