Learning rules for CMOS-memristive co-integrated neuromorphic hardware

Von der Fakultät für Elektrotechnik und Informationstechnik der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation vorgelegt von

Sebastian Siegel, M.Sc.

aus Bonn

Berichter*innen: Universitätsprofessorin Dr. Regina Dittmann
Universitätsprofessor Dr. Emre Neftci

Tag der mündlichen Prüfung: 14. September 2023

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Curriculum Vitae Sebastian Siegel

14.09.2023	Examination for Dr.-Ing.
2019 - 2023	Research Assistant and Doctoral Candidate Peter-Grünberg-Institute (PGI-10) Forschungszentrum Jülich GmbH & RWTH Aachen University
03.2019	Master of Science
02.2017 - 07.2017	Exchange Semester at Korea Advanced Institute of Science and Technology (KAIST) Daejeon, Republic Korea
09.2016	Bachelor of Science
2013 - 2019	Studies of Electrical Engineering, Information Technology and Computer Engineering
07.2013	General Higher Education Entrance Qualification (Abitur) Carl-von-Ossientzky Gymnasium, Bonn
1993	born in Bonn, Germany

Acknowledgements

This work would not have been possible without the support of numerous people to which I would like to express my deep gratitude.

First, I want to thank Prof. Dr. Regina Dittmann accepting me as a PhD student and supervising my work. I am especially grateful for freedom in exploring my topic and the the invaluable support no matter of the direction I decided to take. Furthermore, I want to thank Prof. Dr. Emre Neftci for being my second examiner and supporting me with valuable discussion on machine learning with memristive devices.

My deep gratitude goes to all professors, PIs, colleagues, and staff at PGI-7/10 and PGI-15 of Forschungszentrum Jülich and IWE2 of RWTH Aachen University who made my PhD time an unforgettable experience. I especially wish to thank those I had the honor of working with, among which some deserve separate mentioning: I want to thank Tobias Ziegler for the collaborative effort of designing the MAD200 memristive demonstrator chip and his support with the electrical measurements on this device. Furthermore, the work on neuromorphic sequence learning would have been impossible without Dr. Younes Bouhadjar, with whom I collaborated on the memristive implementation of the Spiking Temporal Memory and shared trips to the nanoge MatNeC22 conference and the Capo Caccia workshop. Last but not least, I want to thank Dr. Alexander Gutsche for great collaborations on PCMO synapses and STO switching kinetics and in general support with any kind of fabrication related questions.

I also want to thank the administrative and technical staff specifically at PGI-7 and ZEA-2 for supporting me and taking so much time to help me with any kind of issues.

There are far more persons who helped me along the way that led finally to this thesis, but I cannot name all. But I am grateful nevertheless.

I. Abstract

Today, applications of artificial intelligence based on machine learning algorithms pervade various aspects of human society. The training and inference of the large artificial neural networks (ANN) is executed in server farms on conventional von-Neumann computers. This computer architecture renders unfit for the highly parallelizable workloads of machine learning and neuromorphic applications, leading to the consumption of vast amounts of energy during training and inference. Furthermore, the data evaluation far away from the actual user that could be consumer smartphone or a remote IoT application raises concerns in sectors like medical, industrial, and military applications where data privacy is crucial. Potentially a solution to both issues, innovative computer architectures and novel electronic device concepts hold the promise of local and energy-efficient training and application of artificial intelligence algorithms. Prominent among these concepts, In-Memory-Compute (IMC) computer architectures based on memristive switching devices can overcome the limitations of the von-Neumann architecture by co-localizing data storage and computation in an energy-efficient manner while saving silicon real estate. Memristive devices are an emerging non-volatile memory consisting of nanoscale resistors with an electronically adjustable resistance.

However, to fully leverage the application of memristive devices in IMC architectures for machine learning and neuromorphic applications, it is necessary to develop dedicated learning rules for these hybrid systems that mitigate potential shortfalls of the memristive switching dynamics or even exploit these for the implementation of efficient algorithms. Therefore, in this thesis on the one hand, different methods for the compensation of the effect of memristive switching dynamics on the training of ANNs are investigated. An approach is developed that allows for the same training performance on a classical benchmark task as optimal floating-point weights. On the other hand, the integration of memristive devices with conventional CMOS-circuitry is investigated for neuromorphic applications. Therefore, the famous Hierarchical Temporal Memory algorithm is adapted for an application on a memristive crossbar array. This approach conserves the crucial context-sensitivity of this algorithm in a sequence learning application and renders energy-efficient in comparison to conventional storage technology. Lastly, the algorithm is validated on real hardware on a hybrid memristive CMOS-chip that is developed in the course of this thesis.

In this thesis a crucial issue for the integration of memristive devices into machine learning accelerators is tackled: the loss of training performance due to maleficent memristive switching dynamics. It is furthermore showcased how algorithm-hardware co-design can adapt neuromorphic algorithms for the use with energy-efficient CMOS-memristive hardware.

II. Kurzfassung

Motivation, Ziele und Aufgabenstellung der Arbeit

Die Entwicklung und Integration von künstlichen intelligenten (KI) Systemen durchdringt heute viele Bereiche des menschlichen Lebens und verspricht dabei neue Anwendungsmöglichkeiten vom Automobil- und Unterhaltungssektor bis hin zur Medizin- und Kommunikationstechnik. Diese modernen KI-Systeme basieren auf Prinzipien des maschinellen Lernens (ML) und des neuromorphen Rechnens. Dazu werden künstliche neuronale Netze (KNN) mit teilweise mehreren Milliarden Parametern in Großrechenanlagen trainiert und dort später mit durch den Nutzer eingegebene Daten angewendet. Diese Berechnungen beruhen auf simplen mathematischen Operationen, die hochgradig parallelisiert werden können. Durch die strikte Trennung von Speicher und Recheneinheit in klassischen von-Neumann-Rechnerarchitekturen kann dies jedoch nicht optimal ausgenutzt werden, was zu einem hohen Energieverbrauch und einer hohen Latenz beim Training und der Nutzung dieser Netze führt.

Um dem entgegenzuwirken, wird an neuen Rechnerorganisationen gearbeitet, die es erlauben, Berechnungen direkt im Datenspeicher durchzuführen („In-memory-Computing“(IMC)) ohne dass dafür die Daten bewegt werden müssen. Als eine mögliche Hardwareplattform für IMC-Architekturen werden memristive Schaltzellen und Matrixstrukturen aus diesen erforscht, sogenannte Crossbar-Arrays. Ein IMC-System basierend auf memristiven Schaltzellen verspricht eine hohe Integrationsdichte und geringe elektrische Leistungsaufnahme, was es ermöglichen würde, KNN-Anwendungen direkt beim Anwender zu integrieren. Dies ist besonders relevant für mobile Anwendungen im Consumer und Internet-of-Things (IoT) Bereich, wobei dieser Ansatz eine hohe Energieeffizienz und durch die lokale Datenverarbeitung einen besseren Datenschutz verspricht. Die Vorteile einer memristive IMC-Architektur können nur vollständig ausgenutzt werden, wenn das Schaltverhalten der eingesetzten memristiven Schaltzellen bei ihrem Design mitberücksichtigt wird. Daher ist das Ziel dieser Arbeit die Entwicklung und Untersuchung von Trainingsmethoden und Lernregeln, die speziell auf memristive Hardwareplattformen zugeschnitten sind. Diese Lernregeln beziehen sich sowohl auf den Bereich des maschinellen Lernens als auch des neuromorphen Rechnens. Dabei soll sowohl der negative Einfluss der Schaltdynamik memristiver Zellen abgewendet als auch deren nicht-flüchtige und graduelle Natur ausgenutzt werden.

Auf Basis von Schaltungs- und Netzwerksimulationen wird zum einen das Training vom memristiven KNNs mit Mitteln des maschinellen Lernens untersucht und optimiert, als auch ein aus den Neurowissenschaften inspirierter neuromorpher Algorithmus adaptiert, um effizient auf einer memristiven Hardwareplattform ausgeführt zu werden. Zudem wird ein Demonstratorchip mit CMOS-memristiven Schaltungen entworfen, um die in Simulationen entwickelten Konzepte validieren zu können.

Wesentliche wissenschaftliche Beiträge der Arbeit

Für den Erfolg des Trainings eines KNNs mit Methoden des maschinellen Lernens ist die Initialisierung des Netzes von entscheidender Bedeutung. Hierzu existieren etliche Studien, die genaue Parameter für die initiale Gewichtsverteilung vorschlagen. Da die Leitfähigkeit von memristive Schaltzellen bzw. Gewichten aber nur bestimmte Werte annehmen und mit begrenzter Genauigkeit geschrieben werden kann, können diese Methoden nicht direkt angewendet werden. In dieser Arbeit wurden daher Methoden entwickelt, wie die Aktivierungsfunktionen eines KNNs genutzt werden können, um die gleichen positiven Effekte auf den Lernerfolg mit beinahe beliebigen memristiven Initialisierungen zu erzielen. Als ein weiteres Problem können besonders die Nicht-Linearität und Asymmetrie der memristiven Schaltdynamik die Gewichtsgradienten beim Training mit dem Gradientenabstiegsverfahren derart beeinflussen, dass ein hoher Verlust von Genauigkeit im Vergleich zu optimalen Gewichten in einem digitalen Computer entsteht. Um dies zu vermeiden, wurden in dieser Arbeit Methoden untersucht, die die negativen Effekte des memristiven Schaltverhaltens kompensieren. Besonders erfolgreich gelang dies mit einem Ansatz für das Update der Gewichte beim Training. Dieser gleicht besonders die Asymmetrie aus und sorgt so dafür, dass für ein klassisches Benchmarkproblem des maschinellen Lernens die gleiche Genauigkeit mit memristiven Gewichten wie mit idealen Gewichten erreicht werden konnte.

Beide Konzepte, die Kompensation des memristiven Leitfähigkeitsbereichs und des Schaltverhaltens sind wichtige Schritte hin zu CMOS-memristiven hybriden Systemen, die mit Methoden des maschinellen Lernens trainiert werden können.

Des Weiteren wurde der Einsatz memristiver Schaltelemente, insbesondere von matrixartigen Strukturen dieser Zellen, in neuromorphen Systemen untersucht. Dazu wurde eine Version des Temporalspeichers des Hierarchischen Temporalspeicherkonzepts, die mit biologischen Elementen und Lernregeln umgesetzt wurde, für den Einsatz auf einer memristiven Speichermatrix adaptiert. Unter strikter Einhaltung der Prinzipien des IMC konnte so ein neuromorphes Hardwaresystem entworfen werden, das kontinuierlich Sequenzen lernen kann und dabei deren kontextuelle Einbettung beachtet. Das so entworfene System ist die erste Hardwareumsetzung der biologischen Variante des Temporalspeichers und zeigt durch den Einsatz memristiver Schaltzellen einen geringeren Energieverbrauch als mit herkömmlichen Speichertechnologien.

Um die Funktionalität der in beiden Bereichen entwickelten Konzepte in Hardware validieren zu können, wurde weiterhin ein Demonstratorchip mit mehreren memristiven Matrixstrukturen entworfen. Ein Schaltungskonzept wurde entwickelt, das den flexiblen Einsatz des Chips für die Untersuchung verschiedenster Anwendungen memristiver Matrixstrukturen, vom KNN bis hin zum inhaltsadressierbaren Speicher, erlaubt. In seiner Flexibilität ist der Chip damit einzigartig. Mit ihm gelang unter anderem die erfolgreiche Validierung des entwickelten neuromorphen Algorithmus mit einem komplexen Problem aus dem Bereich des Lernens von Sequenzen. Solche kompletten Validierungen auf realer memristiver Hardware sind bisher noch selten zu finden, aber für die Entwicklung von CMOS-memristiven hybriden Systemen für neuromorphen Algorithmen von entscheidender Bedeutung.

Contents

I. Abstract . . . . . f
II. Kurzfassung . . . . . g

1. Introduction **1**

2. Fundamentals **4**
2.1. Machine learning toward artificial intelligence 4
2.1.1. Overview of artificial neural networks 4
2.1.2. Detailed view on Multilayer Perceptrons 5
2.1.3. Training with Gradient Descent and Backpropagation 6
2.1.4. Initialization of weight layers 9
2.2. Neuromorphic computing 10
2.3. Memristive devices 13
2.3.1. Redox-based resistive random access memory - Valence-change mechanism 14
2.3.2. Forming 16
2.3.3. Filamentary and area-type switching 17
2.3.4. Investigated resistive switching devices 17
2.4. Memristive devices for machine learning 19
2.4.1. Memristive weights for artificial neural networks 19
2.4.2. Co-integration of memristive devices with CMOS technology . . 23
2.4.3. Memristive neuromorphic hardware 24
2.4.4. Memristive synapses 24

3. Experimental **29**
3.1. Electrical measurements 29
3.1.1. Quasi-static current voltage measurements 29
3.1.2. Potentiation and depression measurements 31
3.2. Software frameworks for artificial neural network simulations 33
3.2.1. PyTorch 34
3.2.2. TensorFlow 35

4. A memristive Compute-in-Memory demonstrator for neural network applications **37**
4.1. Concept and circuitry 38
4.1.1. Active crossbar arrays 42
4.1.2. Routing logic and address circuitry 43

4.2. Device characterizations . . . 47
4.2.1. Quasi-static switching . . . 47
4.2.2. Potentiation & Depression Measurements . . . 49
4.3. Array operations . . . 55
4.3.1. Multiply and Accumulate (MAC) . . . 55
4.3.2. Parallel write . . . 58
4.4. Summary . . . 64

5. Memristive devices as weights for Machine Learning 65
5.1. Initialization of memristive weight layers . . . 65
5.1.1. Naive memristive initialization . . . 66
5.1.2. Kaiming initialization . . . 67
5.1.3. Xavier initialization . . . 68
5.1.4. Comparison of initialization methods for memristive ANN layers 68
5.2. Memristive update dynamics . . . 69
5.3. Training ANNs with memristive update dynamics . . . 70
5.3.1. Update schemes . . . 70
5.3.2. Loss of training accuracy due to memristive update dynamics . 74
5.4. Compensation of memristive update disturbance . . . 75
5.4.1. Compensating neuron model . . . 76
5.4.2. Compensation of the update asymmetry . . . 82
5.5. Summary of Memristive ANNs . . . 85

6. Memristive Spiking Neural Networks 87
6.1. Spiking Temporal Memory with memristive synapses . . . 87
6.1.1. Principles of the Hierarchical Temporal Memory (HTM) and biologically plausible SpikingTM . . . 88
6.1.2. Memristive weight dynamics for synapses in SpikingTM . . . 89
6.2. MemSpikingTM - Spiking Temporal Memory on memristive neuromorphic hardware . . . 94
6.2.1. Network and algorithm . . . 95
6.2.2. Context-specific prediction - Training of high-order sequences . . 102
6.2.3. Parameterization of the MemSpikingTM network . . . 106
6.2.4. Large-scale sequence learning benchmarks . . . 109
6.3. Energy-efficient sequence learning . . . 111
6.4. MemSpikingTM on a real memristive crossbar array . . . 114
6.5. Summary of memristive spiking neural networks . . . 121

7. Conclusion 122

Bibliography 124

List of Figures

2.1. MLP structure 7
2.2. Valence Change Mechanism 15
2.3. Sneak paths in memristive crossbar arrays 20
2.4. Memristive 1T1R arrays 22

3.1. Overview of electrical measurements 30
3.2. Definition of the ON/OFF ratio 33
3.3. Definition of noise and non-linearity 34
3.4. Network setup in PyTorch and Tensorflow 35
3.5. Implementation of a weight layer in PyTorch 36

4.1. Photographic overview of MAD200 chip 37
4.2. Circuit outline of MAD200 demonstrator chip 39
4.3. Array circuit sketch 41
4.4. Unit cells of the three implemented arrays 42
4.5. Layouts of array unit cells 43
4.6. Word line control circuit 44
4.7. Source line control circuit 45
4.8. Bit line control circuit 46
4.9. Peripheral measurement setup 47
4.10. Quasi-static current voltage sweeps 48
4.11. Device initialization with quasi-static voltage sweeps 49
4.12. Potentiation Pulse Length Series 51
4.13. Potentiation Pulse Voltage Series 51
4.14. ON-OFF ratio and non-linearity for potentiation measurements 52
4.15. Depression Pulse Length Series 53
4.16. Depression Pulse Voltage Series 53
4.17. Depression Gate Voltage Series 54
4.18. ON-OFF ratio and non-linearity for depression measurements 54
4.19. Pulse Parameter Dependency of ON/OFF ratio 55
4.20. Sneak path effect in the matrix read out 57
4.21. Array size dependency for parallel initialization 58
4.22. Parallel array-wide potentiation 60
4.23. Parallel potentiation switching statistics 61
4.24. Parallel array-wide depression 62
4.25. Parallel depression switching statistics 63

