Simulation and programming strategies to mitigate device non-idealities in memristor based neuromorphic systems

by

Pedro Nuno de Jesus Francisco Freitas

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

March 2023

Declaration

The work presented in this thesis was carried out at the School of Engineering, Liverpool John Moores University. Unless otherwise stated, it is the original work of the author.

While registered as a candidate for the degree of Doctor of Philosophy, for which submission is now made, the author has not been registered as a candidate for any other award. This thesis has not been submitted in whole, or in part, for any other degree.

Pedro Freitas
School of Engineering
Liverpool John Moores University
Byrom Street
Liverpool
L3 3AFD
UK

MARCH 2023

Abstract

Since its inception, resistive random access memory (RRAM) has widely been regarded as a promising technology, not only for its potential to revolutionize non-volatile data storage by bridging the speed gap between traditional solid state drives (SSD) and dynamic random access memory (DRAM), but also for the promise it brings to in-memory and neuromorphic computing.

Despite the potential, the design process of RRAM neuromorphic arrays still finds itself in its infancy, as reliability (retention, endurance, programming linearity) and variability (read-to-read, cycle-to-cycle and device-to-device) issues remain major hurdles for the mainstream implementation of these systems.

One of the fundamental stages of neuromorphic design is the simulation stage. In this thesis, a simulation framework for evaluating the impact of RRAM non-idealities on NNs, that emphasizes flexibility and experimentation in NN topology and RRAM programming conditions is coded in MATLAB, making full use of its various toolboxes.

Using these tools as the groundwork, various RRAM non-idealities are comprehensively measured and their impact on both inference and training accuracy of a pattern recognition system based on the MNIST handwritten digits dataset are simulated.

In the inference front, variability originated from different sources (read-to-read and programming-to-programming) are statistically evaluated and modelled for two different device types: filamentary and non-filamentary. Based on these results, the impact of various variability sources on inference are simulated and compared, showing much more pronounced variability in the filamentary device compared to its non-filamentary counterpart. The staged programming scheme is introduced as a method to improve linearity and reduce programming variability, leading to negligible accuracy loss in non-filamentary devices. Random telegraph noise (RTN) remains the major source of read variability in both devices. These results can be explained by the difference in switching mechanisms of both devices.

In training, non-idealities such as conductance stepping and cycle-to-cycle variability are characterized and their impact on the training of NNs based on backpropagation are independently evaluated. Analysing the change of weight distributions during training reveals the different impacts on the SET and RESET processes. Based on these findings, a new selective programming strategy is introduced for the suppression of non-idealities impact on accuracy. Furthermore, the impact of these methods are analysed between different NN topologies, including traditional multi-layer perceptron (MLP) and convolutional neural network (CNN) configurations.

Finally, the new dynamic weight range rescaling methodology is introduced as a way of not only alleviating the constraints imposed in hardware due to the limited conductance range of RRAM in training, but also as way of increasing the flexibility of RRAM based deep synaptic layers to different sets of data.

Main contributions

[1] P. Freitas, Z. Chai, W. Zhang, J. Marsland, P. Lisboa, and J. F. Zhang, "Programming strategies to mitigate the impact of device non-idealities in training synaptic resistive-switching memory based analog neural networks," 2023. [In preparation].

[2] P. Freitas, Z. Chai, W. Zhang, J. Marsland, P. Lisboa, J. F. Zhang, F. Hatem, and K. Jones, "Evaluation of different variabilities in resistive-switching memory devices and their impacts on inference accuracy of synaptic neural network," 2023. [In preparation].

[3] M. E. Pereira, J. Deuermeier, P. Freitas, P. Barquinha, W. Zhang, R. Martins, E. Fortunato, and A. Kiazadeh, "Tailoring the synaptic properties of a-IGZO memristors for artificial deep neural networks," *APL Materials*, vol. 10, p. 011113, jan 2022.

[4] P. Freitas, Z. Chai, W. Zhang, J. F. Zhang, and J. Marsland, "Impact of RTN and Variability on RRAM-Based Neural Network," in *2020 IEEE 15th International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, vol. 1, pp. 1–4, IEEE, nov 2020.

[5] D. Joksas, P. Freitas, Z. Chai, W. H. Ng, M. Buckwell, C. Li, W. D. Zhang, Q. Xia, A. J. Kenyon, and A. Mehonic, "Committee machines – a universal method to deal with non-idealities in memristor-based neural networks," sep 2019.

[6] Z. Chai, P. Freitas, W. Zhang, F. Hatem, J. F. Zhang, J. Marsland, B. Govoreanu, L. Goux, and G. S. Kar, "Impact of RTN on Pattern Recognition Accuracy of RRAM-Based Synaptic Neural Network," *IEEE Electron Device Letters*, vol. 39, pp. 1652–1655, nov 2018.

[7] Z. Chai, W. Zhang, P. Freitas, F. Hatem, J. F. Zhang, J. Marsland, B. Govoreanu, L. Goux, G. S. Kar, S. Hall, P. Chalker, and J. Robertson, "The Over-Reset Phenomenon in Ta 2 O 5 RRAM Device Investigated by the RTN-Based Defect Probing Technique," *IEEE Electron Device Letters*, vol. 39, pp. 955–958, jul 2018.

Other contributions

[1] Z. Hu, W. Zhang, R. Degraeve, D. Garbin, Z. Chai, N. Saxena, P. Freitas, A. Fantini, T. Ravsher, S. Clima, J. F. Zhang, R. Delhougne, L. Goux, and G. Kar, "New Insights of the Switching Process in GeAsTe Ovonic Threshold Switching (OTS) Selectors," *IEEE Transactions on Electron Devices*, pp. 1–7, 2022.

[2] W. Zhang, Z. Chai, P. Freitas, J. F. Zhang, and J. Marsland, "Relaxation in GeSe Ovonic Threshold Switching Device," in *2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, pp. 1–4, IEEE, oct 2022.

[3] Z. Chai, P. Freitas, W. Zhang, J. F. Zhang, and J. Marsland, "True random number generator based on switching probability of volatile Ge X Se 1-X ovonic threshold switching selectors," in *2021 IEEE 14th International Conference on ASIC (ASICON)*, pp. 1–4, IEEE, oct 2021.

[4] Z. Chai, W. Zhang, S. Clima, F. Hatem, R. Degraeve, Q. Diao, J. F. Zhang, P. Freitas, J. Marsland, A. Fantini, D. Garbin, L. Goux, and G. S. Kar, "Cycling Induced Metastable Degradation in GeSe Ovonic Threshold Switching Selector," *IEEE Electron Device Letters*, vol. 42, pp. 1448–1451, oct 2021.

[5] J. Brown, J. F. Zhang, B. Zhou, M. Mehedi, P. Freitas, J. Marsland, and Z. Ji, "Random-telegraph-noise-enabled true random number generator for hardware security," *Scientific Reports*, vol. 10, p. 17210, dec 2020.

[6] Z. Chai, P. Freitas, W. D. Zhang, F. Hatem, R. Degraeve, S. Clima, J. F. Zhang, J. Marsland, A. Fantini, D. Garbin, L. Goux, and G. S. Kar, "Stochastic Computing Based on Volatile GeSe Ovonic Threshold Switching Selectors," *IEEE Electron Device Letters*, vol. 41, pp. 1496–1499, oct 2020.

[7] Z. Chai, P. Freitas, J. Marsland, A. Fantini, D. Garbin, L. Goux, G. S. Kar, W. Shao, W. Zhang, J. Brown, R. Degraeve, F. D. Salim, S. Clima, F. Hatem, and J. F. Zhang, "GeSe-Based Ovonic Threshold Switching Volatile True Random Number Generator," *IEEE Electron Device Letters*, vol. 41, pp. 228–231, feb 2020.

[8] F. Hatem, J. F. Zhang, J. Marsland, P. Freitas, L. Goux, G. S. Kar, Z. Chai, W. Zhang, A. Fantini, R. Degraeve, S. Clima, D. Garbin, J. Robertson, and Y. Guo, "Endurance improvement of more than five orders in Ge x Se 1-x OTS selectors by using a novel refreshing program scheme," in *2019 IEEE International Electron Devices Meeting (IEDM)*, no. 1, pp. 35.2.1–35.2.4, IEEE, dec 2019.

[9] Z. Chai, W. Zhang, R. Degraeve, S. Clima, F. Hatem, J. F. Zhang, P. Freitas, J. Marsland, A. Fantini, D. Garbin, L. Goux, and G. S. Kar, "Dependence of Switching Probability on Operation Conditions in Ge x Se 1–x Ovonic Threshold Switching Selectors," *IEEE Electron Device Letters*, vol. 40, pp. 1269–1272, aug 2019.

[10] Z. Chai, W. Zhang, R. Degraeve, S. Clima, F. Hatem, J. F. Zhang, P. Freitas, J. Marsland, A. Fantini, D. Garbin, L. Goux, and G. S. Kar, "Evidence of filamentary switching and relaxation mechanisms in Ge x Se 1-x OTS selectors," in *2019 Symposium on VLSI Technology*, no. 1, (Kyoto), pp. T238–T239, IEEE, jun 2019.

Acknowledgements

Firstly, I would like to thank my supervisors: prof. Wei Zhang, prof. John Marsland and prof. Paulo Lisboa, for all of the availability and continuous support they have shown me throughout this project.

I would also like to thank the other members (current and past) of the LJMU microelectronics reliability and characterization research group that were alongside me for this journey. They are: Dr Zheng Chai, Dr Firas Hatem, Dr James Brown, Dr Mehzabeen Mehedi, Dr Rui Gao, Mr Zeyu Hu, Dr Nishant Saxena, Mr Kean Tok and Mr Dale Hodgkinson. Special thanks to Dr Zheng Chai, who has provided me with a lot of guidance throughout the project.

Next, I would like to thank the university support staff, particularly Alexia Montaubin and Natasha Walden-Jones that supported me through the administrative processes throughout my years in the university.

Finally, my biggest thanks goes to my family and friends. These are the people who motivate me to keep pushing and bring this project to fruition.

PEDRO FREITAS MARCH 2023

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

1S1R	**1**-**S**elector-**1**-**R**esistor
1T1R	**1**-**T**ransistor-**1**-**Resistor**
Acc	**Acc**uracy
ADC	**A**nalogue-to-**D**igital **C**onverter
ALD	**A**tomic **L**ayer **D**eposition
API	**A**pplication **P**rogramming **I**nterface
a-Si	**a**morphous **Si**licon
a-VMCO	**a**morphous-**V**acancy **M**odulated **C**onductive **O**xide
BE	**B**ottom **E**lectrode
BN	**B**atch **N**ormalization
BNN	**B**inarized **N**eural **N**etwork
BP	**B**ack**P**ropagation
C2C	**C**ycle-**2**-**C**ycle
CBRAM	**C**onductive **B**ridge **R**andom **A**ccess **M**emory
CC	**C**ompliance **C**urrent
CDF	**C**umulative **D**istribution **F**unction
CE	**C**ross **E**ntropy
CF	**C**onductive **F**ilament
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
D2D	**D**evice-**2**-**D**evice
DAC	**D**igital-to-**A**nalogue **C**onverter
DRAM	**D**ynamic **R**andom **A**ccess **M**emory

EBP	**E**xpectation **B**ack**P**ropagation
eCDF	**e**mpirical **C**umulative **D**istribution **F**unction
ECM	**E**lectro**C**hemical **M**etallization
eNVM	**e**merging **N**on-**V**olatile **M**emory
EP	**E**xpectation **P**ropagation
ETML	**E**lectro**T**hermal **M**odulation **L**ayer
FC	**F**ully **C**onnected
FP32	**F**loating **P**oint **32**-bit
GA	**G**enetic **A**lgorithm
GPIB	**G**eneral **P**urpose **I**nterface **B**us
GPU	**G**raphics **P**rocessing **U**nit
GradCAM	**Grad**ient-weighted **C**lass **A**ctivated **M**apping
GS	Conductance (**G**) **S**tepping
GUI	**G**raphic **U**ser **I**nterface
HDD	**H**ard **D**isk **D**rive
HMM	**H**idden **M**arkov **M**odel
HPC	**H**igh **P**erformance **C**omputing
HRS	**H**igh **R**esistance **S**tate
IGZO	**I***n***G***a***Z***n***O**
IoT	**I**nternet **o**f **T**hings
I/O	**I**nput/**O**utput
ISPP	**I**ncremental **S**tep **P**ulse **P**rogramming
KapS	**Kap**pa **S**tatistic
KDE	**K**ernel **D**ensity **E**stimation
LFN	**L**ow-**F**requency **N**oise
LP	**L**imited **P**recision
LRS	**L**ow **R**esistance **S**tate
MAC	**M**ultiply-**Ac**cumulate
MAE	**M**ean **A**bsolute **E**rror
MAPR	**M**acro **A**verage Mean **P**robability **R**ate
MAvA	**M**acro **Av**erage **A**rithmetic

MAvG	**M**acro **Av**erage **G**eometric
MIM	**M**etal-**I**nsulator-**M**etal
ML	**M**achine **L**earning
MLC	**M**ulti **L**evel **C**ell
MNIST	**M**odified **N**ational **I**nstitue of **S**tandards and **T**echnology
MPR	**M**ean **P**robability **R**ate
MSE	**M**ean **S**quared **E**rror
MTJ	**M**agnetic **T**unnel **J**unction
MVM	**M**atrix-**V**ector **M**ultiplication
NCF	**N**on **C**onductive **F**ilament
NDR	**N**egative **D**ifferential **R**esistance
NN	**N**eural **N**etwork
NR	**N**atural **R**esponse
NVM	**N**on-**V**olatile **M**emory
ORN	**O**ther **R**ead **N**oises
OxRAM	**Ox**ide **R**andom **A**ccess **M**emory
P2P	**P**ulse-**2**-**P**ulse
PCM	**P**hase-**C**hange **M**emory
PCMO	$\mathbf{P}r_{0.7}\mathbf{C}a_{0.3}\mathbf{M}n\mathbf{O}_3$
PDF	**P**robability **D**ensity **F**unction
PIV	**P**rogramming **I**nduced **V**ariability
PLU	**P**iece-wise **L**inear **U**nit
PSD	**P**ower **S**pectral **D**ensity
PVD	**P**hysical **V**apour **D**eposition
RAM	**R**andom **A**ccess **M**emory
RNN	**R**ecurrent **N**eural **N**etwork
ROC	**R**eceiver **O**perating **C**haracteristics
RRAM	**R**esistive **R**andom **A**ccess **M**emory
RTN	**R**andom **T**elegraph **N**oise
SCM	**S**torage **C**lass **M**emory
SGD	**S**tochastic **G**radient **D**escent

SGDM	**S**tochastic **G**radient **D**escent with **M**omentum
SMU	**S**ource **M**easurement **U**nit
SNN	**S**piking **N**eural **N**etwork
SPGU	**S**emicondutor **P**ulse **G**enerator **U**nit
SPICE	**S**imulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis
SRAM	**S**tatic **R**andom **A**ccess **M**emory
SSD	**S**olid **S**tate **D**rive
STT-RAM	**S**pin-**T**ransfer-**T**orque **R**andom **A**ccess **M**emory
TE	**T**op **E**lectrode
TMDC	**T**ransition **M**etal **D**i**C**halcogenides
TMR	**T**unneling **M**agneto **R**esistance
TPU	**T**ensor **P**rocessing **U**nit
VB	**V**isual **B**asic
VCM	**V**alence **C**hange **M**emory
VISA	**V**irtual **I**nstrument **S**oftware **A**rchitecture
VMM	**V**ector-**M**atrix **M**ultiplication
WGFMU	**W**aveform **G**enerator/**F**ast **M**easurement **U**nit
W-V	**W**rite-**V**erify

Symbols

C	Capacitance	F
L	Inductance	H
M	Memristance	Ω
P	Power	W (Js^{-1})
q	Elementary charge	C
R	Resistance	Ω
$V_{Forming}$	Forming voltage	V
V_{Read}	Read voltage	V
V_{Write}	Write voltage	V

η	Learning rate	
μ	Mean	
ϕ	Flux linkage	Wb
σ	Standard deviation	

Chapter 1

Introduction & literature review

In this section the concepts that are discussed throughout the thesis will be introduced through literature review. This work focuses on the application of emerging non-volatile memories (eNVM) towards neuromorphic systems, as such, this introductory section will be split into three main sections:

1. Emerging non-volatile memories
2. Learning algorithms for neuromorphic systems
3. Neuromorphic systems with non-volatile memories

In the first section an overview of existing eNVM devices will be given with special emphasis on the topic of resistive random access memories (RRAM), since most of the work throughout this thesis is centered around RRAM. This subsection will focus more on eNVM physical and operating principles, as well as the relevant key performance metrics.

The second section will shift the focus towards different machine learning (ML) concepts. While the main focus of this work will be on Backpropagation (BP) based learning, different learning strategies will also be discussed, including limited precision (LP) variations of BP algorithms as well as global search methods such as genetic algorithms (GA).

Finally, an overview of existing application of these ML concepts onto hardware neuromorphic arrays found in the literature will be given. Beyond that, a large part of the neuromorphic design process revolves around the simulation of training and/or inference based on modelled devices, so a subsection will also be dedicated towards reviewing existing neuromorphic simulation frameworks.

1.1 Memory technology

1.1.1 Conventional memory technology

Traditional computing paradigms have relied on the same basic principles first proposed in 1945, known as the von Neumann architecture [1].

The operating principle of this paradigm relies on the data transfer between input/output (I/O) devices, a central processing unit (CPU) and a memory unit. Through technological development over the years, the processing speed of CPU's and read/write speeds of the memory units have increased significantly, however the data transfer between CPU and the memory element through the memory bus has not kept up with the latency and energy consumption demand of these sub-systems, which has led to

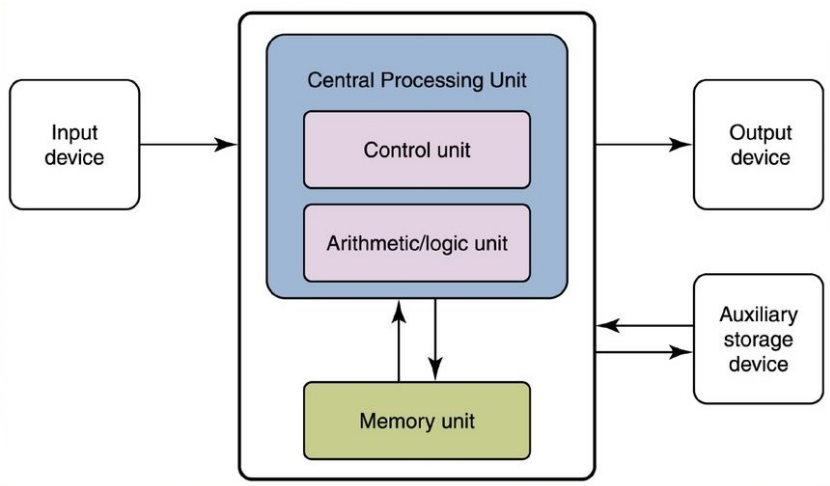


Figure 1.1: Schematic illustrating the von Neumann architecture scheme [1].

what is now known as the von Neumann bottleneck. The surge of interest in the Internet of Things (IoT) and Machine Learning (ML) elements recently has accentuated the need for alternative computing architectures that rely on in-memory computing [2] to circumvent the von Neumann bottleneck.

As seen in Figure 1.1, there exists a difference in storage elements regarding the interconnectivity with the CPU: a memory unit that communicates directly with the CPU which is known as Random Access Memory (RAM) and an auxiliary storage type memory device that can be seen as an I/O device. This comes down to the limitations of different type of traditional memory designs, since there is no single memory element that can combine all of the advantages without any of the drawbacks in terms of area consumption, read/write speed, volatility and endurance.

Semiconductor memories can typically be categorized according to their volatility, this is, whether it is able to hold the stored information when its power supply is turned-off (non-volatile) or not (volatile). Conventional volatile semiconductor memories are

comprised of static random-access memory (SRAM) and dynamic random access memory (DRAM), while traditional non-volatile memories (NVM) were initially only viable using mechanical hard disk drives (HDD) until the introduction of solid state drives (SSD) based on semiconductor Flash memory technology [3], which initially were relegated to high speed but low capacity auxiliary storage devices, but with the growth of the Flash memory technology and manufacturing cost reduction, higher capacities are achievable and are steadily replacing mechanical HDDs as the most widely adopted NVM technology.

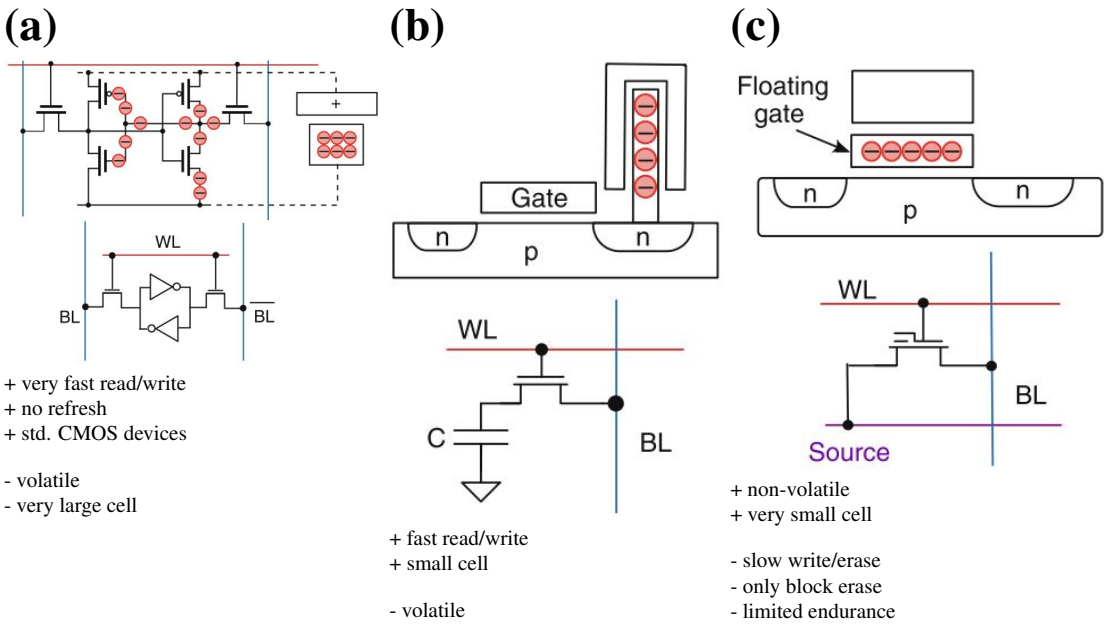


Figure 1.2: Conventional semiconductor memory technologies: (a) SRAM, (b) DRAM, (c) Flash.

Among these conventional memory technologies (illustrated in Figure 1.2), even with recent advances of Flash technology, there are still significant discrepancies in the capabilities of conventional volatile and non-volatile techonologies in terms of speed, endurance, multilevel cell (MLC) capacity and power consumption. These discrepancies mean that SRAM and DRAM still need to be used as low latency volatile solutions

while Flash, due to limited speed and endurance can only be used as auxiliary storage solutions, hindering the practicality of conventional memory technologies to be applied towards in-memory computing systems, whereas other eNVM types can be viewed as alternatives to confront these issues.

1.1.2 Emerging non-volatile memory

In 1971, Leon Chua [4] identified a theoretical symmetry between the three (known) non-linear circuit elements:

- non-linear resistor : $R = dv/di$ [4]
- non-linear capacitor : $C = dq/dv$ [4]
- non-linear inductor : $L = d\phi/di$ [4]

From this symmetry he inferred there was a missing link between flux linkage (ϕ) and the amount of electric charge that has flowed (q):

$$M = d\phi/dq\,[4] \tag{1.1}$$

This missing link was denominated as Memristance (short for memory resistance) [4].

Various experimental accounts of memristance date as far back as to the late 19^{th} century [6], with early accounts on the observation of negative differential resistance (NDR) and hysteresis, two distinctive characteristics of memristive devices, dating as

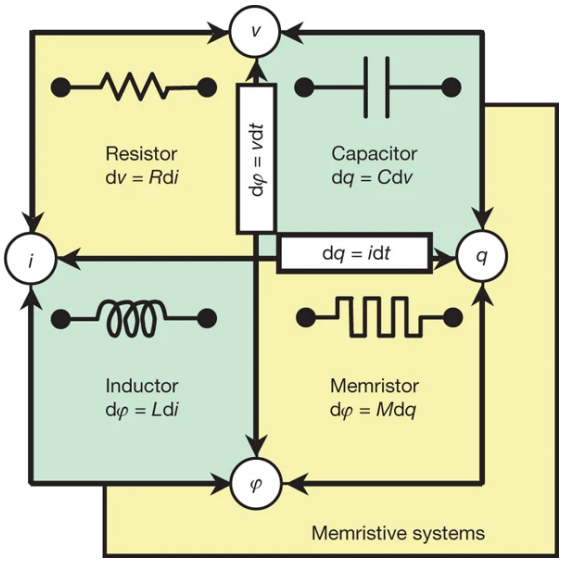


Figure 1.3: The four fundamental two-terminal circuit-elements: resistor, capacitor, inductor and memristor [5].

far back as 1896 [7] and 1904 [8] respectively. Despite this, the link between an experimental device and Leon Chua's theoretical memristor was only established in 2008 by an HP research group led by R. Stanley Williams, by associating the hysteretic I-V characteristics of a 2-terminal $Pt/TiO_2/TiO_{2-x}/Pt$ device, denominated RRAM as per its variable resistance characteristics, to that of a simulated voltage-driven memristive device [5].

Following this discovery from HP, interest by the scientific community on the research of memristive devices grew considerably and a number of different types of devices, beyond RRAM surged as alternatives.

Figure 1.4 provides a simple visual method of categorizing current memory technologies. At the highest level, these technologies can be split according to its volatility. On the volatile side, this taxonomy includes only SRAM and DRAM as two mature technologies, however, on the non-volatile side, these devices can be further categorized in terms of their technological maturity. As a baseline for non-volatile memories due to

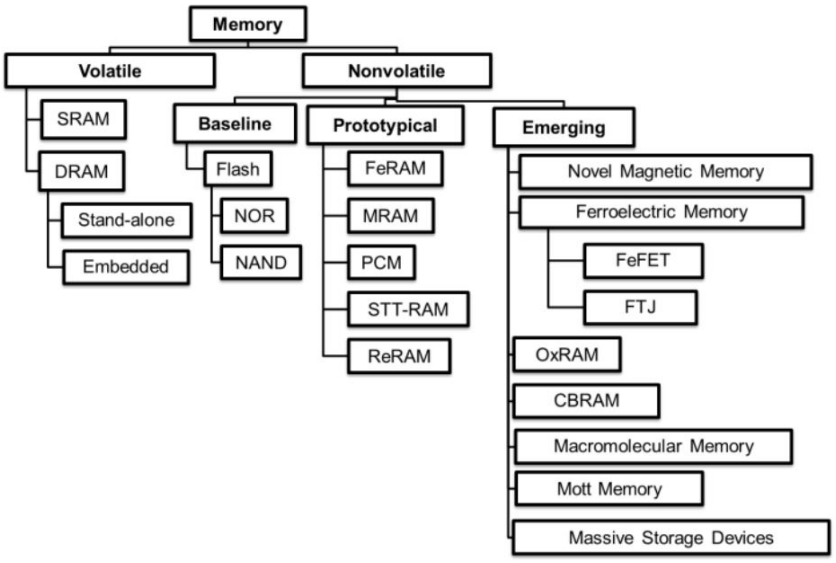


Figure 1.4: Memory taxonomy from the IEEE IRDS 2021 update [9].

its state of optimization and significant commercial presence, Flash can be considered as the baseline most mature technology and a benchmark for comparison with other products. Prototypical memory devices are at an intermediate point of maturity where there exists already significant scientific and technological background available in the literature and certain niche commercial products are making their way into the market. Beyond this point exists the emerging type of technology, which can be regarded as the least mature category but have been shown to have the potential to offer significant benefits if various scientific and technological hurdles can be overcome.

Due to the nature of this thesis and its connection to ML architectures, the category of highest interest is that of the prototypical memory devices due to its intermediate state of maturity where integrating these devices into larger architectures is now possible but there is still ample research opportunities to be developed. Beyond that, ML applications also incentivize the use of high bit density within its chips, so the focus will be on devices capable of programming to multiple conductance levels designed as multi-level cell (MLC) devices and also on 2-terminal devices due to their smaller area footprint.

With these requirements three device categories stand out: phase-change memory (PCM), spin-transfer-torque RAM (STT-RAM) and RRAM.

- PCM devices consist on a chalcogenide material sandwiched between two electrodes, which is able to transition between an amorphous (high resistance) phase and a polycrystalline (low resistance) phase triggered by heat surges generally achieved through high power electric pulses [10]. The memory cell resistance can then be read out using low voltage non destructive pulses.

- STT-RAM is based on the operating principles of magnetic tunnel junction (MTJ) devices. MTJs consist of two ferromagnetic layers sandwiching an insulating tunnel barrier, the device state is determined by tunneling magneto-resistance (TMR), which is defined as: $(R_{AP} - R_P)/R_P \times 100(\%)$; where R_{AP} and R_P are the tunneling resistance of the MTJ when the two magnetic layers are in anti-parallel and parallel alignment, respectively. The writing mechanism of STT-RAM originates from the theoretical prediction that spin-polarized electrical current can generate a torque to switch the magnetic moment of a magnet through the transfer of angular momentum [11–13].

- RRAM devices generally consists of a 2-terminal metal-insulator-metal (MIM) structure that is able to switch between a high resistance state (HRS) and a low resistance state (LRS) through the application of different voltages. The choice of materials (both metals and insulators) of the structure, as well as their dimensions, will dictate the physical phenomenons responsible for the switching of the resistive states. Depending on these physical principles, RRAM can be further categorized into oxide RAM (OxRAM) [14–16] or conductive bridge RAM (CBRAM)

[17, 18]. Since this thesis will have a higher emphasis relating to RRAM, a more detailed overview will be presented in the following subsection.

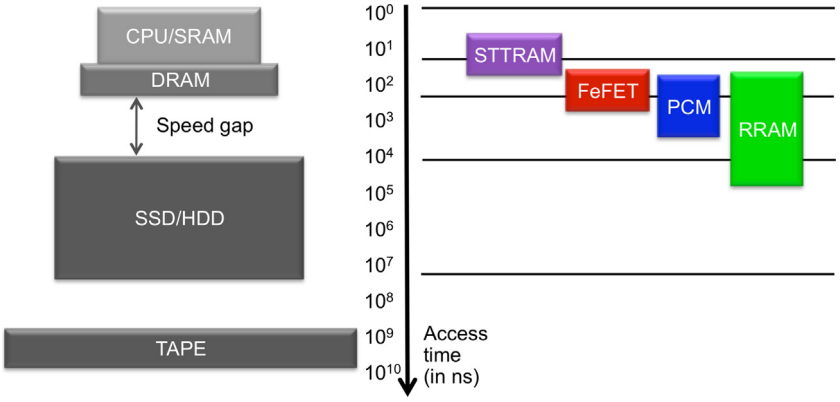


Figure 1.5: An illustration of memory hierarchy with typical access speed, lined up with the performance range of major eNVM devices [19].

The aforementioned eNVM devices all possess uniquely distinct physical operation principles that are able to bridge the speed gap that exists between conventional volatile and non-volatile memory technologies (Figure 1.5), this in turn opens up several possibilities on applications beyond the scope of the von Neumann architecture, such as: in-memory [2, 20] and bio-inspired computing [21–23] and novel hardware security primitives [24–26].

1.1.3 Introduction to RRAM

The observation that nominally insulator oxide materials could undergo abrupt switching events into conductive states date has far back as the 1960s [27–30], however, these early observations were not robust enough to support practical development of memory applications. It was only in the early 2000s, firstly with the demonstrations of NiO memories integrated in a conventional $0.18\mu m$ CMOS node in a one-transitor-one-resistor

(1T1R) structure by Samsung [31], and then by the memristor discovery by HP [5], that the scientific interest in RRAM took off.

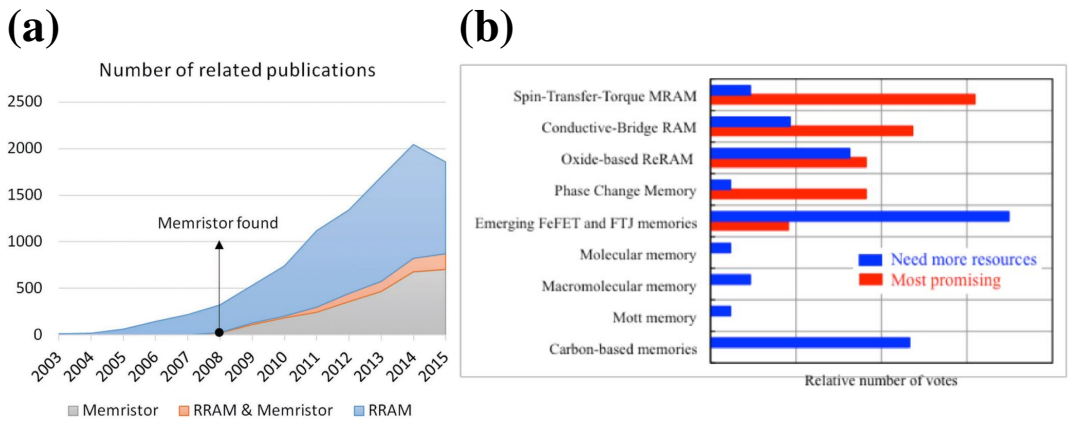


Figure 1.6: (a) Number of publications filtered by specific keywords: RRAM, RRAM & Memristor, and memristor (excluding RRAM) from 2003 up to 2015 [32]. (b) Survey of Emerging Memory Devices from the 2014 ERD Emerging Logic Workshop [9].

These discoveries not only led to a massive surge of interest in RRAM but also in memristive devices in general for storage class and beyond CMOS applications (Figure 1.6a). Concerning the different types of eNVM, RRAM (OxRAM & CBRAM combined), according to the IEEE International Roadmap for Devices and Systems, comes in as the 2nd most promising eNVM technology, just behind STT-RAM (Figure 1.6b).

RRAM operating principles are based on the migration of defects inside the insulating medium of a MIM structure (usually caused by the application of an external electric field) which will affect the resistivity of the overall device. Beyond this simple principle, the underlying mechanisms causing defect migration and resistance switching are varied, complex and heavily dependent on device engineering. Several non exclusive subcategorizations can be made for the broader RRAM category; for the purpose of this thesis, RRAM will be classified according to its: nature of defects (oxygen

vacancies for OxRAM or metalic ions for CBRAM), switching model (filamentary or non-filamentary) and operation mode (unipolar or bipolar).

1.1.3.1 Switching model

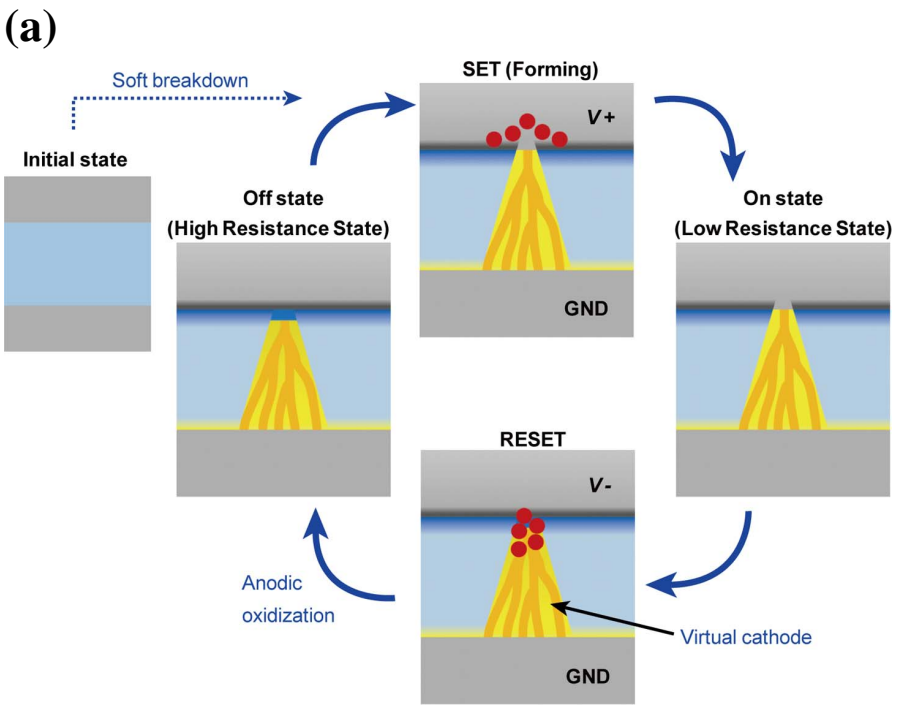
Most RRAM devices reported in the literature are of the filamentary type. In filamentary RRAM (Figure 1.7a), switching is controlled by building up a conductive filament (CF) inside the insulating layer which forms a percolation path that shorts the two conducting electrodes, therefore switching towards a LRS. This initial filament is achieved by applying an electric field to the MIM structure that is strong enough to lead to a soft dielectric breakdown and electromigration of defects [33]; this initial process is called the forming process [34]. Once on the LRS, the filament can be ruptured, depending on the nature of defects in the device, this can either be achieved through the passing of a high current that breaks down the filament through Joule heating, or through the application of a reverse electric field leading to the filament dissolution through redox reactions and consequently, ”resetting” the device resistance back to the HRS. The filament can then be restored by using a similar electric field to the forming process but with lower magnitude; this is called the ”Set” process. The memory element of the device is therefore present on whether the filament is ruptured or shorting the two electrodes.

Non-filamentary switching (Figure 1.7b) contrasts with the filamentary type in the sense that there is no CF formation on the insulator material, instead, resistance changes by modulating the defect profile close to the interfacial regions of the device, either between in the metal-oxide interface [35] or inner-interfacial region of multi-layer devices

[36]. The absence of one single critical percolation path in non-filamentary devices generally translates in a few distinct characteristics, such as: low operating currents (sub-μA) [37], controllable MLC capability [38, 39], low intrinsic variability [40], forming-free [37] and since the change of resistance happens across the volume of the device, an external current limiter is not a necessity to prevent a hard breakdown (self-compliance) [37, 41].

1.1.3.2 Nature of defects

Beyond the switching model, RRAM devices can also be categorized according to the nature of the moveable defects. The defects responsible for switching can be anionic (negative charges) or cationic (positive charges). In this context, devices where



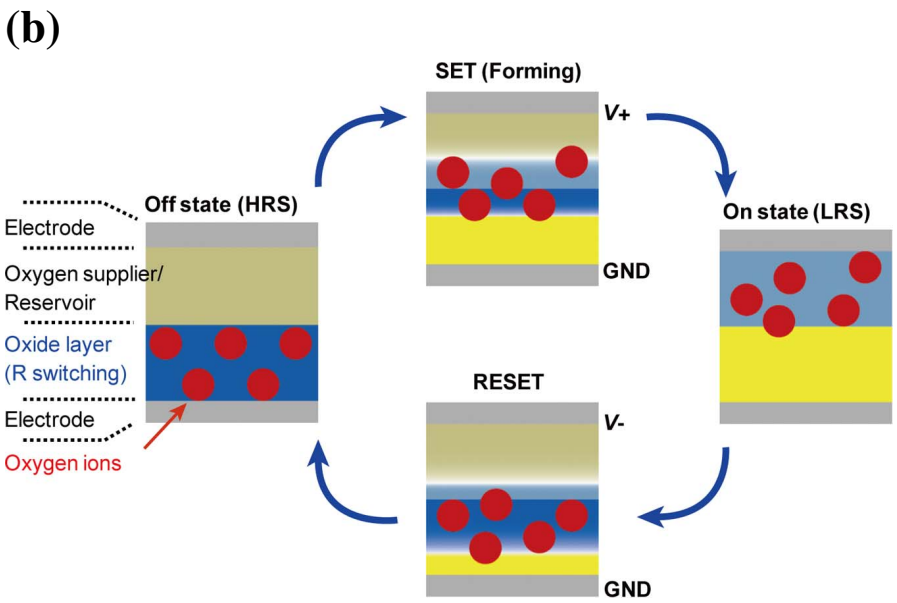


Figure 1.7: Schematic illustrating: (a) Filamentary and (b) Non-filamentary resistive switching [15].

the switching is based on cation migration can generally also be referred to as electrochemical metallization (ECM) memory or CBRAM, and anionic based devices can also be referred to as valence change memory (VCM) or OxRAM.

In CBRAM, switching is based on the migration of metal ions across the insulator material and subsequent reduction/oxidation (redox) reactions [17, 42]. To enable this phenomenon, CBRAM devices need to generally be composed of asymmetric electrodes: one oxidizable (ionizable) electrode such as Ag, Cu or Ni and a relatively inert electrode, e.g. W, with an ion conducting medium between them that is able to transport the metal cations. Providing a positive voltage to the oxidizable electrode leads to the dissolution of the metal cations (e.g. $Ag \rightarrow Ag^{+} + e^{-}$) that are transported by the electric field towards the inert electrode, where the ions are reduced (e.g. $Ag^{+} + e^{-} \rightarrow Ag$) and deposited upon contact. This process leads to the formation of a conductive bridge

that shorts the two electrodes, therefore switching to the LRS. In some cases the filament can be dissolved through Joule heating, but in other cases, reversing the voltage polarity leads to the opposite redox reactions causing filament dissolution instead of formation, reverting the device back to HRS. The description for CBRAM seems very similar to that of filamentary switching so it is worth noting that due to the nature of the involved materials, most CBRAM devices are of the filamentary type, however, there have already been interesting reports of non-filamentary type cation-migration devices [43].

In OxRAM, the physical mechanism that is responsible for resistive switching is generally associated with the generation of oxygen vacancies (V_o^{2+}) and subsequent relocation of oxygen ions (O^{2-}). Common reports are found for both filamentary and non-filamentary type OxRAM devices.

For filamentary OxRAM, the switching mechanisms are very similar to those of the filamentary CBRAM devices, with the main difference that instead of having redox reactions on one of the electrodes, oxygen atoms are knocked out of the lattice of the insulating layer and drift toward the anode under the application of high electric fields, while oxygen vacancies are generated and get accumulated near the cathode, creating a phase conductive filament that connects both electrodes resulting in switching from HRS to LRS. Similar to the CBRAM case, in some cases this filament can be re-oxidized and ruptured by reversing the voltage polarity (valence change mechanism [44]), in other cases the filament can be ruptured by driving a large current and subsequent joule heating effects (thermochemical mechanism [45, 46]). Various materials systems possessing filamentary switching dynamics have been studied, such as: metal

oxides [47, 48], 2D transition metal dichalcogenides (TMDC) [49], perovskite [50, 51] and organic materials [52]. Of these, the class of binary oxides is of particular interest due to its general CMOS compatibility, multistate switching and simple chemical composition; material systems in this class include: HfO_2 [53–56], ZnO [57], NiO [58], TiO_2 [59], WO_3 [60] and TaO_x [61, 62].

Beyond this, OxRAM devices can also be of the non-filamentary type, which involve oxygen vacancy exchange on the interface between two different materials of the stack. A typical example is based on redox processes at these interfaces, leading to varying thicknesses of the interfacial layers and thus different resistance states [63]. Typical material systems used for non-filamentary switching include: $Pr_{0.7}Ca_{0.3}MnO_3\,(PCMO)$ [38, 64], $InGaZnO\,(IGZO)/\alpha - IGZO$ [65], TiN/TiO_x [66], $Al/a - TiO_2$ [67], and $a - Si/TiO_2$ (a-VMCO) [37].

1.1.3.3 Operation scheme

General RRAM operation can be divided into four steps: Forming, Reset, Set and Read.

Forming: The forming process is generally the first step of operation on a fresh RRAM device and is generally only applied once. This involves the application of a high electric field that induces a soft dielectric breakdown that allows for the initial defects to be delocalized into the switching layers, changing the device state to the LRS (logic "1"); all subsequent operating voltages are then lower than the forming voltage ($V_{Forming}$). Typically filamentary devices require some sort

of current compliance (CC) mechanism to prevent a hard permanent dielectric breakdown to occur due to high current. Non-filamentary devices, due to the nature of their switching mechanism, may not need either the application of CC or the forming process and are known as self-compliant and forming-free [37, 41] devices respectively.

Reset: The Reset process refers to the application of a specific electrical stimuli that allows the device to change back to the HRS (logic ”0”). This involves the breaking (or thinning) of the CF in filamentary type devices and the barrier modulation in the interfacial layers of non-filamentary devices.

Set: The Set process is in essence very similar to the forming process, with the difference that subsequent ”Sets” do not require high electric fields because there are already mobile defects present in the switching layers induced by the initial forming process.

Read: The Read process simply consists on the application of a carefully selected voltage value across the 2-terminal RRAM so that the current (or resistance) across the device can be registered. The read voltage (V_{Read}) that is low enough has to not induce any further changes in resistance but high enough so that a reasonable current window between the HRS and LRS can be read.

Depending on both the switching model and the nature of defects of the device, the RRAM device can operate under two different schemes: unipolar or bipolar switching.

In unipolar switching (Figure 1.8a), the switching operation is independent of the polarity of the write voltage (V_{Write}). The Set process typically requires to be limited

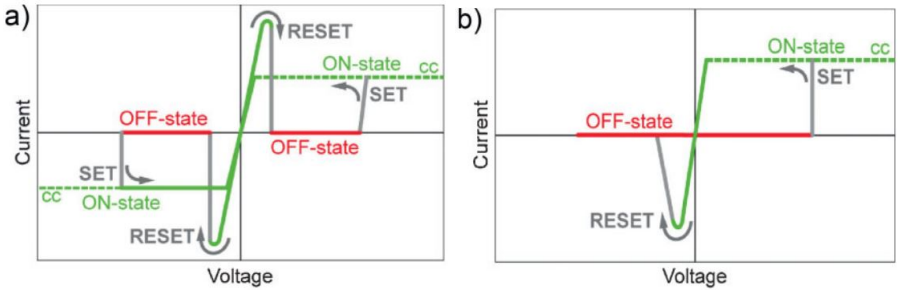


Figure 1.8: The two basic operation schemes of RRAM. I-V curves recorded for a triangular shaped voltage signal. CC denotes the compliance current. Dashed lines indicate that the real voltage at the system will differ from the control voltage because of the cc in action. (a) Unipolar switching. (b) Bipolar switching [14].

by a CC, while the Reset process is conducted by applying a voltage signal of the same polarity while removing this current limiter, therefore inducing a high current that will change the device back to the HRS, normally induced by joule heating effects [45, 46]. This type of operation tends to be more common on filamentary type devices.

Bipolar switching devices on the other hand (Figure 1.8b), require that the voltage polarity of the ”Set” and ”Reset” processes be opposite to each other. This is brought about on devices in which the physical mechanisms is strongly dependent on redox reactions caused by voltage application. In this case, there is a dedicated polarity which will induce carrier migration in one direction that will be responsible for the ”Set” process, and the ”Reset” process, contrary to the unipolar scheme, is not done through application of high currents but rather through the change of voltage polarity in the device which leads to the reverse redox reaction and consequent reversal of resistance state. This type of operation tends can be found on both filamentary and non-filamentary type devices.

1.1.4 Key performance metrics

In this subsection, an emphasis on the different performance requirements that need to be met by eNVM will be given. Since a large focus of this thesis is on neuromorphic computing, a distinction will be made between the performance metrics for storage class memory (SCM), denominated basic reliability metrics, and functional reliability metrics for neuromorphic specific applications (Figure 1.9).

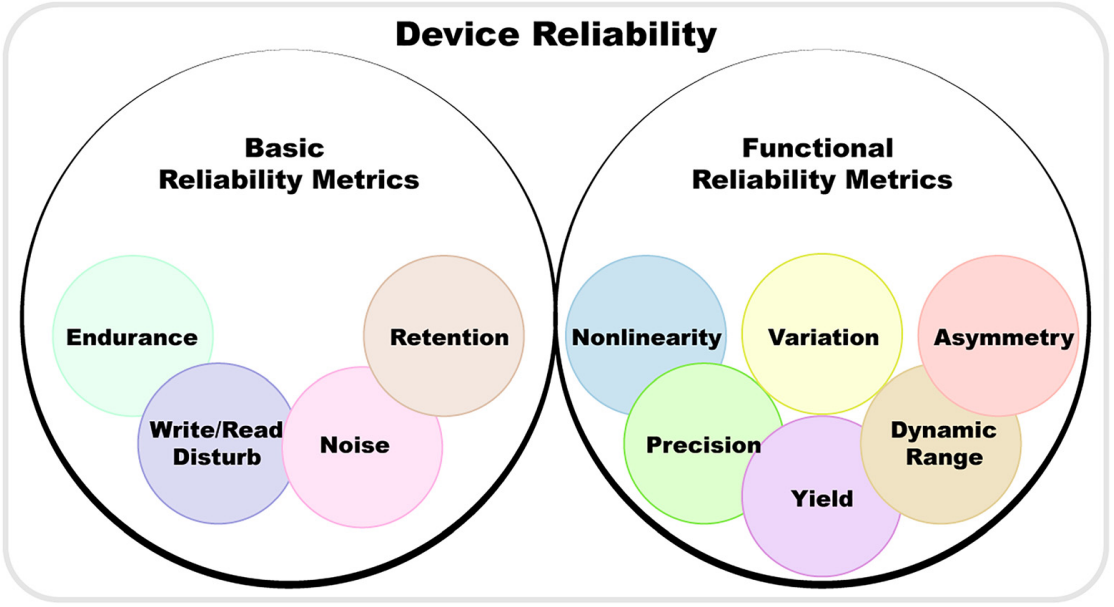


Figure 1.9: Reliability metrics of the neuromorphic device. Device reliability metrics are classified into basic and functional reliability metrics [68].

Basic reliability metrics include: endurance, retention, write/read disturbances and noise; while functional reliability metrics, given the importance of MLC behaviour for neuromorphic applications, go beyond the requirements for SCM and include: programming non-linearity, asymmetry, precision, dynamic range, variability and yield.

It should be noted that basic reliability metrics are also fundamental requirements for neuromorphic computing. Even though the concepts are similar for both SCM and neuromorphic computing, the requirements for both may vary, as shown in Figure 1.10.

The main difference in requirements between SCM and neuromorphic computing can also be viewed as a difference between digital data storage (SCM) and analogue data processing and storage (neuromorphic). In traditional digital data storage, the requirements need to comply as to maintain the device resistance (or conductance) between certain reference lines that still allow the bits to perceivable without errors, as such, the basic reliability metrics for SCM are more lenient as long as those reference lines are not crossed. For analog data storage on the other hand, the goal is to achieve as many perceivable resistance levels in one device without errors, as such, the same basic reliability metrics become much more strict in cases such as: retention, endurance and noise, where small variations or drifts can have a big impact on the perceivable state of the device and therefore on computing accuracy. More in-depth descriptions for each of these metrics will follow in the next sections.

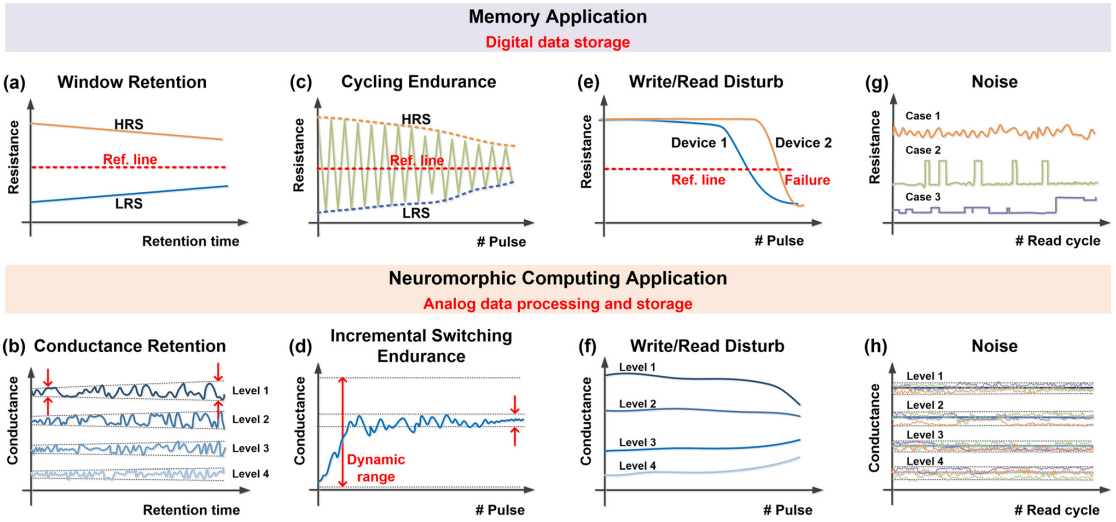


Figure 1.10: Schematic diagram of different basic reliability metrics of memory application for digital data storage and neuromorphic computing application for analog data processing and storage. (a) Window retention of digital memory and (b) conductance retention of analog data in the computing process. (c) Cycling endurance of digital memory and (d) incremental switching endurance of analog data in the computing process. (e) Write/read disturb in memory and (f) computing. (g) Noises in memory and (h) computing [68].

1.1.4.1 Retention

Retention is the metric that evaluates how long the device can maintain its conductance value, in short, it quantitatively evaluates the non-volatility of the device. The commonly accepted standard for data retention for SCM applications is around 10 years at a temperature of $85°C$. Testing for such a long period of time is obviously unrealistic, as such there are different methods by which the retention properties can be inferred. One common method is a simple linear extrapolation where the device is baked at high temperature and multiple read measurements are taken during a long period of time ($10^5 s$) and the data is then extrapolated towards the 10 year mark. A more accurate method however is to use the Arrhenius equation:

$$k = Ae^{\frac{-E_a}{k_B T}} \text{ [69, 70]} \tag{1.2}$$

where:

k is the rate constant

T is the absolute temperature (in degrees Kelvin)

A is the pre-exponential factor

E_a is the activation energy

k_B is the Boltzmann constant

In this method, time to failure can be recorded at different high temperatures (in a reasonable amount of time), the Arrhenius plot can then be used to extract the activation energy and extrapolate for the reasonable device operating temperatures [71].

On the neuromorphic side, the retention concern is extended beyond not only maintaining a long retention time of binary states, but also the ability to keep a tight conductance distribution with minimal drifts of multiple analog conductance states. Zhao *et al.* [72] has devised a compact model describing retention drift based on the statistical collection of filamentary RRAM retention behaviours across 8 different conductance levels and measured the impact of retention time on an inference NN trained for MNIST pattern recognition.

1.1.4.2 Endurance

Endurance is the measure that quantitatively evaluates the maximum number of switching cycles a memory device can perform before failure. During successive cycling operations in RRAM, the resistance values of its defined states may shift beyond its defined reference thresholds(Figure 1.10c & d). Depending on its switching mechanisms, RRAM can have different types of endurance failures.

In filamentary RRAM, Chen et al. [73] reported three types of endurance failure behaviours (Figure 1.11).

Type I is characterized by a drift of both resistance states towards the middle of the resistance window and can be attributed to an interfacial electron barrier created by oxidation of a metal electrode subjected to large power/current and high temperature.

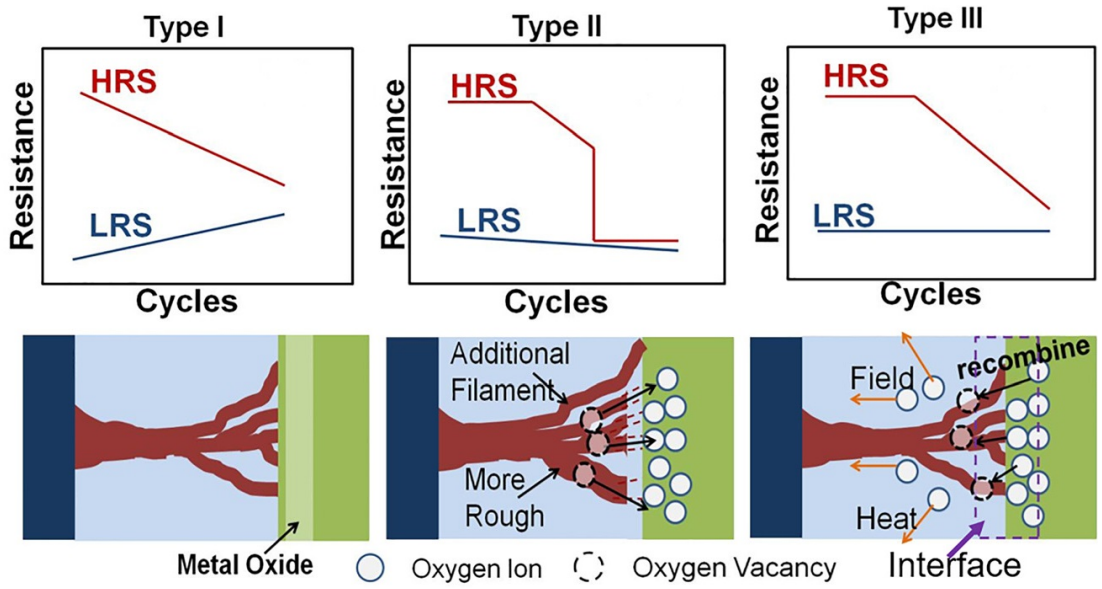


Figure 1.11: Schematic of the endurance failure mechanism of RRAM. Three failure types of endurance degradation are illustrated. Adapted from [73].

In type II, there is a gradual continual drift of LRS towards lower resistance values accompanied by a degradation of HRS in two stages, a first stage where the HRS drifts to lower values and a second stage where there is a sudden drop of resistance and reset is no longer possible. This type of failure is caused by redundant V_o^{2+} generation that incrementally increase the thickness of the CF up to the point of failure where the CF can no longer be broken by the reset process.

Failure type III is identified by a gradual loss of the HRS resistance while maintaining a steady LRS. Frequent switching gradually leads to the depletion of O^{2-} ions that are required for V_o^{2+} recombination in the reset process [44], in turn leading to gradually more deficient reset processes.

In non-filamentary type devices, particularly in a-VMCO, tends to occur with closure of the resistance window for both LRS and HRS, similar to the filamentary failure type I (Figure 1.11).

However, due to the inherently different switching mechanism, this failure attribute is also different. Subhechha et al. [74] proposed that defect movement in the switching layer involves a combination of an electric field-driven model [75], where electrons play a significant role, and a current-driven electromigration-based model [76]. In the field-driven switching mechanism, the distribution of defects becomes more symmetric, the tunneling gap between neighbouring defects becomes more uniform, and the device conductance increases, driven by the movement of electrons. Conversely, in current-driven switching, electrons push defects closer to the device or grain boundary edge, rendering these defects inactive in the conduction mechanism, consequently reducing the device conductance.

In storage-type SCM applications, 10^6 endurance cycles is generally considered as the base target for memory devices to achieve (as listed in the IEEE IRDS 2021 Update [9]). However, neuromorphic applications, particularly in NNs, can require a wide range of weight updates depending on the task and dataset, although for small tasks such as MNIST, an endurance of around 10^6 may be sufficient [77], larger datasets may require significantly more weight updates. On the other hand, NN training does not require full switching between HRS and LRS, but only the addition of small increments of current at a time, which reduces the endurance strain on the device [78]. Another mitigating factor on NN endurance requirements is that there exists some flexibility in terms of mapping the weights to the synaptic device, as such, several pieces of work exploit intelligent programming and weight mapping solutions towards mitigating endurance effects and extending device lifetime for online training [79–81].

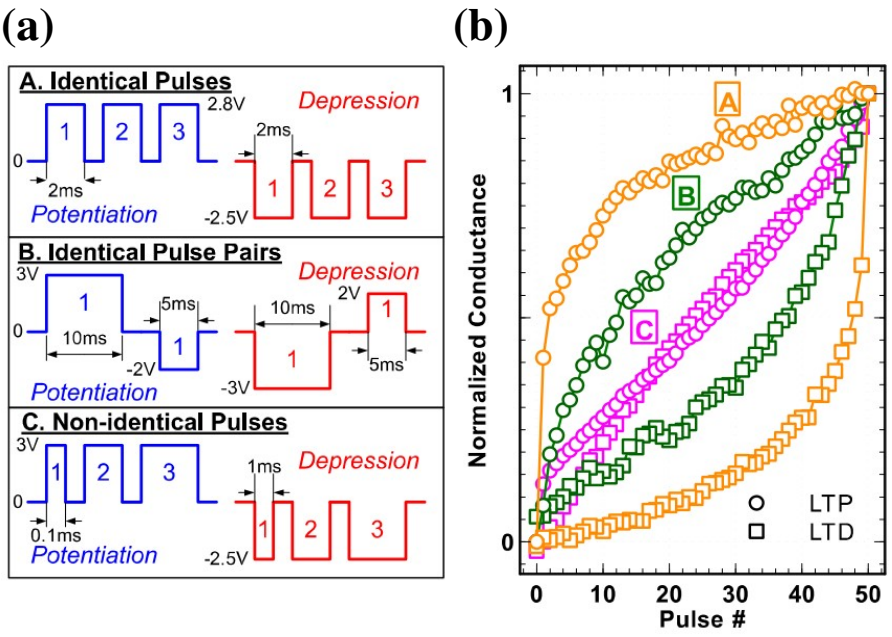


Figure 1.12: (a) Different programming schemes and (b) the corresponding experimentally measured data of conductance modulation in TaO_x/TiO_2 based synaptic device [82].

1.1.4.3 Nonlinearity

As previously mentioned in Figure 1.9, nonlinearity is a functional reliability metric that mostly affects performance on neuromorphic applications. Nonlinearity can be seen as the rate of conductance change as a function of the number of applied voltage pulses and can also be seen as a quantitative measure of the device's analog capabilities.

Ideal synaptic behaviour would be a perfectly linear evolution in conductance with the applied voltage pulses, however, most synaptic devices exhibit inverse exponential conductance evolution due to their intrinsic switching mechanism, which still imposes a major challenge for NN online training.

Nonlinearity in weight update is a specially relevant issue in filamentary RRAM, as the forming process mostly controls CF growth through a drift process [83], after the CF is completed, filament thickness is controlled by diffusion [83–85] of only a few

defects which have a big impact on device conductance hence larger jumps with each voltage pulse.

In non-filamentary devices, however, conductance is mostly controlled by barrier modulation of a large number of defects, the contribution for each defect is evened out throughout the device area and each voltage pulse can have small incremental contributions towards conductance change, therefore improved linearity [86–88].

Nonlinearity mitigations strategies can be achieved either through material and structure engineering or novel programming schemes.

Through materials engineering several pieces of work have been explored towards improving nonlinearity. Wu et al.[89] proposed the insertion of an electrothermal modulation layer (ETML) over the switching layer of an HfO_x device for better control of electric field and thermal distributions, improving Reset and Set linearity respectively. Chandrasekaran et al. [90] introduced uneven Al dopants, creating oxygen rich and oxygen poor regions that confine filament formation, therefore improving programming linearity. Moon et al. [91] designed a 1T2R structure with a serial-connected resistor for voltage division, resulting in lower voltage drop across the RRAM and improved linearity.

Beyond solutions in device engineering, nonlinearity mitigation strategies coming from novel programming methodologies can also be explored. Bipolar programming pulse trains with positive and negative pulse pairs can be used towards reducing the effects of over-programming in the steepest parts of the conductance curves [82, 92].

