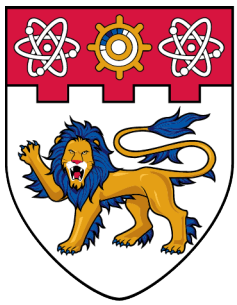
Spin-based Neuromorphic Computing (Simulation)



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SUBMITTED
BY

Chong Tian En

DIVISION OF PHYSICS & APPLIED PHYSICS
SCHOOL OF PHYSICAL AND MATHEMATICAL SCIENCES

A final year project report
presented to
Nanyang Technological University
in partial fulfillment of the
requirements for the
Bachelor of Science (Hons) in Physics / Applied Physics
Nanyang Technological University

June 2020

Spin-based Neuromorphic Computing (Simulation)

Abstract

In the recent year of artificial intelligence and spintronics memory device technology advancement, there is a potential to create high performance and low power neuromorphic network, a hardware-based implementation of neural network. Spintronics memory device is involved in the design of a synapse in a neuromorphic network. In this project, we designed 4 versions of neuromorphic network, trained MNIST dataset off-ship on TensorFlow platform, post-processed the trained weights into 8 levels, discretised form, corresponding to the weight range representable by SOT/SHE MRAM, before simulating the same dataset on the neuromorphic network in Cadence Virtuoso. The intermediate output of TensorFlow was used to simulate the 2nd layer (10 by 20 synapses) and achieved an accuracy of 81.02% vs TensorFlow model accuracy of 80.24%. We have also attempted to simulate a full, multi-layer network but faced with scaling challenges. Furthermore, we studied the challenges posed by the practical, manufacturable and non-ideal neuromorphic network in detail. Future work may include sorting out the shortcoming in the current implementation of neuromorphic network, extending to very large scale simulation, simulating the behaviour model of read/write cycles of MRAM in Cadence Virtuoso, conversion to spike-based (SNN) architecture and ultimately on-chip training of the SNN network.

Supervisor: Assoc Prof S. N. Piramanayagam

Co-Supervisor: Asst Prof Mohamed M. Sabry

Acknowledgements

I would like to thank my supervisor, Assoc Prof S. N. Piramanayagam, for introducing me to Spintronics technologies, keeping me on the right track and training me in my presentation skill.

I would like to acknowledge Asst Prof Mohamed M. Sabry for his guidance through each stage of the design process and defining challenging next steps.

My PhD mentor, Arko, was instrumental in assisting me in setting up the software environment and diagnosing software technical problem.

I am also thankful to HESL lab manager, Mr Chua for providing reliable server access so that I can work remotely from home on this project, particularly in times of pandemic.

Also not forgetting Cadence Design System and foundry partners for making the software and technologies available for access on the computing platform in the lab.

For these, I am extremely grateful to all of them.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

1.1.1 Artificial Intelligence

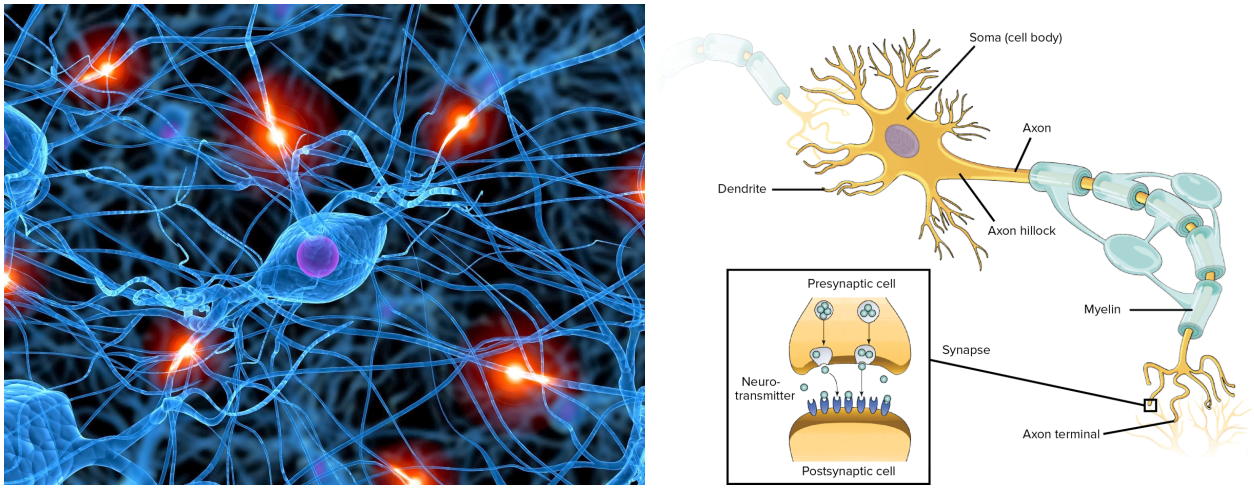
Artificial Intelligence (AI) is widely used in today's industry. It is an emerging field where machines are empowered to mimic human thinking, decisions and actions, to increase industries productivity in the sense of higher-level cognitive processing. In the not so distant past where robotics were programmed and used in manufacturing, today robotics incorporate not just programmed logic but also trained experience via continual exposure to external stimuli, action feedback and judgement adaptation to make a decision even on abstract information. For example, in the field of image classification, images are a 2D array of data. In a traditional programmable logic computing, it is architecturally impossible to recognize its unstructured content as it only deals with well defined, structured data. However, with new AI architecture like a neural network, it is possible to leverage the similar and minimalistic structural architecture of a human brain, to efficiently and precisely deal with the processing of such data. The more complex structure would mean a better ability to extract subtler and finer details. Based on these details, the classification of images can then be made. In the medical field, we have a classification of images for the presence or severity of the tumour. In the search engine, we have a classification of images for objects, scenery, etc. In a self-driving vehicle, we also have a classification of images for objects on road (See Figure 1).

Figure 1: Objects recognised by AI on the road [1]

Beside image classifications, AI is widely used in control and actuation. In a self-driving vehicle, AI can process data collected from its surrounding through different types of sensors, not just data from the objects to recognise their presence, but also data from intrinsic measurable, like distance to those objects and their trajectories, to steer the vehicle in the right direction and away from danger. For example, if a child suddenly ran out from nowhere on the road, an AI can force the

vehicle to stop, and effectively avoid an accident.

1.1.2 Biological Neural Network



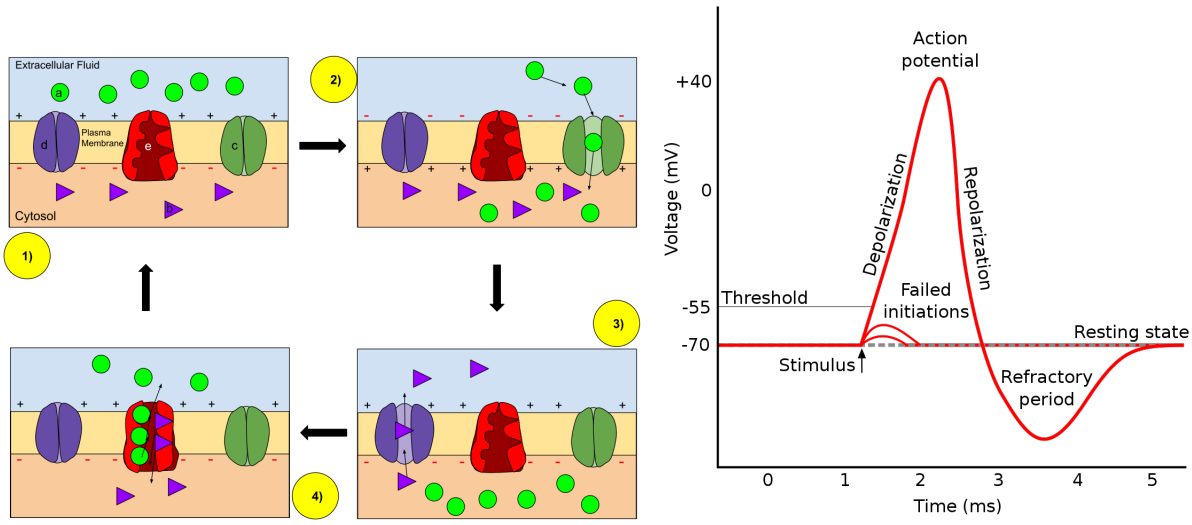
(a) A biological neural network

(b) A biological neuron

Figure 2: The biological neural network

The biological neural network (Figure 2) found in a human brain structure is extremely complex. A human brain contains more than 8 billion neurons and more than 100 trillion synapses (the connection between neurons). A neuron is a nerve cell that communicates with other cells via nerve impulses. It receives signals from one end known as dendrites, processes them via a mechanism known as an action potential, and send signal via elongated axon to another end. For the signal to pass from this terminal to the next neuron dendrite, it has to cross between a gap known as the synapse. The synapse can grow weaker or stronger over time, determining the strength of the synaptic connection, known as neural plasticity.

A neuron maintains a negative voltage (i.e. polarized) across its membrane at $-70mV$. It is achieved by complex protein structures sitting on the membrane known as an ion channel and ion pump. Whenever there are stimuli from the dendrites, they are integrated over time. As these stimuli are not perfect spikes, their strength dies down through time. There are 2 types of stimuli, one is excitatory post-synaptic potential, while the other is inhibitory post-synaptic potential. We can thus think of them as plus signal and minus signal. When these signals of equal strength come together at the same time, they cancel out each other and the net effect on a neuron is 0. If the sum of stimuli has a net positive effect, they come into the action potential mechanism picture.



(a) Biological neuron membrane

(b) Action Potential

Figure 3: Across a neuron membrane

In Figure 3, for a neuron to fire, i.e. generate a spike to other neurons downstream, the stimuli strength needs to cross the threshold at $-55mV$. Weak stimuli result in failed initiation action potential, whereas a strong stimulus results in depolarisation, triggering its action potential, peaking at about $+40mV$ before dropping back down, known repolarization. This repolarization process can cause its voltage to overshoot its initial negative voltage, before recovering to its resting state, known as the refractory period, and be ready for the next cycle.

In various stages of the action potential, the permeability of the neuron changes, as there are an exchange of ions. These are the 4 stages of an action potential cycle:

1. Resting-State:

   Sodium (Na^{2+}) and potassium (K^{+}) ions have difficulty to pass through the membrane, and the neuron has a net negative charge internally.

2. Depolarisation:

   This happens when the action potential is triggered. The sodium channels are activated, allow sodium ions to flow into the cell through the channel, resulting in a positive potential difference with respect to the extracellular fluid outside.

3. Repolarization:

   This happens when the action potential peak is reached. The sodium channels close and potassium channel open, allowing potassium ions to flow out of the membrane through its channel into the extracellular fluid, reducing the membrane potential to a negative value.

4. Refractory Period:

   The voltage-dependent ion channels are inactivated as the ions return to their resting distribution state across the membrane. The neuron is ready for the next action potential cycle.

Memory is stored in the synaptic connection strength [4], subjected to its plasticity. This plasticity exists for both short term and long term. Short term means this type of change only lasts for a time below a second before reverting to normal, whereas long term plasticity can last for years. Long term synaptic plasticity involves long term potentiation (LTP), i.e. strengthening or long term depression (LTD) i.e weakening. Intuitively, the frequent activity of synapse results in LTP, i.e. better memory, whereas prolonged infrequent activity result in LTD, i.e. memory loss. Neuron firing timing contributes to LTP and LTD too. For two neurons that are connected through a synaptic junction, if a pre-synaptic (i.e. previous) neuron fires $20ms$ or less before the post-synaptic (i.e. next) neuron, it results in LTP for the synapse connecting them. However, if a post-synaptic neuron fires $20ms$ or less before pre-synaptic neuron, it results in LTD. Hebbian theory, in particular, states that coincidental activity of synaptically connected neuron leads to lasting change in the effectiveness of synaptic transmission [5]. In other words, it means neurons that fire together, are connected together.

1.1.3 Artificial Neural Network

An artificial neural network (ANN) is an architecture that mimics the human brain structure, which consists of synapses and neurons. Similarly, an ANN can receive inputs and process those input before reaching its outputs. Output can then be used for classification. For example, in a self-driving car, the input would be from the camera and sensors, and the output would be steering direction and magnitude and acceleration. Input can be in many form, i.e. 1D vector or 2D matrix. Processing would then involve passing these input through many neuron layers, with each deeper hidden layer extracts subtler details (See Figure 4).

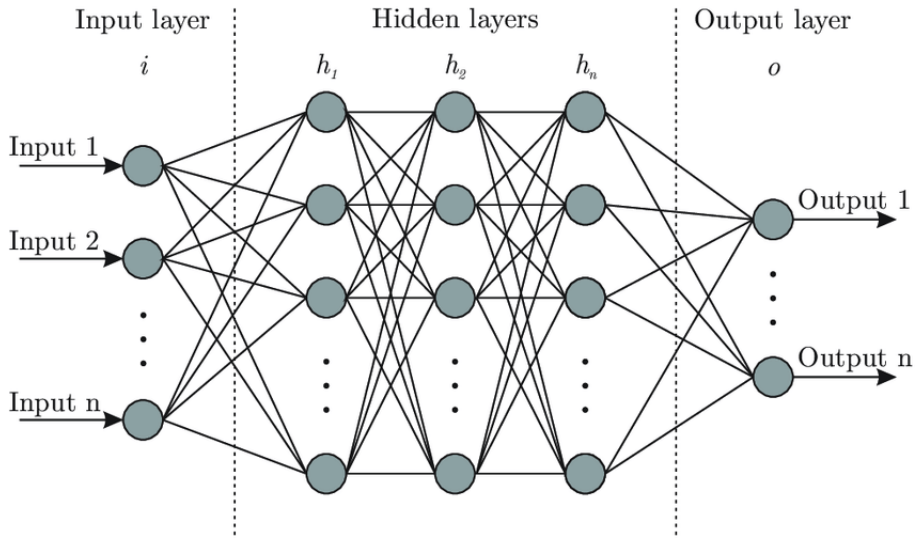


Figure 4: Fully connected neural network overview [2]

For example, the input layer (1^{st} layer) will have nodes/neurons with its numbers correspond to the number of inputs, i.e. 1D vector with n elements would have n number of inputs, whereas 2D matrix with $m \times n$ elements would have $m \times n$ number of inputs. In a fully connected layer (i.e. a dense layer), each neuron in the previous layer is connected to every neuron in the next layer. The connection between neuron A and neuron B is known as a synapse. Each of this synapse carries a weight and a stronger synaptic connection would mean a greater weight. This weight would then act on the input signal from the neuron to extract certain details. Basic processing is done as followed:

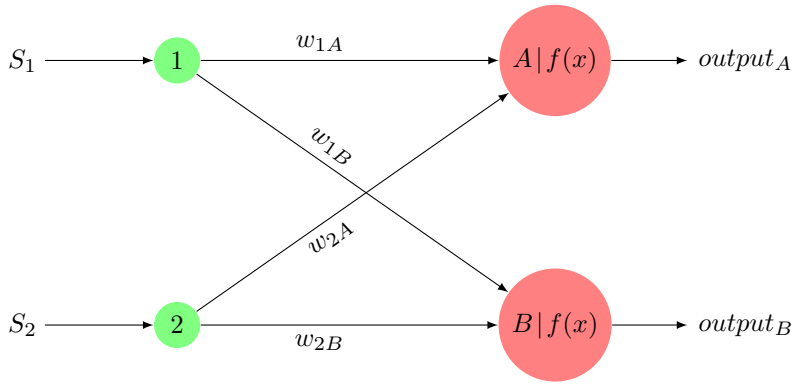


Figure 5: A simple artificial neural network illustration

A layer is defined as a column of arrows, with/without weight attached, and usually ended with a column of neurons. Suppose in the input layer we have neuron 1 and neuron 2, and layer 2 we have neuron A and neuron B. Neuron 1 and neuron 2 synaptic connection to neuron A would be w_{1A} and w_{2A} respectively. Likewise, neuron 1 and neuron 2 synaptic connection to neuron B would be w_{1B} and w_{2B} respectively. Let say after layer 2 is the output layer:

1. Neuron 1 and neuron 2 receive signal S_1 and signal S_2
2. Before reaching neuron A, by multiplying with the synaptic weight, the 2 signals become $net_A = S_1 \times w_{1A} + S_2 \times w_{2A}$
3. Before reaching neuron B, by multiplying with the synaptic weight, the 2 signals become $net_B = S_1 \times w_{1B} + S_2 \times w_{2B}$
4. The 2 signals for neuron A can be fed into an activation function $f(x)$, hence the output for neuron A becomes $output_A = f(net_A)$
5. The 2 signals for neuron B can be fed into an activation function $f(x)$, hence the output for neuron B becomes $output_B = f(net_B)$

Here, we are assuming that we already have the weight. i.e. the network is in inference mode. If we have defined inputs and desired corresponding outputs but without the weights, such that we need to figure out each weight, we can put the network into training mode to set the weight. A common way to work out the weights would be back propagation technique. In a simple back propagation technique, we would randomly set the weight of each of the synapses. Then, we inject signals into the input layer and measure the output from the last layer. We then compare the difference between the desired output and the output from the network, known as the error. From there, we can use partial differentiation to work out the error dependencies and tune the corresponding weight accordingly. The exact detail for back propagation technique for previous example is as followed:

1. Calculate error:

$$E_{total} = E_{output\,A} + E_{output\,B} = \frac{1}{2}(desired_A - output_A)^2 + \frac{1}{2}(desired_B - output_B)^2$$

. Here, the purpose for the error for each of the output to appear in a halved and squared form is to simplify differentiation.

2. we want to know how much w_{1A} contributes to the total error E_{total}. Applying chain rule, we get:

$$\frac{\partial E_{total}}{\partial w_{1A}} = \frac{\partial E_{total}}{\partial output_A} \times \frac{\partial output_A}{\partial net_A} \times \frac{\partial net_A}{\partial w_{1A}}$$

3. As mentioned before in the $1st$ step, we use squared form to facilitate differentiation, here we have:

$$\frac{\partial E_{total}}{\partial output_A} = -(desired_A - output_A)$$

4. And, $\frac{\partial output_A}{\partial net_A}$ would depend on the activation function used, we will obtain a numerical value here too.

5. Also,

$$\frac{\partial net_A}{\partial w_{1A}} = S_1$$

6. After obtaining numerical value for $\frac{\partial E_{total}}{\partial output_A}$, $\frac{\partial output_A}{\partial net_A}$, $\frac{\partial net_A}{\partial w_{1A}}$, we can finally have the value for $\frac{\partial E_{total}}{\partial w_{1A}}$.

7. We can then proceed to set

$$w_{1A\,(new)} = w_{1A\,(old)} - \eta \times \frac{\partial E_{total}}{\partial w_{1A}}$$

where η is the learning rate. The smaller the value of η (< 1), the slower or more gradual the update is to the new value of w_{1A}, to prevent over adjustment. Over adjustment will lead to divergent behaviours, which is unfavourable especially if we aim to hit the minimum error spot. (See Figure 6).

8. we can repeat the same process above to set for w_{1B}, w_{2A}, w_{2B}.

The same technique can be extended to n layers fully connected network.

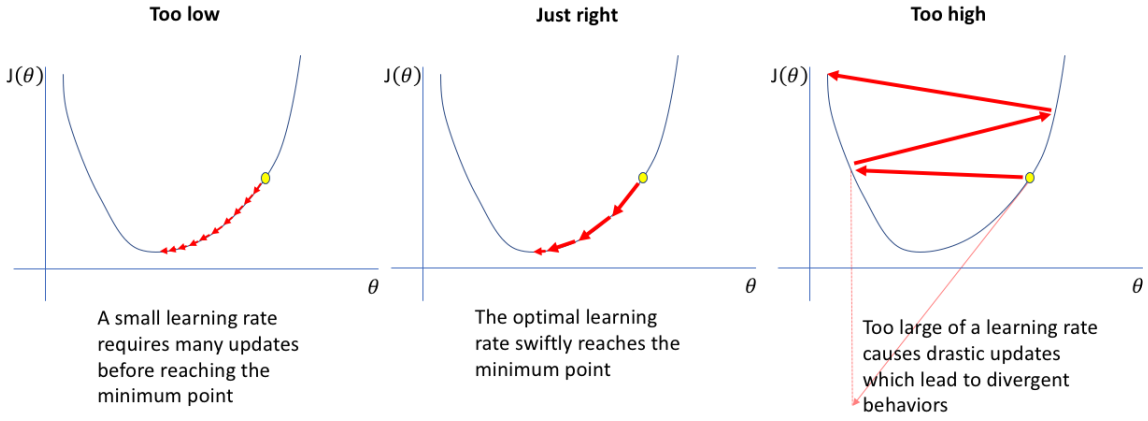


Figure 6: Different learning rate and their contribution to the Cost Function $J(\theta)$ where θ is any parameter we want to optimize (e.g. weight w) and $J(\theta)$ is the function that computes Error E_{total} [3]

1.1.4 von Neumann Computing

In traditional computing, von Neumann architecture is being used. That means processing is only done in the central processing unit (CPU) and there constant shuttling of data between different memory device. There are mainly 2 types of memory devices on a traditional computing system, i.e. CPU cache and random access memory (RAM). A CPU cache is made up of expensive, superfast but small memory capacity whereas RAM is made up of slow but huge memory capacity. All neural network data are loaded to RAM before they are moved to the CPU cache for it to work with and the final processing result is stored back to the RAM. This long-distance bi-directional information flow between CPU and RAM constitutes most of CPU instruction cycles, consuming a huge amount of time and power (Figure 7).

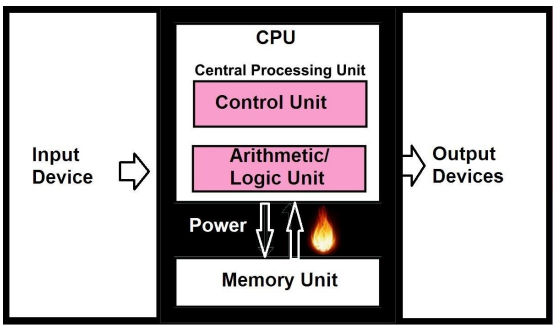


Figure 7: von Neumann architecture

Neuromorphic computing comes into rescue when we have processor specially designed for neural computing. Although information can be analogue/digital, the processing is done with analogue processing. For example, a typical neural network involves summing and multiplication. On a von Neumann architecture, the process flow goes like this:

Let say we are summing number a, b, c:

1. Load number a from RAM to CPU register 1
2. Load number b from RAM to CPU register 2
3. Sum a and b on CPU and store the result in register 3
4. Save the value of $a+b$ in CPU register 3 to RAM
5. Load value of $a+b$ from RAM to CPU register 1
6. Load number c from RAM to CPU register 2
7. Sum the value of $a+b$ and c on CPU and store the result in register 3
8. Save the value of $a+b+c$ in CPU register 3 to RAM

We would require so many steps to compute a simple summation, nevermind that CPU has billions of cycles to take care of these steps.