5.1. Comparison of different initializations for memristive ANN layers 69

5.2. Exemplary combined potentiation and depression measurement for Ti/STO devices . . . 70
5.3. Sign Rule training with different PCMO memristive devices . . . 72
5.4. Fit parameters for PCMO and STO devices for network simulations . . 76
5.5. Exemplary training with Ti / $SrTiO_3$ device dynamics . . . 77
5.6. Direct Random Target Projection for forward determination of the gradient sign . . . 81
5.7. Effect of binomial compensation on pulse length . . . 84
5.8. Effect of binomial compensation on networks with different initializations 85
5.9. Binomial Backward Compensation for PCMO devices . . . 86

6.1. Memristive synapse models in [141] . . . 91
6.2. Results from [141] . . . 93
6.3. Results from [141] on stuck-at faults . . . 94
6.4. System overview for MemSpikingTM . . . 96
6.5. Neuron model for MemSpikingTM . . . 98
6.6. Prediction and inhibition functionality . . . 99
6.7. Exemplary walk through of the MemSpikingTM algorithm at the beginning of the training . . . 100
6.8. Exemplary walk through of the MemSpikingTM algorithm on a mature network . . . 101
6.9. Training of simple high-order sequences . . . 103
6.10. Origin of context specific prediction . . . 105
6.11. Training of complex high-order sequences . . . 107
6.12. Parameters for large scale simulations . . . 109
6.13. Sequence learning benchmark tasks . . . 111
6.14. Energy consumption during training of high order sequences . . . 113
6.15. Electrical parameters for MemSpikingTM on the demonstrator chip . . 115
6.16. Training of the sequence {A, B, C} on the demonstrator chip . . . 117
6.17. Training of high-order sequences {ABC} and {DBA} on an 8-by-8 memristive array . . . 118
6.18. Evolution of the conductance matrix during on-chip training . . . 119
6.19. Conductive pathways for context sensitive learning . . . 120

1. Introduction

Since the demonstration of the first integrated circuit in 1958 by Jack Kilby, computers based on digital electronic circuits in Complimentary Metal-Oxide-Semiconductor (CMOS) design revolutionized numerous aspects of human life and society. In scientific research, super-computers can solve coupled differential equations with speed and accuracy that could never be matched by all mankind's brains. The results headed the development of various new scientific concepts and inventions which just a decade ago could not even be imagined.

The World Wide Web fundamentally changed the way of human communication and allowed for an unparalleled democratization of information.

Modern mobile computers in smartphones made it possible to access this cornucopia of information from almost everywhere and allow to perform complex image and video data processing, all while fitting into a jeans pocket.

However, there were still functionalities, computers on all scales struggled with, e.g., the recognition of faces or voice patterns or the conduct of conversations. Abilities that come natural to humans could hardly be realized by conventional deterministic computer programs. The solution was found in an approach coined artificial intelligence as it was thought that it is the human intelligence that allows it to complete the above-mentioned tasks with such extraordinary performance.

The development of Artificial Intelligence in the form of Machine Learning from its beginning followed two seemingly contradictory approaches regarding the training of artificial neural systems [1]: On the one hand a top-down approach, where an error signal is generated from the final output of the system which can then be assigned to each part of the system by means of mathematical optimization. The most prominent concept here is Error Backpropagation. On the other hand, researcher argued that in the biological brain, there is no such thing as a global error signal. Instead, biological neural networks rely on local computation in synapses and neurons only using locally available quantities such as neighboring action potentials and spikes. Today, especially the first approach is used and gave rise to various specialized Artificial Neural Networks that come close to human performance in their specific tasks, even sometimes surpassing it, especially in terms of inference speed.

However, it turned out that the fundamental architecture of modern computers suits none of these approaches perfectly: Since the beginning of the modern age of digital computers, the overall architecture for computer systems follows the von-Neumann principle where the actual computation is strictly spatially separated from the storage of data and program instructions. Both data and instructions are transferred through the same narrow channel to the computational unit. As for a long time the computation of logical or mathematical operations within this computational unit consumed

most of the processing resources like time and energy, this imposed no problem and led to a high programming flexibility. The resources consumed for data transfer were neglectable. With advances in circuit design and silicon technology however, the processing inside the computational unit became increasingly fast, at one point exceeding the speed of the data transfer. This left the highly advanced computational unit waiting idly while the data must be transferred from a separate storage, consuming vast amounts of time and energy. This issue is called the von-Neumann bottleneck[2]. Due to the generality of the von-Neumann architecture it allows to compute any kind of turing-complete algorithm on it. Therefore, it is also possible to build and train Artificial Neural Networks on such a computer. However, the costs especially in term of energy consumption are immense and thus devastating toward the goal of responsible and sustainable use of electrical power and the limited resources it is generated from.

One way of addressing this issue is the implementation of completely new concepts of computer architecture, especially breaking up the strict separation of data and computation. This concept is called In-memory-Computing (IMC). The core idea is to perform computation and eventually alter stored data without transferring it to a spatially separate computational unit. Only the instructions must be transferred. This concept is promising especially for applications where simple operations are performed on large amounts of data as it is for example the case in the training and inference of Artificial Neural Networks (ANN). Since these become increasingly interesting also for remote sensing and mobile applications like Internet-of-Things (IoT) and smartphones, it makes sense to investigate such dedicated compute architectures to overcome the von-Neumann bottleneck. These examples for possible applications already hint that the overarching goal is to make it possible to execute complex AI algorithms in-place, close to the generation of the data. Today, most applications that are seemingly executed on mobile devices simply send acquired data to a cloud computer where the data is actually evaluated by ANN algorithm. The result is then returned to the mobile device. This demands the transfer of immense data packages sometimes beyond continent and thereby imposes problems on multiple respects. On the one hand, the transmission of large amounts of data consumes large amounts of energy. This limits the battery run time of the mobile device. On the other hand this leads to vast consumption of natural resources. Once for the energy consumed in the data transmission and also for the wide-band world-wide data infrastructure. Energy-efficient data-local compute solutions would allow evaluation of the data in-place where it is generated and therefore save energy for transmission. Furthermore, this approach could also restore data privacy to the end-user.

As the name suggests, the memory, its architecture and performance, plays a crucial role for In-Memory Computing. Thus, when reorganizing the structure, memory is organized in, one can go even a step beyond and integrate new physical types of memory. As of today, the main working memory in computers is based on electric charge or bi-stable electric switches. Both concepts suffer from a high energy consumption due to the need of a constant supply of electrical power. These memory types are volatile and loose stored information upon the loss of supply power. To address this issue of constant power consumption it can be considered to use non-volatile memory

that only consumes power when being read-out or altered. Prominent candidates here are floating-gate transistors and FLASH memory. However, these require high operation voltages whose generation is costly in terms of power and silicon real estate. Furthermore, the technological shrinking of FLASH transistors halted [3] and thus only allows progress in integration density by growing in the third dimension.

An alternative here are so-called memristive devices. These are electrical resistors with a memory of the past applied electrical stimuli. There are various types of physical mechanisms that create memristive switching behavior and numerous materials used. The switching dynamics of these devices can be analog and to exploit the full potential for the use in AI applications, this behavior and the special switching dynamics of memristive devices must be respected.

Therefore, the goal of this thesis is the integration of memristive devices into ANN and neuromorphic algorithms that goes beyond using them for pure data storage. As memristive devices exhibit specific properties in terms of read-out and switching behavior, dedicated learning rules are developed to mitigate or exploit these dynamics.

Outline

After a description of the core technologies used in this thesis, namely machine learning algorithms for memristive artificial neural networks, neuromorphic computing and memristive devices, and an overview of the state-of-the-art in memristive solutions to Artificial Intelligence algorithms, a demonstrator chip is described which is developed during this thesis. The chip is intended as a test vehicle to implement and evaluate various concepts for the implementation of memristive devices in machine learning and neuromorphic algorithms. Afterwards, firstly the implementation of memristive devices as weights in ANNs is discussed and means and methods developed how to mitigate the impact on their switching behavior on the training accuracy of such networks. In the next section, a concept for the implementation of memristive devices into a neuromorphic algorithm based on the Hierarchical Temporal Memory concept is developed. It is shown how this algorithm is adapted to fit and exploit the memristive switching dynamics and how this can lead to energy savings. The section is concluded with a validation of the algorithm on the demonstrator chip.

2. Fundamentals

2.1. Machine learning toward artificial intelligence

Brief history of machine learning

Early ideas of artificial systems with human abilities of reasoning and decision-making exist in Greek Mythology, e.g., with the *automata*, created by Hephaistos [4]. In modern times, beginning in the 19th and early 20th century, concepts for artificial intelligent systems were presented in popular Science Fiction literature. But even first real "intelligent" machinery like chess machines resembled human level reasoning even though their creators them-self refused to call them intelligent [4]. The first wave of modern AI research began with the Turing machine[5] as a computational concept for and the Turing test or Imitation Game [6] as an approach to define artificial intelligence. This term however was first coined during the Dartmouth Conference in 1956 [7]. A wide-spread idea stated that intelligence could be reduced to a combination of mathematical calculations. This paved the way toward connectionist visions for the functionality of the human brain like Rosenblatt's Perceptron in 1958 [8], igniting a first wave of enthusiasm in AI research.

Exaggerated performance predictions and in particular the publication of Minsky's and Papert's book *Perceptrons* in 1969 [9] broke this wave. Hans Moravec argued that computers were not powerful enough to allow them to exhibit intelligence [10].

Research in alternative computing principles, so-called *expert systems*, sparked the interest in artificial intelligence again in the 1980s [11]. With sufficient computational power and large-scale computers in the late 1990s artificial intelligence systems started to solve real-world tasks like playing chess on a human level [12] and recognizing speech [13]. Today, artificial intelligence systems are applied in various fields, ranging from advanced control for robotics[14] and in the automotive sector [15], over virtual assistants[16] with vocal interaction, to complex reasoning and planning tasks in medicine [17] and the economic sector [18].

2.1.1. Overview of artificial neural networks

Today, there are different general architectures for artificial neural networks employed for different tasks. For static or frame-based classification workloads, such as image recognition, for example Multilayer Perceptrons (MLP) are used since the 1980s [19]. This fully connected network is similar to Rosenblatt's Perceptron but with one or more hidden layers of nodes that allow the network to classify non-linearly separable

data as well. More complex, but also more powerful are Convolutional Neural Networks (CNN) [20]. Here, convolutional layers, where small filter kernels scan the input data, are combined with pooling layers that detect the presence of certain features. Often multiple stages of convolutional and pooling layers are arranged in series to perform feature extraction before a fully connected network does the actual classification. Due to the pooling layers, CNNs are invariant to spatial transitions in the input data.

For temporally linked data streams like audio and video, but also financial data sets, Recurrent Neural Networks (RNN) are most prominent. The simplest form resembles an MLP, but allows not only feed-forward connections, but also recurrent connections[21]. A prominent version of RNNs with time-dependent and plastic neuronal nodes is the Long Short-term Memory (LSTM) [22].

Besides classification, ANNs can also be used to generate data. Generative Adversarial Networks consist of two networks, where one generates data according to an input class, while the other network classifies the result and returns errors to the generator network to improve the generation [23].

Training paradigms

All these networks require training on sample data that is connected to the task. During training, the parameters, which need not only be weights but also neuron parameters and biases, of a network are adjusted according to a learning rule. There are three different categories into which most learning rules can be subsumed. These categories differ in the way a training stimulus or error is generated. The first type is supervised learning. As the name indicates, a supervisor or trainer knows the correct answer to every sample task the network should produce. In a classification task, the learning rule receives the correct class and can compare this to the network's response.

In the second class of training concepts, the trainer does not tell the learning rule the exact answer to the task, but only if the network's result was correct or false. Correct answers cause strengthening of connections, while false responses trigger changes in the network. This method therefore is called Reinforcement learning.

Lastly, it is possible to train a network without having a trainer at all. In such a situation the network aims at clustering data based on similarities among multiple samples. This method is called Unsupervised Learning.

2.1.2. Detailed view on Multilayer Perceptrons

In the machine learning section of this work, of the above-mentioned network types only MLPs and their supervised training are investigated in detail. Therefore, in the following section a detailed mathematical description of an MLP is provided and its supervised training with Gradient Descent and Error Backpropagation explained.
MLPs consist of neurons ordered in layers L of size N^L and connected by weights $w_{i,j}^L$ as depicted in Figure 2.1. In this structure, each neuron j receives inputs x_i^L, which are the previous layers outputs, that are weighted with the respective weights $w_{i,j}^L$

from multiple other neurons. These inputs are summed up, yielding the activation a_j^L of the neuron j with

$$a_j^L = \sum_{i=0}^{N^L} x_i^L * w_{i,j}^L$$

An activation function σ that depends on the sum of the inputs and in some cases a neuron specific bias b_j determines the output of the neuron y_j^L which is the input x_j^{L+1} of the next layer to

$$x_j^L + 1 = y_j^L = \sigma(a_j^L + b_j^L)$$

In this thesis no biases are used. Therefore, the b_j^L term is dropped in the later discussion. The activation function σ is a monotonically increasing, usually non-linear function. Common examples are the sigmoid function, the hyperbolic tangens, the Rectifying Linear Unit $ReLU = \begin{cases} a_j & , a_j \geq 0 \\ 0 & , a_j < 0 \end{cases}$, or the Heavyside function $H = \begin{cases} 1 & , a_j \geq 0 \\ 0 & , a_j < 0 \end{cases}$ for McCullogh-Pitts neurons [24]. The input of the first layer is the data that is to be classified. In the case of the common Modified NIST (MNIST) data set of hand-written letters, this is the vector of gray scale values of all pixels of one frame. The size of the first neuron layer therefore depends on the format of the input data. In the last layer of the network, the output layer, the neuron with the highest output $y_j = x_j^{L_{max}}$ represents the classification result. The size of the output layer is therefore determined by the number of classes.

2.1.3. Training with Gradient Descent and Backpropagation

For small networks (especially single layer perceptrons) and simple problems, the weights of the network can be determined by combinatorial logic or statistical methods like Multiple Linear Regression. This is for example the case for the implementation of basic Boolean functions like AND, OR, and NOT [25]. For more complex tasks gradient-based training algorithms are used. In such algorithms, the output of the network is compared to a trainer signal T which represents the correct answer. The difference is subject to a loss or cost function $\mathcal{L}$ which can for example be the Mean Square Error (MSE) or Cross Entropy Loss. With knowledge of the neuron function and this loss, a gradient can be attributed to each weight (and bias) in the network that tells how much the respective parameter must be changed in order for the network to match the trainer signal. Since each gradient is calculated with respect to one single input sample and therefore only one specific class, usually only a fraction of the gradient is applied (the learning rate λ). With this approach the parameters p of the network can be gradually trained to classify multiple inputs belonging to different classes like

$$p \longleftarrow p - \lambda * \delta p$$

where δp is the gradient of the loss of the network with respect to the parameter p, $\delta p = \frac{\partial \mathcal{L}}{\partial p}$.