Pulse trains with increasing voltage amplitude or pulse widths can be effective at counteracting the natural exponential nonlinearity that comes from using identical pulses [82] (Figure 1.12). Cai et al. [93] introduced the concept of charge-domain pulse width modulation to mitigate nonlinear conductance update behaviour in RRAM, however, this is only possible with the use of custom current-integrating hybrid analogue-to-digital converter (ADC) and pulse-mode digital-to-analogue converter (DAC) scheme. Novel programming schemes for nonlinearity mitigation therefore come at the cost of increasingly complex peripheral circuitry design.

1.1.4.4 Variability

Variability is defined as any variations between multiple instances of memory switching. In typical SCM applications, variability is often measured between only two distinguishable states (HRS and LRS), as such, two common types of variability stand out: variability between different cycles of switching within the same device, called cycle-to-cycle (C2C) variability, and variability between different devices, denominated device-to-device (D2D) variability.

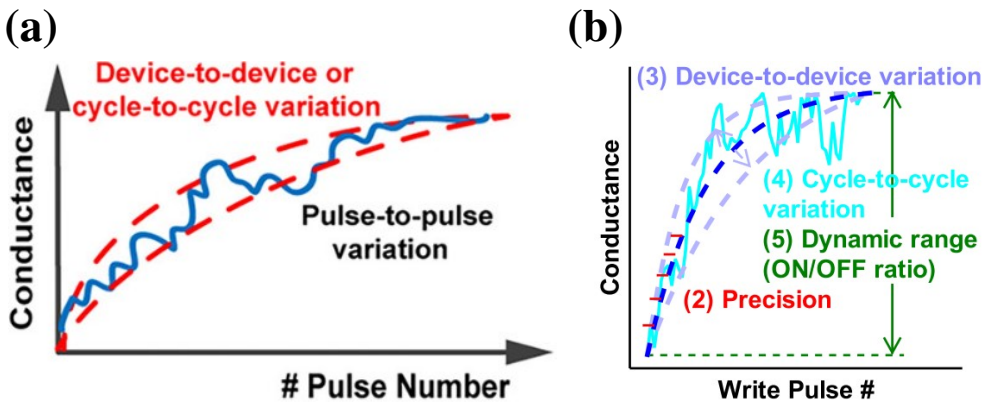


Figure 1.13: (a) Definitions of the different types of variability [68]. (b) An example of a common misconception between C2C and P2P definitions found in the literature [94].

In neuromorphic applications, the device is usually programmed in multiple pulses in order to exploit its analog conductance capabilities. This type of programming gave rise to a third type of variability defined as pulse-to-pulse (P2P) variability (Figure 1.13a). Nevertheless, since neuromorphic application of eNVM devices is still a relatively new and growing field of interest some common misconception on the different definitions are still present in the literature (Figure 1.13b).For the purpose of this thesis the definitions from Figure 1.13a are considered.

Variability in RRAM is an intrinsic property of the device that is linked to the stochastic nature of oxygen vacancy/ionic movement [95]. Therefore the switching model in each device plays a significant role in terms of device variability.

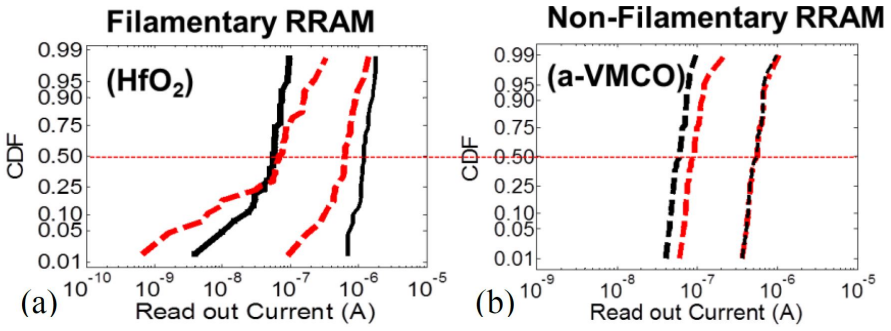


Figure 1.14: Cumulative distribution functions (CDF) of the read-out current between different devices before (black) and after (red) retention tests for (a) filamentary HfO_2 and (b) non-filamentary a-VMCO RRAM [37].

In filamentary RRAM LRS variation comes from both the number of CFs as well as their size, thus the reduction of possible filament paths by restricting the active switching area may reduce LRS variation. HRS variation, on the other hand comes from variations on the ruptured CFs length, as such, any small variations of the tunneling gap can lead to exponential variations on tunneling current (Figure 1.14a).

For non-filamentary RRAM however, current transportation mechanisms dominated by interfacial barrier modulation diminishes the impact of singular defects, resulting not only in similar variability distributions for both HRS and LRS as well as tighter distributions of current overall (Figure 1.14b) [37].

1.1.4.5 Noise

In electronics, noise can be defined as deviations from an average electrical signal (voltage or current) present during measurement.

Several different types of noise exist in electronics, however, in RRAM three types are the most prevalent: thermal noise [96], $1/f^{\alpha}$ [97, 98] and random telegraph noise (RTN) [99, 100].

Thermal noise, often also referred to as Johnson-Nyquist or white noise, is an unavoidable type of electrical disturbance generated by the random thermal motion of charge carriers inside an electrical conductor, which happens regardless of the applied voltage.

$1/f^{\alpha}$ noise, also referred to as Flicker or pink noise, refers to a kind of low-frequency noise (LFN) that is defined by its power spectral density (PSD) function (often also referred to as Lorentzian), which can be fitted by a $1/f^{\alpha}$ power law, where $1 \leq \alpha \leq 2$. Beyond simply being a source of noise in electronics, $1/f^{\alpha}$ noise can be used as a tool for characterizing the underlying conduction and switching mechanisms in RRAM. Yu et al [97] determined that in a $TiN/HfO_x/AlO_x/Pt$ RRAM device, the LFN can be fitted to $1/f^{\alpha}$ with $\alpha \sim 1$ for LRS and $\alpha \sim 2$ for HRS, and a cut-off frequency in this

transition, suggesting that CFs are ruptured and a tunneling gap is formed during the reset process.

RTN, often also referred to as burst noise is a dominant pattern of LFN particularly characterized by a distinct fluctuation between two discrete conductance states (Figure 1.15a), originating from the filling or emptying of one or more traps. Furthermore, RTN can also be seen as the specific type of LFN where the PSD is defined as $1/f^2$ (Figure 1.15b).

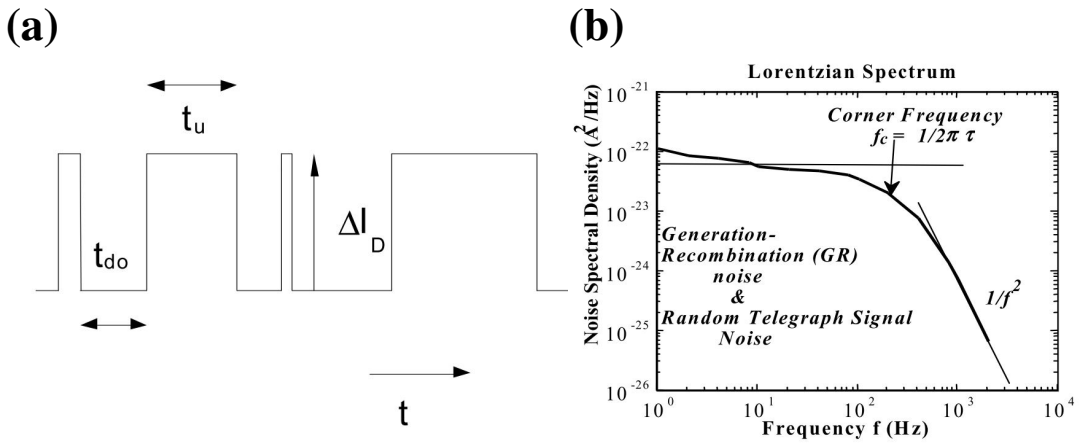


Figure 1.15: (a) Schematic representation of a two-level RTN signal, defining its main parameters and (b) its Lorentzian spectrum [99].

As with $1/f^{\alpha}$ noise, RTN obervations can be used as a powerful characterization tool for a detailed insight into RRAM conduction mechanisms and switching dynamics [101].

Since RTN is a consequence of trapping/detrapping of defects in the RRAM switching layers, there is a significant difference in terms of impact the RTN amplitude present in CF and NCF RRAM types. In CF RRAM, one single defect can be responsible for the restoration/rupture of the CF, hence causing large fluctuations in terms of conductivity [102], while in NCF devices, the impact of trapping/detrapping a singular defect

is evened out throughout the device area, having therefore an attenuated impact in its overall conductivity [103]. This difference between CF and NCF devices can also be observed on the impact that RTN has in the accuracy of a neuromorphic inference engine used for pattern classification [104].

1.1.4.6 Power

With ever increasing circuit complexity and aggressive nanoelectronics scaling, power consumption has become an ever increasing concern. Particularly, neuromorphic systems aim to mimic the efficiency of biological synapses which use approximately $10fJ$ per energy spike, however, the programming energy for most RRAM devices is around $100fJ \sim 10pJ$, and PCM devices may consume $10pJ \sim 100pJ$ per programming operation, meaning that realistic consumption of hardware based synapses is anywhere between 10X to 10 000X higher than the biological plausible goal.

The fundamental challenge comes from the type of ionic movement present in biological and hardware based synapses. Synaptic events in biological synapses come from the movement of ions in a liquid environment, whereas in hardware synapses based on eNVM, programming comes in the form of ionic/defect movement through a solid-state medium which requires more energy.

Even though there are still significant challenges in terms of reducing individual synaptic power consumption on eNVM devices, an even more significant difficulty comes when considering all of the necessary components present in neuromorphic chip design. Peng et al. [105] simulated energy consumption estimates of an entire neuromorphic array including peripheral circuitry (Figure 1.16). It was shown that the eNVM

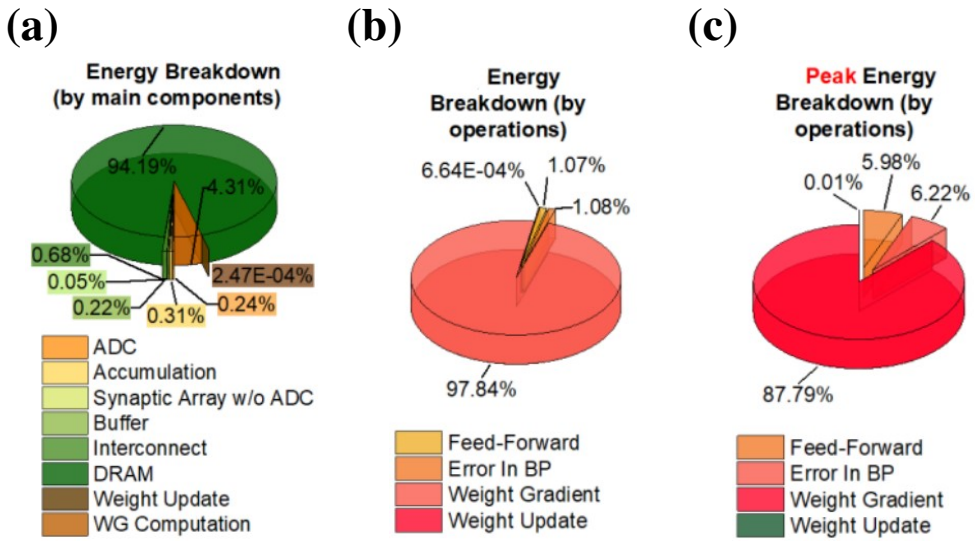


Figure 1.16: Hardware estimation results for each epoch, adapted from [105]. Energy breakdown by (a) main components, (b) by operations and (c) peak energy breakdown by operations. Data extracted simulating the device from [106].

synaptic arrays only account for 0.05% of the total energy estimates, while most of the energy consumption comes from temporary DRAM buffer storage (94.19%) and the second highest consuming component is the ADCs (4.31%)(Figure 1.16a). Another worthy observation is that average and peak energy consumption on the feed-forward operation is of 1.07% and 5.98% (Figures 1.16b & c) respectively, highlighting the large discrepancy in power consumption between inference only and online training applications.

1.1.4.7 Scalability

Coupled to the power consumption issue of neuromorphic arrays comes the concern for scalability. The scaling of NN parameters can be extremely aggressive depending on the desired task and most of the available NN processing solutions come in the form of power hungry graphics processing units (GPU) or application specific tensor processing units (TPU), which can contain over 28M transistors per die.

Most eNVM devices, on the other hand, are 2-terminal devices with MLC capability. Due to these characteristics, eNVM devices such as RRAM not only can be organized as 2D crossbar arrays, where the RRAM device is fabricated at each crossing point between a word and bit line, but multiple 2D arrays can be vertically stacked on top of each other for very high density architectures. Beyond this, the crossbar array is also capable of performing matrix-vector multiplications (MVM) and vector-matrix multiplications (VMM) naturally through a combination of Ohm's and Kirchoff's law, enabling in-memory computing solutions applicable to neuromorphic architectures.

Nevertheless, one of the main issues hindering the practicality of RRAM crossbars is that of sneak currents. Sneak currents are defined as the currents that arise in a passive crossbar array due to multiple parallel conduction paths between a given top electrode (TE) and bottom electrode (BE) and poor non-linear behaviour of the unselected devices.

The sneak path issue is typically solved by combining a transistor in series with the RRAM element, making up a 1T1R structure. This solution however will reduce the benefits of having a 2-terminal high density memory element, as such, significant research interest exists in the development of 2-terminal highly nonlinear selector devices that can be used to replace the transistor in a 1T1R structure so has to build 2-terminal 1-selector-1-resistor (1S1R) structures [107–109].

1.1.4.8 Comparison between different eNVM

Given the different types of performance metrics, clear distinctions can be made regarding the suitability of different eNVM for different applications.

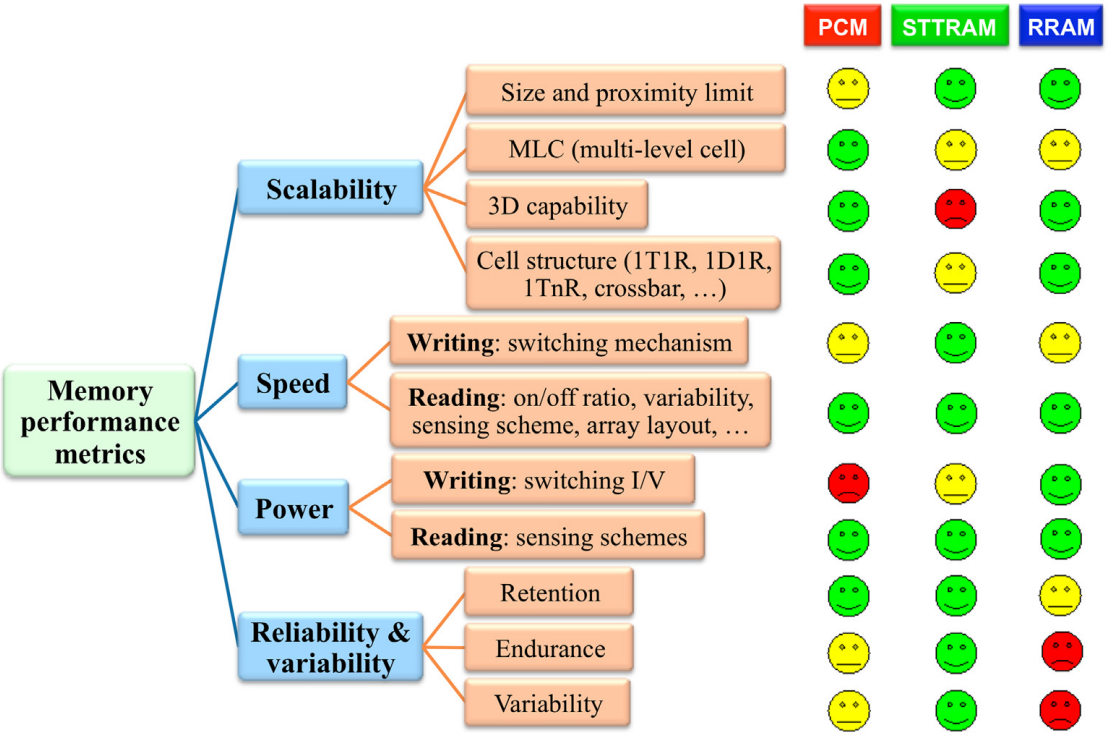


Figure 1.17: Key metrics for memory performance assessment and a qualitative comparison of STTRAM, PCM, and RRAM based on the metrics [19].

Table 1.1: Detailed comparison of different memory technologies

Technology	SRAM	DRAM	NAND Flash	NOR Flash	PCM	STT-MRAM	RRAM
Cell area	$> 100F^2$	$6F^2$	$< 4F^2$	$10F^2$	$4-20F^2$	$6-20F^2$	$< 4F^2$
Cell element	6T	1T1C	1T	1T	1T(D)1R	1(2)T1R	1T(D)1R
Voltage	$< 1V$	$< 1V$	$< 10V$	$< 10V$	$< 3V$	$< 2V$	$< 3V$
Read time	$1ns$	$10ns$	$10\mu s$	$50ns$	$< 10ns$	$< 10ns$	$< 10ns$
Write time	$1ns$	$10ns$	$100\mu s$-$1ms$	$10\mu s$-$1ms$	$50ns$	$< 5ns$	$< 10ns$
Write energy (J/bit)	$1fJ$	$10fJ$	$10fJ$	$100pJ$	$10pJ$	$0.1pJ$	$0.1pJ$
Retention	N/A	$64ms$	$> 10y$	$> 10y$	$> 10y$	$> 10y$	$> 10y$
Endurance	$> 10^{16}$	$> 10^{16}$	$> 10^4$	$> 10^5$	$> 10^9$	$> 10^{15}$	10^6-10^{12}
Multilevel capacity	No	No	Yes	Yes	Yes	Yes	Yes
Non-volatility	No	No	Yes	Yes	Yes	Yes	Yes

A qualitative and quantitative comparison of the different performance metrics on different memory types are displayed in Figure 1.17 and Table 1.1 respectively.

The particular case of RRAM, in comparison with its technological competitors, shows a clear advantage and potential in terms of scalability, speed and low power consumption when compared to PCM or STT-RAM for instance. However, the major challenges for RRAM still lie in terms of reliability (retention and endurance) and variability. Taking these considerations in mind, this thesis will have particular emphasis on

exploring the impact of RRAM variability and solutions towards mitigating its effects on neuromorphic systems.

1.2 Learning algorithms for neuromorphic systems

1.2.1 Machine learning concepts

Neuromorphic computing, as well as deep learning are specific use cases that stem in a more general sense from ML. In this section an overview of relevant ML concepts in the scope of this thesis will be given.

A machine learning algorithm is an algorithm capable of learning from data. Mitchell provides a succinct definition: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E” [110].

1.2.1.1 The task T

ML algorithms should generally be conceived with the purpose of solving a particular task or problem. In this definition of ”task”, the learning process itself is not the task, but learning is rather the means towards attaining the ability to perform said task.

ML tasks can usually be described in terms of how a learning system should process an example. An example is a collection of features that have been quantitatively measured from an object or even meant to be processed by the ML system. The example is

typically represented as a vector $x \in \mathbb{R}^n$ where each entry x_i of the vector is one feature. For example, the features of an image are usually the values of each pixel of that image.

ML tasks can be divided into several categories. Below are listed some of the most common ML tasks:

- **Classification:** In this task, the ML algorithm is asked to specify which of k categories a given input belong to. To solve this task, the algorithm must produce a function $f : \mathbb{R}^n \to \{1,\ldots,k\}$. The classification model can directly output a numeric code that can be translated to the category, or could give a probability distribution over the different classes. A common example of a classification task is pattern recognition, where the input is a vector x containing the pixel brightness values and the output is a numeric code that can be decoded into different classes. The classification task will be the most emphasized within the scope of this thesis work.

- **Regression:** To solve this task, the algorithm must output a function $f : \mathbb{R}^n \to \mathbb{R}$, such that, given a set of real numbers, a numerical prediction is found. Common examples of ML regression tasks are financial or stock market predictions based on previously learned data.

- **Transcription:** This task consists on the transformation of relatively unstructured representations of some kind of data into discrete textual form. A common example of this is speech recognition where the algorithm must transform an audio waveform into a sequence of characters that describe the words spoken in the audio recording [111].

- **Machine translation:** Machine translation is similar to transcription, however, instead of converting from an unstructured data type to text, translation simply converts a sequence of symbols or text in one language to another, such as translating from English to Spanish [112, 113].

- **Structured output:** Structured output tasks involve any task where the output is a vector with important relationships between the its different elements. This is a broad category that can include but is not limited to the above examples of transcription and translation. A different example is that of parsing, a natural language process that breaks down sentences into trees that describe its grammatical structure [114].

- **Anomaly detection:** In this type of task, the algorithm must go through a set of events or objects and flag some of them as unusual or atypical [115]. An example of this would be credit card fraud detection through the modelling of the owner's purchasing habits.

- **Synthesis and sampling:** The goal of this task is to generate new examples similar to those on the training data. This can be extremely useful in situations when generating large amounts of content by hand would be expensive and cumbersome, like in media applications such as video game or computer generated graphic design [116]. Synthesis could be viewed as another form of structured output task with the added characteristics that there is no single correct output for each input, and large amounts of output variation is desired in order to emulate more natural and realistic results.

- **Imputation of missing values:** In this task, the ML algorithm is given a new example $x \in \mathbb{R}^n$, but with some entries x_i of the input vector x missing. The goal is to provide the prediction of the missing entries.

- **Denoising:** In this type of task, the ML algorithm is given an input of a corrupted example $\tilde{x} \in \mathbb{R}^n$ obtained by an unknown corruption process from a clean example $x \in \mathbb{R}^n$. The algorithm must predict the clean example x from the corrupted version $\tilde{x}$, or predict a conditional probability distribution $p(x|\tilde{x})$

- **Density estimation:** In the density estimation problem, the ML algorithm must learn a function $p_{model} : \mathbb{R}^n \rightarrow \mathbb{R}$, where $p_{model}(x)$ can be interpreted as a probability density function(PDF) if x is continuous, or a probability mass function if x is discrete. Most of the tasks described previously may in one form or another require the algorithm to implicitly capture the structure of the probability distribution. In this case however, the goal is to explicitly capture that distribution. In principle, this would allow the algorithm to make predictions of a given dataset, based on $p(x)$. This could be specially useful for example for the missing value imputation or denoising tasks.

1.2.1.2 The performance measure P

Taking into consideration Mitchell's definition of ML, a quantitative measure of performance (also called cost or loss function) is necessary for both the system end-user as an analysis tool of how well the ML algorithm performs, but also for the system itself to use recursively in the learning process, in other words, the continuous improvement of a specified performance measure is what drives the ML algorithm to learn. Usually this

performance measure P should be chosen specifically for the task T that is to be carried out.

In classification tasks, for instance, the accuracy (or error rate) is often measured as performance metric, being that the accuracy simply refers to the proportion of examples that the model correctly classifies. This type of error rate can be referred to as a 0-1 loss, as the error rate for a specific example is 0 if incorrectly classified or 1 if correctly classified. At first this may seem as a straightforward means of choosing a performance measure, however, it is often the case that the penalties imposed on the ML algorithm by this kind of binary decision can be too harsh on the learning experience (on gradient descent algorithms for instance), where the optimization could easily get stuck on saddle points or local minima and optimization stops early. Taking this point into consideration, even though the accuracy is the main quantitative measure of interest for the end-user, a different cost function is often used by the ML algorithm to smooth out the penalties of incorrect classifications to avoid local minima. One common example of a suitable cost function for classification is the cross-entropy function, where the relative entropy between different probability distributions is calculated.

Below, the definitions of some commonly used performance measures in classification will be given. The first 5 are qualitative and the remaining are probabilistic.

For the definitions presented below, the following notation should be considered [117]. Given a dataset, m denotes the number of examples, and c the number of classes. $f(i,j)$ represents the actual probability of example i to be of class j. It is assumed that $f(i,j)$ always takes values in $\{0,1\}$ and is strictly not a probability but an indicator

function. $m_j = \sum_{i=1}^{m} f(i,j)$ denotes the number of examples of class j. $p(j)$ denotes the prior probability of class j, i.e., $p(j) = m_j/m$.

Given a classifier, $p(i,j)$ represents the estimated probability of example i to be of class j taking values in [0,1]. $C_\theta(i,j)$ is 1 if j is the predicted class for i obtained from $p(i,j)$ using a given threshold θ. Otherwise $C_\theta(i,j)$ is 0. θ will be omitted below.

Acc **Accuracy:** This is the most common and simplest measure to evaluate a classifier. It is simply defined as the amount of correct predictions of a model (or conversely, the percentage of misclassification errors)

$$Acc = \frac{\sum_{i=1}^{m}\sum_{j=1}^{c} f(i,j)C(i,j)}{m} \text{ [117]} \tag{1.3}$$

KapS **Kappa statistic:** Originally a measure of agreement between classifiers [118], it can also be applied as a classifier performance measure [119], or for estimating the similarity between the members of an ensemble in Multi-classifiers Systems [120]

$$KapS = \frac{P(A) - P(E)}{1 - P(E)} \text{ [117]}, \tag{1.4}$$

where $P(A)$ is the relative observed agreement among classifiers and $P(E)$ is the probability that agreement is due to chance. $P(A)$ can be simply defined as the classifier accuracy, i.e. $P(A) = Acc$ as previously defined, and $P(E)$ is defined as:

$$P(E) = \frac{\sum_{k=1}^{c}([\sum_{j=1}^{c}\sum_{i=1}^{m} f(i,k)C(i,j)] \cdot [\sum_{j=1}^{c}\sum_{i=1}^{m} f(i,j)C(i,k)])}{m^2} \text{ [117]} \tag{1.5}$$

MAvA **Macro average arithmetic:** Defined as the arithmetic average of the partial accuracies of each class [110].

$$MAvA = \frac{\sum_{j=1}^{c} \frac{\sum_{i=1}^{m} f(i,j)C(i,j)}{m_j}}{C} \text{ [117]} \tag{1.6}$$

MAvG **Macro average geometric:** Defined as the geometric average of the partial accuracies of each class.

$$MAvG = \sqrt[c]{\prod_{j=1}^{c} \frac{\sum_{i=1}^{m} f(i,j)C(i,j)}{m_j}} \text{ [117]} \tag{1.7}$$

MAPR **Macro Average Mean Probability Rate:** Computed as an arithmetic average of the mean predictions for each class [110].

$$MAPR = \frac{\sum_{j=1}^{c} \frac{\sum_{i=1}^{m} f(i,j)p(i,j)}{m_j}}{c} \text{ [117]} \tag{1.8}$$

MPR **Mean Probability Rate:** A measure that analyses the deviation from the true probability. It is a non-stratified version of MAPR, being the arithmetic average of the predicted probabilities [121].

$$MPR = \frac{\sum_{j=1}^{c} \sum_{i=1}^{m} f(i,j) - p(i,j)}{m} \text{ [117]} \tag{1.9}$$

MAE **Mean Absolute Error:** Shows how much the predictions deviate from the true probability.

$$MAE = \frac{\sum_{j=1}^{c} \sum_{i=1}^{m} |f(i,j) - p(i,j)|}{m \cdot c} \text{ [117]} \tag{1.10}$$

MSE **Mean Squared Error:** A quadratic version of MAE, which penalises strong deviations from the true probability. Can also be referred to as Brier score [122].

$$MSE = \frac{\sum_{j=1}^{c} \sum_{i=1}^{m} \left(f\left(i,j\right) - p\left(i,j\right)\right)^2}{m \cdot c} \text{ [117]} \tag{1.11}$$

CE **Cross Entropy:** This is a measure of how good the probability estimates are (also known as LogLoss) and it is commonly used when calibration is an important factor [123, 124].

$$CE = \frac{-\sum_{j=1}^{c} \sum_{i=1}^{m} \left(f(i,j) \log_2 p(i,j)\right)}{m} \text{ [117]} \tag{1.12}$$

1.2.1.3 The experience E

The experience E can be referred to as the dataset that the ML algorithm is intended to learn. In some cases, a ML algorithm may be allowed to experience an entire dataset, while in others, a dataset may be split into training and test data. Training data is the data that the algorithm is allowed to use as learning data to continuously improve its performance, while test data can be thought of as ”unseen” data to evaluate how well a given ML model fits to general data outside of the training data.

In this sense it is also important to make the distinction between unsupervised and supervised learning experiences.

Unsupervised learning algorithms involves the observation of several examples of a random vector x and attempting to either implicitly (e.g. density estimation) or explicitly (e.g. synthesis, denoising, clustering, etc.) learn the probability distribution $p(x)$,

or some interesting properties of that distribution.

Supervised learning, on the other hand, aims at learning by observing several examples of a random vector x that has an associated label or target value or vector y, then learning to predict y from x, usually by estimating $p(y|x)$. The term supervised learning comes from the view that the target y is provided by an instructor that shows the ML system what to do. In unsupervised learning, however, there is no such instructor, and the algorithm must learn to make sense of the data without guidance.

The line between unsupervised and supervised learning can often be blurred, since both methods can be adopted by the same ML technology. For instance, the chain rule of probabilities states that for a vector $x \in \mathbb{R}^n$, the joint distribution can be decomposed as:

$$p(x) = \prod_{i=1}^{n} p\left(x_i | x_1, \ldots, x_{i-1}\right) \text{ [125]} \tag{1.13}$$

meaning that an unsupervised learning problem of modeling $p(x)$ can be split into n supervised learning problems. Alternatively, a supervised learning problem of learning $p(y|x)$ can be solved using unsupervised learning technologies to learn the joint distribution $p(x,y)$, inferring:

$$p(y|\mathbf{x}) = \frac{p\left(\mathbf{x}, y\right)}{\sum_{y'} p\left(\mathbf{x}, y'\right)} \text{ [125]} \tag{1.14}$$

Nonetheless, unsupervised and supervised learning concepts, despite not being formally distinct, traditionally have different applications; regression, classification and structured output problems are generally thought of as more suitable for supervised

learning, while density estimation, for example, is generally considered as unsupervised learning.

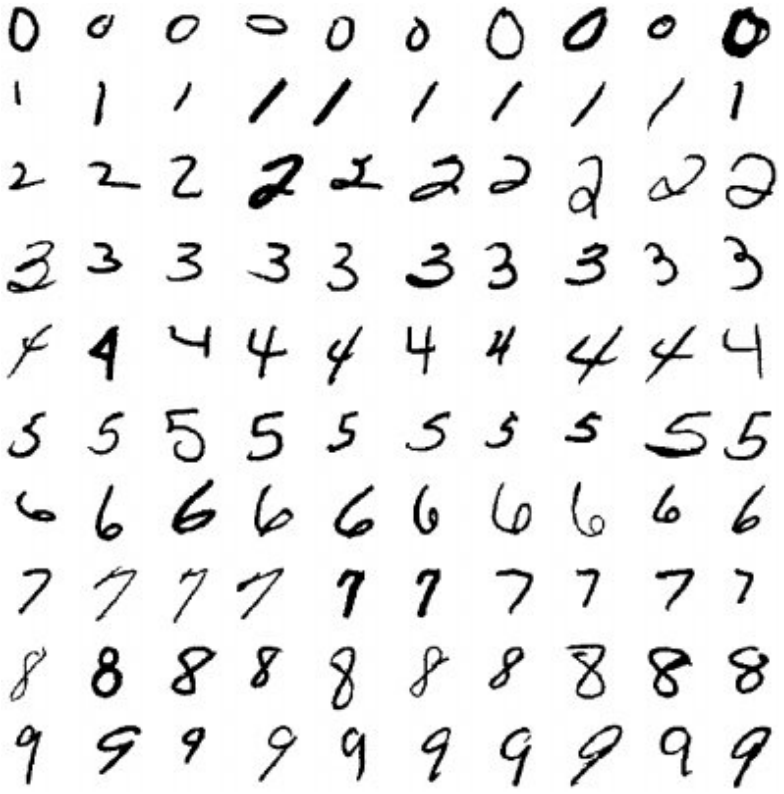


Figure 1.18: Sample images of MNIST database [126].

The main focus of this thesis work will be on supervised learning techniques applied to the classification of the Modified National Institute of Standards and Technology (MNIST) handwritten digits database [126] (Figure 1.18).

1.2.1.4 Gradient-based learning

Most ML algorithms achieve their task by way of optimization (minimizing or maximizing) of some cost function $f(x)$ by altering x (already discussed in subsubsection 1.2.1.2). Gradient-based optimization achieve minimization of a cost function $y = f(x)$ by using the derivative $f'(x)$ to move x in small increments to improve y. For small

enough η, $f(x - \eta sign(f'(x)))$ is less than $f(x)$, so $f(x)$ can be reduced by moving x in small steps with the opposite sign of the derivative. This method is called gradient descent [127].

In most ML cases, the learning problem comes in a multi-dimensional form (multiple inputs), as such, the notion of gradient descent can be extended to a multi-dimensional space by making use of partial derivatives. The partial derivative $\frac{\partial}{\partial x_i} f(x)$ measure how f changes as only the variable x_i increases at point x. The gradient generalizes the notion of derivative to the case where the derivative is with respect to a vector, being that the gradient of f is the vector containing all its partial derivatives, denoted as $\nabla_x f(x)$.

The directional derivative in direction u (a unit vector) is the slope of the function f in direction u. In other words, the directional derivative is the derivative of the function $f(x + \alpha u)$ with respect to α, evaluated at $\alpha = 0$. Using the chain rule, we can see that $\frac{\partial}{\partial \alpha} f(x + \alpha u)$ evaluates to $u^\top \nabla_x f(x)$ when $\alpha = 0$.

So, to minimize f, the direction in which f decreases the fastest should be found:

$$\min_{u, u^\top u = 1} u^\top \nabla_x f(x) \text{ [125]} \tag{1.15}$$

$$= \min_{u, u^\top u = 1} \|u\|_2 \|\nabla_x f(x)\|_2 \cos\theta \text{ [125]} \tag{1.16}$$

where θ is the angle between u and the gradient. Substituting in $\|u\|_2 = 1$ and ignoring factors that do not depend on u, simplifies to $\min_u \cos\theta$. This is minimized when u points in the opposite direction as the gradient. In other words, the gradient points

directly uphill, and the negative gradient points directly downhill. As such, for multi-dimensional spaces, the method of steepest descent is defined as:

$$x' = x - \eta \nabla_x f(x) \, [125] \tag{1.17}$$

where η is the learning rate, a positive scalar that determines the size of each step. The learning rate is an hyper-parameter of the ML algorithm that should be adjusted for each individual problem. Choosing too high of a value results in large steps that may never converge towards the optimal solution, while choosing too low values could make the algorithm to become stuck in suboptimal local minima or saddle points. One approach to determine the learning rate is to evaluate $f(x - \eta \nabla_x f(x))$ for several values of η and choosing the one that results in the smallest objective function value. This method is called a line search.

With the introduction of the gradient descent method for solving complex problems that rely on large training sets, a new issue became apparent: limited computing power. The cost function of a ML algorithm generally decomposes as the sum over all training examples. The computational cost of this operation is a function $O(m)$, where m is the number of training examples. As the training set size grows to billions of examples, the time to take a single gradient step may become impractical.

As a solution for this issue came the Stochastic Gradient Descent (SGD) algorithm [128]. SGD comes as an extension of the traditional gradient descent algorithm, with the main difference that the calculated gradient is an expectation estimated from a small set of training samples called the minibatch. On each gradient calculation step of the

algorithm, a minibatch of examples $\mathbb{B} = \{x^{(1)}, \ldots, x^{(m')}\}$ is drawn uniformly from the training set. The minibatch size m' is another algorithm hyperparameter and is usually set to be lower than a few hundred of examples. Crucially, m' is usually a fixed number while the size of the training set m may grow to very large numbers.

The estimate of the gradient is formed as:

$$g = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} L\left(x^{(i)}, y^{(i)}, \theta\right) \text{ [125]} \tag{1.18}$$

using examples from the minibatch $\mathbb{B}$ and L is the loss function to be calculated. The SGD algorithm then follows the estimated gradient downhill:

$$\theta \longleftarrow \theta - \eta g \text{ [125]} \tag{1.19}$$

The SGD algorithm can therefore be summarized as:

Algorithm 1 Stochastic gradient descent (SGD) update at training iteration k.

Require: Learning rate η_k
Require: Initial parameter θ
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with the corresponding targets $y^{(i)}$.
Compute gradient estimate: $\hat{g} \leftarrow +\frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$.
Apply update: $\theta \leftarrow \theta - \eta \hat{g}$
end while

While SGD remains a popular training algorithm, further optimization in terms of learning speed can be made. SGD with momentum (SGDM) [129] is designed to accelerate learning facing high curvatures, small or noisy gradients. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and

continues to move in their direction.

A variable v that plays the role of velocity is introduced. Similar to the velocity of a ball rolling down a slope, the velocity v in this context can be seen as the direction and speed at which the parameters move through the parameter space. v is set to an exponentially decaying average of the negative gradient. In the momentum algorithm, unit mass is assumed, so the velocity vector v may also be regarded as the momentum of the particle. A hyperparameter $\alpha \in [0,1)$ determines how quickly the contributions of previous gradients exponentially decay. As such, the update rule is given by:

$$v \leftarrow \alpha v - \eta \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L\left(f\left(x^{(i)}; \theta \right), y^{(i)} \right) \right) \text{ [125]}, \tag{1.20}$$

$$\theta \leftarrow \theta + v \text{ [125]} \tag{1.21}$$

and the SGDM algorithm can be summarized as such:

Algorithm 2 Stochastic gradient descent with momentum (SGDM).

Require: Learning rate η, momentum parameter α
Require: Initial parameter θ, initial velocity v
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient estimate: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$.
Compute velocity update: $v \leftarrow \alpha v - \eta g$.
Apply update: $\theta \leftarrow \theta + v$.
end while

Using momentum, the step size will depend on the magnitude and direction of a sequence of gradients. If the momentum algorithm always observes gradient g, then it

will accelerate in the direction of $-g$, until reaching a terminal velocity where the size of each step is:

$$\frac{\eta\|g\|}{1-\alpha}\,[125] \tag{1.22}$$

Therefore, the momentum hyperparameter should be thought of in terms of $\frac{1}{1-\alpha}$. Common practical values of α include 0.5, 0.9 and 0.99.

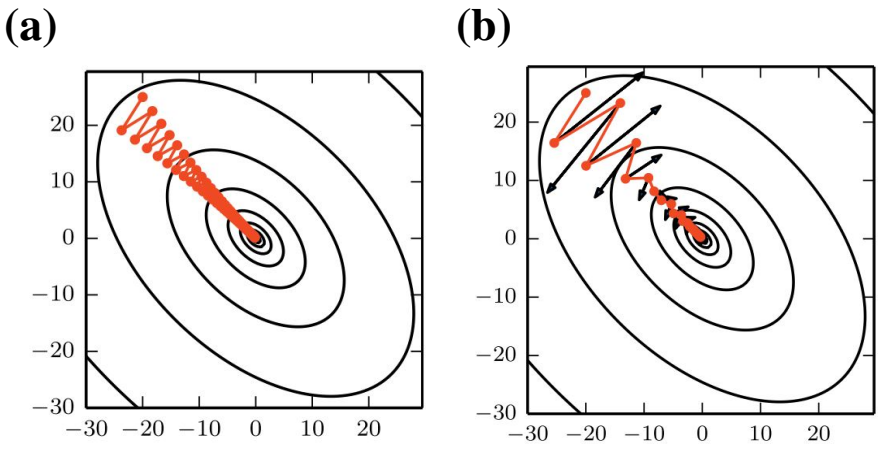


Figure 1.19: Illustration of the path followed by a gradient descent algorithm in a two-parameter hyperspace. (a) typical gradient descent, (b) gradient descent with momentum. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path indicates the path followed by the algorithm. The black arrows in (b) indicate the step that would be taken at each point without momentum.

1.2.1.5 Feedforward Networks

Feedforward NNs can be considered as the basis of many deep learning models. A feedforward NN can be defined as a network of functions that aims to approximate some function f^*. In the case of a classifier, $y = f^*(x)$ maps an input x to a category y. A feedforward NN can define the mapping as $y = f(x;\theta)$, where the values of the parameters θ that result in the best function approximation are learned by the NN itself.

These types of models are called feedforward because the process of evaluating $f^*(x)$ only involves the flow of information in one direction: from the input x to the output y; if feedback connections from the output of the model are fed back into itself, the model is defined as a recurrent NN (RNN).

Feedforward NN are called networks because they are typically represented by chaining together many different functions. A typical example could be formed with three functions: $f^{(1)}$, $f^{(2)}$, $f^{(3)}$ connected in a chain, to form: $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. In this case, $f^{(1)}$ corresponds to the NN first layer, $f^{(2)}$ to the second layer, and so forth. The overall length of this chain gives the depth of the model. The first layer of a feedforward NN is called the input layer and is typically fixed by the NN problem, for example, in an image classification problem the input layer corresponds to the values of the pixels that make up the image, while the final layer is the output layer, using the previous example, the output would be each category that an image may belong to. Any layers between the input and output are referred to as hidden layers.

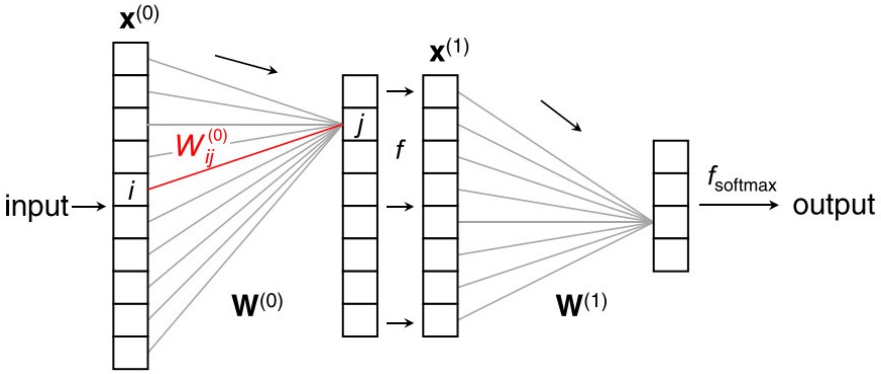


Figure 1.20: Schematic illustrating the forward propagation process in a single hidden layer MLP [130].

A common example of a feedforward NN is one where all previous nodes are connected to all subsequent nodes (fully connected NN). Nomenclature for the components

of NN are inspired in biology, as such, the connections between nodes are referred to as "synapses" or "weights" and each node is defined as a "neuron". In the case that a fully connected NN has at least 1 hidden layer, the NN is designed as a multilayer perceptron (MLP) [131, 132], this type of network was one of the first designed architectures for deep learning and remains as one of its basis due to its structural simplicity.

Figure 1.20 illustrates the forward propagation step in a MLP. The input neuron activations $\mathbf{x}^{(l)}$ to layer l are converted to the next layer's activations $\mathbf{x}^{(l+1)}$ by the transformation:

$$\mathbf{x}^{(l+1)} = f\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)}\right) \text{ [130]}, \tag{1.23}$$

where $\mathbf{W}^{(l)}$ is an $N_l \times N_{l+1}$ matrix of weights connecting layer l with N_l neurons to layer $l+1$ with N_{l+1} neurons. f is the nonlinear activation function of the neuron that is applied element-wise to its argument, which is the product of a VMM.

The function of the neuron f is essential in ensuring that the multiple layers in the network cannot be collapsed into an equivalent single-layer linear network. Table 1.2 shows typical examples of commonly used neuron functions in deep learning.

While the activation functions presented in Table 1.2 are common use for the hidden layers of a NN, the use of a different function called softmax is more common use in the output layer of classifiers to represent a probability distribution over n different classes. In the case of binary variables a predictor $\hat{y}$ can be obtained by:

$$\hat{y} = P(y = 1|x) \text{ [125]}, \tag{1.24}$$

Table 1.2: Different types of mathematical neurons [133].

Neuron/function	Plot	Equation	Derivative	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\inf, \inf)$
Binary step		$f(x) = \begin{cases} 0\, for x < 0 \\ 1\, for x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0\, for x \neq 0 \\ ?\, for x = 0 \end{cases}$	$(0,1)$
Logistic		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	$(0,1)$
Tanh		$f(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1,1)$
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$(-\frac{\pi}{2}, \frac{\pi}{2})$
Softsign		$f(x) = \frac{x}{1+\lvert x\rvert}$	$f'(x) = \frac{1}{(\lvert x\rvert+1)^2}$	$(-1,1)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0\, for x < 0 \\ x\, for x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0\, for x < 0 \\ 1\, for x \geq 0 \end{cases}$	$[0, \inf)$
Leaky ReLU		$f(\alpha, x) = \begin{cases} \alpha x\, for x < 0 \\ x\, for x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha\, for x < 0 \\ 1\, for x \geq 0 \end{cases}$	$(-\inf, \inf)$
Exponential Linear Unit (ELU)		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1)\, for x < 0 \\ x\, for x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(x) + \alpha\, for x < 0 \\ 1\, for x \geq 0 \end{cases}$	$(-\alpha, \inf)$
SoftPlus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{e^{-x}+1}$	$(0, \inf)$

which for gradient-based optimization of the cross-entropy function, can be translated into:

$$z = \log \tilde{P}(y = 1|x) \text{ [125]} \tag{1.25}$$

To generalize to the case of a discrete variable with n values, a vector $\hat{\mathbf{y}}$ should be produced, with $y_i = P(y = i|x)$, and $z_i = \log \tilde{P}(y = i|x)$. The softmax function can then exponentiate and normalize $\mathbf{z}$ to obtain the desired $\hat{\mathbf{y}}$. Formally, softmax is defined as:

$$softmax(\mathbf{z}_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \text{ [125]} \tag{1.26}$$

The softmax function can therefore be seen as the generalization of the logistic function across multiple classes.

1.2.1.6 Backpropagation

Subsubsection 1.2.1.5 described the means through which to obtain a predictor output $\hat{\mathbf{y}}$ by propagating an input $\mathbf{x}$ forward through a series of weights $\mathbf{W}$ and neuron functions f in a process called forward propagation or inference.

The inference process is that by which a trained network can achieve its goal. However a NN is required to train its weight matrices to produce the best generalized output.

Forward propagation continues onwards until a scalar cost is produced at the network output. The backpropagation algorithm [134] (often referred to as simply backprop) extends the use of the forward propagation step into a methodology that allows for the training of the NN weights by allowing the information from the cost to flow backward through the network in order to compute the gradients in each layer.

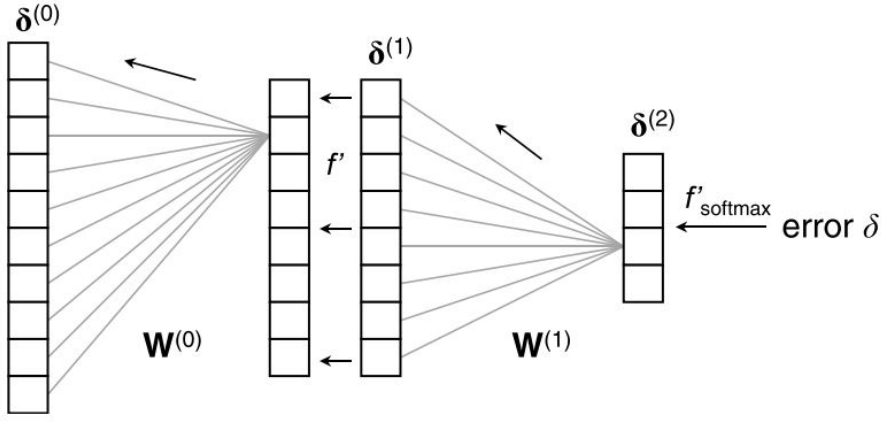


Figure 1.21: Illustration of the backpropagation process in a single hidden layer MLP [130].

For a given example, after the inference step, an error δ is obtained (typically through MSE or CE loss function) based on the known correct output. By differentiating equation 1.23, the error at the output layer can be propagated backwards (Figure 1.21) by recursively applying the chain rule of calculus through the NN layers to find the error values at a layer l:

$$\delta^{(l)} = \left(\mathbf{W}^{(l)}\right)^{\mathrm{T}} \delta^{(l+1)} \odot f'\left(\mathbf{W}^{(l-1)}\mathbf{x}^{(l-1)}\right) \text{ [130]}, \tag{1.27}$$

where $\odot$ denotes an element-wise product. This expression involves the multiplication of the transpose of the weight matrix with an error vector, so it no longer refers to a VMM operation as in equation 1.23, but now involves a MVM operation.

The term backpropagation is often misconstrued as meaning the whole learning algorithm for training MLPs, when in reality, backprop refers only to the method for computing the gradient, while another algorithm, such as SGD or SGDM must be used to perform the learning using this computed gradient.

From the error vectors, the derivative with respect to the prediction error δ can be found for all the weights, and these derivatives can be used to update the weights according to an optimization algorithm, SGD for instance. The weight update can formally be described as:

$$\blacksquare\mathbf{W}^{(l)} = -\eta\,\delta^{(l)}\left(\mathbf{x}^{(l)}\right)^{\mathrm{T}} \text{ [130]}, \tag{1.28}$$

since η is a scalar, the weight update is in essence the outer product of two vectors.

1.2.1.7 Challenges in optimization

The central issue in ML is that the algorithm must perform well on new (previously unseen) inputs, and not just those on which the model was trained. This ability to perform well on unobserved inputs is referred to as **generalization**. What separates ML from a simple optimization problem is the notion that beyond minimizing the training errors, the **generalization** or **test** error must also be reduced as close as possible to the training error.

Generalization spawns two major opposing challenges in ML: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. On the other hand, overfitting occurs when the trained algorithm no longer generalizes well to unseen data, resulting in low errors on training set and large errors on the test data.

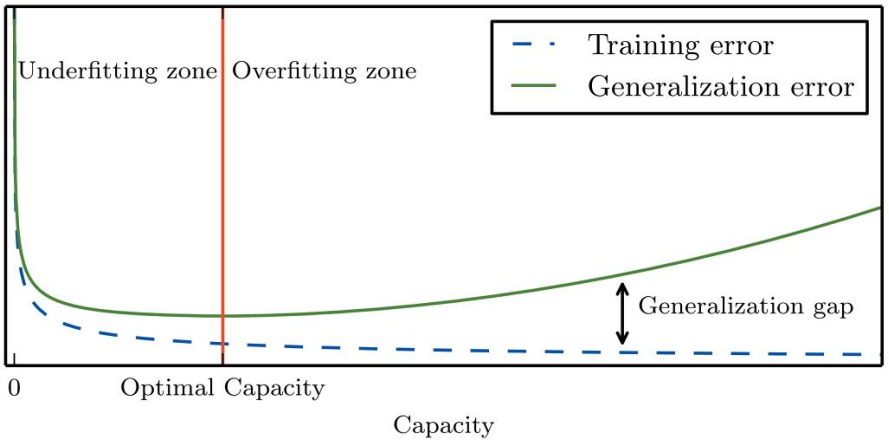


Figure 1.22: Typical relationship between capacity and error. Training beyond the point of optimal capacity may lead to degradation of the test errors despite improvements on the training dataset [125].

The likelihood of underfitting or overfitting can be controlled by altering the model's **capacity** (Figure 1.22). A model's capacity can be thought of as its ability to fit a wide variety of functions. One way to control the capacity of a learning algorithm is

by choosing its **hypothesis space**, the set of functions that the algorithm is allowed to select as a solution. For instance, linear regression can be generalized as polynomials of degree k, such that:

$$\hat{y} = b + \sum_{i=1}^{k} w_i x^i \, [125] \tag{1.29}$$

Choosing a low value of k will result in a low degree polynomial incapable of fitting the training data, while using a high k will allow the algorithm to overfit, probably rendering it unable to fit unseen data (Figure 1.23).

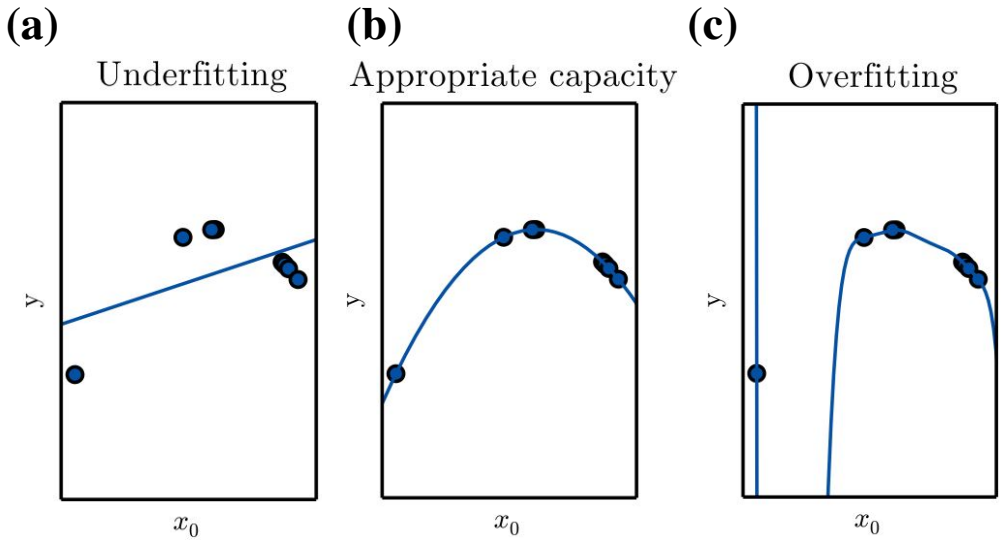


Figure 1.23: Examples of (a) underfitting, (b) appropriate fitting and (c) in a single parameter hyperspace [125].

Another challenge in NN optimization is that of an **ill-conditioned** Hessian matrix **H**, where **H** is the matrix containing all of a function f second derivatives and can be seen as its curvature. This issue can manifest itself on a SGD algorithm by causing it to get ”stuck”, meaning that even very small steps can increase the cost function.

A second-order Taylor series expansion of the cost function predicts that a gradient descent step of $-\eta\mathbf{g}$ will add:

$$\frac{1}{2}\eta^2\mathbf{g}^\top\mathbf{H}\mathbf{g} - \eta\mathbf{g}^\top\mathbf{g}\,[125] \tag{1.30}$$

to the cost. The ill-conditioning problem becomes prevalent when $\frac{1}{2}\eta^2\mathbf{g}^\top\mathbf{H}\mathbf{g}$ exceeds $\eta\mathbf{g}^\top\mathbf{g}$. To determine if the ill-conditioning problem will be detrimental to a NN task, the squared gradient norm $\mathbf{g}^\top\mathbf{g}$ and the $\mathbf{g}^\top\mathbf{H}\mathbf{g}$ term can be monitored. In most cases, the gradient norm does not shrink significantly through training, but the $\mathbf{g}^\top\mathbf{H}\mathbf{g}$ term grows by more than an order of magnitude. This results in slow learning despite strong gradients because the learning rate must be shrunk to compensate for the even stronger curvature.

Local minima can be considered one of the most common challenges in NN optimization. The goal of a gradient descent algorithm is to find the global minimum, however, in NNs with the increase of dimensionality and size, local minima points become more prominent throughout the hypothesis space. Nevertheless, the appearance of local minima will only pose a problem for gradient-based learning if these points have a large cost in comparison to the global minima (Figure 1.24).

Even though the problem of local minima is an active area of research, if the NN size is large enough, then most local minima should have a low enough cost value that finding the true global minima is no longer important, and finding a local minima point with a close enough cost to that global minima is now a viable solution [135–138].

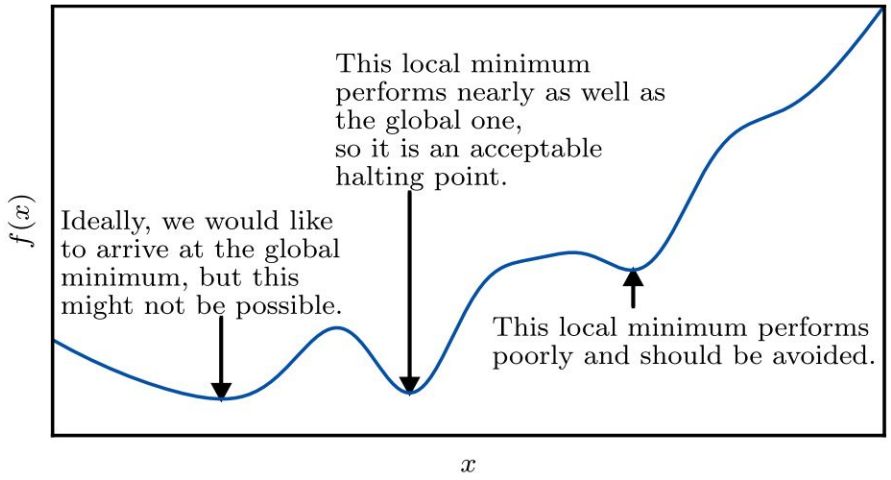


Figure 1.24: Example illustrating a high cost local minima point (right) and low cost (center) local minima point in comparison to the function global minima(left).

Beyond minima, there are other types of critical points characterized by a zero slope: **maximum** and **saddle points** (Figure 1.25).

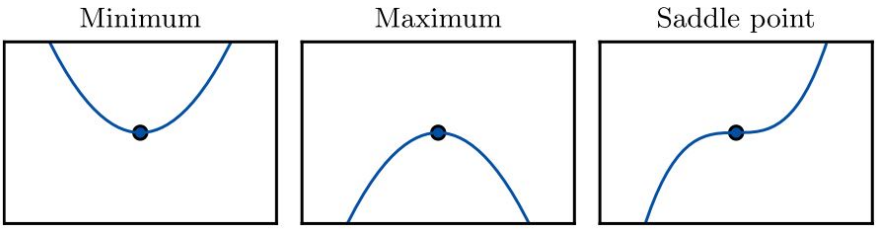


Figure 1.25: Types of critical points.

Contrasting with minima and maxima, the Hessian matrix of saddle points contain both positive and negative eigenvalues, meaning that points along eigenvectors associated with positive eigenvalues have a greater cost than the saddle point, while points lying along negative eigenvalues have a lower cost. Because saddle points have both positive and negative eigenvalues, the presence of saddle points in detriment to minima or maxima increases with dimensionality. For instance, for a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the expected ratio of the number of saddle points to local minima increases exponentially with n. The intuition behind this behaviour can be thought of as having a coin toss

decide the sign of each eigenvalue of the Hessian matrix. In this sense, the likelihood of generating n all positive or all negative eigenvalues decreases as n grows.

The implications of the appearance of multiple saddle points are similar to the local minima problem. In first-order optimization algorithms, the gradient could become very small and eventually vanish. Dauphin et al. [136] introduced a **saddle-free Newton method** for second-order optimization, even though more efficient than the traditional method, it is a complex solution for scaling in large NNs.

NNs with increased depth may often have steep **cliffs** that result from the multiplication of several large weights together. These cliff regions can be a challenging problem, as the gradient may become unreasonably large that can catapult the parameters away from an optimal solution (Figure 1.26a). One solution to this problem may be to clip the gradients to a predefined threshold (Figure 1.26b).

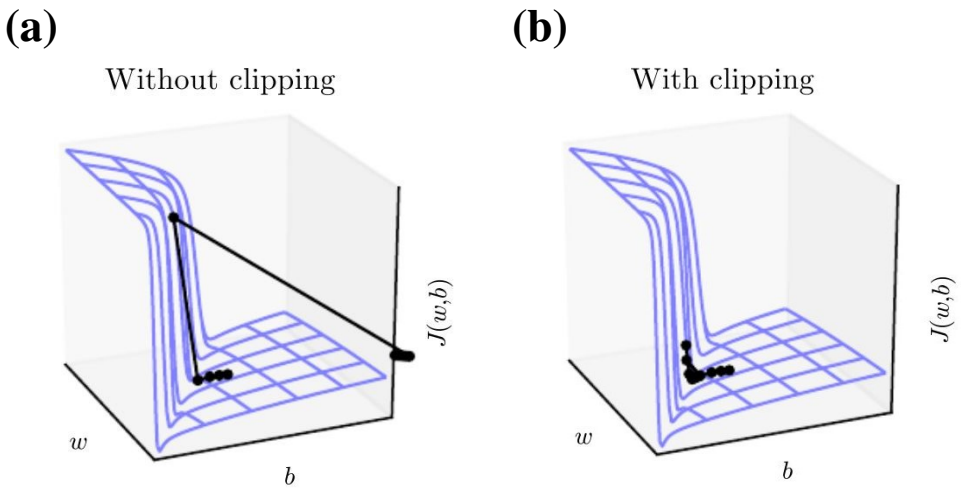


Figure 1.26: (a) An example of the exploding gradient problem without clipping. The gradient overshoots the ravine then receives a very large gradient from the cliff face that propels the parameters outside the axes of the plot. (b) The same example with gradient clipping, the step size is restricted to avoid increasing the gradients to very large values [125].

Another issue that comes with depth is the **vanishing and exploding gradient problem**. A high depth NN may involve repeated application of the same parameters. For instance, supposing that a NN contains a path that consists of repeatedly multiplying by a matrix $\mathbf{W}$. After t steps, this is equivalent to multiplying by $\mathbf{W}^t$. Assuming that $\mathbf{W}$ has an eigendecomposition $\mathbf{W} = \mathbf{V}\,\mathrm{diag}(\lambda)^t\mathbf{V}^{-1}$:

$$\mathbf{W}^t = \left(\mathbf{V}\,\mathrm{diag}(\lambda)\mathbf{V}^{-1}\right)^t = \mathbf{V}\,\mathrm{diag}(\lambda)^t\mathbf{V}^{-1}\,[125] \tag{1.31}$$

Following the above equation, any eigenvalues λ_i that are not near an absolute value of 1 will either **explode** if greater than 1, or **vanish** if less than 1. Vanishing gradients may make it difficult to know which direction the parameters should move to minimize the cost function eventually halting learning, while exploding gradients may make learning unstable.

1.2.1.8 Optimization techniques for deep models

To counteract some of the challenges in learning presented in the previous section, some optimization techniques for deep learning will also be presented. In this section a brief overview of the following concepts will be given:

- **Batch normalization**
- **Regularization**
- **Parameter initialization strategies**
- **Adaptive learning rates**

Batch normalization (BN) [139] comes as not exactly an optimization algorithm, but rather as a methodology of adaptive reparametrization, motivated by the difficulty of training deep NNs. Deep models involve the composition of several layers. The gradients contain the instructions on how to update each parameter, assuming that the other layers do not change, but in practice all layers are updated simultaneously, causing unexpected results in deep NNs (≥ 3 layers). Second-order optimization algorithms somewhat address this issue by taking these second-order interactions into account but since the issue scales with the NN depth, building n-th order optimization algorithms for $n > 2$ layers becomes intractable.

Addressing this issue, the reparametrization from BN reduces the problem in coordinating updates across many layers. BN can be applied to any input or hidden layer of a NN. In this context, defining H as a minibatch of activations of the layer to normalize, where each row of the matrix is a sample from the minibatch, H can be replaced by:

$$H' = \frac{H - \mu}{\sigma} \text{ [125]}, \tag{1.32}$$

where μ is a vector containing the mean of each unit and σ is a vector containing the standard deviation of each unit. The arithmetic here is based on broadcasting the vectors μ and σ to be applied to every row of H. At training time:

$$\mu = \frac{1}{m}\sum_i H_i \text{ [125]}, \tag{1.33}$$

$$\sigma = \sqrt{\delta + \frac{1}{m}\sum_i (H - \mu)_i^2} \text{ [125]}, \tag{1.34}$$

where δ is a small positive scalar (i.e. 10^{-8}), imposed to avoid encountering an undefined gradient of $\sqrt{0}$. Crucially, the μ and σ are backpropagated through the NN, meaning that a gradient will never propose an operation that acts solely on one layer.