1.1.5 Neuromorphic Computing

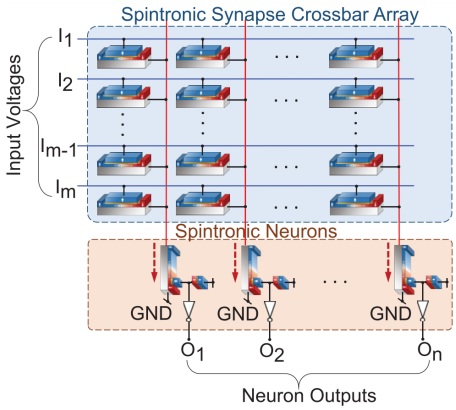


Figure 8: Neuromorphic architecture

In neuromorphic computing (Figure 8), highly specialized circuits are used to do the computation and physical laws like fundamental circuit laws are leveraged. Simple mathematical computation like multiplication and summing that take many processing steps and cycles in von Neumann computing is now irrelevant because Ohm's law is used to do the multiplication and Kirchoff's current law is used to do the summation in massive parallelism, which is an operation known as in-memory computing.

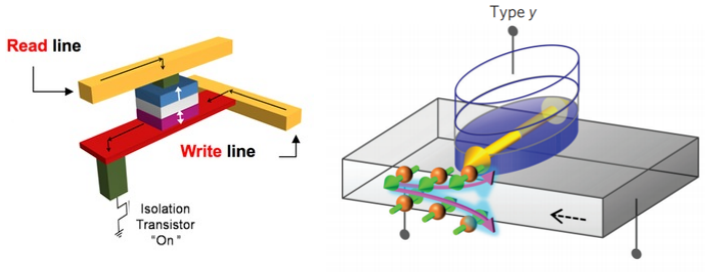
However, due to the non-programmable, non-sequential processing and highly specialized nature of the circuit, it is designed solely for this purpose. It cannot run the software as von Neumann architecture does, so we can forget about running a computer operating system, watching YouTube videos, checking email, etc. Such circuit is known as an application-specific integrated circuit (ASIC). One such example is IBM TrueNorth chip, it consumes as little as $1/10000th$ the power of a traditional von Neumann computer.

What it lacks in its functionality, it more than makes up for in performance in terms of speed and power consumption. Multiple steps in a simple computation in von Neumann architecture that can scale easily up to millions of steps depending on the number of inputs can deter the network efficiency and its response in realtime because the time in sequential computation all add up to a prohibitively large delay. In contrast, all these sequential steps are reduced to a single step in neuromorphic computing architecture as the circuit laws naturally take care of these computations, less the need for purposeful sequential processing.

One notable development in neuromorphic computing is a spiking neural network (SNN). Unlike Von Neumann architecture, that is always on and all instruction executions are synchronized to the CPU clock cycle, SNN is event-driven, i.e. it performs computation and consumes power only when there are external stimuli, and there is no clock cycle to synchronized to. Furthermore, this SNN takes in signals in a form of spikes, unlike all other neuromorphic networks that take in signal in constant signal strength. That is how its power consumption is reduced significantly. Its computation method is biologically inspired, in a way that spikes from the external stimuli are received and accumulated in a neuron, and when a threshold is reached, the neuron will generate a spike to other neurons connected to it through a network of synapses.

1.1.6 Spintronics for Neuromorphic Computing

Spintronics make use of electron spin (orientation) to change its resistance. Spin Hall Effect (SHE) is used to write to the spintronic device, whereas Spin Torque Transfer (STT) is used to read the value from the device, in a form of a current (Figure 9).



(a) Separate write and read path [6] (b) Field free switching write operation

Figure 9: SHE/SOT MRAM

In a SHE write operation, when a current consisting of spin up and spin down electron, flowing through a heavy metal (HM) conductor in either direction, the electric field near the atom deflect spin up and spin down electron (much like sorting) to the edge of the conductor according to the spin. So, we can see spin up electron gather along one side or the conductor, whereas spin down on the opposite side. A different direction of current flow will see swapping side of previously mentioned electron spin (i.e. The side that consists of only spin up electron will now be replaced by only spin down, and vice versa). In the event with one side of the conductor is in contact with an electrode (Free layer), the electron spin in the HM can exert reorientation influence on the spin direction of the free layer.

Similarly, In an STT read operation, current consisting spin up and down electron flow through a normal conductor, but in a perpendicular/orthogonal direction from the top to the bottom. Since fixed layer has its electron spin pinned to one particular direction, when viewed together with electron spin in the free layer, we can observe them being parallel or antiparallel electron spin. Parallel spin results in lower resistance compared to that of antiparallel spin. After passing through and filtered by fixed layer (aligned spins pass through, whereas anti-aligned spin gets bounced), electrons have to tunnel through to a thin, insulating oxide layer to reach the free layer (electron antiparallel spin get bounced again here) to form a read current. This structure is known as a magnetic tunnel junction (MTJ) [7].

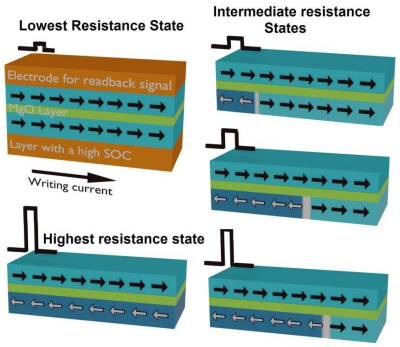


Figure 10: Domain wall device with multiple resistance states

In a domain wall (DW) type MRAM (Figure 10), an MTJ has multiple resistance states instead of just 2 states (low and high). For this project, we consider 8 steps or equivalently 9 resistance states. These states comprise a fully antiparallel state, 7 intermediate states and a fully parallel state. To traverse between these states, multiple short SHE current pulses are applied, to move domain wall in step-like forward and backward fashion, until the desired resistance state is achieved.

Simulation in this project only considers the resistance states of a DW MTJ, not a full simulation model (consisting of both writing and reading operation) because it is not available, and only recently under development by a PhD student.

1.2 Background

There have been numerous development in neuromorphic computing and spintronics.

For neuromorphic computing:

In 1957, there was an invention of the perceptron. Then there was the publication of very large scale implementation (VLSI) architecture for the implementation of the neural network after 29 years. In 1997, we saw the third generation of neural network - spiking neurons being published. There there was more neuromorphic silicon chip invented, mainly by IBM (TrueNorth chip in 2014) and Intel (Loihi chip in 2018).

For spintronics:

In 1989, the GMR effect is discovered in a thin film, GMR application of hard disk head by IBM in 1997. Around the same period, there was an advent of STT. Until recently, we have field-assisted MRAM and STT MRAM.

1.2.1 Literature Review

Different research papers were read and cross-referenced to have a better idea of this project. Concretely, they comprised of around 8 neuromorphic computing related papers, a spintronic paper, a neuromorphic spintronics review paper, and a domain wall based neuromorphic computing paper, with the latter most closely related to this project, supplied by both supervisor and co-supervisor.

A few papers touched on the material and device of crossbar array, involving physical metal oxide (MO) resistive ram (RRAM) by S. Yu et al [8], a simulation model of MO RRAM by M.K.F. Lee et al [9], physical Conductive-Bridge RAM (CBRAM) by M. Suri et al [10], memristor by X. Liu et al [11], domain wall MRAM by D. Kaushik et al [12] and spintronics by J. Grollier et al [13]. They demonstrated the viability of such material and device used in a synapse. S. Yu et al. demonstrated the use of MO as synaptic device and the I-V relationship with multiple continuous SET cycles, followed by RESET cycles variation. These variations although is undesirable but are tolerated, much like what happens in a biological synapse. The performance requirement is more lenient than a digital data storage system because it is part of a massively parallel computing neuromorphic network. It is the collective effect of synapses that determine the firing of a single neuron, so individual variation has much less effect. Still, there is a need to limit these variations. However, shortcoming like 100000 endurance cycle limits the lifespan of the device. Nevertheless, this device consumes a meagre $6pJ$ per operation, and its property of gradual resistance change by pulse amplitude can be used for spike-timing-dependent plasticity phenomenon in an artificial neuron, similar to that of a biological one, and therefore can be used for the neuromorphic network [8]. M.K.F. Lee et al design an RRAM model, by making use of the non-ideality behaviour of the RRAM, they include Stuck-at-Faults where RRAM resistance value refused to change no matter what operation is done on it due to manufacturing

defects, random telegraph noise where RRAM randomly fluctuate between 2 random states due to random trapping and release of charge carriers, as a result of lattice defects and write variability, where the RRAM resistance show random variability during SET/REST operation, due to migration of ionized defects that changes its physical dimension. Monte Carlo simulation was used to simulate the random telegraph noise. They attempted to mitigate write variability using a write-verify scheme and studied its trade-off [9]. M. Suri et al exploit CBRAM variability and stochasticity, which are originality undesirable characteristic to implement probabilistic learning rule. They demonstrated its use in auditory and visual applications with high accuracy, having audio sensitivity of > 2, surpassing that of human and video detection rate $> 95\%$. The synaptic power consumed was ultra-low, with audio and video consuming $0.55\mu W$ and $74.2\mu W$ respectively. However, they found it hard to emulate long term depression behaviour using CBRAM due to difficulty in dissolving the conductive filament controllably. Hence, they decided to switch to binary (1-bit) to reduce the resistance representation resolution [10]. X. Liu et al indicated that memristor (a 2 terminal device which resistance varies as a function of electric charge and magnetic flux, made programmable by duration and amplitude of programming signal) of a same crossbar array row can be programmed all at once, and the positive implication reducing sneak path leakage [11]. D. Kaushik et al pointed out the advantage of using domain wall (DW) device programming compared to resistive random access memory (RRAM) phase-change memory (PCM) programming. In DW device, any resistive changes to the device are symmetrical whereas it is asymmetrical for RRAM and PCM device. Symmetrical means both positive and negative conductance changes of equal magnitude constitute the same pulse number and same energy required. It takes the same $0.18fJ$ for increase/decrease of conductance for DW, but $12pJ - 51pJ/2.28nJ$ for increase/decrease of conductance for RRAM and $5pJ/30pJ$ for increase/decrease for that of PCM. Comparing the total power consumption for on-chip training, the power draws are $9fJ$, $1\mu J$ and $1.1\mu J$ for DW, RRAM and PCM respectively. For on-chip test performance, in ascending order of high accuracy, we have DW (90%), PCM (92%) and RRAM (94%). They also noted the nucleation requirement at the beginning of the training for DW based neuromorphic network, hence the additional time and energy needed. Nevertheless, those energies are insignificant when compared to the total energy involved in training [12]. J. Grollier et al gave a review of spintronics role in neuromorphic computing. The main challenges are footprint and power consumption. Magnetic tunnel junction (MTJ) device is compatible with existing silicon chips manufacturing and can meet the need of small footprint and ultra-low power consumption. They investigate the role of MTJ as synapses and neurons and domain walls as neurons. They also note challenges of scaling up the system in a form of power consumption when more neurons are interconnected and the need for adapting the algorithms to the hardware when the reading of the small change of resistance is slow. They also indicated the possibility of employing low resolution 2 state MTJ device for inference mode with training mode in high-resolution [13].

A few involve employing an existing on-chip system to implement AI application, like a comprehensive study and benchmarking of various digital neuromorphic chips by H.F. Langroudi et al. [14], full-custom hardware implementation by G. Indiveri et al. [15] and conversion of recurrent neural network (RNN) to SNN in IBM TrueNorth chip by P.U. Diehl et al [16]. H.F. Langroudi et al. benchmarked 16 digital neuromorphic chip architectures and found out what optimization contributes to the improvement of performance. They realised the shrinking 16-bit to 8-bit or half the synaptic weight resolution only degrade the accuracy by 0.6%. This saves memory space and reduces computation complexity significantly, while only sustaining a slight reduction in accuracy. They further explored the possibility of binary (1-bit) resolution on a

large dataset but discovered that it reduced by a significant amount. They noted the main optimizations are reducing memory power usage and its bandwidth grant high memory accesses. Input/Output communication overhead is often overlooked when reporting digital system performance (these overheads make up a portion of power usage and time delay) [14]. G. Indiveri et al implement analogue neuromorphic cxQuad (1k neurons and 64k synapses) and ROLLS chip (256 neurons and 128k synapses) with an asynchronous digital circuit. The energy consumptions for both neuromorphic chips at 30Hz operation are $945mW$ and $4mW$ at $1.8V$ respectively. Dynamic vision sensor (DVS) is first used for capturing visual input and it only responds to the changes in the scene. The cxQuad chip used digital circuits to route asynchronous spiking signal within the cores, and across the cores and chips. They pointed out that the analogue circuit is affected by device mismatch. The whole setup was aimed to solve von Neumann bottleneck issue [15]. P.U. Diehl et al stated the importance of RNN in temporal sequence learning and its difficulty in mapping and representing the network in spike-based architecture. They hence introduced a technique, known as train-and-constrain, to backpropagate the signal through time, before quantizing the weight and converting to SNN on a spike-based IBM TrueNorth chip. They conducted a case study on natural language processing (NLP) question classification and achieved 74% accuracy, with only 0.025% chip utilisation, and estimated power consumption of $17\mu W$ compared to traditional hardware like CPU and GPU which can consume at least $100W$ [16].

Those with purely neuromorphic network end-to-end simulation includes very efficient reconfigurable neuromorphic computing accelerator design by X. Liu et al [11], domain wall-based synaptic neural network by D. Kaushik [12], RRAM based neuromorphic computing by M.K.F. Lee et al [9] and conversion from an ordinary convolutional neural network (CNN) to spike-based neuromorphic (SNN) by Y.Q. Cao et al [17]. X. Liu et al demonstrated multilayer perceptron network, with 1st layer as the input layer, a middle layer as a hidden layer and an output layer as the last layer. Hidden layer involves summing at the neuron and applying a sigmoid activation function, known as a summing amplifier. They have router talking to memristor-based crossbar array, and they also imparted hybrid analogue and digital signal conversion in their network. They used the Cadence Virtuoso simulation environment. They made a comparison to a typical von Neumann based processor and their mixed-signal neuromorphic computing accelerator achieved $178.4 \times (27.06\times)$ better performance and $184.2 \times (25.23\times)$ better power efficiency, all while achieving similar computation accuracy [11]. D. Kaushik demonstrated the on-chip DW based neuromorphic network with all neuron output being feedbacked to the DW writing operation of synapses of the same column (only where the summing and multiplication operation are involved, as other columns are not involved in the operations at all). The feedback signal is comprised of a custom cost function after the pre-amplification and activation function. Here, the R_F of the opamp is set to 1Ω to ease the computation complexity. Due to long RESET duration pulse needed for RRAM/PCM, training of these network takes a longer time compared to that of DW type [12]. M.K.F. Lee et al attempted various spiked-based neuromorphic network setup by cascading the network, chaining the network in a ring fashion and pan out the network in a mesh-like fashion and discovered the highest performing type is cascade style network. They developed network architecture with multiple cores consisting of RRAM crossbar array with configurable network interface linking the cores and a router to route the information between them [9]. Y.Q. Cao et al employ steps like negative values elimination, ReLU for neuron activation and remove bias from all neural layers to convert ordinary CNN to tailored CNN, which can be easily converted to SNN. Their architecture involves preprocessing the image dataset into YUV colour space, generate spikes based on the colour values, passed through 2 units of convolution and subsampling layers, followed by a convolution layer and finally a fully connected linear classifier, all with spike-based

neurons. The last layers utilised a spike counter to compare the count rate of the classifier outputs [17].

A handful deal with SNN computing, involving its architectures by G. Indiveri et al [15], conversion of recurrent neural network (RNN) to its spike-based form by P.U. Diehl et al [16], its application in object recognition by Y.Q. Cao et al [17] and spike-based neuromorphic simulation by M.K.F. Lee et al [9]. G. Indiveri et al implemented an adaptive-exponential-integrate-and-fire model to predict real neuron voltage traces and its dynamic behaviour. The synapse circuit of ROLLS chip employs synaptic behaviour like short term plasticity and long-term potentiation to account for the dynamics. They argued that embedded SNN system can deal better with realtime sensory processing application [15]. P.U. Diehl et al quantized the synaptic weight to 4-bit to accommodate the requirement of TrueNorth network parameter. To map to the spiking neurons, network rectified linear units (ReLU) are used without bias. When a spike is generated, the membrane voltage is reduced by a threshold voltage, a minimum voltage required for a spike event [16]. Cao et al make use to converted SNN network to conduct case studies on Neovision2 Tower dataset and Cifar-10 dataset. Neovision2 Tower dataset consists of objects like nontarget, bus, car, cyclist, person and truck for image recognition. The performance in accuracy for both datasets are 98.43% and 99.29% respectively [17]. M.K.F.Lee et al developed a simulation model for spiked based leaky-integrate-and-fire neuron to get a balance between accuracy and complexity and it is realisable in the hardware level [9].

All papers assume the reader to have some level of foundation in training a neural network and testing its efficacy.

1.3 Motivation

Neuromorphic computing offers huge application in AI Computer Vision by mimicking human brain neuron and synapse network. It consumes very low power compared to von Neumann Computing and in-memory processing and hardware-based implementation of the neural circuit make it very efficient for visual object recognition. It can be designed to have multiple deeper layer network that extracts subtler visual details and its final network output can be fed for image classification.

Spintronic devices are originally used in hard disk drive read/write head. They are involved in:

1. Spin Transfer Torque (STT)
2. Spin Orbit Torque (SOT, Spin Hall Effect)
3. Tunneling Magnetoresistance (TMR)
4. Giant Magnetoresistance (GMR)

for a magnetoresistive application like Magnetoresistive Random-access Memory (MRAM). They are non-volatile (information retention during power loss) like solid-state drive (SSD) unlike metal oxide semiconductor (MOS) and capacitors. Similar to SSD, magnetic tunnel junction (MTJ) uses electron tunnelling. The main motivation of using MTJ is it has a smaller footprint compared to that of CMOS. But, MTJ uses electron spin orientation instead of electron presence in SSD. Charge retention in a typical SSD changes its threshold voltage, causing wear out. In contrast, an MTJ offers a further advantage in higher read/write speed, lower power through flipping of electrons' spin rather than the charge motion that tend to result in Joule heating and no wear out (Unlimited read/write cycle).

1.4 Problem Statement

A neuromorphic network is an efficient way to realise neural network, together with domain wall based MRAM with benefits like ultra-fast switching, ultra-low power consumption and excellent memory retention, therefore, it is very desirable to combine these two. An ideal neuromorphic network involves an infinite range of resistance, vastly simplifying the design of the network, however, with the limited range resistance, an additional circuit is needed to overcome the shortcoming. In this study, we create and simulate models of neuromorphic network, taking into account the resistance range of domain wall based spintronic MRAM and compare the performance between neuromorphic model and traditional software implemented neural network model.

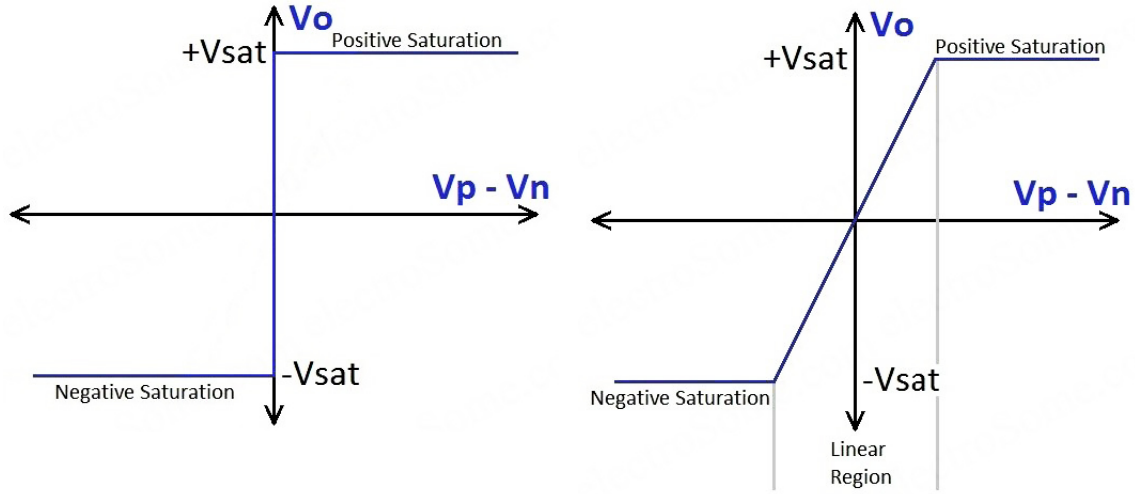
1.5 List of Contribution

In this final year project report:

1. We show a basic construction of a neuron with circuit theory and derivations.
2. We demonstrate Cadence Software implementation of an extended n by m scale neuromorphic circuit.
3. We bridge the gap between theory and practical implementation.
4. We showed a practical, foundry manufacturable, component level implementation of the neuromorphic circuit from actual third party foundry supplied 65nm process design kit, without resorting to the ideal model.
5. We illustrate 4 versions of neuromorphic circuit.
6. We compare and contrast the practical and ideal model with practical limitation.
7. We explain sources of error in a practical model with detailed diagrams and equations.
8. We demonstrate the training of different neural network models for a case study - MNIST handwritten digit database using open-source deep learning TensorFlow, Keras Library.
9. We cherry-pick the best model most suited for transferring to neuromorphic network, with high accuracy.
10. We use computer vision library - OpenCV to demonstrate the result of neuromorphic circuit output for our case study.

2 Design of A Single Neuron

2.1 Theory



(a) Ideal

(b) Non-ideal

Figure 11: Operational amplifier (opamp) characteristics

In an ideal operational amplifier (opamp), it has 2 input channel (inverting "$-$" and non-inverting "$+$"). When DC signals flow into these 2 channels, the difference ($V_+ - V_-$) gets amplified with infinite gain (i.e. infinite gradient), only to be limited by supply voltage, reaching saturation voltage $\pm V_{sat}$. In a non-ideal case, the gain is limited, which means we see a finite gradient.