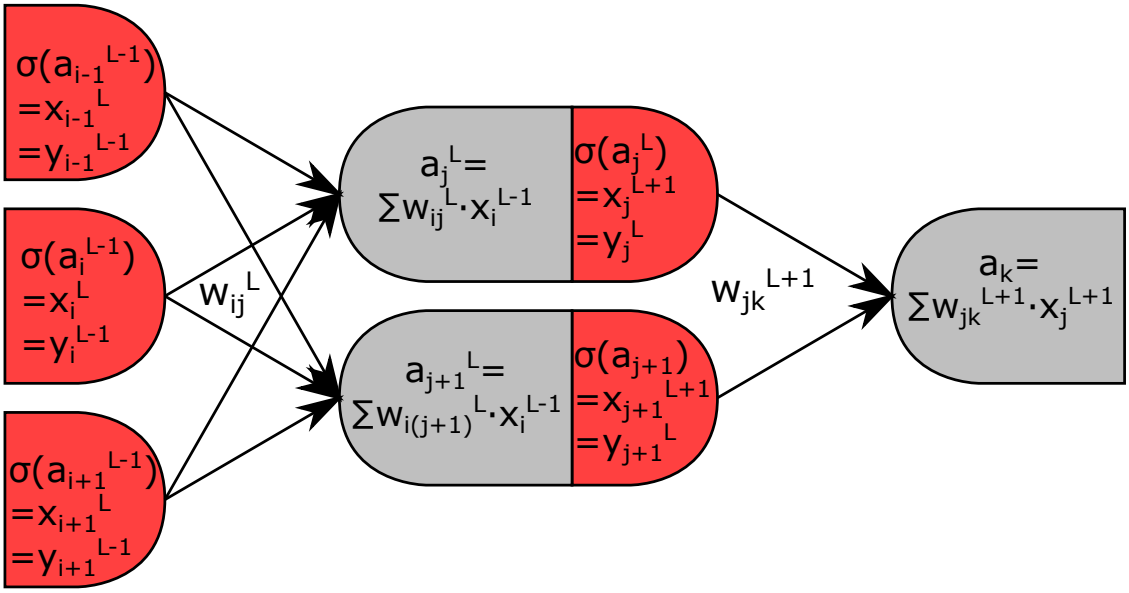

Figure 2.1.: **Layer structure of a Multilayer Perceptron** Neurons (red and gray shapes) are connected by weight layers (arrows). Neurons consist of an accumulation part in which the weighted inputs from the previous layer are summed up (gray) and an activation function (red) providing input to the next layer via a fully connected weight layer.

For a perceptron network without recurrent connections consisting of only a single layer of weights, the gradients for these weights and the biases can be calculated in a simple manner. As described above, the output of the (single) output layer of neurons can be calculated as

$$y_j = \sigma\left(b_j + \sum_{i=0}^{N^{in}} w_{i,j} * x_i\right)$$

where x_i is the ith input from the current sample and y_j the jth output values. $\underline{y}$ is achieved by performing a vector-matrix multiplication (VMM) of the weight matrix W and the vector of input values $\underline{x}$, adding a bias vector $\underline{b}$, and performing the activation function elementwise on the sum.

The error signal e is calculated by elementwise subtracting the output y from the trainer signal T which is simply a one-hot encoded representation of the class that belongs to the input sample. When using Mean-Square-Error (MSE) as loss function, the loss calculates to

$$\mathcal{L} = \sum_{j=0}^{N^{out}} e_j^2 = \sum_{j=0}^{N^{out}} (y_j - T_j)^2 = \sum_{j=0}^{N^{out}} \left(\sigma\left(b_j + \sum_{i=0}^{N^{in}} w_{i,j} * x_i\right) - T_j\right)^2$$

With this formula, the gradient of the loss with respect to the weight and bias parameters can now be calculated using the Chain Rule of differentiation:

$$\begin{aligned}\delta w_{i,j} = \frac{\partial \mathcal{L}}{\partial w_{i,j}} &= \frac{1}{\partial w_{i,j}} \sum_{j=0}^{N^{out}} \left(\sigma \left(b_j + \sum_{i=0}^{N^{in}} w_{i,j} * x_i \right) - T_j \right)^2 \\ &= 2 * e_j * \frac{1}{\partial w_{i,j}} \sigma \left(b_j + \sum_{i=0}^{N^{in}} w_{i,j} * x_i \right) \\ &= 2 * e_j * \sigma' \left(b_j + \sum_{i=0}^{N^{in}} w_{i,j} * x_i \right)\end{aligned} \tag{2.1}$$

where σ' is the derivative of the activation function to its argument. The gradient for the bias b_j can be calculated in a similar fashion to

$$\delta b_j = \frac{\partial \mathcal{L}}{\partial b_j} = 2 * e_j * \sigma' \left(b_j + \sum_{i=0}^{N^{in}} w_{i,j} * x_i \right) \tag{2.2}$$

For perceptrons with multiple (hidden) layers, the calculation is slightly more complex, but follows the same approach using the Chain Rule. Since the weight $w_{i,j}^{L-1}$ indirectly influences all signals beyond layer L+1 of the network, it is no longer possible to continue the calculation with only one term of the outermost sum. Nevertheless, the gradients can be calculated using the Chain Rule. For the sake of simplicity, sub-indices are dropped where possible.

For the gradients of the weights of an arbitrary layer L, one can state

$$\begin{aligned}\delta w^L &= \frac{\partial \mathcal{L}}{\partial w^l} \\ &= \frac{\partial \mathcal{L}}{\partial y^L} * \frac{\partial y^L}{\partial w^L} \\ &= \frac{\partial \mathcal{L}}{\partial y^L} * \frac{\partial 1}{\partial w^L} * \sigma \left(\sum_{i=0}^{N^{L-1}} w_{i,j} * y_i^{L-1} \right) \\ &= \frac{\partial \mathcal{L}}{\partial y^L} * \sigma'(a^L) * y^{L-1}\end{aligned} \tag{2.3}$$

This term contains the derivative of the activation function σ' which is evaluated at the activation a^L that is calculated during the forward pass, and the input to the weight layer which is the last layer's output y^{L-1}, which is also known from the forward pass. The only term which still needs to be determined is $\frac{\partial \mathcal{L}}{\partial y^L}$, the derivative of the loss with respect to the output of layer L. When executing the Chain Rule to

determine this term, a regularity is observed that helps to simplify this calculation.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial y^L} &= \frac{\partial}{\partial y^L} \sum_{j=0}^{N^{out}} \left(\sigma \left(b_j^{(L)} + \sum_{i=0}^{N^{L-1}} w_{i,j}^L * y_i^{L-1} \right) - T_j \right)^2 \\
&= \frac{\partial}{\partial y^l} \sum_{j=0}^{N^{out}} \left(\sigma \left(a_j^L \right) - T_j \right)^2 \\
&= 2 * \sum_{j=0}^{N^{out}} \left(y_j - T_j \right) * \frac{\partial}{\partial y^l} y_j^L = 2 * \sum_{j=0}^{N^{out}} \left(y_j - T_j \right) * \frac{\partial}{\partial y^L} \sigma \left(a^L \right) \\
&= 2 * \sum_{j=0}^{N^{out}} \left(y_j - T_j \right) * \sigma' \left(a^L \right) \frac{\partial}{\partial y^l} \left(w^L * y^{L-1} \right) \\
&= 2 * \sum_{j=0}^{N^{out}} \left(y_j - T_j \right) * \sigma' \left(a^L \right) * w^L * \frac{\partial}{\partial y^l} \left(y^{L-1} \right) \\
&= 2 * error * \sigma' \left(a^L \right) * w^L * \frac{\partial}{\partial y^L} \sigma \left(b + w^{L-1} * y^{L-2} \right) \\
&= 2 * error * \left(\sigma' \left(a^L \right) * w^L \right) * \left(\sigma' \left(a^{L-1} \right) * w^{(L-1)} \right) * ... * \left(\sigma' \left(a^{L+1} \right) * w^{L+1} \right) * \frac{\partial y^l}{\partial y^L} \\
&= 2 * e * \prod_{k=L+1}^{L} max \left(\sigma' \left(a^k \right) * w^k \right)
\end{aligned} \tag{2.4}$$

This is the Backpropagation of the error. This formula can be understood in a way that, in order to propagate the network's error backwards, it is multiplied by the weight matrix of the next higher weight layer and the derivative of the activation function evaluated at the activation of the respective layer. Thereby, a local gradient can be assigned to the output of each neuron layer in a successive manner. With the knowledge of these local gradients and equation 2.3, gradients for each weight layer can be determined.

2.1.4. Initialization of weight layers

A crucial aspect to neural network training is the initialization of the weights, which, if done in the wrong manner, can not only slow down learning but can render training impossible completely.

The weights affect the computed gradients in two ways. In the forward pass, the input to a neuron layer obviously depends on the weights as it is the vector-matrix multiplication of the previous layers output and the weight layer. In the backward pass, the gradients are backpropagated from the output to higher layers by again multiplying the gradients with the weight matrix as described in equation 2.4. From this perspective, it is obvious that weights must not be initialized with zero. Furthermore, a constant initialization does not make sense because if all weights are equal in

one layer each weight will be subject to the exact same gradient and a differentiation cannot be achieved.

Therefore, weights are usually initialized by drawing from normal or uniform distributions. Here, the question is how to appropriately set the parameters of these distributions, the minimum and maximum value or the mean and standard deviation, respectively. A poor choice can, especially in deep networks, lead to vanishing or exploding signals in the forward pass or vanishing or exploding gradients in the backward pass, which all can impede the learning.

Vanishing or Exploding Signal Problem If the mean of the neuron activation decreases from one layer to the next, numerical rounding errors and the activation function's non-linearity can perturb the signal. In the context of analog electric signals, a signal will become unrecognizable if it drops below the noise limit. On the other hand, if the mean increases from layer to layer, when using activation functions with limiting characteristics, such as sigmoid, the output would only assume either the minimum or the maximum of the activation function's value range. In analog hardware, current and voltage ratings of components limit the usable range of signals.

Vanishing or Exploding Gradient Problem If the mean of the gradients propagated from one layer to the next higher layer changes drastically, the weights of the different layers adapt at different speed. This can in the case of vanishing gradients cause the network to adapt only in the lowest layers, while higher layers near the input remain as initialized. Therefore, the potential benefit of having multiple layers is diminished as they are not trained. Exploding gradients, which occur if the mean of the gradients is increasing during backpropagation, however, can lead to oscillating weights in higher layers and can cause the training to fail entirely.

In the context of analog electric signals, all constrains described above for Vanishing and Exploding Signals also hold for Vanishing and Exploding Gradients.

To address these issues, a number of initialization recipes were developed. Two of the most prominent ones, "Xavier"[26] and "Kaiming"[27] initialization aim at preserving the magnitude of the propagated signals and gradients.

2.2. Neuromorphic computing

The training and inference in ANNs for Machine Learning is governed by a top-down end-to-end mathematical description of the information flow and a global-to-local loss attribution by the Backpropagation algorithm. Neuromorphic computing starts from the opposite direction by developing network dynamics bottom-up, starting from the behavior of single fundamental compute elements.

One reason for this fundamentally different approach is the origin of Neuromorphic Computing and Engineering in the Neuroscience research. The field of Neuromor-

phic Computing takes inspiration from Neuroscience and aims to mimic biological functionalities to achieve nervous or brain-like performing systems.

Brief history of neuromorphic computing

The idea behind neuromorphic computing as introduced by Carver Mead in 1980 is the introduction of neuroscience research results into electronic computing systems [28], [29]. The first systems developed in his group used biological principles for signal processing in sensory systems such as vision [30], [31] and hearing [32]. Recent development also shows concepts for olfaction [33] and touch [34]. Additionally, motor control was implemented by neuromorphic hardware [35] as well. All these systems use analog [36] or mixed-signal electronic circuits to facilitate early signal processing of sensory information, but implemented no or very little high-level combination and abstraction of the results that could be regarded as reasoning. From a neuroscientific perspective, this however makes sense because these two functions, raw sensory data processing and high-level reasoning, are performed in different areas of the brain [37], [38]. This led to the development of more general multi-purpose systems that implement neuromorphic principles[39]. All systems employ the idea of parallel computing and local communication, that is sparse in most cases but can also be dense, i.e., for IBM's $TrueNorth$ system [40]. The systems can further be discriminated between digital systems (like $SpiNNaker$ [41] and $Loihi$ [42]) and mixed signal / analog implementations ($BrainScaleS$ [43]). Alongside these large-scale systems, new algorithms are developed for the sections of the brain that are responsible for high-levels tasks such as cognition, reasoning and planning, like the neo-cortex. One prominent concept is the Hierarchical Temporal Memory (HTM) [44], [45], developed by Jeff Hawkins and his colleagues at Numenta, Inc. This concept takes the neurophysiological finding of multiple functional layers in the cerebral structure and proposes a hierarchical organization in which tasks are solved in equal circuits (mini columns) but undergoing different abstraction steps. Important in this context is also the abstraction of events in so-called reference frames[46]. This allows a similar spatial and temporal encoding and processing of completely different types of events and tasks .

Differentiation between ANNs in Machine Learning and Neuromorphic Computing

It is argued that the overall structure of ANNs with nodes and massively parallel signal propagation through weight layers is inspired by the architecture of the biological brain. Nevertheless, there are fundamental differences between ANNs in Machine Learning and Neuromorphic Computing, both on the algorithmic and system design level[47], [48].

Algorithmic differences deal with the working principles of the system:

- **High parallelism:** Instead of single multi-purpose compute unit, which are used to train and operate ANNs in machine learning, a neuromorphic system

executes computation in highly parallel fashion, distributed among many more simple compute tiles. These compute blocks are called neurons and synapses after their biological counterparts.

- **Event-based computing** Even though the computing is highly parallel, the overall compute activity is low compared to parallelized conventional compute architectures, e.g., in GPUs. The reason is that computation in neuromorphic systems is only executed if data in the form of an event appears.

- **Biological time scales:** In contrast to modern digital compute units like ASICs and CPUs, in the biological brain events and data are not processed on a sub-nanosecond time scale. Biological neurons only show an average spike rate of up to 10Hz and synaptic plasticity relies on ionic processes that take place in the tenth to ten second regime.

It is important to note that neuromorphic algorithms can nevertheless be executed on conventional digital hardware. This is for example done in neuroscience research because it allows the simple adjustment and observation of parameters and complete algorithms. This is called a simulation of a neuromorphic system. The NEST framework runs on conventional CPUs and GPUs, but also dedicated systems like SpiNNaker and Loihi implement this approach. In contrast to the simulation of biological computing principles, other systems emulate these. This means that biological time constants and processes like ionic currents are reflected into electronic hardware, for example by exploiting the sub-threshold dynamics field-effect transistors [30]. Systems like *BrainScaleS* on the other hand emulate biological neuron behavior but not at biological time scales which leads to a speed-up of up to three orders of magnitude. The emulation of neuromorphic systems mostly aims at these speed-ups or energy-efficiency.

In both simulation and emulation systems, the above mentioned algorithmic differences in the computing paradigms can lead to significant changes in the computing substrate:

- **Asynchronous operation:** In an event-based computing paradigm, circuits perform operations only if they receive input. Especially external stimulus inputs are not necessarily synchronized to a global clock. Therefore, a neuromorphic hardware system consists of asynchronous or only locally synchronized circuits.

- **3D connectivity** Neurons in the brain have up to 10^6 synaptic connections in three dimensions. Conventional ANNs mostly feature 2D connections in the form of vector-matrix multiplication. In terms of implementation in integrated circuits 3D connectivity is also complicated to achieve in conventional planar CMOS technology [47].

- **Compute and memory collocation:** Conventional ANN networks and algorithms are run on von-Neumann hardware that dictates the strict separation of

memory and computing operations. In the biological brain, information is stored locally within the synapses and even neurons. This proposes the co-localization of memory and compute also in neuromorphic hardware systems.

- **Local computation:** Hebbian learning which is proposed as one of the core learning rules of the biological brain [49], relies exclusively on the signals generated by the neurons directly connected to a synapse. It is however proposed that global signals such as surprise (experiencing an unknown event) and reward (satisfaction by an event) can modulate this learning [50]. In a conventional system reward or loss are computed in a central compute entity separate from the memory (weight layers) and then assigned to each weight. There are however approaches to unite both concepts by the use of so-called Three Factor Rules [51]. The three factors can be derived from the theory of BP and two of its components then assigned to locally available quantities, while the third factor represents a global reward or loss [52].
- **Stochasticity:** ANNs in machine learning rely on exact mathematical calculations and integrated circuit elements have been tailored to exhibit deterministic behavior. Synapses and neurons in the brain however are inherently stochastic. Neuromorphic algorithms are not only tolerant to this stochasticity, but it can also be an inherent part of them. Therefore, neuromorphic substrates can operate with stochastic compute elements. Geoffrey Hinton even suggests that the exact functionality of computing devices in a neuromorphic circuit can be unknown [1].

2.3. Memristive devices

Memristive devices are electrical resistors with an electrically adjustable resistance. Most memristive devices have two terminals. Upon the application of an electric stimulus between these two, the resistance can be changed. There are also device types with three terminals in a transistor-like structure, where one terminal can be used to change the resistance between the common ground and a third terminal. Different physical principles can be employed to realize this effect, yielding different types of memristive devices. Among these, Phase Change Memory (PCM), Magnetic Random Access Memory (MRAM), Ferroelectric RAM (FeRAM) and Redox-based RAM (ReRAM) are the most prominent [53]–[55].