At test time, μ and σ may be replaced by "moving" averages that were collected during training time, or it can be based on the entire "population" of data. Furthermore, a choice can also be made on whether BN is applied before (X) or after $(XW+b)$[139] the Multiply-accumulate (MAC) seen at the neuron input.

Another central issue in ML is that of generalization to previously unseen inputs. Strategies explicitly designed to reduce test error (sometimes even at the expense of training error) are known collectively as **regularization**. Regularization can assume many different forms, however, in the scope of this work, L^2 regularization (often also called **weight decay** or **ridge regression**) will be in focus.

The strategy behind L^2 regularization aims to drive the weights closer to the origin by adding a regularization term $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$ to the objective function.

Using this method, the regularized cost function $\tilde{J}$ is written as:

$$\tilde{J}(w;X,y) = \frac{\alpha}{2} w^\top w + J(w;X,y)\,[125], \tag{1.35}$$

with the corresponding parameter gradient:

$$\nabla_w \tilde{J}(w;X,y) = \alpha w + \nabla_w J(w;X,y)\,[125]. \tag{1.36}$$

The weight update is given by:

$$w \leftarrow w - \eta \left(\alpha w + \nabla_w J(w; X, y)\right) \text{ [125]} \tag{1.37}$$

$$w \leftarrow (1 - \eta\alpha)w - \eta \nabla_w J(w; X, y) \text{ [125]}. \tag{1.38}$$

In this way, the addition of the weight decay term α multiplicatively shrinks the weight vector by a constant factor on each step before the gradient update. Considering a regularized solution $\tilde{w}$, the purpose of weight decay is to rescale w^* along the axes defined by the eigenvectors of the Hessian H. Along the directions where the eigenvalues of H are relatively large, the effect of regularization is relatively small, while being more significant for smaller eigenvalues (Figure 1.27).

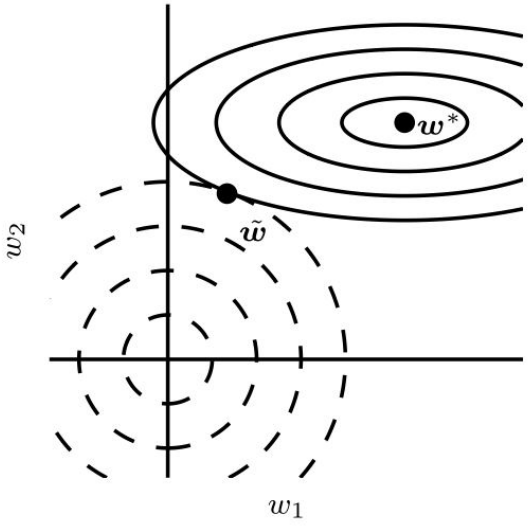


Figure 1.27: Illustration of the effect of L^2 regularization on the value of the optimal w. The solid ellipses represent contours of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer.

The training of deep models is a complex iterative task that is strongly affected by **parameter initialization**. The initial point of training may determine how fast the algorithm converges or at times if it actually converges at all. Beyond this, convergence

points of comparable cost affected by initialization can also have very different generalization errors. Adding to the complexity on initialization choice, some initial points may be beneficial in optimizing the training data, but detrimental for generalization of the test data.

Despite these complexities, one known certainty in initialization is that the parameters should be to stochastic enough to ”break symmetry” between different units. This will avoid that a deterministic learning algorithm updates different units in the same way (avoiding redundancy).

The most common method to initialize the weights in a NN is simply from choosing a Gaussian or uniform distribution from which to draw all of the weights from. In this context, choosing between a Gaussian or an uniform distribution does not seem to matter, however, the scale of these distributions has a large impact on training. Larger initial weights will yield a stronger ”symmetry-breaking” effect, helping in avoiding redundancy, as well as the loss of signal through forward and back-propagation. Nonetheless, very large weights may result in either exploding gradients or early saturation of the activation functions, causing subsequent loss of gradient through the saturated units.

Common heuristics for weight initialization include:

- **Xavier** [140]

$$W_{i,j} \sim U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right), \tag{1.39}$$

- **Normalized Xavier** [140]

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right), \tag{1.40}$$

- **He** [141]

$$W_{i,j} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{m^l}}\right), \tag{1.41}$$

where m denotes the number of inputs, and n the number of outputs of a given layer l.

The learning rate is arguably one the most difficult to set hyperparameters in NNs because of its high impact on training. To solve the issue of the difficult to set fixed learning rate, **adaptive learning rates** that adjust themselves for each parameter throughout the course of learning may be used.

The **delta-bar-delta** algorithm [142] is an early heuristic approach on adaptive learning rates. The approach is based on a simple idea: if the partial derivative of the loss, with respect to a given model parameter remains the same sign, then the learning rate should increase, if it changes signs, it should decrease. The drawback of this method is that it can only be applied to full batch optimization.

More recently, different adaptive learning rate algorithms based on minibatch processing have been introduced, such as: **AdaGrad**, **RMSProp** and **Adam**.

AdaGrad individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical values of the gradient [143]. In convex optimization, AdaGrad has some desirable theoretical

properties, however, in empirical deep NN models, the accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate.

Algorithm 3 AdaGrad algorithm.

Require: Global learning rate η
Require: Initial parameter θ
Require: Small constant δ (default = 10^{-7})
Initialize gradient accumulation variable $r = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\left\{x^{(1)},\ldots,x^{(m)}\right\}$ with corresponding targets $y^{(i)}$.
Compute gradient: $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta\right),y^{(i)}\right)$.
Accumulate squared gradient: $r \leftarrow r + g \odot g$.
Compute update: $\Delta\theta \leftarrow -\frac{\eta}{\delta+\sqrt{r}} \odot g$. (Division and square root applied element-wise)
Apply update: $\theta \leftarrow \theta + \Delta\theta$
end while

RMSProp [144] comes in as a modification of the AdaGrad algorithm for better performance in a nonconvex setting by changing the gradient accumulation into an exponentially weighted moving average, instead of shrinking the learning rate based on the entire history of the squared gradient.

Algorithm 4 RMSProp algorithm.

Require: Global learning rate η, decay rate ρ
Require: Initial parameter θ
Require: Small constant δ (default = 10^{-6})
Initialize accumulation variable $r = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\left\{x^{(1)},\ldots,x^{(m)}\right\}$ with corresponding targets $y^{(i)}$
Compute gradient: $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta\right),y^{(i)}\right)$.
Accumulate squared gradient: $r \leftarrow \rho r + (1-\rho)\, g \odot g$.
Compute parameter update: $\Delta\theta = -\frac{\eta}{\sqrt{\delta+r}} \odot g$. ($\frac{1}{\sqrt{\delta+r}}$ applied element-wise).
Apply update: $\theta \leftarrow \theta + \Delta\theta$.
end while

The **Adam** algorithm derives its name from the phrase "adaptive moments" [145]. In this context, Adam can be seen as a variant on the combination of RMSProp and momentum.

Algorithm 5 Adam algorithm.

Require: Global learning rate η (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0,1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ (Suggested default: 10^{-8}
Require: Initial parameter θ
Initialize 1st and 2nd moment variables $s = 0, r = 0$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\left\{x^{(1)},\ldots,x^{(m)}\right\}$ with corresponding targets $y^{(i)}$.
Compute gradient: $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta\right),y^{(i)}\right)$
$t \leftarrow t+1$
Update biased first moment estimate: $s \leftarrow \rho_1 s + (1-\rho_1)\, g$
Update biased second moment estimate: $r \leftarrow \rho_2 r + (1-\rho_2)\, g \odot g$
Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
Correct bias in second moment: $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
Compute update: $\Delta\theta = -\eta \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$ (operations applied element-wise)
Apply update: $\theta \leftarrow \theta + \Delta\theta$
end while

1.2.1.9 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [146] are a specialized kind of NN for processing data arranged in a grid-like topology. Examples of this type of data include: time-series data seen as 1-D grid sampled at regular time intervals, or more commonly, image data arranged as a 2-D grid of pixels.

At the heart of CNN lies the **convolutional** operation. In its most general form, convolution is an operation on two functions of a real-valued argument. The convolution

between two functions f and g can be written as $f * g$ such that:

$$s(t) = (f * g)(t) = \int f(\tau)\, g(t-\tau)\, d\tau \text{ [125]}, \tag{1.42}$$

here, the first argument (f) is often referred to as the **input**, and the second argument (g) as the **kernel**. The output of the operation ($f * g$) may be referred to as the **feature map**.

In most ML cases, it might be more realistic to work with data sampled at regular intervals. In this case, equation 1.42 can be modified to:

$$s(t) = (f * g)(t) = \sum_{\tau=-\inf}^{\inf} f(\tau)\, g(t-\tau) \text{ [125]}. \tag{1.43}$$

In ML applications, the input is usually a multidimensional array of data, and the kernel a multidimensional array of parameters that are adapted by the learning algorithm. These arrays are often referred to as **tensors**. Since each element of the input and kernel must be explicitly stored separately, it can be assumed that the functions values are zero for every element outside of the finite set of points that are stored. This means that in practice, an infinite summation can be implemented as a finite summation of the stored elements.

Extending these notions to a two-dimensional example such as an image with input I and kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n) \text{ [125]}, \tag{1.44}$$

since convolution is commutative:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)\,[125]. \tag{1.45}$$

Equation 1.45 tends to be more straightforward to implement in ML because there is less variation in the range of values of m and n.

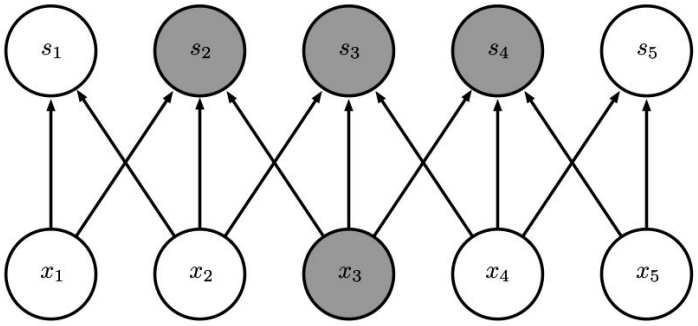
Convolution leverages three important ideas for ML systems: **sparse interactions**, **parameter sharing** and **equivariant representations**.

Contrary to fully connected NNs, using convolutional features in a NN means that not all neurons of a given layer l need to be connected to all neurons of the previous layer $l-1$. This is achieved by using a kernel smaller than the input in the convolution operation (Figure 1.28). This particular feature is designated as **sparse interactions** (often also referred to as **sparse connectivity** or **sparse weights**).

Taking an input image composed of millions of pixels for instance, means that by using sparse interactions, the focus shifts from detecting the whole image to detecting only small and meaningful features such as edges using kernels that occupy a fraction of the total image, significantly reducing the model's memory requirements but also improving its statistical efficiency.

Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional NN, each element of the weight matrix is used exactly once when computing the output of a layer, while in a CNN each member of the kernel is used at every position of the input. Parameter sharing used by the convolution operation

(a)



(b)

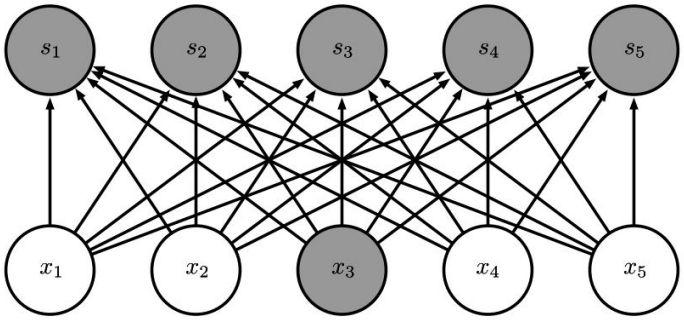


Figure 1.28: (a) Example of sparse connectivity in comparison to (b) a fully connected layer. The darkened neurons in this example show the effect of a single neuron (x_3) on its next layer for both connection schemes

means that rather than learning a separate set of parameters for every location, only one set is learned.

Using convolution with parameter sharing causes a layer to have a property called **equivariance** to translation. Equivariant functions means that if the input of a function changes, then the output changes in the same way. This is particularly useful in image recognition, where certain transformations, such as rotations or translations may be applied to an image, but the output of the image classifier will remain insensitive to these transformations.

A typical layer of a CNN consists of three stages: **convolution**, **detector** and **pooling** stages (Figure 1.29a).

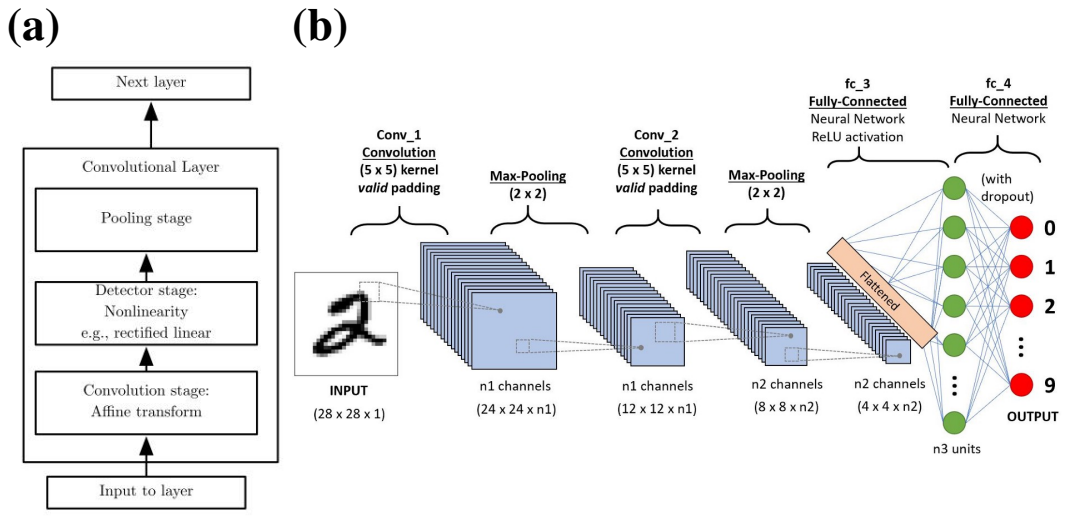


Figure 1.29: (a) An illustration of the typical composition of a CNN layer. First a convolution operation is performed, followed by the activation function and finally the pooling layer. (b) Typical example representing the architecture of a whole CNN. In image classifiers, the final layer is typically flattened and fully connected to the output.

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For instance, **max pooling** [147] reports only the maximum output within a rectangular neighbourhood. Similar methods can be used with an average, weighted average or the L^2 norm. In all cases, pooling helps to make the representation invariant to small translations of the input, meaning better generalization and resilience to input noise. Another benefit of pooling is that it allows the handling of inputs with varying size by offsetting the pooling regions in a manner that ensures that the classification layer always receives the same number of summary statistics.

1.2.1.10 Genetic Algorithms

In the previous subsections, learning using some form of gradient-based optimization has been focused on. Nonetheless, several other learning heuristics exist that deviate from that concept. One such example is that of population-based learning algorithms

that rely on changing the properties of a population of solutions via some determined metaheuristic with the goal of finding the best candidate within said population to solve the optimization problem.

One notable case of population-based optimization is genetic algorithms (GA). GA was introduced in the early 1970s by John Holland [148] and takes its inspiration in biology, specifically, Charles Darwin's theory of biological evolution, where the strongest individuals within a population survive. GAs attempt to mimic this behaviour by simulating a population of individuals, where each one of these individuals carries information (chromosome) regarding the solution to a proposed problem and the highest ranked individuals are allowed to survive in the next iteration and reproduce offspring.

GAs can be described by five major components: **Initialization**, **Fitness evaluation**, **Selection**, **Crossover** and **Mutation**.

In the **initialization** phase, a population of individuals is created. Each individual contains information of size M pertaining to the number of inputs that a given problem may require, while the number of individuals within the population N may be variable, thus the population is of size $M \times N$. Typically, initialization is random, however, there are cases in which initialization may be "seeded" in areas where optimal solutions are likely to be found.

During each generation the **fitness** of each individual to provide the solution to the problem is evaluated by a cost function (often referred to as fitness function in this context) and each individual within the population is ranked accordingly.

Following that, a predefined **selection** criteria will choose which individuals are allowed to survive towards the next generation. Typically, only the fittest based on the previous ranking survive, however, the selection criteria may often be designed to also preserve poorly ranked individuals in an effort to introduce some diversity in the population and prevent premature GA convergence on poor solutions.

The individuals that survive the selection process then are tasked with spawning a new generation of individuals in order to replenish the same amount of individuals that was present before the selection process. This is achieved by the genetic operators: **crossover** (also called recombination) and/or **mutation**.

In **crossover**, a pair of ”parent” solutions are selected to breed a ”child” solution. The chromosome of this new ”child” will be a combination of the information of both parents. While this reproduction method based on the use of two parents is more ”biology inspired”, GAs are not constrained in the number of parents used in crossover to spawn a child, and some research suggests that the use of three or more parents provides better quality chromosome [149].

Mutation works in a similar way as crossover, however, whereas crossover was a direct combination of the chromosomes of the parent solutions, in mutation, the chromosome of the child suffers random alterations. Similar to biology, mutations can be an important part of the evolutionary strategy in order to provide solutions that are caused by stochasticity, when crossover may be limited by its gene pool. GAs can be designed in such a way that only mutation, crossover, or a combination of both are utilized.

The process described above is then repeated for the successive generations, until a satisfactory solution is found. Figure 1.30 shows the basic flow of a GA.

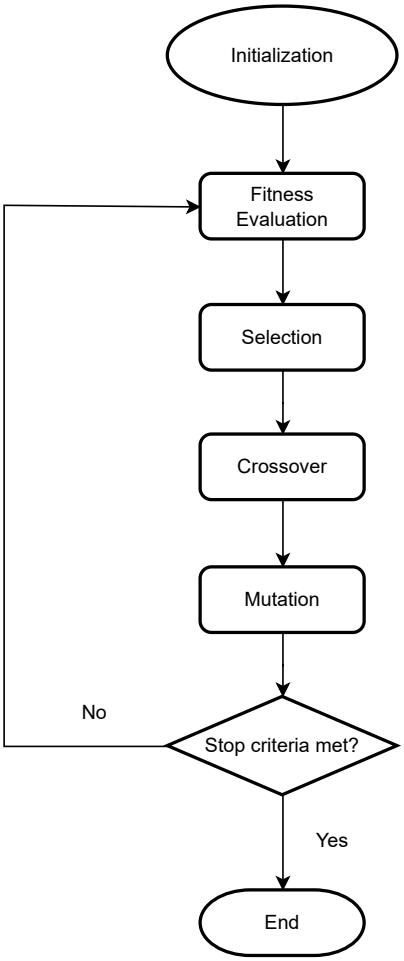


Figure 1.30: Flowchart of a standard Genetic Algorithm.

1.2.2 Limited Precision algorithms

As discussed in the previous sections, the balance between performance and the size of NN architectures is one of critical concern, since a high-performance NN that is not able to train with a reasonable amount of time and resources becomes impractical. One way of improving this balance is the use of sparse connectivity of CNNs.

Broadly speaking, the reduction of the memory footprint can be achieved in two distinct forms:

- **Quantization:** The process of reducing the numerical precision of the NN parameters (weights, activations and gradients) from the typical FP32 to lower bitwidths.
- **Pruning:** The process of shrinking NNs to a reduced size by eliminating redundant parameters or neurons that do not significantly contribute towards its accuracy results.

In this section, the focus will only be in quantization methods.

Typically, NN parameters are 32-bit floating point (FP32) numerical representations of real numbers; depending on the NN task, this precision could be quantized to lower values, aiming at improved efficiency while maintaining accuracy [150]. In this scope, there exists a rich body of literature, however, a disproportionately large majority of these studies only focuses on the inference stage, assuming that the NN is trained beforehand with high precision computations [151]. In the following subsections a brief overview on limited precision algorithms that focus on training will be provided.

1.2.2.1 Expectation Backpropagation

Early quantized networks were trained on the basis of a variation of Bayesian inference called Expectaction Backpropagation (EBP) [152, 153].

EBP derives its methodology from the widely applicable expectation propagation (EP) technique, specifically applied to the approximation of the posterior of the weights using a ”mean-field” factorized distribution in an online setting.

Assuming a NN with large fan-in, the precision of weights and activations can be reduced, in its limits down to 1 bit representations. Beyond this, EBP also has the advantage of being parameter-free, given the prior and the NN architecture.

This method, however, comes with the drawbacks that the bias must be real valued and that the NN architecture must be fully connected, therefore not being applicable to CNNs.

1.2.2.2 Binarized Neural Networks

EBP therefore paved the way towards the appearance of novel limited precision algorithms that rely on the idea of using binary values (± 1) for the weights, coining the term Binarized Neural Network (BNN) [154].

Instead of optimizing the weights posterior distribution such as in EBP, BNNs rely on preserving the floating-point weights to compute the weight update and then passing those weights through a sign function to quantize them into binary values. Deterministic binarization of the weights could lead to a significant loss of information, so instead, BNNs tend to use stochastic binarization:

$$w_b = \begin{cases} +1 \text{ with probability } p = \sigma(w), \\ -1 \text{ with probability } 1-p, \end{cases} \quad [154] \tag{1.46}$$

where, w_b is the binarized weight, w the real valued weight and σ is the "hard sigmoid" function:

$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right) \text{ [154]} \tag{1.47}$$

As such, SGD training in a BNN can be summarized in algorithm 6:

Algorithm 6 SGD training with a BNN [154]. C is the cost function for minibatch and the functions binarize(w) and clip(w) specify how to binarize and clip weights. L is the number of layers.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.
Ensure: updated parameters w_t and b_t.
Forward propagation:
$w_b \leftarrow \text{binarize}(w_{t-1})$
for $k = 1$ to L **do**
compute a_k knowing a_{k-1}, w_b and b_{t-1}
end for
Backward propagation:
Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$
for $k = L$ to 2 **do**
compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b
end for
Parameter update:
Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{\partial b_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}
$w_t \leftarrow \text{clip}\left(w_{t-1} - \eta \frac{\partial C}{\partial w_b}\right)$
$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$

1.2.2.3 Other Limited Precision Approaches

In the previous subsections, the quantization approach relies on diminishing the bitwidth of the weights to the extreme end (1-bit) while still training using a gradient descent

method. This is only possible because the FP32 weight values are preserved to enable the gradient calculations, therefore significantly attenuating the benefits of limited precision computations in terms of memory footprint.

Nonetheless, different approaches can be considered towards limited precision where the precision may be reduced from FP32, while maintaining a large enough bitwidth to allow gradient calculations. Gupta et al. [150] has demonstrated that training using constrained weights, biases and its respective updates to fixed-point representations of 16 bits allows for comparable training to the FP32 counterpart without any significant loss of accuracy. Furthermore, a stochastic rounding scheme was introduced that showed improved training using numerical representations with a fixed fractional length of 8-bits, when compared to its deterministic rounding counterpart. More recent GPU and TPU architectures now include optimization for NN training with half-precision (16-bit) as a standard design for their architectures.

Following on this research, several groups focus on training with reduced gradient precision in order to avoid preserving the FP32 weight values. DoReFa-Net [155] showed comparable training to FP32 on ImageNet using 1-bit weights, 2-bit activations and 6-bit gradients by introducing stochastic quantization to the gradients during BP. The original authors of the BNN expand on this idea by combining deterministic quantization with the use of a ”straight-through-estimator” first proposed by Hinton [156], a shift-based BN and a novel shift-based AdaMAX algorithm [157]. Wang et al. [158] moves away from fixed-point to novel floating point representations with 8-bit precision using stochastic rounding and chunk-based accumulation during training. A

different approach involves the simulation of the effects of quantization during inference and adds correction to the training updates by introducing quantization noise in the gradient updates [159, 160].

The previous examples all relied on some form of gradient-based learning, however, as seen with BNNs, gradient calculation with limited precision becomes challenging considering the discontinuities introduced on the derivatives of the activation functions. As such, Stromatias et Marsland [161] introduced a new limited precision supervised learning algorithm for training Spiking Neural Networks (SNN) based on GA. Population based optimization algorithms such as GA do not require the gradients to be calculated for weight update and therefore present themselves as an interesting alternative in the context of LP training.

Table 1.3 shows a chronological summary of some recent developments in LP training.

1.3 Neuromorphic systems with non-volatile memories

The growing size and demand for NNs call for hardware innovations in parallel with the algorithmic ones in order to make large, high-performance NNs available to users and researchers who are constrained by the cost of computation. Generally speaking, large NNs cannot be implemented efficiently on general-purpose CPUs in either the inference or training phases, since CPUs are specialized in executing only a few potentially very complicated instructions at a time, which goes against the flow of NNs

Table 1.3: Chronology of recent approaches on NN training using limited precision. Adapted from [162].

	Approach	Keywords	Quatization			Benchmark	
			Forward	Backward	Parameter Update	Data	Model
2014	EBP [152]	Expectation Back Propagation	1 bit, FP	-	-	used in [163]	Proprietary MLP
2015	Gupta et al. [150]	Stochastic Rounding	16 bits	16 bits	16 bits	MNIST	Proprietary MLP, LeNet-5
			20 bits	20 bits	20 bits	CIFAR-10	used in [164]
	Binary Connect [154]	Stochastic Binarization	1 bit	1 bit	Float 32	MNIST CIFAR-10 SVHN	Proprietary MLP, CNN
2016	Lin et al [165]	Stochastic Binarization No forward pass multiplication Quantized back propagation	1 bit	1 bit	Float 32	MNIST CIFAR-10 SVHN	Proprietary MLP, CNN
	Bitwise Net [166]	Weight compression Noisy back propagation	1 bit	1 bit	1 bit Float 32	MNIST	Proprietary MLP
	XNOR-Net [167]	Binary convolution Binary dot-product Scaling binary gradient	1 bit	1 bit	1 bit Float 32	ImageNet	AlexNet ResNet-18 GoogLenet
	DoReFa-Net [155]	Stochastic gradient quantization Arbitrary bit-width	1-8 bit	1-8 bit	2-32 bit	SVHN	Proprietary CNN
						ImageNet	AlexNet
2017	QNN [157]	Deterministic binarization Straigth through estimators Shift-based BN Shift-based AdaMAX	1 bit	1 bit	1 bit	MNIST	Proprietary MLP
						CIFAR-10 SVHN	CNN from [154]
						ImageNet	AlexNet GoogLenet
			4 bit	4 bit	4 bit	Penn Treebank	Proprietary RNN LSTM
2018	Wang et al. [158]	Novel floating point chunk-based accumulation Stochastic rounding	8 bit	8 bit	8 bit	CIFAR-10	Proprietary CNN ResNET
						BN50 [168]	Proprietary MLP
						ImageNet	AlexNet ResNET18 ResNET50
	Jacob et al. [159]	Training with simulated quantization	8 bit	8 bit	8 bit	ImageNet	ResNet Inception v3 MobileNet
						COCO	MobileNet SSD
						Flickr [169]	MobileNet SSD
2019	WAGEUBN [170]	BN layer quantization 8-bit integer representation Combination of direct, constant and shift quantization	8 bit	8 bit	8 bit	ImageNet	ResNet18/34/50
2020	S2FP8 [171]	Shifted and squeezed FP8 representation of tensors Tensor distribution learning	8 bit	8 bit	32 bit	CIFAR-10	ResNet20/34/50
						ImageNet	ResNet18/50
						English-Vietnamese	Transformer-Tiny
						MovieLens	Neural Collaborative Filtering (NCF)
	Wiedemann et al. [172]	Stochastic gradient quantization Induce sparsity Non-subtractive dither	8 bit	8 bit	32 bit	MNIST	LeNet
						CIFAR-10/100	AlexNet ResNet18 VGG11
						ImageNet	ResNet18
	Quant-Noise [160]	Training using quantization noise	8 bit	8 bit	8 bit	Wikitext-103 MNLI ImageNet	RoBERT RoBERT EfficientNet-B3

that require large data volumes and highly regular workloads built from a small set of computational primitives.

GPUs, which contain thousands of co-processors that compute in parallel significantly improves performance on NN processing. The GPU's co-processors share access to a very fast local memory or to a global memory in a highly parallelized fashion. Combining these advantages in terms of parallelism with already mature software in

terms of drivers and libraries facilitates the usage of GPUs for inference and training, making them the preferred choice of hardware for NNs.

Nonetheless, in spite of the advantages over CPUs, memory transfer still remains a major bottleneck for GPUs when processing large NNs [173, 174], making the prospect of in-memory processing evermore appealing.

1.3.1 Architectures for inference

In the context of neuromorphic acceleration, the use of inference-only accelerators are a nearer-term application compared to the full training experience due to the stringent device and circuit requirements. In this sense, inference accelerators can be seen as a ”read-only” circuit where the weights of a NN are trained externally (e.g. GPU, TPU or cloud processing) and subsequently loaded onto the inference accelerator. For accelerators based on crossbars of eNVM devices, this means programming all of the devices once, possibly followed by occasional re-programming to mitigate component failures caused by retention drifts over time.

1.3.1.1 VMM in crossbars

By embedding NN computations directly inside the memory elements that store the weights, analog neuromorphic accelerators based on eNVM arrays can greatly reduce the energy and latency costs associated with data movement. In particular, 2-terminal devices such as RRAM, allow building crossbar arrays that gave rise to the concept of massively parallel VMM operations to accelerate the inference of NNs (Figure 1.31).

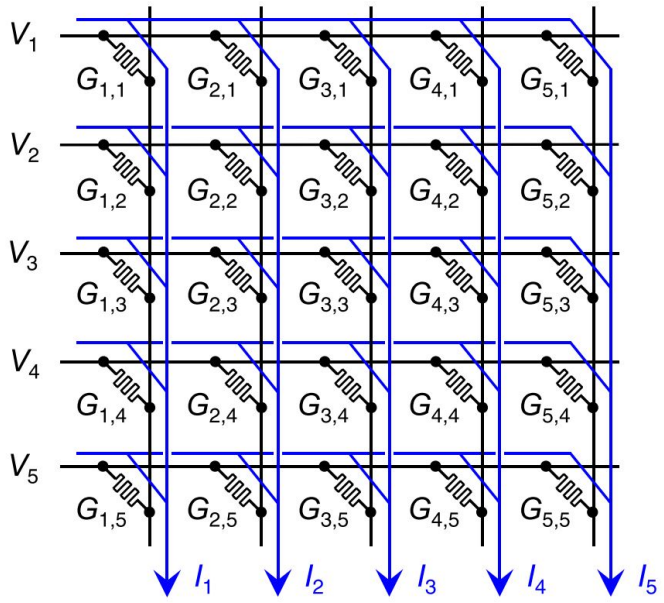


Figure 1.31: The basic concept of a massively parallel analog VMM within an RRAM crossbar [130].

As introduced in section 1.2.1.5, the inference stage of a feedforward NN relies on MAC operations on the weights seen at the input of each neuron. The structure of these crossbar arrays naturally performs a vector dot product when all the rows are activated simultaneously, driven by a voltage (input) V_i at each row i, such that the current (output) collected at each jth column is:

$$I_j = \sum_{i=0}^{N_r-1} G_{ij}V_i,\ 0 < j < N_c - 1\ [130] \tag{1.48}$$

where, G_{ij} is the conductance (weight) of the eNVM element at array position (i, j), N_r is the number of rows, and N_c the number of columns. This natural operation is possible because the multiplications between G_{ij} and V_i are realized by Ohm's law and the summation of currents by Kirchhoff's law. Since the currents flow through all the columns in parallel, the crossbar executes the full VMM in a single operation. Additionally, the bias **b** can be seen as an extra row in the crossbar array.

1.3.1.2 Input signal encoding

As mentioned in the previous subsection, the input data for neuromorphic crossbars based on eNVM devices is the voltage applied to each row of the crossbar. Nevertheless, the way in which the voltage signal is encoded as an input can assume many forms, and this aspect is one of the main differentiators between different inference accelerator architectures. Figure 1.32 shows different categories of input representation.

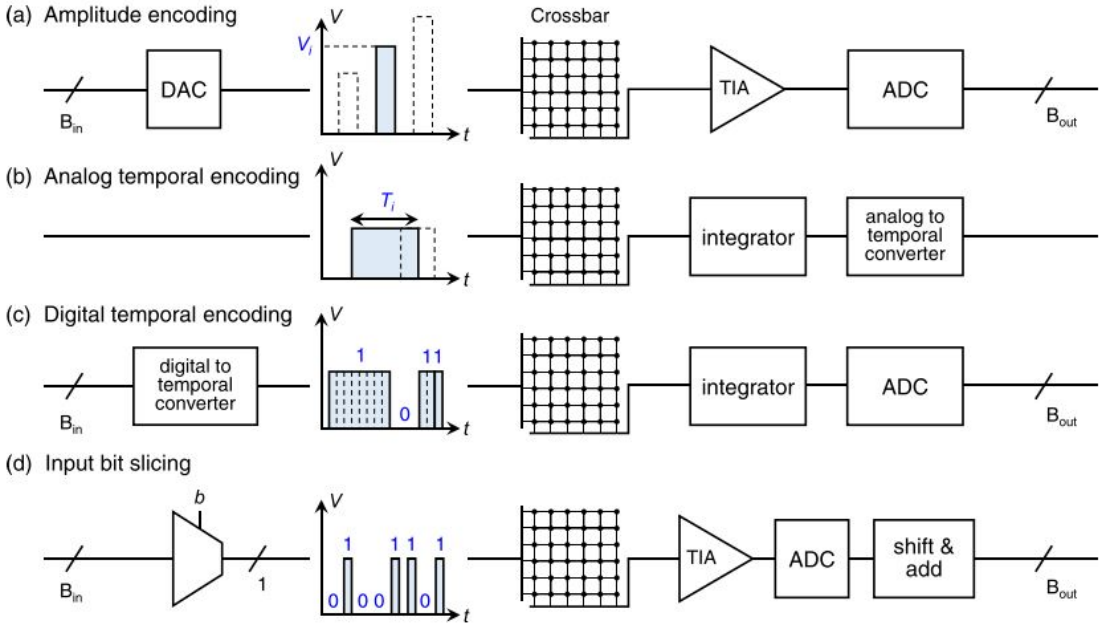


Figure 1.32: Four different schemes for representing the crossbar input signal and the associated peripheral circuitry: (a) Voltage amplitude encoding [175, 176]. (b) Analog temporal encoding [177, 178]. (c) Digital temporal encoding [179] and (d) input bit slicing [180, 181].

The most direct method of implementing the VMM operation of equation 1.48 is through **amplitude encoding**. In other words, an input x_i could simply be encoded as the amplitude of a voltage signal V_i as shown in Figure 1.32a [175, 176]. For an input with B_{in} bits of precision, this method requires a B_{in}-bit DAC to supply the $2^{B_{\text{in}}}$ possible analog voltage levels to a crossbar row.

The advantage of this method is that all of the rows can be driven simultaneously, and the full VMM can be realized in a single crossbar read operation without any latency dependency on the timing of the inputs. Nevertheless, some significant drawbacks exist in this methodology. The area and energy consumption of the DAC could scale exponentially with B_{in}, and while a singular DAC scales similarly to an ADC, the DACs cannot be shared across the inputs in the same way that an ADC can be shared or multiplexed over the outputs. Another major drawback is that the memory device utilized in the crossbar must have a highly linear I-V characteristics across the whole range of voltage amplitudes to be read, and as already discussed in section 1.1.4, RRAM nonlinearity is one of its major performance concerns.

A different methodology can be used by driving the rows of the crossbars with voltage pulses of fixed amplitude ($\pm V_0$) but variable duration [177], effectively eliminating the requirement of device I-V linearity. When used with analog voltage pulses, this method is known as **analog temporal encoding** (Figure 1.32b), in this way, the input x_i is encoded in the pulse duration T_i, converting the VMM from equation 1.48 into:

$$I_j = V_0 \sum_{i=0}^{N_r-1} S_i G_{ij} T_i \,[130], \tag{1.49}$$

where $S_i = \pm 1$ is the sign of the pulse.

Besides the relaxed requirements regarding device I-V linearity, this temporal approach has the added benefit that the activation data is encoded in an analog way, therefore bypassing the conventional ADC step and potentially requiring no DACs at the input. Nevertheless, this approach still requires ADC-like circuitry to convert the

outputs of each crossbar (voltage or current) into temporally coded signals and since these circuits have finite temporal resolution, the pulse duration (and consequently the VMM latency) scales exponentially with the effective number of input bits represented. Another issue regarding the purely analog approach is that the noise associated with the analog signals will effectively be accumulated and propagated through the different layers of a feedforward NN, becoming a limiting factor in very deep NNs [175].

A variation of the previous approach is **digital temporal encoding** (Figure 1.32c). This method shares its principals of operation with its analog counterpart, but with the difference that the inputs are taken in the digital form and converted by a digital logic circuit into a voltage pulse train with one pulse per input bit [179]. The column currents produced by this pulse train are then accumulated in an integrator and passed through an ADC.

Input bit slicing (Figure 1.32d) avoids the use of analog voltages while maintaining a read latency that scales linearly with its input resolution. In this approach, a digital input x_i can be passed one bit at a time using binary voltage pulses of fixed length. The output of the VMM with input bit slicing is:

$$Y_j = \sum_{b=0}^{B_{\text{in}}-1} 2^b \left(\sum_{i=0}^{N_r-1} G_{ij} V_i^{(b)} \right) \text{[130]}, \tag{1.50}$$

where, $V_i^{(b)} \in \{0, V_0\}$ is the binary voltage pulse amplitude corresponding to the bth bit of the input x_i.

This VMM operation requires B_{in} sequential crossbar read operations (inner sum), each of which requires an ADC step at the output. By organizing the bits from lowest to

the highest significance the VMM can be implemented by shifting the digitized crossbar output one position to the right prior to adding the output for the next bit. Therefore, the outer sum can be implemented using a digital shift-and-add circuit at the outputs [180, 181].

An advantage of this approach is that the 1-bit inputs do not require sophisticated DACs and the inputs pulse trains could potentially simply be modulated by access transistors that enables or disables the input rows. Additionally, the binary resolution of the inputs consequently reduces the required ADC resolution at the outputs.

1.3.1.3 Synaptic bit slicing

One of the limiting factors hindering the use of neuromorphic eNVM crossbars for inference and training is the number of distinguishable conductance levels that each synaptic weight is able to represent when accounting for the device non-ideal behaviour. Realistically, 6 to 8-bit precision remains at the upper limit of what singular eNVM elements are able to achieve [182–184], so to enable higher precision crossbar processing a method called **synaptic bit slicing** can be employed.

Synaptic bit slicing relies on a similar concept to the previously discussed input bit slicing but applied to the synaptic elements instead of the inputs. The basic principle relies on the use of multiple eNVM elements to represent a single weight, such that the B_w bits of a synaptic weights can be segmented into N_w slices with $\tilde{B}_W = B_w/N_w$ bits each, making so that each eNVM element is only required to store the bitwidth of the sliced weight $\tilde{B}_W$.

The devices representing the sliced weights are organized such that each slice of the same weight is partitioned along the columns of the same row, therefore creating blocks of N_w columns to represent each weight commonly referred to as **stripes** that are subsequently aggregated by shift-and-add reduction trees. (Figure 1.33).

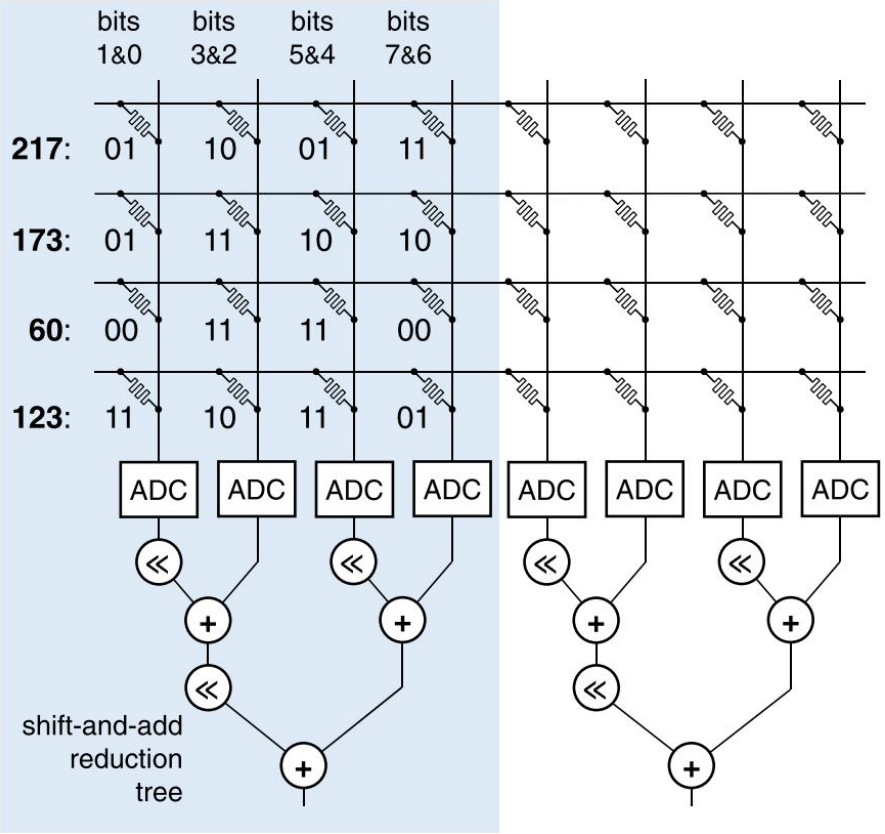


Figure 1.33: Column-wise synaptic bit slicing. Each weight is represented by an 8-bit integer and implemented using four 2-bit memristors spread across four columns. Each column block is aggregated using a shift-and-add reduction tree [130].

One approach for inference accelerators is to combine synaptic bit slicing with input bit slicing in what has been dubbed as the ”internally analog, externally digital” approach to VMM [185]. In this instance, the VMM equation can be expressed as:

$$Y_j = \sum_{b=0}^{B_{\text{in}}-1} \sum_{c=0}^{N_w-1} 2^{b+c} \left(\sum_{i=0}^{N_r-1} G_{ij}^{(c)} V_i^{(b)} \right) \text{[130]}, \tag{1.51}$$

where b indexes the input bit and c indexes the weight bit slice. Typically, the weight bit slices are aggregated first, followed by the input bit slices.

1.3.1.4 Signed computation

One discrepancy that arises between the NN concepts and its implementation in analog circuitry with eNVMs is the implementation of negative synaptic weights. The use of both positive and negative weights is a fundamental aspect in the functioning of NNs, however, implementations with eNVM crossbars rely on the eNVM conductance to represent each weight which makes the representation with negative conductance impossible.

The most common solution to tackle this issue is to use a differential pair of conductances. In other words, each weight must be represented by a pair of devices, instead of a singular device, such that the conductance of the differential pair is represented by:

$$G_{ij} = G_{ij}^{+} - G_{ij}^{-}. \tag{1.52}$$

As such, the VMM operation in the analog domain is described by:

$$I_j = \sum_{i=0}^{N_r-1} \left(G_{ij}^{+} - Gij^{-} \right) V_i \, [130] \tag{1.53}$$

The current subtraction in the above equation can easily be implemented by applying the same voltage input with different polarity to the two elements of the differential pair and summing the currents of each column [179] as shown in Figure 1.34.

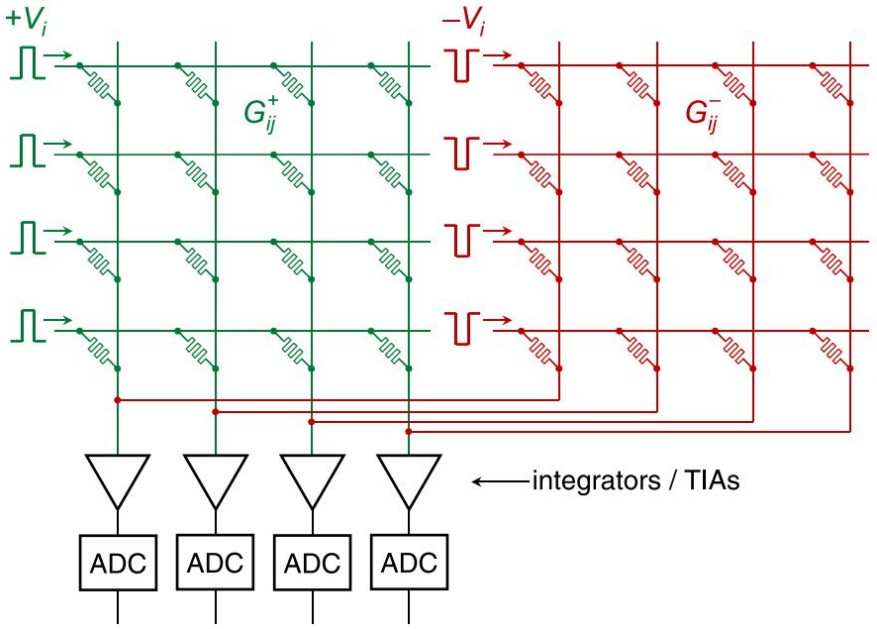


Figure 1.34: A general scheme to represent positive and negative weights. When a positive input pulse is sent to the left crossbar, a negative pulse of the same magnitude is sent to the right crossbar and vice versa, and a subtraction is performed by Kirchhoff's law [130].

Alternative approaches to negative weight implementation using a single crossbar have also been proposed, where a column of reference bias resistors (or memristors) is added to each crossbar, whose fixed conductance G_b is effectively subtracted from the conductance of each synaptic element using an analog inverter [186, 187]. While being more area-efficient than the two-crossbar approach, the susceptibility to errors coming from variability, drift or offset in the shared reference resistors rises.

1.3.2 Architectures for training

Training of large NNs using analog neuromorphic accelerators is considered a more long-term goal due to the challenges presented by additional device non-ideal behaviours, such as, update non-linearity, symmetry, precision, latency, energy consumption and endurance. Most of the already discussed considerations for inference also apply for the

feedforward step in training and in this section, additional considerations regarding BP and weight update will be presented.

1.3.2.1 Backpropagation in neuromorphic architectures

As with the feedforward step, the major advantage of using eNVM crossbars lies in its intrinsic parallelism. In order to support BP, both the computation of the layer-by-layer errors δ and the weight updates ΔW should be designed with parallelism in mind.

The BP of errors is by design the MVM operation that corresponds to the transpose of the VMM operation used in the inference step (equation 1.48). As such, the implementation of MVM can be parallelized in the same fashion as the VMM, as long as the circuit supports forward and backward flow of data. The peripheral circuitry that is used for VMM can also be reused for the MVM operation with additional re-routing [179, 188].

An additional requirement of the BP step is the usage of the derivative of the neuron function f' (equation 1.27). Activation functions such as the sigmoid or hyperbolic tangent can have its derivative implemented by separate digital lookup tables, however, other activation functions such as ReLU or a piece-wise linear version of the hyperbolic tangent become much easier to implement in circuit since their derivative are simple step functions [189].

Figure 1.35 shows the schematic of the reconfigurable neural core implemented by Marinella et al [179] and the peripheral circuit blocks involved in the three stages required for training: VMM, MVM, and outer product weight update.

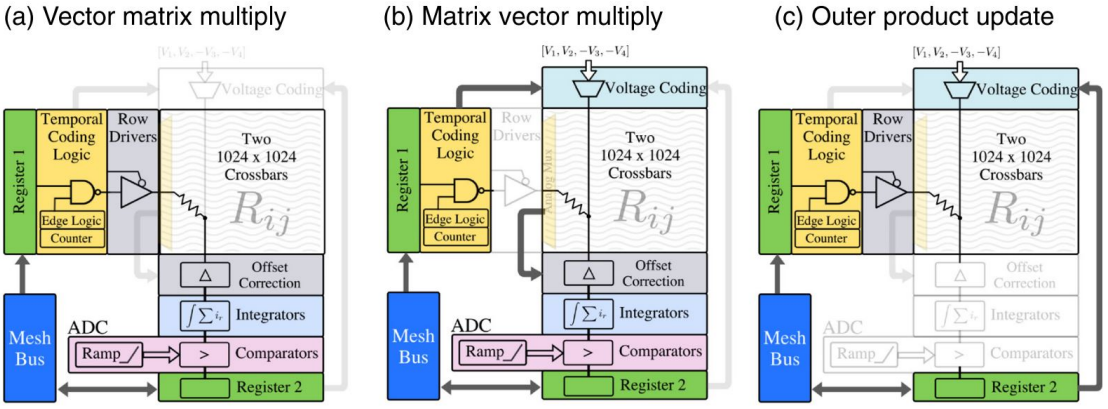


Figure 1.35: Reconfigurable neural core from [179] for implementing (a) VMM, (b) MVM and (c) outer product update.

1.3.2.2 Parallel weight update

As with VMM and MVM, the weight update step in the training of a neuromorphic array should be parallelized to avoid impractical latency issues. Derived from equation 1.28, Marinella et al. [179] suggest that the activations $\mathbf{x}$ can be applied as a temporal coded input to one edge of the crossbar, while the errors δ are applied as an amplitude encoded signal at the opposite end, resulting in a multiplication effect seen at each crosspoint as shown in Figure 1.36. The learning rate η can be controlled by scaling the pulse lengths or the number of pulses fired [190].

Rosenthal et al. [191] implements a similar variation to this but takes an analog input for δ instead of a digital signal and applies those signals to access transistors along a column instead of directly across the devices, allowing for fully analog implementations of the parallel outer product update.

Alternatively, the multiplication effect can also be achieved at constant voltage amplitudes by encoding one variable in the length or duty cycle and the other variable in the repetition rate of two overlapping pulse trains [61, 192]. The benefits of using constant

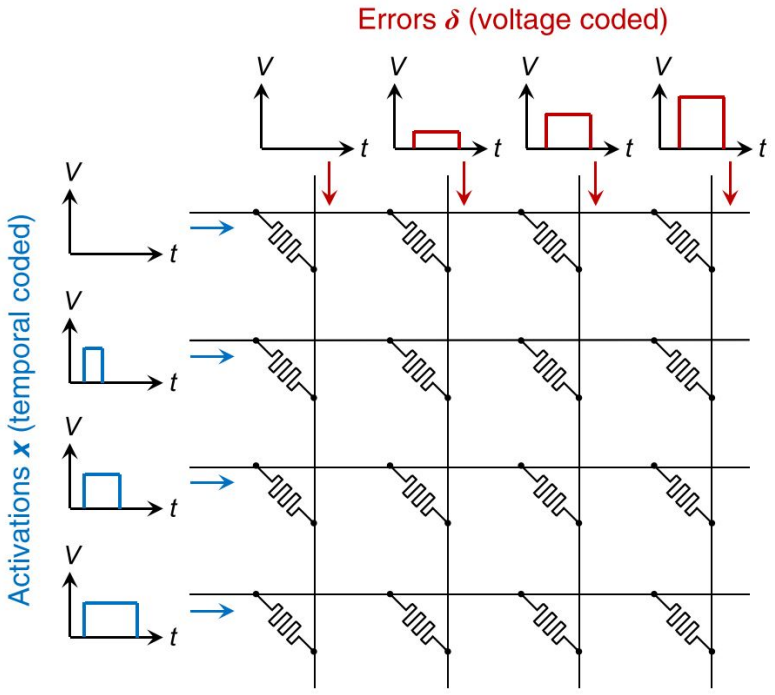


Figure 1.36: Demonstration of parallel outer product update of a crossbar array using temporal encoding for the activations and amplitude coding for the errors.

voltages come in the form of reduced area/power consumption of the ADCs/DACs, but at the cost of increased latency.

Using a parallel outer product in detriment of a serial row-by-row programming scheme has not only the more straightforward advantage of improving the weight update latency by $O(N)$, but also comes with the added benefit of reducing the temporary storage requirements on the peripheral memory buffers, since only the $\mathbf{x}$ and δ vectors need to be stored rather than the full weight update matrix $\Delta\mathbf{W}$.

One potential drawback in this approach is the large instantaneous power draw that comes with programming all of the weights at once. To reduce this large power consumption, the weight updates may be sectioned off into blocks that update in a parallel fashion inside the block, but each block updates sequentially [177]. The design of these

blocks is application dependent and the balance between power consumption and latency must be considered.

1.3.2.3 Batch training

The previous section discussed the benefits of a parallel weight update in terms of latency, energy consumption and storage overhead, however, the parallel outer product comes with the inherent drawback of being incompatible with training with batch sizes greater than one, since the weight update matrix $\Delta\mathbf{W}$ is no longer stored. This limitation can not only be problematic in calculating accurate estimates of the true gradients [125], but also imposes greater endurance requirements on the devices, since they need to be programmed after every training sample.

Nevertheless, batch training while maintaining parallel weight updates have been realized by using two crossbars per weight matrix [190]. In this way, while the first crossbar handles the VMMs from forward propagation, parallel weight updates are being applied to the second crossbar. At the end of a batch, the total accumulated weight update is read from the second crossbar and serially transfered to the first one.

The PipeLayer architecture [193] built upon and expanded this approach by duplicating each weight matrix so that while one matrix is being used for forward propagation, the other is used in BP, therefore pipelining the two processes within a batch without weight conflicts. At the end of each batch, the accumulated weight updates are copied from the buffer array to all crossbars containing the copies of the weight matrix. The schematic for dataflow in PipeLayer can be seen in Figure 1.37.

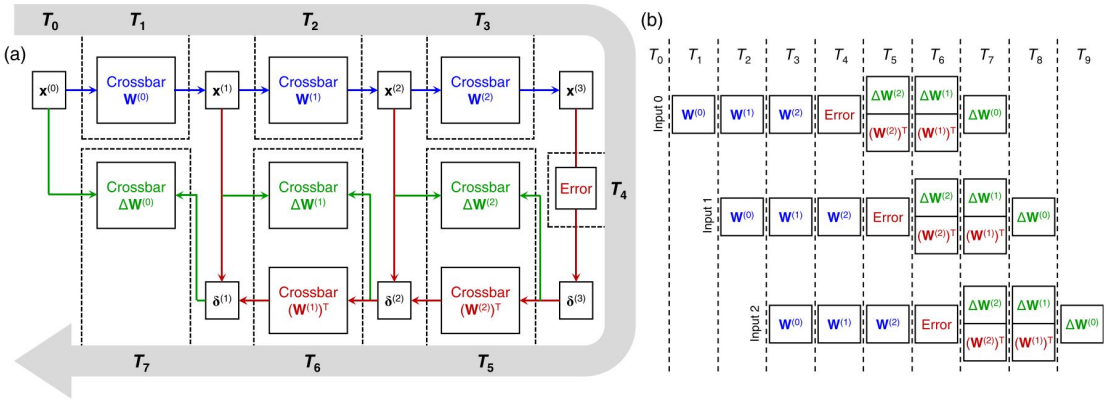


Figure 1.37: The PipeLayer architecture. (a) Dataflow of a single example through a 3-layer NN for training. Each time step may correspond to several computational cycles. The computational blocks active in each time step are labeled: forward propagation (blue) backpropagation (red) and weight update (green). (b) Pipeline for multiple training examples. The weight updates $\Delta\mathbf{W}$ are serially written to the $\mathbf{W}$ and $\mathbf{W}^{\mathsf{T}}$ crossbars at the end of a batch. Adapted from [193].

The intermediate activations $\mathbf{x}$ are also stored in dedicated local RRAM buffers. This pipelining of the forward and backward propagation means that the storage requirement for the activations is determined by each layer's depth within the NN instead of the batch size.

1.3.3 Neuromorphic simulation frameworks

As discussed in the previous sections, the focus on synaptic crossbars using eNVM devices is rising. Despite this surge of interest, eNVM designs for NNs are still in a prototypical phase, consequently, due to the difficulty and cost constraints on eNVM fabrication, simulating device and circuit behaviour prior to circuit-level realization becomes a necessity.

Traditional circuit-level relied on general purpose Simulation Program with Integrated Circuit Emphasis (SPICE), but as the complexity of the underlying systems and

neuromorphic architectures being simulated grew, SPICE-based simulation, which is difficult to parallelize, became prohibitively slow and impractical.

There is therefore a growing need for customized simulation frameworks that are able to process large designs in a parallel and efficient manner. Table 1.4 and Figure 1.38 compare some of the more modern simulation frameworks for eNVM neuromorphic circuitry design.

Table 1.4: A comparison of modern simulation frameworks [194].

Simulation framework	Prog. language(s)	GPU	Pre-trained DNN conversion	TF/ PyTorch integration	Inference	Training	Peripheral circuitry	Supported devices	Open-source
RAPIDNN [195]	C++, SPICE		✓	✓	✓		✓	Single-level memristive devices	
PUMA [196]	C++		✓	✓	✓		✓	eNVM and legacy NAND flash	
DL-RSIM [197]	Python	✓	✓	✓	✓		✓	eNVM	
Tiny but Accurate [198]	MATLAB		✓	✓	✓		✓	eNVM	✓
Ultra-Efficient Memristor-Based DNN [199]	C++, MATLAB		✓	✓	✓		✓	eNVM	✓
MemTorch [200, 201]	Python, C++, CUDA	✓	✓	✓	✓		✓	eNVM and legacy NAND flash	✓
NeuroSim [94, 105, 202, 203]	C++, Python	✓	✓	✓	✓	✓	✓	eNVM and legacy NAND flash	✓
IBM Analog Hardware Acceleration Kit [204]	C++, Python, CUDA	✓	✓	✓	✓	✓	✓	eNVM	✓

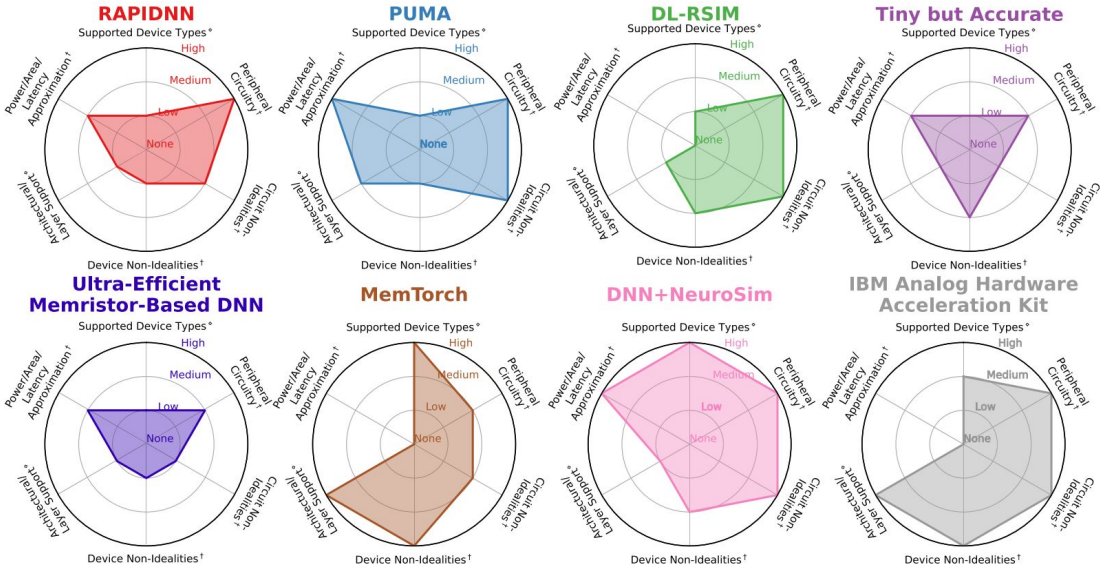


Figure 1.38: Radar chart comparing different simulation frameworks [194].

At first glance it would seem that there is an abundance of similar frameworks to choose from, however, Figure 1.38 shows that most of these frameworks have distinct

strengths and weaknesses due to the adoption of different software designs and usability approaches, making them complimentary to each other more so than direct competitors.

For instance, both Tiny but Accurate and the Ultra-Efficient Memristor-Based DNN are built upon NVSim [205], whereas all of the other presented frameworks are either written from scratch in a lower level language (C++ or Python), or are extensions to popular existing high-level GPU-accelerated computing libraries such as TensorFlow or PyTorch which allow faster processing at the cost of more sophisticated hardware requirements.

Furthermore, while RAPIDNN, PUMA, Tiny but Accurate, Ultra-Efficient Memristor-Based DNN and DNN + NeuroSim can be used to generate estimate reports on power consumption, area and latency, these frameworks have limited flexibility in terms of different NN layer types and device non-ideality simulation, when compared to MemTorch and the IBM Analog Hardware Acceleration Kit.

Moreover, out of the listed frameworks in Table 1.4, only DNN + NeuroSim and the IBM analog hardware Acceleration Kit (denoted as aihwkit in short-form) are capable of simulating training using eNVM based deep CNN architectures. Lammie et al. [194] benchmarked these two simulation frameworks on training a VGG-8 network architecture on the CIFAR-10 dataset, using a High Performance Computing (HPC) cluster with the following run-time hardware configuration:

- 1 node and 8 CPU cores (Intel Xeon 6132 series CPU sockets)
- 100 GB DDR4 3200 MHz RAM
- 1 PCI-E 32GB Volta V100 GPU

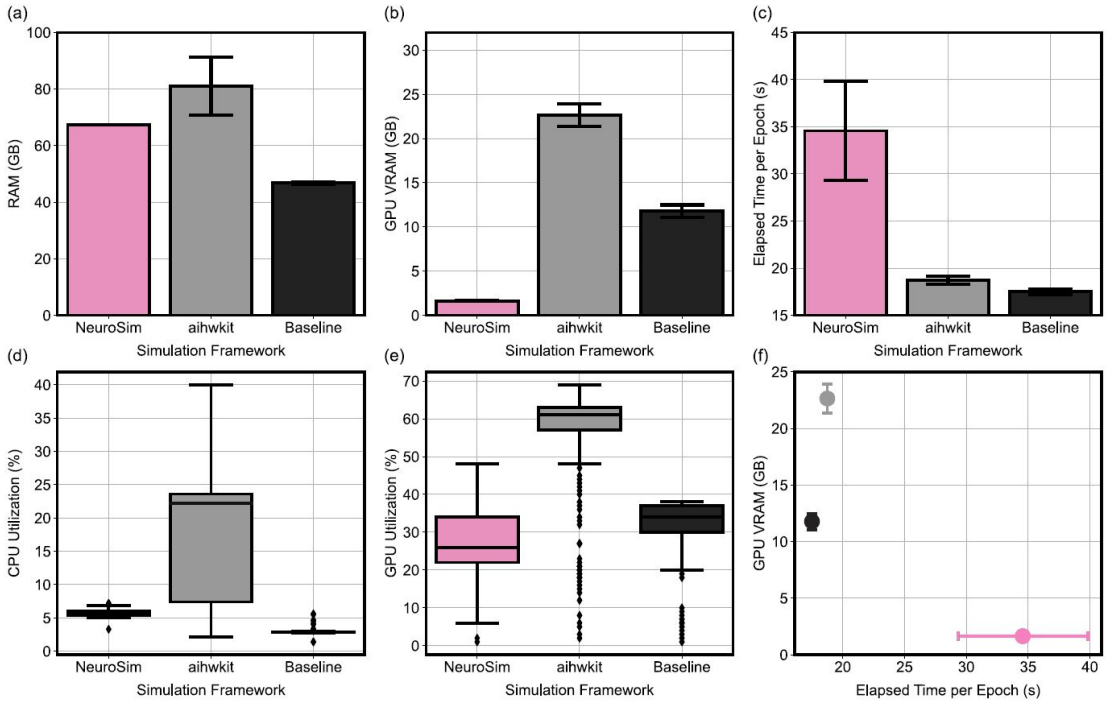


Figure 1.39: Comparison of training routines between DNN + NeuroSim V2.1, the IBM Analog Hardware Acceleration Kit (aihwkit) and a Baseline PyTorch ML library for the VGG-8 network architecture, using the CIFAR-10 dataset. Adapted from [194].