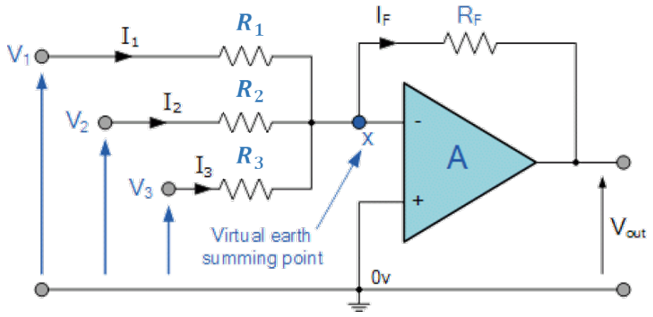


Figure 12: Ideal inverting summing opamp

In an ideal inverting summing opamp (Figure 12), non-inverting input is grounded. inverting input exhibit a critical virtual ground behaviour. Virtual ground here means, it is not actually grounded, but its voltage is $0V$, and also, no current flows into it. So, based on Kirchoff's Current Law, which states that sum of all current flowing in to a node equals to sum

of current flowing out of a node. I_1, I_2, and I_3 flowing into a node marked "x" in the diagram, since they cannot flow into non-inverting input, all of them flow out as I_F into R_F. Since voltage at node x is 0, and a current only flow from low potential to higher potential, we expect V_{out} to be negative, where the magnitude of V_{out} equal sum of all current multiplied by R_F. Concretely,

$$V_{\text{out}} = -R_F \left(\frac{V_1}{R_1} + \frac{V_2}{R_2} + \frac{V_3}{R_3} \right)$$

Given 3 weights of w_1, w_2 and w_3, we would want:

$$\boxed{V_{\text{out}} = w_1 V_1 + w_2 V_2 + w_3 V_3}$$

So, given $R_F = 1$, w_1, w_2 and w_3 correspond to $\frac{1}{R_1}$, $\frac{1}{R_2}$ and $\frac{1}{R_3}$, it infers that:

$$w \propto \frac{1}{R}$$

Now, we have:

$$V'_{\text{out}} = -\left(\frac{V_1}{R_1} + \frac{V_2}{R_2} + \frac{V_3}{R_3} \right)$$

To obtain a positive V_{out}, we can go through the same procedure again (attach another opamp to the existing opamp output):

$$V_{\text{out}} = -R'_F \left(\frac{V'_{\text{out}}}{R'_{\text{out}}} \right) \tag{1}$$

When we set $R'_F = R'_{\text{out}}$, we obtain below as needed:

$$V_{\text{out}} = -V'_{\text{out}} = \frac{1}{R_1} V_1 + \frac{1}{R_2} V_2 + \frac{1}{R_3} V_3 \tag{2}$$

Of course, there are many practical issues when we implement the theory above to simulation circuit. This is be covered in Challenges section later on.

2.2 Simulation Environment

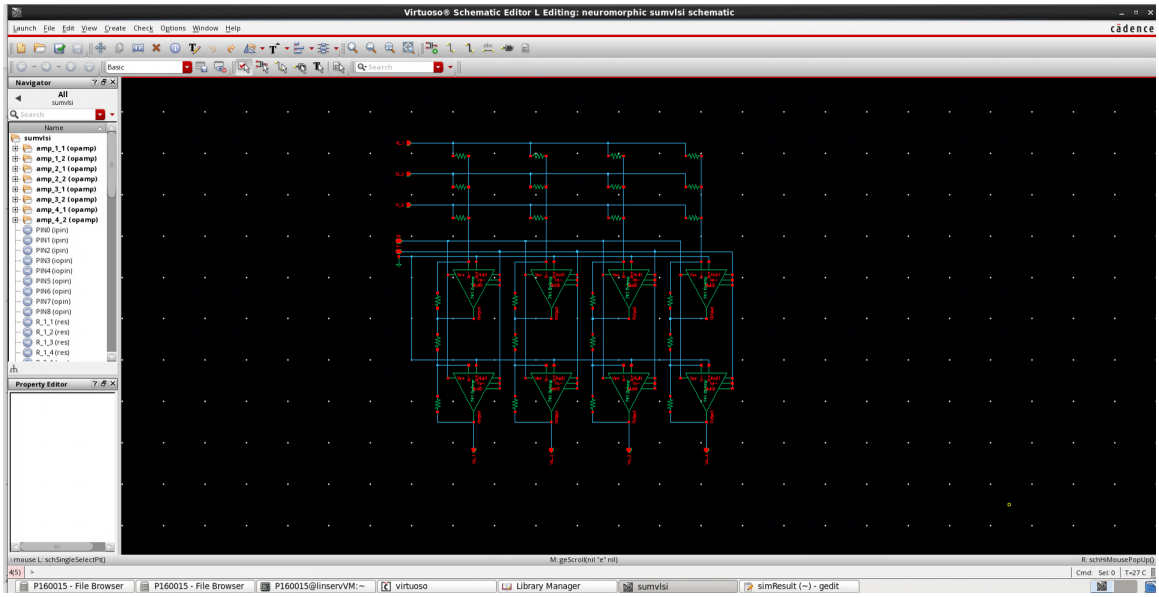


Figure 13: Cadence Design Environment

Cadence environment is very powerful for us to build and simulate circuit, even for very large scale simulation. For instance, we can build silicon level layout, fill in the physics details and represent it in a symbolic form. This symbolic form can be used as circuit component in a schematic circuit. Also, we can represent a simple schematic circuit in a symbolic form too, and use it as circuit component in an even larger schematic circuit. The 8 opamps as shown in 13 are represented with triangle symbols with input and output pins. Besides, it is entirely possible to simulate a component with just programming its behaviour using scripting language with mathematical and physics description, like spiceText, verilogA, and etc.

Cadence has its own scripting language for automating schematic circuit diagram design, known as Skill Script. Automation functionality extends to component placement by specifying coordinates, rotation, vertical and horizontal flipping, instance id, resistance value, methodical wiring routing and etc. A subset of Skill Script is Ocean Script, meant for automating the testing of the design. For example, in Skill script, we can set the resistance value, input voltage signal to variables (i.e. not hardcoded), so during simulation, we can use Ocean Script to manipulate these variables to automate a sequence of simulations.

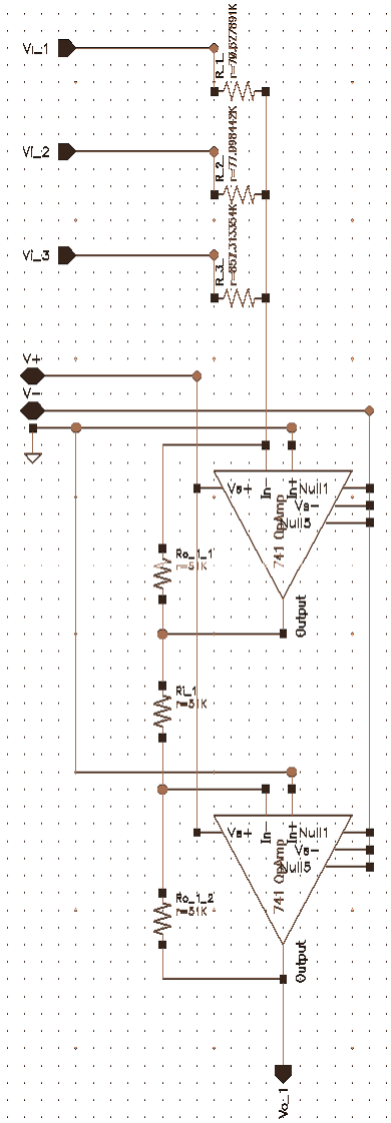


Figure 14: Single summing neuron implemented in Cadence

In Figure 14, we implemented a single summing neuron with 3 voltage source inputs. The diagram was hand-drawn initially before we moved to programmatic schematic design generation. To ensure minimal errors, all resistance values are hardcoded and the expected neuron output (V_{out}) was calculated manually. Then, Cadence Virtuoso Analog Design Environment (ADE) simulator was used to feed in the voltages (supply voltages inclusive) and measure the output. ADE also allowed us to specify any wiring to measure its voltage and specify the endpoint of any circuit schematic component to measure its current. Current flowing into the specified endpoint is rated positive, whereas the current flowing out is rated negative. Of course, the readout value contains errors, not exactly as calculated. We will cover these errors in our challenges section.

3 Extension to Full Macro

Large scale full macro extension involves Skill Script schematic circuit design automation, scaling it up to arbitrary size, because it is not practical and almost impossible to have hand drawn circuit at large scale. 4 version iterations were undergone to overcome design challenges (See Challenges section) and adding new features. All testing wise are automated. Generally, V_{in} are randomly generated. Each of the n random V_{in} has a value $0V \geq V_{\text{in}} \geq \frac{V_{\text{sat}}}{n}$ so that sum of all these values will not exceed V_{sat} and get clipped. Weights w are randomly generated and the output of all neuron are monitored and compared against the ideal values through finding the mean and standard deviation of all neurons.

3.1 1st version

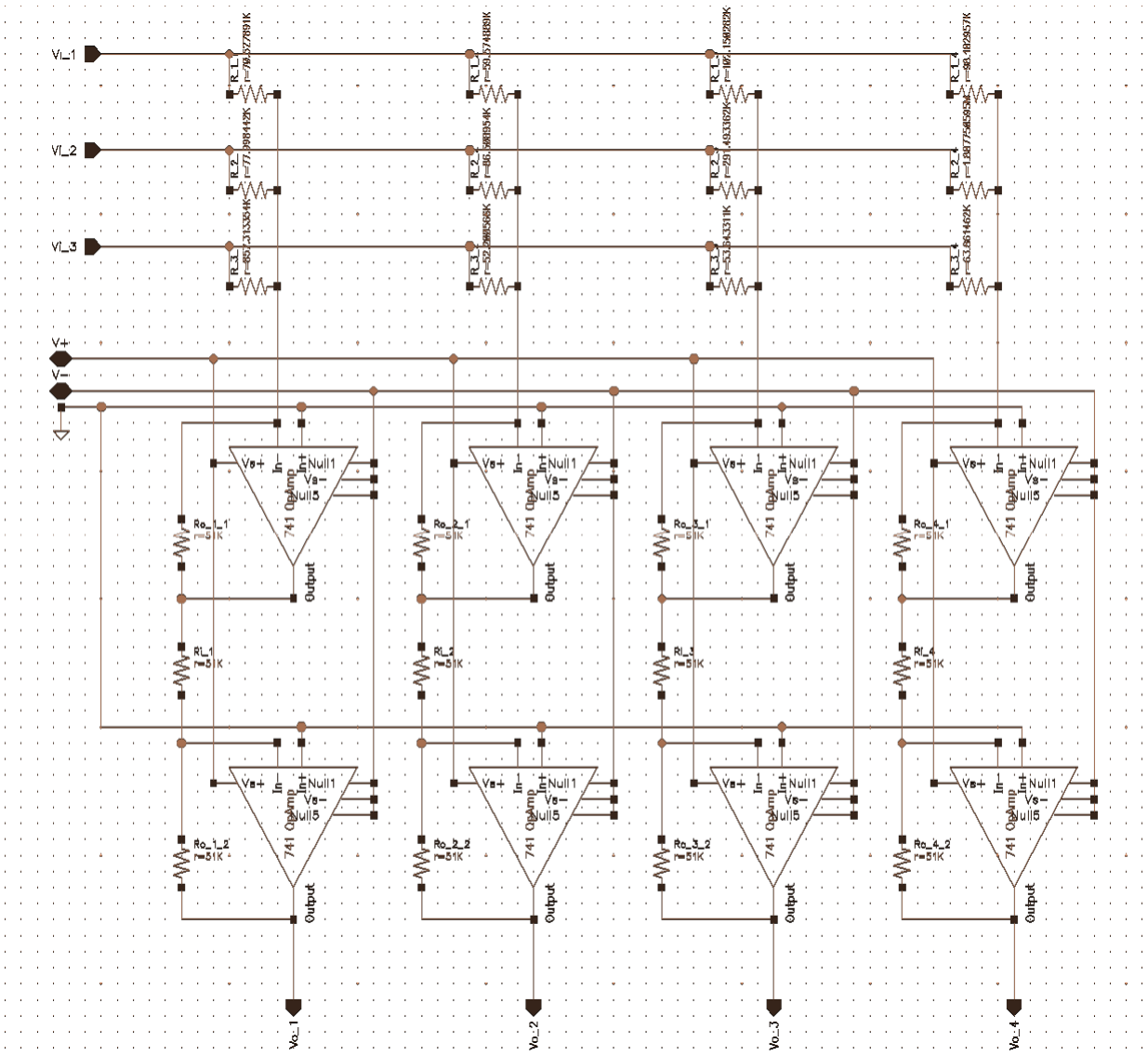


Figure 15: 4 by 3 generated summing neuron design implemented in Cadence

Figure 15 is simply a scaling up of Figure 14. Since the weight is between $0 \geq w \geq 1$, the corresponding resistance is $\infty \leq R \leq 1$, but for practical consideration, ∞ resistance is limited at $\frac{1}{0.001}\Omega = 1000\Omega$ which should suffice when compared to high weight ($w = 1$) or low resistance of ($R = 1\Omega$).

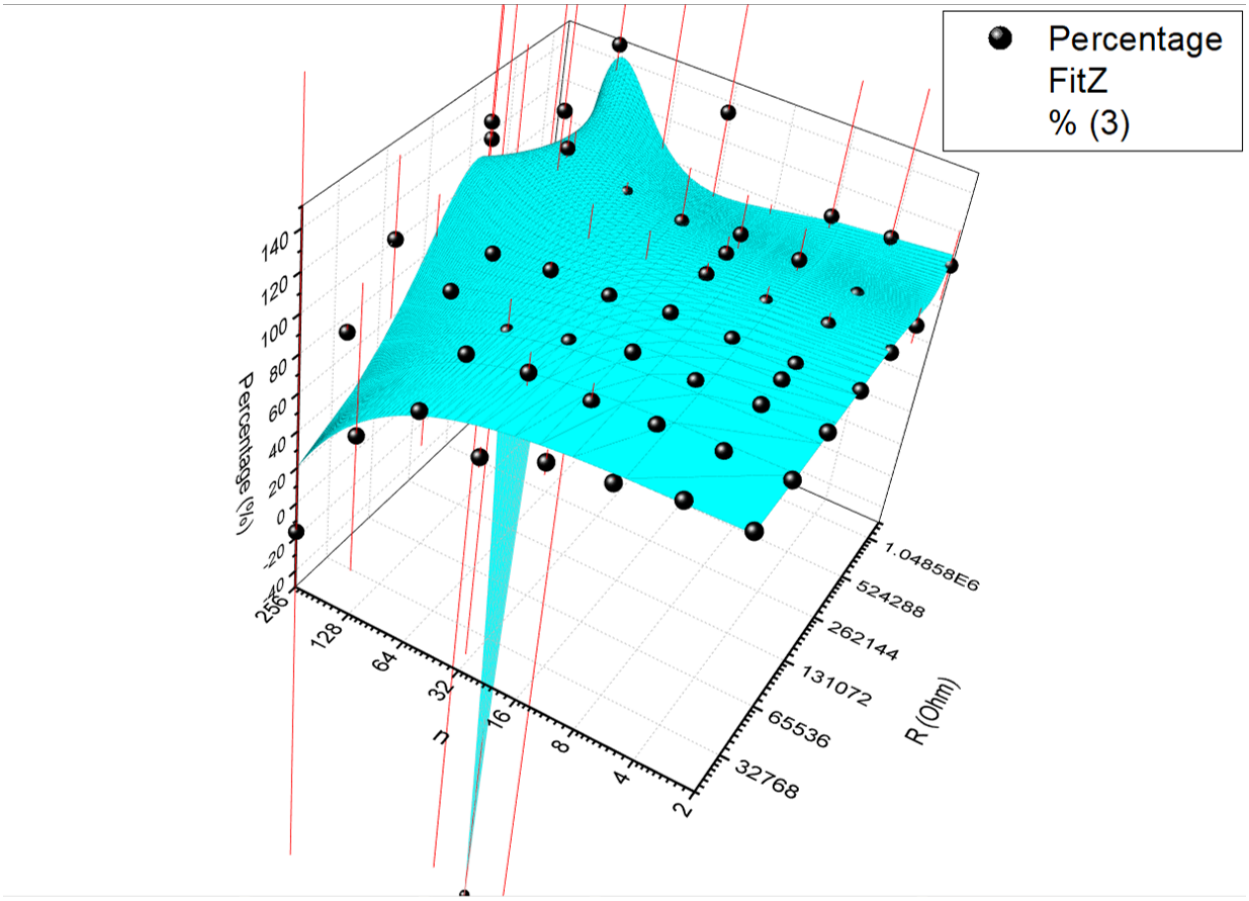


Figure 16: Optimal R_F value determination for large scale circuit (for parameters data, see Appendix Table 8)

From Equation 1, the full equation for a single neuron output is:

$$V_{\text{out}} = -\frac{R_{F2}}{R_{\text{out}'}}\left(-R_{F1}\left(\frac{V_1}{R_1\,|R_F|} + \frac{V_2}{R_2\,|R_F|} + \frac{V_3}{R_3\,|R_F|} + \ldots + \frac{V_n}{R_n\,|R_F|}\right)\right)$$

We could have set $R_{\text{out}'}$, R_{F2}, R_{F1} to 1Ω but this is not possible due to practical challenges (see challenges section), so an $|R_F|$ magnitude is add to every R_n such that:

$$R = |R_{\text{out}'}| = |R_{F2}| = |R_{F1}| = |R_F| \tag{3}$$

to maintain the same Equation 2 and thus the physical $R_{\text{phy}} = R_n\,|R_F|$.

In a matrix form for m neuron outputs, we have:

$$\begin{bmatrix} V_{o1} \\ V_{o2} \\ \vdots \\ V_{om} \end{bmatrix} = -R_{F2} \begin{bmatrix} \frac{1}{R_{out'}} & & \\ & \ddots & \\ & & \frac{1}{R_{out'}} \end{bmatrix} \begin{bmatrix} -R_{F1} \begin{bmatrix} \frac{1}{R_{11}|R_F|} & \frac{1}{R_{21}|R_F|} & \cdots & \frac{1}{R_{n1}|R_F|} \\ \frac{1}{R_{12}|R_F|} & \frac{1}{R_{22}|R_F|} & \cdots & \frac{1}{R_{n2}|R_F|} \\ \vdots & \vdots & & \vdots \\ \frac{1}{R_{1m}|R_F|} & \frac{1}{R_{2m}|R_F|} & \cdots & \frac{1}{R_{nm}|R_F|} \end{bmatrix} \end{bmatrix} \begin{bmatrix} V_{i1} \\ V_{i2} \\ \vdots \\ V_{in} \end{bmatrix}$$

In Figure 16, average percentage of measured V_{out} value is plotted against n number of input V_{in} and magnitude R. When n and R are small, percentage average is predictable, as we can see the surface plot fit nicely, but towards the extreme ends, the values get unpredictable as the errors got out of hand (see Challenges section), as the limit of scaling is reached.

3.2 2nd version

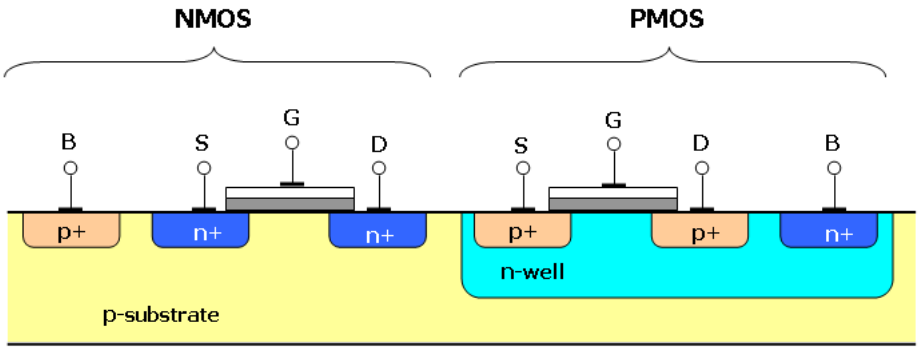


Figure 17: Complementary metal oxide semiconductor (CMOS)

Complementary metal oxide semiconductor (CMOS) is involved as we are looking for voltage-controlled switches to isolate and program physical resistances R_{phy}. We assume the resistances are programmable when they are isolated and extreme voltages (outside of normal operating range) are applied and caused a physical lasting change in their resistances. There are a few similar types of resistive memory pertaining to our assumption here (conductive-bridging RAM (CBRAM), and phase-change memory, etc.) but they are out of our scope of discussion and we will not explore them here.

In Figure 17, we have n channel MOSFET (NMOS) and p channel MOSFET (PMOS). These 2 MOSFETs, together are known as CMOS. PMOS sit in n-well, with p-type diffusion on an n-type substrate. For PMOS to operate, source S voltage V_s is held at high, gate G voltage V_g lower than V_s, and drain voltage at the lowest among all. When $V_g = V_s$, PMOS does not operate. However, when we lower V_g to a point where $V_{gs} \leq V_t$ (V_{gs} being negative and V_t is the threshold voltage), PMOS switches on (p channel is formed from source to drain and current flow with holes as the majority carrier).

Similarly and yet being different, NMOS operation involves source S voltage V_s held at low, gate G voltage V_g higher than V_s, and drain voltage at the highest among all. When $V_g = V_s$, NMOS is at the cut off region. However, when we increase V_g to a point where $V_{gs} \geq V_t$, NMOS switches on (n channel is formed from source to drain and current flow with electrons as the majority carrier).

Both NMOS and PMOS body terminal voltage are tied to their source voltage to maintain their V_t.

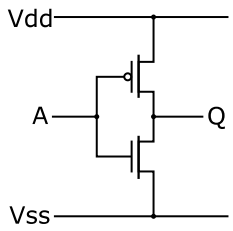


Figure 18: Complementary metal oxide semiconductor (CMOS) inverter with PMOS on top and NMOS below

A simplest device by PMOS and NMOS is the inverter. PMOS is marked with a hollow circle above whereas NMOS is situated below. A is the input and Q is the output. V_{dd} is held at high voltage and V_{ss} is held at low voltage. When input A is high (V_{dd}), PMOS is off and NMOS is on, hence Q is low (V_{ss}). Conversely, if input A is low (V_{ss}), NMOS is off and PMOS is on, hence Q is high (V_{dd}). Hence, we are assuming ideal short circuit on and off behaviour of CMOS.