In PCM type devices an amorphous and a crystalline phase of a chalcogenide glass exhibit different electrical resistivities [56]. Sandwiched between two metal electrodes, this change in phase can be triggered by melting the chalcogenide glass and subsequently cooling it down slowly, leading to crystallization, or quenching the mold yielding the amorphous phase [56], [57]. PCM is already available in commercial storage applications like the Intel X-Point technology [58] where its benefits are long term

[1] https://cacm.acm.org/news/267268-we-will-see-a-completely-new-type-of-computer-says-ai-pioneer-hinton/fulltext

stability and a comparably high resistance window [59]. Drawbacks for neuromorphic applications can be found in the highly asymmetric switching behavior (a SET pulse is much longer than the RESET) [60].

Magnetic RAM relies on the filtering of electron by their spins in magnetic materials. Most devices consist of two magnetic layers with spin-ordered domains of which one's spin direction can be altered to be either parallel or anti-parallel to the fixed other. Electrons with an anti-parallel spin to the polarized layer cannot pass. Therefore, if the two layers are polarized anti-parallel, all electrons will be blocked in one or the other layer. If both layer's polarizations are in parallel, electrons can flow [61], [62]. There are different ways how to influence the polarization of the changeable layer, with Spin Torque Transfer (STT) [63] and Spin Orbit Transfer (SOT) [64], [65] being the most relevant. MRAM is also already available in commercial semiconductor processes [66] and applied e.g., in cache-like circuits [67], [68]. Drawbacks for neuromorphic applications are the comparably low ON/OFF ratios [69] and considering the goal of energy efficiency, the high switching currents of the STT mechanism [70], [71]. SOT-MRAM can partially mitigate these shortcomings [70].

2.3.1. Redox-based resistive random access memory - Valence-change mechanism

In this work, memristive devices are investigated that change their resistive state through a redox process between and within different transition metal oxide and metal layers [72]. This type of memristive device is called OxRAM (for Oxide RAM) or ReRAM (for Redox RAM). In this thesis the latter term is used.

ReRAM devices consist of a stack of a transition metal oxide (TMO) and a metal electrode [53]. For many devices where the TMO's bulk conductivity is low, the TMO is sandwiched between two metal electrodes, which are usually of different materials.

ReRAM devices rely on the Valence Change Mechanism (VCM) to produce a chance of electrical conductivity. If the oxygen concentration changes in the vicinity of a transition metal ion in a TMO, the ion changes its valence state [73]. The valence change causes an increase or decrease of the free electrons this ion provides to the conduction band or holes to the valence band, respectively. This is the Valence Change Mechanism.

In ReRAM devices, this effect mainly takes place at the interface between the TMO and the noble metal electrode. A visualization of the switching process can be found in Figure 2.2. At TMO / noble metal interface, a Schottky barrier forms due the different work functions of the TMO and the noble metal [74]. This potential blocks electron flow across the interface (see Figure 2.2 a) and b)). If, for example due to the VCM, the density of free electrons in the vicinity of this interface barrier increases, the electrons partially compensate the Schottky barrier's blocking potential (see Figure 2.2c) and d)). This decreases the electrical resistance measured across the TMO / metal device stack and thereby the resistance of the ReRAM device. This process is reversible if the oxygen concentration at the interface is increased and the

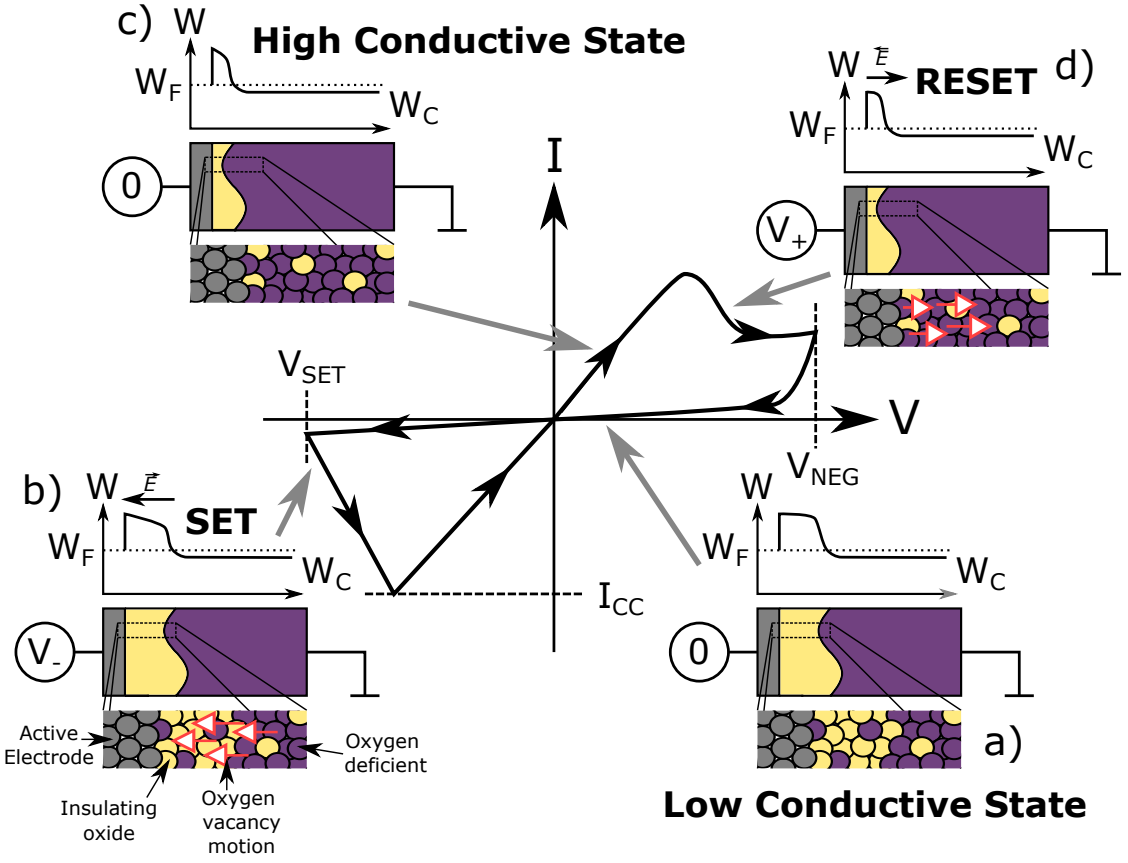

Figure 2.2.: **Principles of the Valence Change Mechanism (VCM)** a) If the concentration of oxygen vacancies is low (yellow) at the interface between the high conductive oxygen deficient filament (purple) and the active electrode (gray), a wide Schottky barrier forms in this interface region due to the difference in work function of the switching oxide and the active electrode. This results in a low conductive state of the device. In the highly oxygen deficient area, the concentration of free electrons pulls the conduction band W_C below the Fermi-level W_C, leading to metal-like conduction. b) A negative voltage applied to the active electrode creates an electric field in the interface region and thus a driving force for the two-fold positively charged oxygen vacancies to move toward the interface. The oxygen vacancies act as mobile donors and decrease the width of the Schottky barrier, increasing the device conductance which leads to higher current and higher temperature, increasing the vacancy mobility further. This positive feedback leads to a sharp SET event at V_{SET} which is stopped when the current reaches the compliance I_{CC}. c) In the high conductive state, the concentration of oxygen vacancies near the interface is high which decreases the width of the Schottky barrier. d) If a positive voltage is applied to the active electrode, the resulting electric field drives positively charged oxygen vacancies away from the interface, oxidizing the interface region and thereby increasing the Schottky barrier and the device resistance. The decrease of current decreases the temperature and with it the ion mobility. Therefore, the RESET is self-limiting. The reset-stop voltage V_{NEG} can determine the conductance after the RESET.

concentration of free electrons subsequently decreased [75], [76].

The oxygen content or, in reverse, the concentration of oxygen vacancies near the interface between the TMO and the noble metal electrode determines the electrical resistance of ReRAM devices. In order to change the resistance electrically, the concentration of oxygen vacancies must be changed by electrical stimuli. If a voltage pulse is applied to the device, the resulting electric field is strong enough to move two-fold positively charged oxygen vacancies. If the concentration of oxygen vacancies in the region of the interface barrier is increased and the device's conductance increases (see Figure 2.2 b)), which is called a SET process. The opposite decrease of device conductance due to a decrease of oxygen vacancy concentration (see Figure 2.2 d)) is called a RESET [77].

The polarity of the applied voltage pulses determines the movement direction of the oxygen vacancies. If a negative voltage is applied to the active electrode, the electric field drives oxygen vacancies toward the TMO / noble metal interface, causing a SET (see Figure 2.2 b)). A positive polarity drives oxygen vacancies away from the interface, leading to a RESET (see Figure 2.2 d)). This switching mechanism is called the normal or counter-eightwise switching, due to the shape of the resulting IV-curve. However, it is also observed that certain devices can be set with a positive votlage polarity to the active electrode and reset with a negative polarity. This was attributed to oxygen movement between the TMO and the noble metal at their interface [78]. This switching mechanism is called anomalous or eightwise switching. While the counter-eightwise switching relies on movement of oxygen (vacancies) within the TMO layer, for the eightwise switching, the oxygen undergoes an electron transfer reaction at the TMO / noble metal interface.

The process of oxygen vacancy movement can be additionally accelerated by increased temperature. Even if the material is low-conductive, upon a significant applied electric field, a high electrical power density is produced which results in a (local) increase of up to multiple hundred Kelvin. Since the mobility of the oxygen vacancies follows an Arrhenius Law, it increases exponentially with temperature. This contribution of temperature to the mobility of oxygen vacancies and therefore the switching speed can trigger a positive feedback mechanism. An increased oxygen vacancy concentration causes a higher conductivity and subsequently a higher current density if the electric field remains the same. This leads to a higher electrical power density that locally dissipates and therefore a local increase of temperature, which again increases the movement of oxygen vacancies[79].

2.3.2. Forming

Most ReRAM devices require an initialization step before they can be switched normally. If the pristine TMO layer has a low electrical conductivity, a high voltage pulse drives initial oxygen vacancies toward the Schottky barrier interface. In this process parts of the material become highly conductive, and a so-called virtual cathode grows from the TMO's boundary opposite to the noble metal interface in the direction of this electrode, forming a filament of oxygen vacancies. Due to inhomogeneities in

the material the movement of this virtual cathode is faster in some regions of the device area. A faster growth of the filament reduces the spatial distance between the virtual cathode and the opposite electrode. This increases the electric field in this region which enhances the field-driven movement of oxygen vacancies. This is a self-accelerating process leading to one single filament with a small diameter compared to the device area which finally reaches the opposite noble metal electrode. Further switching will take place mostly at the interface between this filament and the noble metal electrode. Such devices are called filamentary switching devices. These devices especially benefit from the thermal acceleration of the switching process because the small diameter of the filament leads to a high electrical current density and therefore a high power density.

2.3.3. Filamentary and area-type switching

The forming process described above leads to the formation of conductive filaments in the switching oxide that have a significantly higher electrical conductivity than their laterally surroundings [80], [81]. For some material systems, these filaments can even be observed by local conductance measurements [82] or spatially resolving XPS techniques [83]. The switching takes place at the interface between these high-conductive filaments and the high work function top electrode. On the one hand the small current cross sections can lead to high switching speeds [79], but at the disadvantage of increased variability [82], [83].

However, there are also ReRAM devices that do not exhibit switching only in confined filament regions, but across the complete area of the device [84]. This type of switching is called homogeneous or area-type switching. Unlike for the filamentary switching, the mechanism behind the area-type switching is not yet understood completely. There are numerous, some even competing, mechanisms proposed, ranging from magnetic [85] and purely electronic [86] effects to combinations of electronic effects and ion motion [84], [87]. In some material systems even a coexistence between area-type and filamentary switching can be observed[88].

2.3.4. Investigated resistive switching devices

In this thesis three types of resistive switching devices are investigated. All these belong to the class of redox-based resistive switching devices (ReRAM) that rely on the VCM mechanism for resistive switching. They differ in their actual material stacks and details in the switching process. In the following, these device types and their differences are discussed.

$SrTiO_3$-based ReRAM devices Even though $SrTiO_3$ exhibits a relatively complex chemical composition and crystal structure as a ternary oxide with perovskite crystal structure, it is widely used in ReRAM research due to its well understood defect chemistry. This qualifies $SrTiO_3$ as a model material for research in the underlying physical and chemical processes in ReRAM devices [89], [90]. $SrTiO_3$ based ReRAm devices

can exhibit both normal (counter-eightwise) and anomalous (eightwise) switching [88], [91]. These two switching directions rely on different The material system has been used to investigate a wide range of effects of ReRAM devices like retention, switching speed, and variability [92]–[94].

The particular devices investigated in this thesis are based on a $SrTiO_3$ doped with Nb to degeneration. This forms a conductive substrate for an undoped stoichiometric $SrTiO_3$ film, grown with Pulsed Laser Deposition (PLD). This 10**nm** thin film is the switching oxide. On this film, a 1**nm** Ti film is deposited. Due to the high oxygen affinity of Ti, this layer is most likely oxidized to TiO_x. The active electrode is formed by a stack of 10**nm** Pt and 80**nm** Au.

HfO-based ReRAM devices Since it is already used as a high-k gate dielectric in conventional CMOS technology, HfO was easily adapted by many semiconductor foundries, e.g., [95], [96]. Compatibility with advanced CMOS nodes has been demonstrated. Similar to $SrTiO_3$-based ReRAM, devices with a HfO switching layer exhibit filamentary switching. But in contrast to $SrTiO_3$-based ReRAM devices, HfO ReRAM devices are reported to switch exclusively in the normal (counter-eightwise) direction [97], [98].

The HfO-based ReRAM devices used in this thesis are fabricated by CEA-LETI (France) on top of a conventional 130nm CMOS technology from ST Microelectronics (France). The HfO layer and a titanium interlayer are sandwiched between two titanium nitride electrodes.

$Pr_{0.7}Ca_{0.3}MnO_3$-based ReRAM devices In contrast to the two above-described ReRAM cells, ReRAM devices based on $Pr_{0.7}Ca_{0.3}MnO_3$ have been shown to be area-type switching [99]–[101]. The devices discussed in this thesis consist of a 20**nm** $Pr_{0.7}Ca_{0.3}MnO_3$ layer deposited by PLD, and an interlayer sandwiched between two Pt electrodes. The interlayer material is either Al which is partially oxidized, TaO_x, or WO_x. Details on the fabrication can be found in [102].

The switching mechanism of these particular devices is attributed to the movement of oxygen vacancies across the AlO_x / $Pr_{0.7}Ca_{0.3}MnO_3$ interface, as proposed by Gutsche et al. [87]. The change of oxygen vacancy concentration shows opposite effects for these two materials. Oxygen vacancies act as traps in AlO_x and thereby enhance the hopping-based conductance in this layer. In contrast, in $Pr_{0.7}Ca_{0.3}MnO_3$ an increase of oxygen vacancies reduces the conductivity. Therefore, a migration of oxygen vacancies from the $Pr_{0.7}Ca_{0.3}MnO_3$ into the AlO_x increases the conductivity of both materials. This migration can be triggered by applying a negative voltage across the AlO_x / $Pr_{0.7}Ca_{0.3}MnO_3$ into the AlO_x interface. A decrease of the conductivity can be achieved with the opposite voltage polarity.

2.4. Memristive devices for machine learning

The von-Neumann bottleneck impedes the execution of both training and inference of artificial neural network algorithms in multiple regards. First, the mostly serial execution of instructions dislocated from the memory leads to high latencies because all commands and data have to be fetched and transferred to the computational unit before a relatively simple calculation is executed. Then the result has to be transferred back to the memory. Furthermore, the data transfer back and forth between storage and computational unit consumes large amounts of energy. Data is stored in Static Random Access Memory (SRAM) or Dynamics Random Access Memory (DRAM). When accessed, the data is transferred via bus lines to an Arithmetic Logic Unit (ALU). This transfer can consume energy multiple order of magnitudes higher than the actual computation. Furthermore, SRAM and DRAM consume static energy, meaning they rely on constant current flow (SRAM) or need a refresh of the saved data every couple milliseconds (DRAM). These processes additionally consume energy even if no computation is performed.