Figure 1.39 shows the results of those benchmarks. While the aihwkit consumes more resources overall, it also has a comparable time per epoch to the PyTorch baseline, which makes sense given that the additional computational resources are directed towards the simulation of the crossbar arrays and non-idealities, without a large penalty on simulation time. More interestingly, DNN + NeuroSim consumes more RAM than the baseline implementation, but significantly less VRAM, while taking almost 3x longer per epoch, suggesting an overall under utilization of the parallel GPU resources. One large factor in play could be that DNN + NeuroSim simulates modular crossbar tiles, while the aihwkit does not, and while the aihwkit may be limited in terms of hardware resources in case of NN scaling, DNN + NeuroSim could be limited in terms of impractical training times due to the under-utilization of the parallel capabilities of the hardware.

Chapter 2

Devices and characterization methodology

In this chapter, the characterization methodology as well as a brief overview on the fabrication of the devices used throughout the thesis work will be overviewed. First, the details pertaining the used RRAM devices will be given, followed by details on the used instrumentation for characterization.

Throughout the thesis work, a combination of DC and AC electrical characterization techniques were performed which will be discussed in further detail. Furthermore, one important aspect discussed throughout this work is that of RTN, so the methodology of RTN testing and time constant extraction will be discussed in detail.

Finally, custom software was built with the purpose of extending the stock instrumentation capabilities in regards to neuromorphic specific testing in RRAM. This methodology will be further explored in the final section of this chapter.

2.1 Devices

One overarching theme throughout this work is based on the differences in terms of noise and variability between filamentary and non-filamentary switching RRAM devices. As such, all measurements are taken from either a filamentary Ta_2O_5 based device or the non-filamentary a-VMCO counterpart.

Both filamentary and non-filamentary RRAM devices are fabricated in singular device isolated crosspoint structures. All used samples were fabricated at IMEC [37, 206] and subsequently sourced to LJMU for further characterization.

The filamentary Ta_2O_5 RRAM consists of a $TiN/Ta_2O_5/TaO_x/TaN/TiN$ stack processed in an integrated process [56]. The TiN BE was sputtered at room temperature and patterned. A 4nm thick stoichiometric Ta_2O_5 layer was deposited by Atomic Layer Deposition (ALD). A nonstoichiometric 20nm thick TaO_x film was deposited by reactive DC magnetron sputtering using a Ta target under oxygen ambient. Without breaking the vacuum, a 10nm thick TaN capping layer was sputtered. Finally, a 30nm thick TiN film was sputtered.

The non-filamentary a-VMCO devices consist of a $TiN/a-Si/TiO_2/TiN$ stack processed in a CMOS-compatible process [41, 56, 207]. The active stack consists of an 8nm amorphous silicon (a-Si) layer deposited by Physical Vapour Deposition (PVD) and an 8nm ALD TiO_2 layer crystallized in the anatase phase. This stack is sandwiched between TiN BE and TE with 30nm and 40nm thicknesses respectively [37].

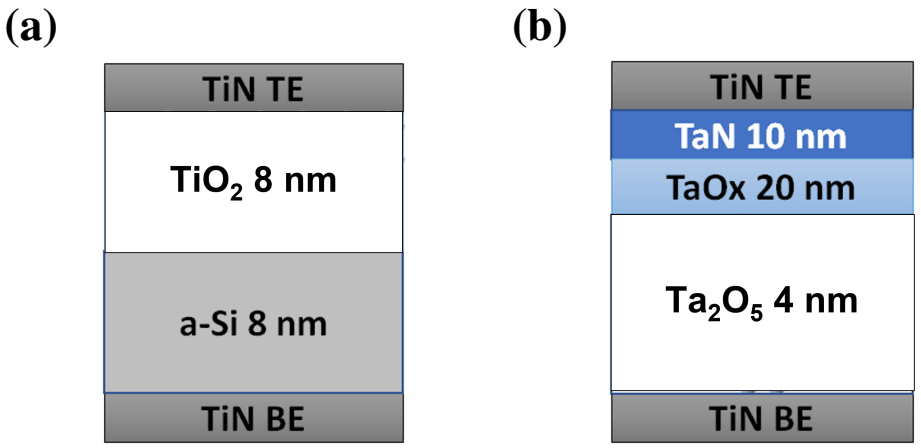


Figure 2.1: Illustration of the device stack of the (a) aVMCO and (b) Ta_2O_5 RRAM devices.

The dies provided by IMEC contain a range of different sized devices. Nevertheless, for the purpose of this work, unless stated otherwise, all measured Ta_2O_5 devices have a size of $75 \times 75 nm^2$ and the measured a-VMCO devices have a size of $135 \times 135 nm^2$.

2.2 Instrumentation

In this section a brief overview of the instrumentation used for characterization throughout the thesis work will be given.

Figure 2.2 shows the measurement system used throughout this work, based on the Keysight B1500A Semiconductor Device Parameter Analyser connected to a Signatone S-1600S probe station.

The Keysight system is equipped with 4 Source Measurement Units (SMUs), as well as 4 B1530A Waveform Generator/Fast Measurement Units (WGFMUs) and 2 B1525A Semiconductor Pulse Generator Units (SPGUs). This system features current-voltage (IV) measurement capabilities of spot, sweep, sampling and pulse measurement in the

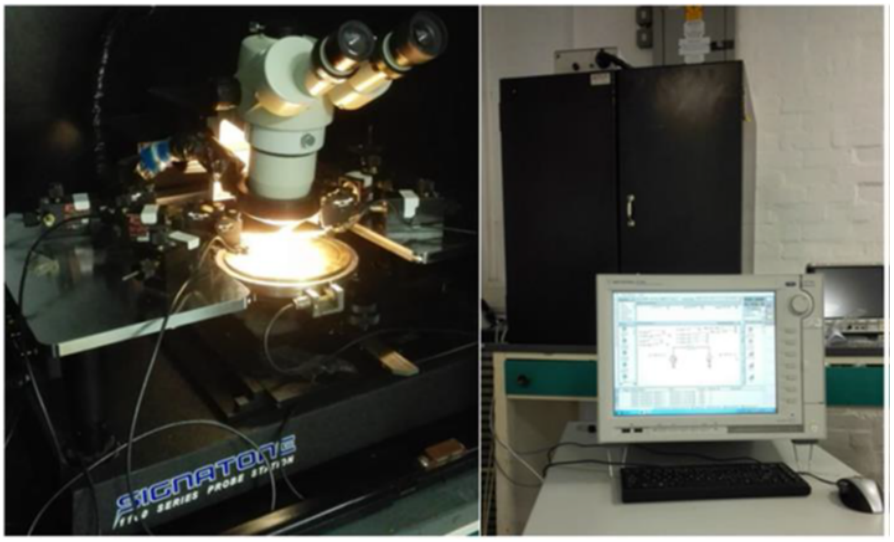


Figure 2.2: Signatone S-1160S probe station (left) and Keysight B1500 semiconductor parameter analyser (right).

range of $0.1\text{fA} - 1\text{A} \,/\, 0.5\mu\text{V} - 200\text{V}$, AC capacitance measurement in multi frequency from 1kHz to 5MHz and Quasi-Static Capacitance-Voltage (QS-CV) measurement capabilities, pulsed IV and ultra-fast IV measurement capabilities from a minimum sampling interval of 10ns (100MSa/s), and up to 40V high voltage pulse.

The Keysight system can be controlled either through its embedded EasyExpert software, or by its General Purpose Interface Bus (GPIB) that allows a connection to an external computer capable of triggering the Keysight's Sourcing/Measurement subroutines from inside a script, extending the flexibility offered by the default EasyExpert software.

2.3 DC measurements

As mentioned in section 1.1.3.3 RRAM devices can either belong to the bipolar or unipolar category in what comes to its operation scheme. However, in the scope of this

thesis, both the non-filamentary a-VMCO devices and the filamentary Ta_2O_5 operate in the bipolar mode (Figure 1.8b), with the added caveat that the a-VMCO devices are self-compliant while the Ta_2O_5 are not, and require an external CC on the SET process to prevent device breakdown.

2.3.1 Stepped IV measurements

Arguably the most basic and essential form of RRAM characterization lies in its basic IV characteristics. In this work, all DC IV measurements are performed using only two-terminals as the measured devices are not integrated with a transistor.

The DC IV characteristics can be obtained by either sweeping the current and measuring the voltage drop (current-controlled), or by sweeping the voltage across the device and measuring the current (voltage-controlled) in a predefined range $\left(\left[V_{Initial} : V_{Step} : V_{Final}\right]\right)$. In the scope of this work, all DC measurements are voltage-controlled. Additionally, the sweeps can be defined as either single or double as illustrated in Figure 2.3. Due to the characteristic hysteretic behaviour of RRAM, the usage of the double sweep becomes a necessity to observe the full IV behaviour, and as such, all DC sweeps performed in this work are double sweeps unless stated otherwise.

Using the conditions specified in table 2.1, the DC IV characteristics were measured for both the Ta_2O_5 and aVMCO devices, and are presented in Figure 2.4. These measurements serve as the reference point throughout the thesis work.

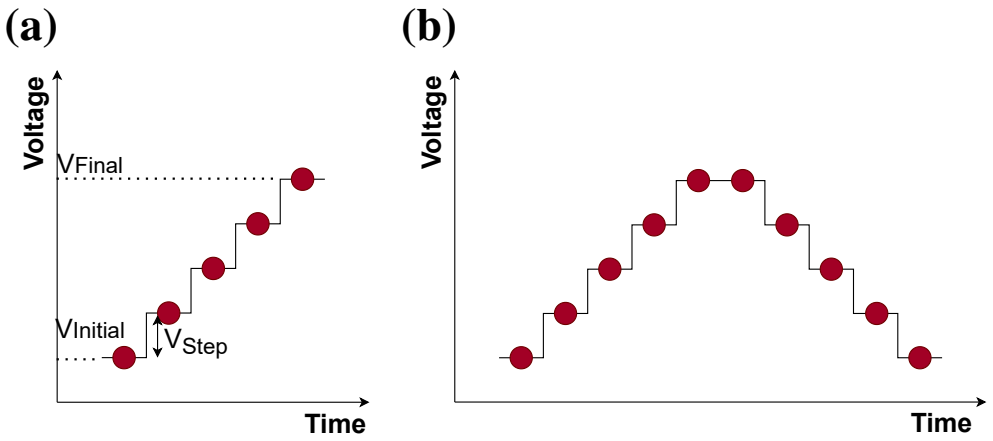


Figure 2.3: Schematic of a DC IV (a) single sweep and (b) double sweep measurement. The red circles represent the point in time at which the current is recorded.

Table 2.1: Conditions used for recording the DC IV characteristics in the Ta_2O_5 and aVMCO devices.

		$V_{Initial}$	V_{Step}	V_{Final}	CC
Ta_2O_5	SET	$0V$	$0.1V$	$3V$	$60\mu A$
	RESET	$0V$	$0.1V$	$-1.7V$	-
aVMCO	SET	$0V$	$0.1V$	$-3V$	-
	RESET	$0V$	$0.1V$	$5.5V$	-

2.3.2 DC RTN measurements

One important factor considered in this work is the impact of read noises, specifically RTN, on the pattern recognition accuracy of neuromorphic networks containing RRAM devices.

As such, the RTN measurement methodology is based on taking multiple long DC measurements at the specified read-out voltage of each device after programming to a desired resistance level. Each measurement is taken with a sampling time of 2 ms/sample and contains $10,000$ samples. The read-out voltages are $0.1V$ and $3V$ for the Ta_2O_5 and aVMCO devices respectively.

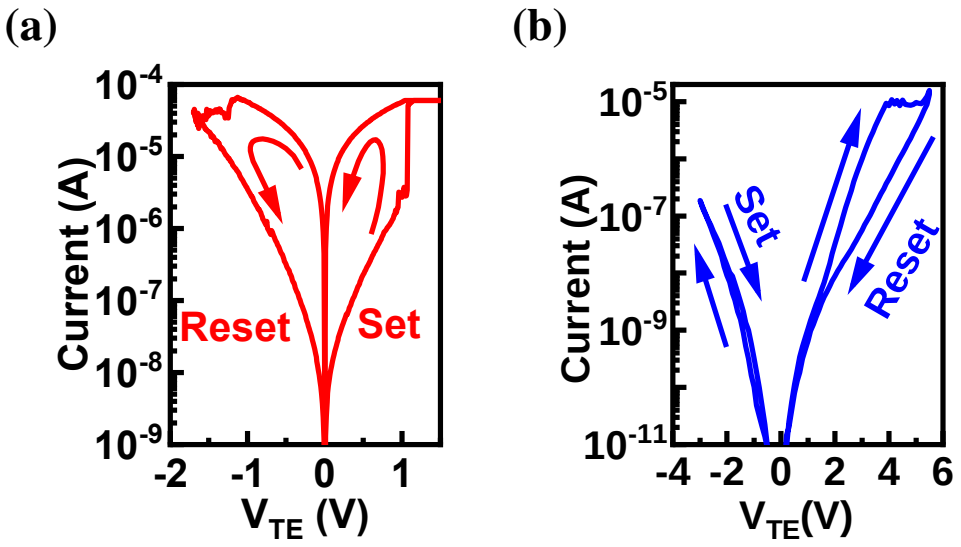


Figure 2.4: Standard DC IV measurements of the (a) Ta_2O_5 and (b) aVMCO devices.

RTN occurrence rate is defined as the percentage of time where the RTN is at the high state across the DC measurement.

The RTN measurement process is repeated after programming to 8 distinct levels with the goal of statistically describe RTN amplitude distributions and occurrence rates which will serve as the basis for neuromorphic simulations.

2.4 AC Programming

DC programming was discussed in the previous section, and while being a suitable method for characterization, real world applications of RRAM mostly make use of AC programming.

Figure 2.5 illustrates the basic form of AC single pulse programming in an aVMCO device.

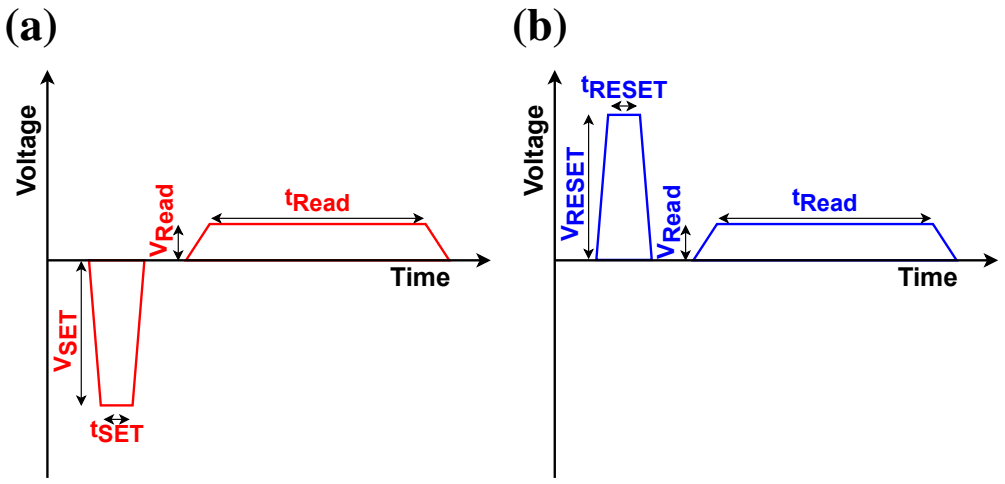


Figure 2.5: Schematic illustrating a single AC (a) SET and (b) Reset process in an aVMCO device.

As described previously in Figure 2.4b, aVMCO devices characteristically are Set at negative voltages and Reset at positive voltages. Typically, AC programming is divided into the programming step (defined by $V_{Set/Reset}$ & $t_{Set/Reset}$) and the reading step (defined by V_{Read} & t_{Read}). While the reading step is necessary for obtaining the device state at each programming step, specific programming applications may forego the reading step in certain cases where knowing the device state at each step is unnecessary.

The Ta_2O_5 device, being a filamentary device requires some form of CC to avoid hard dielectric breakdowns to occur during the Set process. While this could be achieved by coupling the RRAM device with some selectors or by using certain instrumentation systems that are capable of applying CC on AC signals, both of these solutions fall outside the scope of this work. The AC characterization for the Ta_2O_5 device in this work occurs only for the Reset step, while using DC programming with CC (as in Figure 2.4a) for the Set process.

With AC programming surges the involvement of a controlled timing parameter ($t_{Set/Reset}$)

that heavily impacts the final conductance state after programming. The combination of the impacts of both voltage and time in programming RRAM devices is known as the Voltage-Time dilemma. Figure 2.6 shows the extracted relationship of voltage and time in AC single pulse programming for a typical aVMCO device used throughout this work.

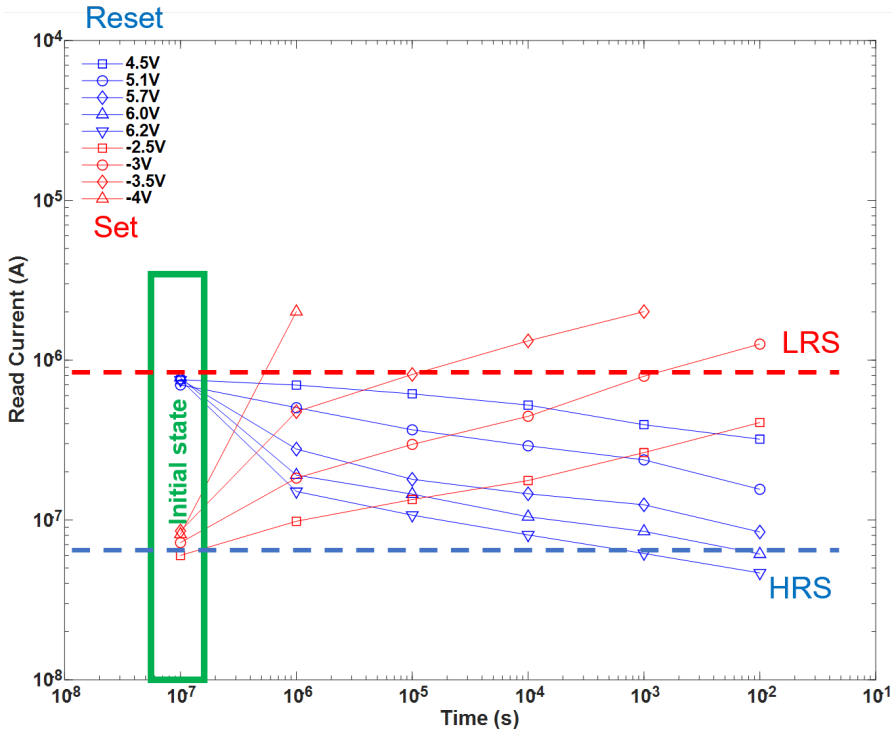


Figure 2.6: Relationship between voltage and time in single AC programming in a 135 × 135nm aVMCO device.

2.5 Neuromorphic interface and programming

AC single pulse programming can achieve different conductance states depending on its programming conditions as previously illustrated in Figure 2.6. However, RRAM

programming for neuromorphic applications aims to achieve as many distinct conductance states as possible. For this purpose RRAM is typically programmed using a voltage pulse trains with multiple pulses that gradually change the device conductance. This form of programming will be named as neuromorphic programming in this thesis work.

Many different forms of neuromorphic programming have been explored in the literature [82, 92, 93, 208], yet, in the scope of this work emphasis will be given to identical pulse train programming and staged programming.

2.5.1 Identical pulse train programming

Arguably the simplest form of neuromorphic programming, identical pulse train programming simply involves the application of a voltage pulse train with fixed voltage/-timing conditions and P_n number of pulses (Figure 2.7b).

Despite the simple nature of this programming scheme being an advantage, this usually comes with the disadvantage of typically high nonlinear programming behaviour of the RRAM (Figure 2.7a), which we denominate as Natural Response (NR).

Beyond this point, the optimum voltage amplitude must be carefully chosen, as increasing the voltage results in a higher overall On/Off ratio but at the cost of also increased nonlinearity.

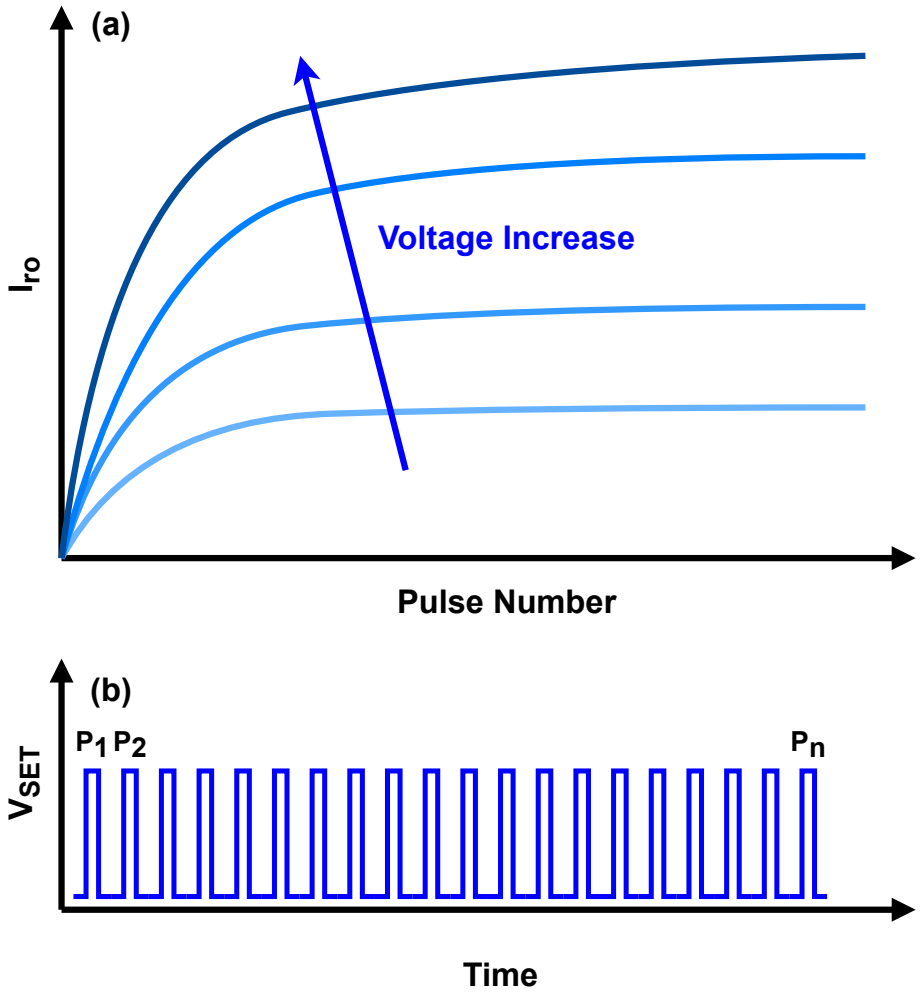


Figure 2.7: Schematic illustrating the identical pulse train programming scheme. (a) illustrates how current evolves with programming for different applied voltages (Natural Response) and (b) shows a typical example of the applied voltage pulses.

Table 2.2: Identical pulse train programming conditions used for the Ta_2O_5 and aVMCO devices.

		$V_{Programming}$	$t_{Programming}$	V_{Read}	t_{Read}	P_n	CC
Ta_2O_5	SET	$1.5V$ (DC)	DC	$0.1V$	DC	DC	$100\mu A$
	RESET	$-1.7V$	$100\mu s$		$100\mu s$	100	-
aVMCO	SET	$-2.5V$	$20\mu s$	$3V$	$100\mu s$	500	-
	RESET	$5.3V$	$100\mu s$			500	-

2.5.2 Staged programming and linear response

To address the nonlinearity issue usually present in the identical pulse train programming scheme, the staged programming scheme is proposed (Figure 2.8).

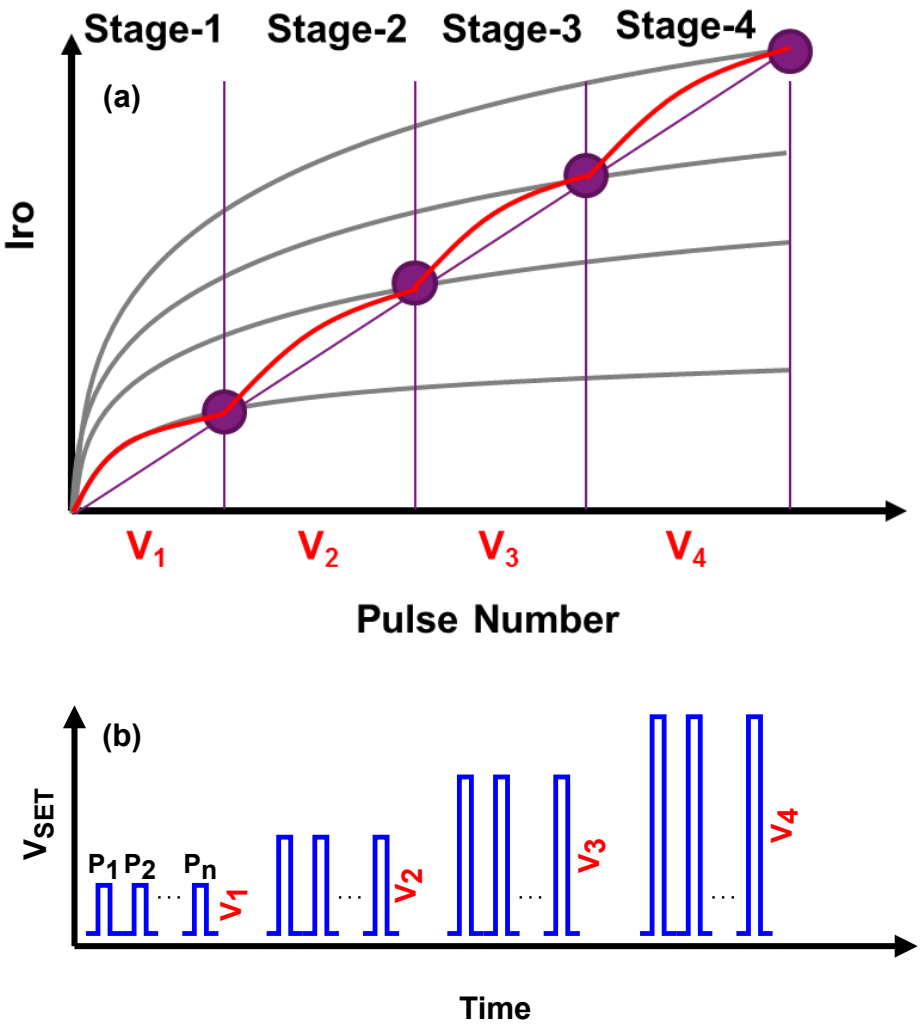


Figure 2.8: Schematic illustrating the staged programming scheme. (a) details how current evolves with programming (Linear Response) and (b) illustrates the applied voltage pulses in stages.

In this scheme, the programming is divided into stages of P_n pulses, where the voltage amplitude is increased by V_{Step} whenever the stage changes (Figure 2.8b). The underlying idea being that combining the initial low voltage amplitudes with the subsequent increasing amplitudes will allow for a more Linear Response (LR) without sacrificing the overall On/Off conductance ratio.

Methods that improve on device linearity such as Incremental Step Pulse Programming (ISPP) have already been extensively covered in literature [74], nevertheless, the staged programming scheme differs from the traditional ISPP methods in the sense that

the staged programming simply follows a predetermined pulse scheme and verification at each pulse/stage is not required, making it a more attractive solution for online training where the latency requirements turn ISPP impractical.

In this scheme, only the staged (variable) voltage conditions are touched upon while maintaining fixed timing pulse parameters. This practical decision was based upon initial observations that small variations on the voltage amplitude result in more coarse adjustments to the RRAM conductance, while the timing parameters only result in fine-tuned adjustments to conductance. As such, the scope of this work focuses on the variable voltage conditions while leaving variable timing conditions as a future prospect.

Table 2.3 details the staged programming conditions used for programming the Ta_2O_5 and aVMCO devices.

Table 2.3: Staged programming conditions used for the Ta_2O_5 and aVMCO devices.

		$V_{Initial}$	V_{Step}	V_{Final}	$t_{Programming}$	V_{Read}	t_{Read}	P_n	nStages	CC
Ta_2O_5	SET	1.5V (DC)			DC	0.1V	DC	DC	DC	100μA
	RESET	$-1.2V$	$-0.1V$	$-1.7V$	100μs		100μs	100	6	-
aVMCO	SET	$-1.5V$	$-0.1V$	$-2.6V$	300μs	3V	100μs	50	10	-
	RESET	3.5V	0.1V	5.4V	1ms			25	20	-

2.5.3 Neuromorphic programming GUI

With the creation of the staged programming scheme, the limited flexibility of the default Keysight EasyExpert suite for programming increasingly complex RRAM tests (such as the staged programming scheme) becomes apparent.

As mentioned in section 2.2, the Keysight instrument was connected to an external computer through a GPIB connection to allow the control of the instrument subroutines from inside of a script. As such, for extended practicality and flexibility, an instrument

control framework including a Graphic User Interface (GUI) was developed in Visual Basic (VB) .NET, with special focus on ease of use of the staged programming scheme. The framework is based around the Virtual Instrument Software Architecture (VISA) API and the dedicated Keysight WGFMU drivers.

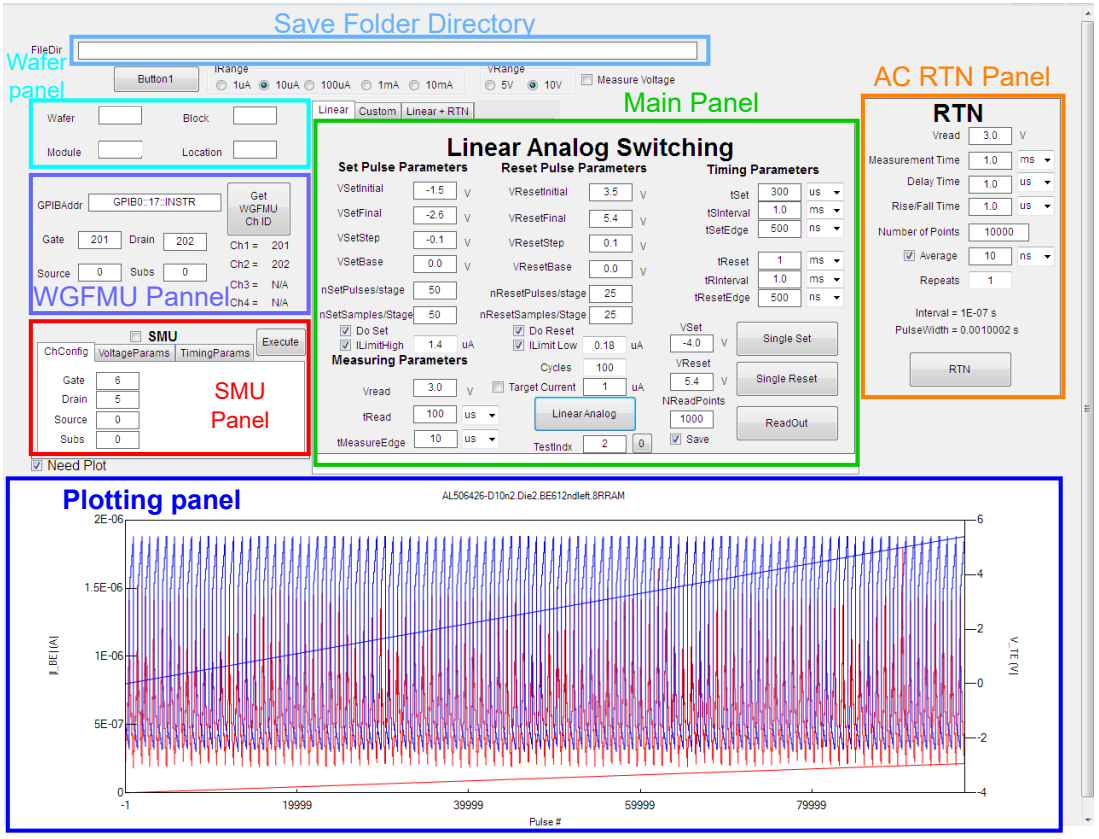


Figure 2.9: Capture of the VB GUI for neuromorphic programming.

Figure 2.9 shows a capture of the aforementioned VB GUI. The GUI is sectioned off into different panels:

- **Main panel:** responsible for setting up all of the required parameters (Voltage and timing) for testing as well as including the buttons that trigger the actual test to execute.

- **Save folder directory:** determines where all the saved files will be stored in the external computer.
- **Wafer panel:** saves information regarding the location of the tested device in the wafer, such as: Wafer #, Block, Module, Location.
- **SMU panel:** is a specific panel for the use of the instrument's SMU's for DC testing.
- **AC RTN panel:** allows for the use of the WGFMU modules to trigger fast constant voltage measurements for RTN detection of lower time constants than the capabilities of the standard DC RTN measurements.
- **Plotting panel:** displays the measured results in real-time.

The main panel includes 3 different tabs that are purposefully built for different measurements. The first tab is simply called "linear" and is displayed in Figure 2.9 and was built with the goal of facilitating the use of the staged programming scheme with a constant V_{Step}.

The "linear" tab beyond including definitions for all of the parameters of the staged programming scheme already mentioned in sections 2.4 & 2.5.2, also includes the additional parameters:

- $\mathbf{t_{Edge}}$**:** sets the rise/fall time of the pulse.
- **Cycles:** defines multiple cycles of the Set/Reset process for endurance/variability testing.

- **Target Current:** defines a target current to set the device as close to the defined value as possible.

- **I_{Limit}:** defines a ”soft” current limit that can be either high or low, stopping the current running SET/RESET process when the limit is reached.

- **nSamples/Stage:** is related to a Keysight/GPIB limitation in imposing I_{Limit}. Communication through the GPIB can be done pulse-by-pulse (which results in increased test time) or can be performed on the basis of multiple pulses per GPIB communication. This parameter defines the rate at which communication through the GPIB connection occurs.

As an alternative to the ”linear” programming approach where every stage is increased by V_{Step} from the previous stage, a more custom approach was designed.

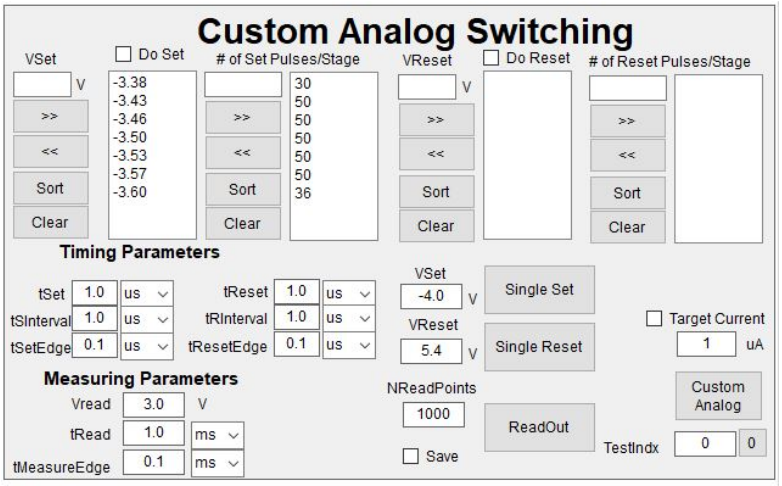


Figure 2.10: Capture of the ”custom” tab of the main panel of the VB GUI.

The second tab of the GUI called ”custom” displayed in Figure 2.10 allows the construction of a list of matching voltages and P_n for the SET and RESET processes, the instrument will then run through the custom list.

The third tab of the main panel called ”Linear+RTN” is dedicated towards the capture of RTN signals at different conductance levels and is shown in Figure 2.11.

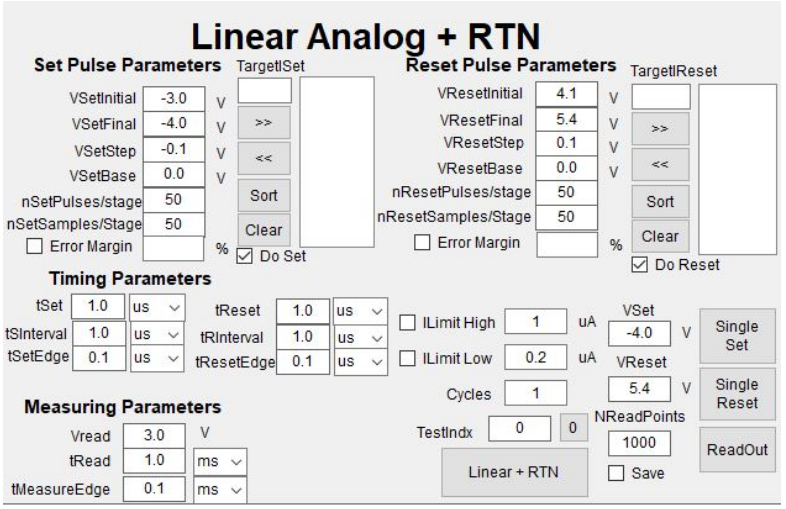


Figure 2.11: Capture of the ”Linear+RTN” tab of the main panel of the VB GUI.

This method uses the same principle introduced with the ”linear” tab, however, a list of target currents is defined for SET and RESET, each time a target current is reached during the SET/RESET process, an AC RTN measurement with the conditions defined in the AC RTN panel is run, allowing for more accessible extraction of read noises at different conductance levels in the RRAM devices.

Chapter 3

Simulation framework

In the sequence of thought provided in section 1.3.3, although there is an abundance of neuromorphic device/circuit-level simulation frameworks, we feel that the currently available software is often lacking in respects to flexibility in both NN layer topology and training algorithms, as well as in hardware-level accessibility for the end-user at runtime.

As such, a new simulation framework was designed in MATLAB, with the support of its various toolboxes, that aims to emphasize flexibility and usability, which was named FlexiNNSim. Since FlexiNNSim is focused on user flexibility, it is able to run on both Windows and Linux environments (tested on Windows 10 and Ubuntu 22.04).

FlexiNNSim is designed around a GUI that contains two main panels: variability analysis and NN definition, the former handles of the data manipulation required to calculate RRAM non-idealities, while the latter is responsible for defining NN topologies and the various training options as will be discussed in the following sections.

3.1 Variability analysis

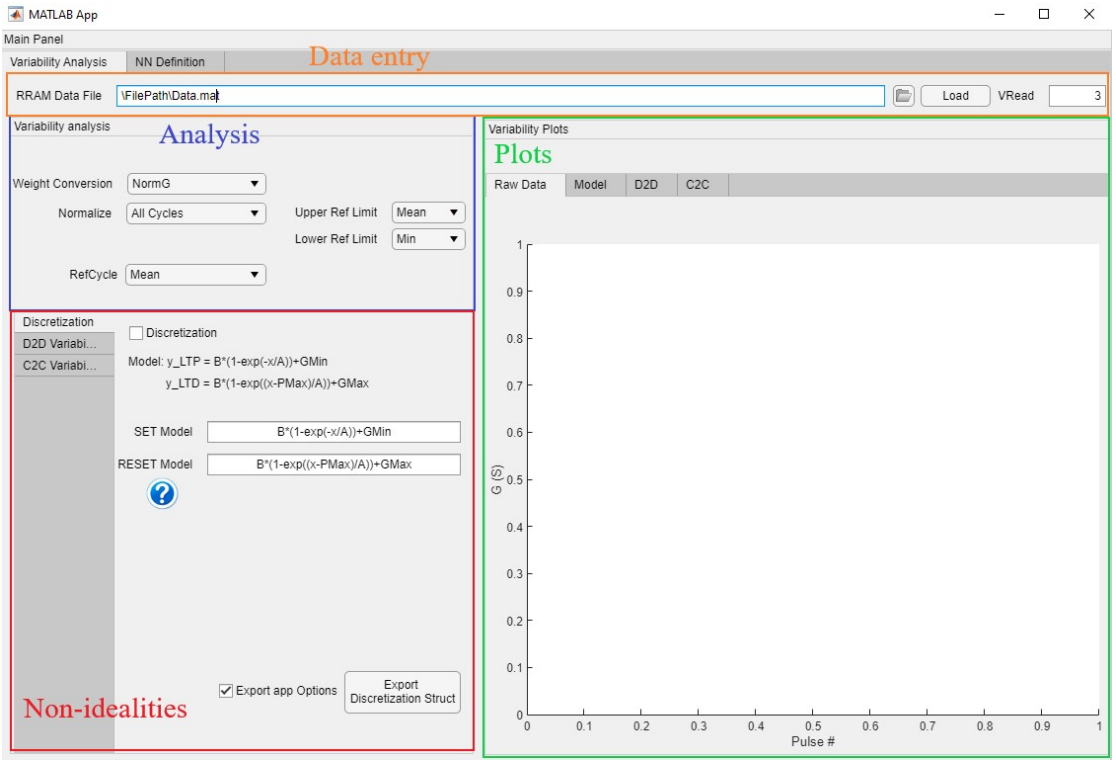


Figure 3.1: Variability analysis panel of the GUI.

The variability analysis panel shown in Figure 3.1 handles all of the functionalities required for the post-processing of the RRAM test data and is divided into four sub-panels:

- Data entry
- Analysis
- Non-idealities
- Plots

3.1.1 Data entry

The data entry section of the variability analysis panel is responsible for defining the file path of the RRAM data and to load it into the framework for post-processing. This file path can be entered manually in the RRAM Data File Edit Field or alternatively by opening the file selection dialog box by clicking on the browse button beside it.

Since the idea of this framework is meant to be able to operate data processing and NN simulation under the same MATLAB environment, all data entry and output is done with recourse to .mat files. As such, the RRAM data to be loaded should be a .mat file containing 3 columns of data ordered from left to right pertaining to: Pulse #, Voltage and |Current|. The program will automatically recognize the different polarities and switching cycles based on sign changes provided in the Voltage data (it is always assumed that SET is the first programming step). For simplicity of integration, this data organization matches the output that the Neuromorphic programming GUI (introduced in section 2.5.3) produces.

All of the calculations from this point forward are made over the RRAM conductance values, so it is necessary to provide the V_{Read} value in the VRead Edit Field.

After these considerations, pushing the Load button will import the data from the file defined in the RRAM Data File Edit Field and perform all of the necessary calculations based on the options selected in the remaining sub-panels.

3.1.2 Analysis

The analysis sub-panel displays three different dropdown menus with options on how to process the RRAM data:

- ”RefCycle”
- ”Normalize”
- ”Weight Conversion”

All of the RRAM non-idealities are calculated with respect to a reference cycle that is based on the RRAM data provided. The ”RefCycle” dropdown menu has two options:

- **”Mean”**: Calculates the reference cycle based on the mean conductance of all of the data provided.
- **”From Data”**: Allows the user to specify one specific cycle of the provided conductance data to use as the reference cycle.

Additionally, the data is converted to a 0 to 1 normalized conductance range. The ”Normalize” dropdown menu defines how that normalization procedure occurs, and has the following options:

- **”Individually”**: Normalizes each cycle to the [0, 1] conductance range independently. This method can be useful to discard the impact of On/Off conductance window fluctuations between cycles, which can have large impact in training.

- **"All Cycles"**: This method takes all of the data into account when normalizing. Conductance thresholds are set for the upper and lower limits based on one of the following: "min", "mean", "median" or "max" of LRS and HRS respectively, and any values above/below the upper/lower thresholds are capped. When using this mode, it is recommended to set the upper/lower thresholds to "min" and "max" respectively because the impact of having unreachable weight values is far greater than the cost of the slightly reduced conductance window.

Finally, the "Weight Conversion" dropdown menu specifies how conductance variability is converted into weight disturbance values and has the following options:

- **"NormG"**: In this case the disturbance from the non-ideality (D2D or C2C) is directly multiplied to the undisturbed weight. The data residuals with respect to the reference cycle are defined by:

$$NormG = \frac{G}{G_{Ref}} \tag{3.1}$$

- **"DeltaG"**: Here the variability is taken in the form of $\left|\frac{\Delta G}{G}\right|$, and is subsequently added to the undisturbed weights during simulation. The residuals are:

$$DeltaG = \left|\frac{G - G_{Ref}}{G_{Ref}}\right| \tag{3.2}$$

By default, the results presented in the following chapters are obtained using the "Mean", "Individually" and "DeltaG" options for the "RefCycle", "Normalize" and "Weight Conversion" menus respectively.

3.1.3 Non-idealities

The Non-idealities sub-panel displays all of the options that are available in terms of fitting of three different non-idealities: **Discretization** (also referred to as Conductance Stepping (GS)), **D2D** and **C2C** variability.

In all cases, the Non-idealities sub-panel allows the generation of a .mat file containing a struct that has all of the necessary information to be applied during simulation. Figure 3.2 shows an example of each of these structs.

Discretization **D2D** **C2C**

(a)

Field	Value	Size	Class
Model	*1x2 cell*	1x2	cell
Fit	*1x2 cell*	1x2	cell
DependentVar	*1x2 cell*	1x2	cell
IndependentVar	*1x2 cell*	1x2	cell
Coeff	*1x1 struct*	1x1	struct
NPulses	*1x2 cell*	1x2	cell

(b)

Field	Value	Size	Class
dist	'Normal'	1x6	char
nParams	2	1x1	double
ParamA	*1x1 struct*	1x1	struct
ParamB	*1x1 struct*	1x1	struct
WeightConversion	'DeltaG/G'	1x8	char

(c)

Field	Value	Size	Class
dist	'Weibull'	1x7	char
nParams	2	1x1	double
ParamA	*1x1 struct*	1x1	struct
ParamB	*1x1 struct*	1x1	struct
UserInput_bool	0	1x1	logical
WeightConversion	'DeltaG/G'	1x8	char

Figure 3.2: Example of the data contained in the (a) Discretization, (b) D2D and (c) C2C structs.

Additionally, the ”Export app Options” checkbox in each of the Non-idealities sub-panels also exports a list of the used options used when generating the aforementioned struct, which has no effect during simulation but can be used as a log to track the options used in multiple files.

3.1.3.1 Discretization

The Discretization non-ideality is responsible for applying the gaps (or unreachable weights) of the conductance curves. This non-ideality is always applied in respect to the reference curve, as such, the options in this sub-panel only refer to the model that should fit the reference curve data.

It is possible to define separate models for the SET and RESET portion of the reference curve in the ”SET Model” and ”RESET Model” Edit Fields respectively. The model should be written with respect to ’x’ as the independent variable. One example is displayed in the GUI itself, where the model from [94] was adapted for this interface. Alternatively, any of the MATLAB Curve Fitting Toolbox [209] library models can be used (documentation can be found in [210]), such as ”pchip” or other interpolation models that have no physical meaning in the RRAM context, but provide a perfect fit of the reference curve. An example of this panel is given in Figure 3.3.

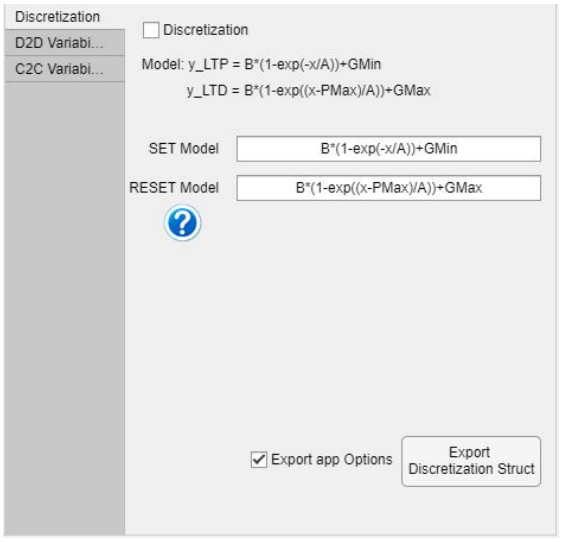


Figure 3.3: Example of the Discretization Non-ideality sub-panel.

3.1.3.2 D2D

In the D2D sub-panel it is possible to define D2D variability based on a specific user-input statistical distribution. Due to time-constraints, the current version of this framework only allows for a single distribution to be defined that represents all conductance levels in both SET and RESET programming polarities. An illustration of the D2D variability sub-panel can be seen in Figure 3.4.

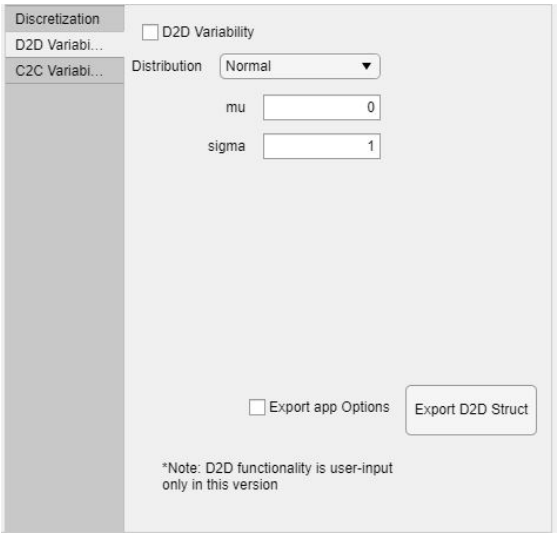


Figure 3.4: Example of the D2D variability Non-ideality sub-panel.

3.1.3.3 C2C

The C2C variability non-ideality can be defined in two different modes: ”user-input” and ”extract from data” modes. Figure 3.5 illustrates this sub-panel under the two diferent modes.

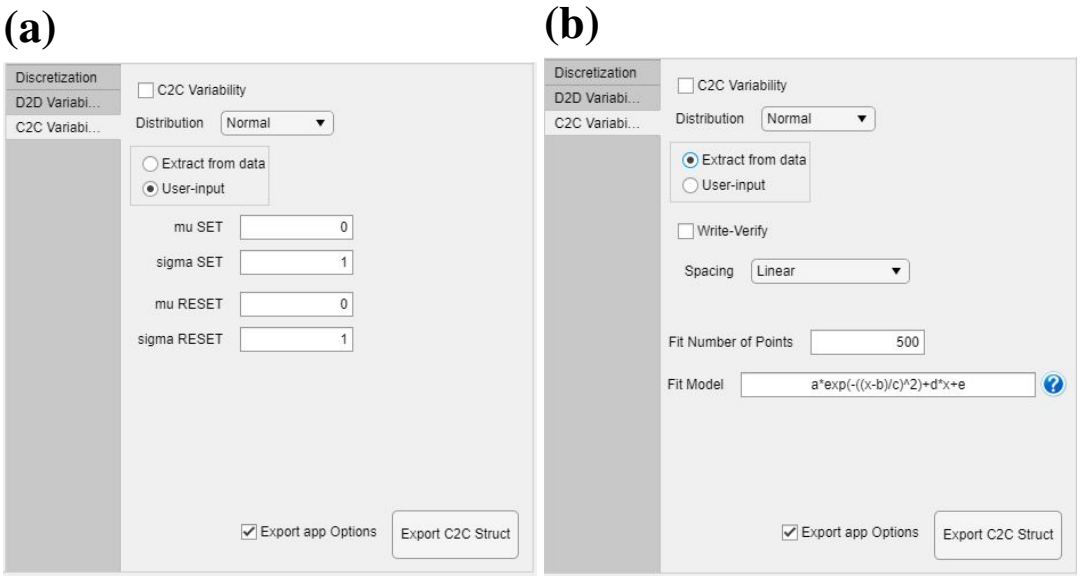


Figure 3.5: Example of the C2C variability Non-ideality sub-panel when using (a) the ”user-input” mode and (b) the ”extract from data” mode.

In the ”user-input” mode the definition is similar to the one used for the D2D variability sub-panel, where a single statistical distribution with user defined parameters are directly fed into program.

In the ”extract from data” mode, the user defines the type of statistical distribution to which the data belongs to, along with the number of points to fit along the normalized conductance range. These fit points can be seen as virtual conductance levels to which the deviation from the reference cycle will be calculated to generate the data for the C2C variability.

Along with defining the number of fit points, it is possible to define the spacing as either ”linear” or by ”pulse-number”. In the ”linear” spacing case, the points are evenly distributed across the normalized conductance range, whereas spacing by ”pulse-number” refers directly to the conductance levels available in the reference cycle. In this latter case, the number of fit points is directly extracted from the reference cycle and the user-input in the ”Fit Number of Points” Edit Field is disregarded. In the results of the following chapters, by default, the ”linear” spacing is utilized with 500 fit points.

Finally, it is possible to simulate a type of Write-Verify methodology, where the residuals are calculated on the pulse that gives the minimum error for each cycle independently, whereas the alternative without Write-Verify is calculated based on the pulse that gives the minimum error seen in the reference cycle.

Based on these options, the C2C variability is then calculated at each of the fit points using the defined type of statistical distribution, which will give different distribution

parameters (e.g. μ & σ for Lognormal or α & β for Weibull distributions) for the different fit points. The curve generated by these fit points across the normalized conductance range can then be fitted by the model defined in the ”Fit Model” Edit Field, which will then be used during NN simulation.

The statistical distribution parameters saved in the C2C struct are stored in the generic variables ”ParamA” and ”ParamB” which changes depending on the distribution type. Table 3.1 shows a list of available distributions and corresponding ”ParamA” and ”ParamB” variables:

Table 3.1: Table listing the statistical distributions available to use in the simulation framework for the D2D and C2C Non-idealities sub-panels.

dist	ParamA	ParamB
Normal	μ	σ
Exponential	μ	N/A
Extreme Value	μ	σ
Half-Normal	μ	σ
LogNormal	μ	σ
Logistic	μ	σ
LogLogistic	μ	σ
Rayleigh	B	N/A
Weibull	α (Scale)	β (Shape)

3.1.4 Plots

The plots sub-panel allows the preview of the RRAM raw data supplied to the program as well as the different fits involved in the Non-idealities sub-panel. Figure 3.6 shows examples of the different plots that can be previewed in this sub-panel.

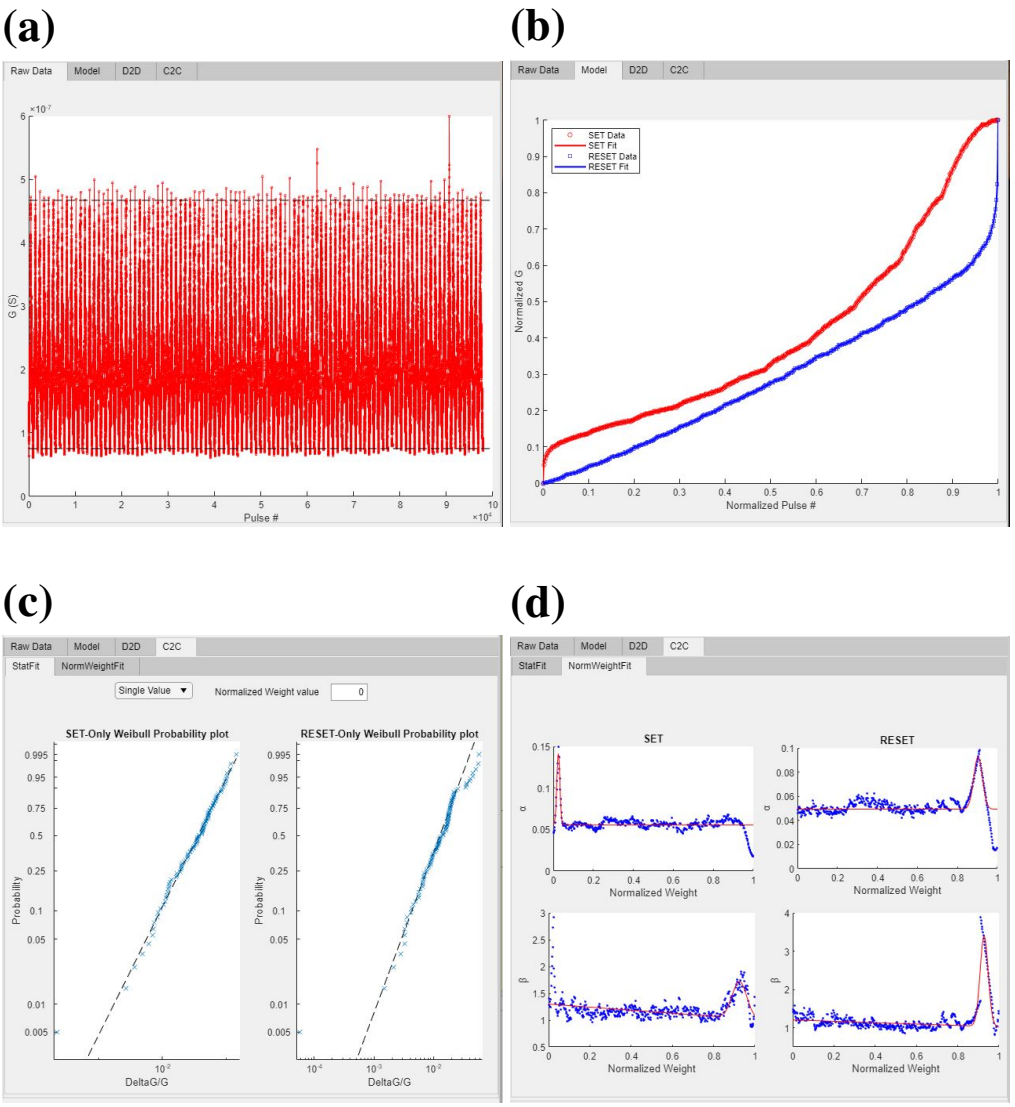
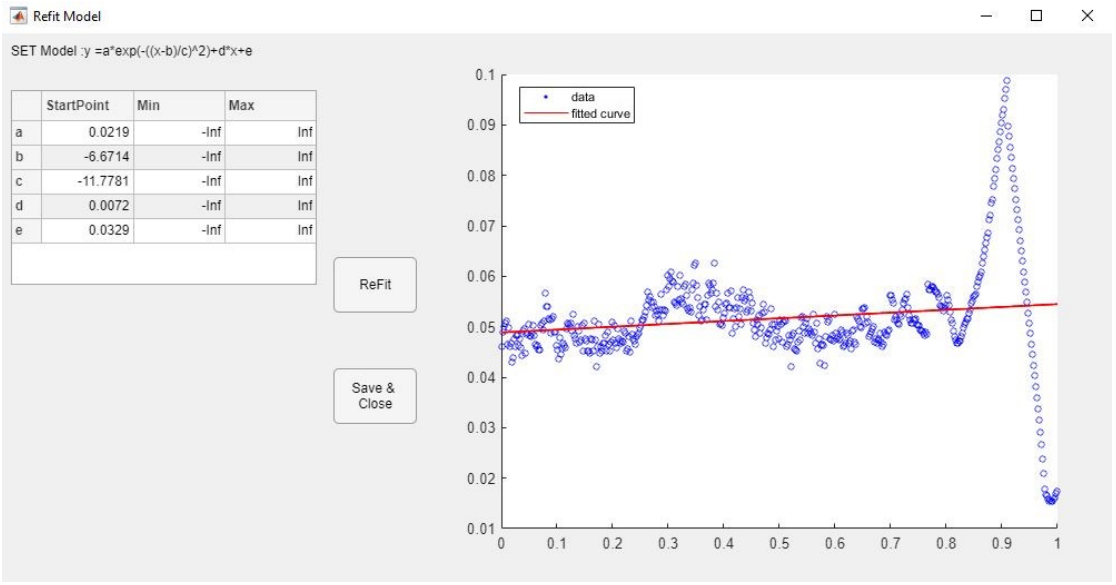


Figure 3.6: Example of the preview plots available in the Plots sub-panel. (a) shows all of the raw data that is fed into program (the black dotted lines represent the thresholds set in the "All Cycles" Normalize mode), (b) shows the fitting of the reference cycle by the model specified by the user, (c) shows a preview of the statistical fit of the C2C variability of a specified conductance level (in this case Weibull distribution) and (d) shows the fitting of the C2C variability parameters ("ParamA" and "ParamB") across the normalized conductance range for the SET and RESET polarities. The D2D variability plots preview uses the same functionality as the C2C example and is not shown for simplicity.

Clicking on one of the preview plots of either the Discretization model or the C2C variability will open a pop-up dialog box that allows to tune the fitting parameters for a more refined fitting as shown in Figure 3.7.

It should be noted that by selecting the spacing method as "linear", the C2C variability

(a)



(b)

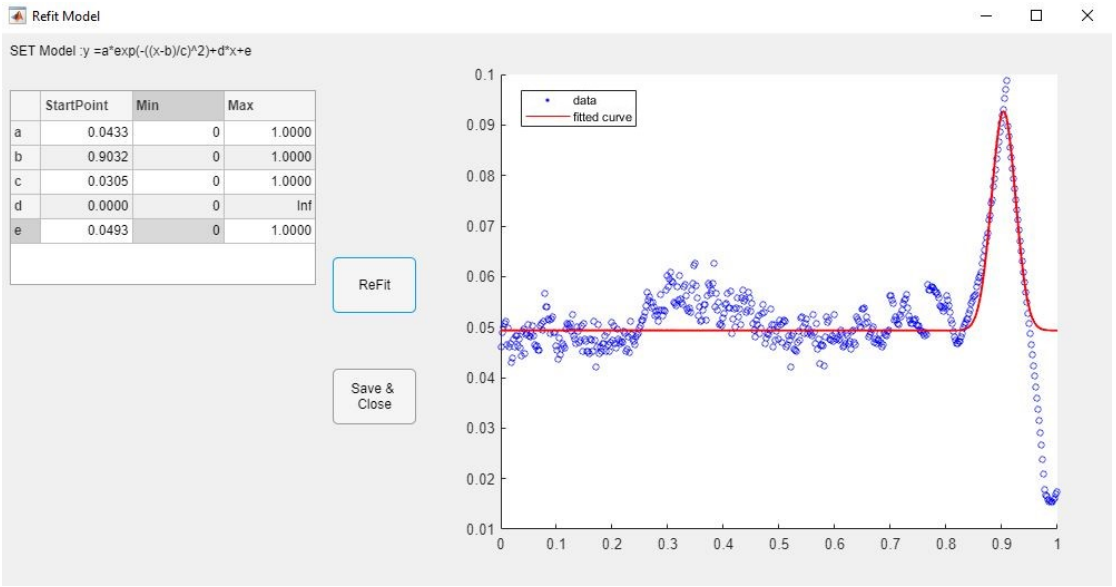


Figure 3.7: An example of the model refit window (a) before and (b) after adjusting the fitting parameters.

parameters are extracted evenly across the normalized weight range and since generally there is a region in this range (close to 0 in SET and close to 1 in RESET) where the programming nonlinearity is higher, this typically leads to an inferred peak of higher variability in these regions.