However, practically, PMOS and NMOS have finite resistance. Their ADE measured on resistance is at $k\Omega$ range, and their off resistance is at $G\Omega$ range. For domain wall based MRAM resistance range of ($1k\Omega$ to $3k\Omega$), we have trouble differentiating the MRAM resistance and CMOS on resistance, thus they affects our result negatively.

MOSFET Type	On Resistance (Ω)	Off Resistance (Ω)
pch_18	3.43k	1.316G
pch	2.32k	24G
nch	1.2k	3.44G
nch_18	1.5k	785.8M

Table 1: Table of MOSFET type supplied by process design kit (PDK) from foundries and their on/off resistance

PMOS and NMOS based design were tested as shown in Figure 19 (only showing PMOS). Circuit execution wise is the same as that of the 1st version. The only difference is the circuit weights are made programmable through isolation.

In MOSFET execution mode, all MOSFETs are set to on, such that all the programmable resistors (weights) array is connected to the summing opamps neuron. In programming mode, resistors are programmed row by row because we have $V_{\text{en\_n}}$, wires routed to the MOSFETs in series with the resistors row by row. MOSFETs of the current row of interest are all turned on, while the remaining are turned off, thus isolating all other rows from the programming action. V_{in} or $V_{\text{i\_n}}$ pins on the left of the row of interest is set as reference voltage whereas $V_{\text{in,set}}$ or $V_{\text{is\_n}}$ (from the top) of the individual resistances on that row is set individually.

NMOS performance is better than PMOS, but not by a wide margin. With $n = 10$ number of inputs and $m = 20$ number of neuron outputs and $R = 50k\Omega$ (Condition 3), NMOS circuit performance (average measured percentage in relation to ideal) vs that of PMOS is $80.3 \pm 3.0\%$ vs $67.7 \pm 4.0\%$. One reason is NMOS majority charge carriers (electrons) have higher mobility as compared to that (holes) of PMOS.

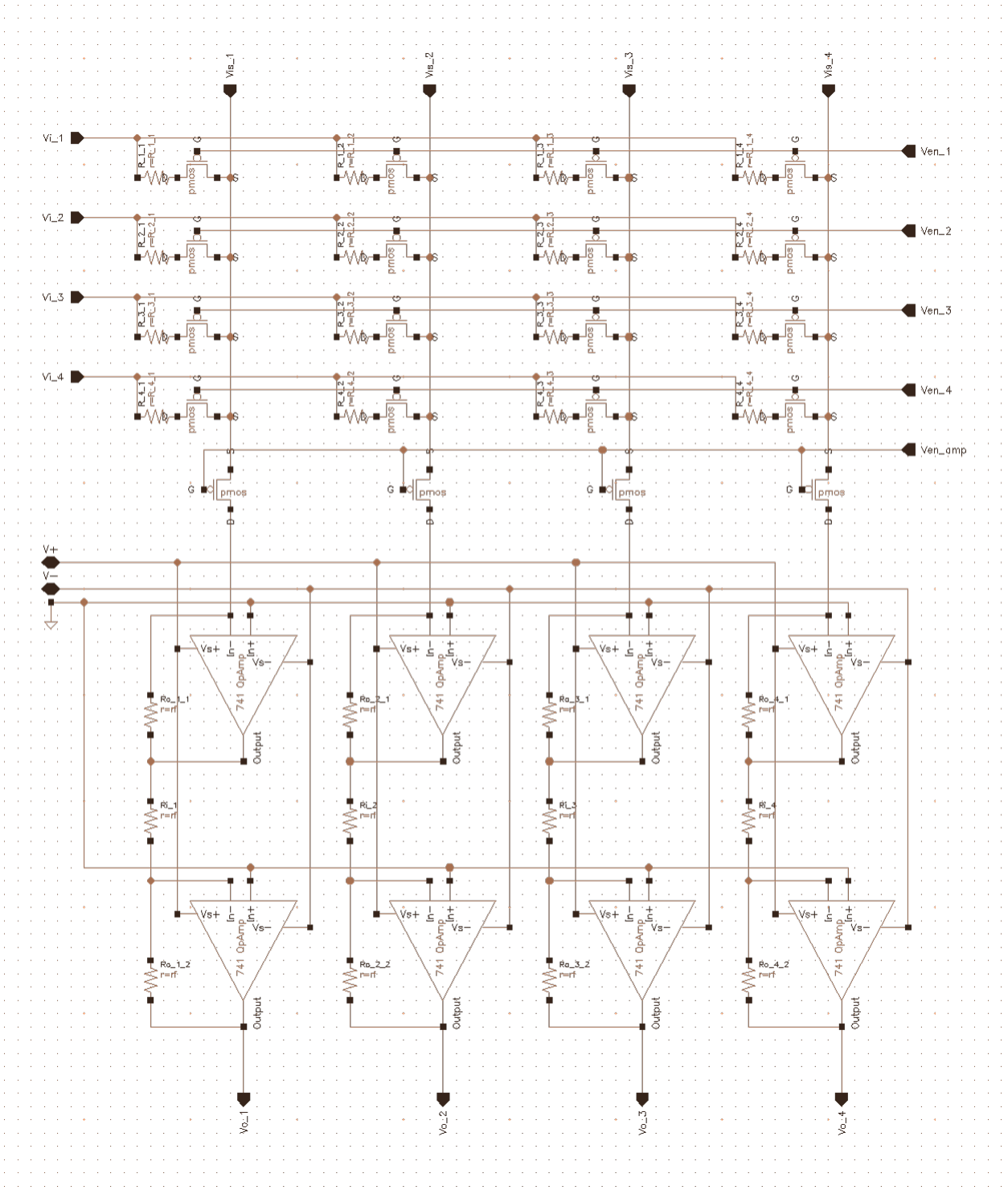


Figure 19: 4 by 4 generated summing neuron with pmos design implemented in Cadence

3.3 3rd and 4th versions

A proposal has been put forward by co-supervisor about mimicking simplified MRAM resistance property (with limited 8 step $1k\Omega - 3k\Omega$), while preserving the actual weight range (also see Case Study section for additional detailed explanation). Resistance and weight relationship in Table 2 is used when designing the 3rd and 4th versions neuromorphic circuit. The

physical resistance r_p is what actually went into the circuit and they represent compressed weight w_c, additional circuit are added to restore the original weight representation (decompressed weight w_d).

Index	Physical Resistance r_p (Ω)	Resistance r (Ω)	Compressed Weight w_c	Decompressed Weight w_d
0	1000	1.0	1.0	1.0
1	1250	1.25	0.8	0.7
2	1500	1.5	0.6667	0.4999
3	1750	1.75	0.5714	0.3571
4	2000	2.0	0.5	0.25
5	2250	2.25	0.4444	0.1667
6	2500	2.5	0.4	0.1000
7	2750	2.75	0.3636	0.0454
8	3000	3.0	0.3333	0

Table 2: Table of resistances and weights

Here, we set $w_{c,0} = 1$ and $r_0 = 1/w_{c,0}$, where the subscript number after comma is the index.
Hence,

$$r_i = \frac{r_{p,i}}{1000}$$

$$w_{c,i} = 1/r_i \tag{4}$$

$$w_{d,i} = \frac{w_{c,i} - w_{c,8}}{w_{c,0} - w_{c,8}} \tag{5}$$

The decompression mechanism for circuit implementation is derived as follows:

We need to achieve:

$$V_{\text{out}} = w_{d,i_1} V_1 + w_{d,i_2} V_2 + ...$$

and we know from:

1. Equation 4:

   Compressed weights $w_{c,i}$ are directly related to resistance r and thus the physical resistances r_p.

2. Equation 5:

   The relationship between compressed weight $w_{c,1}$ and decompressed weight $w_{d,1}$.

Hence,

$$V_{\text{out}} = \frac{w_{c,i_1} - w_{c,8}}{w_{c,0} - w_{c,8}} V_1 + \frac{w_{c,i_2} - w_{c,8}}{w_{c,0} - w_{c,8}} V_2 + ...$$

and

$$V_{\text{out}} = \frac{\frac{1}{r_1} - w_{c,8}}{w_{c,0} - w_{c,8}} V_1 + \frac{\frac{1}{r_2} - w_{c,8}}{w_{c,0} - w_{c,8}} V_2 + ...$$

and finally for a single neuron output:

$$V_{\text{out}} = \frac{1}{w_{c,0} - w_{c,8}} \left(\left(\frac{1}{r_1} V_1 + \frac{1}{r_2} V_2 + ... \right) - (w_{c,8} V_1 + w_{c,8} V_2 + ...) \right) \tag{6}$$

for m neuron outputs:

$$\begin{bmatrix} V_{\text{o1}} \\ V_{\text{o2}} \\ \vdots \\ V_{\text{on}} \end{bmatrix} = \frac{1}{w_{c,0} - w_{c,8}} \left[\begin{bmatrix} \frac{1}{R_{11}} & \frac{1}{R_{21}} & \cdots & \frac{1}{R_{n1}} \\ \frac{1}{R_{12}} & \frac{1}{R_{22}} & \cdots & \frac{1}{R_{n2}} \\ \vdots & \vdots & & \vdots \\ \frac{1}{R_{1m}} & \frac{1}{R_{2m}} & \cdots & \frac{1}{R_{nm}} \end{bmatrix} \begin{bmatrix} V_{\text{i1}} \\ V_{\text{i2}} \\ \vdots \\ V_{\text{in}} \end{bmatrix} - \begin{bmatrix} w_{c,8} & \cdots & w_{c,8} \\ \vdots & & \vdots \\ w_{c,8} & \cdots & w_{c,8} \end{bmatrix} \begin{bmatrix} V_{\text{i1}} \\ V_{\text{i2}} \\ \vdots \\ V_{\text{in}} \end{bmatrix} \right] \tag{7}$$

The term $\left(\frac{1}{r_1} V_1 + \frac{1}{r_2} V_2 + ... \right)$ can be implemented by normal summation using the 1st row opamp, which results in $-\left(\frac{1}{r_1} V_1 + \frac{1}{r_2} V_2 + ... \right)$. By going through a opamp again with a fixed scale factor $\frac{1}{w_{c,0} - w_{c,8}}$, the term $(w_{c,8} V_1 + w_{c,8} V_2 + ...)$ can be sum up separately, together with the previous term, and an inverting sign, yields Equation 6 as needed.

Table 2 was examined further by manipulating the scale factor s of compressed weight i.e. $\left[\frac{1}{3s}, \frac{1}{s} \right]$, with the boundary plotted in Figure 20. When $s = 1$, we obtain w_c boundary $w_{c,0} = 1.0$ and $w_{c,8} = 0.3333$ same as that of Table 2. As scale factor s increases, the boundary tends towards 0 and compressed weight can no longer be represented faithfully. This figure is mainly used in our case study weight processing functions and their influence on our result (see more in that section). Figure 21 shows clearly the inverse relation between compressed weight w_c and resistance r. Ideally, we would want uniform steps in w. However, due to this inverse relationship, what initially uniform in terms of r results in non uniformity of w. This non uniformity of w can distort our represented compressed weight and affect our result negatively.

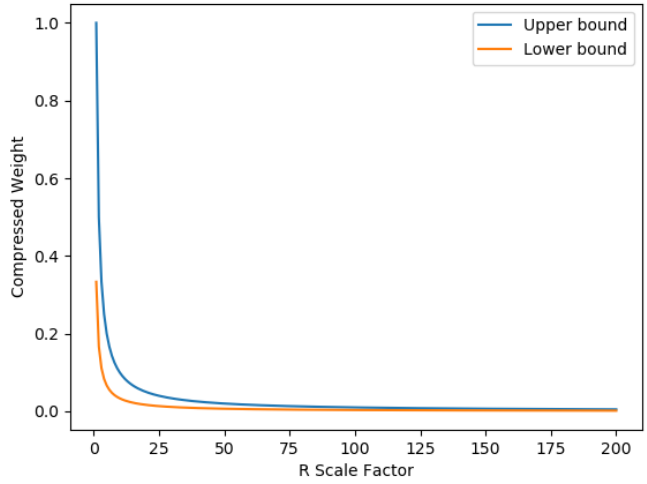


Figure 20: Graph of range of values for compressed weight w_c with scaled R

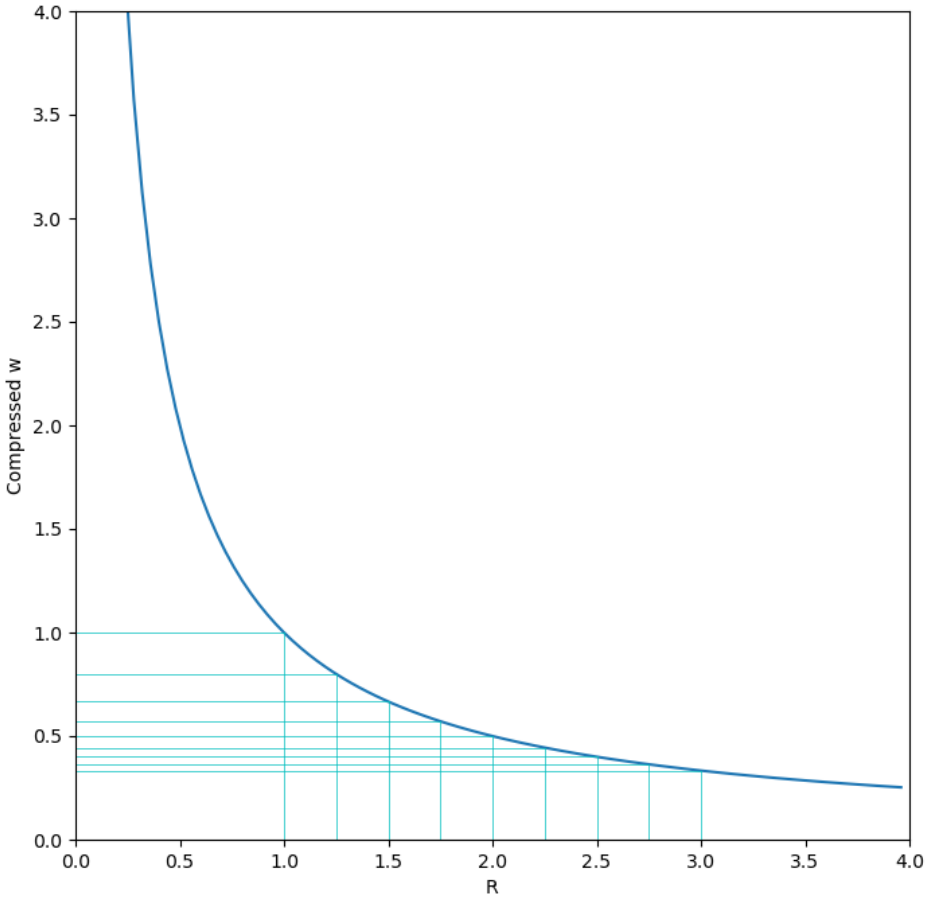


Figure 21: Graph of relationship between uniform R and non-uniform compressed weight

3.3.1 3rd version

A design put forward by D. Kaushik et al [12] showed that programming 8 bit DW MRAM doesn't require isolation because read and write operation in DW MRAM are independent of each other. Hence, the problem encountered in 2nd version where MOSFET on resistance in the range of MRAM resistance range can be side stepped. Hence, all MOSFET are removed.

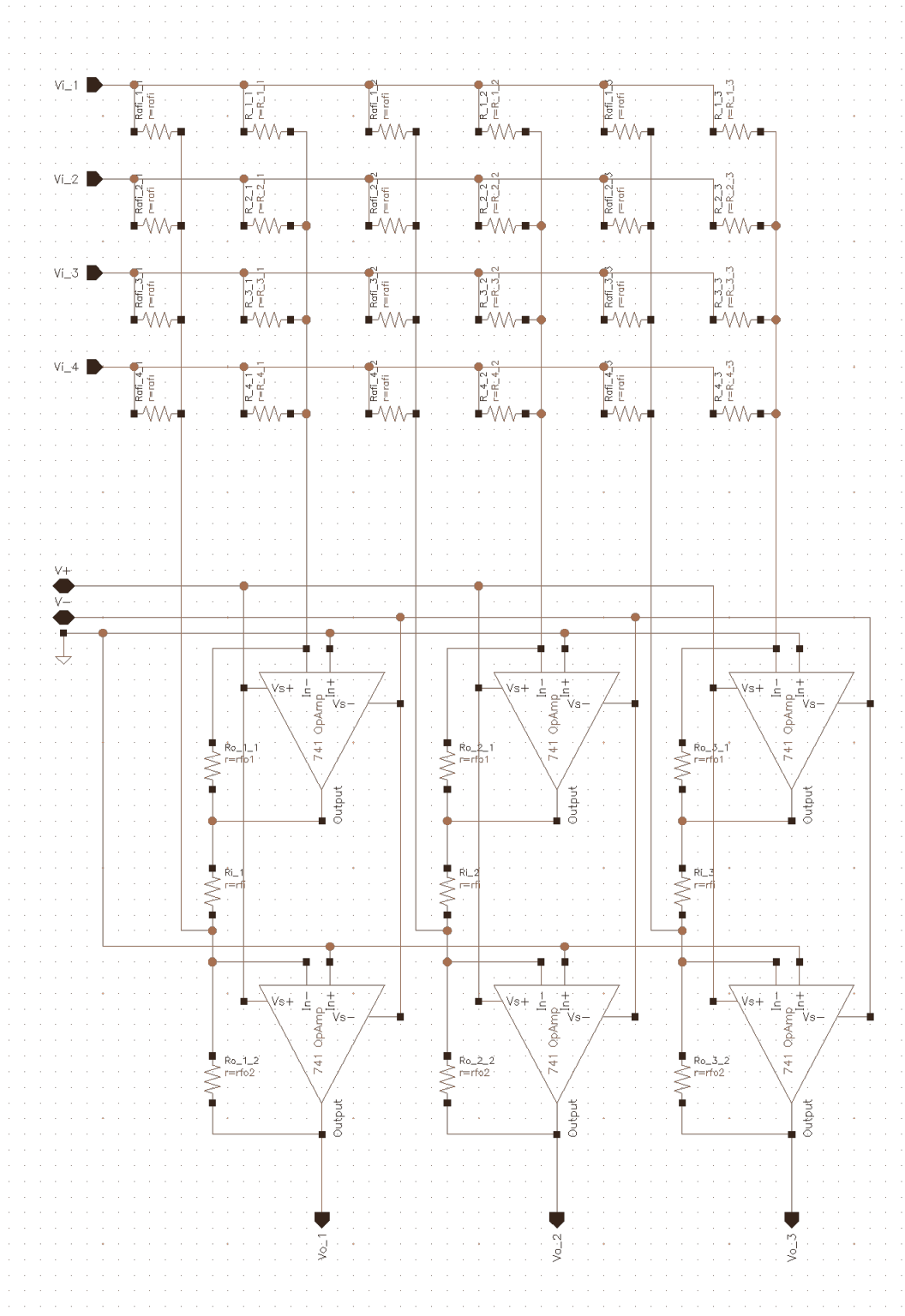


Figure 22: 4 by 4 generated summing neuron with weight decompression circuit implemented in Cadence

3.3.2 4th version

This version design is put forward by co-supervisor due to disappointing performance of very large scale (784 by 20 synapses) implementation of last version. Last version has extremely good performance in closeness in percent relation to ideal, with average scaled up to 100%, and error rate (S.D. $= 2.17\%$) in small scale (20 by 10 synapses) but S.D. $\approx 54\%$ in large scale.

Therefore, the idea was to break up large scale implementation to an equivalent 2 stage small scale implementation to improve the performance. Interestingly, the idea doesn't work, as we still see S.D. $\approx 54\%$ in this version too. For more info, see Challenges section.