2.4.1. Memristive weights for artificial neural networks

Memristive architectures have been proposed to overcome these issues. A core principle here is called In-Memory-Computing (IMC) or Compute-In-Memory (CIM). In these systems, data is not transferred to be computed. Instead, operations are performed directly on the memory, changing the stored values in place. This on the one hand reduces the energy overhead of transferring data and instructions back and forth and can also relieve the latency overhead if multiple operations can be performed in parallel. While there are many more implementations of CIMs with conventional CMOS technology, this thesis will focus on CIM architectures based on memristive devices, in particular on memristive crossbar arrays. After an introduction of these architectures, their special abilities to enhance vector-matrix-multiplications (VMMs), a fundamental operation for ANN workloads, will be explained.

Memristive crossbar arrays While memristive devices due to their small dimension can anyway be co-integrated with high density close to a computational core, the fundamental structure for many CIM architectures is memristive crossbar arrays.

In these arrays, memristive devices are ordered in a matrix-like structure with m rows and n columns and $m \cdot n$ devices. One terminal of each device in the same row is connected forming m input lines. Similarly, the second terminal of each memristive devices are connected along the same column, which forms n output lines. Such an array consisting of memristive devices and connection lines only is called a passive array.

So-called sneak paths are one major problem of this type of array. If a device is in the high-resistive (R_{HRS}) but there are low-resistive devices (R_{LRS}) connecting the same input line to another output line, this output line to another input line, and another low resistive device leading from this input line to the same output line

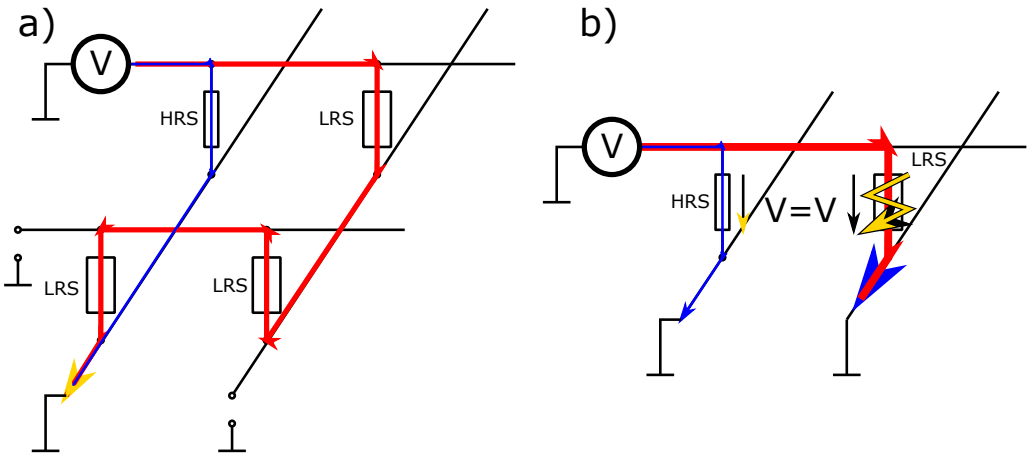

Figure 2.3.: **Sneak path effects in memristive crossbar arrays** a) In a passive crossbar array, a high resistive device can have a parallel network of low resistive devices. When attempting to read out the high resistive device, the current through this parallel network is large, while the actual device current contribution is only minor. The apparent resistance is therefore much smaller than the actual. b) If a high resistive device is supposed to be SET with low resistive devices on the same input line, these devices will conduct a large current before the device-of-interest can be set. This can lead to thermal destruction of the low resistive devices.

the first high-ohmic device is connected to, the three low resistive devices form a connection parallel to the high-resistive device. This situation is depicted in Figure 2.3a). Since the ratio between HRS and LRS often is multiple orders of magnitude the apparent resistance of the high-ohmic device appears much lower than its actual value. More exactly, if the resistance of the first device was to be measured by applying a read voltage to its input line and reading the current at its output line, the apparent resistance would be

$$R_{measured} = R_{HRS} \parallel (3 \cdot R_{LRS}) \sim 3 \cdot R_{LRS} \text{ for } R_{HRS} >> R_{LRS}. \tag{2.5}$$

Moreover, also the switching operation is impeded if devices in the LRS have a much lower resistance than devices in the HRS and require a current compliance during SET or Forming. This comprises almost all filamentary type switching ReRAM devices and therefore, a majority of devices which are used for CIM architectures. When forming or setting such a device in a passive crossbar array, a high voltage is applied to the input line, while the output line is grounded. If all other output lines are grounded as well, all devices on the same input line will experience the same voltage. It is therefore possible to unintentionally form or set a different device. Moreover, if a device goes to the LRS, the current through this device will become significantly higher. If the current is not limited this device can be thermally destroyed. On the

other hand, a current compliance at the input line driver would decrease the supplied voltage and stop the forming or SET process of the addressed device. There are different approaches how to overcome this issue. If output lines which do not connect to the device which is to be formed or set are not grounded but set to the half of the supplied voltage on the input line, the voltage drops over devices which are not to be formed or set is only half of the voltage over the intended device (V-half scheme). The voltages can be chosen such that the half voltage is not sufficient to SET a device. This way, only the intended device would experience sufficient voltage to be set or formed. However, especially in the forming case this approach may not work, because often the forming voltage is significantly higher than the SET voltage.

1T1R arrays These issues with passive crossbar arrays lead to the development of active arrays where a transistor is placed in series with each ReRAM device to gate the access. Now, one source/drain contact of the transistor connects to one terminal of the memristive devices while the other is attached to the input line (now called source line, SL). The gate contacts of the transistors along either each row or column are also connected and form control or word lines (WL). The vertical outputs are referred to as bit lines (BL).

If transistor gates are connected parallel to the output lines along the columns of the array, the structure is called a vertical 1T1R (v1T1R) array. This is the most common active memristive crossbar array in literature [103] because with this structure it is possible to precisely target single devices for operation by applying a voltage on the connected input line and opening only the respective transistor control line while all output lines can remain grounded.

The second possibility is to have all transistor gates along one row connected, in parallel to the source line. This structure was proposed by Qian and Yu and titled "pseudo-crossbar" [104] and claimed to be more energy efficient for vector-matrix multiplication (VMM) operations [105]. This structure can effectively suppress sneak paths by opening only transistor gates along single input lines. However, it is still necessary to apply V-half or other schemes to selectively form or set devices.

Vector-Matrix-Multiplication with memristive crossbar arrays Memristive crossbar arrays can be used to perform VMM operations within a single step. In this way, the conductances of the memristive devices act as the weights of a layer in a neural network. For this operation, either a passive array or an active array with completely opened transistor gates can be employed.

If the conductances G are programmed according to the weights of a neural network layer, an input to the layer can be applied to the crossbar array as a vector of voltages to the input or source lines V. The outputs or bit lines are (virtually) grounded. Each device will produce a current due to the applied voltage and its conductance according to Ohm's Law

$$I_{i,j} = V_i * G_{i,j}. \tag{2.6}$$

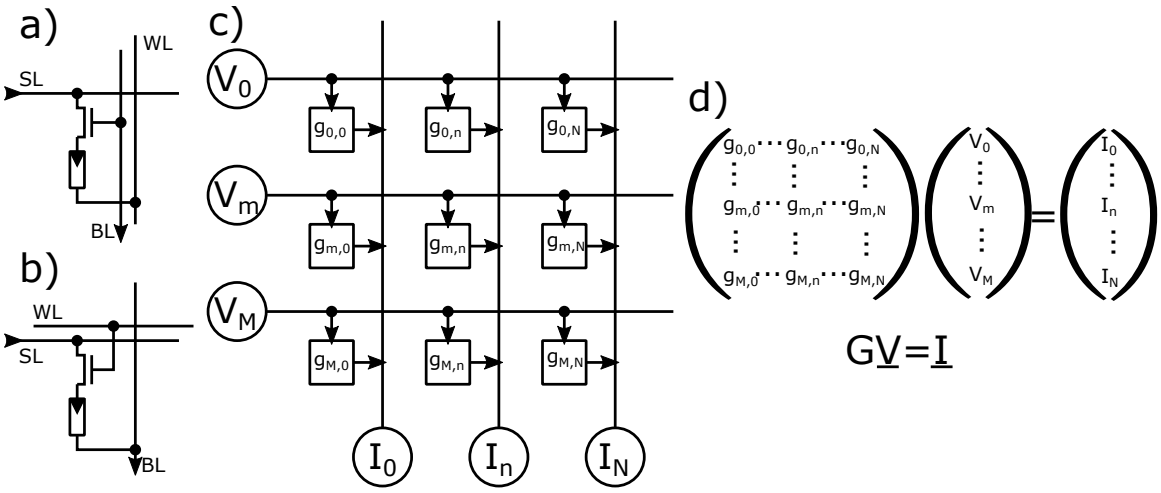

Figure 2.4.: **Memristive 1T1R arrays** a) Basis cell of a vertical 1T1R array. The transistor control line WL is perpendicular to the input line SL. b) Basis cell of a pseudo-crossbar array. The transistor control line WL is in parallel to the input line SL. c) A memristive crossbar array resembles a matrix structure. Each $g_{i,\,j}$ element consists of a memristive device (passive crossbar) or the basis cells in a) or b). The voltage vector $\underline{V} = (V_0 \dots V_M)$ is applied at the horizontal input lines. The memristive devices in the array produce a current upon the applied voltage, which is accumulated on the vertical output lines, yielding the current vector $\underline{I} = (I_0 \dots I_N)$. d) The operation described in c) resembles a vector-matrix multiplication VMM with the matrix of conductances G, the voltage input vector $\underline{V}$ and the resulting current vector $\underline{I}$.

These currents accumulate on the bit lines according to Kirchhoff's Law

$$I_j = \sum_i I_{i,j}. \tag{2.7}$$

The output currents $\underline{I}$ represent the results of the VMM. If the input vector is applied in one step and all currents read out at the same time this operation is performed with $\mathcal{O}(1)$ complexity.

There are other read-out mechanisms for memristive crossbar arrays proposed like a charge-based read-out where analog inputs are encoded in the length of the read pulse instead of the voltage magnitude [106]. The resulting currents are integrated at the bit line outputs. This approach is especially interesting for memristive devices with a high non-linearity of the current-voltage relation even for low read voltages. Another approach resembles the read-out of DRAM cells by pre-charging a line before opening the access transistors of the 1T1R devices [107]. This method is mainly used in Content-Addressable Memory systems. For ANN applications the first described current based read-out scheme is most common in literature.

2.4.2. Co-integration of memristive devices with CMOS technology

ReRAM devices co-integrated with CMOS transistors have already been demonstrated by various groups and with numerous combinations of CMOS technology nodes down to the 22nm node [108] and different ReRAM material stacks. Among these stacks, the most prominent materials are HfO [96], [109]–[111] (often with a TiO thermal enhancement or oxygen vacancy reservoir layer) and TaO_x [112]–[114]. In this thesis, the taped-out IMC demonstrator is based on the HfO ReRAM devices fabricated by CEA LETI, Grenoble, France, and a commercially available 130nm CMOS process by ST Microelectronics [96], [115]. In this process, the standard CMOS process is performed up to the metal 4 Cu layer. After planarizing with Chemical Mechanical Polishing (CMP) and the deposition and structuring of a TiN bottom electrode, the switching layer of 10nm HfO is deposited followed by a 10nm Ti layer. Due to the high oxygen affinity of Ti, a TiO_x layer will form at the interface. Lastly, a TiN top electrode is deposited. With this configuration, the bottom TiN electrode is the active electrode while the top TiN electrode forms the ohmic contact.

Most ReRAM devices co-integrated with CMOS only are used in a binary switching fashion. However, there are concepts to use multiple devices to produce multiple conductance levels [96] and also multiple levels within the same device have been demonstrated [116]. Even though for single devices thousands of conductance levels can potentially be discriminated[82], in a memory system the definition of separable conductive states is more constrained because it also comprises that these states must be the same for all devices across the system. Another approach here are gradually switching devices where no distinct states are defined for all devices and not even for multiple switching iterations of the same device, but only the number of states between two conductance limits.

The most common circuit architecture for the co-integration of ReRAM and CMOS technology are memristive crossbar arrays. These structures are inherently two dimensional. Since the biological brain's connectivity is three dimensional, significant research efforts are put into the development of three dimensional crossbar array structures [117], [118]. A corner stone of this development are selector devices that can be co-integrated back-end-of-line where in most approaches the ReRAM devices are located as well[119]. With the current CMOS transistors, which are located in the silicon substrate front-end-of-line and therefore only in 2D, this would require an immense density of transistors and contact vias through the metal layers.

Many architectures for memristive-CMOS co-integrated circuits have been proposed so far. These range from problem specific solutions to flexible and reconfigurable circuits. An example for a problem specific architecture can be found in [110]. Here a two-weight-layer MLP with memristive weights is designed in series with analog / mixed-signal circuitry that implements the activation functions and the learning rule. It is shown that this system can successfully be trained on the MNIST data set. However, the system architecture is fixed and can only implement the particular 784-100-10 neuron network. The efforts to design and tape-out, especially a mixed-signal,

integrated circuit are huge, and it can be questioned if the approach presented is sustainable. Therefore, many other works propose reconfigurable circuit architectures. This approach is for example conceptualized in [120].

A recent example for such a reconfigurable architecture is the NeuRRAM chip [111]. Similar to the STM / LETI process used in this thesis, the ReRAM devices in this chip are based on HfO and a thermal enhancement layer sandwiches between two TiN electrodes. This architecture is reconfigurable not only in the size of the arrays, but also in the connections of the respective array to its peripheral circuitry of ADCs and DACs. This allows MAC operations of the matrix of conductances of a crossbar array and also of its transpose, which is especially interesting for ANN training with error backpropagation. It is demonstrated that this chip can be used to train and operate CNNs (for classification tasks with the MNIST data set), ResNet (CIFAR), LSTM networks for voice command recognition, and Restricted Boltzmann machines (RBMs) for image recovery tasks.

2.4.3. Memristive neuromorphic hardware

While machine learning algorithms can already solve many tasks of the biological brain successfully and in some cases even surpass performance of humans, the execution on general purpose CPUs and GPUs consumes much more energy than the biological tissue. Therefore, it is proposed to directly implement the functionality of biological computing nodes in electronic hardware to exploit the energy-saving principles of sparse temporal and spatial activity as described in Section 2.2. To reach a maximum of energy-efficiency, it is proposed to combine neuromorphic algorithms with memristive devices, thereby uniting low-power algorithms with low-power compute substrates. Memristive devices are investigated as neurons as well as synapses for neuromorphic hardware.

Memristive neurons often realize simple integrate-and-fire dynamics. The memristive device then integrates the input to the neuron in the internal state variable similar to the membrane potential on a membrane capacity [121], [122]. The memristive device is mostly used in combination with CMOS circuitry and it can replace a large capacitor as it would be necessary for long, bio-plausible time constants. However, Sung et al. show that CMOS-memristive co-integrated neurons can so far not significantly surpass comparable CMOS-only neurons in terms of the energy consumption [123]. Memristive neuron circuits are not further investigated in this thesis.

2.4.4. Memristive synapses

Input data types Memristive devices are further used as synaptic devices in neuromorphic hardware. There are various proposals how this can be achieved. A non-exhaustive selection is listed in Table 2.1. A first differentiation can be found looking at the temporal nature of the processed data in combination with this the solved tasks. Temporal nature in this case means that data can either be static like in single image frames or even based, where data is presented in a sequential manner with a

temporal correlation between the consecutive events. Given that the nature of biological information processing, which neuromorphic systems aim to mimic, is sequential, it is surprising that still many works solve static tasks like the classification of handwritten digits. The reason is that this task is a common benchmark for AI systems. When looking at the results (see "Comments") the results with these static data sets are poor with up to 84% classification accuracy compared to simple MLPs in Machine Learning which reach up to 97.6%. However, the hand-written digit task can also be solved in an event-based manner with the Neuromorphic MNIST (NMNIST) DVS (Dynamic Vision Sensor) gesture set like in [124], where the writing is recorded in a time sequence manner by a DVS camera. Other systems train on synthetic sequence learning and temporal correlation tasks, but there are also works that solve real world temporal classification tasks like gait identification [125].