3.2 NN definition

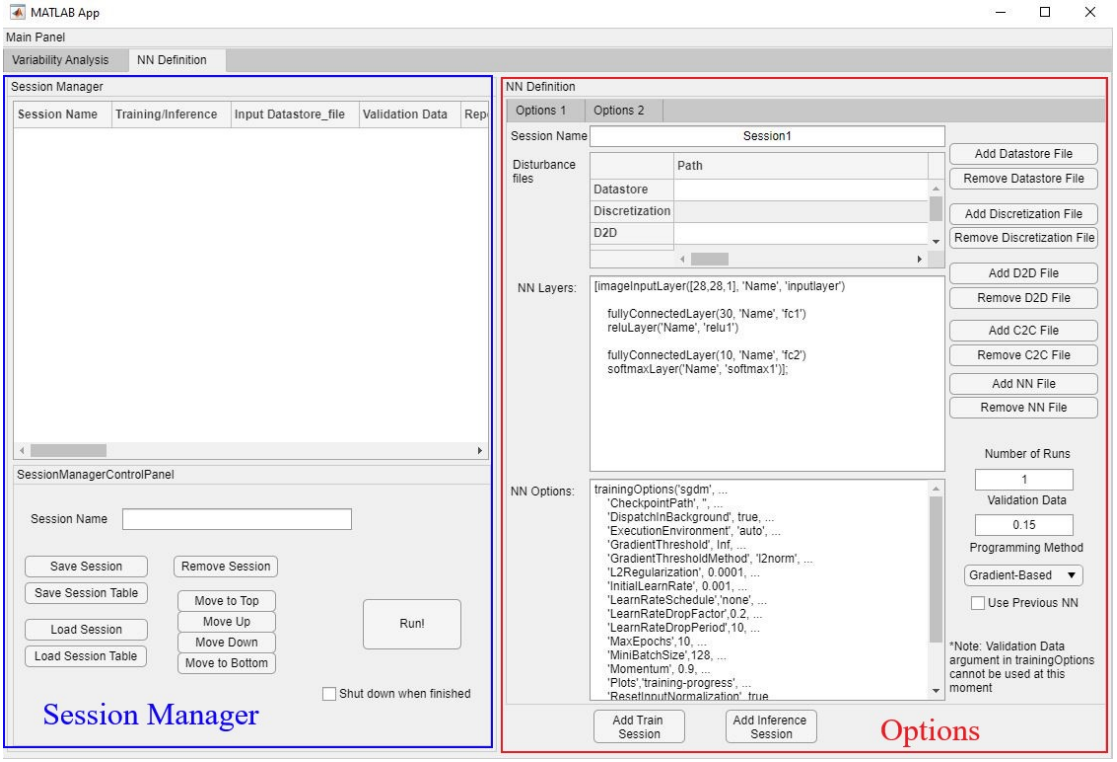


Figure 3.8: NN definition panel of the GUI.

The NN definition panel shown in Figure 3.8 handles all of the functionalities respecting to setting up the NN simulation sessions and is divided in two sub-panels:

- Session Manager
- Options

3.2.1 Session Manager

The Session Manager sub-panel allows to create a queue of NN simulation sessions. This can be particularly useful since the focus of this framework is to allow experimentation with different parametrizations. The Session Manager Control Panel allows

control of the session order in the queue as well as the saving and loading of .mat files containing Session parameters or alternatively loading whole Session tables.

Additionally, a .mat file is automatically saved in "/SessionManager/CheckpointTable.mat" at the end of each session, containing the session table of the remaining sessions. This is particularly useful in the event of an error or unexpected system shutdown since it allows to quickly resume the queue at the last checkpoint, avoiding the need to repeat sesssions that were completed.

3.2.2 Options1

The Options sub-panel defines all of the parametrization to take effect during each NN simulation session, and is divided into two tabs: Options1 and Options2 as illustrated in Figure 3.9.

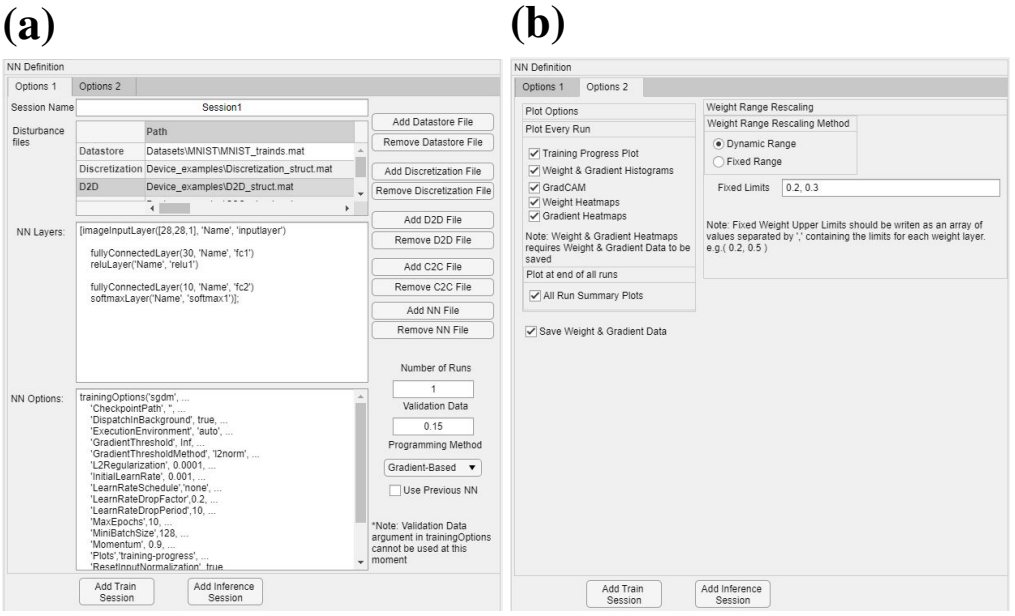


Figure 3.9: Illustration of the two tabs of the Options sub-panel: (a) Options1 and (b) Options2.

3.2.2.1 Loading files

In the first Options tab, the Disturbance files table will indicate the file paths of the dataset to train on and the Non-idealities structs generated (following section 3.1.3. An example illustrating how to fill the Disturbance files table is shown in Figure 3.10.

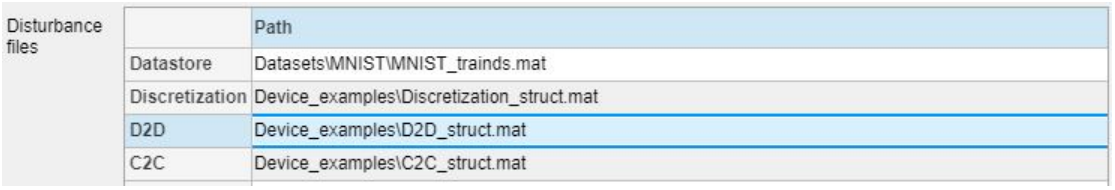


Figure 3.10: Example of the Disturbance files path definition table.

In this table, the Datastore file must be defined, while the remaining structs that pertain to device Non-idealities are optional. The Datastore file is a .mat file that contains a single datastore object that references how the dataset should be read during simulation. Detailed documentation on datastore object creation can be found in [211]. Using datastore objects for importing the datasets may have limited usefulness handling smaller datasets such as the MNIST database, but this idea will ease the testing of large datasets as well as remote datasets, as the data is loaded into the simulation framework in batches, limiting its impact on RAM.

Loading the files regarding the Non-ideality structs (Discretization, D2D or C2C) will activate its use for the particular simulation session. To deactivate a particular non-ideality simply leave its specific file path blank.

3.2.2.2 NN Layers

The NN Layers Edit field, allows the definition of the NN topology. The text entered in this field will be passed to the layerGraph [212] function of the MATLAB Deep Learning Toolbox [213] to create a LayerGraph object that is subsequently used in the dlnetwork function [214] to create the dlnetwork object used in training. Therefore all layers supported by the dlnetwork function can be used to build the NN topology. Figure 3.11 shows a simple example on how to define a 784x30x10 MLP with ReLU activations in the hidden layer.

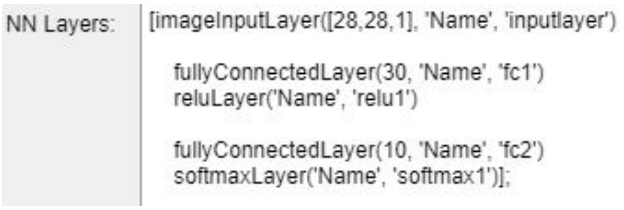


Figure 3.11: Example of the NN Layers Edit Field.

Since the network layers need to comply with the supported layers of the dlnetwork function, a limitation of the framework on this current version is the inability to choose the cost function and initialization. By default the used cost function is CE (equation 1.12) and the initialization is done with the Normalized Xavier heuristic (equation 1.40).

3.2.2.3 NN Training Options

In the NN training Options Edit field, the options regarding the training algorithm is defined along with any parametrizations for training. These options rely on the trainingOptions object of the MATLAB Deep Learning Toolbox and its specific documentation can be found in [215]. Figure 3.12 shows an example on how to setup these options.

Figure 3.12: Example of the NN training Options Edit field.

It should be noted that the 'CheckpointPath' and 'Plots' options are deactivated in this Edit field since they are handled elsewhere in the framework GUI.

The 'ExecutionEnvironment' 'parallel' option is also unavailable in the current version, but all other options are available for 'ExecutionEnvironment', it is therefore recommended to set this option to 'auto' which will search the system for a CUDA compatible GPU and will handle the calculations in the GPU if possible, if not, calculations will be performed in the CPU.

Additionally, if 'DispatchInBackground' is set to 'true' then a parallel pool based on the MATLAB default cluster will be used to handle the mini-batch inputs.

3.2.2.4 Others

Finally, the last few options in the "Options1" tab are displayed in the lower-right corner of the GUI, illustrated by Figure 3.13.

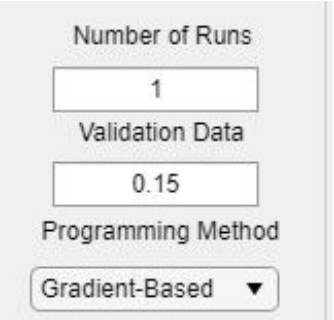


Figure 3.13: Example of the remaining options in the ”Options1” tab.

The ”Number of Runs” Edit field refers to the number of times one Session is simulated. Repeating a session for multiple runs can be useful for gathering statistically significant data.

The ”Validation Data” Edit field defines what portion of the dataset provided will be used for validation. This can be referred to as an absolute value if the value in the Edit field is higher than 1 or a fraction of the total dataset if the value is lower than 1. Unless stated otherwise, the simulations in this work are done with 0.15 of the dataset reserved for validation.

The ”Programming Method” will set how the RRAM will be programmed during simulation and consists of four different options: ”SET-only”, ”RESET-only”, ”Gradient-based” and ”Selective”. In-depth explanations of these programming modes can be found in sections 5.2 & 5.4.

Finally, the two buttons at the bottom of the ”Options” sub-panel: ”Add Train Session” and ”Add Inference Session” will pass all of the defined options to the Session Manager Table while defining if the Session will be a Inference or Training session. Although sharing similar functionality, a Training session will apply the Non-idealities

in every weight update during simulation, while the Inference session will train without Non-idealities and will only apply them after the last iteration.

3.2.3 Options2

In the ”Options2” tab the options regarding Plotting and Weight Range Rescaling are defined.

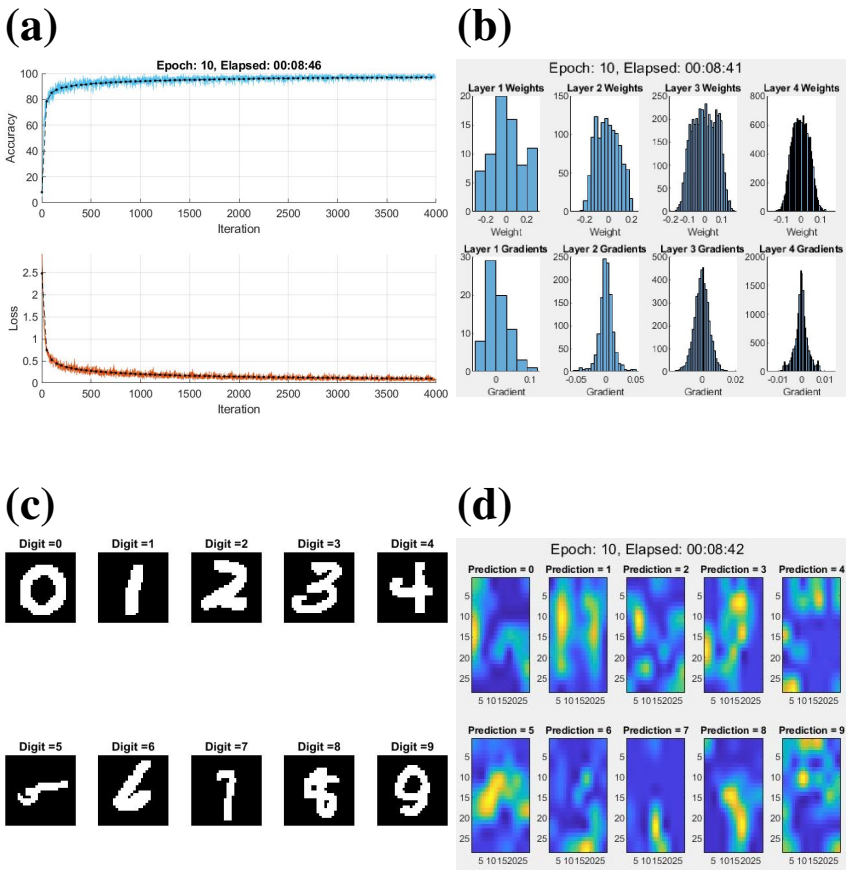
3.2.3.1 Plot Options

The ”Plot Options” is composed of checkboxes that define which plots are displayed during training. The available plots are:

- **Training Progress Plot**: Shows how accuracy and loss evolve during training and is updated every iteration.
- **Weight & Gradient Histograms:** Shows the histograms of the weights and gradients for every layer and is updated at every validation step. When the session is finished, a GIF is saved showing this plot at every validation step.
- **GradCAM:** Plots the gradient-weighted class activation mapping (GradCAM) map [216, 217]. This can be particularly useful in complex image datasets such as CIFAR-10 or CIFAR-100 to explain which parts of a certain image are being focused on by a large class CNN to make its decisions.
- **Weight & Gradient Heatmaps:** At the end of the session heatmaps are generated for each validation step, as well as a kernel density estimation (KDE) plot of

weight/gradient vs validation step. In the current version of the framework it is required that the weights/gradients are saved to allow for these plots, which can generate large files.

Figure 3.14 shows examples of the available plots in this framework.



3.2.3.2 Weight Range Rescaling

In respects to the Weight Range it can be set as a Fixed Range to mimic the RRAM limited conductance range or it can be set as a Dynamic range that changes in every iteration. An in-depth explanation of the mechanisms of the dynamic weight range rescaling can be found in section 5.7.

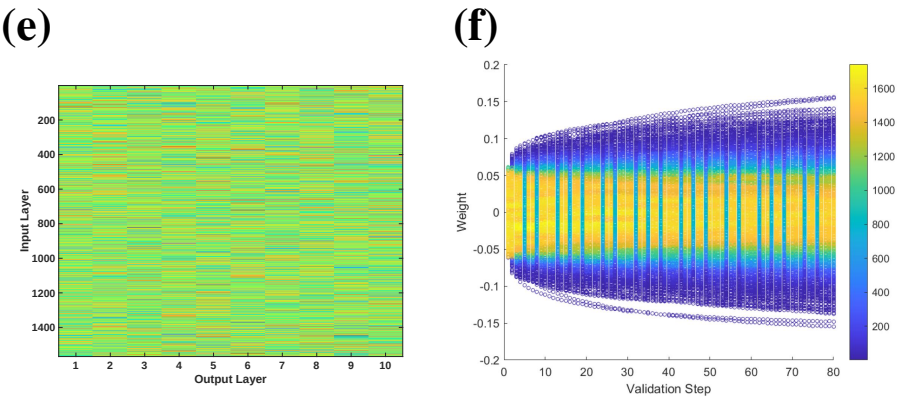


Figure 3.14: Examples of the plots available in the simulation framework. (a) shows the Training Progress Plot, (b) the Weight & Gradient Histograms, (c) the Test Images to be analysed by GradCAM, (d) GradCAM, (e) Weight Heatmap of the final layer of a CNN and (f) its respective Weight vs Validation step KDE.

In the case of using a Fixed Range, the weight range limits for each NN layer must be specified in the ”Fixed Limits” Edit field. Each scalar value that is written in this edit field will define a range as being the minimum of the negative weights crossbar and the maximum of the positive weights crossbar (e.g. inputting a value of 0.2 for one layer will define the limits as [-0.2, 0] for the negative crossbar and [0, 0.2] for the positive crossbar).

3.3 Summary

In this chapter, a simulation framework named FlexiNNSim was designed and written from the ground up using MATLAB programming language, with the support of its various toolboxes to be able to handle both the data analysis to determine RRAM non-idealities (Discretization, C2C and D2D) based on experimental electrical characterization data, as well as the application of those non-idealities in a neuromorphic simulation environment.

The main focus of this framework is on flexibility and accessibility of testing. As such, all of the framework's functionalities are accessible through a GUI that allows easy manipulation of both the way RRAM data is analysed (according to the desired programming methodology) as well as NN topology and hyperparameter settings. This framework constitutes the basis that allows for the development of the contents displayed in the following chapters.

Chapter 4

Impact of RRAM non-idealities on inference

The emergence of oxide based RRAM devices as prospective candidates for integration as synapses in large scale NNs due to its MLC capability, low energy consumption and CMOS-compatible 3D integration potential [218] has been a hot topic lately. However, noise and variability caused by the stochasticity of defect movements in the oxide layers and its impact on the overall NN inference accuracy remains a major concern [219].

In this chapter, the focus will be on the impact of read noises, as well as programming variability on the inference process of feedforward NNs. Two different devices based on distinct switching mechanisms: filamentary (Ta_2O_5) and nonfilamentary (aVMCO) are tested and compared between them. In terms of non-idealities, the focus will be

on the effects of read noises (with special emphasis on RTN), as well as programming induced variability (PIV) on the accuracy of a trained NN during inference.

4.1 Impact of RTN

As previously mentioned in section 1.1.4.5, RTN is a form of read noise uniquely characterized by the fluctuation between two distinct current states, caused by trapping/detrapping of defects.

As RRAM devices are scaled down below 10nm [56], the impact of singular defects on the read current can become significant [220] leading to memory window reduction and read errors.

This section will focus on the quantitative description of RTN fluctuations for both CF (Ta_2O_5) and NCF (aVMCO) RRAM devices and its comparative analysis on the impact of RTN on the accuracy of the inference process of a feedforward NN [104].

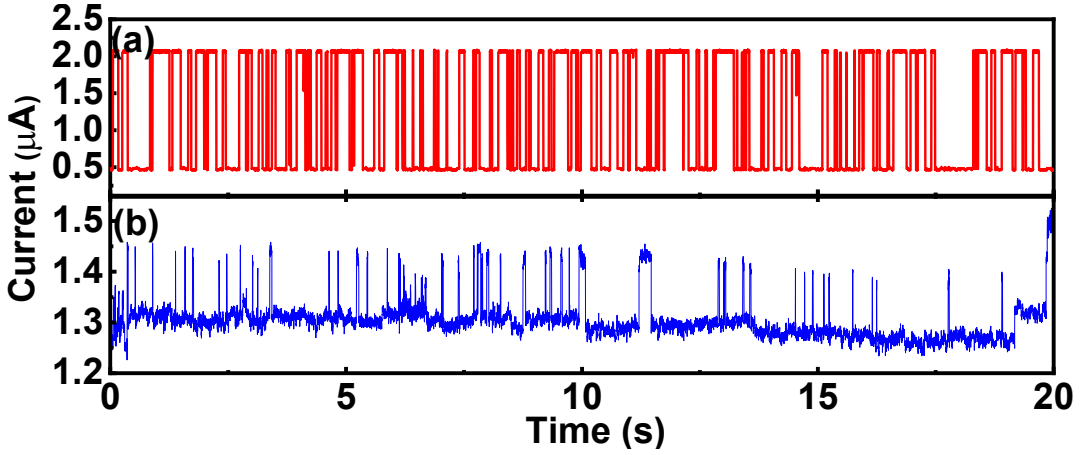


Figure 4.1: Examples of RTN signals captured in (a) Ta_2O_5 and (b) a-VMCO devices.

As shown in Figure 4.1, the maximum relative RTN amplitude ($\Delta I/I_{read}$) can be as high as $\approx 300\%$ in the Ta_2O_5 device, but only $\approx 10\%$ in the NCF aVMCO device.

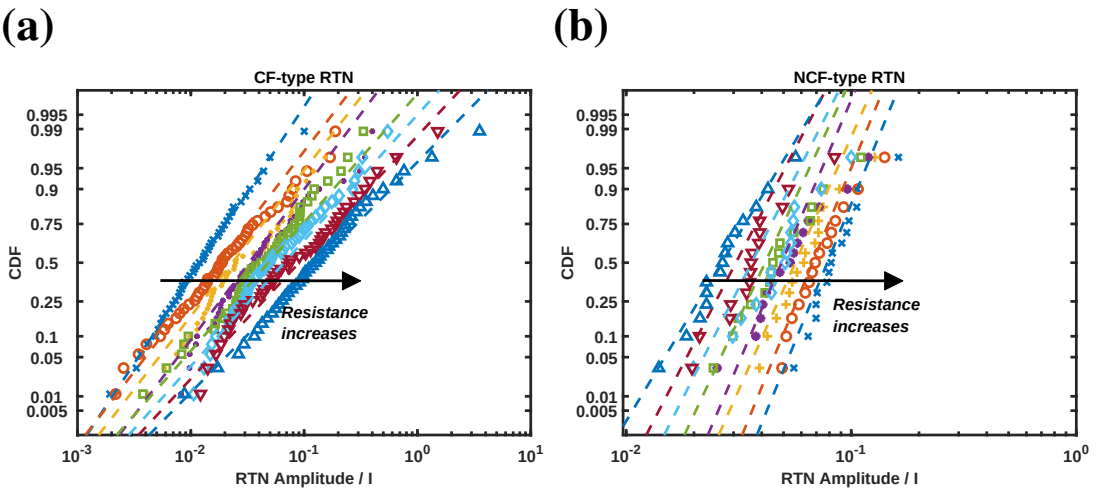


Figure 4.2: Lognormal distributions of the relative RTN amplitude (RTN amplitude/I) of 8 distinct resistance levels for the (a) Ta_2O_5 and (b) aVMCO devices.

Taking the CDF distributions measured at 8 different resistance levels show that the RTN amplitude follows lognormal distributions in both devices [221], as shown in Figure 4.2. Besides the higher RTN amplitude, the CF device also presents increased spread in its CDF, ranging anywhere between 0.1% to 300%, compared to its NCF counterpart, varying only between 1% to 10%.

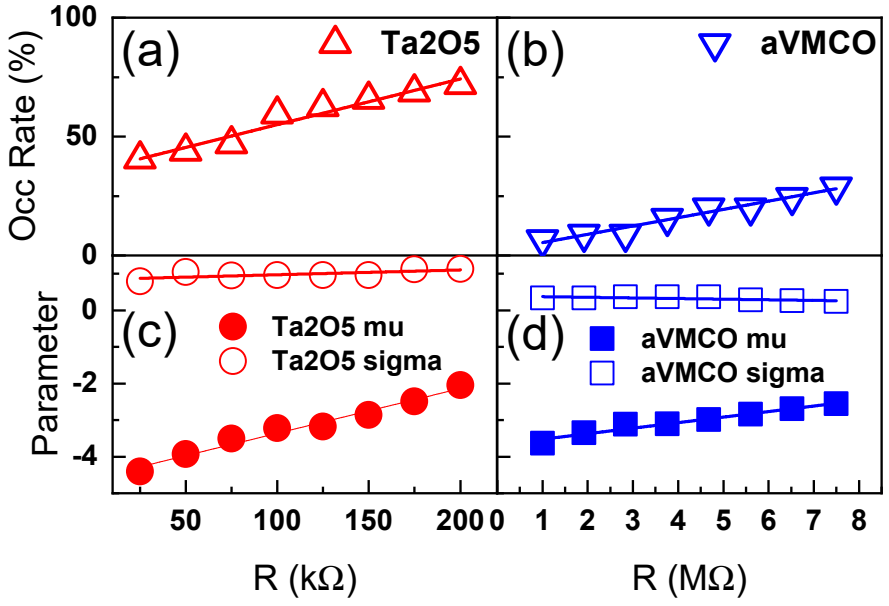


Figure 4.3: Occurrence rate of RTN signals at 8 distinct resistance levels in the (a) Ta_2O_5 and (b) aVMCO devices. (c - d) Extracted parameters from the lognormal distributions at 8 levels for Ta_2O_5 and aVMCO devices respectively.

In both devices, the occurrence rate of RTN increases linearly with resistance, nevertheless, the occurrence of RTN in the Ta_2O_5 device is much more pronounced than in aVMCO, as shown in Figure 4.3a & b.

The significant difference in RTN amplitude distribution and occurrence rates can be attributed to the different dynamics responsible for switching in both devices. While the CF Ta_2O_5 resistance switching is caused by the formation/rupture of a conductive filament, the NCF aVMCO operates through an areal modulation switching mechanism (section 1.1.3.1).

In the case of the Ta_2O_5, at HRS, there are only a few critical defects in the constriction of the CF responsible for current conduction, and as such, trapping/detrapping of these critical defects lead to large RTN amplitudes that increase with resistance [222].

In the NCF aVMCO however, switching by the uniform modulation of the defect profile limits the contribution of singular defects in the overall conduction, which in turn leads to reduced RTN amplitude and occurrence rate.

The PDF and CDF of a lognormal distribution are described as:

$$y = f(x|\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}} \text{ [223]} \tag{4.1}$$

$$p = F(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_0^x \frac{e^{\frac{-(\ln t-\mu)^2}{2\sigma^2}}}{t} dt \text{ [223]} \tag{4.2}$$

respectively. The mean (μ) and standard deviation (σ) are extracted from 8 resistance levels and displayed in Figures 4.3c & d.

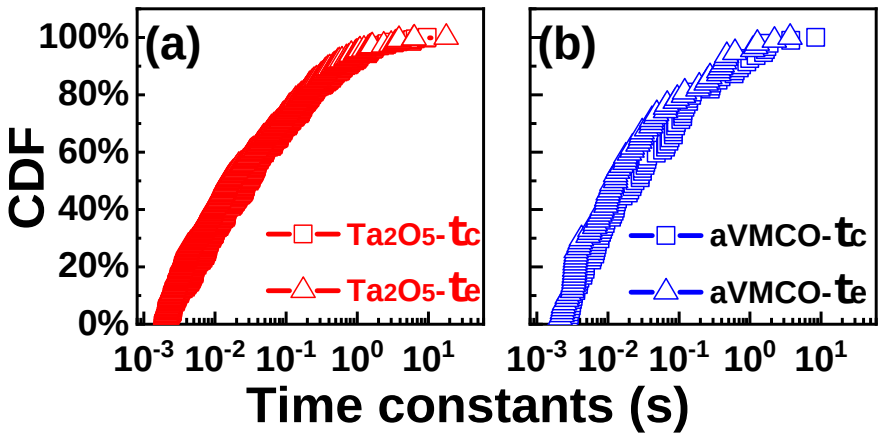


Figure 4.4: CDF of RTN time constants (a) in Ta_2O_5 and (b) aVMCO devices.

The RTN time constants are extracted using a Hidden Markov Model (HMM) for both devices and are shown in Figure 4.4. Both devices exhibit similar distributions in regards to their time constants, and for the purpose of a comparative analysis, we consider that the impact of time constant is equivalent for both CF and NCF devices.

To carry out the comparative simulation of RTN in an inference NN between the two devices, the testing procedure is divided into three steps:

1. NN is trained using a minibatch SGD algorithm. A single hidden layer MLP with a topology of 784x30x10 and sigmoid activations is used as the standard benchmark in this work as shown in Figure 4.5a. This NN is firstly trained in an ideal manner without the involvement of RRAM non-idealities, which typically wields a satisfactory accuracy of $\approx 95\%$ [177, 224, 225].

2. RTN induced disturbance based on the RTN amplitude lognormal distribution parameters and the occurrence rate is generated for each synapse and applied accordingly. The parameters μ, σ and the occurrence rate are linearly fitted and interpolated for the normalized weight values (Figure 4.3).

3. An accuracy comparison is established between the well trained software benchmark and the RTN disturbed NNs for both devices.

It should be noted that the weights in this simulation can be both positive and negative. To implement this feature in a hardware synaptic array, two separate RRAM crossbars are required, one for positive and one for negative weights (see section 1.3.1.4).

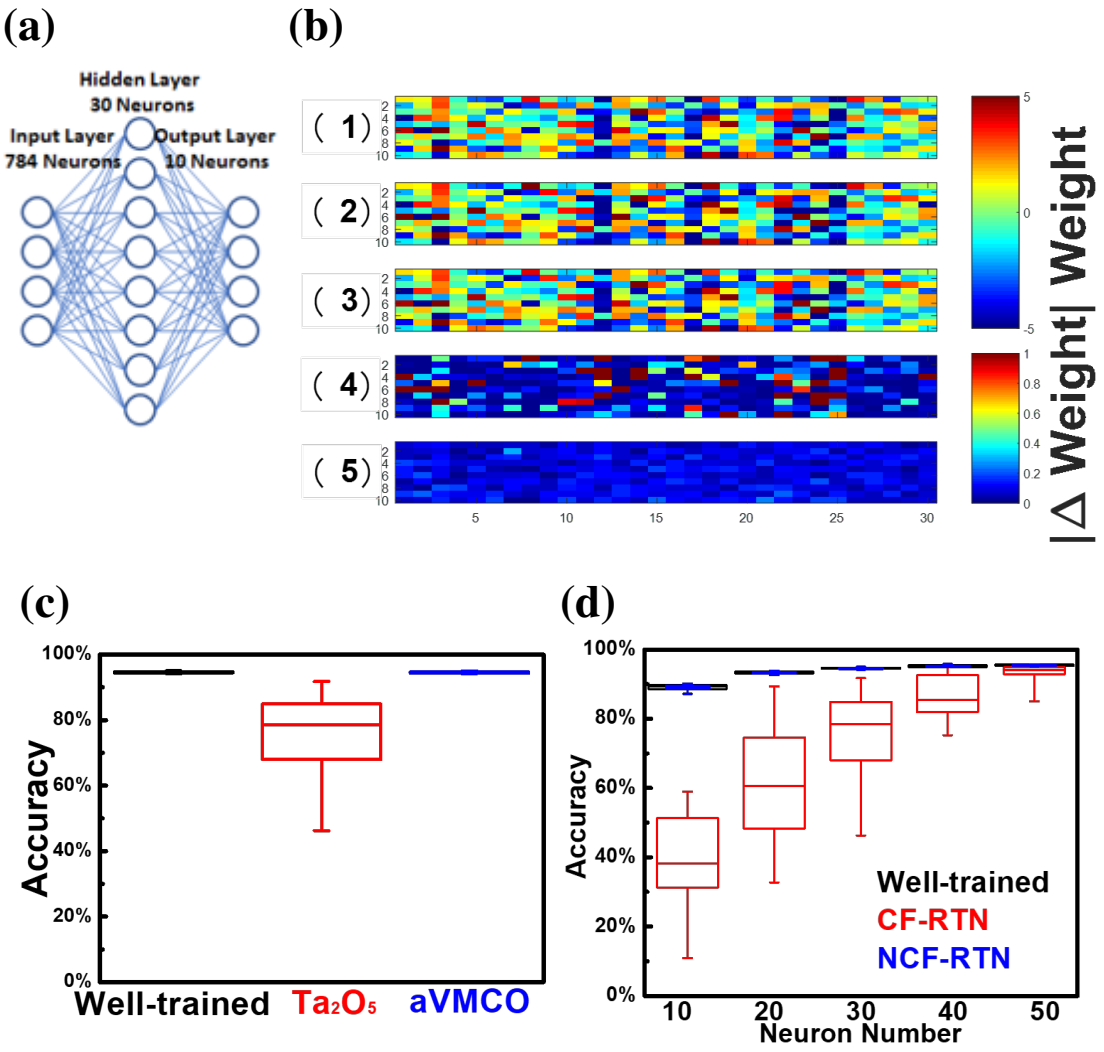


Figure 4.5: (a) Topology of the used pattern recognition NN. (b) Visualization of weights: (1) directly after training; (2) with CF RTN disturbance; (3) with NCF RTN disturbance; (4-5) their differences to case (1) respectively. (c) Statistical accuracy in 50 training-disturbance procedures. (d) Accuracy comparison of NN with different number of neurons in the hidden layer.

The change in weights caused by RTN disturbance procedure can be visualized in Figure 4.5b, in which the weights are shown in (1) without disturbance, (2) after the CF disturbance, and (3) after the NCF disturbance. The weight differences (ΔWeight) to case (1) are shown in (4) for CF and (5) for NCF.

To obtain a statistically reliable result, the training-disturbance procedure is repeated 50 times for both CF and NCF devices, and the accuracy after disturbance is shown in Figure 4.5c. After the CF RTN disturbance, the average accuracy drops to $\approx 75\%$ with a wide repeatability distribution, having its lowest accuracy below $\approx 50\%$, while after the NCF disturbance, the accuracy only drops negligibly to $\approx 94\%$, with similar repeatability to the inference without disturbance. This proves that the NCF RRAM has a significant advantage for synaptic application when compared to the conventional CF devices, due to its small RTN amplitudes and low occurrence rate.

Furthermore, as shown in Figure 4.5d, the NN with NCF devices is able to maintain a high accuracy of $\approx 90\%$ when reducing the number of hidden nodes to only 10, whilst the accuracy with CF devices drops sharply with reduced hidden layer size. This shows that NNs built with NCF devices are much more resilient to RTN and can achieve higher accuracy while using less neurons and synapses than in the CF case.

One concern is that the impact of RTN may diminish in neurons with large fan-in, as the averaging effects of independent variation sources scales following the $\frac{1}{\sqrt{N}}$ rule. To examine this possibility, the impact of RTN using different input layer sizes (and therefore different neuron fan-in) is studied. As shown in Figure 4.6a-c, the input layer is resized from the standard 28x28 pixels (784 input neurons), to a downsized 14x14 scale (196 input neurons) and an upscaled image of 56x56 pixels (3136 input neurons).

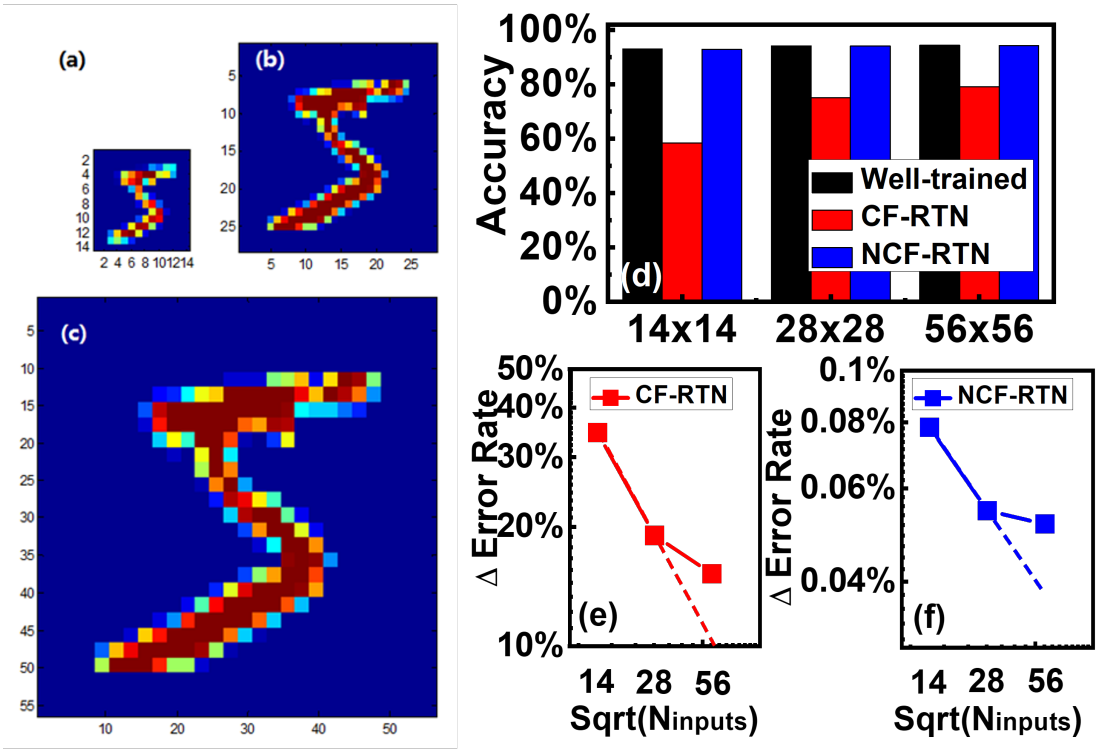


Figure 4.6: Pattern recognition accuracy with MNIST images of different resolutions. (a - c) Example of the rescaled MNIST image with (a) downscaled 14x14 pixels, (b) original 28x28 pixels and (c) upscaled 56x56 pixels. (d) Effect of RTN disturbance on the different input layer scaled NNs. (e - f) log-log plot of relative error rate $(\text{Accuracy}_{\text{well-trained}} - \text{Accuracy}_{\text{CF or NCF-RTN}})$ against square root of the neurons in the input layer (N). The straight dash lines are guides for the $\frac{1}{\sqrt{N}}$ scaling rule.

Figure 4.6d-f illustrates the influence of RTN while varying the size of the input layer in our neural network. As expected, increasing the number of input nodes generally leads to improvements in accuracy and lower error rates. However, it's worth noting that these improvements are not strictly following the $\frac{1}{\sqrt{N}}$ rule, where N represents the number of input nodes. Instead, we observe only slight improvements with the expansion of the input layer. This deviation from the expected rule can be attributed to the nuanced roles that different neurons play within the neural network. While some synapses may serve redundant functions in the network, others are more critical in pattern recognition tasks. The weight fluctuations in these critical synapses can have a more pronounced impact on the overall accuracy compared to less critical ones, thereby weakening the

averaging effects of increased neuron count. One noteworthy observation is that RTN continues to be a significant factor even in larger neural networks. This finding challenges the assumption that the impact of RTN diminishes in neurons with large fan-in, as predicted by the $\frac{1}{\sqrt{N}}$ rule. Instead, our results suggest that the complex interplay of neuron roles and the nature of RTN make the relationship between neuron count and performance more intricate than previously anticipated. Consequently, addressing RTN remains an important consideration in the design and optimization of neural networks, especially in cases where specific neurons play pivotal roles in the network's function.

4.2 Impact of other read noises

Besides the impact of RTN discussed in the previous section, there is also the concern of other forms of read noise that can impact inference performance, such as thermal noise [96] and $1/f^{\alpha}$ [97, 98].

This section is dedicated to investigating the impact of other read noises (ORN) besides RTN on the accuracy of a feedforward inference neural network. It is important to note that RTN and ORN are often intertwined within the same signal, which can make it challenging to distinguish their effects. To address this challenge and gain a deeper understanding of their individual impacts, it is necessary to decouple these two forms of noise into separate signals.

In particular, random telegraph noise (RTN) exhibits distinctive time constants associated with its behavior. RTN is typically induced by electron trapping and detrapping processes, which require overcoming energy barriers between the electrode's Fermi

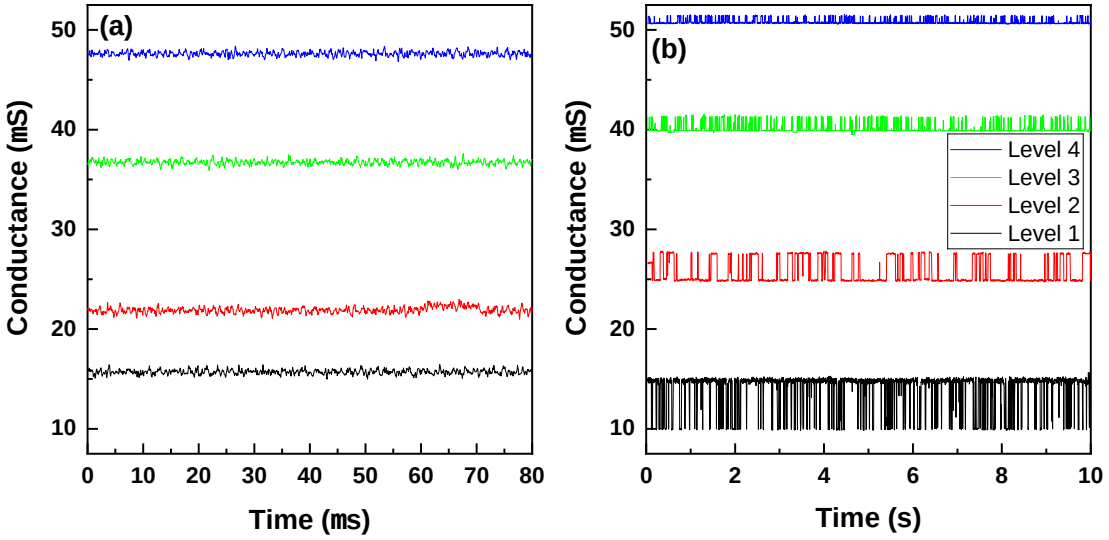


Figure 4.7: (a) read signals shorter than 100μs. (b) read signals between 10ms and 10s, captured in a CF device.

level and defects within the material[226, 227]. These processes occur on a relatively slow timescale, typically longer than 100μs (Figure 4.4 [104]).

To separate RTN from other read noises, such as ORN, it is crucial to capture signals on a timescale that is significantly faster than the RTN capture and emission constants. Figure 4.7 provides a visual representation of this concept by programming a CF device at 4 different conductance levels and probing its read signals at 0.1V in different timescales. Figure 4.7a shows a reduced timescale of 20ns to 80μs (captured with Keysight B1530A WGFMU), where we observe no visible RTN signals, while ORN signals become more apparent. However, when using the Keysight B1500A SMUs, the read signal timescale is extended between 10ms and 10s, RTN becomes prevalent (Figure 4.7b).

This observation is not only important for distinguishing between RTN and ORN but also has practical implications. It demonstrates that by capturing signals on a faster

timescale, it becomes possible to minimize the effects of RTN on PIV. When programming the device using read signals in the same timescale as ORN, we can effectively isolate the influence of ORN on programming variability. This decoupling is essential for a more accurate assessment of the impact of each noise source on the performance of our feedforward inference neural network.

It should be noted that even though this is an effective method towards decoupling RTN from ORN, further decoupling ORN into other separate forms is much more complex, and as such, in the scope of this work, these other forms of read noise ($1/f^{\alpha}$, thermal) will simply be designated as ORN.

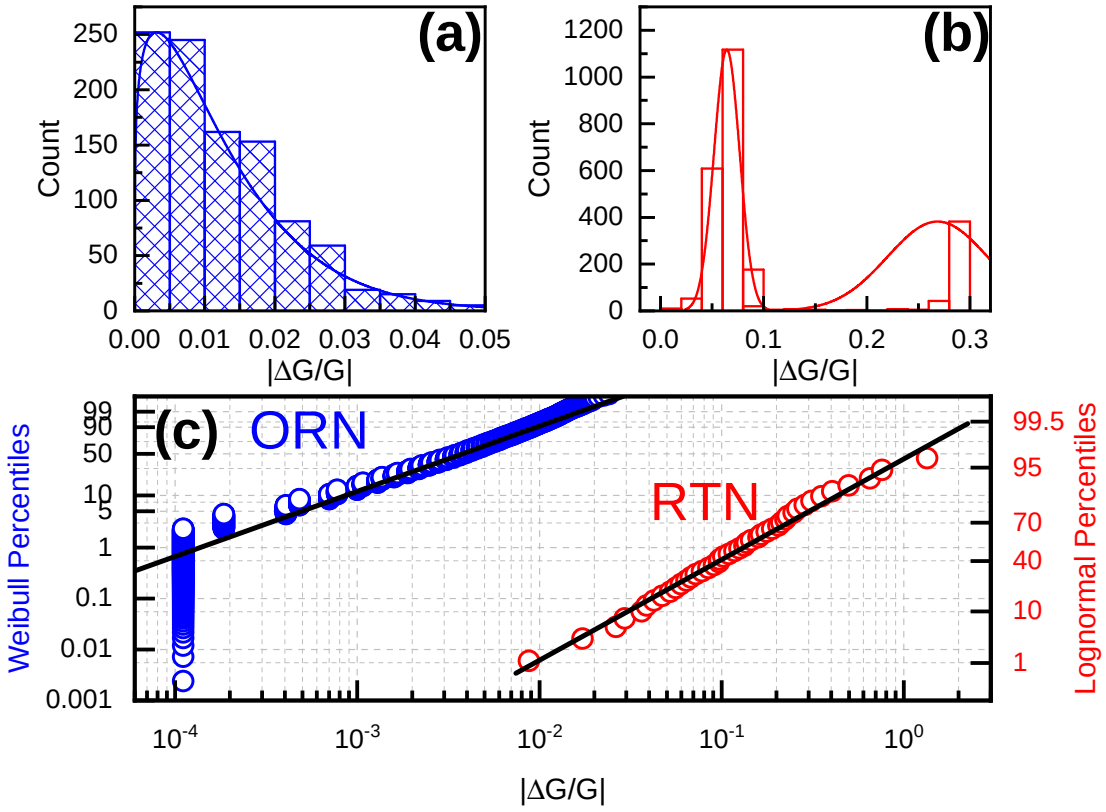


Figure 4.8: Comparison of the read variability induced by RTN and other read noises captured in a CF device.

Figure 4.8 compares the magnitude ($|\Delta G/G|$) of a RTN and ORN signal taken at similar conductance levels in a CF device.

While RTN can be typically described by a lognormal distribution, Weibull better fits the ORN data in CF devices. Most importantly, when comparing the magnitude of both distributions (Figure 4.8c), ORN is generally one order of magnitude lower than RTN.

In terms of occurrence rate however, we consider that ORN is an ever present source of read noise, in practical terms, being impossible to fully remove from any of the measurements taken, and as such, we consider that the occurrence rate for ORN is 100%.

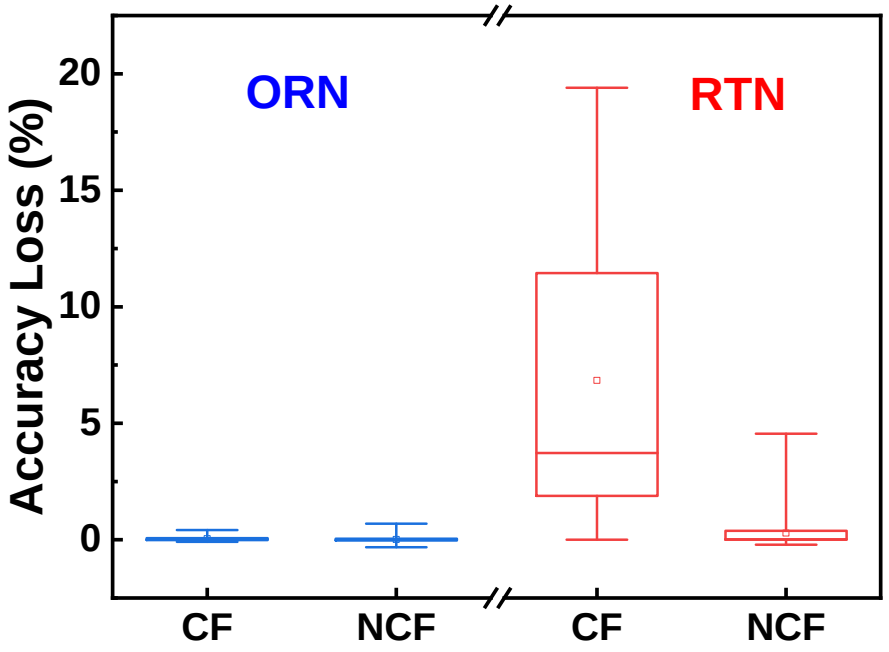


Figure 4.9: Pattern recognition accuracy loss comparison between ORN (blue) and RTN (red) in both CF and NCF devices.

With the previous considerations in mind, the impact of ORN is simulated in inference and its comparison with the RTN from the previous section is displayed in Figure 4.9. The very low magnitude of ORN ($\approx 10^{-3}$) makes its impact in inference negligible, being that the accuracy loss induced by this form of noise is typically lower than 1%, whereas with RTN, the accuracy loss can range from having a low impact with NCF devices (1% to 5%) to a significant degradation in CF devices (up to 20%).

In summary, the separation of RTN from other read noises on a faster timescale is not only a crucial step for understanding their individual impacts but also provides valuable insights into mitigating PIV when using read signals in the same timescale as ORN during device programming.

4.3 Impact of programming variability on inference

Programming induced variability (PIV) in RRAM can be seen as the type of variability that originates from deviations of a pre-set target conductance level at each programming stage. PIV originates from the stochastic ionic defect movement during the program operation, which could be affected by different physical switching mechanisms and programming schemes.

It has already been reported that under carefully controlled operation conditions, negligible difference in resistive switching variability is seen between C2C and D2D variability, indicating the intrinsic nature of RRAM variability [228] and as such, in this section, the broader category of PIV is used to test the impact of variability under different programming schemes and device types in inference.

The linearity of synaptic weight updates driven by device conductance change due to consecutive input pulses is a critical parameter of synaptic devices. The conventional programming scheme (section 2.5.1) consists of a train of identical square pulses. Under this scheme, the device's conductance changes dramatically during the initial few pulses and becomes saturated as the number of pulses increase, causing large non-linearity [92].

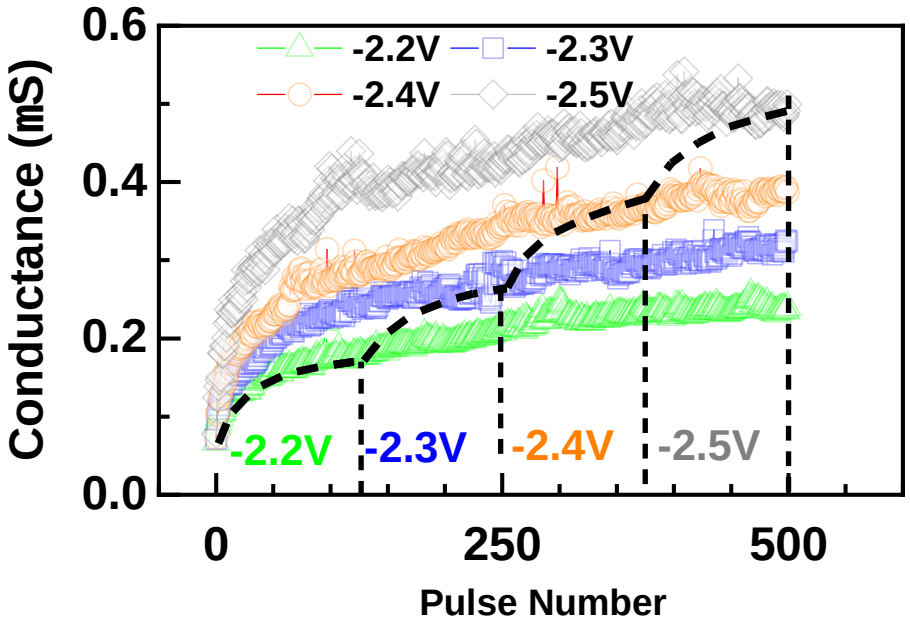


Figure 4.10: Demonstration of the saturation of the aVMCO NR at different voltage amplitudes. The dotted line represents the expected LR when using the staged weight update scheme.

On the other hand, it has been discovered that the shape of the exponential-like NR is dependent on the pulse amplitude and width. When the device is gradually programmed into the saturation region, changing to a stronger pulse condition could speed up the conductance change and alter the saturation conductance level as illustrated in Figure 4.10.

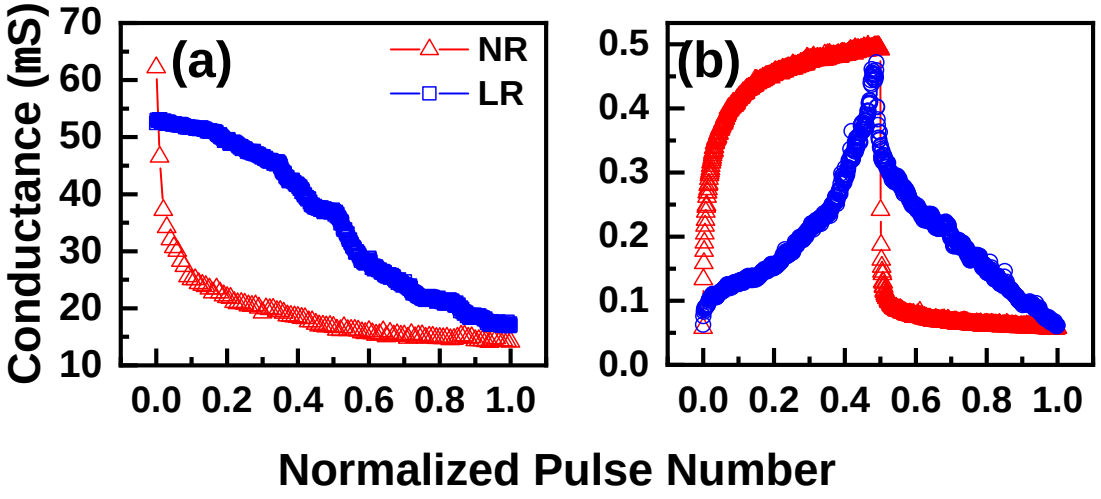


Figure 4.11: Typical examples of Natural (red) and Linear (blue) responses captured in (a) a Ta_2O_5 and (b) aVMCO devices. In Ta_2O_5, only RESET is represented, while in the aVMCO device, both SET and RESET are represented.

Figure 4.11 shows how the linearity improvement on the gradual conductance programming reference curves when using the staged programming scheme introduced in section 2.5.2 with the conditions from table 2.3. It should be noted that at this stage the SET programming step in the CF Ta_2O_5 device is only possible using DC programming conditions with CC to avoid hard device breakdown, as is the case for typical CF devices (section 1.1.3.3), and as such, only the RESET gradual programming is shown. For the NCF aVMCO device, however, due to its self-compliant nature [37, 41], gradual programming is possible for both SET and RESET.

Besides the linearization of programming response, a different methodology to seek improvement in RRAM programming instability is that of adaptive program algorithms [183, 229]. Specifically write-verify (W-V) methods are often regarded as an efficient, although slower programming method that involves more complex circuitry [183, 230], to tackle programming variability for inference applications. Nonetheless it has already been reported that at least for binary conductance states (HRS and LRS), W-V methods cannot fully resolve programming instabilities in OxRAM [229].

Here we implement a W-V method were the current is read after each programming step and a target current is set. Once the read programmed current is higher/lower when programming with the SET/RESET steps respectively, programming stops. On the other hand, programming without the Figure 4.12 illustrates the used W-V method.

Figure 4.13 shows a demonstration on how an aVMCO device programmed with the NR with W-V can still deviate from the desired current value due to non-linearity (Figure 4.13a) and PIV (Figure 4.13b).

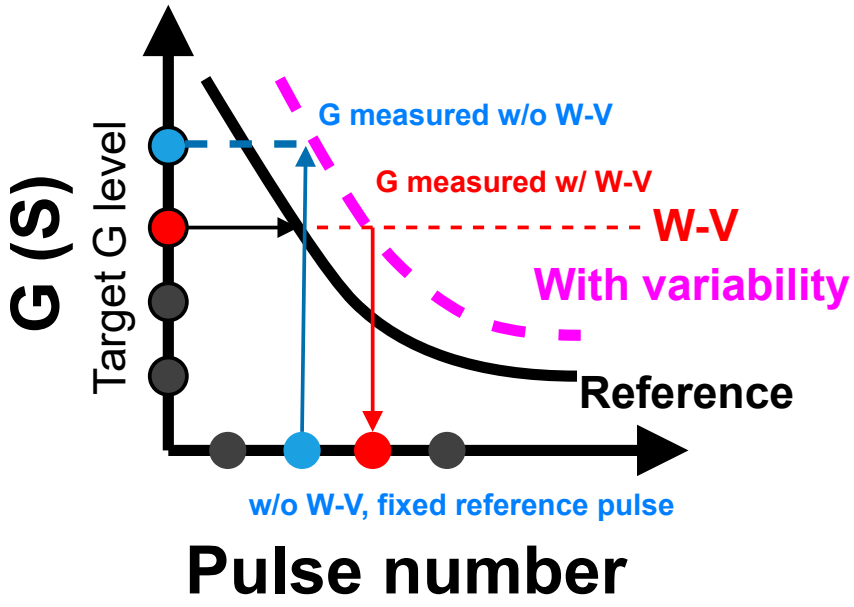


Figure 4.12: Schematic of the implemented write-verify methodology.

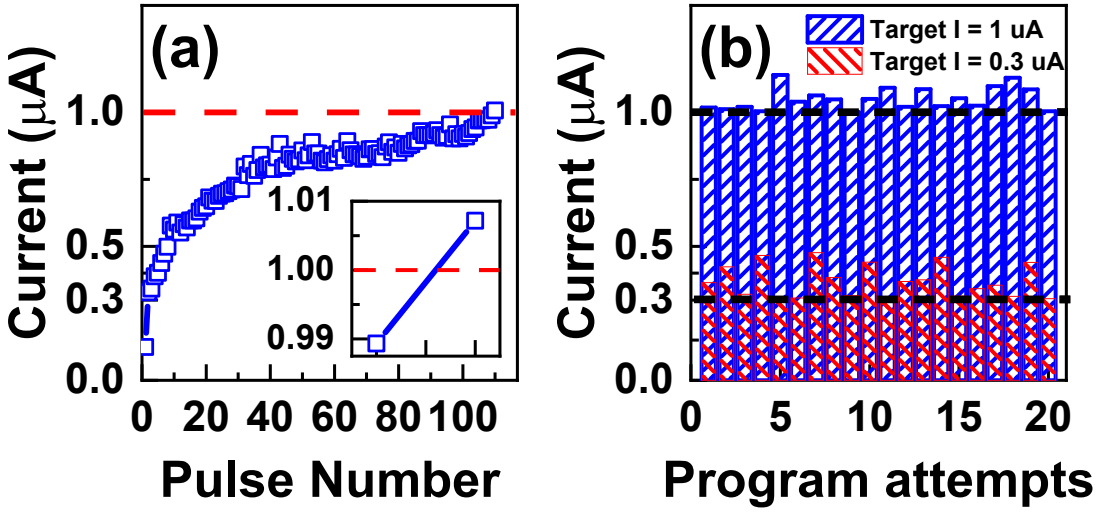


Figure 4.13: (a) demonstration of programming to a high target current in the SET process of aVMCO NR, the inset shows the small discrepancy between the target current and the actual programmed current value. (b) current achieved during the initial 20 programming attempts at both high (1μA) and low (0.3μA) currents. Large PIV still exists at low target current.

In this section, the concepts of PIV, NR, LR and W-V have been introduced. We aim to discuss the impact of these concepts in the inference process of a pattern recognition NN based on synaptic RRAM. Using the same programming methods, CF and NCF devices could also exhibit different PIV due to their inherent differences in switching.

For a systematic evaluation, the PIV will be quantified and simulated in inference for both CF and NCF devices in the following 4 cases: (1) NR without W-V, (2) LR without W-V, (3) NR with W-V and (4) LR with W-V.

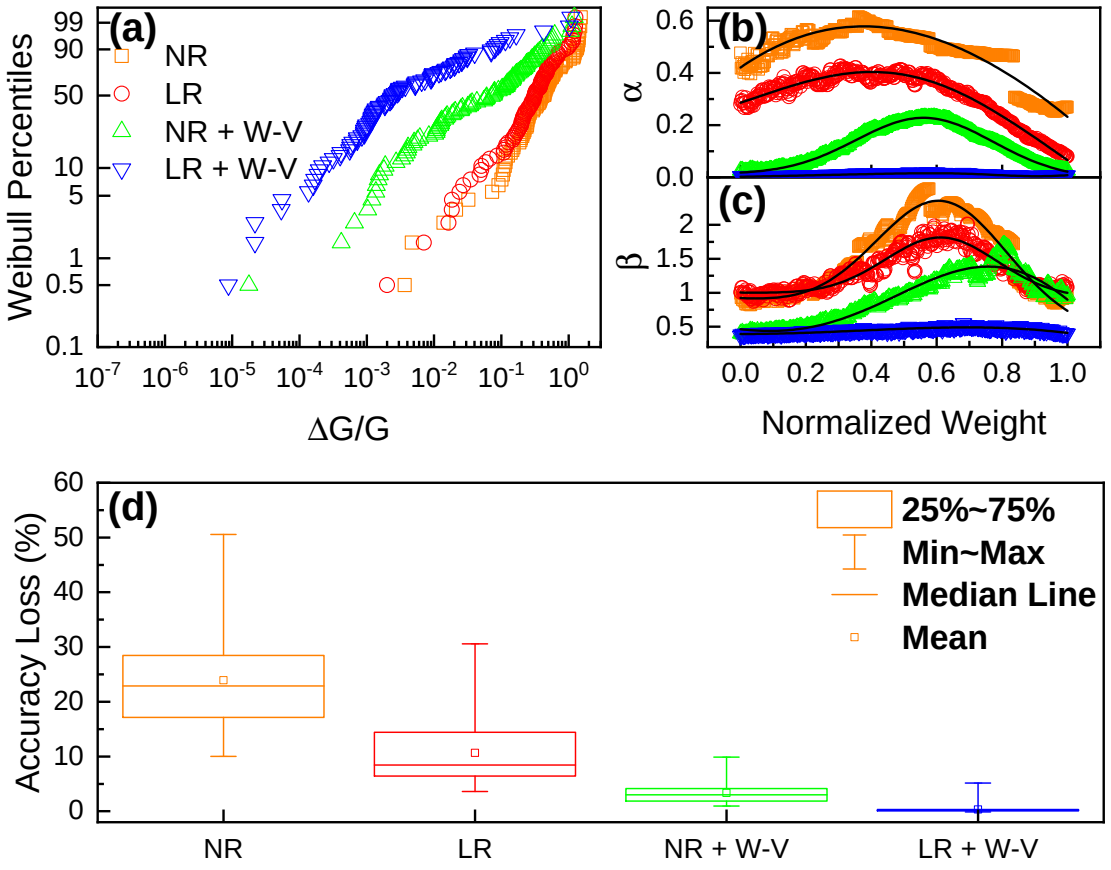


Figure 4.14: (a) Demonstration of the Weibull distributions for the 4 programming cases in a CF device. (b - c) Weibull parameters (b) α and (c) β across the normalized weight range. (d) NN accuracy loss caused by the CF variabilities in the 4 programming cases.

The PIV of the CF device taken from 100 gradual RESET program cycles follows Weibull distribution for all 4 programming cases and is shown in Figure 4.14a. The Weibull PDF and CDF can be described respectively as:

$$y = f(x|\alpha,\beta) = \begin{cases} \frac{\beta}{\alpha}\left(\frac{x}{\alpha}\right)^{\beta-1} e^{-\left(\frac{x}{\alpha}\right)^{\beta}} & ,x \geq 0 \\ 0 & ,x < 0 \end{cases} \quad [231] \tag{4.3}$$

$$p = F(x|\alpha,\beta) = \begin{cases} 1 - e^{-\left(\frac{x}{\alpha}\right)^{\beta}} & ,x \geq 0 \\ 0 & ,x < 0 \end{cases} \quad [231] \tag{4.4}$$

The Weibull scale (α) and shape (β) parameters that describe equations 4.3 & 4.4 can be taken across the whole normalized weight range and are shown in Figures 4.14b&c. Furthermore, the Weibull parameters across the normalized weight range can be described by a 1st degree tilted Gaussian model:

$$y = ae^{-\left(\frac{x-b}{c}\right)^2} + dx + e, \tag{4.5}$$

where a,b and c are the height, position of the peak and width of the gaussian shape respectively, while d and e describe the tilt.