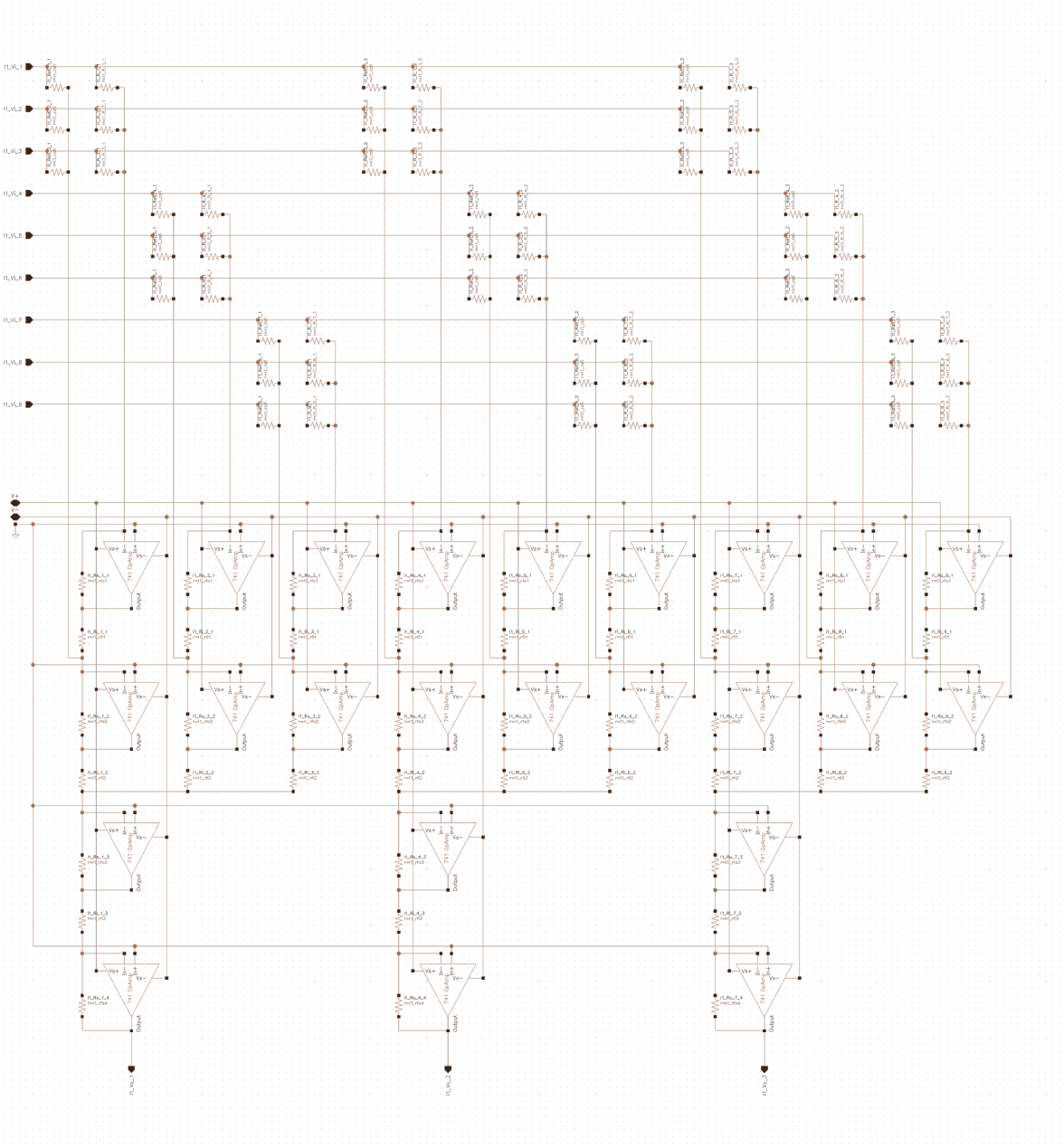


Figure 23: 9 by 3 dual-stage with 3 in 1st stage and weight decompression circuit generated summing neuron implemented in Cadence

4 Challenges

In previous sections of large scale circuit implementation, there are a few challenges, but all boils down to non ideal opamp behaviour. The non ideality characteristics arise in every practical devices, but the opamp implemented before has worse condition. This is especially the case when the internal circuit design of a LM741 opamp (Figure 24) is obtained from Texas Instrument but implemented using TSMC 65nm process design kit (PDK). High precision opamp design are proprietary and therefore not available on the manufacturer's website. The model provided us with 2 null input pins to calibrate opamp

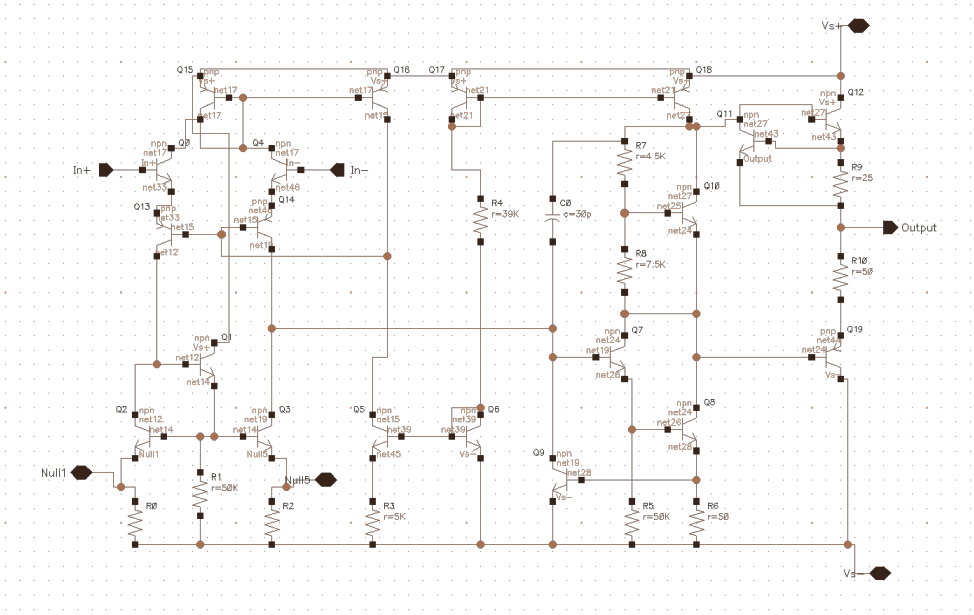


Figure 24: Internal circuitry of TI LM741 operational amplifier (OpAmp)

performance if there are device mismatches (transistor pairs from different vendor, collector and emitter resistors, etc.) due to manufacturing imperfection. In a non ideal case in Figure 11, if $V_+ = V_- = 0$, $V_o = 0$. However, with mismatches, there is a finite V_o. In Figure 25, we can add resistors in parallel with R_0 and R_2 as shwon in Figure 24. These resistors are adjustable, together known as potentiometer as R_0 and R_1 in Figure 25 add up to a fixed resistance. If adjustment is successful, we get $V_o = 0$ as needed. However, this is not the case. It doesn't matter if we swap out the potentiometer for 2 independent resistor, it still does not work as it makes only tiny difference to the output. Figure 26 show the effective resistance we can achieve if we keep adding resistance in parallel with R_0 and R_2 in Figure 24. We observe that the limit of $1k\Omega$ (resistance of R_0 and R_2 where $R_0 = R_2 = 1k\Omega$) is reached when parallel resistance tends to infinity.

The formula for the graph plot is:

$$R_{\text{effective}} = \frac{R_{//} \times R}{R_{//} + R}$$

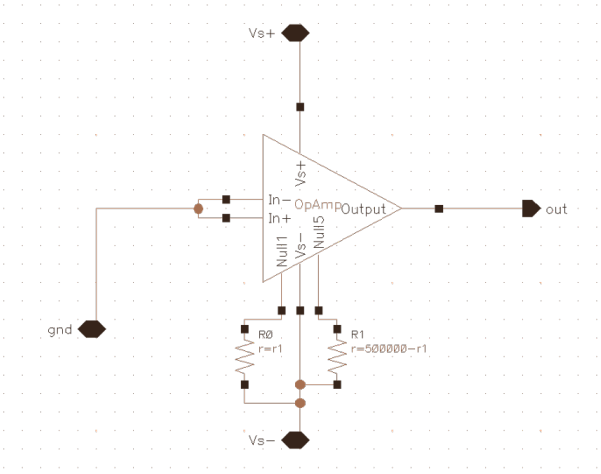


Figure 25: TI LM741 OpAmp null offset

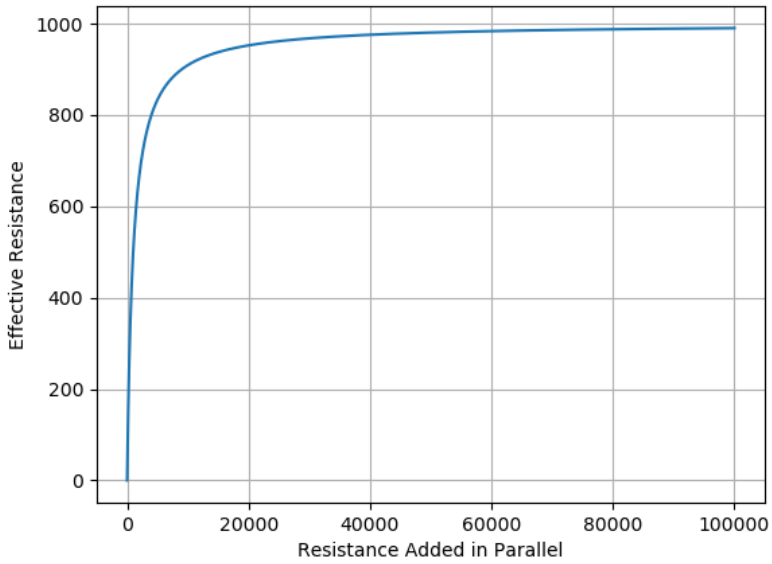


Figure 26: Graph of effective resistance vs resistance added in parallel with existing resistance $R = 1\text{k}\Omega$

Out-of-the-box thinking was required for opamp to work. Hence, both 2 null input pins were removed and the 2 $1k\Omega$ resistors in Figure 24 were tuned independently and it worked. The final resistance setting for those 2 resistors are $1k\Omega$ and $1.138M\Omega$ for left and right resistors respectively.

In Figure 27, we want to identify optimal operation condition of opamp with minimal error. Initially, we fixed $R_0 = R_1$ as shown in Figure 28 as increased their value in tandem. In an ideal opamp summing application, a single V_{in} should yield $V_{\text{in}} = -V_o$. It works as expected for $R_1 = 50k\Omega, R_0 = 50k\Omega$ and $R_1 = 500k\Omega, R_0 = 500k\Omega$ where V_o decreases linearly until it reaches $-V_{\text{sat}}$. However, opamp exhibits erratic behaviour for $R_1 = 2k\Omega, R_0 = 2k\Omega$ where V_o decreases initially before rising again unpredictably. This is suspected as caused by high power consumption. Using $\frac{V_{\text{in}}{}^2}{R}$ and assume virtual ground at $0V$, for the same V_{in}, taking $\frac{V_{\text{in}}{}^2}{2k}$ as reference, for $R_1 = 50k$, the power is reduced by 25 times, and for $R_1 = 500k$, the power is further reduced by 10 times.

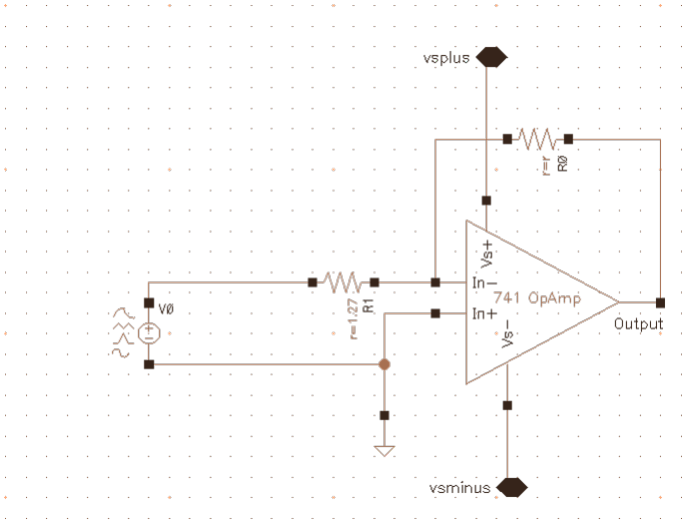
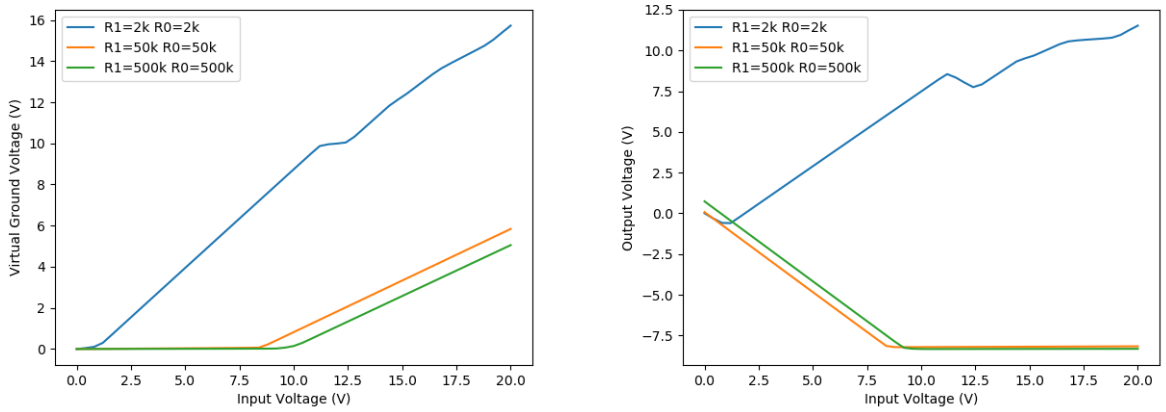


Figure 27: Tuning output resistance R_F (R_0) for input resistance (R_1) between 1kΩ and 3kΩ



(a) virtual ground voltage with different resistor pair

(b) Output voltage with different resistor pair

Figure 28: Non ideal opamp characteristic (based on Figure 27)

For virtual ground voltage, ideally, we expect the voltage to be $0V$ regardless since it is a critical condition for our computation. However, this condition is not met for the high power case ($R_1 = 2k\Omega$,$R_0 = 2k\Omega$). The virtual ground voltage is low initially and when V_{in} increases, virtual ground condition breaks and its voltage increases too.

Table 3 follows that of Figure 27, where different value for R_1 and $R_0(R_F)$ were tested. Range of R_1 is between $1k\Omega$ to $3k\Omega$ and hence only these 2 extreme values of R_1 are examined with different R_F values to find out the optimal setting through varying V_{in}. The best case is having the lowest value for $V_{\text{virtual gnd,min}}$ and $V_{\text{virtual gnd,max}}$ to closely approximate virtual ground condition. When $R_1 = 1k\Omega$, the optimal R_F is almost indiscernible. One notable phenomenon in this table is higher $V_{\text{virtual gnd,max}}$ corresponds to lowest $I_{\text{in,max}}$ and therefore lowest $I_{\text{out,max}}$ due to reduced potential difference between V_{in} and $V_{\text{virtual gnd}}$.

A way to calibrate $V_{\text{virtual gnd}}$ has been discovered though adjusting $+V_{\text{sat}}$ and $-V_{\text{sat}}$. This is possible due to the effect

of Voltage Divider Rule.

In a voltage divider rule, $V_{\text{virtual gnd}}$ is determined by:

$$V_{\text{virtual gnd}} = \frac{\frac{+V_{\text{sat}}}{R_0} + \frac{-V_{\text{sat}}}{R_1}}{\frac{1}{R_0} + \frac{1}{R_1}}$$

In a case with 2 V_{in}, $V_{\text{virtual gnd}}$ becomes:

$$V_{\text{virtual gnd}} = \frac{\frac{+V_{\text{sat}}}{R_0} + \frac{-V_{\text{sat}}}{R_3} + \frac{V_1}{R_1} + \frac{V_2}{R_2}}{\frac{1}{R_0} + \frac{1}{R_3} + \frac{1}{R_1} + \frac{1}{R_2}}$$

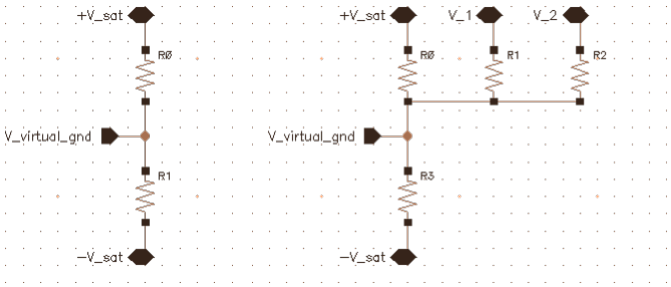
We can expected many more V_{in} for a larger scale circuit. Since we can cannot tune individually the resistors, we stick to tuning $\pm V_{\text{sat}}$. However, in a very large scale circuit where we have many terms, tuning 1 or 2 terms has insignificant effect, which explains why we have huge, similar $S.D.$ error for large scale 3rd version and 4th dual stage version neuromorphic circuit.

All these suboptimal performance (i.e. non-zero $V_{\text{virtual gnd}}$ and low output current) leads to poor gain. To counter this effect, different resistor pertaining to opamp scaling (not resistor array linked to weights) are tuned.

Concretely, Equation 6 is modified to the form:

$$V_{\text{out}} = R_{Fo2}\left(\frac{R_{Fo1}}{R_{Fi}}\left(\frac{1}{|R_F|r_1}V_1 + \frac{1}{|R_F|r_2}V_2 + ...\right) - \left(\frac{1}{R_{aFi}}V_1 + \frac{1}{R_{aFi}}V_2 + ...\right)\right)$$

where $R_{aFi} = \frac{1000s}{w_{c,8}}$, $R_{Fi} = 1000s$, $R_{Fo2} = \frac{1000s}{w_{c,0} - w_{c,8}}$ and s is a scale factor to scale the summing neuron output to ideal value.



(a) Virtual ground voltage (b) Virtual ground voltage with many V_{in}

Figure 29: Voltage Divider Rule

Simulation time is also one of the challenges. Simulation time grow proportionally with network size. There are 10000 test samples in our MNIST test data set (See Case Study Section). For a network size of 10 by 20, the simulation time foreach sample take about 1 second. However, for the same test dataset with a network size of 20 by 784, the simulation time for a sample takes around $2\frac{1}{2}$ hours. So, to tune a parameter and to see its result before proceeding to the next adjustment of the same parameter is very time consuming, not to mention tuning for all other parameters. It is mostly a waiting game. The parameters tuned for a small network is also incompatible for a larger network due to practical non ideality.

R_F(kΩ)	$I_{\text{in,min}}$(A)	$I_{\text{in,max}}$(A)	$I_{\text{out,min}}$(A)	$I_{\text{out,max}}$(A)	$V_{\text{virtual gnd,min}}$(V)	$V_{\text{virtual gnd,max}}$(V)
1	-730.89n	313.18u	-2.2189u	310.71u	730.9u	86.86m
2	-727.74n	312.09u	-2.2157u	309.61u	727.7u	87.91m
4	-721.438n	309.897u	-2.2093u	307.4u	721.4u	90.1m
8	-708.83n	305.39u	-2.1965u	302.85u	708.8u	94.61m
16	-684.22n	295.73u	-2.1716u	293.14u	684.2u	104.3m
32	-636.65n	261.518u	-2.12332u	258.761u	636.7u	138.5m
64	-541.745n	135.729u	-2.03317u	132.789u	547.7u	264.3m
128	-391.155n	70.1832u	-1.87439u	67.2269u	392.1u	329.8m
256	-142.562n	36.8194u	-1.62232u	33.8592u	142.6u	363.2m
512	-194.7n	19.971u	-1.2803u	17.009u	-197.4u	380.0m
1024	566.9n	11.495u	-902.95n	8.5331u	-566.9u	388.5m

(a) $R_1 = 1\text{k}\Omega$

R_F(kΩ)	$I_{\text{in,min}}$(A)	$I_{\text{in,max}}$(A)	$I_{\text{out,min}}$(A)	$I_{\text{out,max}}$(A)	$V_{\text{virtual gnd,min}}$(V)	$V_{\text{virtual gnd,max}}$(V)
1	-257.14n	362.95u	-1.7457u	360.31u	771.4u	111.2m
2	-256.26n	362.14u	-1.7448u	359.49u	768.8u	113.6m
4	-254.47n	360.45u	-1.7429n	357.78u	763.4u	118.6m
8	-251.21n	356.71u	-1.7395u	353.98u	753.6u	129.9m
16	-244.09n	347.28u	-1.7321u	344.46u	732.2u	158.2m
32	-230.374n	236.374u	-1.7178u	233.866u	691.1u	90.88m
64	-203.673n	117.933u	-1.68998u	115.848u	611.0u	46.2m
128	-152.904n	69.0335u	-1.63708u	66.15u	458.7u	192.9m
256	-60.6307n	36.5236u	-1.54094u	33.5749u	181.9u	290.4m
512	94.008n	19.889u	-1.3798u	16.931u	-282.0u	340.3m
1024	321.43n	11.471u	-1.1429u	8.5107u	-964.3u	365.6m

(b) $R_1 = 3\text{k}\Omega$

Table 3: Table of R_1, current I_{in} flowing from R_1 into virtual ground , current I_{out} flowing from virtual ground into R_F and voltage $V_{\text{virtual gnd}}$ of virtual ground (Optimal performance is highlighted in green)

5 Case Study

5.1 MNIST

The neuromorphic network designed earlier on is applied to computer vision application of MNIST (Figure 30) dataset. This dataset has 60000 samples available for training and 10000 samples available for testing. For training wise, TensorFlow python library and Keras framework were used. For computer vision prediction result visualisation as shown in Figure 45, OpenCV library was deployed. These libraries and framework is briefly described in Table 4. Their description are adapted from their official websites.

Figure 30: Modified National Institute of Standards and Technology (MNIST) Handwritten digit database

Tools	Description
TensorFlow	Google's open source library for machine learning training and development. [18]
Keras	TensorFlow's high-level application interface for deep learning models developments. It's used for fast prototyping, state-of-the-art research, and production. [19]
OpenCV	Open Source Computer Vision Library (OpenCV) is an open source computer vision and machine learning software library. OpenCV provides a common infrastructure for computer vision applications and speeds up the commercial use of machine perception. [20]

Table 4: Table of tools of trade

As discussed before, each neuron output is fed to an activation function like shown in Figure 31 and Figure 32. Without an activation function, our network is just a linear regression model. That is, its continuous output is as a result of linear function. Linear functions cannot properly fit complex non-linear dataset. Therefore, we need these non-linear activation function as follows. Although ReLU looks like linear activation function, it is in fact non-linear due to changing gradient

at $x = 0$. For softmax activation function, let say we have 3 neuron with output a, b and c respectively. the final output of neuron 1, 2, and 3 would be $\frac{e^a}{e^a+e^b+e^c}$, $\frac{e^b}{e^a+e^b+e^c}$ and $\frac{e^c}{e^a+e^b+e^c}$ respectively.

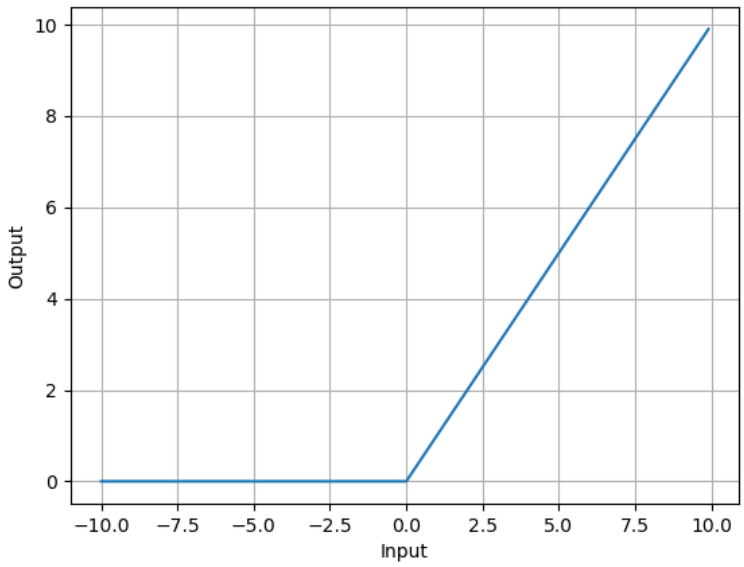


Figure 31: Rectified linear unit (ReLU) activation function

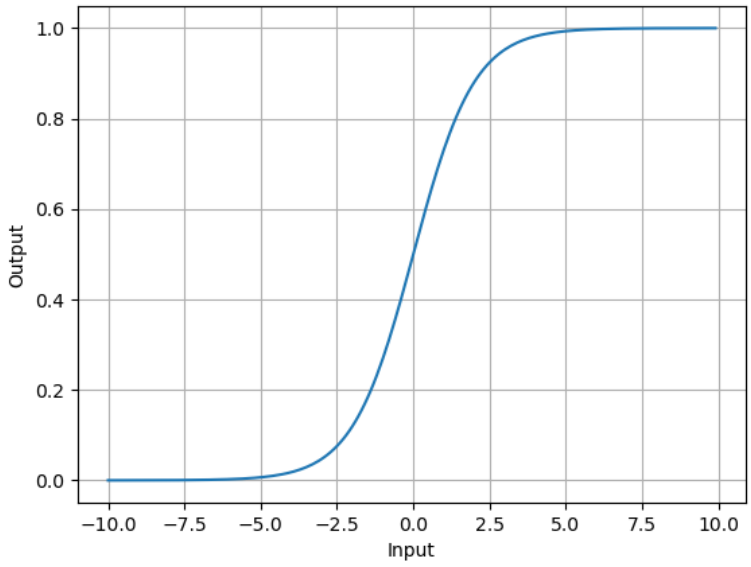


Figure 32: Sigmoid function, a special case of Softmax activation function with 2 outputs and one output set to 0

A few network architecture has been attempted to test the performance of previous activation function (Table 5). FCN 100 here means fully connected network, i.e. all input neurons in input layer connect to each of the 100 neuron in the hidden layer. The table shows ReLU activation function is meant for hidden layers with softmax function meant for output

layer. Techniques such as maxpooling (Figure 33) and 2D convolution (Figure 34) are used. Maxpooling reduces matrix

Network Architecture	Accuracy
Input → FCN 100 → output	11.78%
Input → FCN 100 with ReLU activation → output	11.35%
Input → FCN 100 with ReLU activation → output with ReLU activation	9.8%
Input → FCN 100 with ReLU activation → output with softmax activation	96.91%
Input → FCN 100 with softmax activation → output with softmax activation	54.97%

Table 5: Table of network architecture with different activation function variation and their accuracies (highest accuracy is highlighted in green)

to smaller size with only the largest values. Furthermore, multi layers of convolution matrix being used to extract different features, i.e. curvature, strokes and etc. It is achieved by summing the multiplication of individual elements between large data matrix and small convolution matrix of the overlapped region. For these 2 techniques, we have a small matrix sliding across a large data matrix to perform operation. Visually, it means sliding from left to right, from 1st on top to last row at the bottom. The sliding motion is in a form of 1 step or few steps at once. For maxpooling, the sliding step size is 2, whereas for 2D convolution matrix the sliding step size is 1.