Learning rule One of the most important differentiating factors is the learning rule that is used to train the network of memristive synapses. Here two mayor categories can be found: Gradient descent and STDP/Hebbian learning. For the gradient descent approach, it is necessary to calculate a gradient for each synaptic device from a (global) error or loss signal. Since synapse and especially neuron dynamics can be non-differentiable, many of these systems use so-called Surrogate-Gradients. These are functions that are similar to the original neuron or synapse dynamic, but differentiable, and are used to calculate gradients in a backward pass in lieu of the original function which is only used in the forward pass. The STDP/Hebbian learning rule is derived from observations of biological synapses [126] and takes only spikes of the neurons into account which are directly connected to a particular synapse. From their temporal correlation, an update is generated. The benefit of this method is that only locally available signals are used to update the synapse, while the gradient descent approach needs the generation and distribution of a global error signal. On the other hand, gradient-based methods are well researched in Machine Learning and their success is almost mathematically guaranteed, while STDP and Hebbian learning rules are assumed to work because they resemble biological learning processes.

Synaptic plasticity Lastly, once an update is generated, it must be applied to the synaptic weight. Here an important difference among the proposed neuromorphic systems can be found. Some apply the generated update exactly, meaning the synaptic (memristive) weight changes of exactly the desired amount, while other apply an update signal that resembles the generated update, but the memristive update dynamics can influence the actual update of the memristive weight. In Table 2.1 these differences can be found the columns "Plasticity" and "plasticity mediation". The plasticity here is differentiated from the learning rule in a way that the learning rule determines the update of the synaptic weight while the plasticity tells if and how this update is applied to the device's conductance value. The plasticity mediation mechanism then gives the process which is employed to write the update. Three major groups can be formed among the representative papers listed in Table 2.1: The first category is the

application of exact gradients or updates, as in [127]–[129]. Since the dynamics of the memristive switching must not interfere with the applied update here, the update is performed in a write-verify fashion. This means that after the update is applied (write), it is checked by a read-out if the conductance of the device changed of the correct amount (verify). If not, the update is repeated. This approach ensures the most correct implementation of the respective learning rule, but it requires potentially large circuitry and latency overheads because it demands a complex memory controller.

The second category is strongly connected to STDP learning rules and exploits the non-linear switching dynamics of memristive devices by applying two equal voltage wave forms to the pre- and post-synaptic terminal of the memristive device [102], [130], [131]. The time difference of these wave forms is the spike time difference of the pre- and post-synaptic neurons. Depending on the sign of the spike time difference, the maximum voltage across the device has a different polarity and therefore determines the direction of the update of the memristive device. Furthermore, the magnitude of the spike time difference can modulate the magnitude of the update. In order for this mechanism to work, the pre- and post-synaptic must overlap in time. An overview of this method can be found in [132]. A benefit of this approach is that in a memristive crossbar array, the voltage wave forms can be applied by peripheral circuitry and no controller is necessary for each synaptic device individually. On the other hand, the need for temporal overlap between pulses is not bio-plausible and strongly restricts the time constants the system can operate at. Long pulses also consume more energy. Furthermore, the wave forms are complex and therefore require precise and elaborate pulse generators. Nevertheless, it can be doubted if the wave forms reach each device in the same way due to line resistance and frequency dispersion on the arrays metal lines. An approach to realize this concept with simple rectangular pulses is presented in [133], [134] where the transistor gate of a 1T1R synapse is opened by the first pulse from either the pre- or the post-synaptic neuron while and the second pulse updates the memristive device. The same concept can also be realized with bi-directional selector devices in a 1S1R structure [135].

In the third category, the update is applied by single voltage pulses to the memristive devices. Therefore, the plasticity is dominated by the switching dynamics of the memristive device and only influenced by the update calculated by the learning rule in so far as the length and amplitude of the update voltage pulse can be influenced. This approach can be found in [124], [136]–[138]. Due to the simplicity of the pulses, this method has the potential to be realized in the simplest fashion in electronics, promising area and energy benefits. On the other hand, the memristive update dynamics can potentially heavily disturb the applied updates. One of the most common memristive switching characteristics that can be observed when using the two latter methods is the saturation of the conductance at a high and a low level when the same update signal is applied repeatedly. With the write-verify method this effect is circumvent by the exact application of gradients as long as the demanded update value is within the dynamic range of the device. However, while it could be assumed that this saturation effect is an issue for the learning with bio-plausible algorithms,

in [139] it is shown that this saturation can be even beneficial.

Platform	Temporal data type	Tasks	Spike encoding	Learning Rule	Plasticity	Plasticity mediation	Reference	Comments
Simulation	Event-based	Classification NMNIST DVS Gesture	Rate	Three Factor; Gradient-based; Surrogate-gradient	Pure memristive	Direct pulse application	[124]	
Simulation	Static	Classification; MNIST	Rate	Gradient-based; Surrogate-gradient	Exact gradient	Write-verify	[127]	84% on MNIST; NbO neurons
Simulation	Static	Classification; MNIST	Rate	Gradient-based; Surrogate-gradient	Exact gradient	Write-verify	[128]	
Simulation	Static	Classification	Rate	Unsupervised STDP	STDP	Pulse shape	[130]	
Simulation	Static	Classification; MNIST	Time	Unsupervised STDP	Exact update	Write-verify	[129]	80% on MNIST
Mixed	Event-based	Temporal Correlation	Time	Unsupervised STDP	Pure memristive	Direct pulse application	[131]	
Simulation	Static	Classification; MNIST	Rate	Unsupervised STDP	STDP	Write-verify	[140]	
Simulation	Static	Classification; MNIST	Rate	Semi-supervised STDP	STDP	Direct pulse application	[136]	
Mixed	Static	Classification	Rate	Unsupervised Hebbian	STDP	1T1R gated pulses	[133], [134]	
Simulation / Mixed	Event-based	Sequence Learning	Time	Unsupervised Hebbian	Pure memristive	Direct pulse application	[137], [138], [141]	

Table 2.1.: **Memristive synapses for SNNs**

3. Experimental

3.1. Electrical measurements

Two types of electrical measurements are important in this thesis. The first being quasi-static current voltage sweeps is one of the standard tools for the characterization of ReRAM devices. The second type of measurement are potentiation and depression measurements that aim at replicating the gradual change of synaptic output currents upon consecutive learning stimuli.

The setup for both measurement is depicted in Figure 3.1 a). A voltage pattern is generated and applied to the memristive device. The voltage over the ReRAM device is always defined such that the potential is applied at the active electrode while the ohmic electrode is grounded. During the application of the voltage pattern the current is measured. Furthermore, the series resistance of the measurement setup is important to note. For most off-the-shelf measurement setups this series resistance is a 50Ω termination resistance to match the measurement setup to the attached coaxial cables. Since the memristive devices usually do not have a 50Ω resistance and feature no matching network, high-frequency measurements may suffer from signal reflections. But since the measurements performed in this thesis at maximum have pulse lengths of $100ns$ which corresponds to the low megahertz regime, it is assumed that this effect can be neglected.

3.1.1. Quasi-static current voltage measurements

Quasi static current voltage measurement yield some of the most basic characteristics of memristive devices like the SET and RESET voltages and the ON-OFF ratio. The procedure is depicted in Figure 3.1b). Two triangular voltage patterns, called sweeps, with opposite polarity are applied to the device one after another. The pattern can start with either a positive or a negative polarity sweep. Usually, the voltage polarity of the first sweep (indicated in red) is chosen such that it increases the conductance of the memristive device, e.g., causes a SET. The maximum voltage and the sweep rate are measurement parameters. For some, especially filamentary ReRAM devices, a current compliance is applied during the SET sweep to hinder the current through the ReRAM device from increasing over a defined magnitude I_{CC}. This parameter is crucial if the compliance is reached because it limits the applied voltage such that the demanded voltage value is not reached.

The second sweep with the opposite voltage polarity as the first sweep causes a decrease of the device's conductance, a RESET. Here, the maximum voltage, called

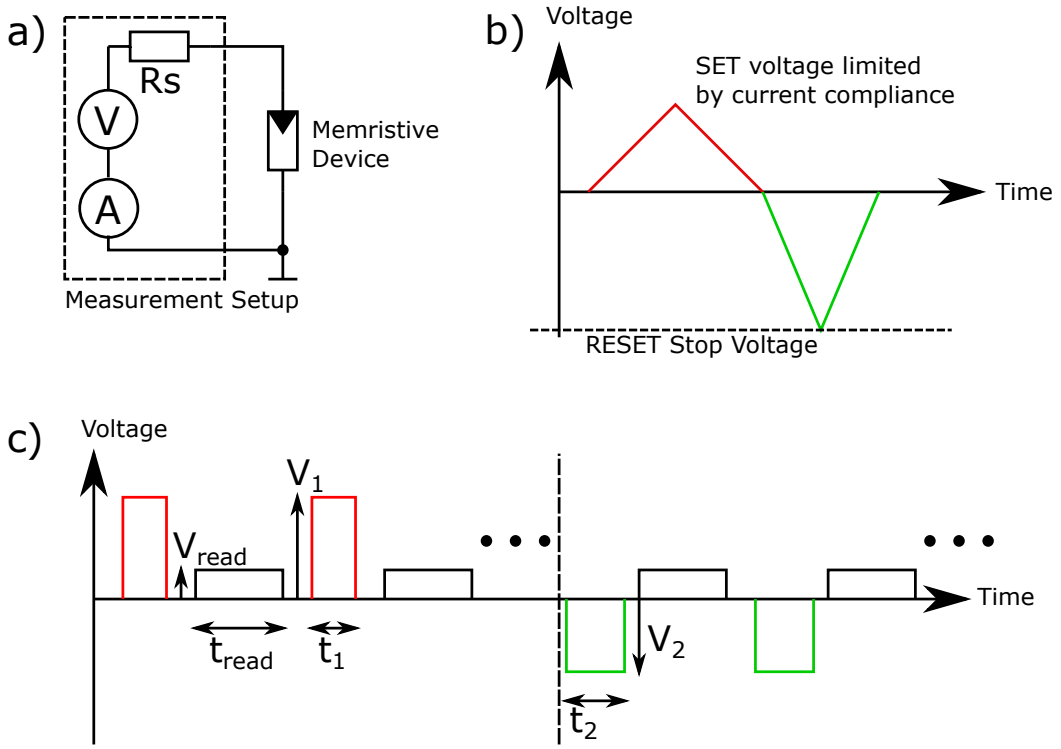

Figure 3.1.: **Overview of electrical measurements** a) Measurement principle. A voltage is applied to the memristive device from a setup which has an internal series resistance which is in in most cases a 50Ω termination resistance. In the single cell measurements, the voltage source simultaneously measures the current. For the measurements on the MAD200 demonstrator chip, application of voltage and current measurement are distributed in different setups. b) Quasi-static current voltage sweep measurements consist of two triangular voltage pulses with opposite polarity, which are described by their maximum voltage V_{POS} or V_{NEG} and the sweep rate. In the SET direction, the current compliance I_{CC} is important. c) Potentiation and depression measurements are conducted by applying a train of pulses with alternating stimulus and read pulses. The stimulus pulses have an amplitude of V_1 and a length of t_1 and further switching conditions like a gate voltage for 1T1R devices can be applied. READ pulses have a height of V_{READ} and a length of t_{READ} and further conditions are set such that the read out of the memristive conductance is disturbed as little as possible (i.e. transistor gates are fully open). A complete potentiation / depression cycle furthermore consists of a second pulse train with stimulus pulses with a height V_2 and length t_2. Read pulses remain the same.

RESET stop voltage, and the sweep rate are defining parameters.

The quasi-static current voltage measurements shown in this thesis are performed in a setup of commercial tools. A Labjack is used to generate the voltage patterns which are routed by an Aixacct S2 switch matrix and a custom printed circuit board (PCB) and the routing circuitry on the MAD200 vector-matrix multiplication accelerator to the devices. The current measurements are also performed by the Labjack. The setup is controlled in a custom MATLAB program.

3.1.2. Potentiation and depression measurements

Another important characterization method, especially for ANN and neuromorphic applications, are pulse train switching experiments. These are also called potentiation and depression measurements because they resemble the gradual increase and decrease of synaptic currents upon repeated stimuli.

Experimental procedure and results

A potentiation and depression measurement begins by bringing the memristive device to a defined conductance, for example by a quasi-static sweep measurement, followed by two pulse train experiments with opposite voltage polarity. Each pulse train consists of stimulus pulses with V_1 magnitude and t_1 length and read pulses with V_{read} magnitude and t_{read} length. The doublet of pulses repeats n times as depicted in Figure 3.1 c). The stimulus pulse causes a change of the device conductance while the read pulse determines the resistive state but should not affect it. The stimulus pulses of the second pulse train feature a potentially different magnitude of the opposite voltage polarity V_2 and can have a different pulse length t_2. The read pulses are the same as in the first pulse train. Usually, the second pulse train consists of the same number of pulses as the first. For some ReRAM devices a current compliance is applied in the SET direction to protect the device under test from damage due to large electrical power dissipation when a device changes to a high conductance.

Typically, the pulse shape of the stimulus pulses is chosen such that the conductance changes gradually in the SET (potentiation) and RESET (depression) direction. In Figure 3.2 typical measurement results of pulse train experiments are shown for both a system with current compliance (a)) and a system without compliance (b)). If no current compliance is applied, each measurement determines the starting resistance of the next. Therefore, the potentiating and the depressing pulse train experiments have to be considered together. If a current compliance plays a role, this will determine the final resistance independent of the starting resistance. In this case, each measurement can be considered by itself.

The measurements in Figure 3.2 show each a gradual change of resistance. The potentiation measurements (in blue) have to be read from left to right with pulse 0 yielding the conductance before the first pulse and pulse 100 the conductance after the final pulse. The depression measurements, if plotted together with the potentiation

go from right to left. Here, pulse 100 is the conduction before the first pulse of the measurement and pulse 0 the value after the last pulse. For potentiation and depression measurements on the MAD200 vector-matrix multiplication accelerator, the same setup as for the quasi-static current voltage measurements is used. For all other samples, a commercial ArC ONE memristor measurement board is employed which is controlled in a custom Python environment.

Evaluation aspects of potentiation and depression measurements

ON-OFF ratio An important parameter of a pulse train measurement is the ON-OFF ratio. For measurements with an effective current compliance, this value is defined as

$$r_{\text{ON-OFF}} = \frac{max(g_{\text{start}}, g_{\text{end}})}{min(g_{\text{start}}, g_{\text{end}})} \tag{3.1}$$

the final conductance g_{end} divided by the initial conductance g_{start} for the SET direction and the inverse of this value for the RESET direction. For measurements where both potentiation and depression direction are considered, e.g., such without effective current compliance, the final conductance of the potentiation divided by the final value of the depression yields the ON-OFF ratio.

$$r_{\text{ON-OFF}} = \frac{g_{end,pot}}{g_{end,dep}} \tag{3.2}$$

The ON-Off ratios achieved in potentiation/ depression measurements need not necessarily be the same as in quasi-static IV measurement. Usually, in pulse experiments smaller values are obtained.

Resolution The resolution in general is the number of distinguishable states in a measurement. Resolution can also be a local property of a measurement. In this case it means how large the difference in conductance between neighboring distinguishable states in the vicinity of a conductance value is.

Noise When performing multiple measurements with equal parameters, the conductance values for the same pulse number differ between measurements. This is called noise. As a measure of the noise, the average conductance $g_{\text{avg}}(p)$ of all measurements after a given pulse p is determined (see blue line in Figure 3.3) and the quadratic sum of differences between each actual measurement i $g_i(p)$(pink dots) and this average for each pulse number is calculated. The square root of the value is drawn to account for iffthe quadrating of the error. This method calculates the sum of the Mean-Square-Root (MSR) differences between a particular measurement cycle and the average all cycles.

$$\sigma_{\text{abs}} = \sum_{\text{i}}^{N_{\text{meas}}} \sqrt{\sum_{\text{p=0}}^{p_{\text{max}}} g_i(p) - g_{\text{avg}}(p)} \tag{3.3}$$

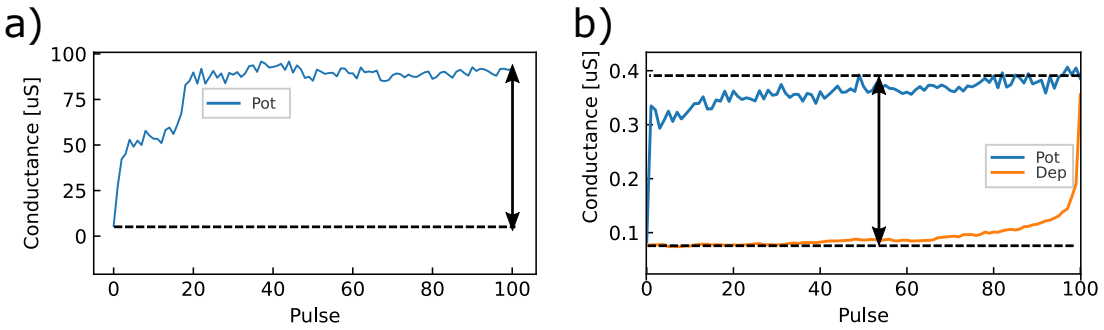

Figure 3.2.: **Definition of the ON/OFF ratio** a) in current limited systems: Ratio between start and end conductance. b) in systems without current compliance: Ratio between the end conductances of both pulse trains.