It can be noted that the NR without W-V, (case (1)), shows the worst PIV, which can be explained by the non-existence of any operational scheme to improve performance. After using the staged programming scheme or W-V method, PIV is obviously improved. The best PIV is achieved when using a combination of staged programming and W-V schemes (case (4)). These variability parameters are fitted by equation 4.5 and are superimposed onto a trained NN in a similar manner to the methodology used in sections 4.1 & 4.2.

In order to collect statistically significant data, the NN is trained and disturbed with PIV for 100 times. This process is repeated using the PIV of the four aforementioned programming cases in the CF device. The NN accuracy loss, described by:

$Accuracy_{well-trained} - Accuracy_{disturbed}$ for the NN with the CF device is shown in Figure 4.14d. It is noticeable that the NR without W-V in the CF device results in a significant mean accuracy loss of $\approx 24\%$, as well as a broad distribution of accuracy loss values. Linearizing the programming response (case (2)) results in limited improvements, with a mean accuracy loss of $\approx 11\%$ and using a W-V with the NR (case (3)) further improves the inference accuracy loss to $\approx 3\%$. The best accuracy, in accordance to the PIV parameters, results from combining LR and W-V, which is able to bring down the accuracy loss to values lower than 1%.

It should be noted that analog switching in the Ta_2O_5 device is only possible in the RESET process due to its filamentary nature. Such unipolarity greatly limits the flexibility in synaptic application, not only in training, but also in the types of adaptive algorithms that can be used for inference. On the other hand, NCF devices as aVMCO allows analog switching in both SET and RESET operations and shows exponential NR in both polarities (Figure 4.11b). Therefore, analysis of the PIV of the NCF device under the aforementioned different programming cases in both polarities will be meaningful.

In order to make a reasonable comparison between the CF Ta_2O_5 and NCF aVMCO devices, firstly, only the PIV in the RESET polarity will be analyzed and shown in Figure 4.15.

As with the CF device, the NCF PIV variability also follows Weibull distribution in all 4 programming cases (Figure 4.15a). Also, similarly to the CF device, improvements in PIV are found when using one of the mentioned programming schemes and the best results are obtained when combining both LR and W-V (case (4)).

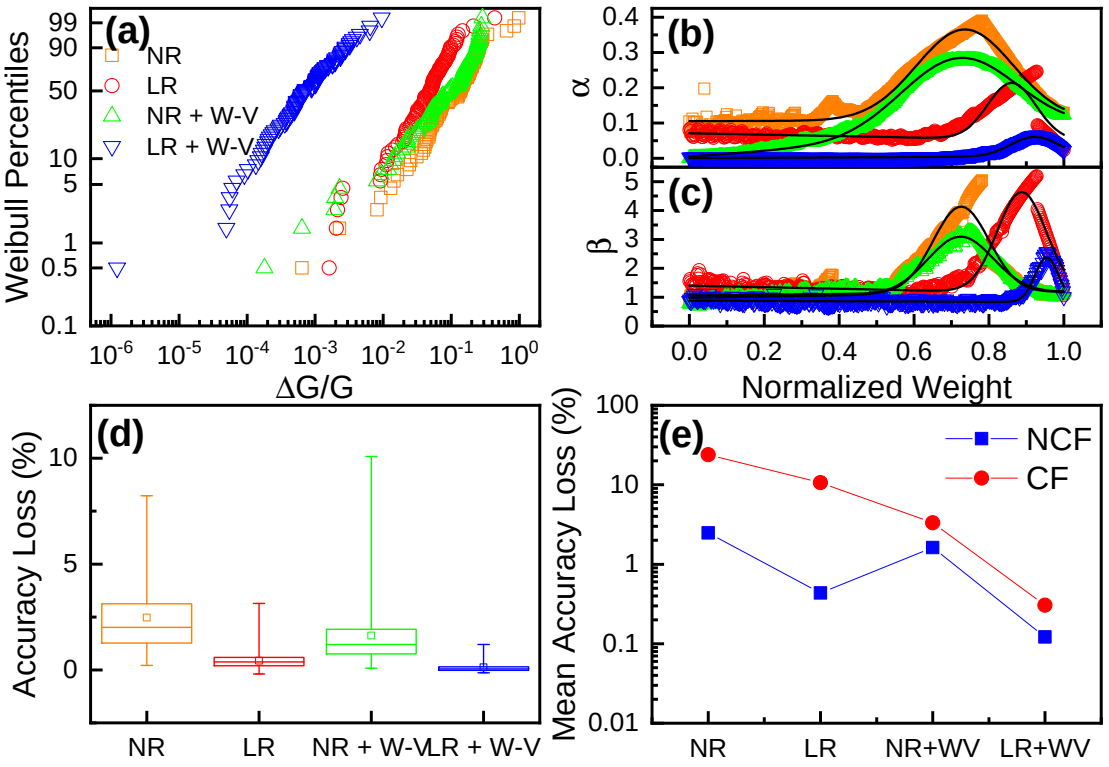


Figure 4.15: (a) Demonstration of the Weibull distributions for the 4 programming cases in a NCF device. (b - c) Weibull parameters (b) α and (c) β across the normalized weight range. (d) NN accuracy loss caused by the NCF variabilities in the 4 programming cases. (e) Mean accuracy loss comparison between the CF and NCF devices.

Those are, however, the extent of the similarities between the two devices. It is noticeable that $\frac{\Delta G}{G}$ is generally one order lower in the aVMCO, compared to the Ta_2O_5 device. One other contrast to the CF case is that PIV with LR without W-V (case (2)) is superior to the case where the NR is done with W-V (case(3)). This can be explained by two factors: the superior improvements that the staged programming can apply to the conductance curve in the NCF device compared to the more limited linearization in the CF device and the large gap in the CF NR conductance curve that the W-V method is unable to mitigate. The prospect of improved PIV without resorting to W-V is not only interesting in terms of less complex circuitry for inference, but also as a solution for training algorithms which will be explored in the following chapter.

The impact of the mentioned PIV also translates into improved accuracy of the NCF device (Figure 4.15d). Even in the worst case scenario (case (1)), the NCF impact in inference results in a mean accuracy loss of $\approx 2.5\%$, using the LR with or without W-V results in negligible mean accuracy loss in inference (less than 1%).

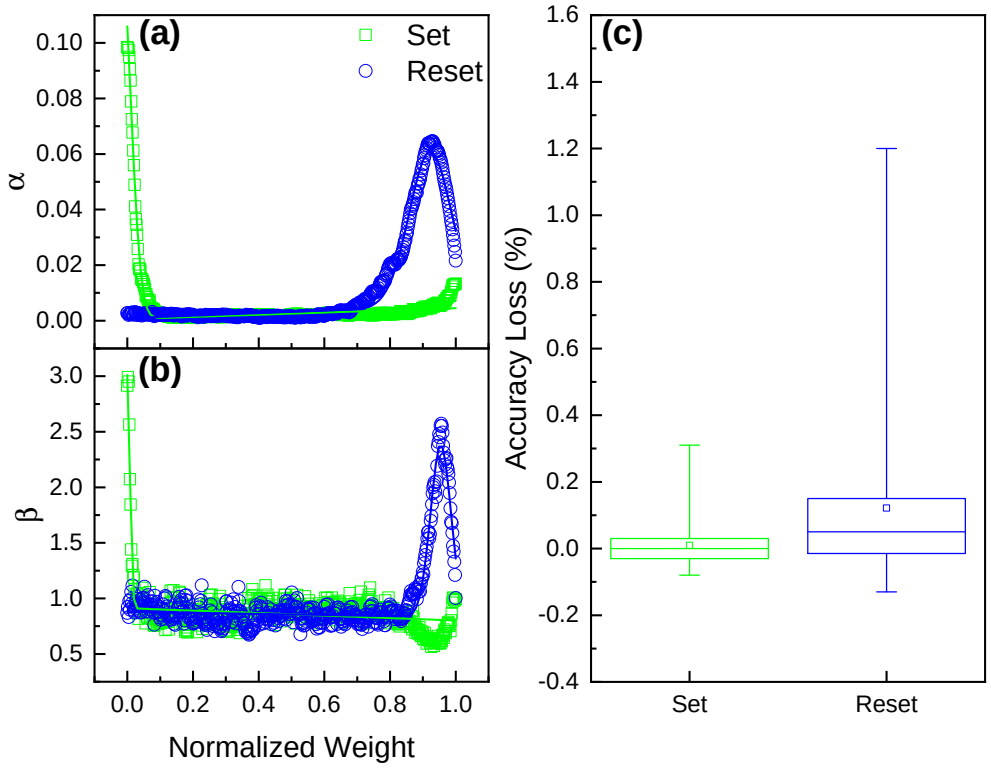


Figure 4.16: (a - b) Weibull parameters of the PIV variability programmed with SET and RESET for the NCF device. (c) Accuracy loss comparison between the PIV programmed with SET and RESET.

In addition, the bipolar analog switching is an advantage of the NCF devices. The PIV comparison of the SET and RESET polarities both using the LR with W-V (case (4)) is shown in Figures 4.16a&b, revealing that programming with the SET polarity has further improved in PIV and consequently in the accuracy loss shown in Figure 4.16c. These improvements can be explained by the even higher degree of linearization that is able to be achieved during the analog SET process of the NCF device opposed to

the RESET process that still suffers from small nonlinearities, particularly at the region close to LRS.

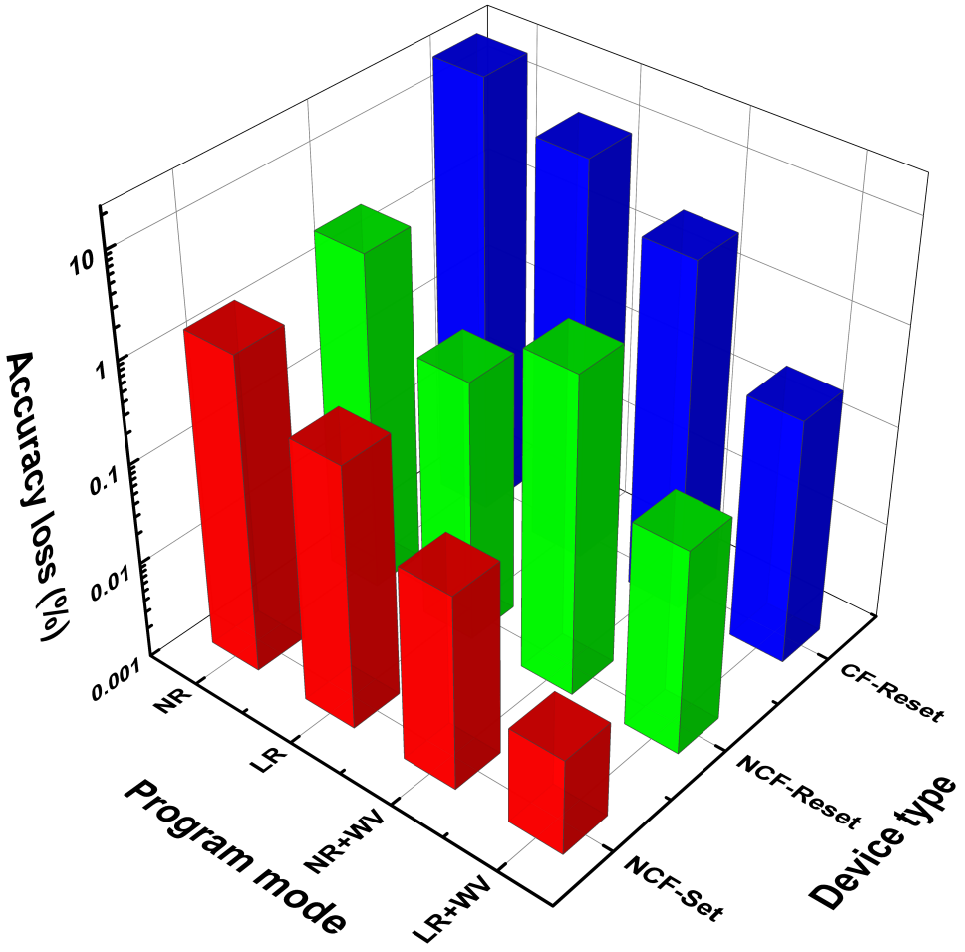


Figure 4.17: Comparison of accuracy loss caused by PIV. Non-filamentary programmed with a combination of weight-updating and write-verify scheme in the set polarity lead to the best accuracy loss.

In summary, the accuracy loss induced by different devices, i.e. CF and NCF, and by different operation modes, i.e. case (1) - (4), in different programming polarities, i.e. RESET and SET are comprehensively compared in Figure 4.17. Here it is possible to conclude that:

- The NCF device shows reduced variability than the CF counterpart, which is reflected in the NN accuracy.

- The combination of staged programming and W-V schemes always lead to the best result, leading to acceptable accuracy loss below 1% for the CF device.
- Using a LR without W-V in the NCF device is sufficient for reducing the NN accuracy loss to values below 1%.
- The SET polarity of the NCF device leads to further improvements when compared to RESET due to improved linearity of this polarity.

4.4 Summary

In this chapter, the use case of two different RRAM devices: aVMCO and Ta_2O_5, which are based on NCF and CF switching mechanisms respectively, in a neuromorphic setting for inference is analysed, particularly focusing on read noises and PIV.

Regarding the read noise, it was possible to distinguish it into two separate categories: RTN and ORN, by manipulating the sampling rate of the applied read signals above or below the known device RTN time constants. Collection of statistical relevant experimental results allowed the development of read noise disturbance models that were posteriorly applied on trained synaptic arrays to simulate its impact on inference accuracy. RTN is revealed to play the major contribution in read noise, while the impact of ORN seems to be negligible. Furthermore, due to its intrinsic switching mechanism, the aVMCO device shows smaller RTN amplitude, tighter RTN distributions and lower RTN occurrence rate compared to the Ta_2O_5 filamentary device. This results in a negligible impact of read noise in the NCF device and therefore better applicability of this type of device for inference.

Statistical measurements of PIV reveals that this variability source can be modelled by Weibull distributions and it was possible to create a model that describes the Weibull parameters across the normalized weight range. NN simulations of different models of variability based on different RRAM programming methods (weight update scheme, W-V schemes and programming polarity) on both devices reveal that: (1) the NCF device shows an overall much lower PIV than the CF device, (2) using a W-V scheme with the NR in the CF device still wields worse results than the NR in the NCF device without W-V, meaning that (3) to guarantee the best results, a combination of LR and W-V is required. Finally, (4) the flexibility in programming polarity of the NCF device allows for further reducing PIV in this device by switching to the more linear SET programming polarity.

Chapter 5

Impact of RRAM non-idealities on training

In the previous chapter, the importance of neuromorphic circuits using RRAM was highlighted for NN inference and the impact of different types of noise and variability on inference accuracy was evaluated.

Another, more challenging but interesting prospect is that of using synaptic RRAM in the training process of BP based NNs. However, the large non-idealities such as the nonlinearity, asymmetry and C2C variability during analog SET and RESET programming in RRAM devices, due to the stochastic nature of the ionic movements involved in resistive switching, remain a major issue that hinders its practical applications [102, 104, 232].

CF based RRAMs suffer the most severe variability and nonlinearity, as a single ionic movement event in the critical constriction region of the filament can lead to large

changes in its conductance state [102, 104, 232]. On the other hand, significant improvement has been observed in NCF or multi-filamentary RRAMs by reducing the effect of singular defects in the switching layer, but considerable non-idealities still exist [37, 89].

The ISPP scheme with W-V can improve the linearity and variability to a limited extent but the drawbacks of implementation, particularly in terms of latency, can become prohibitive for training [74].

Different simulation methods, such as in NeuroSim [94] and CrossSim [233], have been developed to estimate the impact of device non-idealities on the training accuracy. However, detailed evaluation on the impact of individual types of non-idealities within the training process is still difficult due to the limited accessibility and flexibility of these platforms. For example, details are missing on how the synaptic weights in the array are affected during training by individual non-idealities and their combinations, making it difficult to identify the root cause of training accuracy degradation. Also the effects of large conductance stepping (GS) and its position in the normalized conductance range have not yet been evaluated separately from the effects of non-linearity and C2C variability.

In this chapter, to better understand the origin of the impact of different non-ideality mechanisms in analog NN applications, GS and C2C variability of the NCF aVMCO RRAM device are experimentally extracted and characterized in the SET and RESET processes, using the standard NR and the LR obtained through staged weight programming. The evolution of weight distribution in the synaptic array during training are

simulated using the custom developed simulator, so that the detailed origin of the impact of different non-idealities and their combinations in various programming schemes can be evaluated independently, including SET-only, RESET-only, and two different practical combinations of SET and RESET.

5.1 Natural response and Non-idealities

This chapter emphasizes the testing of the NCF aVMCO device which is based in the conductance modulation by the vertical distribution of oxygen vacancies in the switching TiO_2 layer near its interface with the a-Si layer (Figure 5.1a) [76]. The width of this defect-less region becomes larger at higher positive bias, resulting in the analog modulation of multiple conductance levels (Figure 5.1b). Since most of the individual defect movements are averaged out in the switching of multiple defects, it has a much smaller variability in comparison with the CF RRAMs [228].

Here, GS is defined as the gaps in conductance that are present between each programming pulse and C2C variability as the deviations of conductance from a reference conductance curve, between different cycles, as illustrated in Figure 5.1c.

Despite sharing a strong to the most commonly referenced in literature, nonlinearity (section 1.1.4.3) reliability metric, GS differs from nonlinearity in the sense that most cases of nonlinearity are tested using a modelled fit to the data. However, it has been noted that even very small residuals between the fitted model and actual conductance data in the regions with the highest gaps can have a dramatic impact in simulation results. As such, GS adopts the approach of a piecewise linear interpolation that allows

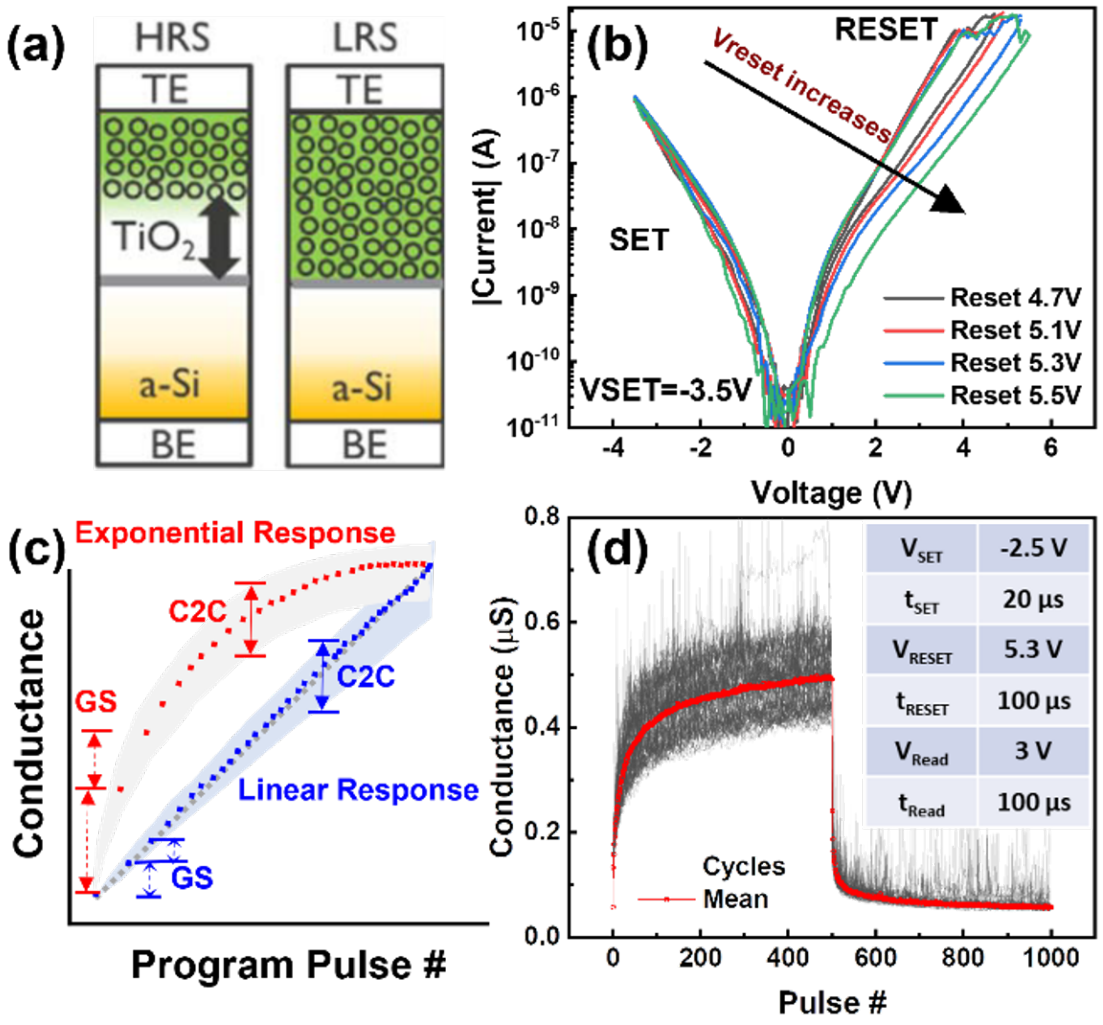


Figure 5.1: (a) Illustration of the non-filamentary switching mechanism in the a-VMCO RRAM. (b) DC I-V characteristics at different RESET voltages. (c) Illustration of the large conductance stepping (GS) caused by initial pulses, and the large C2C variability (in shade), in both linear and natural SET processes. (d) Typical natural programming with 100 cycles (grey) and its mean (red).

for direct quantification of each gap in the reference conductance curve and its direct impact in training, while allowing for the effects of C2C variability to be analysed both together with GS and separately.

Figure 5.1d shows the data of the aVMCO NR after 100 analog SET/RESET cycles, and its mean reference curve. Substantial non-idealities, such as the large GS at the initial SET and RESET pulses and C2C variability can be seen in the NR when programming with a train of identical pulses.

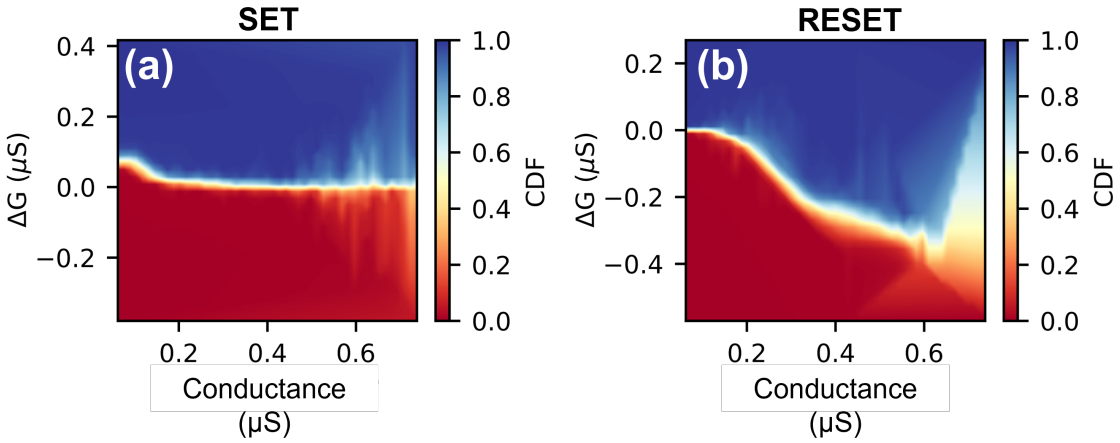


Figure 5.2: Empirical CDF (eCDF) of the GS and C2C induced conductance changes at different conductance levels in aVMCO RRAM when programed by identical pulses with the natural response during (a) SET and (b) RESET.

The eCDF in Figure 5.2 shows that RESET in general suffers from a much higher degree of non-ideality than the SET, with a very high ΔG and much larger dispersion observed at high conductance levels. This is associated with the larger GS at the initial RESET pulses as in Figure 5.1d. On the other hand, the much smaller ΔG and tighter dispersion at low conductance levels for both SET and RESET suggest that non-idealities in this device occur predominantly at high conductance where the defect-less region becomes narrower, and individual defect dispersion have much more significant impact than at low conductance (Figure 5.1a).

5.2 Impact of non-idealities during SET and RESET

It is not clear, however, how GS and C2C in SET/RESET independently affect the NN training accuracy. To clarify this issue, three different programming methods are devised in the simulation.

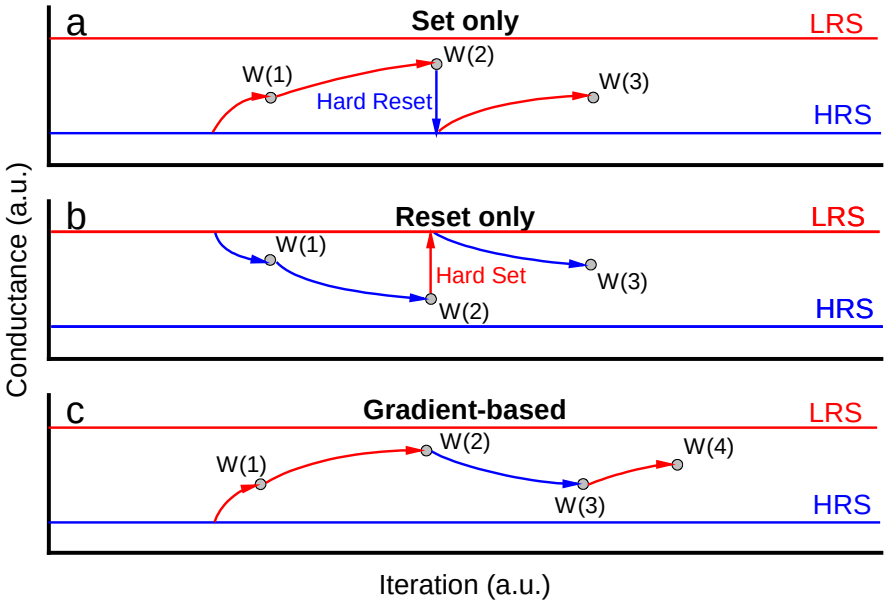


Figure 5.3: Illustration of the different programming methods used in this work. (a) Set-only, (b) Reset-only, (c) Gradient-based. Red arrows represent a SET step and blue arrows represent a RESET step.

As shown in Figure 5.3, the SET-only and RESET-only methods allow the conductance only to increase and decrease, respectively. Although the practicality and applicability of these two use cases for on-chip training can be debatable, in this work, it is used as an 'academic' study. The programming to an arbitrary level is achieved by applying a Hard RESET/SET whenever the conductance needs to be changed to the opposite directions as illustrated by the step from W(2) to W(3) in Figures 5.3a&b.

As a more practical approach in real-world applications, the Gradient-based programming method in Figure 5.3c is also applied, which allows the device to be programmed in any direction as required by the weight gradient during the BP of NN training, without any Hard SET or RESET.

The MNIST handwritten digit database is used for training with a SGDM algorithm (Algorithm 2). Out of the total 60,000 images, 85% are used for training and the remaining 15% are used for validation. After each mini-batch iteration in training, the

weights are disturbed with the mean values of GS alone, then C2C without GS and then GS combined with C2C. A mini-batch size of 128, learning rate = 0.001, momentum = 0.9, validation frequency of 50 iterations and a cost function based on CE loss (equation 1.12). The MNIST input data is shuffled once before each training.

The simulation results using the aVMCO NR are shown in Figure 5.4 trained using a CNN, in which, each convolution has a filter size of 3x3, followed by a batch normalization layer to standardize the inputs, and a piece-wise linear unit (PLU)[189] that approximates a tanh function. The outputs of the activations then pass through a 2x2 Max-Pooling function. The final layer is a fully connected (FC) layer with a softmax activation function. The PLU activation function is used because it is a bounded function that prevents the exploding gradients problem, unlike the ReLU (Table 1.2), and it can achieve a NN training accuracy comparable to the tanh function, while being much easier to implement in hardware [189]. The software benchmark of this simple CNN topology achieves a satisfactory validation accuracy of 96.8% (Figure 5.4b) [189].

As shown in Figure 5.4c, the C2C variability alone without GS has minimal impact on the accuracy degradation of the CNN, achieving about 90% accuracy in all three program methods. In sharp contrast, GS alone causes significant accuracy drops to about 65% for all programming methods. More surprisingly, when applying both GS and C2C, the accuracy further decreases to 55% for the SET-only but increases to 94.8% for the RESET-only, and only drops slightly to 88.4% for the Gradient-based programming method. This indicates that the C2C overcomes the GS's adverse impact for the RESET but makes it even worse for SET. This seems to contradict the results in Figures 5.1d&5.2, where RESET exhibits significantly worse GS and ΔG dispersion at high

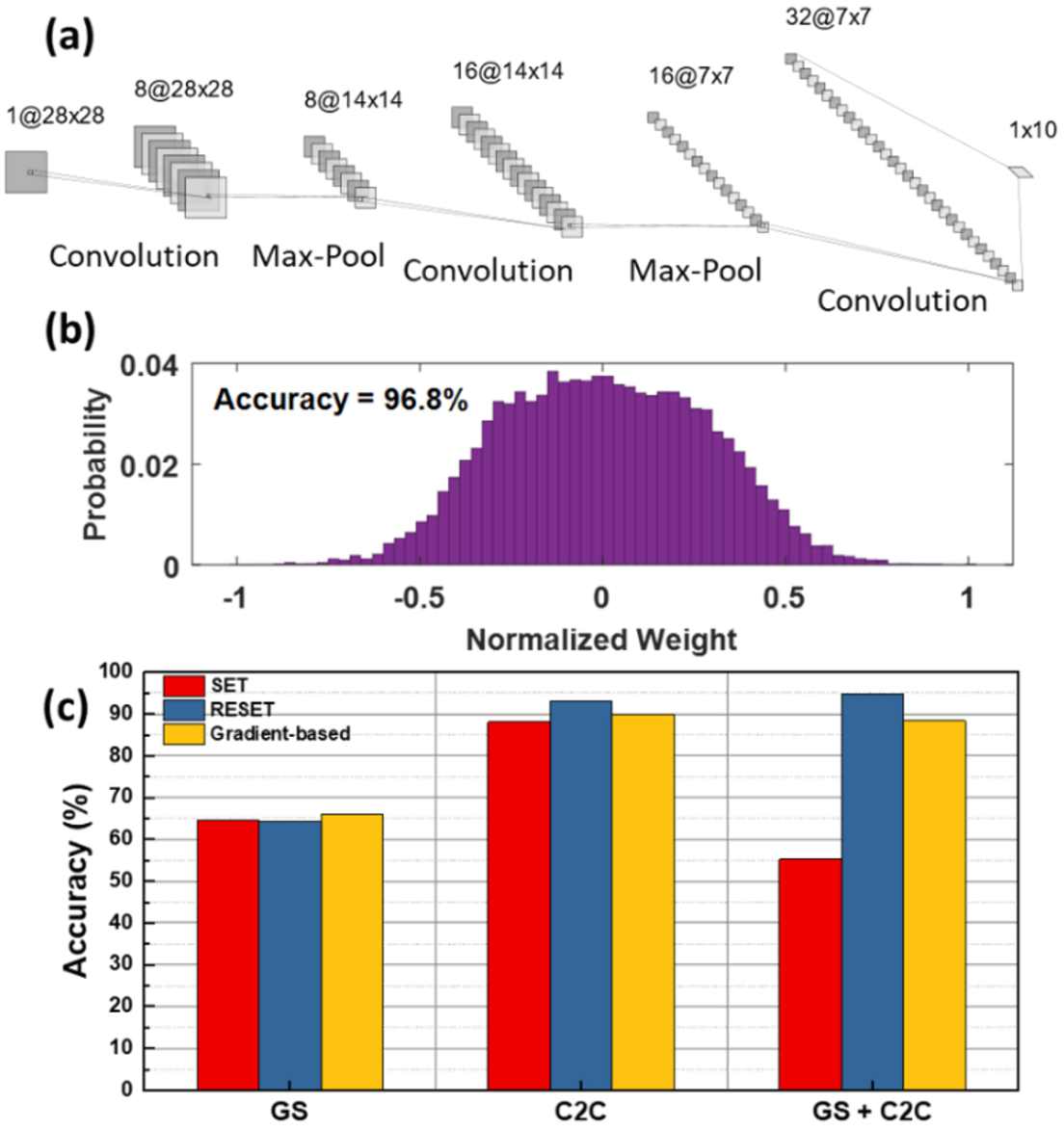


Figure 5.4: (a) Topology of the CNN used in simulation. (b) Normalized weight histogram of the trained final layer of the CNN. (c) Validation accuracy on the MNIST database achieved using the natural response with different programming methods and non-idealities.

conductance levels than SET. To explain this, we will compare the weight distributions after training with SET-only and RESET-only to the software benchmark shown in Figure 5.4b, and how it is associated with the accuracy degradation mechanisms.

Figure 5.5a-c shows the trained normalized weight histograms using the SET-only, RESET-only and Gradient-based programming with GS alone considered, by using the custom-built simulation framework. When GS alone is considered, the final trained

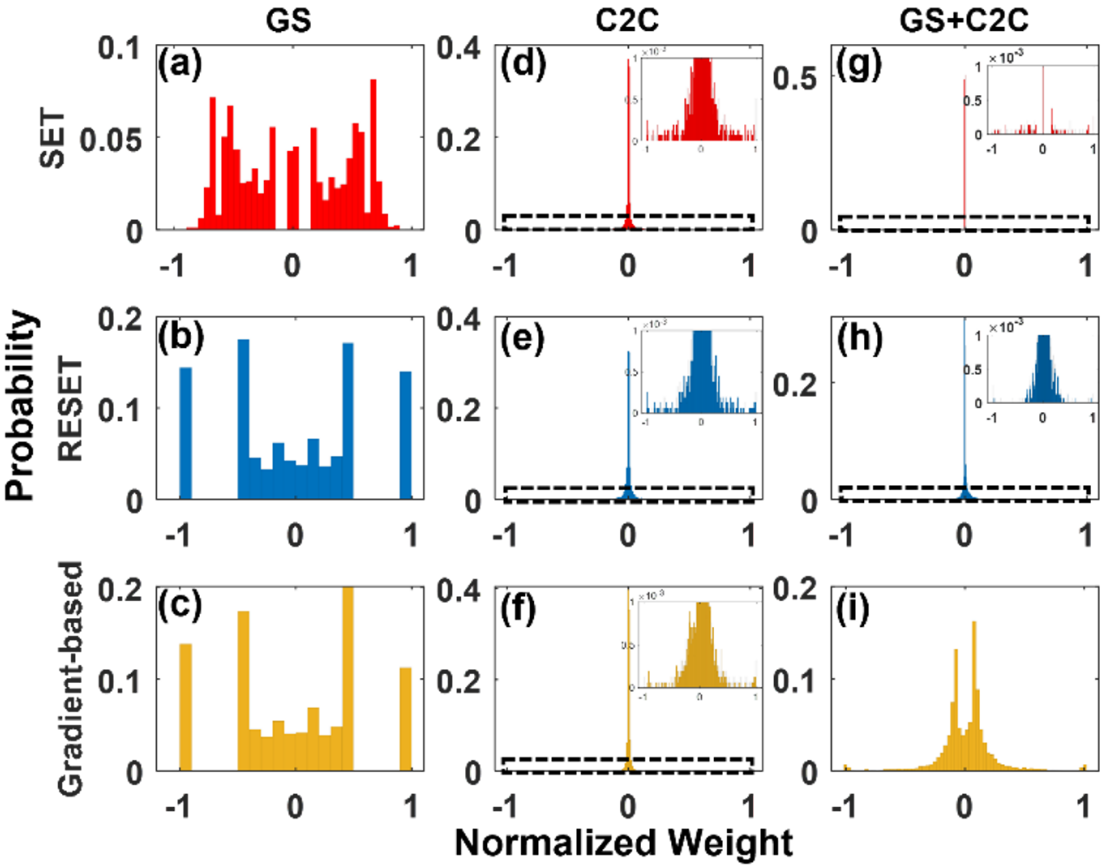


Figure 5.5: Trained normalized weight histograms of the final layer of the CNN with the GS (a - c), C2C (d - f) and GS combined with C2C (g - i) using the different programming methods (SET, RESET and Gradient-based). Insets in (d - h) show a zoomed in version on the y-axis of the histograms.

weights remain very close to their random initialization while showing a clear gap due to GS in all cases. For SET, the gap is situated between normalized weights 0 and ± 0.2, which can be attributed to the GS in the initial SET pulses at low conductance levels (Figures 5.1d&5.2a), while for RESET, a much larger gap is observable above normalized weights ± 0.5, due to the high GS at the initial RESET pulses at high conductance levels. The gradient-based programming method achieves a weight distribution closer to the RESET cases.

Despite these differences, Figure 5.4c shows that these different set of weights lead to similarly bad accuracy with GS alone, suggesting that even though the gap is much

larger during RESET at higher weight range, as the larger gap during RESET causes some weights to become stuck at ± 1, the availability of low value weights is more critical in training than the high weights. This is confirmed by comparing with the software ideal weight distribution in the absence of non-idealities (Figure 5.4b), where the weights show a Gaussian distribution centered around normalized weight = 0, evolved from the initial random distribution.

Furthermore, the impact of C2C variability alone is also studied. Figure 5.5d-f shows that the weight distribution is completely different from the case of GS-only in Figure 5.5a-c. A similar Gaussian distribution can be obtained for both SET-only and RESET-only as shown in the insets, albeit both appear tighter than the ideal case due to the larger number of 0 weights. SET-only has a slightly worse accuracy due to the increased variability at the low value weights than RESET. Therefore c2C alone has a limited impact on the training accuracy [82].

On the other hand, as shown in Figure 5.5g-i, when adding C2C variability on top of GS, the weight histograms show different trends for SET and RESET again. For SET, since the gradients decrease throughout trainined leading to the convergence of weights towards 0, the gap of missing weights near 0 caused by GS makes almost all weights stuck at 0. The lower C2C variability at low conductance levels shown in Figure 5.2a is not sufficient to help the weights surpass the gap that GS imposes, leading to worse degradation than GS-only. For RESET, however, the gap caused by GS is at higher weights, while most of the more important smaller weights are achievable. The larger C2C variability at high conductance levels also randomly increase the gradients, so that the weights are unstuck, leading to a Gaussian distribution of weights centered around

0, which is still effective albeit being narrower than the ideal case. For the gradient-based programming, however, the effects of both SET and RESET randomly come into play as the weights need to increase or decrease, leading to a broad distribution caused by the RESET process, combined with a gap at low value weights caused by the SET process.

This analysis revealed the different roles played by GS and C2C in SET and RESET in CNN training. The key difference lies in the availability of lower value weights in the region close to 0, where most of ideally trained weights in the synaptic array are located. The GS induces a gap of missing weights in this region during SET and can predominantly degrade the training accuracy. The larger gap during RESET, however, is not located in this region and the larger C2C at the gap can help the weights become unstuck, improving the gradients and hence the training accuracy. Therefore, less device non-ideality does not necessarily lead to a better NN accuracy, which is predominately controlled by weight availability in the key region. It can also explain the performance of Gradient-based program method as being a combination of both the SET and RESET processes, it has a mixture of weight degradation mechanisms in SET and RESET, leading to an intermediate NN training accuracy close to RESET-only, as shown in Figure 5.4c.

5.3 Linear response and Non-idealities

To improve the training accuracy, various linear program schemes, such as the ISPP with incremental pulse amplitude and/or pulse duration and W-V have been proposed

to replace the NR with identical pulses [74, 234] and to improve the program linearity and reduce the C2C variability. The aforementioned simulation method developed for section 5.2 will be used in this section to analyse the effectiveness of the LR, and to identify possible ways to further improve the NN training accuracy.

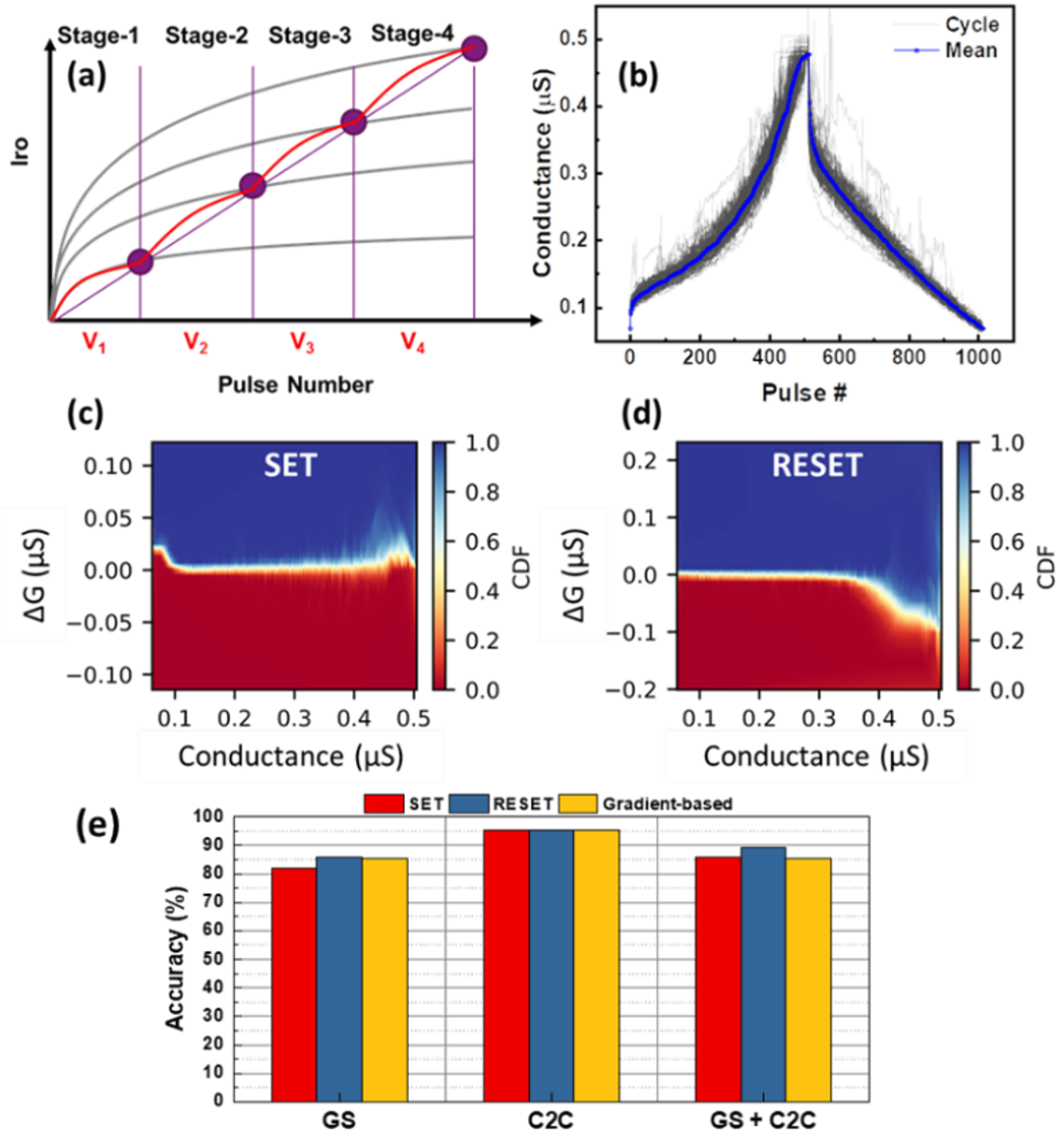


Figure 5.6: (a) Illustration of the staged weight update scheme and (b) the obtained linear like response, with 100 cycles shown in grey and the mean in blue. (c) eCDF of ΔG and dispersion in the linear response for SET and (d) for RESET. (e) Validation accuracy on the MNIST database achieved by the linear response with different programming methods and non-idealities.

A simple staged weight update (see section 2.5.2) is used in this work to reduce the

nonlinearity and therefore the GS in SET and RESET, during which the pulse amplitude increments by stages, and is kept constant within each stage to mimic an overall LR, as illustrated in Figure 5.6a. The achieved LR is shown in Figure 5.6b. As confirmed in the eCDF in Figures 5.6c&d, ΔG and its dispersion, representing the GS induced by nonlinearity and the C2C variability, respectively, are both significantly reduced when compared to those in the NR shown in Figures 5.1d&5.2.

The final weight distributions of the LR with the SET-only, RESET-only and Gradient-based methods are compared in Figure 5.7, and all have improved against those in the NR, with smaller weight gaps and smoother Gaussian distributions centered at 0. The lower magnitude of both GS and C2C in the LR also lead to a less significant weight convergence towards 0.

This has led to a significant improvement of GS-only training accuracy from the 65% in the NR to 85% in the LR (Figure 5.6e), proving the effectiveness of the LR in reducing the GS and hence the missing weight gap in training. On the other hand, C2C-only introduces a very small accuracy loss of around 1% from the ideal case, which is negligible when compared to the more than 11% accuracy loss for GS-only. The addition of C2C variability on top of the GS results in an intermediate accuracy between GS-only and C2C-only for all three program methods, and most noticeably, the SET-only method no longer causes a very large degradation, in contrast with that in the NR as in Figure 5.4c. This result suggests that the GS during the initial SET pulses becomes smaller in the LR and has led to a significant improvement to within 5% accuracy achieved by RESET-only. However, as shown in Figure 5.6b there is still a considerable GS induced by the initial SET and RESET pulses in the LR even when using a very small program

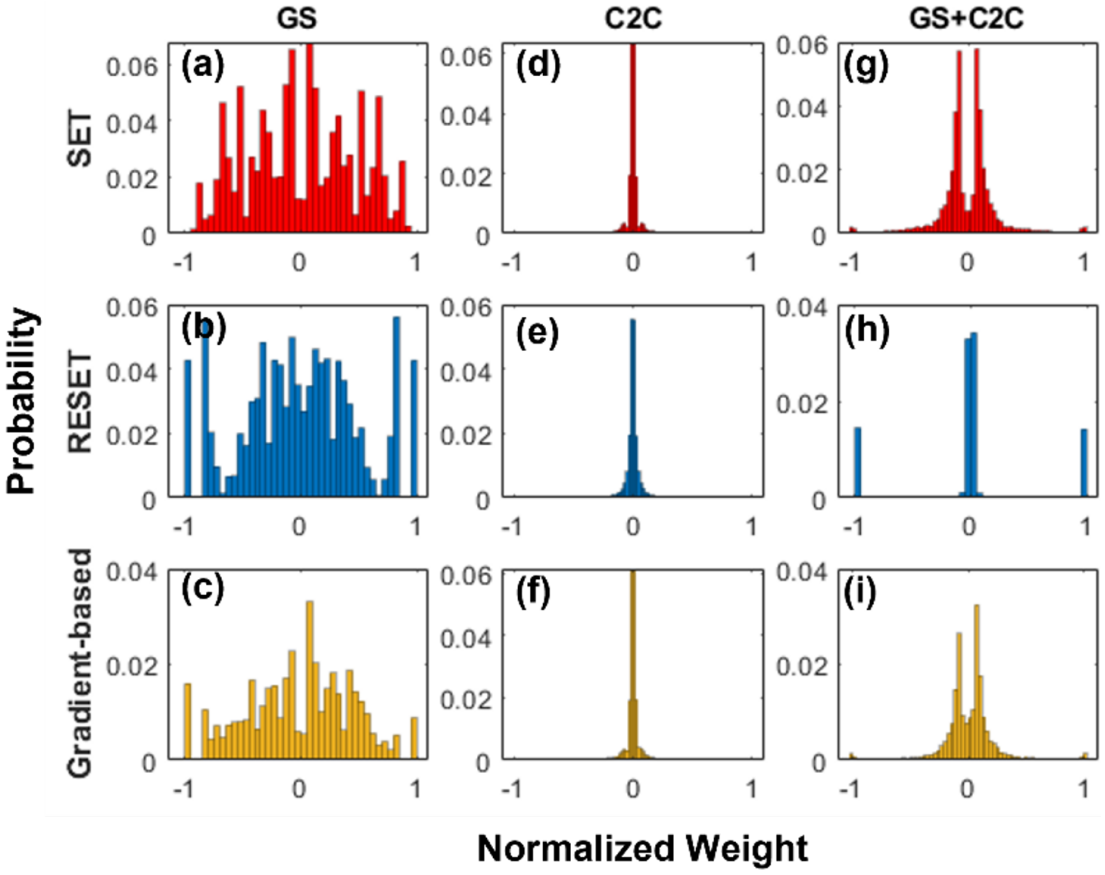


Figure 5.7: Trained normalized weight histograms of the final layer of the CNN using the linear response with the GS (a - c), C2C (d - f) and GS combined with C2C (g - i) using the different programming methods (SET, RESET and Gradient-based).

pulse amplitude, GS still plays a dominant role in training accuracy loss in comparison with C2C as shown in Figure 5.6e.

5.4 Selective programming

Based on the analysis made in section 5.3, a different programming method is developed to further reduce the impact of GS, especially in the low weight range close to 0. Since the different impacts on NN accuracy in SET and RESET are directly linked to where the large GS is situated in the weight range: large GS is only observed close to weight 0 for SET, and it is inly close to weight 1 for RESET, a **selective** programming

is introduced to mitigate the adverse effects of GS by combining the other half of the SET and RESET weight range that has no large GS.

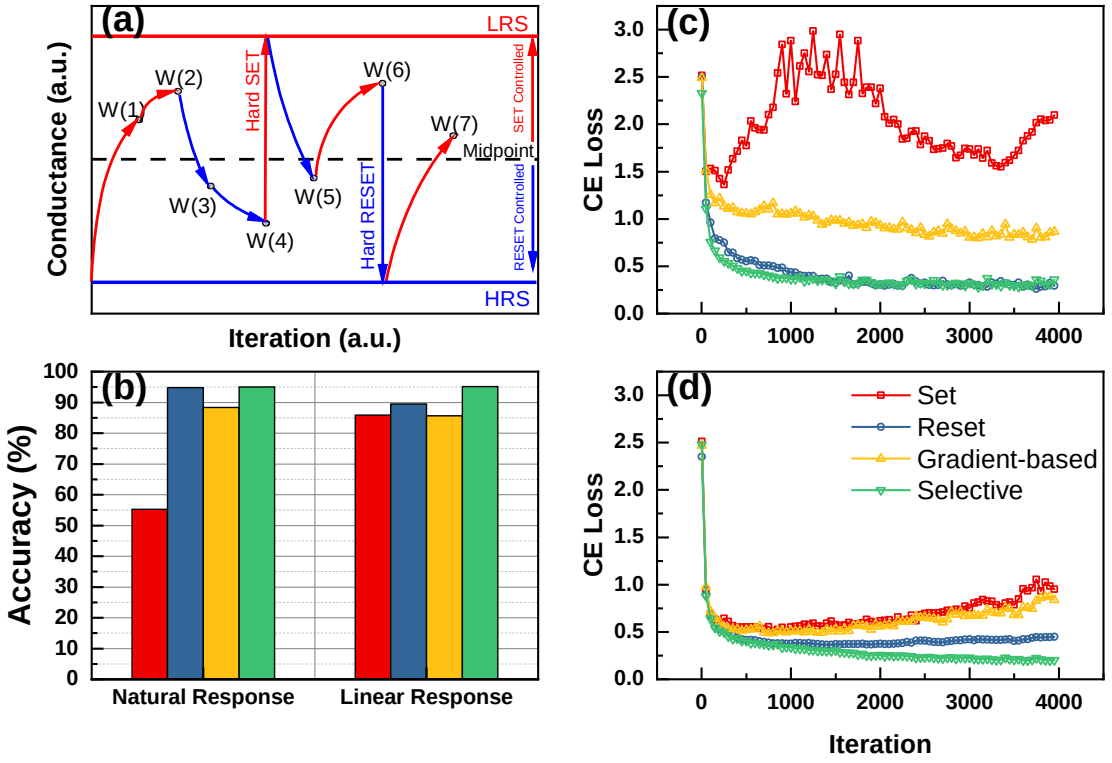


Figure 5.8: (a) Illustration of the Selective programming method. (b) Summary of the achieved accuracies of the CNN with different programming methods and weight update schemes. (c) CE loss throughout training while using the Natural and (d) Linear Programming Response.

As illustrated in Figure 5.8a, the selective programming is based on the idea of always utilising the 2nd half of the weight range in SET/RESET to avoid the large GS caused by the initial pulses in their first halves. This can be achieved by allowing the device to be programmed using continuous SET/RESET pulses similar to the gradient-based method at W(1)-W(2) and W(3)-W(4) where the weight is changing in the same direction as in the 2nd half of SET and RESET, respectively. When the weight needs to change towards the opposite target weight direction in their 2nd half, for example, at W(4)-W(5) and W(6)-W(7), a Hard SET/RESET is applied first and then followed by a gradual RESET/SET program, respectively, to ensure the 2nd half of program response is always

used without large GS. Furthermore, this programming scheme is independent of the staged weight update scheme and can be used in conjunction with both the NR and LR.

The detailed training results for all of the previously described program methods and responses (from sections 5.2 - 5.4) are shown in Figure 5.8b. Based on the evolution of the CE loss throughout training, as shown in Figures 5.8c&d, a significant improvement can be observed when switching from the NR to the LR, particularly on the SET-only and Gradient-based programming methods. Moreover, the introduction of the selective programming allows for the CE loss to be reduced to the minimum level for both the NR and LR. The best training accuracy of 95% can be achieved using either the NR or LR showing that even for highly non-linear devices with large GS at the initial pulses, this is an effective programming strategy for mitigating device non-idealities, at the cost of an increase in power and latency in the iterations with the hard SET/RESET.

5.5 Impact of NN topology

A CNN model has been used so far to illustrate the impact of RRAM non-idealities. Although CNNs are commonly used for pattern recognition, its weight mapping on RRAM devices is usually more challenging than in FC MLPs [105, 202].

The impact of different NN topologies is evaluated in this section, including two different MLP topologies for MNIST classification. One MLP consists of one hidden layer (1L-MLP) with a 784x30x10 topology, and the other is a three hidden layer MLP (3L-MLP) with a 784x400x200x100x10 topology. The activation functions of the MLPs are the PLU, and the final layer has a softmax activation function, the same as in the

CNN. The validation accuracy for the software benchmarks are 91.6% and 95.5% for the 1L-MLP and the 3L-MLP respectively.

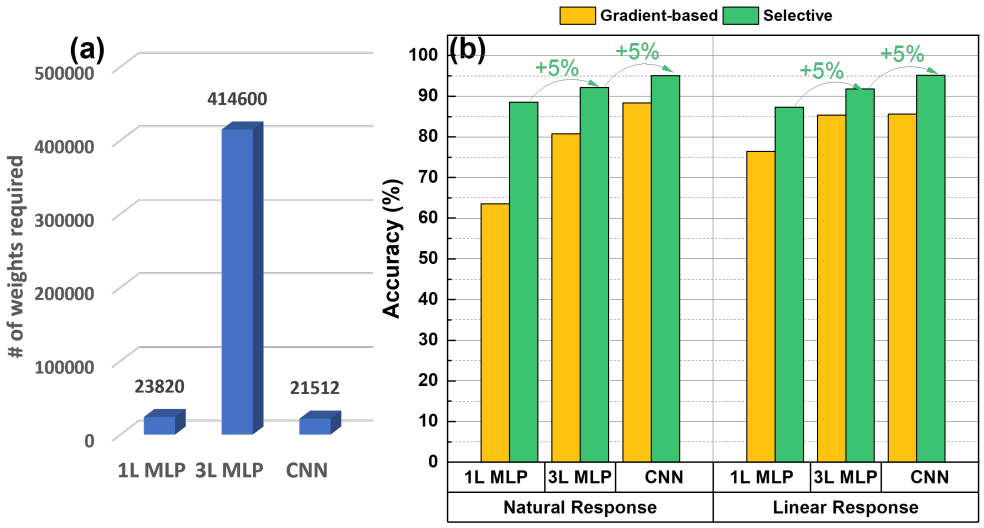


Figure 5.9: Comparison of different NN topologies. (a) Number of weights required for each topology. (b) Accuracy achieved with different topologies on the Natural and Linear Response.

The 1L-MLP has the comparable number of required weights to the CNN, as shown in Figure 5.9a. The 3L-MLP has the comparable software benchmark accuracy to the CNN, but the number of synaptic weights need to increase dramatically, highlighting the benefits of the CNN. Figure 5.9b shows the achieved accuracy on the three different topologies using the NR and LR and the Gradient-based and Selective programming methods. For the Selective programming, a 5% increment in accuracy is observable between the different topologies (1L-MLP $<$ 3L-MLP $<$ CNN) for both the NR and LR, while the Gradient-based programming produces a lower accuracy in all cases in comparison with the Selective method.

The maximum accuracy achieved by the Selective method is at 95%, which is very close to the software benchmark ideal value of 96.8% that does not take any non-idealities into account. Most remarkably, the Selective method can achieve an accuracy

of 90% even with the simplest 1L-MLP and the simplest NR achieved with identical pulses, which has the worst non-idealities, demonstrating its potential applications in practical analog NN. Therefore, the Selective programming scheme can significantly improve the accuracy of analog NN by avoiding the large GS in the key weight range and mitigating this critical issue.

5.6 Impact of Learning Rate

As was shown in sections 5.2 & 5.3, GS is shown to be the type of non-ideality that assumes the most impact during training. Since GS is related to the size of the step between pulses in the device conductance curve, a direct parallel can be drawn to a NN hyperparameter: the learning rate.

The learning rate can be seen as the scalar value that controls the magnitude and proportionality of the gradients in a gradient descent algorithm (see section 1.2.1.4). In other words, the learning rate controls the size of the step (in the software perspective) before applying any device non-idealities, and the GS controls the size of the step (in the hardware perspective) after applying the device non-idealities.

It is well known that even without involving non-ideal devices, the choice of learning rate can have a big impact on training of different datasets. Choosing too high of a learning rate value can lead to training instability, while choosing too low of a value can lead to slow training that can become stuck in sub-optimal local minima or saddle points. Figure 5.10 shows the accuracy achieved in the software benchmarks of the three tested NN topologies. A slight increase in accuracy can be observed each learning

rate decade up to a optimal learning rate value of 10^{-1}, at which point the accuracy decreases only slightly for the MLPs, but renders the CNN untrainable. Pushing the learning rate further to a value of 10 turns all topologies practically untrainable.

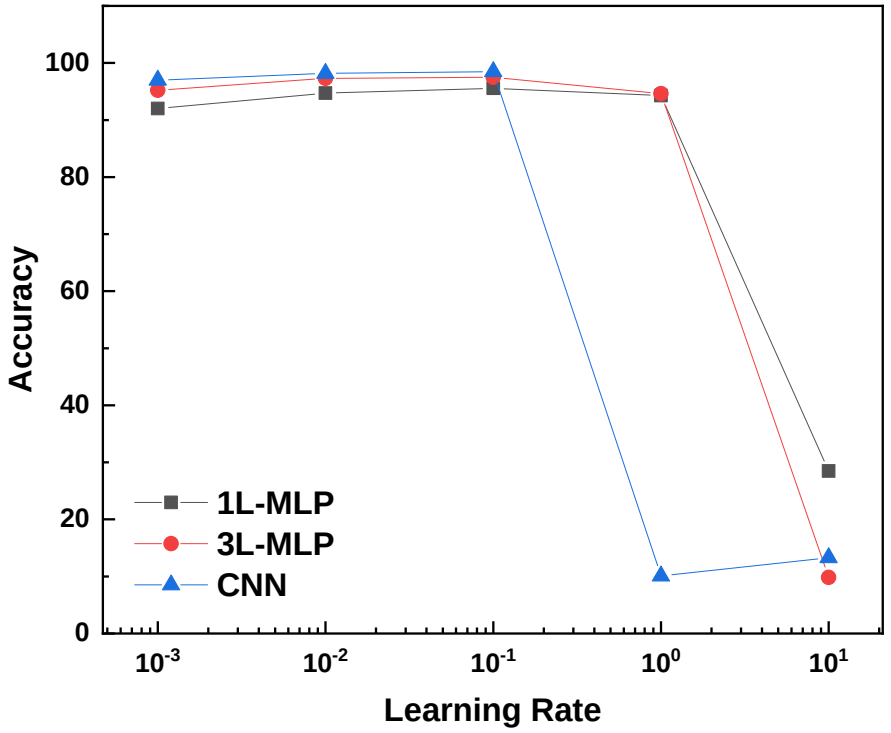


Figure 5.10: Impact of learning rate on the software benchmark accuracies of the three tested NN topologies.

Adding the GS on top of the learning rate issue, will have the added effect of pushing the step beyond the weight values predicted by the gradients if the calculated weight update lands on the areas with high GS.

It is therefore important to analyse the impact that different learning rate values can have on a NN affected by RRAM non-idealities. Although the maximum achievable accuracy by the NN could give some insights into this effect (Figure A.1), looking into the evolution of validation accuracy through 10 epochs of training shows that the accuracy can display instabilities as training evolves, when adding the non-idealities

(Figure A.2), which is detrimental in an online training setting. Therefore, the final accuracy taken at the end of training is a more reliable metric for this effect.

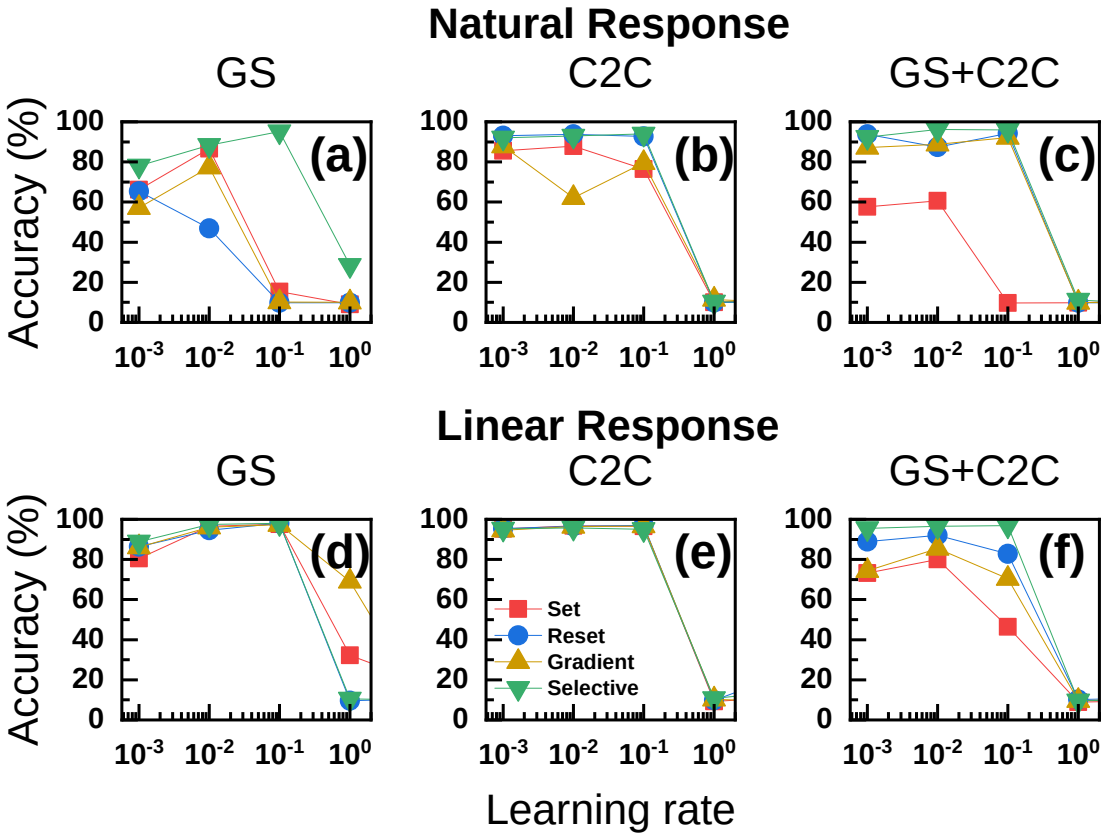


Figure 5.11: Impact of learning rate on the **final** accuracy achieved by the CNN model from Fig. 5.4a after 10 epochs of training, using the Natural Response (a - c), evaluated with the (a) GS, (b) C2C and (c) GS combined with C2C non-idealities. (d - f) shows the same methodology using the Linear Response.