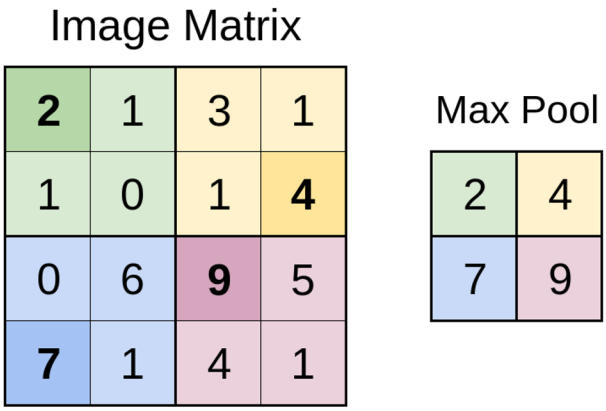
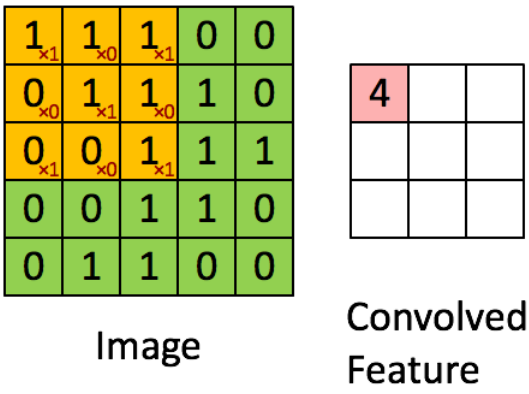
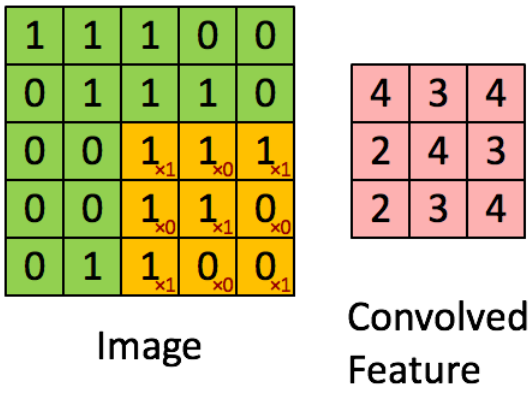


Figure 33: Maxpooling



(a) 2D convolution at the start



(b) 2D convolution at the end

Figure 34: 2D convolution operation

Unlike the architecture of Table 5, a higher performing one involves more than just FCN. It consists of multiple repetitive units of 2D convolution, followed by maxpooling, before it is flatten from 2D matrixes to 1D vector, and finally reaching the FCN and output layer (See Figure 35). Each deeper repetitive unit of 2D convolution and maxpooling extracts subtler features. Let's look at the 2 repetitive units in greater detail. An output of convolution layer is known as feature map. If we have $n1$ convolution matrixes working on a 2D input, there will be $n1$ feature maps, which look like a 3D stack. These feature maps will go through maxpooling to reduce the its 2D dimension, retaining its 3D height. The next convolution layer that has $n2$ convolution matrixes will have each of its matrices working on the 3D stack individually. The way it work on the 3D stack is a slightly modified version of Figure 34. In the figure, we have 1 matrix working on 1 feature map of the previous repetitive unit. Now, we have $n1$ feature map in a form of 3D stack, hence, the matrix will work on the intersected volume ($n1$ height), not just intersected area. Let's say the top most feature map matrix operation of top left area result is 4 as shown in Figure 34, the intersected lower layers we have value $a, b, c, ...$, therefore the output (top left corner unit of a 2D feature map) will be $4 + a + b + c + ...$ ($n1$ terms of them). A complete operation yields a complete 2D feature map. Do note that this is just 1 2D feature map of the 3D matrix operation from 1 of the $n2$ matrices. All $n2$ 3D matrix operations yield $n2$ layers of 3D feature map stack. Flattening will simply be extracting rows by rows of a 3D volume into a 1D vector. The rest of FCN network is already explained in subsection 1.1.3.

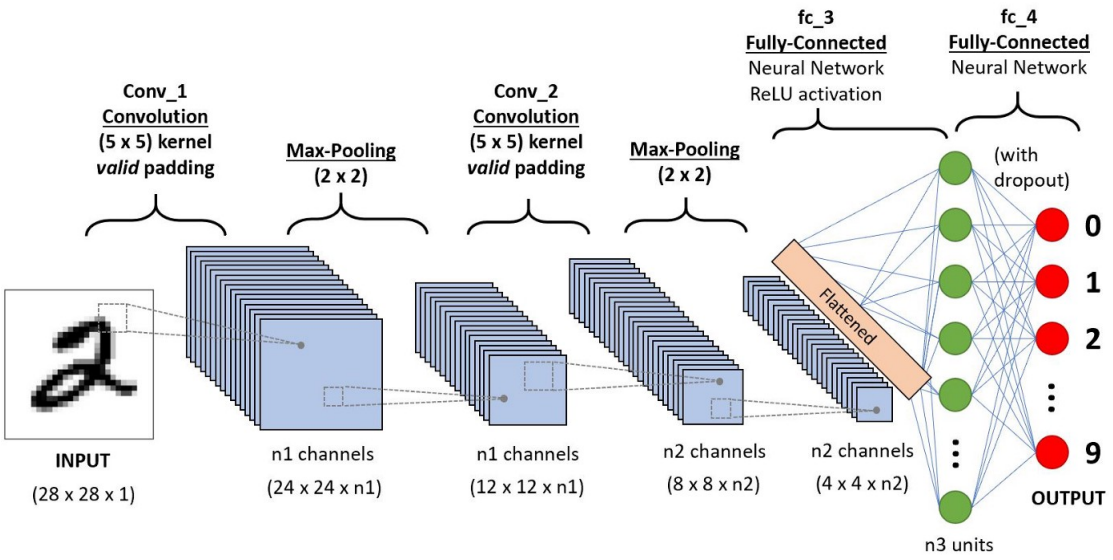


Figure 35: An example convolutional neural network sequence

A few types of network architecture have been attempted to discover the network characteristic by tuning the number of neuron in FCN, introducing convolutional layers, and varying the number of convolutional network matrices.

Abbreviated Title	Full Description
n FCN	Input Layer $\rightarrow$ Flattening Layer $\rightarrow$ n Neurons Fully Connected Layer $\rightarrow$ 10 Neurons Fully Connected Layer $\rightarrow$ Softmax Activation Layer
n Conv Maxpool 10 FCN	Input Layer $\rightarrow$ n 2D Convolutional Layers with 5×5 Kernel Size $\rightarrow$ n ReLU Activation Layers $\rightarrow$ n 2D Max Pooling Layers with 2×2 Window Size & 2 Strides $\rightarrow$ Flattening Layer $\rightarrow$ 10 Neurons Fully Connected Layer $\rightarrow$ Softmax Activation Layer
(20 Conv Maxpool) $\times$2 10 FCN	Input Layer $\rightarrow$ 20 2D Convolutional Layers with 5×5 Kernel Size $\rightarrow$ 20 ReLU Activation Layers $\rightarrow$ 20 2D Max Pooling Layers with 2×2 Window Size & 2 Strides $\rightarrow$ 20 2D Convolutional Layers with 5×5 Kernel Size $\rightarrow$ 20 ReLU Activation Layers $\rightarrow$ 20 2D Max Pooling Layers with 2×2 Window Size & 2 Strides $\rightarrow$ Flattening Layer $\rightarrow$ 10 Neurons Fully Connected Layer $\rightarrow$ Softmax Activation Layer
20 Conv 50 Conv Maxpool 10 FCN	Input Layer $\rightarrow$ 20 2D Convolutional Layers with 5×5 Kernel Size $\rightarrow$ 20 ReLU Activation Layers $\rightarrow$ 20 2D Max Pooling Layers with 2×2 Window Size & 2 Strides $\rightarrow$ 50 2D Convolutional Layers with 5×5 Kernel Size $\rightarrow$ 50 ReLU Activation Layers $\rightarrow$ 50 2D Max Pooling Layers with 2×2 Window Size & 2 Strides $\rightarrow$ Flattening Layer $\rightarrow$ 10 Neurons Fully Connected Layer $\rightarrow$ Softmax Activation Layer

Table 6: Table of legends for TensorFlow trained graphs

A few weight post processing function were used on trained weight in Tensorflow and Keras, to prepare our trained neural network for neuromorphic circuit weight transfer in Cadence. Following are the weight processing function name and their description:

1. step:

Originally, we have discrete 8 steps w_c in Table 2. In Figure 20, we have s (x axis), a scale factor and the corresponding compressed weight value between upper bound and lower bound. This is computed by w_c/s. There is no whatsoever compressing being done here although we talked about those compressed weight values as we are merely using them. With these 8 steps or 9 valued w_c/s, any input w below the lower bound gets clipped to the lower bound value, whereas those above upper bound get clipped to upper value and those within the boundary get rounded to nearest w_c/s.

2. stepUniform:

   Unlike previously where we used non-uniform values from Table 2, we process the input directly. We assumed the input w to be $0 \leq w \leq 1$, for given n steps, or $n+1$ levels with uniform spacing, input w is rounded to one of these $n+1$ levels that $w_{\text{processed}}$ is still $0 \leq w_{\text{processed}} \leq 1$. In the event our assumption of $0 \leq w \leq 1$ is false, where w can be > 1, w is still rounded to one of the uniformly spaced $n+1$ levels with $w_{\text{processed}} > 1$.

3. lim:

   Similar to "step" function before, we still have the upper bound and lower bound with values from w_c which set the clipping value in case our input w exceeds them. The difference here is we have continuous, unprocessed, original input w within the boundary, not rounded to nearest step values.

4. compdecomp:

   There are actual compression and decompression done here. For example, we want to map w, a.k.a. w_d, within $[0,1]$ to w_c within $[a,b]$ for compression, and vice versa for decompression, since decompression is just an inverse function of compression . We can derive the decompression function as follows:

   For boundary value $w_c = b$, $\frac{b-a}{b-a} = 1$ as needed. For boundary value $w_c = a$, we get $\frac{a-a}{b-a} = 0$ as expected. Hence, the actual decompression function is:

   $$f(w_c) = \frac{w_c - a}{b-a}$$

   and compression function is:

   $$f(w_d) = w_d(b-a) + a$$

   For actual compression and decompression, a and b are $\frac{1}{3s}$ and $\frac{1}{s}$ respectively where s is a scale factor like before. Steps taken to implement this function starts with compression to range $[\frac{1}{3s}, \frac{1}{s}]$, "step" function to round off to nearest w_c level within the same range (see "step" function), and finally decompression to scale back to range $[0,1]$.

Figure	Figure Description	Weight Processing Function Used
Figure 38	$0 \leq$ weights ≤ 1 (MRAM 8 steps non-uniform scaling with lossy rounding and capping)	step
Figure 39	$0 \leq$ weights ≤ 1 (Continuous scaling with lossy capping)	lim
Figure 40	$0 \leq$ weights (Uniform scaling with n steps lossy rounding)	stepUniform
Figure 41	$0 \leq$ weights ≤ 1 (MRAM 8 steps non-uniform lossy rounding with lossless compression/decompression)	compdecomp
Figure 42	$0 \leq$ weights ≤ 1 (Continuous scaling with lossy capping)	limit
Figure 43	$0 \leq$ weights ≤ 1 (Uniform scaling with n steps lossy rounding)	stepUniform

Table 7: Table of figures and weight processing function used

Those 6 figures in Table 7 were evaluated in the following pages. First 3 figures have their weights ranging > 1 due to a false assumption of TensorFlow constraint in training that positive weight is never > 1, whereas last 3 figures are all between 0 and 1. There are a few values, falling slightly to ≈ -0.1 (See Figure 36 and these values will be clipped) due to gradual implemention of constraint in training the network so as to achieve optimal learning rate as shown in Figure 6. In general sense, factors that contribute significantly to low network performance are weight clipping and large size of network. The former limits the range of weight that can be represented and therefore distort the trained model (architecture + weight), whereas the latter amplifies the distortion when the network scales and complexity increases. Worst case for all the network accuracy is at around 10%, i.e. the network gets 1/10 correct by getting stuck to a particular digit prediction for all 0-9 digits in uniform distribution. Clipping/Capping is very bad for weight value that fall outside the boundary. Where clipping has failed, weight compression and decompression comes to rescue by flat lining throughout (Figure 41). The flat lining effect is due to the process compression and decompression is lossless and the only lossy process is rounding to nearest step. Nevertheless, the performance still drop for weight processed with "compdecomp" due to larger network size (errors from rounding to nearest step and clipping of weight falling slightly below 0 got amplified). Other than 8 steps weight from MRAM, n steps weight are also experimented so that we can get the optimal n steps for future MRAM without sacrificing too much network performance. Those figures with uniform scaling (Figure 40 and Figure 43) with n steps converge to extremely high accuracy. Another phenomena we observed is a network performance is the lowest for highest scaled R due to a few important and influential (on the outcome) high weight w got filtered out (outside the boundary), leaving many low weight w lying around within the boundary, forming the model.

Finally, 20 FCN model (highlighted in green in Figure 41) was chosen for its attributes such as very reasonably high accuracy of 80.24% and significantly smaller scale and simpler implementation of network, minus away complex convolution and maxpooling function. These 2 functions are good in neural network, but they complicate simple neuromorphic network task of recognizing handwritten digits by demanding for more complex circuit, and asking for more errors.

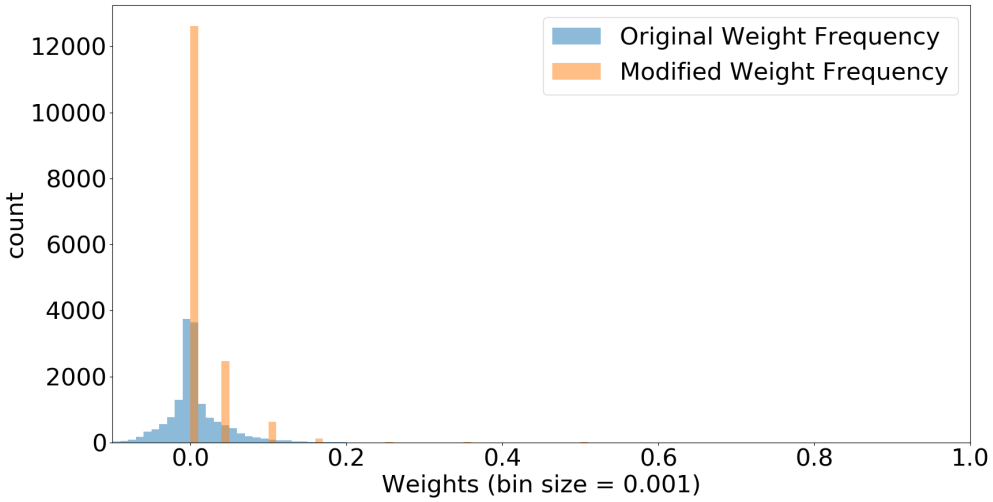


Figure 36: Original and modified (rounded) weight distribution for 20 FCN with "compdecomp" weight processing function

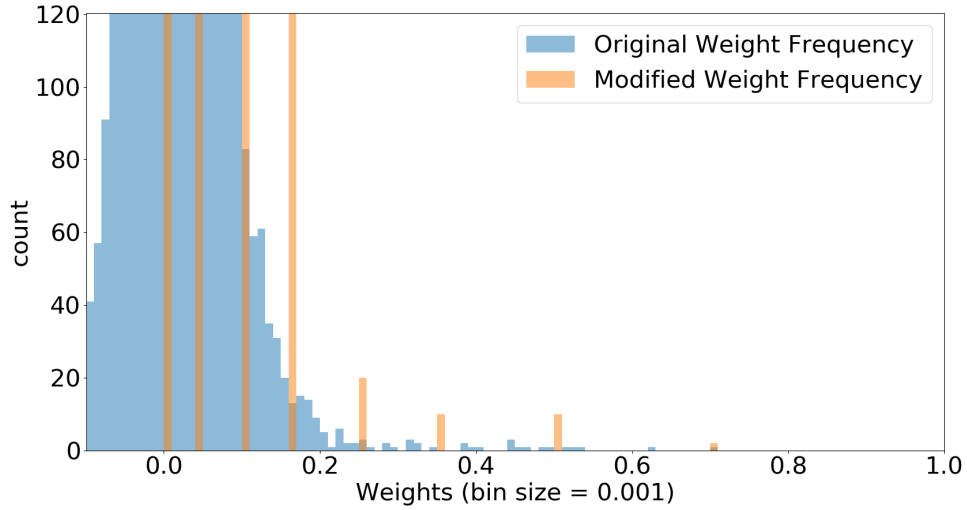
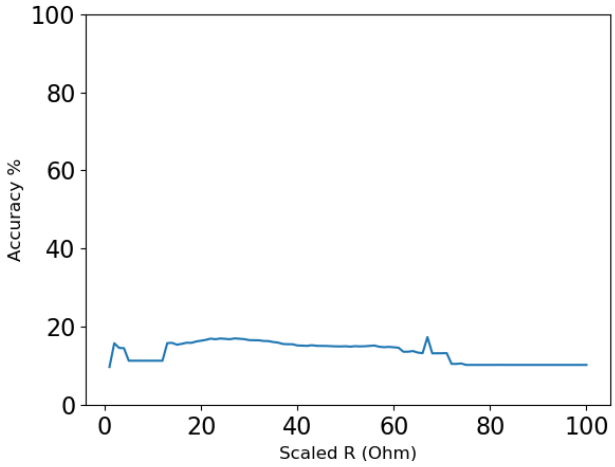
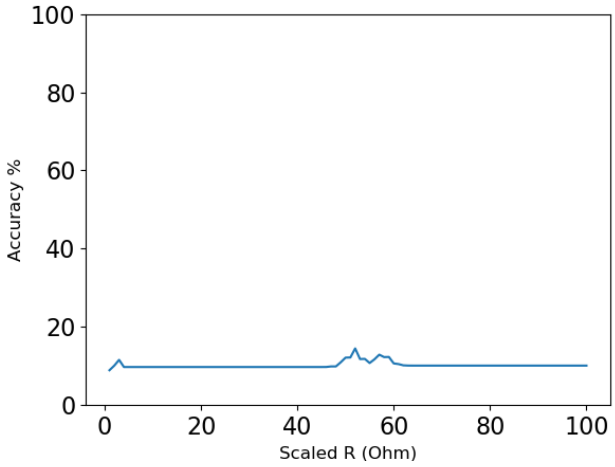


Figure 37: Figure 36 (zoomed). Note those higher but sparser weight distribution towards the right.

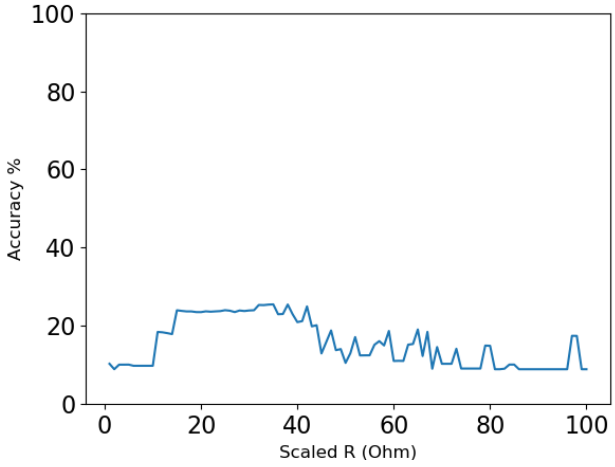
In Figure 36, most weight cluster around 0, with some extending slightly before 0 as discussed earlier. These weight will be clipped to the leftmost modified weight at the extreme end of boundary (orange). It is quite lucky in a sense that the distribution of 8 steps modified weight of mram (orange) closely mimicks the distribution of weight (blue) due to its non linear spacing. By that it means, denser/tighter spacing of modified weight (orange) is observed towards the left end where the lower weight (blue) has the highest distribution towards the end (at 0, ignore those < 0), and increasingly sparser spacing of modified weight (orange) towards the right where the higher/more influential weight (blue) has the lowest distribution (see zoomed Figure 37).