To allow for comparison between different measurements in the same resistance regime, this value is divided by the number of pulses per measurement $p_{\max}$ times the number of measurements N_{meas}.

$$\sigma_{\text{norm}} = \frac{\sigma_{\text{tot}}}{p_{\max} * N_{\text{meas}}} \tag{3.4}$$

To make comparison between different resistance regimes, each difference between average and actual value must be divided by the average's value at the pulse number. Then this relative noise is accumulated.

Linearity In the beginning of a measurement, the evolution of conductance is faster than in the end where a saturation can be observed. This means that the conductance step measured for different absolute conductances is different. If the step was independent, the potentiation and depression would result in a straight line. Therefore, this effect is coined non-linearity.

To get a measure for the linearity, the integral (more exactly the sum at each pulse number) over the difference between the average conductance of a set of measurements and a linear function between the first and last conductance is calculated. To make the linearity comparable between different measurements, this value is normalized by the number of pulses and the value of the average.

3.2. Software frameworks for artificial neural network simulations

The neural network simulations are conducted with the Python frameworks of either Tensorflow or Torch. In the following, the general setup of an artificial neural network simulation in both networks is described and it is shown how memristive update dynamics are included into the simulation.

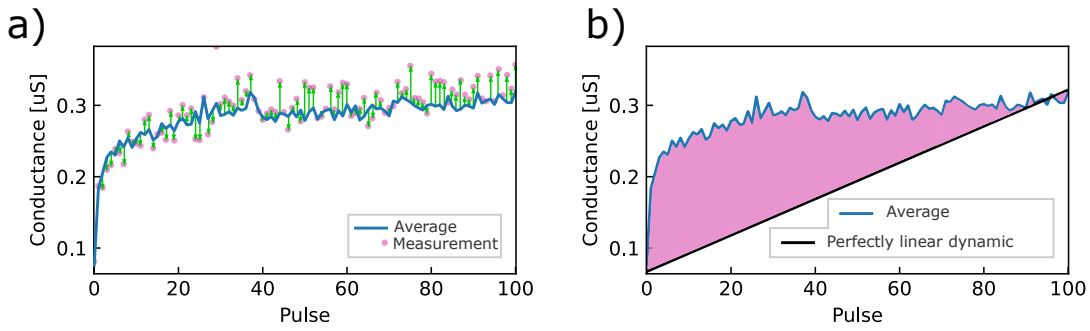

Figure 3.3.: **Definition of noise and non-linearity in potentiation and depression measurements** a) Noise is the difference between a single measurement cycle (pink dots) and the average of multiple with the same parameters (blue curve). b) Non-linearity is the accumulated difference between an actual measurement and the direct (linear) connection of its start and end conductances.

Figure 3.4 shows the different modules and their interaction when simulating a neural network. The information flow during the forward pass or inference is drawn in red while blue arrows represent the flow of information in the backward pass. The actual network consists of a series of weight and activation layers. At the beginning of a forward pass, a data loader chooses one or a batch of samples of the data set and sends it to the network. The network propagates the signal through weight and activation layers. The result of the weight layer is forwarded to an activation layer which applies an activation function to the weight layers output vector and forwards the result to the next weight layer or, if it is the final layer of the network, to a max pooling function which yields the index of the output's maximum value. This is the result of the classification and indicates the class the networks input is predicted in.

If the training of the network is simulated, an error signal is generated and backpropagated through the network. Therefore, the network's output, i.e., the last activation layers output, is processed by a loss function. This function receives a target output vector (or a batch of) from a teacher which knows the correct label(s) of the currently processed (batch of) input sample(s). The result of the loss function which is the classification error is backpropagated through the backward functions of the network's layers.

The structure of the weight layers for both the Tensorflow and the PyTorch frameworks are similar but eventual memristive dynamics are applied at different positions.

3.2.1. PyTorch

The implementation of a weight layer in PyTorch as it is used in this thesis is depicted in Figure 3.5. An input to the weight layer is processed by the forward function. This would resemble a read operation in a memristive crossbar implementation. Therefore,

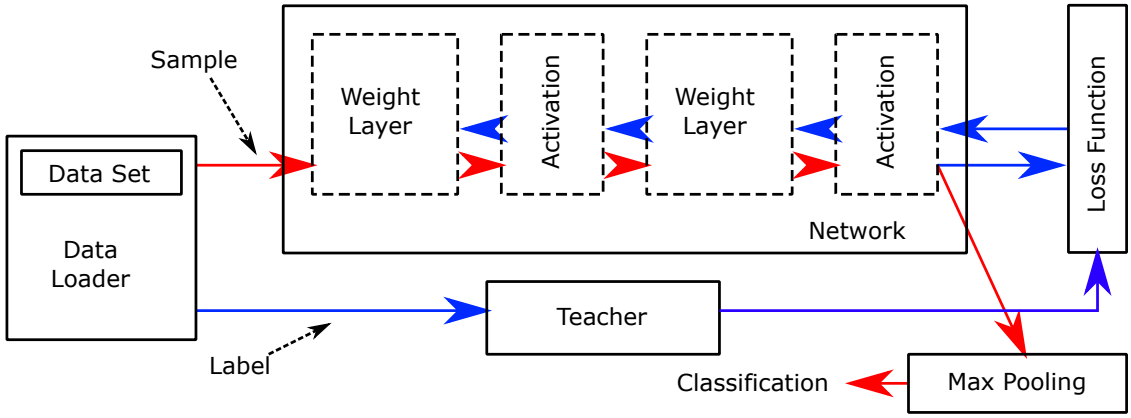

Figure 3.4.: **Network setup in PyTorch and Tensorflow** Formatted data from the data loader is input to the network with alternating weight and activation layers. In the inference case the index of the maximum output is the achieved classification (Max Pooling). In the training phase, a loss is calculated via a loss function with knowledge of the correct class by the Teacher. The error is then backpropagated through the network.

at this stage read noise and read non-linearity can be applied. Furthermore, if a custom backward function is implemented, any information necessary for this process is stored in a context object. For example, the matrices necessary for the compensation of the update dynamics described in section 5.4 are calculated during the forward pass, save in the context object, and later applied during the backward function.

A custom backward function receives the gradient with respect to the output of the next layer $l+1$ during the backward pass in the network. With knowledge of the weight matrix and eventually information stored in the context object, the gradient with respect to the weight matrix can be calculated. At this point it is possible to apply any compensation mechanism for the memristive update dynamics. Furthermore, the backward function calculated the gradient with respect to the input of the layer. This is forward to the previous layer $l-1$.

The calculated weight gradient is used by an optimizer to update the values of the weight matrix. In the PyTorch implementation, the optimizer implements any perturbation of the update by memristive dynamics. In this thesis, PyTorch is used in the discussion and development of the memristive switching dynamic compensation for memristive ANN applications.

3.2.2. TensorFlow

The training with PCMO-based memristive device as weights discussed in section 5.3.1 is performed in a Tensorflow framework. The implementation of the weight layers in this framework is similar to PyTorch but has some differences. In the Tensorflow

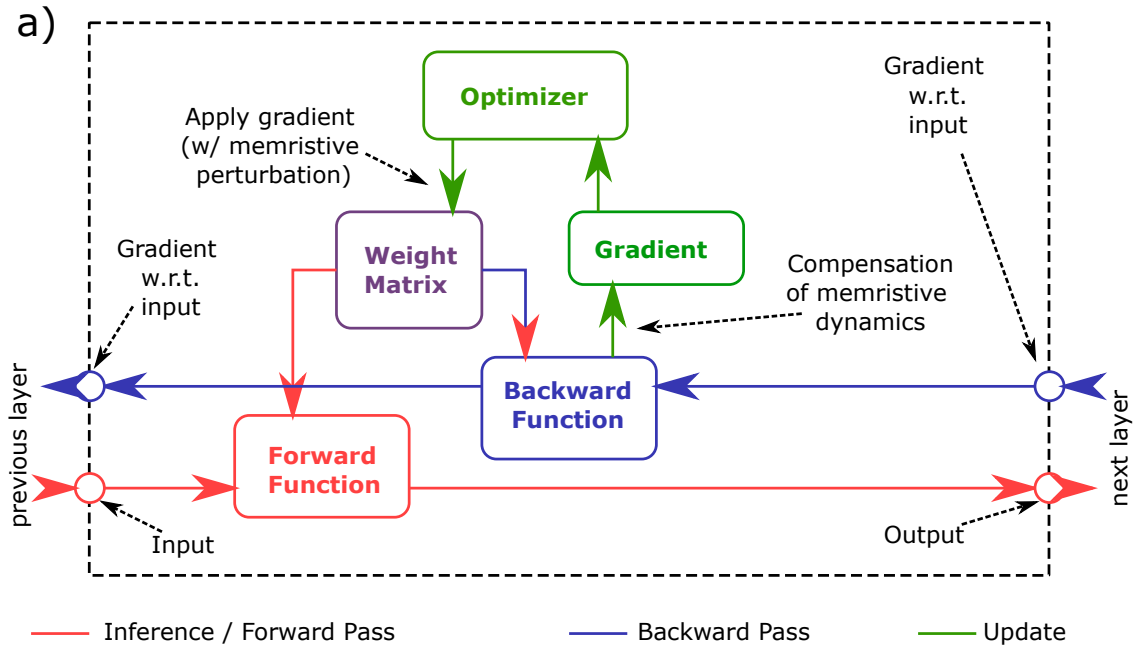

Figure 3.5.: **Implementation of a weight layer in PyTorch** The two directions of data flow are indicated in red (forward pass) and blue (backward pass). In the forward pass the layer receives input from the previous (activation) layer and calculates an output signal with the Forward Function. Usually, this is a VMM of the input with the weight layer, but it can further be used to implement the compensating neuron model (see Section 5.4.1). In the backward pass, the neuron calculates a gradient with the error backpropagated from layer $L+1$ and emits an error signal to the previous layer $L-1$. The backward function can implement the Update Compensation discussed in Section 5.4.2.

simulations in this thesis, no custom forward or backward functions are employed. Instead, the gradient as calculated by the autograd function is permutated in the optimizer before it is applied to the weight layer. A TensorFlow environment is employed in the investigation of the Gradient Sign Rule.

4. A memristive Compute-in-Memory demonstrator for neural network applications

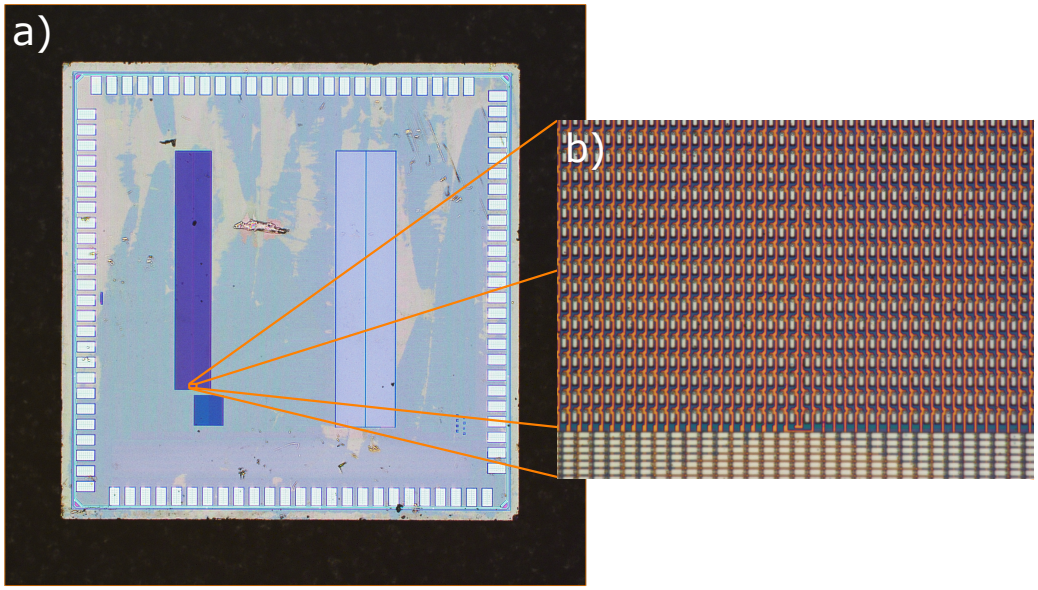

Figure 4.1.: a) Micro photograph of a MAD200 chip. The dark areas are the realized crossbar arrays. b) Detailed view of the vertical 1T1R array structure. Bright turquoise pads are LETI metal5 and orange stripes are metal1 of the ST process.

Sophisticated physics-based compact models for memristive devices and powerful circuit simulators with accurate models for CMOS circuit elements put high confidence in simulations of memristive CMOS co-integrated circuits. However, for multiple reasons it is necessary to validate any simulation result with an experiment on real fabricated hardware. Therefore, in this chapter the development and fundamental characterization of a demonstrator chip are shown which features different memristive CMOS co-integrated crossbar arrays. This structure of memristive memory is chosen because it is very general yet effective in the operation of digital and analog memory

schemes. The versatility of active 1T1R crossbar arrays is reflected in a number of publications in literature [103].

There are different schemes of active memristive crossbar arrays, vertical 1T1R arrays and pseudo-crossbar arrays. Since it is not clear which type of array can be used best for which application, both structures are designed in two separate arrays on each chip. Additionally, there are several test structures added that are not connected to the array circuits to test and eventually debug signal pathways. A small experimental 2T1R array can be used to directly compare the two array organization styles.

The chip is designed and taped-out in a 130nm CMOS technology node by ST Microelectronics and the memristive TiN/HfO devices are fabricated back-end-of-line (BEOL) by CEA LETI. A microphotograph of the chip can be found in Figure 4.1a) with the metal layers of the BEOL integrated memristive devices visible in the inset 4.1b).

4.1. Concept and circuitry

The implementation and layout of the circuitry described below was developed together with Tobias Ziegler (IWE2, RWTH Aachen University).

The main goals for the development of the CIM demonstrator chip are versatility and simplicity of use. It is not intended to design a particularly fast or energy-efficient system. Both these conditions could only be met by a complete System-on-Chip (SoC) design. However, the design and debugging of such a system renders much more complicated than the realization of key elements of a memristive CMOS co-integrated circuit, in this case active memristive crossbar arrays, alone. This requires a compromise between the amount of circuitry that is designed on chip and functionality which has to be performed off-chip. Since analog & mixed signal circuits in general require the highest amount of debugging, the generation and measurement of analog signals are completely performed off-chip. On the one hand, this of course adds noise to the signal and slows all operations down to the speed of the external measurement setups. But on the other hand, this approach reliefs the need for complex ADC and DAC circuits on chip which would have made the design more complex and prone to additional error sources. Furthermore, the external measurement setup can be tailored to the concrete application.