Figure 5.11 shows the impact of learning rate on the CNN affected by the different non-idealities trained using the four programming methods (SET-only, RESET-only, Gradient-based and Selective) and weight staging responses (NR and LR). As expected when taking into account the software benchmark results (Figure 5.10), a learning rate of 1 is too unstable for training the CNN in every combination of non-idealities. However, for lower values, it is possible to notice some differences in regards to the software benchmark.

In the case of disturbing the NN only with GS, the high GS of the NR clashes with high learning rate values and instabilities in training start to appear at 10^{-2} particularly caused by RESET, evidenced in Figure A.2a which is likely caused by the higher GS in RESET opposed to SET (Figure 5.1d). The lower GS in the LR, however, allows for the learning rate to be increased to its ideal software benchmark value of 10^{-1} without instabilities (Figure A.2b).

When disturbing the NN with C2C, the impact on accuracy is much more restrained than with GS-only and the effects of increasing the learning rate is very limited in both NR and LR, shown in Figures 5.11b&e, respectively. This limited impact is most likely happening due to the fact that the very limited improvements in accuracy observed in the software benchmark (Figure 5.10) are comparable to the very limited negative contributions of C2C for the NN accuracy. Nevertheless, observing the training evolution (Figure A.2c) it is possible to notice that at least for the NR, even though there is no clear trend of accuracy degradation with time, there is more discrepancy at each validation step when the learning rate is increased.

The impact of learning rate in a NN disturbed with both GS and C2C (Figure 5.11c&f) agree with the previous conclusions from sections 5.2 - 5.4. Programming using SET-only when combining GS and C2C is the most impacted by the choice of learning rate, while RESET-only is able to train well under a low learning rate. Despite the apparently small difference in maximum accuracy that the SET and RESET programmed LR exhibited in Figures 5.6e&b, the degradation of accuracy occurring using SET-only is much more evidenced than while using RESET-only (Figure A.2f). This effect is then

aggravated as the learning rate increases for SET-only, and to a lesser extent RESET-only. Since the Gradient-based programming is a stochastic combination of SET & RESET, that is also reflected in an intermediate combination of the two results.

However, one noticeable aspect common to all the different learning rate and weight staging response cases is that the selective programming is more resilient to instabilities during training and in general does not suffer from accuracy degradation evolving with time.

5.7 Dynamic weight range rescaling

One particular issue in neuromorphic systems with synaptic RRAM devices is the discrepancy between a dynamically scalable weight range in the software training algorithms and the finite conductance range of RRAM devices. Most implementations are also bounded by the range of the digital interface circuitry at the edge of the crossbars [2, 130], however, implementations that rely on fully analogue signal propagation are able to avoid ADC/DAC peripheral circuitry overhead [235] and therefore have their weight range only limited by the synaptic devices.

Most simulation frameworks, such as NeuroSim [94, 105, 202] or CrossSim [233] rely on the standard digital approach that needs the weights in each layer to be scaled to a specific fixed range, each layer's fixed range becomes one additional NN hyperparameter that requires careful calibration to achieve the optimal results. As NNs become more complex with increased hidden layers, choosing this hyperparameter can become

an issue, especially in an online training setting where there is no prior knowledge of the incoming data feed.

In this section, we simulate the possibility of dynamically changing the weight range in accordance to the NN requirements at training time. In the case of mixed-precision chip in loop systems [236], the scaling factors responsible for controlling the dynamic weight range in each NN layer could be calculated by the external high-precision compute unit.

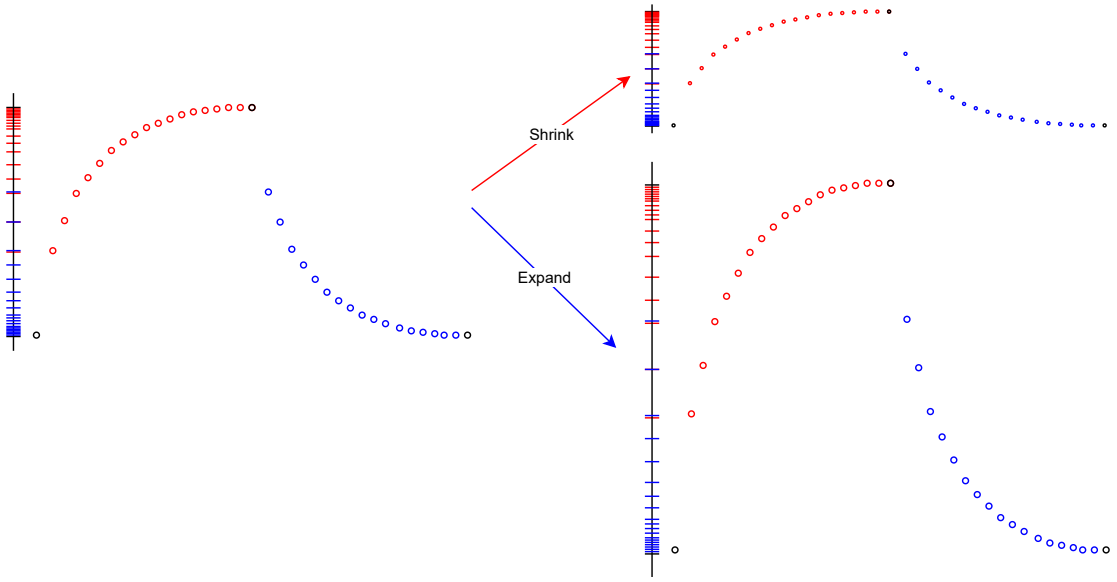


Figure 5.12: Illustration of the dynamic weight range rescaling effect on the weight availability of a typical Natural Response curve. Red ticks show the weights available through SET and blue ticks show the weights available through RESET.

Since the weight range is rescaled at each iteration, but the conductance range of the RRAM devices cannot be changed, the dynamically changing scaling factor, in practice, changes the weight availability that the device is able to provide, shrinking or expanding the conductance curve depending if the weight range is decreased or increased respectively. Figure 5.12 shows a schematic of how the dynamically changing weight range

affects a generic RRAM conductance curve (similar to the aVMCO NR) and consequently the GS.

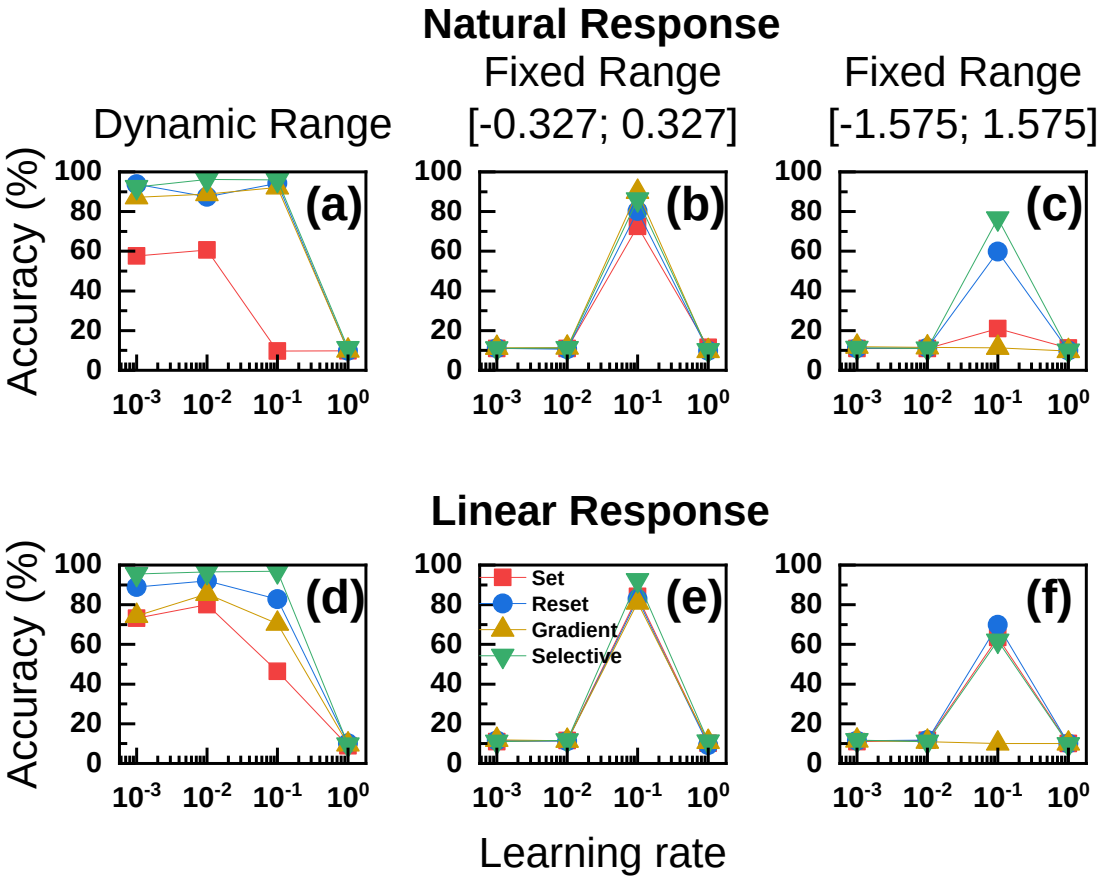


Figure 5.13: Impact of Dynamic Weight Range rescaling compared to two fixed ranges: [-0.327; 0.327] & [-1.575, 1.575] on the final validation accuracy of a CNN. Different programming methods (SET, RESET, Gradient and Selective) illustrated on both the aVMCO Natural Response (a - c) and Linear Response (d - f).

A similar methodology from section 5.6 is applied here to test the impact of using dynamic versus a fixed range in training. Figure 5.13 shows the final accuracy of our CNN model (Figure 5.4a) when GS and C2C of the aVMCO device is applied during training, comparing the performance between training with the dynamic weight range rescaling and two different fixed ranges: [-0.327; 0.327] and [-1.575; 1.575] [233].

The main noticeable difference between the dynamic range method and the fixed range counterparts is that training is only possible when using a learning rate of 10^{-1},

while with dynamic range, a broad range of learning rate values allows for training with mixed results. Nevertheless, when using the selective programming combined with the LR (best case scenario), the best results achieved with the proper choice of learning rate are: 96.92%, 92.36% and 61.92% for the dynamic range, [-0.327; 0.327] and [-1.575; 1.575] fixed ranges respectively, with correspondent CE losses of: 0.105, 0.578 and 1.682 (Figure B.1). Compiling of these results shows that the dynamic range method not only is able to provide a better accuracy result in the best case scenarios, but the significantly lower CE loss in the dynamic can also be taken as an indication of training stability between iterations that can be observed in Figure B.2.

Additionally, comparing the two fixed range cases shows that accuracies between 72% to 92% are achieved when selecting the [-0.327; 0.327] range, while using the [-1.575; 1.575] range, all values are below 77%, being that in this range, training with the Gradient-based programming method is impossible for both NR and LR. The clear superiority of one fixed range over the other is an indication of the issue of calibration of this additional NN hyperparameter for training using synaptic devices.

One issue that could rise from the use of dynamic weight range rescaling in detriment of a fixed range is that of increased device endurance requirements. In the hardware implementation, a weight range rescale event could mean that a device that would maintain its weight value from one iteration to the other could still be required to be reprogrammed due to the discrepancy in weight between iterations due to the rescaling. Although we acknowledge this issue, we consider that this effect should be of minor impact, nevertheless, the true impact of dynamic weight range rescaling on device endurance should be examined in future work.

5.8 Summary

In this chapter, the impact of two different non-idealities of the NCF aVMCO RRAM device: GS and C2C variability are experimentally extracted and characterized in the SET and RESET programming polarities using two different staged weight programming options: NR and LR. Using FlexiNNSim, the weight distributions are tracked during SET, RESET and two different practical implementations of the two (gradient-based and selective programming). This tracking revealed that the mechanisms behind accuracy degradation are different depending on its programming polarity, and this is linked to where the non-idealities are situated in the normalized weight range rather than simply their magnitude. It was also revealed that the GS non-ideality has a larger impact on accuracy degradation while C2C variability not only shows a minimal impact on its own, but when combined with GS, can help improve training in certain scenarios.

Furthermore, the learning rate was established as one of the most affected NN hyperparameters by the RRAM non-idealitites, as high learning rates tend to aggravate the effects of both GS and C2C.

The dynamic weight range rescaling methodology was introduced to challenge the convention of using fixed weight ranges in RRAM neuromorphic circuits that accommodate the device fixed conductance window. This method allows the weight range boundaries to be rescaled independently between NN layers and training iterations. Training with fixed weight ranges requires careful calibration of this range for each individual layer in a deep network, effectively adding an additional NN hyperparameter to consider. Dynamic weight range rescaling shows that avoiding letting the system

itself train the weight range not only allows for overall better accuracy, but better compatibility with other hyperparameters such as the learning rate.

Finally, based on the previous analysis, the accuracy improvements by the LR is evaluated and mainly attributed to the overall reduction of GS, and to a lower degree the C2C variability. A Selective programming approach is proposed as a combination of SET/RESET programming that has the device GS in mind. Using the Selective programming not only shows improved compatibility with other parameters such as the choice of learning rate, but combining this with the LR it was possible to achieve a state-of-the-art accuracy level of 95.1% in a simple CNN model, and 90% in a simple 1-L MLP using the NR, demonstrating its applicability in a wide range of NN settings.

Chapter 6

Conclusions & future perspectives

6.1 Conclusions

The goal of this work was focused on characterization of RRAM non-idealities and its effects on the accuracy of neuromorphic circuits for pattern recognition implemented RRAM synaptic crossbars, investigated through simulation and subsequently originating strategies for improvement. The impact of these non-idealities are analysed in two major parts: (1) impact on inference and (2) impact on training. The conclusions taken for each of these parts are given below:

6.1.1 Conclusions on inference

In Chapter 4, two types of RRAM devices: CF and NCF are characterized in terms of read noise and PIV.

By manipulating the sampling rate of read signals above or below the known devices RTN time constants, read signals at different conductance levels were decoupled into RTN and ORN signals respectively. Based on statistical experimental results taken from both types of noise, a read noise disturbance model is developed and applied to the trained synaptic arrays to simulate its impact on NN accuracy. Comparing both types of read noise reveals that the low amplitude of ORN results in negligible impact on NN accuracy for both devices. On the other hand, RTN comes in as the major source of read noise. Furthermore, it is concluded that NCF RRAM devices show smaller RTN amplitude, tighter RTN distribution, and lower RTN occurrence rate compared with its CF counterpart. As a result, the NNs with NCF synapses shows that RTN has a negligible impact in NN accuracy, and even smaller NNs can achieve better accuracy than larger NNs built with CF devices that rely on synaptic redundancy to counteract the effects of RTN.

Additionally, PIV was statistically measured at different conductance levels, using different weight update, W-V schemes and programming polarity, on two distinct devices based on CF and NCF switching. It is revealed that PIV $\frac{\Delta G}{G}$ variability can be modeled by Weibull distributions and in addition, these Weibull parameters across the normalized weight range can be described by a tilted Gaussian model (equation 4.5). Based on these models of variability, the impact of PIV programmed with different conditions is evaluated on the inference accuracy of a trained pattern recognition MLP with one hidden layer. The simulations reveal that: (1) the NCF device shows an overall much lower PIV than the CF device, (2) using a W-V scheme with the NR in the CF device still wields worse results than the NR in the NCF device without W-V, meaning

that (3) to guarantee the best results, a combination of LR and W-V is required. Finally, (4) the flexibility in programming polarity of the NCF device allows for further reducing PIV in this device by switching to the more linear SET programming polarity.

6.1.2 Conclusions on training

In Chapter 5, two RRAM non-idealities: GS and C2C variability, are independently measured, characterized and their impacts during the training of analog NNs are evaluated on the NCF device. The analysis of weight distribution evolution during training revealed that GS plays a dominant role in accuracy degradation. On the other hand, C2C variability not only a minimal impact but can also improve the accuracy upon the GS in specific use cases. Furthermore, it is revealed that the mechanisms behind accuracy degradation are different for SET and RESET, and this is directly linked to where the non-idealities are situated in the conductance range instead of only to its magnitude.

Additionally, a link has been established between the learning rate NN hyperparameter and non-ideal RRAM NNs has higher learning rates tend to aggravate the effects of GS and C2C.

Moreover, the dynamic weight range rescaling methodology aims to challenge the convention of matching the fixed RRAM conductance window to a fixed weight range by allowing the weight range boundaries to be rescaled independently between NN layers and training iterations. It was shown that fixed weight ranges in training of RRAM NNs requires careful calibration that is not only dependent on NN topology but also

other NN hyperparameters such as the learning rate, which is entirely avoidable with dynamic weight range rescaling.

Based on this analysis, the accuracy improvement by the LR is evaluated and linked mainly to the reduction of GS, and to a lower degree of relevance also the C2C variability. A Selective programming approach is proposed to further mitigate the GS by utilizing the 2^{nd} halves of SET or RESET program range. It was not only observed that the Selective programming method shows improved resilience to limiting factors such as the choice of learning rate, but with the combination with the LR is able to achieve a state-of-the-art accuracy level of 95.1% in our simple CNN model, and reaches 90% in the simplest one hidden layer MLP with the NR, which has the worst non-idealities, demonstrating its potential in practical analog NN applications.

6.2 Future perspectives

The work presented in this thesis was centered around characterization of individual RRAM devices and simulation of different strategies for accuracy improvement in NNs built with RRAM crossbar arrays. Nevertheless, the road from simulation to design and implementation is still long and challenging. Following is some prospective work towards neuromorphic implementation:

6.2.1 Extending the simulation framework

The simulation framework presented in this thesis can be seen as a complement to other existing frameworks. While most of the other frameworks are more detailed and complex, they are also very demanding in terms of hardware requirements and have limited flexibility. The framework built in this thesis is focused on providing flexibility for the end-user not only in regards to the options available concerning NN topology, hyperparameter setup, RRAM programming and user-accessible GUI, but also in terms of hardware requirements for the simulation itself, this however, comes at the cost of the level of detail provided for the peripheral circuitry.

Nevertheless, the natural progression for this framework would be to forego the simulation of peripheral circuitry in favour of optimized calculation speed towards the construction of a chip-in-loop system provided the correct interface with the RRAM crossbar arrays.

Even though the scope of this work is focused on simulating NN accuracy, further estimation tools could be built on top the existing simulation framework to estimate other crossbar parameters such as: device endurance requirements, read/write latency and power consumption.

Finally, with the required hardware, a comparison should be drawn between this framework and other existing frameworks relating parameters such as: time per epoch, CPU/GPU utilization, RAM and VRAM such as that of Figures 1.38 & 1.39.

6.2.2 Future perspectives for inference

This work analysed the impact of RTN and other forms of read noise as well as PIV in NN inference accuracy. Models that describe RTN, ORN and PIV were proposed in this work, however in real-world scenarios, other device issues such as conductance retention, read disturbances and device yields will come into play. Future work in this regard should contemplate the noise models we propose as complementary material for other types of model in order to build a more complete picture of RRAM behaviour during inference.

We consider that the modelling of Weibull distribution parameters that can be described by our tilted Gaussian model, although being a conceptual model at this stage, can be an important step forward towards the development of a physics based model that could describe PIV in RRAM devices, which in turn is an important developmental step for the design stage of RRAM neuromorphic crossbar arrays.

6.2.3 Future perspectives for training

Regarding training, there is still considerable work to be done to transfer concepts that are proposed not only at the simulation stage, but also between inference based RRAM synaptic arrays and trainable circuits with RRAM. As described in the previous subsection, issues such as: retention, device yields and read disturbances may have added effects, to which training-specific issues such as: read/write latencies, power consumptions and cycling endurance are added. As with the inference case, the work

presented here is meant to be used in addition to other models that accurately portray these additional issues.

Furthermore, we have with the concepts of staged weight update scheme, dynamic range rescaling and Selective programming, interesting perspectives to mitigate the problems of GS and C2C variability. However, the design of the crossbar peripheral circuitry to allow these concepts to take place in real-world scenarios remains a challenging prospect. Additionally, the real device endurance requirements for Selective programming and dynamic range rescaling will need to be analysed and discussed in-depth.

Appendix A

Impact of learning rate

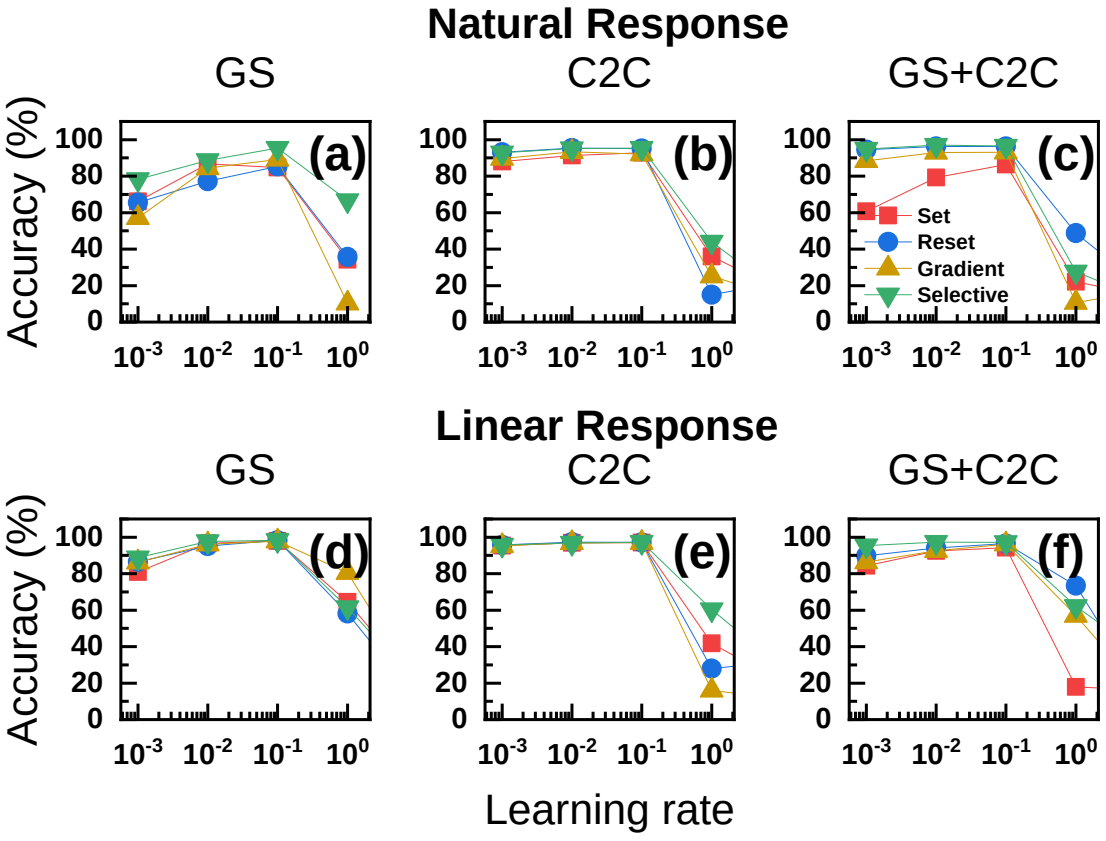
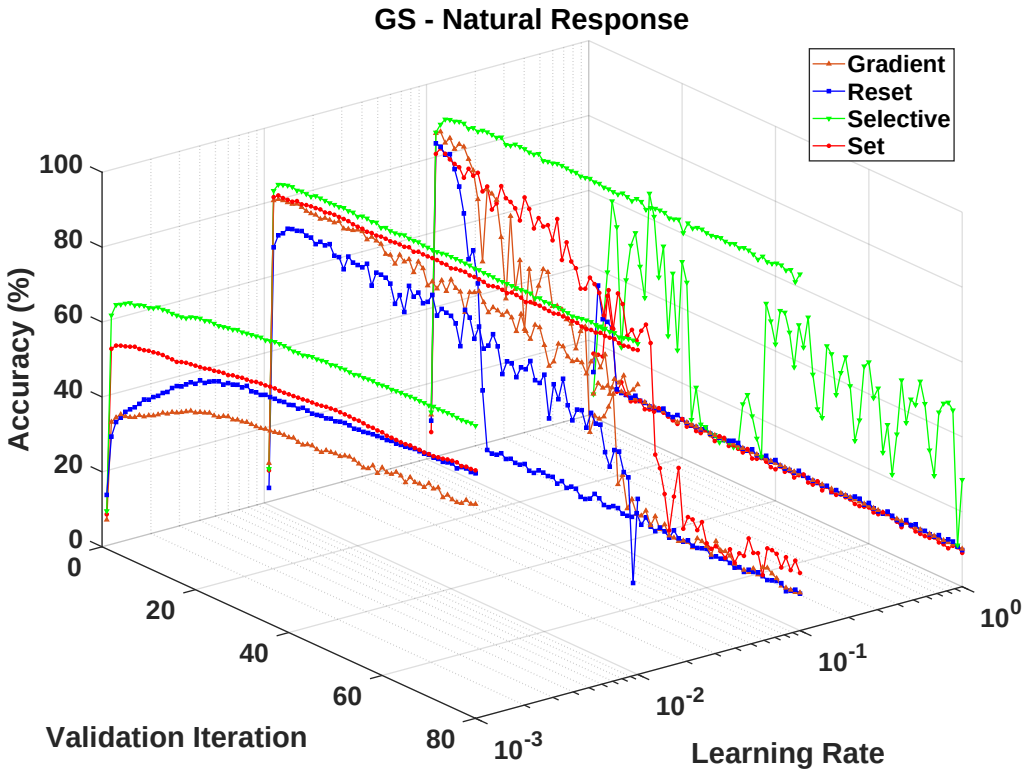


Figure A.1: Impact of learning rate on the **maximum** accuracy achieved by the CNN model from Fig. 5.4a using the Natural Response (a - c), evaluated with the (a) GS, (b) C2C and (c) GS combined with C2C non-idealities. (d - f) shows the same methodology using the Linear Response.

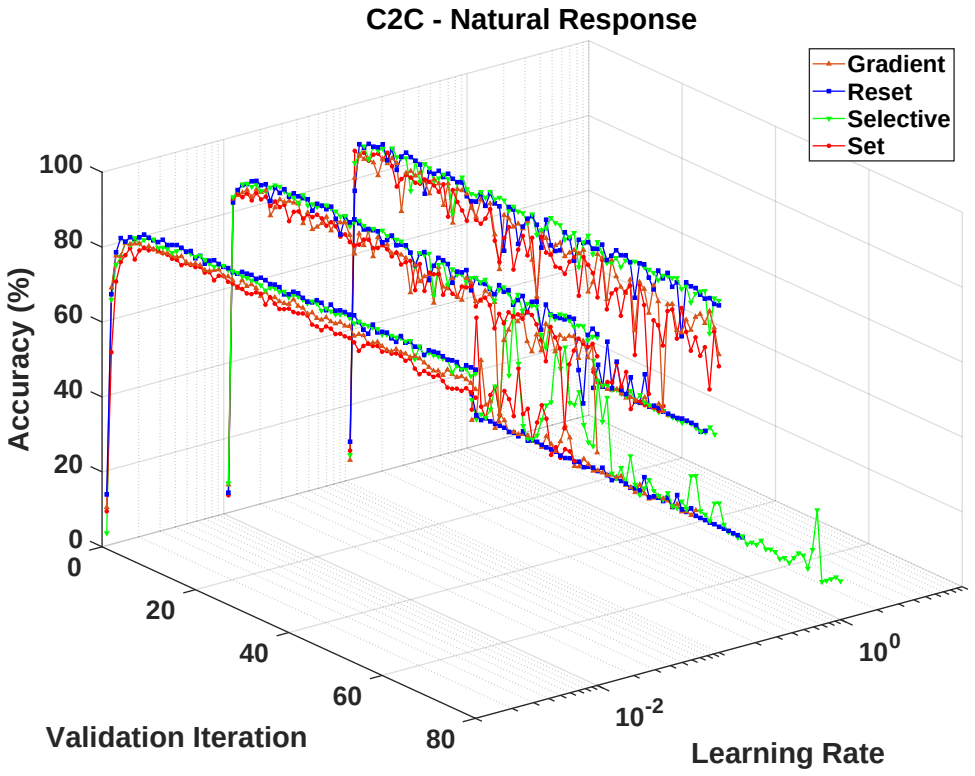
(a)


GS - Natural Response
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-3}
10^{-2}
10^{-1}
10^{0}
Learning Rate

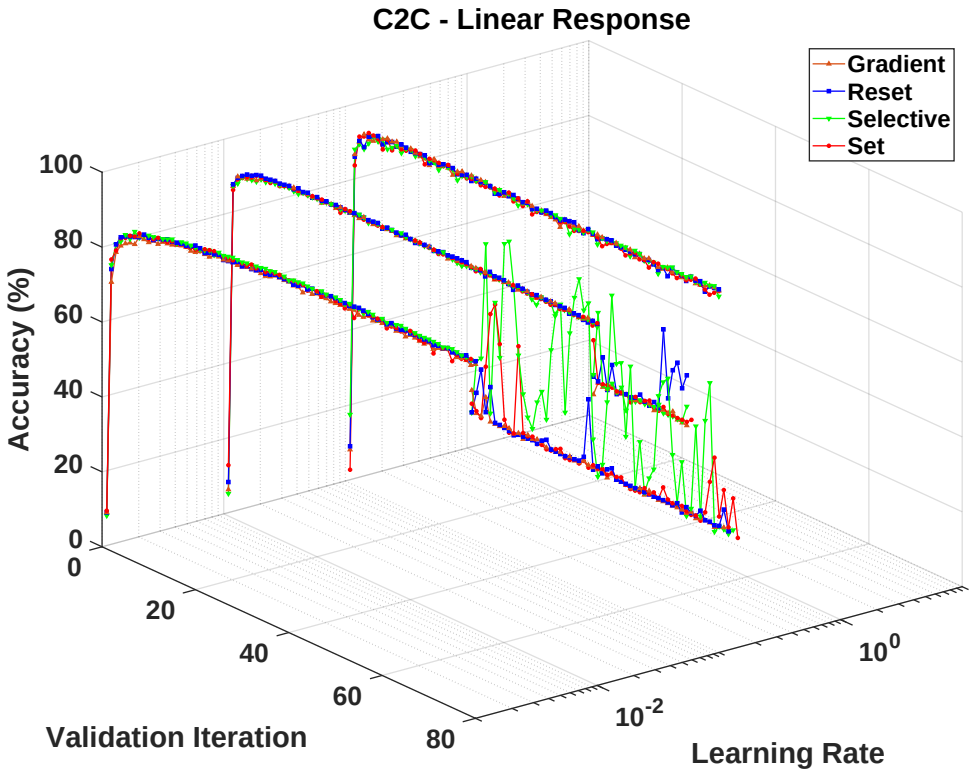
(b)


GS - Linear Response
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-2}
10^{0}
Learning Rate

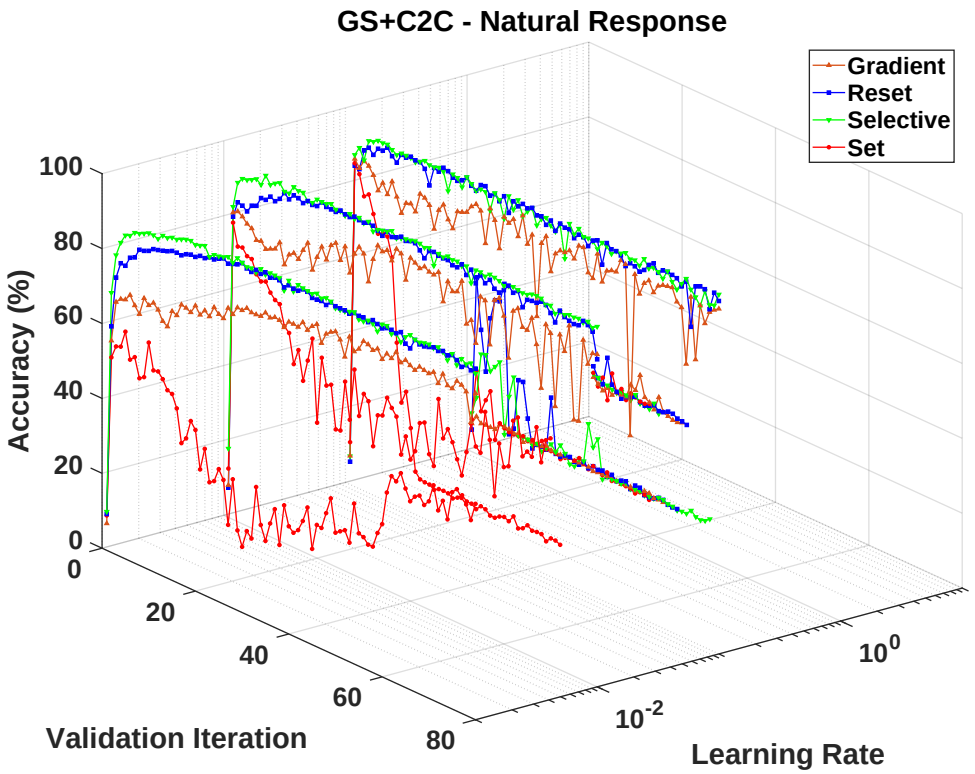
(c)


C2C - Natural Response
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-2}
10^{0}
Learning Rate

(d)


C2C - Linear Response
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-2}
10^{0}
Learning Rate

(e)



(f)

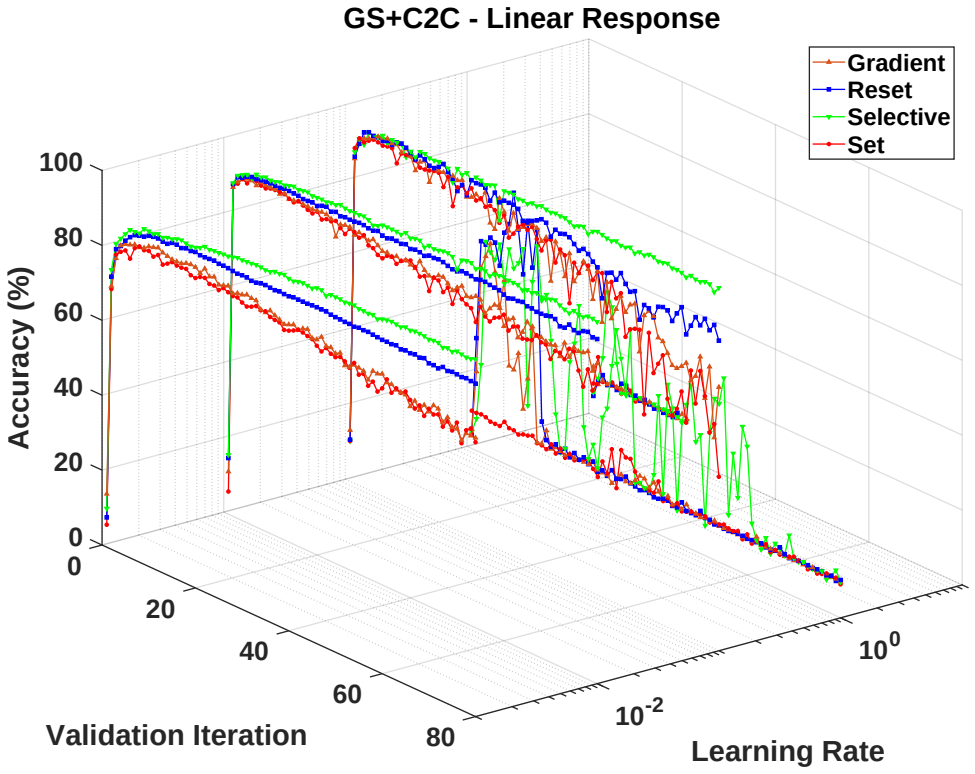


Figure A.2: Impact of learning rate on validation accuracy during training. Different programming modes (SET, RESET, Gradient-based and Selective) are analysed, as well as the impact of the Natural vs Linear Response on the different non-idealities: (a - b) GS, (c - d) C2C and (e - f) GS+C2C.

Appendix B

Impact of Dynamic Range Rescaling

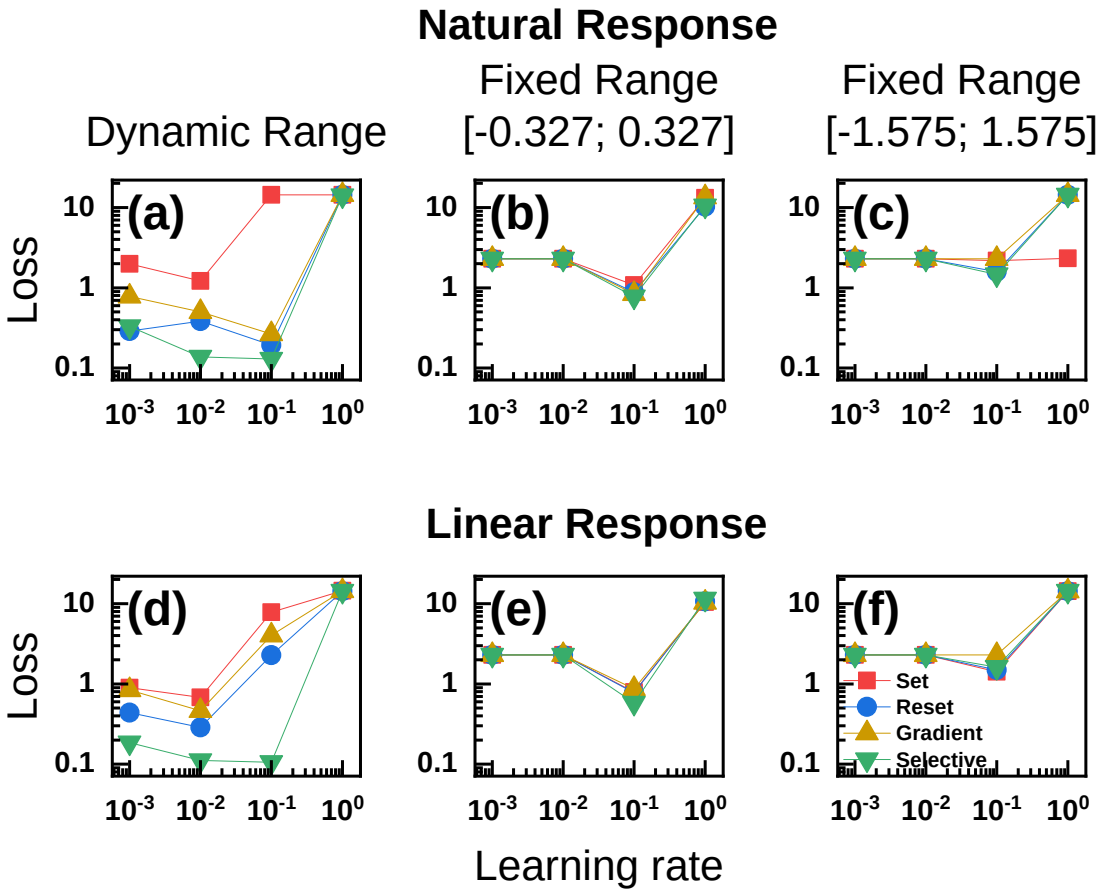
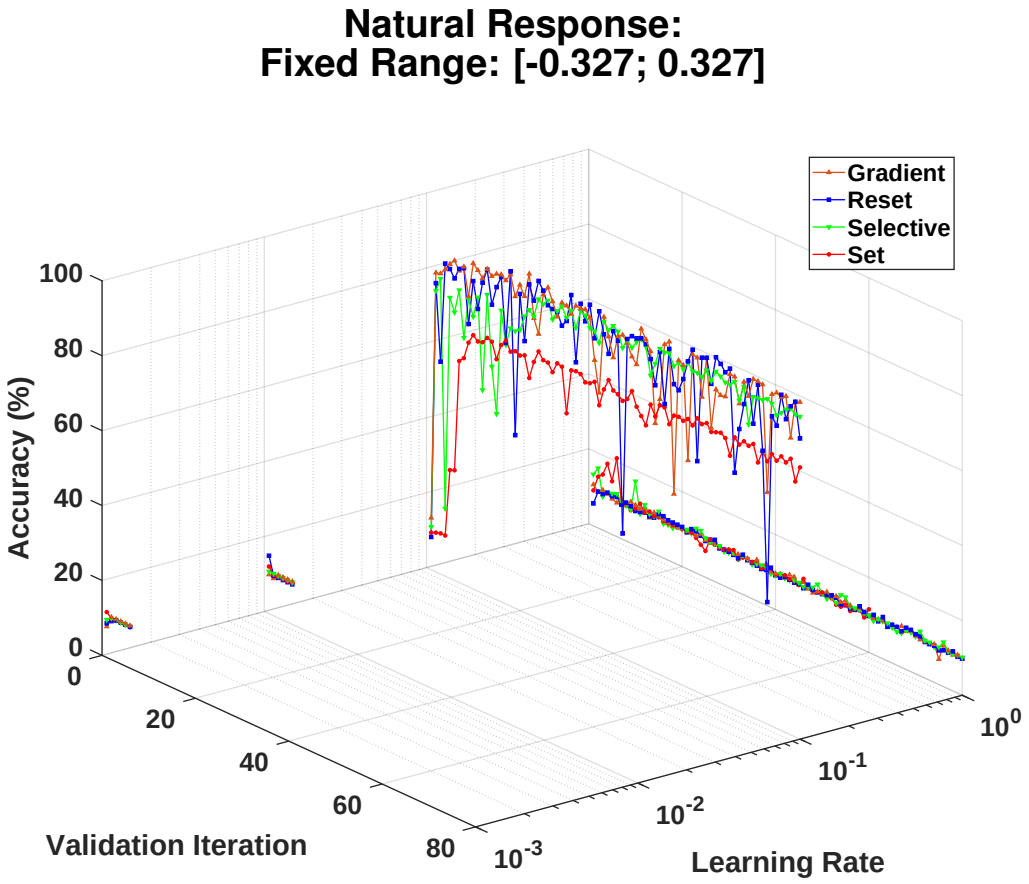


Figure B.1: Impact of Dynamic Weight Range rescaling compared to two fixed ranges: [-0.327; 0.327] & [-1.575, 1.575] on the final validation loss of a CNN. Different programming methods (SET, RESET, Gradient and Selective) illustrated on both the aVMCO Natural Response (a - c) and Linear Response (d - f).

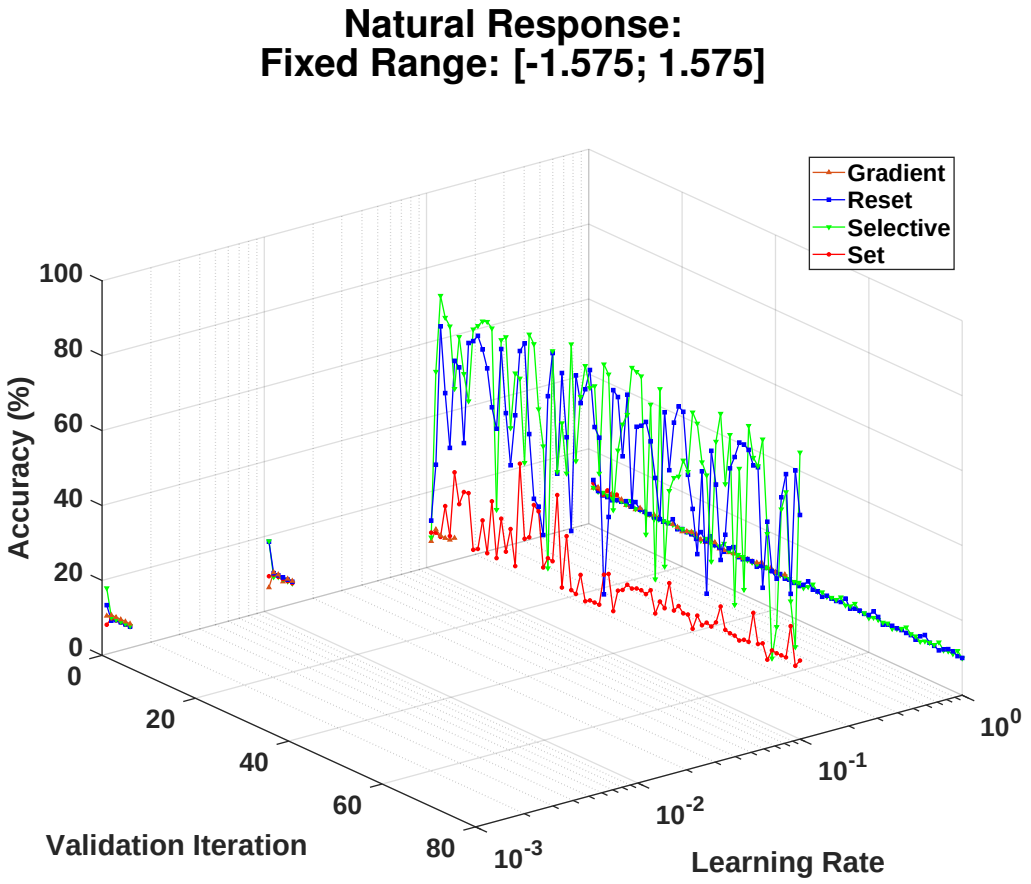
(a)


Natural Response:
Dynamic Range
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-3}
10^{-2}
10^{-1}
10^{0}
Learning Rate

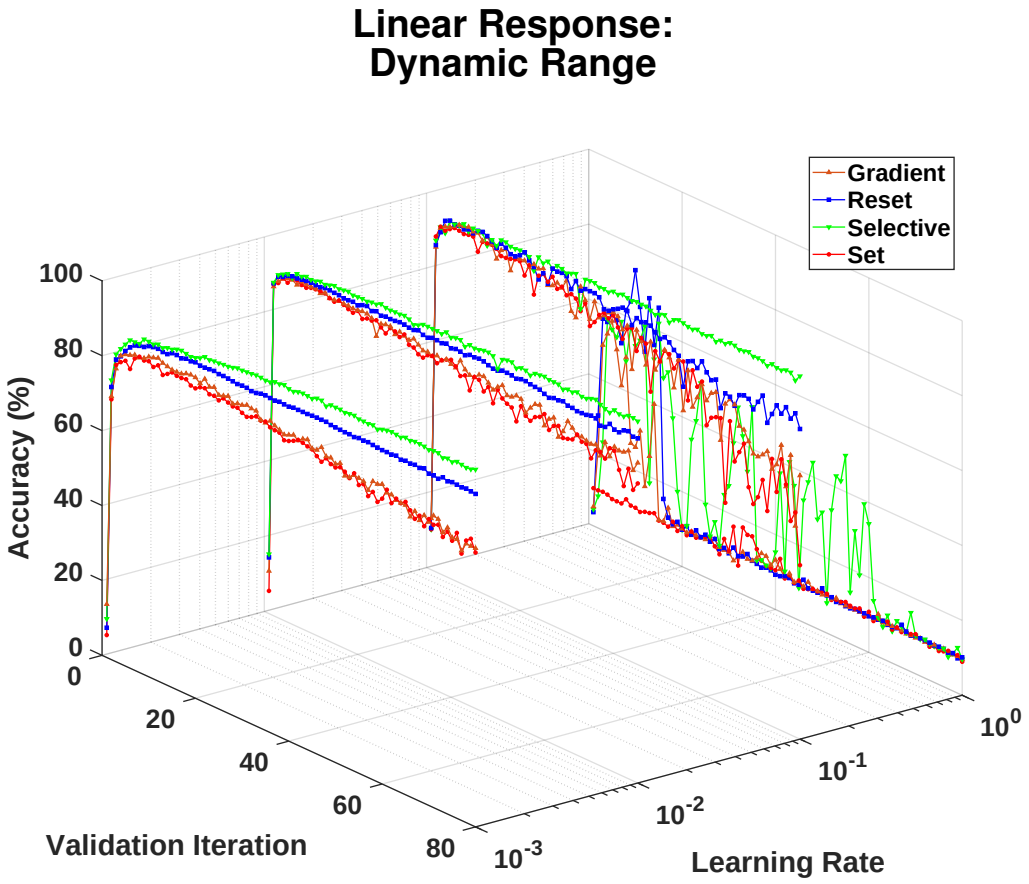
(b)


Natural Response:
Fixed Range: [-0.327; 0.327]
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-3}
10^{-2}
10^{-1}
10^{0}
Learning Rate

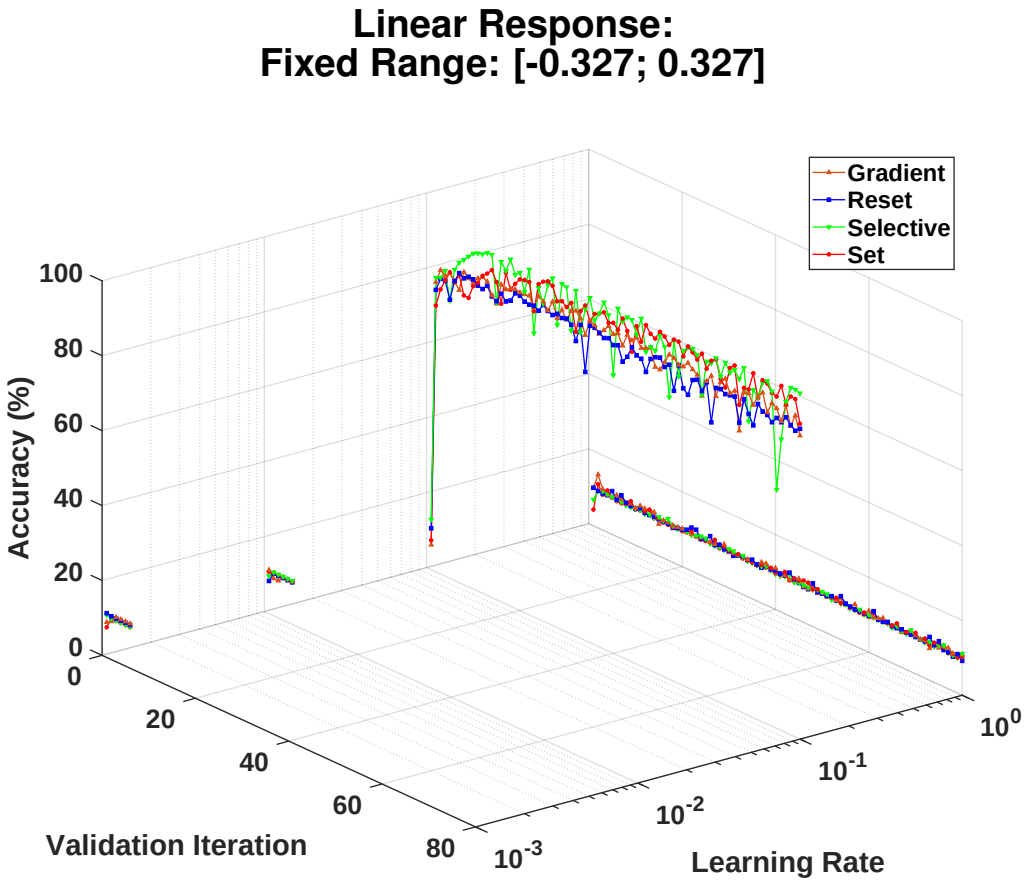
(c)


Natural Response:
Fixed Range: [-1.575; 1.575]
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-3}
10^{-2}
10^{-1}
10^{0}
Learning Rate

(d)


Linear Response:
Dynamic Range
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-3}
10^{-2}
10^{-1}
10^{0}
Learning Rate

(e)


Linear Response:
Fixed Range: [-0.327; 0.327]
Gradient
Reset
Selective
Set
Accuracy (%)
100
80
60
40
20
0
0
20
40
60
80
Validation Iteration
10^{-3}
10^{-2}
10^{-1}
10^{0}
Learning Rate

(f)

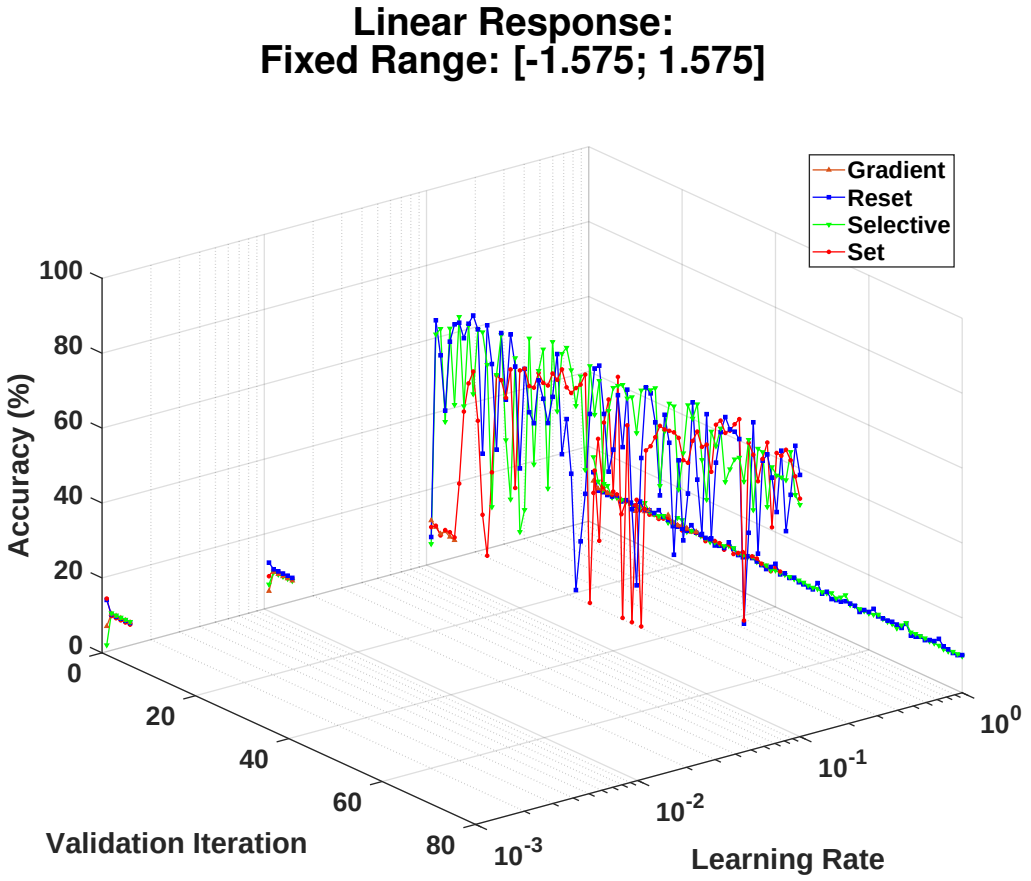


Figure B.2: Evolution of validation accuracy during training of the CNN using dynamic range rescaling (a & d) and two examples of fixed range scaling: one with a range of [-0.327; 0.327] for all CNN layers (b & e) and the other with a range of [-1.575; 1.575] (c & f).

Appendix C

Limited Precision training based on genetic algorithms

In the previous chapters, the impact of noise and variability of RRAM devices was analysed for full precision NNs that are meant to be trained in an analogue setting. However, most neuromorphic systems with RRAM available in the literature are implemented in the digital domain. In the digital setting, the bit precision of the NN weights becomes a significant factor to take into consideration, since this will not only affect the chip area and power consumption footprint that is largely dominated by memory buffers and ADC/DAC peripheral circuitry [105], but also largely affects the impact of hardware noise.

Being so, there is a large demand for software solutions that are able to use limited precision weights to train ML models. Nonetheless, current solutions rely on some variation of gradient-based learning, either by preserving FP32 weights for gradient

calculation, or in the cases that do calculate the gradients with LP, doing so with at least 6-bit precision (see section 1.2.2).

Previously, in this work, the focus was on a top-down design approach focusing on mitigating the non-idealities present in the hardware, using traditional gradient-descent algorithms. In this chapter, however, we aim at constructing LP GA designed from the bottom-up to accommodate the non-idealities of RRAM. GAs were chosen following the work of Stromatias and Marsland [161] where GAs were used to circumvent the problem of weight discontinuity for training of biological inspired SNNs on non-linear two-dimensional classification problems, whereas here we attempt to build LP GAs for training of traditional NNs for image classification.

C.0.1 Algorithm structure and typical results

Our algorithm uses the same basic structure that was introduced in section 1.2.1.10, but with a few particularities:

- Each individual of the population is a bitstream that represents every weight of the NN that is posteriorly decoded into the weight matrices.
- Each bitstream weight is decoded into a gray code representation of a weight value.
- Mutation rate values are distinctly set for odd and evenly numbered individuals so that the mutation rate can be controlled for the individuals that crossover.

- Each crossover event is randomly performed using one of five distinct methods: (1) Uniform crossover - bit wise random; (2) Uniform crossover - weight wise random; (3) Uniform crossover - node wise random; (4) Fold crossover - bit wise; (5) Child is a clone of the parent but mutated.

A wide range of different hyperparameters must be considered in this algorithm for both the traditional NN topologies as well as the GA specific hyperparameters. Table C.1 shows the default hyperparameters used for this algorithm.

Table C.1: Table containing the (left) NN and (right) GA default parameters of the LP GA trained MLP.

NN Parameter	Value
Weight precision	3 bit
Activation Gain	1
Cost Function	CE
Hidden Layer Activations	Leaky ReLU
Leaky ReLU Scale (α)	0.001
Output Layer Activation	Softmax
NN Topology	784x30x10

GA Parameter	Value
Mutation Rate	[70, 70]
Ranking Selective pressure	1.9
Elitism operator	8
Max Generations	1000
Population Size	100

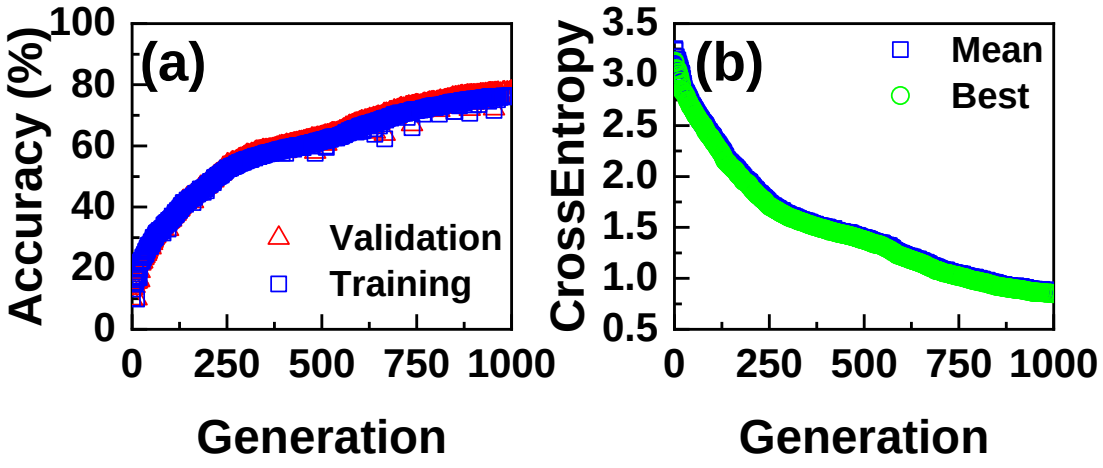


Figure C.1: Typical example of LP GA (a) accuracy and (b) cost curves per generation of a 3-bit trained 1L-MLP on the MNIST database.

Figure C.1 shows the training curves depicting the accuracy and crossentropy per generation when using the default training hyperparameters on the MNIST database. It is noteworthy that $\approx 76\%$ accuracy is able to be achieved in both the training and

validation data, being that the drawback is the long time to convergence that is inherent to GAs for training sparse datasets such as the MNIST database.

C.0.2 Impact of weight bitwidth

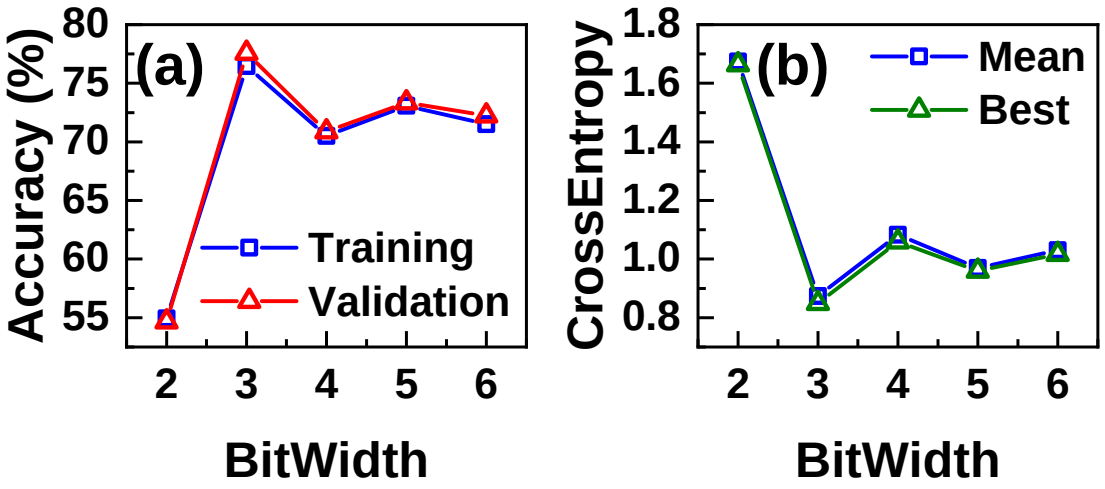


Figure C.2: Impact of weight bit-precision on the (a) accuracy and (b) cost of a LP GA trained 1L-MLP.

Using the same set of parameters, the weight bitwidth was varied between 2 to 6 bits and a summary of bitwidth impact on accuracy is displayed in Figure C.2. As expected, training with 2-bits results in a penalty of accuracy of $\approx 20\%$, however, counter-intuitively, increasing the bitwidth beyond 3-bits results in worse performance than the standard 3-bit case. One possible explanation is that in a global search algorithm such as GA, increasing the bitwidth of the weights also increases the search space of solutions that the algorithm is required to cover in what is already a very sparse setting.

C.0.3 Impact of mutation rate and activation functions

The impact of mutation rate and choice of hidden layer activation function was studied in this work, however, due to time-constraints and hardware limitations, a Limited MNIST dataset containing 10% of the original database was used. Figure C.3 compares the impact in accuracy of the limited dataset compared to the full dataset, using the default training parameters. It was shown that a small penalty of $\approx 10\%$ occurs in the training with the limited data, and $\approx 20\%$ for the validation data.