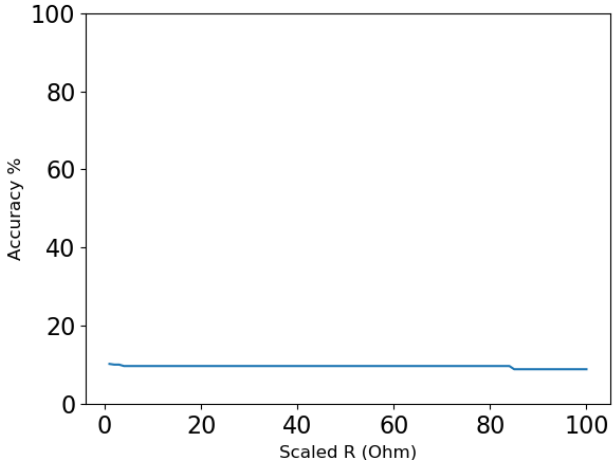
(a) 10 FCN



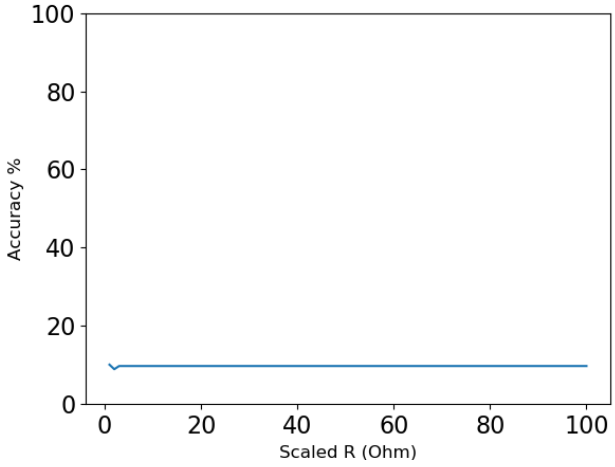
(b) 1000 FCN



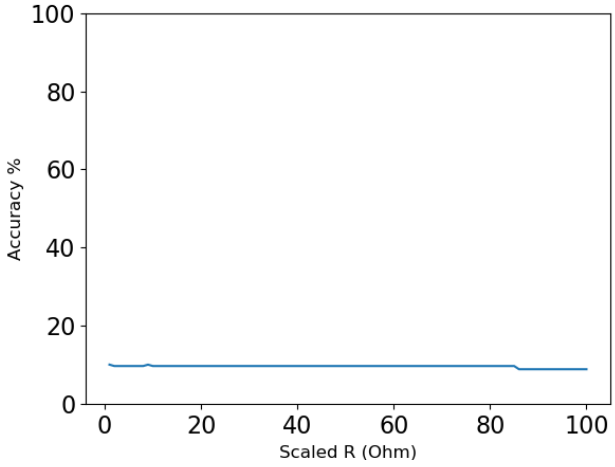
(c) 20 Conv Maxpool 10 FCN



(d) 100 Conv Maxpool 10 FCN

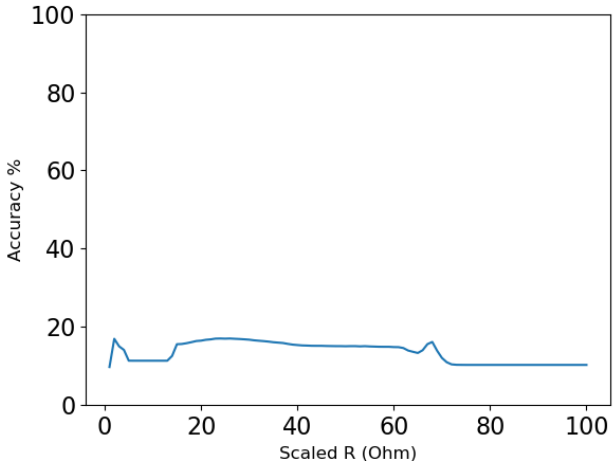


(e) (20 Conv Maxpool)×2 10 FCN

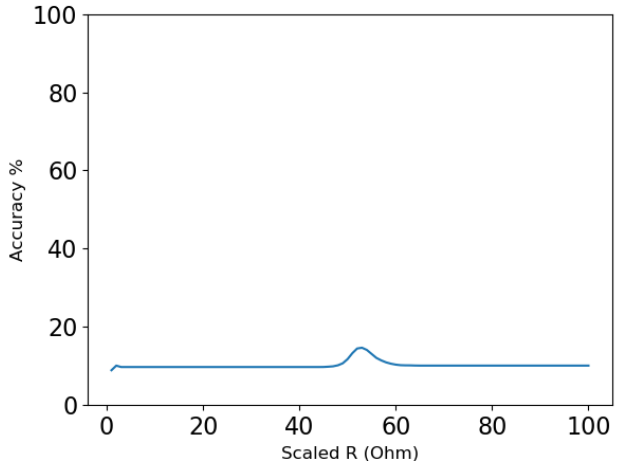


(f) 20 Conv Maxpool 50 Conv Maxpool 10 FCN

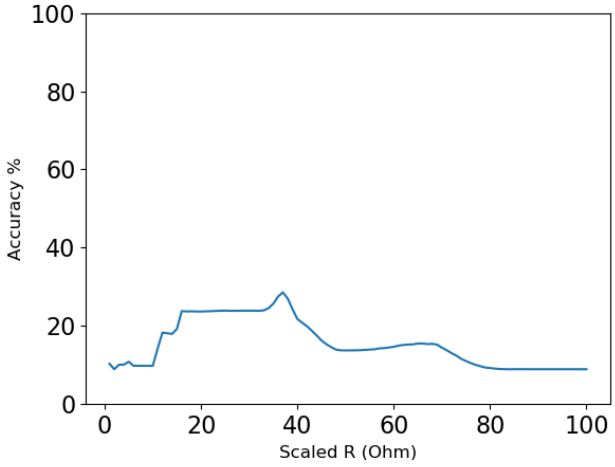
Figure 38: $0 \leq$ weights ≤ 1 (MRAM 8 steps non-uniform scaling with lossy rounding and capping)



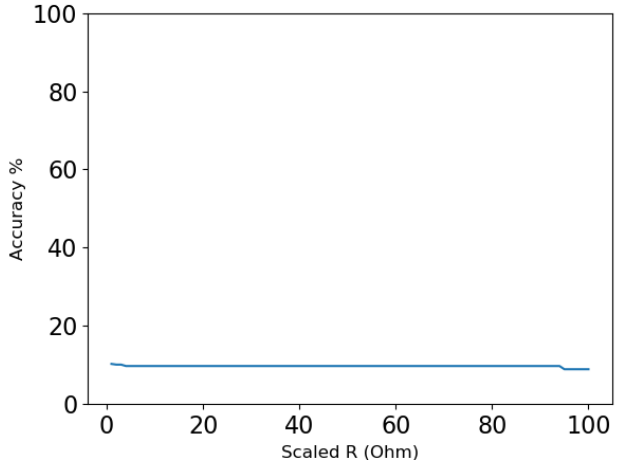
(a) 10 FCN



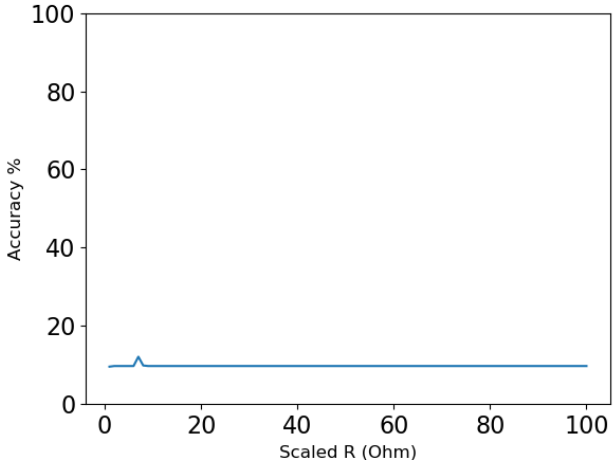
(b) 1000 FCN



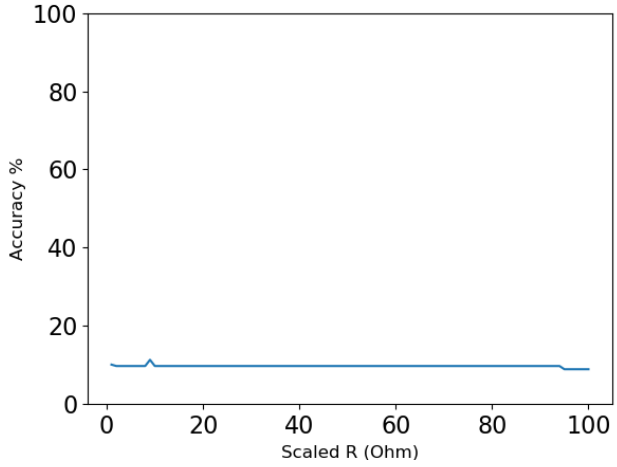
(c) 20 Conv Maxpool 10 FCN



(d) 100 Conv Maxpool 10 FCN

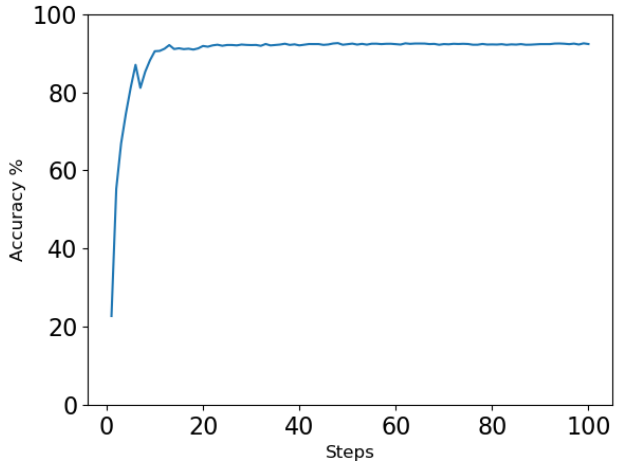


(e) (20 Conv Maxpool)×2 10 FCN

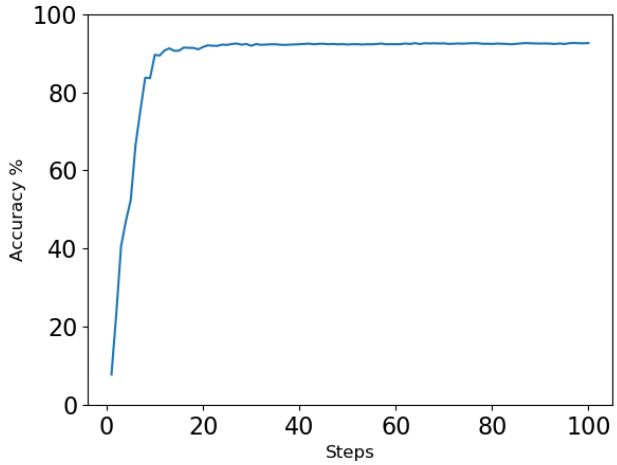


(f) 20 Conv Maxpool 50 Conv Maxpool 10 FCN

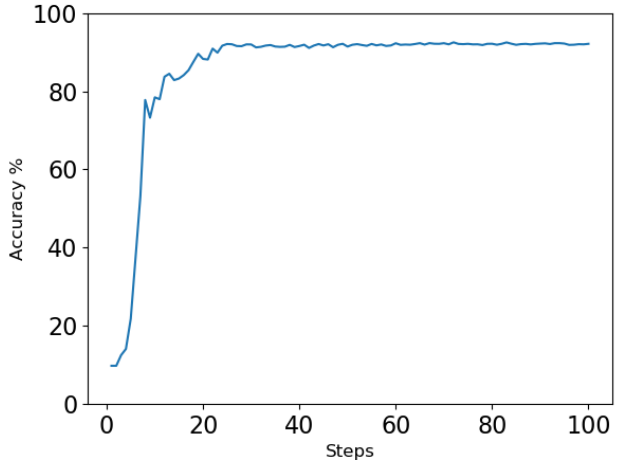
Figure 39: $0 \leq$ weights ≤ 1 (Continuous scaling with lossy capping)



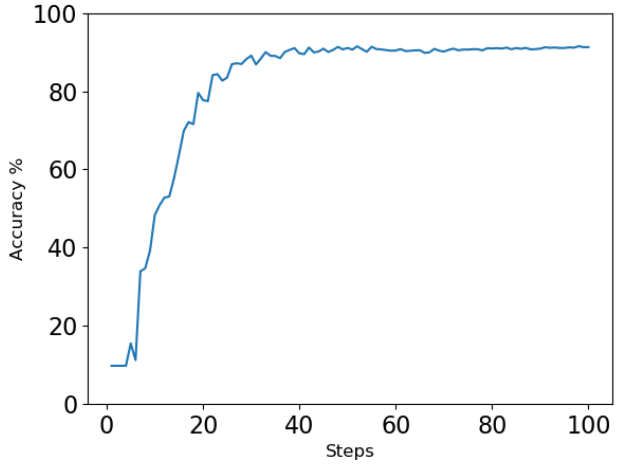
(a) 10 FCN



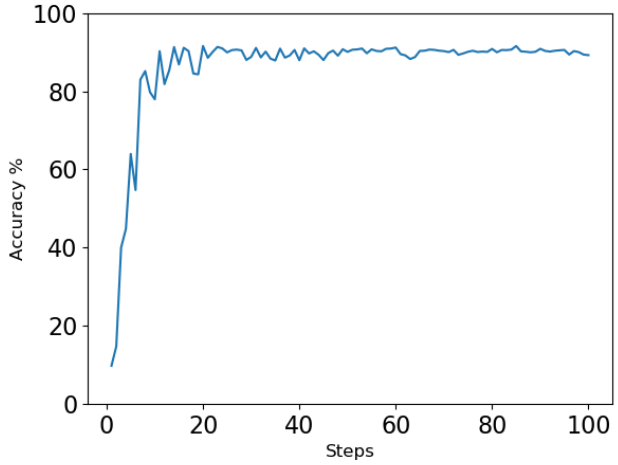
(b) 1000 FCN



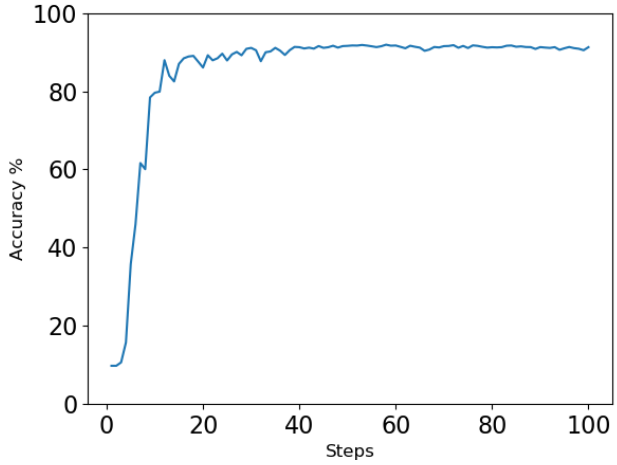
(c) 20 Conv Maxpool 10 FCN



(d) 100 Conv Maxpool 10 FCN

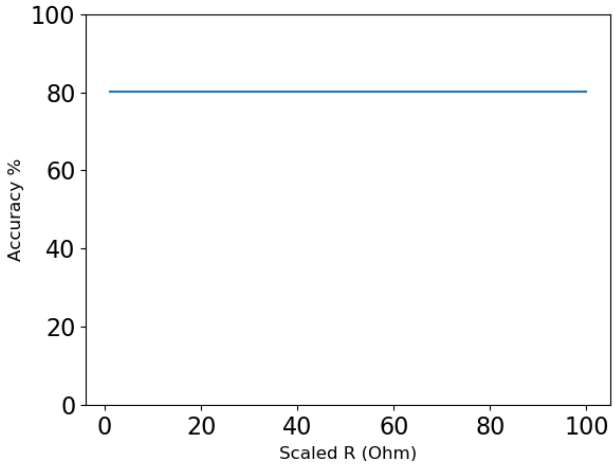


(e) (20 Conv Maxpool)×2 10 FCN

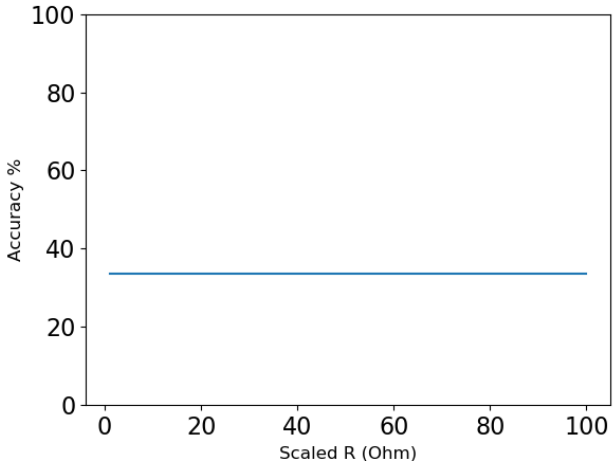


(f) 20 Conv Maxpool 50 Conv Maxpool 10 FCN

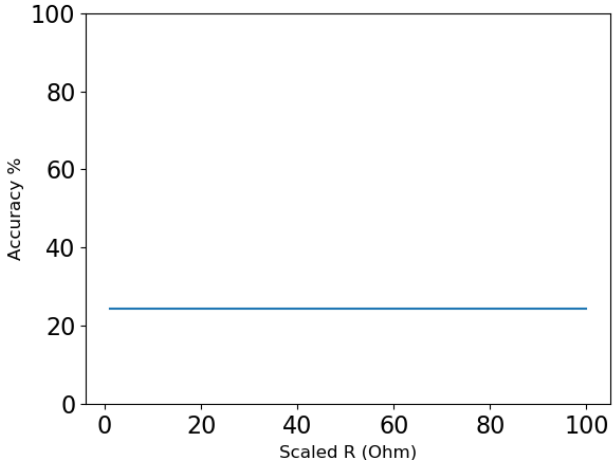
Figure 40: $0 \leq$ weights (Uniform scaling with n steps lossy rounding)



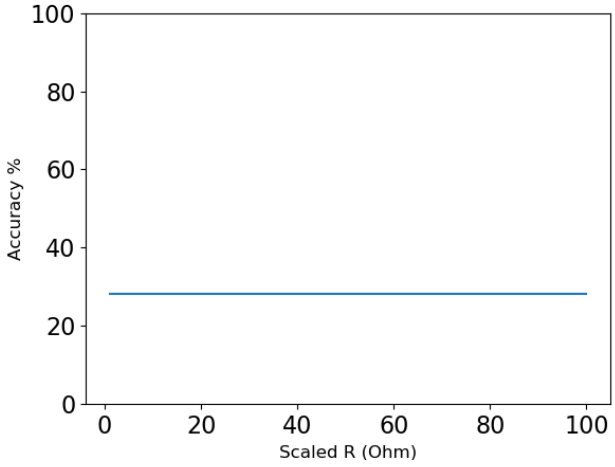
(a) 20 FCN



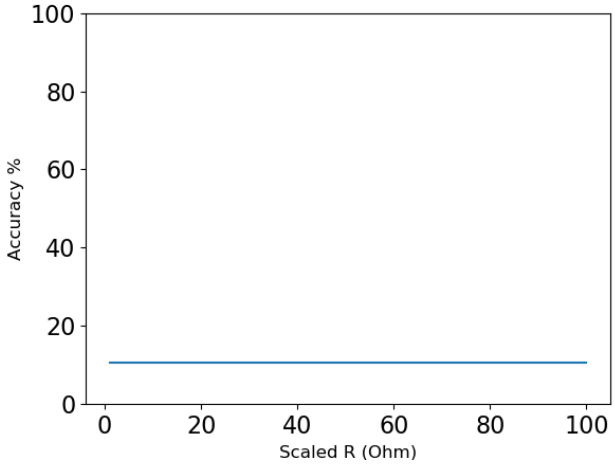
(b) 1000 FCN



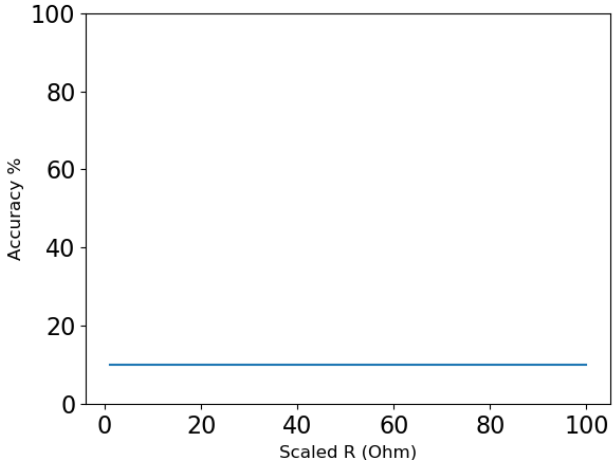
(c) 20 Conv Maxpool 10 FCN



(d) 100 Conv Maxpool 10 FCN

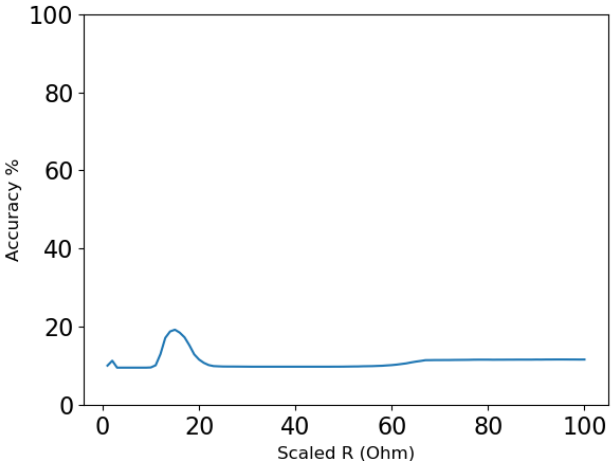


(e) (20 Conv Maxpool)×2 10 FCN

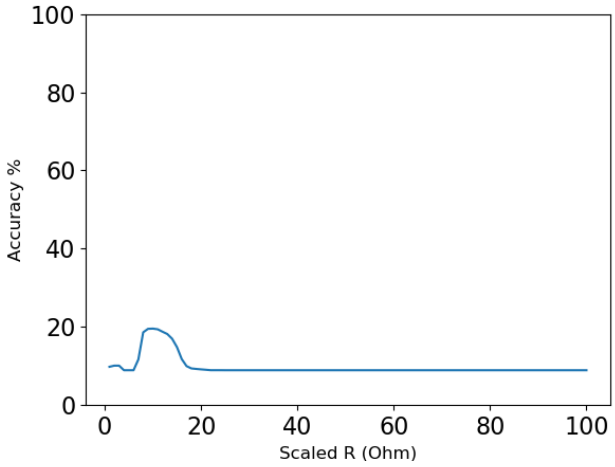


(f) 20 Conv Maxpool 50 Conv Maxpool 10 FCN

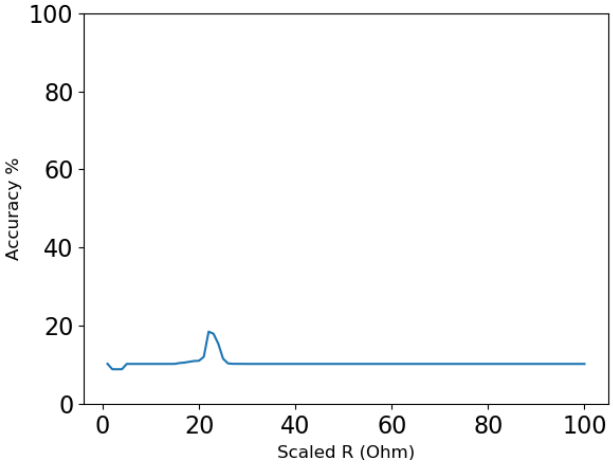
Figure 41: $0 \leq$ weights ≤ 1 (MRAM 8 steps non-uniform lossy rounding with lossless compression/decompression)



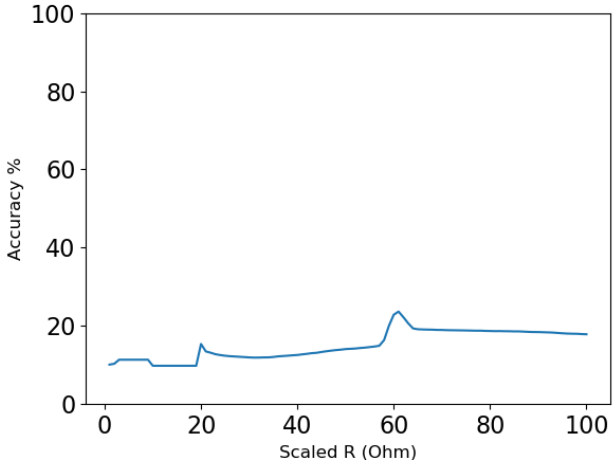
(a) 20 FCN



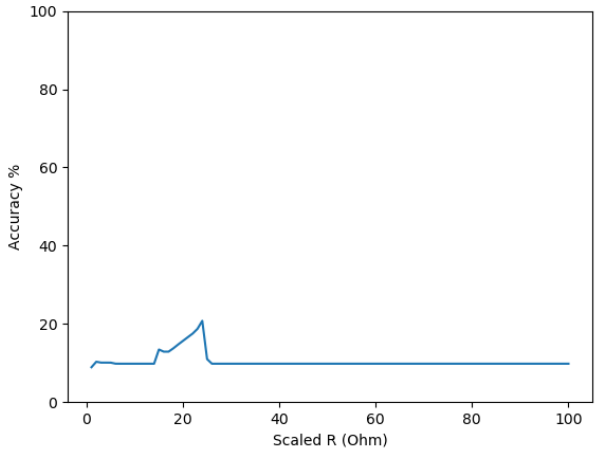
(b) 1000 FCN



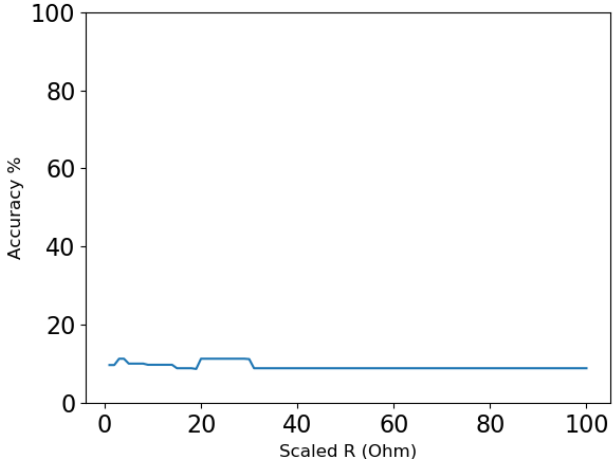
(c) 20 Conv Maxpool 10 FCN



(d) 100 Conv Maxpool 10 FCN

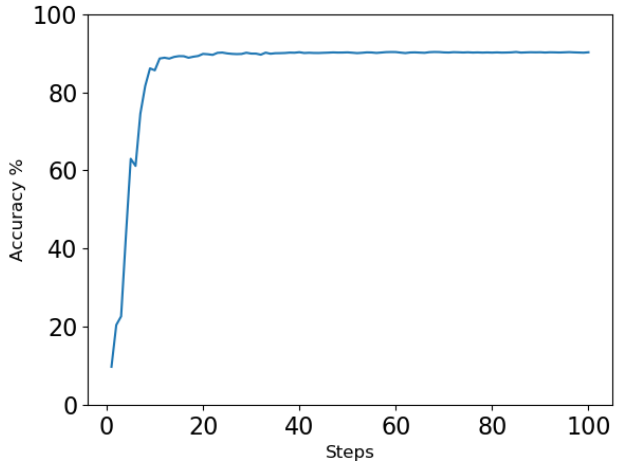


(e) (20 Conv Maxpool)×2 10 FCN

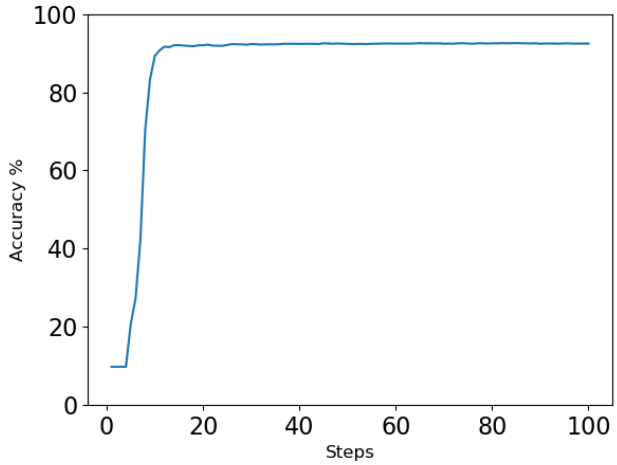


(f) 20 Conv Maxpool 50 Conv Maxpool 10 FCN

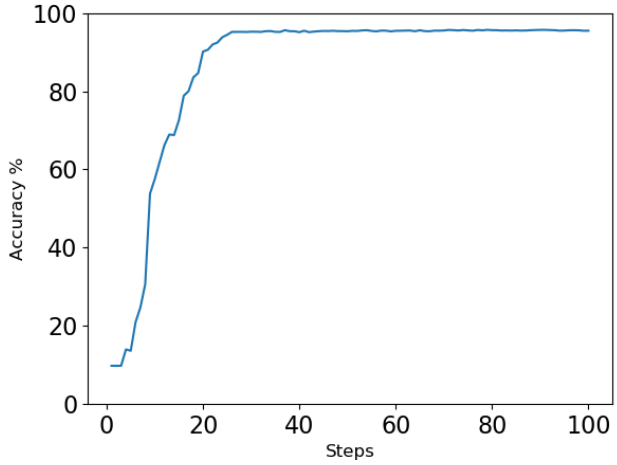
Figure 42: $0 \leq$ weights ≤ 1 (Continuous scaling with lossy capping)



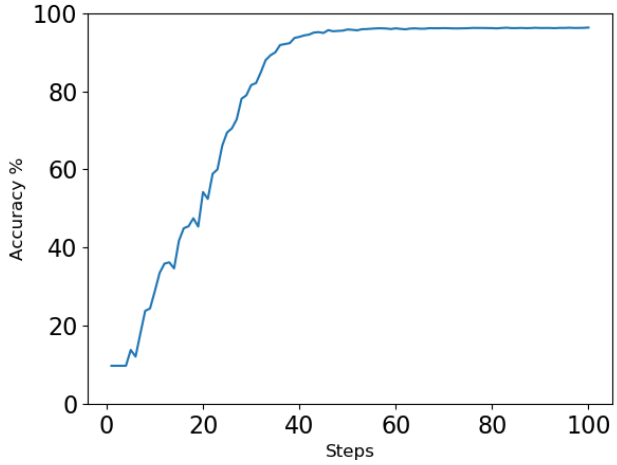
(a) 20 FCN



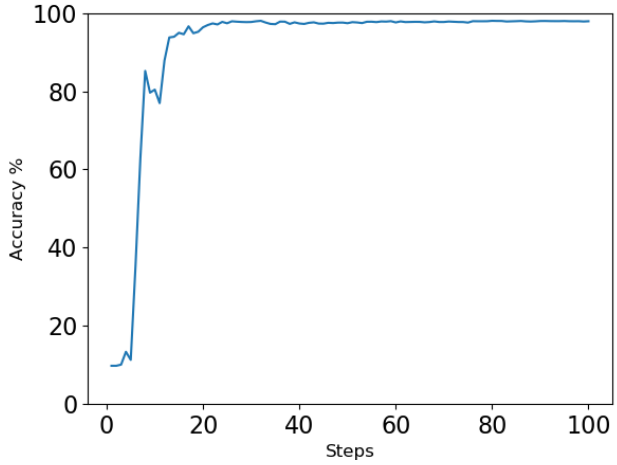
(b) 1000 FCN



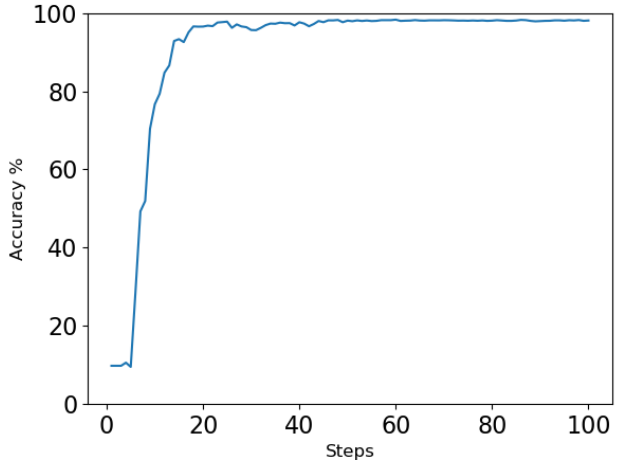
(c) 20 Conv Maxpool 10 FCN



(d) 100 Conv Maxpool 10 FCN



(e) (20 Conv Maxpool)×2 10 FCN



(f) 20 Conv Maxpool 50 Conv Maxpool 10 FCN

Figure 43: $0 \leq$ weights ≤ 1 (Uniform scaling with n steps lossy rounding)

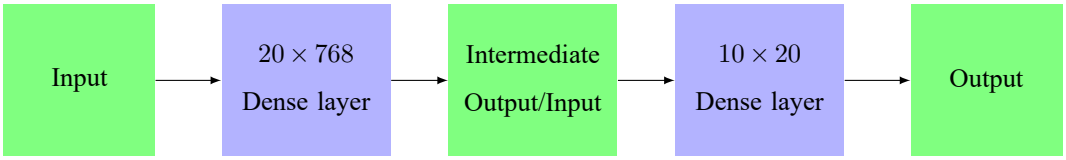


Figure 44: 20 FCN for neuromorphic circuit

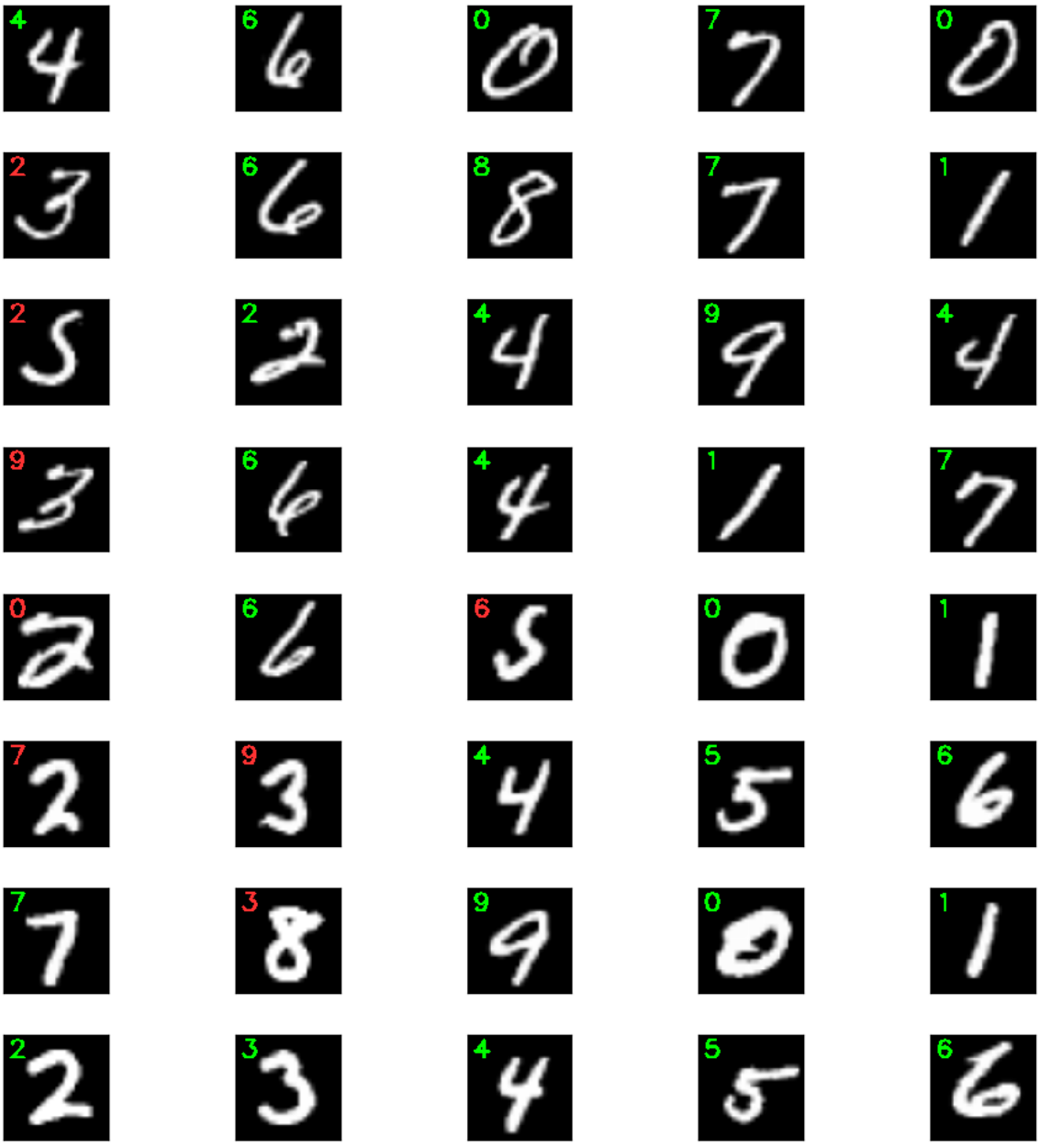


Figure 45: Prediction of last 40 handwritten digits from intermediate output/input of 10000 MNIST test data (predicted in top left corner with true in green and false in red)

In previous chosen 20 FCN model for neuromorphic network implementation, we make the ReLU activation function redundant by processing the weight to $0 \geq w_c \geq 1$, and with our input $0 \geq V_{\text{in}} \geq 1$, all the outputs are naturally > 0, which

are already the same as outputs of a ReLU function even when it is implemented. Softmax function are not implemented in neuromorphic network due to the inherent complexity, so only post processing for Softmax are used.

After taking the architecture of 20 FCN, trained weights of 10×20 dense layer, and intermediate output/input from 20×768 dense layer (Figure 44) from TensorFlow and transferring it to neuromorphic network in Cadence, the simulation on Cadence was run to test the performance of off-chip learning for 1000 MNIST test dataset. OpenCV was used to visualise the last 40 samples of test dataset and their predicted values (Figure 45).

The performance for small scale 10×20 dense layer was astonishing. It achieved an accuracy of 81.02% vs 80.24% predicted by TensorFlow. Rounding errors may be very helpful in improving network performance by chance.

Full simulation of 20 FCN was too conducted with a miserable accuracy of around 10% due to the reasons in Challenges section (section 4) as all outputs are stuck at predicting 0. Basically, access to better performing opamp model is needed to pull the feat off.

6 Conclusion and Future Work

Both the neuromorphic network and spintronics have been gaining traction in the field of artificial intelligence and power-efficient, high-performance memory device. The combination of these two is evolutionary in future development and advancement in the field. This project, therefore, explores the practical side of implementation and migration of off-chip trained data to a neuromorphic circuit, and compare the performance between the two, and highlight the challenges and the shortcomings. Small scale neuromorphic network has been performing phenomenally with MNIST dataset at 81.02% accuracy, sometimes even better than off-chip Tensorflow performance at 80.24%, but large scale performance suffers due to non-ideality of an opamp. Therefore, immediate future work includes sourcing a higher precision opamp or developing an opamp capable of sustaining its virtual ground voltage, by being immune to both supply voltage and varying input voltage. Further work can include completing the simulation of large scale neuromorphic network once the opamp shortcoming is sorted, full domain wall synapse software Cadence simulation model implementation and simulation, and finally exploring the horizon of domain wall spiked based neuromorphic network, encouraged by benefits such as ultra-low power consumption, as pointed out by the reviewed literature.

References

[1] Computer vision solving real-world problems with digital visuals. [Online]. Available: https://www.analyticsinsight.net/computer-vision-solving-real-world-problems-with-digital-visuals/

[2] F. Bre, J. M. Gimenez, and V. D. Fachinotti. (2017, 11) Prediction of wind pressure coefficients on building surfaces using artificial neural networks. [Online]. Available: https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051

[3] Setting the learning rate of your neural network. [Online]. Available: https://www.jeremyjordan.me/nn-learning-rate/

[4] J. H.Byrne, K. S.LaBar, J. E.LeDoux, G. E.Schafe, and R. F.Thompson, *From Molecules to Networks*, 3rd ed. UK: Academic Press, 2014.

[5] D. Hebb, *The Organization of Behavior*. New York: Wiley, 1949.

[6] (2012) The spot breakthrough. [Online]. Available: http://www.spot-research.eu/The-spOt-project/The-spOt-breakthrough

[7] S. Bhatti, R. Sbiaa, A. Hirohata, H. Ohno, S. Fukami, and S. N. Piramanayagam, "Spintronics based random access memory: a review," *Materials Today*, vol. 20, pp. 530–548, 2017.

[8] S. Yu, Y. Wu, R. Jeyasingh, D. Kuzum, and H. . P. Wong, "An electronic synapse device based on metal oxide resistive switching memory for neuromorphic computation," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2729–2737, 2011.

[9] M. Lee, Y. Cui, T. Somu, T. Luo, J. Zhou, W. Tang, W.-F. Wong, and R. Goh, "A system-level simulator for rram-based neuromorphic computing chips," *ACM Transactions on Architecture and Code Optimization*, vol. 15, pp. 1–24, 01 2019.

[10] M. Suri, O. Bichler, D. Querlioz, G. Palma, E. Vianello, D. Vuillaume, C. Gamrat, and B. DeSalvo, "Cbram devices as binary synapses for low-power stochastic neuromorphic systems: Auditory (cochlea) and visual (retina) cognitive processing applications," in *2012 International Electron Devices Meeting*, 2012, pp. 10.3.1–10.3.4.

[11] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Yu Wang, Hao Jiang, M. Barnell, Qing Wu, and Jianhua Yang, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[12] D. Kaushik, U. Sahu, I. Sreedevi, and D. Bhowmik, "Comparing domain wall synapse with other non volatile memory devices for on-chip learning in analog hardware neural network," *AIP Advances*, vol. 10, p. 025111, 02 2020.

[13] J. Grollier, D. Querlioz, K. Y. Camsari, K. Everschor-Sitte, S. Fukami, and M. D. Stiles, "Neuromorphic spintronics," *Nature Electronics*, Mar 2020. [Online]. Available: https://doi.org/10.1038/s41928-019-0360-9

[14] H. F. Langroudi, T. Pandit, M. Indovina, and D. Kudithipudi, "Digital neuromorphic chips for deep learning inference: a comprehensive study," in *Applications of Machine Learning*, M. E. Zelinski, T. M. Taha, J. Howe, A. A. S. Awwal, and K. M. Iftekharuddin, Eds., vol. 11139, International Society for Optics and Photonics. SPIE, 2019, pp. 84 – 95. [Online]. Available: https://doi.org/10.1117/12.2529407

[15] G. Indiveri, F. Corradi, and N. Qiao, "Neuromorphic architectures for spiking deep neural networks," in *2015 IEEE International Electron Devices Meeting (IEDM)*, 2015, pp. 4.2.1–4.2.4.

[16] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," 2016.

[17] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, pp. 54–66, 2014.

[18] (2020) About. [Online]. Available: https://www.tensorflow.org/

[19] Keras. [Online]. Available: https://www.tensorflow.org/guide/keras

[20] (2020) About. [Online]. Available: https://opencv.org/about/

A Appendix

Model	xycube3 (User)		
Equation	$ax^3y^3+bx^3y^2+cx^3y+dx^3+ex^2y^3+fx^2y^2+gx^2y+hx^2+ixy^3+jxy^2$ $+kxy+lx+my^3+ny^2+oy+p$		
Reduced Chi-Sqr	714.846761334228		
Adj. R-Square	0.88555125529964		
		Value	Standard Error
Percentage	a	-3.40118964147559E-23	4.44279966677437E-22
Percentage	b	-3.3808777225304E-17	8.2557808192898E-16
Percentage	c	-2.0259689587371E-12	3.61725948885066E-10
Percentage	d	3.05421829039265E-7	3.11204579295629E-5
Percentage	e	1.92465908629804E-20	1.64731730408421E-19
Percentage	f	-1.55450903885609E-14	3.06407421899329E-13
Percentage	g	1.13182576777577E-8	1.34640724225518E-7
Percentage	h	-3.85131129416287E-4	0.0114965996521702
Percentage	i	-1.48264432713436E-18	1.4260049552801E-17
Percentage	j	1.33599679889964E-12	2.65612549419855E-11
Percentage	k	-1.4137600682012E-7	1.1725381816539E-5
Percentage	l	-0.260480613981356	0.989939319105463
Percentage	m	5.92946586687354E-17	2.14080692056229E-16
Percentage	n	-9.87499996805226E-11	3.98894294313358E-10
Percentage	o	4.46259065749002E-5	1.76439168641693E-4
Percentage	p	93.4097157841867	14.8960306390529

Table 8: Table of user surface fit parameters for R_F