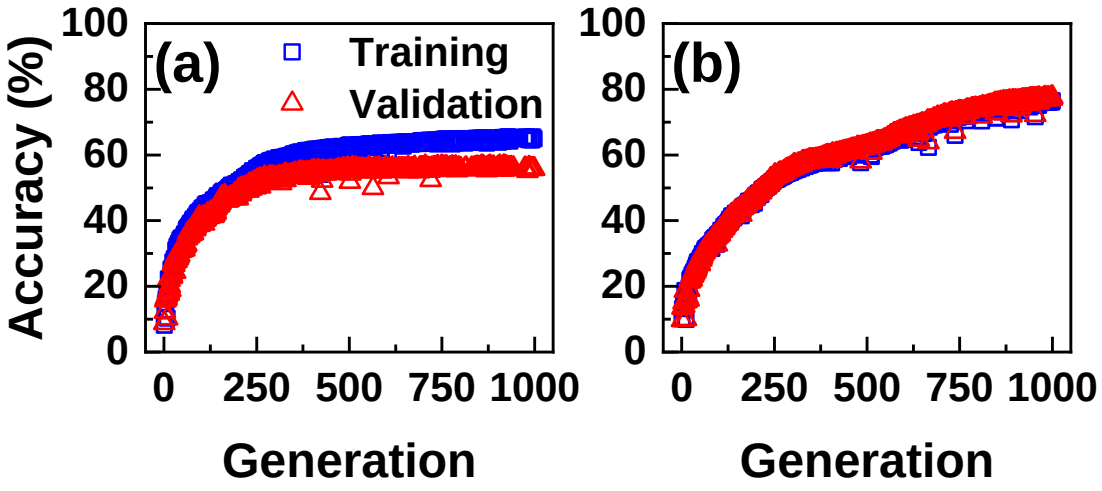


Figure C.3: Training curves of GA LP using (a) the limited dataset and (b) the full dataset.

As such, the accuracy of the training data on the limited dataset is analysed for the study on mutation rate and activation function, due to its smaller penalty in accuracy but significant speed-up in simulation time. Figure C.4 shows the impact of varying the mutation rate with different hidden layer activation functions. It was determined that a high mutation rate value of 70 shows the best accuracy results while using the Leaky ReLU activation function when using 2-bits, while for 3-bits, the best mutation rate values maintain the same, but sigmoid or ReLU activations have comparable accuracy results.

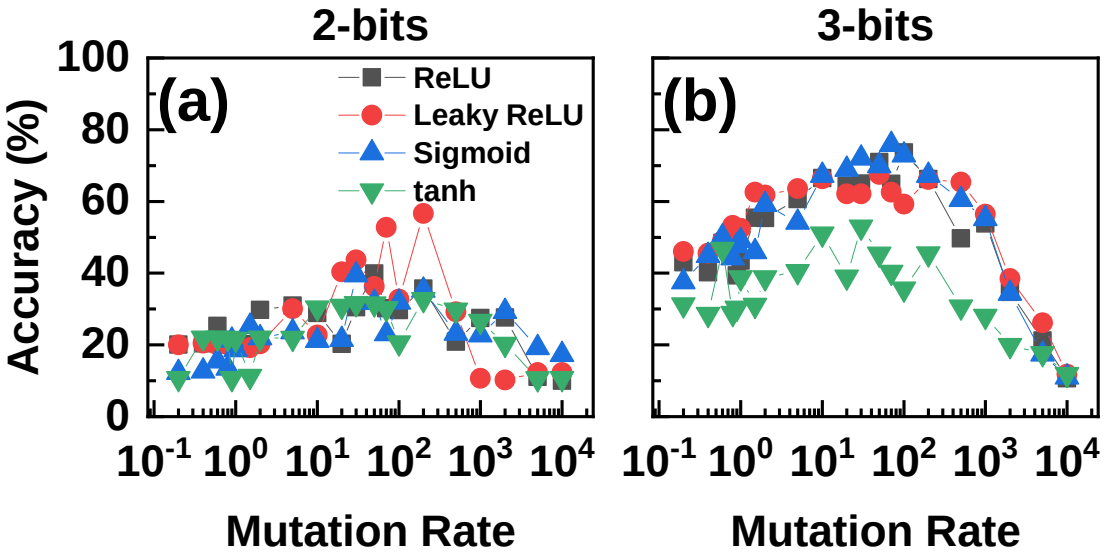


Figure C.4: Impact of mutation rate on the training accuracy of GA trained 1L-MLP using different hidden layer activation functions with LP of (a) 2-bits and (b) 3-bits.

Furthermore, different mutation rates for odd and even numbered individuals in an effort to examine the effects of crossover between individuals with different mutation rates was tested. Figure C.5 shows, however, that there is little change in accuracy by changing the mutation rate of odd numbered individuals, which could be an indication that the training of this algorithm is dominated by the high mutation rate of the even numbered individuals, suggesting that mutation plays a higher role in training than crossover.

C.0.4 Impact of read noise

Finally, the impact of read noise introduced in the LP GA network during every feed-forward operation as an attempt to simulate read noise such as RTN is examined in Figure C.6. It is shown that even for relatively small normalized deviation values of 0.1, there is a high decrease in accuracy of $\approx 18\%$ and $\approx 25\%$ for 2-bit and 3-bit training respectively. One possible explanation for this effect could be the large fan-in that

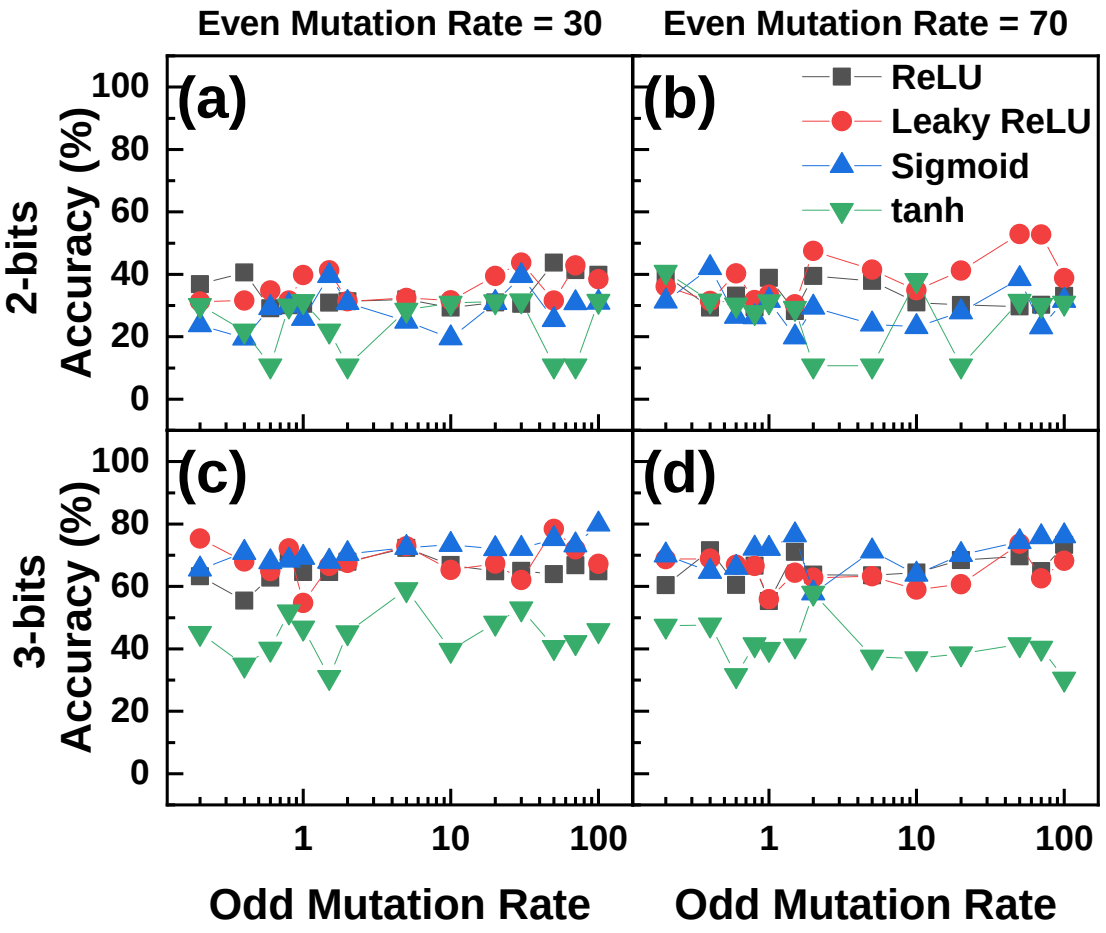


Figure C.5: Impact of odd individual mutation rate combined with fixed even individual mutation rate trained with LP of (a - b) 2-bits and (c - d) 3-bits.

exists in the 784x30x10 topology's first hidden layer, as the noises of a large number of devices on each single column can accumulate.

C.0.5 LP GA Conclusions

Training with limited precision was attempted using GAs as a global search strategy in which every individual represents the weight matrices of a 1-L MLP to train in the MNIST database. This algorithm allows for training without recurring to gradient calculations with discrete weight values, which is the major challenge in LP algorithms based on gradient descent.

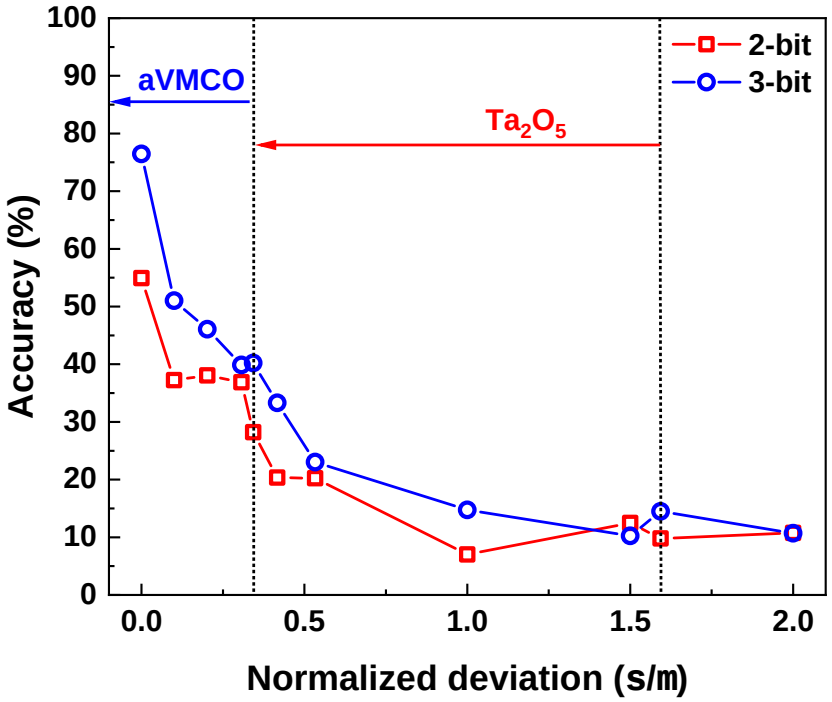


Figure C.6: Impact of noise in accuracy of the LP GA trained NN. Typical RTN amplitude values of the aVMCO and Ta_2O_5 devices are represented for reference.

A training accuracy with 3-bit weights of $\approx 76\%$ was achieved using this method. Nevertheless, training with LP GAs comes with its own set of challenges, most notably the slow speed to convergence that is inherent to GAs for sparse datasets such as MNIST.

However, it was also determined that increasing the weight bit-width in this global search algorithm is detrimental to accuracy, likely due to an increase of the search space that the algorithm is required to cover.

Even though major hurdles still need to be crossed for the practical implementation of LP GAs, the conclusions provided in this work will hopefully not only be applicable to other GA methods but also to the wider category of LP global search algorithms.

Bibliography

[1] J. Von Neumann, *First draft of a report on the EDVAC*. Moore School of Electrical Engineering, University of Pennsylvania, 1945.

[2] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, pp. 529–544, jul 2020.

[3] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, pp. 489–502, apr 2003.

[4] L. Chua, "Memristor-The missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[5] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, may 2008.

[6] T. Prodromakis, C. Toumazou, and L. Chua, "Two centuries of memristors," *Nature Materials*, vol. 11, pp. 478–481, jun 2012.

[7] J. Frith and C. Rodgers, "On the resistance of the electric arc," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 42, no. 258, pp. 407–423, 1896.

[8] W. Duddell, "On the resistance and electromotive forces of the electric arc," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 203, pp. 305–342, jan 1904.

[9] IEEE, "International Roadmap for Devices and Systems 2021 Update - Beyond CMOS," tech. rep., 2021.

[10] S. Lai, "Current status of the phase change memory and its future," in *IEEE International Electron Devices Meeting 2003*, pp. 10.1.1–10.1.4, IEEE, 2003.

[11] J. C. Slonczewski, "Conductance and exchange coupling of two ferromagnets separated by a tunneling barrier," *Physical Review B*, vol. 39, pp. 6995–7002, apr 1989.

[12] J. Slonczewski, "Current-driven excitation of magnetic multilayers," *Journal of Magnetism and Magnetic Materials*, vol. 159, pp. L1–L7, jun 1996.

[13] L. Berger, "Emission of spin waves by a magnetic multilayer traversed by a current," *Physical Review B*, vol. 54, pp. 9353–9358, oct 1996.

[14] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-Based Resistive Switching Memories - Nanoionic Mechanisms, Prospects, and Challenges," *Advanced Materials*, vol. 21, pp. 2632–2663, jul 2009.

[15] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2237–2251, 2010.

[16] D. Ielmini, "Resistive switching memories based on metal oxides: mechanisms, reliability and scaling," *Semiconductor Science and Technology*, vol. 31, p. 063002, jun 2016.

[17] I. Valov, R. Waser, J. R. Jameson, and M. N. Kozicki, "Electrochemical metallization memories—fundamentals, applications, prospects," *Nanotechnology*, vol. 22, p. 289502, jul 2011.

[18] M. Kozicki, M. Mitkova, M. Park, M. Balakrishnan, and C. Gopalan, "Information storage using nanoscale electrodeposition of metal in solid electrolytes," *Superlattices and Microstructures*, vol. 34, pp. 459–465, sep 2003.

[19] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electronics*, vol. 125, pp. 25–38, nov 2016.

[20] B. Li, B. Yan, and H. Li, "An Overview of In-memory Processing with Emerging Non-volatile Memory for Data-intensive Applications," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, (New York, NY, USA), pp. 381–386, ACM, may 2019.

[21] D. Kuzum, S. Yu, and H.-S. Philip Wong, "Synaptic electronics: materials, devices and applications," *Nanotechnology*, vol. 24, p. 382001, sep 2013.

[22] D. Querlioz, O. Bichler, A. F. Vincent, and C. Gamrat, "Bioinspired Programming of Memory Devices for Implementing an Inference Engine," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1398–1416, 2015.

[23] E. Vianello, T. Werner, A. Grossi, E. Nowak, B. De Salvo, L. Perniola, O. Bichler, and B. Yvert, "Bioinspired Programming of Resistive Memory Devices for Implementing Spiking Neural Networks," *Proceedings of the on Great Lakes Symposium on VLSI 2017 - GLSVLSI '17*, vol. 1, pp. 393–398, 2017.

[24] G. S. Rose, N. McDonald, L.-K. Yan, and B. Wysocki, "A write-time based memristive PUF for hardware security applications," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 830–833, IEEE, nov 2013.

[25] A. Iyengar, K. Ramclam, and S. Ghosh, "DWM-PUF: A low-overhead, memory-based security primitive," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 154–159, IEEE, may 2014.

[26] J. Rajendran, R. Karri, J. B. Wendt, M. Potkonjak, N. McDonald, G. S. Rose, and B. Wysocki, "Nano Meets Security: Exploring Nanoelectronic Devices for Security Applications," *Proceedings of the IEEE*, vol. 103, pp. 829–849, may 2015.

[27] T. W. Hickmott, "Low-Frequency Negative Resistance in Thin Anodic Oxide Films," *Journal of Applied Physics*, vol. 33, pp. 2669–2682, sep 1962.

[28] J. Gibbons and W. Beadle, "Switching properties of thin Nio films," *Solid-State Electronics*, vol. 7, pp. 785–790, nov 1964.

[29] G. Dearnaley, A. M. Stoneham, and D. V. Morgan, "Electrical phenomena in amorphous oxide films," *Reports on Progress in Physics*, vol. 33, p. 306, sep 1970.

[30] J. G. Simmons, "Conduction in thin dielectric films," *Journal of Physics D: Applied Physics*, vol. 4, p. 202, may 1971.

[31] S. Seo, M. J. Lee, D. H. Seo, E. J. Jeoung, D.-S. Suh, Y. S. Joung, I. K. Yoo, I. R. Hwang, S. H. Kim, I. S. Byun, J.-S. Kim, J. S. Choi, and B. H. Park, "Reproducible resistance switching in polycrystalline NiO films," *Applied Physics Letters*, vol. 85, pp. 5655–5657, dec 2004.

[32] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, and C. S. Hwang, "Memristors for Energy-Efficient New Computing Paradigms," *Advanced Electronic Materials*, vol. 2, p. 1600090, sep 2016.

[33] N. Xu, L. Liu, X. Sun, X. Liu, D. Han, Y. Wang, R. Han, J. Kang, and B. Yu, "Characteristics and mechanism of conduction/set process in TiN/ZnO/Pt resistance switching random-access memories," *Applied Physics Letters*, vol. 92, p. 232112, jun 2008.

[34] J. Joshua Yang, F. Miao, M. D. Pickett, D. A. A. Ohlberg, D. R. Stewart, C. N. Lau, and R. S. Williams, "The mechanism of electroforming of metal oxide memristive switches," *Nanotechnology*, vol. 20, p. 215201, may 2009.

[35] C.-W. Hsu, Y.-F. Wang, C.-C. Wan, I.-T. Wang, C.-T. Chou, W.-L. Lai, Y.-J. Lee, and T.-H. Hou, "Homogeneous barrier modulation of TaO x /TiO 2 bilayers for ultra-high endurance three-dimensional storage-class memory," *Nanotechnology*, vol. 25, p. 165202, mar 2014.

[36] B. Govoreanu, L. Di Piazza, J. Ma, T. Conard, A. Vanleenhove, A. Belmonte, D. Radisic, M. Popovici, A. Velea, A. Redolfi, O. Richard, S. Clima, C. Adelmann, H. Bender, and M. Jurczak, "Advanced a-VMCO resistive switching memory through inner interface engineering with wide (¿10 2) on/off window, tunable μA-range switching current and excellent variability," in *2016 IEEE Symposium on VLSI Technology*, vol. 2016-Septe, pp. 1–2, IEEE, jun 2016.

[37] B. Govoreanu, D. Crotti, S. Subhechha, L. Zhang, Y. Chen, S. Clima, V. Paraschiv, H. Hody, C. Adelmann, M. Popovici, O. Richard, and M. Jurczak, "A-VMCO: A novel forming-free, self-rectifying, analog memory cell with low-current operation, nonfilamentary switching and excellent variability," in *2015 Symposium on VLSI Technology (VLSI Technology)*, vol. 2015-Augus, pp. T132–T133, IEEE, jun 2015.

[38] K. Moon, A. Fumarola, S. Sidler, J. Jang, P. Narayanan, R. M. Shelby, G. W. Burr, and H. Hwang, "Bidirectional Non-Filamentary RRAM as an Analog Neuromorphic Synapse, Part I: Al/Mo/Pr 0.7 Ca 0.3 MnO 3 Material Improvements and Device Measurements," *IEEE Journal of the Electron Devices Society*, vol. 6, pp. 146–155, dec 2018.

[39] A. Fumarola, S. Sidler, K. Moon, J. Jang, R. M. Shelby, P. Narayanan, Y. Leblebici, H. Hwang, and G. W. Burr, "Bidirectional Non-Filamentary RRAM as an Analog Neuromorphic Synapse, Part II: Impact of Al/Mo/Pr 0.7 Ca 0.3 MnO 3 Device Characteristics on Neural Network Training Accuracy," *IEEE Journal of the Electron Devices Society*, vol. 6, no. 1, pp. 169–178, 2018.

[40] Jen-Chieh Liu, I-Ting Wang, C.-W. Hsu, W.-C. Luo, and Tuo-Hung Hou, "Investigating MLC variation of filamentary and non-filamentary RRAM," in *Proceedings of Technical Program - 2014 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, pp. 1–2, IEEE, apr 2014.

[41] B. Govoreanu, A. Redolfi, L. Zhang, C. Adelmann, M. Popovici, S. Clima, H. Hody, V. Paraschiv, I. Radu, A. Franquet, J.-C. Liu, J. Swerts, O. Richard, H. Bender, L. Altimime, and M. Jurczak, "Vacancy-modulated conductive oxide resistive RAM (VMCO-RRAM): An area-scalable switching current, self-compliant, highly nonlinear and wide on/off-window resistive switching cell," in *2013 IEEE International Electron Devices Meeting*, pp. 10.2.1–10.2.4, IEEE, dec 2013.

[42] M. N. Kozicki and H. J. Barnaby, "Conductive bridging random access memory—materials, devices and applications," *Semiconductor Science and Technology*, vol. 31, p. 113001, nov 2016.

[43] C.-W. Huang, J.-Y. Chen, C.-H. Chiu, and W.-W. Wu, "Revealing Controllable Nanowire Transformation through Cationic Exchange for RRAM Application," *Nano Letters*, vol. 14, pp. 2759–2763, may 2014.

[44] B. Gao, S. Yu, N. Xu, L. Liu, B. Sun, X. Liu, R. Han, J. Kang, B. Yu, and Y. Wang, "Oxide-based RRAM switching mechanism: A new ion-transport-recombination model," in *2008 IEEE International Electron Devices Meeting*, pp. 1–4, IEEE, dec 2008.

[45] T.-N. Fang, S. Kaza, S. Haddad, A. Chen, Y.-C. J. Wu, Z. Lan, S. Avanzino, D. Liao, C. Gopalan, S. Choi, S. Mahdavi, M. Buynoski, Y. Lin, C. Marrian, C. Bill, M. VanBuskirk, and M. Taguchi, "Erase Mechanism for Copper Oxide Resistive Switching Memory Cells with Nickel Electrode," in *2006 International Electron Devices Meeting*, pp. 1–4, IEEE, 2006.

[46] U. Russo, D. Ielmini, C. Cagli, and A. L. Lacaita, "Filament Conduction and Reset Mechanism in NiO-Based Resistive-Switching Memory (RRAM) Devices," *IEEE Transactions on Electron Devices*, vol. 56, pp. 186–192, feb 2009.

[47] T. Chang, S.-H. Jo, and W. Lu, "Short-Term Memory to Long-Term Memory Transition in a Nanoscale Memristor," *ACS Nano*, vol. 5, pp. 7669–7676, sep 2011.

[48] X. Yan, J. Zhao, S. Liu, Z. Zhou, Q. Liu, J. Chen, and X. Y. Liu, "Memristor with Ag-Cluster-Doped TiO 2 Films as Artificial Synapse for Neuroinspired Computing," *Advanced Functional Materials*, vol. 28, p. 1705320, jan 2018.

[49] M. Wang, S. Cai, C. Pan, C. Wang, X. Lian, Y. Zhuo, K. Xu, T. Cao, X. Pan, B. Wang, S.-J. Liang, J. J. Yang, P. Wang, and F. Miao, "Robust memristors based on layered two-dimensional materials," *Nature Electronics*, vol. 1, pp. 130–136, feb 2018.

[50] Z. Xiao and J. Huang, "Energy-Efficient Hybrid Perovskite Memristors and Synaptic Devices," *Advanced Electronic Materials*, vol. 2, p. 1600100, jul 2016.

[51] Y. Ren, V. Milo, Z. Wang, H. Xu, D. Ielmini, X. Zhao, and Y. Liu, "Analytical Modeling of Organic-Inorganic CH 3 NH 3 PbI 3 Perovskite Resistive Switching and its Application for Neuromorphic Recognition," *Advanced Theory and Simulations*, vol. 1, p. 1700035, apr 2018.

[52] G. Liu, C. Wang, W. Zhang, L. Pan, C. Zhang, X. Yang, F. Fan, Y. Chen, and R.-W. Li, "Organic Biomimicking Memristor for Information Storage and Processing Applications," *Advanced Electronic Materials*, vol. 2, p. 1500298, feb 2016.

[53] Y. S. Chen, H. Y. Lee, P. S. Chen, P. Y. Gu, C. W. Chen, W. P. Lin, W. H. Liu, Y. Y. Hsu, S. S. Sheu, P. C. Chiang, W. S. Chen, F. T. Chen, C. H. Lien, and M.-J. Tsai, "Highly scalable hafnium oxide memory with improvements of resistive distribution and read disturb immunity," in *2009 IEEE International Electron Devices Meeting (IEDM)*, pp. 1–4, IEEE, dec 2009.

[54] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves, Z. Li, J. P. Strachan, P. Lin, Z. Wang, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, and Q. Xia, "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, vol. 1, pp. 52–59, jan 2018.

[55] U. Chand, K.-C. Huang, C.-Y. Huang, C.-H. Ho, C.-H. Lin, and T.-Y. Tseng, "Investigation of thermal stability and reliability of HfO 2 based resistive random access memory devices with cross-bar structure," *Journal of Applied Physics*, vol. 117, p. 184105, may 2015.

[56] B. Govoreanu, G. Kar, Y.-Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. Wouters, J. Kittl, and M. Jurczak, "10×10nm2 Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *2011 International Electron Devices Meeting*, pp. 31.6.1–31.6.4, IEEE, dec 2011.

[57] J. Song, Y. Zhang, C. Xu, W. Wu, and Z. L. Wang, "Polar Charges Induced Electric Hysteresis of ZnO Nano/Microwire for Fast Data Storage," *Nano Letters*, vol. 11, pp. 2829–2834, jul 2011.

[58] S. G. Hu, Y. Liu, T. P. Chen, Z. Liu, Q. Yu, L. J. Deng, Y. Yin, and S. Hosaka, "Emulating the Ebbinghaus forgetting curve of the human brain with a NiO-based memristor," *Applied Physics Letters*, vol. 103, p. 133701, sep 2013.

[59] S. Porro, A. Jasmin, K. Bejtka, D. Conti, D. Perrone, S. Guastella, C. F. Pirri, A. Chiolerio, and C. Ricciardi, "Low-temperature atomic layer deposition of TiO 2 thin layers for the processing of memristive devices," *Journal of Vacuum Science & Technology A: Vacuum, Surfaces, and Films*, vol. 34, p. 01A147, jan 2016.

[60] T. D. Dongale, S. V. Mohite, A. A. Bagade, P. K. Gaikwad, P. S. Patil, R. K. Kamat, and K. Y. Rajpure, "Development of Ag/WO3/ITO thin film memristor using spray pyrolysis method," *Electronic Materials Letters*, vol. 11, pp. 944–948, nov 2015.

[61] L. Gao, I.-T. Wang, P.-Y. Chen, S. Vrudhula, J.-s. Seo, Y. Cao, T.-H. Hou, and S. Yu, "Fully parallel write/read in resistive synaptic array for accelerating on-chip learning," *Nanotechnology*, vol. 26, p. 455204, nov 2015.

[62] C. La Torre, A. Kindsmüller, D. J. Wouters, C. E. Graves, G. A. Gibson, J. P. Strachan, R. S. Williams, R. Waser, and S. Menzel, "Volatile HRS asymmetry and subloops in resistive switching oxides," *Nanoscale*, vol. 9, no. 38, pp. 14414–14422, 2017.

[63] S. Park, S. Jung, M. Siddik, M. Jo, J. Lee, J. Park, W. Lee, S. Kim, S. M. Sadaf, X. Liu, and H. Hwang, "Memristive switching behavior in Pr0.7Ca0.3MnO3 by incorporating an oxygen-deficient layer," *physica status solidi (RRL) - Rapid Research Letters*, vol. 5, pp. 409–411, nov 2011.

[64] A. Herpers, C. Lenser, C. Park, F. Offi, F. Borgatti, G. Panaccione, S. Menzel, R. Waser, and R. Dittmann, "Spectroscopic Proof of the Correlation between Redox-State and Charge-Carrier Transport at the Interface of Resistively Switching Ti/PCMO Devices," *Advanced Materials*, vol. 26, pp. 2730–2735, may 2014.

[65] Z. Q. Wang, H. Y. Xu, X. H. Li, H. Yu, Y. C. Liu, and X. J. Zhu, "Synaptic Learning and Memory Functions Achieved Using Oxygen Ion Migration/Diffusion in an Amorphous InGaZnO Memristor," *Advanced Functional Materials*, vol. 22, pp. 2759–2765, jul 2012.

[66] J. Park, M. Kwak, K. Moon, J. Woo, D. Lee, and H. Hwang, "TiO x -Based RRAM Synapse With 64-Levels of Conductance and Symmetric Conductance Change by Adopting a Hybrid Pulse Scheme for Neuromorphic Computing," *IEEE Electron Device Letters*, vol. 37, pp. 1559–1562, dec 2016.

[67] H. Young Jeong, S. Kyu Kim, J. Yong Lee, and S.-Y. Choi, "Role of Interface Reaction on Resistive Switching of Metal/Amorphous TiO2/Al RRAM Devices," *Journal of The Electrochemical Society*, vol. 158, no. 10, p. H979, 2011.

[68] M. Zhao, B. Gao, J. Tang, H. Qian, and H. Wu, "Reliability of analog resistive switching memory for neuromorphic computing," *Applied Physics Reviews*, vol. 7, p. 011301, mar 2020.

[69] S. Arrhenius, "Über die dissociationswärme und den einfluss der temperatur auf den dissociationsgrad der elektrolyte," *Zeitschrift für Physikalische Chemie*, vol. 4U, no. 1, pp. 96–116, 1889.

[70] S. Arrhenius, "Über die reaktionsgeschwindigkeit bei der inversion von rohrzucker durch säuren," *Zeitschrift für Physikalische Chemie*, vol. 4U, no. 1, pp. 226–248, 1889.

[71] A. Pirovano, A. Redaelli, F. Pellizzer, F. Ottogalli, M. Tosi, D. Ielmini, A. Lacaita, and R. Bez, "Reliability Study of Phase-Change Nonvolatile Memories," *IEEE Transactions on Device and Materials Reliability*, vol. 4, pp. 422–427, sep 2004.

[72] M. Zhao, H. Wu, B. Gao, Q. Zhang, W. Wu, S. Wang, Y. Xi, D. Wu, N. Deng, S. Yu, H.-Y. Chen, and H. Qian, "Investigation of statistical retention of filamentary analog RRAM for neuromophic computing," in *2017 IEEE International Electron Devices Meeting (IEDM)*, vol. 2017, pp. 39.4.1–39.4.4, IEEE, dec 2017.

[73] B. Chen, Y. Lu, B. Gao, Y. Fu, F. Zhang, P. Huang, Y. Chen, L. Liu, X. Liu, J. Kang, Y. Wang, Z. Fang, H. Yu, X. Li, X. Wang, N. Singh, G. Q. Lo, and D. L. Kwong, "Physical mechanisms of endurance degradation in TMO-RRAM," in *2011 International Electron Devices Meeting*, pp. 12.3.1–12.3.4, IEEE, dec 2011.

[74] S. Subhechha, R. Degraeve, A. Belmonte, L. Goux, G. Luca Donadio, P. Roussel, K. De Meyer, J. Van Houdt, and G. S. Kar, "Understanding Endurance in TiN/a-Si/TiOx/TiN RRAM Devices," in *2018 IEEE International Memory Workshop (IMW)*, pp. 1–4, IEEE, may 2018.

[75] S. Subhechha, R. Degraeve, P. Roussel, L. Goux, S. Clima, K. De Meyer, J. Van Houdt, and G. S. Kar, "Kinetic defect distribution approach for modeling the transient, endurance and retention of a-VMCO RRAM," in *2017 IEEE International Reliability Physics Symposium (IRPS)*, vol. 1, pp. 5A–5.1–5A–5.6, IEEE, apr 2017.

[76] U. Celano, C. Gastaldi, S. Subhechha, B. Govoreanu, G. Donadio, A. Franquet, T. Ahmad, C. Detavernier, O. Richard, H. Bender, L. Goux, G. S. Kar, P. van der Heide, and W. Vandervorst, "Non-filamentary (VMCO) memory: A two-and three-dimensional study on switching and failure modes," in *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 39.1.1–39.1.4, IEEE, dec 2017.

[77] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in *2016 IEEE International Electron Devices Meeting (IEDM)*, pp. 16.2.1–16.2.4, IEEE, dec 2016.

[78] M. Zhao, H. Wu, B. Gao, X. Sun, Y. Liu, P. Yao, Y. Xi, X. Li, Q. Zhang, K. Wang, S. Yu, and H. Qian, "Characterizing Endurance Degradation of Incremental Switching in Analog RRAM for Neuromorphic Systems," in *2018 IEEE International Electron Devices Meeting (IEDM)*, vol. 2018-Decem, pp. 20.2.1–20.2.4, IEEE, dec 2018.

[79] A. Grossi, E. Vianello, M. M. Sabry, M. Barlas, L. Grenouillet, J. Coignus, E. Beigne, T. Wu, B. Q. Le, M. K. Wootters, C. Zambelli, E. Nowak, and S. Mitra, "Resistive RAM Endurance: Array-Level Characterization and Correction Techniques Targeting Deep Learning Applications," *IEEE Transactions on Electron Devices*, vol. 66, pp. 1281–1288, mar 2019.

[80] C. Pan, M. Xie, J. Hu, Y. Chen, and C. Yang, "3M-PCM: Exploiting Multiple Write Modes MLC Phase Change Main Memory in Embedded Systems," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, (New York, NY, USA), pp. 1–10, ACM, oct 2014.

[81] Y. Yamaga, Y. Deguchi, S. Fukuyama, and K. Takeuchi, "5x Reliability Enhanced 40nm TaOx Approximate-ReRAM with Domain-Specific Computing for Real-time Image Recognition of IoT Edge Devices," in *2018 IEEE Symposium on VLSI Technology*, vol. 2018-June, pp. 109–110, IEEE, jun 2018.

[82] P.-Y. Chen, B. Lin, I.-T. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J.-s. Seo, Y. Cao, and S. Yu, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 194–199, IEEE, nov 2015.

[83] S. Kim, S. Choi, and W. Lu, "Comprehensive Physical Model of Dynamic Resistive Switching in an Oxide Memristor," *ACS Nano*, vol. 8, pp. 2369–2376, mar 2014.

[84] S. Larentis, F. Nardi, S. Balatti, D. C. Gilmer, and D. Ielmini, "Resistive Switching by Voltage-Driven Ion Migration in Bipolar RRAM—Part II: Modeling," *IEEE Transactions on Electron Devices*, vol. 59, pp. 2468–2475, sep 2012.

[85] Y. Jeong, S. Kim, and W. D. Lu, "Utilizing multiple state variables to improve the dynamic range of analog switching in a memristor," *Applied Physics Letters*, vol. 107, p. 173105, oct 2015.

[86] Y.-F. Wang, Y.-C. Lin, I.-T. Wang, T.-P. Lin, and T.-H. Hou, "Characterization and Modeling of Nonfilamentary Ta/TaOx/TiO2/Ti Analog Synaptic Device," *Scientific Reports*, vol. 5, p. 10150, sep 2015.

[87] M. Das, A. Kumar, R. Singh, M. T. Htay, and S. Mukherjee, "Realization of synaptic learning and memory functions in Y 2 O 3 based memristive device fabricated by dual ion beam sputtering," *Nanotechnology*, vol. 29, p. 055203, feb 2018.

[88] R. Degraeve, A. Mallik, D. Garbin, J. Doevenspeck, A. Fantini, D. Rodopoulos, P. Roussel, B. Govoreanu, P. Hendrickx, L. Di Piazza, J. Stuijt, S. Schaafsma, P. Debacker, G. Donadio, H. Hody, L. Goux, G. S. Kar, A. Furnemont, A. Mocut, and D. Verkest, "Opportunities and Challenges of Resistive RAM for Neuromorphic Applications," in *2018 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, vol. 2018-July, pp. 1–5, IEEE, jul 2018.

[89] W. Wu, H. Wu, B. Gao, P. Yao, X. Zhang, X. Peng, S. Yu, and H. Qian, "A Methodology to Improve Linearity of Analog RRAM for Neuromorphic Computing," in *2018 IEEE Symposium on VLSI Technology*, vol. 2018-June, pp. 103–104, IEEE, jun 2018.

[90] S. Chandrasekaran, F. M. Simanjuntak, R. Saminathan, D. Panda, and T.-Y. Tseng, "Improving linearity by introducing Al in HfO 2 as a memristor synapse device," *Nanotechnology*, vol. 30, p. 445205, nov 2019.

[91] K. Moon, M. Kwak, J. Park, D. Lee, and H. Hwang, "Improved Conductance Linearity and Conductance Ratio of 1T2R Synapse Device for Neuromorphic Systems," *IEEE Electron Device Letters*, vol. 38, pp. 1023–1026, aug 2017.

[92] I.-T. Wang, C.-C. Chang, L.-W. Chiu, T. Chou, and T.-H. Hou, "3D Ta/TaO x /TiO 2 /Ti synaptic array and linearity tuning of weight update for hardware neural network applications," *Nanotechnology*, vol. 27, p. 365204, sep 2016.

[93] F. Cai, J. M. Correll, S. H. Lee, Y. Lim, V. Bothra, Z. Zhang, M. P. Flynn, and W. D. Lu, "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," *Nature Electronics*, vol. 2, pp. 290–299, jul 2019.

[94] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 6.1.1–6.1.4, IEEE, dec 2017.

[95] S. Yu, Ximeng Guan, and H.-S. P. Wong, "On the stochastic nature of resistive switching in metal oxide RRAM: Physical modeling, monte carlo simulation, and experimental characterization," in *2011 International Electron Devices Meeting*, pp. 17.3.1–17.3.4, IEEE, dec 2011.

[96] M. Gupta, "Thermal noise in nonlinear resistive devices and its circuit representation," *Proceedings of the IEEE*, vol. 70, no. 8, pp. 788–804, 1982.

[97] S. Yu, R. Jeyasingh, Yi Wu, and H.-S. Philip Wong, "Understanding the conduction and switching mechanism of metal oxide RRAM through low frequency noise and AC conductance measurement and analysis," in *2011 International Electron Devices Meeting*, pp. 12.1.1–12.1.4, IEEE, dec 2011.

[98] Z. Fang, H. Y. Yu, W. J. Fan, G. Ghibaudo, J. Buckley, B. DeSalvo, X. Li, X. P. Wang, G. Q. Lo, and D. L. Kwong, "Current Conduction Model for Oxide-Based Resistive Random Access Memory Verified by Low-Frequency Noise Analysis," *IEEE Transactions on Electron Devices*, vol. 60, pp. 1272–1275, mar 2013.

[99] E. Simoen, B. Kaczer, M. Toledano-Luque, and C. Claeys, "Random Telegraph Noise: From a Device Physicist's Dream to a Designer's Nightmare," *ECS Transactions*, vol. 39, pp. 3–15, sep 2011.

[100] D. Veksler, G. Bersuker, L. Vandelli, A. Padovani, L. Larcher, A. Muraviev, B. Chakrabarti, E. Vogel, D. C. Gilmer, and P. D. Kirsch, "Random telegraph noise (RTN) in scaled RRAM devices," in *2013 IEEE International Reliability Physics Symposium (IRPS)*, pp. MY.10.1–MY.10.4, IEEE, apr 2013.

[101] Z. Chai, *Characterisation of Novel Resistive Switching Memory Devices*. Phd thesis, Liverpool John Moores University, 2017.

[102] Z. Chai, J. Ma, W. D. Zhang, B. Govoreanu, J. F. Zhang, Z. Ji, and M. Jurczak, "Probing the Critical Region of Conductive Filament in Nanoscale HfO 2 Resistive-Switching Device by Random Telegraph Signals," *IEEE Transactions on Electron Devices*, vol. 64, pp. 4099–4105, oct 2017.

[103] J. Ma, Z. Chai, W. D. Zhang, J. F. Zhang, Z. Ji, B. Benbakhti, B. Govoreanu, E. Simoen, L. Goux, A. Belmonte, R. Degraeve, G. S. Kar, and M. Jurczak, "Investigation of Preexisting and Generated Defects in Nonfilamentary a-Si/TiO 2 RRAM and Their Impacts on RTN Amplitude Distribution," *IEEE Transactions on Electron Devices*, vol. 65, pp. 970–977, mar 2018.

[104] Z. Chai, P. Freitas, W. Zhang, F. Hatem, J. F. Zhang, J. Marsland, B. Govoreanu, L. Goux, and G. S. Kar, "Impact of RTN on Pattern Recognition Accuracy of RRAM-Based Synaptic Neural Network," *IEEE Electron Device Letters*, vol. 39, pp. 1652–1655, nov 2018.

[105] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "DNN+NeuroSim V2.0: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators for On-chip Training," mar 2020.

[106] S. Choi, S. H. Tan, Z. Li, Y. Kim, C. Choi, P.-Y. Chen, H. Yeon, S. Yu, and J. Kim, "SiGe epitaxial memory for neuromorphic computing with reproducible high performance based on engineered dislocations," *Nature Materials*, vol. 17, pp. 335–340, apr 2018.

[107] L. Zhang, *Study of the Selector Element for Resistive Memory*. Phd thesis, KU Leuven, 2015.

[108] F. Hatem, J. F. Zhang, J. Marsland, P. Freitas, L. Goux, G. S. Kar, Z. Chai, W. Zhang, A. Fantini, R. Degraeve, S. Clima, D. Garbin, J. Robertson, and Y. Guo, "Endurance improvement of more than five orders in Ge x Se 1-x OTS selectors by using a novel refreshing program scheme," in *2019 IEEE International Electron Devices Meeting (IEDM)*, no. 1, pp. 35.2.1–35.2.4, IEEE, dec 2019.

[109] Z. Chai, W. Zhang, R. Degraeve, S. Clima, F. Hatem, J. F. Zhang, P. Freitas, J. Marsland, A. Fantini, D. Garbin, L. Goux, and G. S. Kar, "Evidence of filamentary switching and relaxation mechanisms in Ge x Se 1-x OTS selectors," in *2019 Symposium on VLSI Technology*, no. 1, (Kyoto), pp. T238–T239, IEEE, jun 2019.

[110] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1st ed., 1997.

[111] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, nov 2012.

[112] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *Advances in Neural Information Processing Systems*, vol. 4, pp. 3104–3112, sep 2014.

[113] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, sep 2014.

[114] R. Collobert, "Deep learning for efficient discriminative parsing," in *Proceedings of the 14th International Con- ference on Artificial Intelligence and Statistics (AISTATS)*, vol. 15, (Fort Lauderdale, FL, USA), pp. 224–232, 2011.

[115] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection," *ACM Computing Surveys*, vol. 41, pp. 1–58, jul 2009.

[116] H. Luo, P. L. Carrier, A. Courville, and Y. Bengio, "Texture Modeling with Convolutional Spike-and-Slab RBMs and Deep Extensions," *Journal of Machine Learning Research*, vol. 31, pp. 415–423, nov 2012.

[117] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, pp. 27–38, jan 2009.

[118] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, pp. 37–46, apr 1960.

[119] I. H. Witten and E. Frank, "Data mining: practical machine learning tools and techniques with java implementations," *Acm Sigmod Record*, vol. 31, no. 1, pp. 76–77, 2002.

[120] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. EBL-Schweitzer, Wiley, 2014.

[121] G. Lebanon and J. D. Lafferty, "Cranking: Combining Rankings Using Conditional Probability Models on Permutations," in *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, (San Francisco, CA, USA), pp. 363–370, Morgan Kaufmann Publishers Inc., 2002.

[122] G. W. BRIER, "VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY," *Monthly Weather Review*, vol. 78, pp. 1–3, jan 1950.

[123] I. J. Good, "Rational Decisions," in *Breakthroughs in Statistics.*, pp. 365–377, New York, NY, USA: Springer, 1992.

[124] I. J. Good, "Corroboration, Explanation, Evolving Probability, Simplicity and a Sharpened Razor," *The British Journal for the Philosophy of Science*, vol. 19, no. 2, pp. 123–143, 1968.

[125] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, vol. 29. MIT Press, 2016.

[126] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[127] M. A. Cauchy, "Méthode générale pour la résolution des systèmes d'équations simultanées," *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*, vol. 25, no. 2, pp. 536–538, 1847.

[128] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, pp. 400–407, sep 1951.

[129] B. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, pp. 1–17, jan 1964.

[130] T. P. Xiao, C. H. Bennett, B. Feinberg, S. Agarwal, and M. J. Marinella, "Analog architectures for neural network acceleration based on non-volatile memory," *Applied Physics Reviews*, vol. 7, p. 031301, sep 2020.

[131] F. Rosenblatt, "The Perceptron - A Perceiving and Recognizing Automaton," tech. rep., 1957.

[132] F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms," tech. rep., Cornell Aeronautical Lab Inc, Buffalo, NY, dec 1961.

[133] J. Zhu, T. Zhang, Y. Yang, and R. Huang, "A comprehensive review on emerging artificial neuromorphic devices," *Applied Physics Reviews*, vol. 7, p. 011312, mar 2020.

[134] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, oct 1986.

[135] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pp. 1–22, dec 2013.

[136] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *Advances in Neural Information Processing Systems*, vol. 4, pp. 2933–2941, jun 2014.

[137] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, dec 2014.

[138] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," in *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 38, (San Diego, CA, USA), 2015.

[139] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Journalism Practice*, vol. 10, pp. 730–743, feb 2015.

[140] X. Glorot, Y. Bengio, Z. Liu, H. Wang, L. Weng, and Y. Yang, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*

(Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 2010.

[141] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, vol. 2015 Inter, pp. 1026–1034, IEEE, dec 2015.

[142] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, jan 1988.

[143] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[144] T. Tieleman, G. Hinton, and Others, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[145] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, dec 2014.

[146] Y. LeCun, "Generalization and network design strategies," tech. rep., University of Toronto, 1989.

[147] Zhou and Chellappa, "Computation of optical flow using a neural network," in *IEEE International Conference on Neural Networks*, no. 86, pp. 71–78 vol.2, IEEE, 1988.

[148] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM journal on computing*, vol. 2, no. 2, pp. 88–105, 1973.

[149] A. E. Eiben, P. E. Raué, and Z. Ruttkay, "Genetic algorithms with multi-parent recombination," No. June, pp. 78–87, 1994.

[150] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," *IEEE Transactions on Neural Networks*, vol. 1, pp. 71–80, feb 2015.

[151] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," in *Low-Power Computer Vision*, pp. 291–326, Boca Raton: Chapman and Hall/CRC, jan 2022.

[152] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," *Advances in Neural Information Processing Systems*, vol. 2, no. January, pp. 963–971, 2014.

[153] Z. Cheng, D. Soudry, Z. Mao, and Z. Lan, "Training Binary Multilayer Neural Networks for Image Classification using Expectation Backpropagation," pp. 1–8, mar 2015.

[154] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *NIPS*, p. 10, nov 2015.

[155] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," vol. 1, pp. 1–13, jun 2016.

[156] G. E. Hinton, "Neural networks for machine learning.," 2012.

[157] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *Journal of Machine Learning Research*, vol. 18, pp. 1–30, sep 2016.

[158] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training Deep Neural Networks with 8-bit Floating Point Numbers," *Advances in Neural Information Processing Systems*, vol. 2018-Decem, pp. 7675–7684, dec 2018.

[159] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, IEEE, jun 2018.

[160] A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin, "Training with Quantization Noise for Extreme Model Compression," 2020.

[161] E. Stromatias and J. S. Marsland, "Supervised learning in Spiking Neural Networks with limited precision: SNN/LP," in *2015 International Joint Conference on Neural Networks (IJCNN)*, vol. 1407.0265, pp. 1–7, IEEE, jul 2015.

[162] S. Dhar, J. Guo, J. J. Liu, S. Tripathi, U. Kurup, and M. Shah, "A Survey of On-Device Machine Learning: An Algorithms and Learning Theory Perspective," *ACM Transactions on Internet of Things*, vol. 2, pp. 1–49, aug 2021.

[163] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," *Advances in neural information processing systems*, vol. 22, 2009.

[164] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," pp. 1–18, jul 2012.

[165] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural Networks with Few Multiplications," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–9, oct 2015.

[166] M. Kim and P. Smaragdis, "Bitwise Neural Networks," vol. 37, jan 2016.

[167] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9908 LNCS, pp. 525–542, 2016.

[168] E. van den Berg, B. Ramabhadran, and M. Picheny, "Training variance and performance evaluation of neural networks in speech," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2287–2291, IEEE, mar 2017.

[169] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.

[170] Y. Yang, L. Deng, S. Wu, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural Networks*, vol. 125, pp. 70–82, may 2020.

[171] L. Cambier, A. Bhiwandiwalla, T. Gong, M. Nekuii, O. H. Elibol, and H. Tang, "Shifted and Squeezed 8-bit Floating Point format for Low-Precision Training of Deep Neural Networks," vol. 8, pp. 1–12, 2020.

[172] S. Wiedemann, T. Mehari, K. Kepp, and W. Samek, "Dithered backprop: A sparse and quantized backpropagation algorithm for more efficient deep neural network training," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, vol. 2020-June, pp. 3096–3104, IEEE, jun 2020.

[173] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, (New York, New York, USA), pp. 1–8, ACM Press, 2009.

[174] A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, B. Catanzaro, and N. Andrew, "Deep learning with COTS HPC systems," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1337–1345, PMLR, 2013.

[175] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, R. S. Williams, J. Yang, and H. P. Labs, "Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication," *IEEE Design Automation Conference*, pp. 1—-6, 2016.

[176] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–39, IEEE, jun 2016.

[177] G. W. Burr, R. M. Shelby, S. Sidler, C. di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi, and H. Hwang, "Experimental Demonstration and Tolerancing of a Large-Scale Neural Network (165 000 Synapses) Using Phase-Change Memory as the Synaptic Weight Element," *IEEE Transactions on Electron Devices*, vol. 62, pp. 3498–3507, nov 2015.

[178] M. Bavandpour, M. R. Mahmoodi, and D. B. Strukov, "Energy-Efficient Time-Domain Vector-by-Matrix Multiplier for Neurocomputing and Beyond," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, pp. 1512–1516, sep 2019.

[179] M. J. Marinella, S. Agarwal, A. Hsia, I. Richter, R. Jacobs-Gedrim, J. Niroula, S. J. Plimpton, E. Ipek, and C. D. James, "Multiscale Co-Design Analysis of Energy, Latency, Area, and Accuracy of a ReRAM Analog Neural Training Accelerator," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, pp. 86–101, mar 2018.

[180] M. N. Bojnordi and E. Ipek, "Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *2017 Fifth Berkeley Symposium on Energy Efficient Electronic Systems & Steep Transistors Workshop (E3S)*, vol. 2018-Janua, pp. 1–3, IEEE, oct 2017.

[181] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 14–26, IEEE, jun 2016.

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

[183] E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, "Repeatable, accurate, and high speed multi-level programming of memristor 1T1R arrays for power efficient analog computing applications," *Nanotechnology*, vol. 27, p. 365202, sep 2016.

[184] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, "Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine," *Advanced Materials*, vol. 30, p. 1705914, mar 2018.

[185] M. R. Mahmoodi and D. Strukov, "An ultra-low energy internally analog, externally digital vector-matrix multiplier based on NOR flash memory technology," in *Proceedings of the 55th Annual Design Automation Conference*, vol. Part F1377, (New York, NY, USA), pp. 1–6, ACM, jun 2018.

[186] S. N. Truong and K.-S. Min, "New Memristor-Based Crossbar Array Architecture with 50-% Area Reduction and 48-% Power Saving for Matrix-Vector Multiplication of Analog Neuromorphic Computing," *JSTS:Journal of Semiconductor Technology and Science*, vol. 14, pp. 356–363, jun 2014.

[187] Y. Zhang, X. Wang, and E. G. Friedman, "Memristor-Based Circuit Design for Multilayer Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, pp. 677–686, feb 2018.

[188] P. Narayanan, A. Fumarola, L. L. Sanches, K. Hosokawa, S. C. Lewis, R. M. Shelby, and G. W. Burr, "Toward on-chip acceleration of the backpropagation algorithm using nonvolatile memory," *IBM Journal of Research and Development*, vol. 61, no. 4, pp. 1–11, 2017.

[189] A. Fumarola, P. Narayanan, L. L. Sanches, S. Sidler, J. Jang, K. Moon, R. M. Shelby, H. Hwang, and G. W. Burr, "Accelerating machine learning with Non-Volatile Memory: Exploring device and circuit tradeoffs," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, IEEE, oct 2016.

[190] S. Agarwal, T.-T. Quach, O. Parekh, A. H. Hsia, E. P. DeBenedictis, C. D. James, M. J. Marinella, and J. B. Aimone, "Energy Scaling Advantages of Resistive Memory Crossbar Based Computation and Its Application to Sparse Coding," *Frontiers in Neuroscience*, vol. 9, pp. 1–9, jan 2016.

[191] E. Rosenthal, S. Greshnikov, D. Soudry, and S. Kvatinsky, "A fully analog memristor-based neural network with online gradient training," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2016-July, pp. 1394–1397, IEEE, may 2016.

[192] D. Kadetotad, Z. Xu, A. Mohanty, P.-Y. Chen, B. Lin, J. Ye, S. Vrudhula, S. Yu, Y. Cao, and J.-s. Seo, "Parallel Architecture With Resistive Crosspoint Array for Dictionary Learning Acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, pp. 194–204, jun 2015.

[193] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 541–552, IEEE, feb 2017.

[194] C. Lammie, W. Xiang, and M. Rahimi Azghadi, "Modeling and simulating in-memory memristive deep learning systems: An overview of current efforts," *Array*, vol. 13, p. 100116, mar 2022.

[195] M. Imani, M. Samragh Razlighi, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Deep Learning Acceleration with Neuron-to-Memory Transformation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–14, IEEE, feb 2020.

[196] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference," *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 715–731, jan 2019.

[197] M.-Y. Lin, H.-Y. Cheng, W.-T. Lin, T.-H. Yang, I.-C. Tseng, C.-L. Yang, H.-W. Hu, H.-S. Chang, H.-P. Li, and M.-F. Chang, "DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning," in *Proceedings of the International Conference on Computer-Aided Design*, (New York, NY, USA), pp. 1–8, ACM, nov 2018.

[198] X. Ma, G. Yuan, S. Lin, C. Ding, F. Yu, T. Liu, W. Wen, X. Chen, and Y. Wang, "Tiny but Accurate: A Pruned, Quantized and Optimized Memristor Crossbar Framework for Ultra Efficient DNN Implementation," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, vol. 2020-Janua, pp. 301–306, IEEE, jan 2020.

[199] G. Yuan, X. Ma, C. Ding, S. Lin, T. Zhang, Z. S. Jalali, Y. Zhao, L. Jiang, S. Soundarajan, and Y. Wang, "An Ultra-Efficient Memristor-Based DNN Framework with Structured Weight Pruning and Quantization Using ADMM," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, vol. 2019-July, pp. 1–6, IEEE, jul 2019.

[200] C. Lammie and M. R. Azghadi, "MemTorch: A Simulation Framework for Deep Memristive Cross-Bar Architectures," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2020-Octob, pp. 1–5, IEEE, oct 2020.

[201] C. Lammie, W. Xiang, B. Linares-Barranco, and M. Rahimi Azghadi, "MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems," *Neurocomputing*, vol. 485, pp. 124–133, may 2022.

[202] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators with Versatile Device Technologies," in *2019 IEEE International Electron Devices Meeting (IEDM)*, vol. 2019-Decem, pp. 32.5.1–32.5.4, IEEE, dec 2019.

[203] A. Lu, X. Peng, W. Li, H. Jiang, and S. Yu, "NeuroSim Simulator for Compute-in-Memory Hardware Accelerator: Validation and Benchmark," *Frontiers in Artificial Intelligence*, vol. 4, no. June, pp. 1–10, 2021.

[204] M. J. Rasch, D. Moreda, T. Gokmen, M. Le Gallo, F. Carta, C. Goldberg, K. El Maghraoui, A. Sebastian, and V. Narayanan, "A Flexible and Fast PyTorch Toolkit for Simulating Training and Inference on Analog Crossbar Arrays," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–4, IEEE, jun 2021.

[205] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Non-volatile Memory," in *Emerging Memory Technologies*, vol. 9781441995, pp. 15–50, New York, NY: Springer New York, 2014.

[206] Y.-S. Fan, L. Zhang, D. Crotti, T. Witters, M. Jurczak, and B. Govoreanu, "Direct Evidence of the Overshoot Suppression in Ta¡sub¿2¡/sub¿O¡sub¿5¡/sub¿-Based Resistive Switching Memory With an Integrated Access Resistor," *IEEE Electron Device Letters*, vol. 36, pp. 1027–1029, oct 2015.

[207] L. Goux, A. Fantini, R. Degraeve, N. Raghavan, R. Nigon, S. Strangio, G. Kar, D. J. Wouters, Y. Y. Chen, M. Komura, F. De Stefano, V. V. Afanas'Ev, and M. Jurczak, "Understanding of the intrinsic characteristics and memory trade-offs of sub-μA filamentary RRAM operation," *Digest of Technical Papers - Symposium on VLSI Technology*, vol. 88, no. 2011, pp. 2012–2013, 2013.

[208] M. E. Pereira, J. Deuermeier, P. Freitas, P. Barquinha, W. Zhang, R. Martins, E. Fortunato, and A. Kiazadeh, "Tailoring the synaptic properties of a-IGZO memristors for artificial deep neural networks," *APL Materials*, vol. 10, p. 011113, jan 2022.

[209] I. The MathWorks, *Curve Fitting Toolbox*. Natick, Massachusetts, United State, 2022.

[210] MATLAB, *List of library models for curve and surface fitting*, 2022. [Online]. Available: https://www.mathworks.com/help/curvefit/list-of-library-models-for-curve-and-surface-fitting.html. [Accessed: 30-Jan-2023].

[211] MATLAB, *Create datastore for large collections of data*, 2022. [Online]. Available: https://www.mathworks.com/help/matlab/ref/datastore.html [Accessed: 31-Jan-2023].

[212] MATLAB, *layerGraph - Graph of network layers for deep learning*, 2022. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layergraph.html [Accessed: 31-Jan-2023].

[213] I. The MathWorks, *Deep Learning Toolbox*. Natick, Massachusetts, United State, 2022.

[214] MATLAB, *dlnetwork - Deep learning network for custom training loops*, 2022. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/dlnetwork.html [Accessed: 31-Jan-2023].

[215] MATLAB, *trainingOptions - Options for training deep learning neural network*, 2022. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html [Accessed: 31-Jan-2023].

[216] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," *International Journal of Computer Vision*, vol. 128, pp. 336–359, oct 2016.

[217] MATLAB, *gradCAM - Explain network predictions using Grad-CAM*, 2022. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/gradcam.html [Accessed: 01-Feb-2023].

[218] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–Oxide RRAM," *Proceedings of the IEEE*, vol. 100, pp. 1951–1970, jun 2012.

[219] D. Ielmini, "Brain-inspired computing with resistive switching memory (RRAM): Devices, synapses and neural networks," *Microelectronic Engineering*, vol. 190, pp. 44–53, 2018.

[220] M. Kirton and M. Uren, "Noise in solid-state microstructures: A new perspective on individual defects, interface states and low-frequency (1/ f) noise," *Advances in Physics*, vol. 38, pp. 367–468, jan 1989.

[221] F. M. Puglisi, N. Zagni, L. Larcher, and P. Pavan, "A new verilog-A compact model of random telegraph noise in oxide-based RRAM for advanced circuit design," in *2017 47th European Solid-State Device Research Conference (ESSDERC)*, pp. 204–207, IEEE, sep 2017.

[222] S. Balatti, S. Ambrogio, A. Cubeta, A. Calderoni, N. Ramaswamy, and D. Ielmini, "Voltage-dependent random telegraph noise (RTN) in HfOx resistive RAM," in *2014 IEEE International Reliability Physics Symposium*, pp. MY.4.1–MY.4.6, IEEE, jun 2014.

[223] MATLAB, *Lognormal Distribution*, 2023. [Online]. Available: https://www.mathworks.com/help/stats/lognormal-distribution.html [Accessed: 12-Sep-2023].

[224] M. Nielsen, *Neural Networks and Deep Learning*. 2015.

[225] J. Kang, Z. Yu, L. Wu, Y. Fang, Z. Wang, Y. Cai, Z. Ji, J. Zhang, R. Wang, Y. Yang, and R. Huang, "Time-dependent variability in RRAM-based analog neuromorphic system for pattern recognition," in *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 6.4.1–6.4.4, IEEE, dec 2017.

[226] S. Choi, Y. Yang, and W. Lu, "Random telegraph noise and resistance switching analysis of oxide based resistive memory," *Nanoscale*, vol. 6, no. 1, pp. 400–404, 2014.

[227] F. M. Puglisi, L. Larcher, A. Padovani, and P. Pavan, "A Complete Statistical Investigation of RTN in HfO2-Based RRAM in High Resistive State," *IEEE Transactions on Electron Devices*, vol. 62, pp. 2606–2613, aug 2015.

[228] A. Fantini, L. Goux, R. Degraeve, D. Wouters, N. Raghavan, G. Kar, A. Belmonte, Y.-Y. Chen, B. Govoreanu, and M. Jurczak, "Intrinsic switching variability in HfO2 RRAM," in *2013 5th IEEE International Memory Workshop*, no. June 2016, pp. 30–33, IEEE, may 2013.

[229] A. Fantini, G. Gorine, R. Degraeve, L. Goux, C. Y. Chen, A. Redolfi, S. Clima, A. Cabrini, G. Torelli, and M. Jurczak, "Intrinsic program instability in HfO2 RRAM and consequences on program algorithms," in *Technical Digest - International Electron Devices Meeting, IEDM*, vol. 2016-Febru, pp. 7.5.1–7.5.4, IEEE, dec 2015.

[230] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "RRAM-Based Analog Approximate Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1905–1917, 2015.

[231] MATLAB, *Weibull Distribution*, 2023. [Online]. Available: https://www.mathworks.com/help/stats/weibull-distribution.html [Accessed: 12-Sep-2023].

[232] S. Kim, M. Lim, Y. Kim, H.-D. Kim, and S.-J. Choi, "Impact of Synaptic Device Variations on Pattern Recognition Accuracy in a Hardware Neural Network," *Scientific Reports*, vol. 8, p. 2638, feb 2018.

[233] S. Agarwal, R. B. Jacobs Gedrim, A. H. Hsia, D. R. Hughart, E. J. Fuller, A. A. Talin, C. D. James, S. J. Plimpton, M. J. Marinella, R. B. Gedrim, A. H. Hsia, D. R. Hughart, E. J. Fuller, A. A. Talin, C. D. James, S. J. Plimpton, and M. J. Marinella, "Achieving ideal accuracies in analog neuromorphic computing using periodic carry," in *2017 Symposium on VLSI Technology*, pp. T174–T175, IEEE, jun 2017.

[234] S. N. Truong, K. V. Pham, W. Yang, S. Shin, K. Pedrotti, and K.-S. Min, "New pulse amplitude modulation for fine tuning of memristor synapses," *Microelectronics Journal*, vol. 55, pp. 162–168, sep 2016.

[235] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, Y. Jaoudi, and G. W. Burr, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, pp. 60–67, jun 2018.

[236] S. R. Nandakumar, M. Le Gallo, C. Piveteau, V. Joshi, G. Mariani, I. Boybat, G. Karunaratne, R. Khaddam-Aljameh, U. Egger, A. Petropoulos, T. Antonakopoulos, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Mixed-Precision Deep Learning Based on Computational Memory," *Frontiers in Neuroscience*, vol. 14, may 2020.