To allow a parallel measurement of all outputs simultaneously, all quantities that should be measured, like output currents of the memristive crossbar arrays, must have separate I/O pins of the chip. If this was done also for input signals, like source and gate voltages, the number of necessary I/O pins of the chip would increase drastically. On a chip with 100 I/O pins, only a single array with for example 33 source, 33 word, and 33 bit lines could be realized. Since these 100 I/O pins could only be located at the circumference of the chip, the chip area would have to be large and yet still

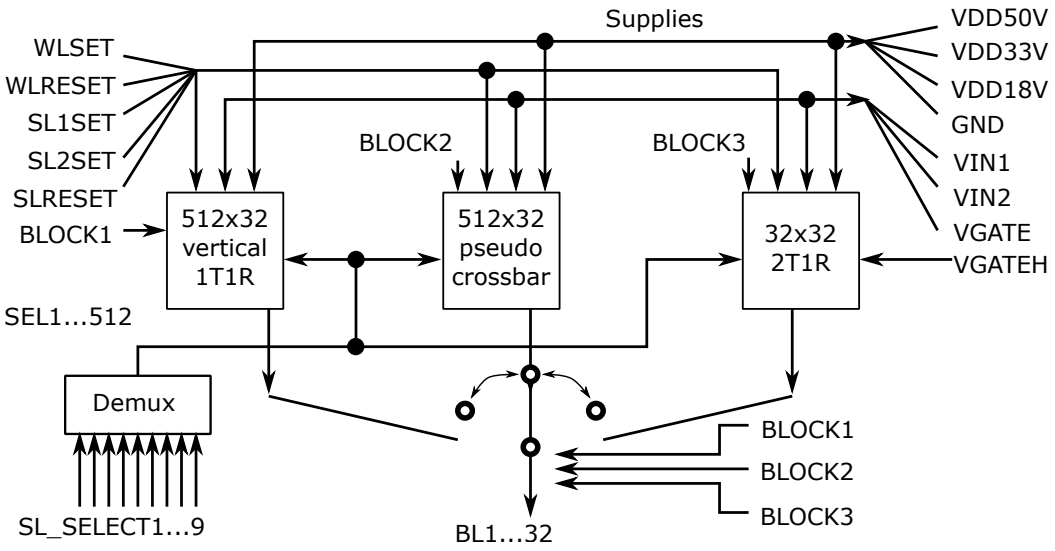

Figure 4.2.: The three active crossbar arrays with a vertical 1T1R (512x32), pseudo-crossbar (512x32), and 2T1R (32x32) receive digital inputs (WLSET, WLRESET, SL1SET, SL2SET, SLRESET), analog inputs (VIN1, VIN2, VGATE, VGATEH (only 2T1R)), and supply voltages (VDD50V, VDD33V, VDD18V, GND). The BLOCK signals determine which array can be addressed and which array's bit line output are connected to the chip output.

end up mostly unused. This pin mapping strategy however would yield the largest flexibility in the use. Nevertheless, in the present chip, it is decided to only connect output signals directly to I/O pins. The source line and word line signals are applied via custom designed digital / mixed-signal circuitry and can be chosen from different externally generated analog signals. For each source line and word line separately one of these signals can be applied. This increases the number of possible source and word lines drastically. In the following, the major components of the chip circuit to realize this approach are discussed in more detail.

Circuit overview An overview of the circuit taped out on the MAD200 demonstrator chip is shown in Figure 4.2. The core element are three active memristive crossbar arrays. Two of these arrays have 1T1R devices and differ by the way the transistor gates are controlled. Both feature a 512-by-32 1T1R device matrix with 512 source line and 32 bit lines. The vertical 1T1R array has 32 word lines, while the pseudo-crossbar has 512. The third array is a smaller 32-by-32 matrix of 2T1R devices. Details on the different array organization structures can be found in 4.1.1. All arrays receive the same externally generated analog signals "VIN1" and "VIN2", which can be connected to the source lines of the arrays, and "VGATE" which can be connected as a gate voltage of the transistor gates of the crossbar arrays. The 2T1R array additionally

receives "VGATEH" which is an additional gate voltage. "VGATE" is applied to one of the transistors in each 2T1R basis cell, "VGATEH" to the other. Furthermore, each array is connected to the same digital control signals. These are the select signals "SEL1...512", the word line control signals "WLSET" and "WLRESET", the source line control signals "SL1SET", "SL2SET", and "SLRESET", and the "BLOCK" signals which indicate the addressed array. The select signals are generated by a demultiplexer (DEMUX) with the "SL_SELECT1...9" signals and propagated to all arrays. These signals indicate to which source or word line an analog signal should be applied. By triggering the respective "SET" or "RESET" signal this analog signal is applied to the source or word line or removed. This process and the relevant circuitry are explained in greater detail in Section 4.1.2. The "BLOCK" signals differentiate in which array the signal attribution is changed. Lastly, the 32 bit line connections of all arrays are forwarded to I/O pins of the chip. By the respective "BLOCK" signals, analog transmission gates connect the array of interest directly to these pins. Furthermore, all arrays and circuits need supply voltages. In this case, three different supply voltages must be provided off-chip in addition to the ground signal "GND". "VDD50V" and "VDD33" are the $5.0V$ and $3.3V$ supply voltages which are used to operate the analog sections of the circuitry, while the $1.8V$ level "VDD18V" is mainly used in the digital circuits.

Array circuits The signal routing within each of the crossbar arrays is depicted in Figure 4.3. The memristive crossbar array is located in the center and indicated by a single 1T1R structure. The inputs of all 1T1R devices are connected to one of 512 (32 for the 2T1R array) source lines. Each source line individually can be connected to one of two input voltage "VIN1" or "VIN2" or set to a high impedance which resembles an open circuit. Therefore, the respective "BLOCK" signal is activated and the "SEL" signal of the source line of interest activated via the DEMUX. If now the signals "SL1SET" or "SL2SET" are triggered, the source line is connected to "VIN1" or "VIN2", respectively. The SL Select circuit does not allow to connect both signals at the same time to the same source line. This would result in a potentially harmful shortcut between the two input voltages. "SLRESET" removes any existing connection and sets the source line to the high impedance of an open transistor. The attribution of input signals is performed in a serial manner for all source lines. It is not possible to change the connected signal of multiple source lines in parallel. Nevertheless, this allows to apply the input voltages to all source lines in parallel. The externally generated input voltages are first applied when all connections to respective source lines are established. Therefore, no synchronization of the signals need be ensured on chip and the complex generation of precise voltages shapes can be performed by commercial signal generators off-chip.

Similarly, the 32 (for vertical 1T1R and 2T1R array) or 512 (for the pseudo-crossbar) word lines can all be separately connected to the external gate voltage "VGATE" or "GND" or "VGATEH" in case of the 2T1R array. Here, the same select lines SEL1...512 are employed to address the respective WL Select circuits. For the

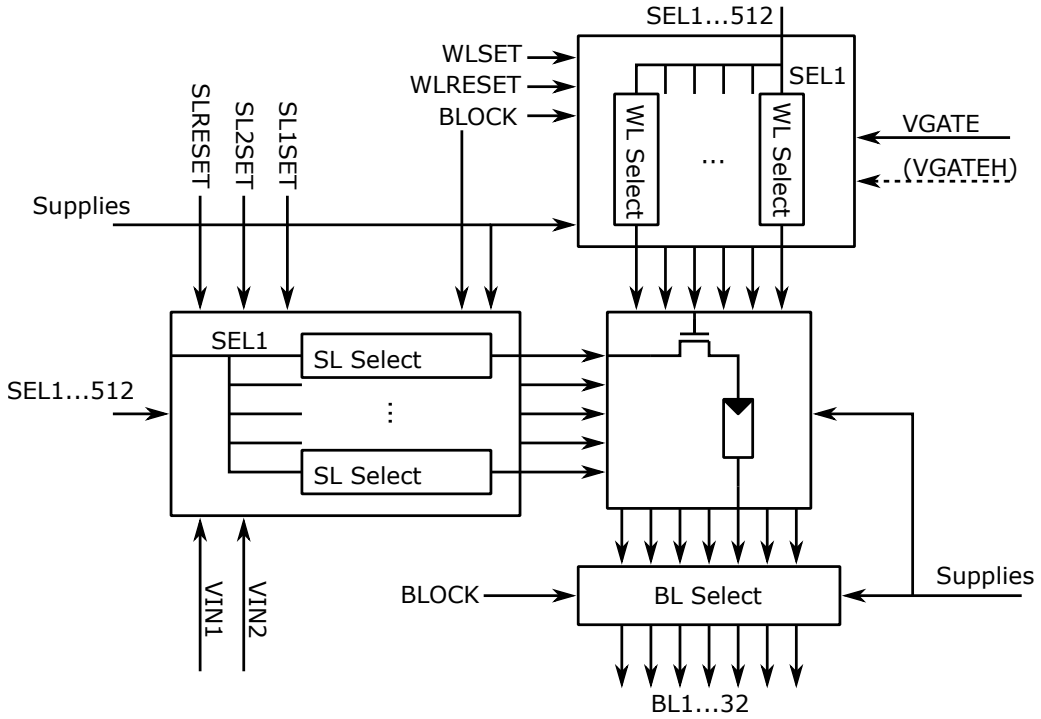

Figure 4.3.: **Array circuitry** The circuitry of each array block is organized around the actual memristive crossbar array (depicted with the 1T1R structure in the center). The source lines of the array are addressed by Source Line (SL) Select blocks which can be used to connect each SL individually to "VIN1" or "VIN2" or to high impedance by the "SEL1...512" and "SLRESET" / "SLSET1" / "SLSET2" signals if the appropriate "BLOCK" signal is HIGH. Similarly, each word line (WL) can be connected to an external "VGATE" (or "VGATEH" for the 2T1R array) signal via "SEL1...512" and "WLSET" or "WLRESET". The bit line (BL) Select block gates the bit lines to the chips I/O pads.

pseudo-crossbar array, all 512 select lines are used. For the vertical 1T1R array however, only the first 32 SEL lines connect to WL Select blocks. For the 2T1R array, the first 32 SEL lines are used to address the vertically connected transistors while the next 32 SEL lines (numbers 33 to 64) can be used to connect the horizontally connected transistors to either GND or the second external gate voltage "VGATEH". In this way, the same SEL line leads to one SL Select circuit and potentially one WL Select circuit per array. If the SEL line is activated, the SLSET/RESET or WLSET/RESET signals decide which of the blocks is changed. A respective analog input signal stays connected to a source or word line even after the corresponding SEL line is no more active. The attribution of a signal is saved within the respective SL Select or WL Select block in a CMOS flip-flop. More details can be found in the extended description of the Select circuits below.

Finally, the 32 outputs of each array are gated by analog transmission gates in the BL Select block and if the respective BLOCK signal is active forwarded to the I/O pins of the chip.

4.1.1. Active crossbar arrays

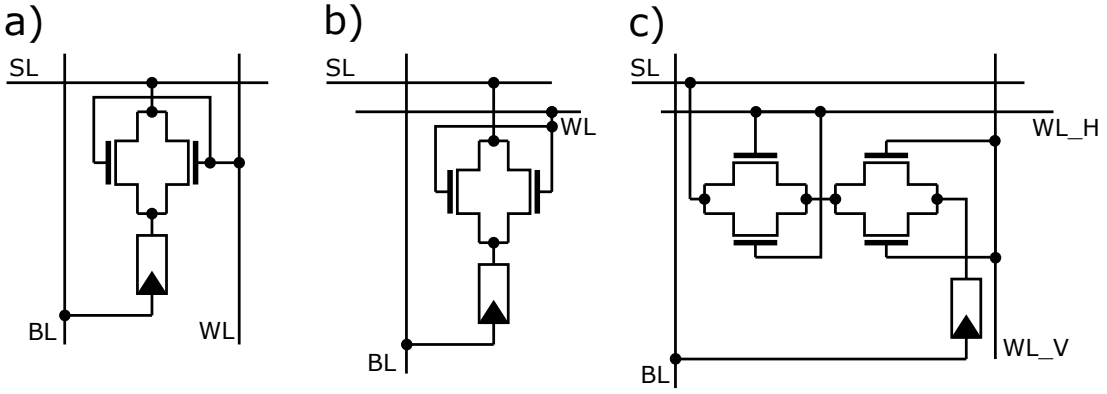

Figure 4.4.: **Unit cells of the three implemented arrays** a) Unit cell of vertical 1T1R-array. The access transistors are controlled via the word line (WL) running in parallel to the bit line (BL).
b) Unit cell of pseudo crossbar array. The word line (WL) controlling the access transistors runs in parallel to the source line (SL).
c) Unit cell of 2T1R array combining with two word lines, one (WL_H) in parallel to the source line (SL), another (WL_V) in parallel to the bit line (BL). The two access switches are in series.

The chip contains two 1T1R arrays with 512 source and 32 bit lines. The unit cell of both arrays consists of a transistor in series with a memristive device. These unit cells are organized in a matrix-like structure with devices on each horizontal line sharing one source line, which connects to the input terminal of the 1T1R circuit. Devices in the same column share one bit line, which is attached to the output of the 1T1R unit cell, forming a grid of perpendicular source and bit lines. However, the arrays feature different organizational structures when it comes to the connections of the transistor gates. In one array, all gates are connected along a column by one word line, perpendicular to the source line. This is the most common structure in the literature called "vertical 1T1R array". In the second array, all transistor gates along one line of the unit cell matrix are connected, in parallel to the source line. This structure was proposed by Qian and Yu and titled "pseudo-crossbar" [104] and claimed to be more energy efficient for vector-matrix multiplication (VMM) operations [105].
The circuits of the unit cells are depicted in Figure 4.4 a) for the vertical 1T1R array and b) for the pseudo crossbar. As one can see, the access transistor is actually a parallel circuit of two identical transistors. This circuit is chosen because the recommended transistor gate length of $6.67\mu m$ would cause a very lengthy shape of the

unit cell which would make the resulting arrays long as well and therefore harder to place on the silicon area. Instead of one long transistor two transistors with $3.35\mu m$ are placed in parallel. In the layout of the unit cells shown in Figure 4.5a) and b) this can be observed as two gates (red solid shape) sharing one terminal between them which is connected to the ReRAM cell on the higher metal layers. The ReRAM device stack is located in the red hollow shape. The size of the access transistors is therefore significantly larger than the actual memristive device stack.
A small third array with 32-by-32 cells features a combination of both connectivity schemes. The unit cell of this array therefore contains two access transistors (see Figure 4.4 c)) of which one is controlled by a word line perpendicular to the source line, like in the vertical 1T1R array, while the other is controlled by a second word line parallel to the source line, imitating the pseudo-crossbar. Similar to the 1T1R and pseudo-crossbar basis cells, each access switch is carried out as two transistors in parallel. The area of this unit cell is larger than the two others. In Figure 4.5 c) the total of four transistor gates (red solid shapes) can be seen.

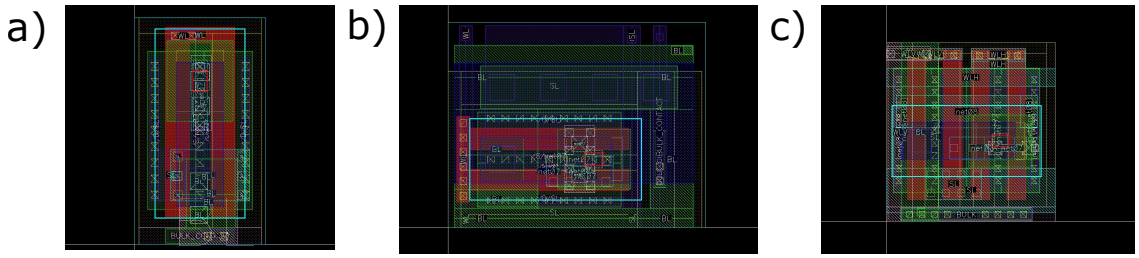

Figure 4.5.: **Layouts of array unit cells** a) vertical 1T1R array. b) pseudo-crossbar array. c) 2T1R array. Red solid planes are gates of access transistors. The memristive stack is located in the red hollow rectangle.

4.1.2. Routing logic and address circuitry

Word line control circuit

The same voltage is applied to all transistor gates along a word line. This voltage either is an externally generated gate voltage at $VGATE$ or zero volts at the GND potential. This way, access transistors can either be opened with a specific voltage or closed completely.
A word line control circuit, as depicted in Figure 4.6 a) for each word line, saves if either the externally generated voltage or GND are applied to its connected word line. If the respective $BLOCK_SELECT$ and $LINE_SELECT$ signals are $HIGH$, the flip-flop in the control circuit can either be set by applying a $WLSET$ signal or reset by applying a $WLRESET$ signal. If the flip-flop is set, its output Q opens a buffered analog switch, depicted in Figure 4.6b), connecting the respective word line to the externally generated voltage signal $VGATE$. If the flip-flop is not set, $\overline{Q}$ is $HIGH$

and activates an equal switch connecting the word line to GND closing all connected transistor gates by pulling their gates to zero volts.
The appropriate gate voltage configuration is saved by the control circuits of the activated array in a sequential manner by iterating over all word lines. Afterwards, the external gate voltage $VGATE$ is generated and all transistors receive their respective gate voltage at the same time.
The control logic, namely the AND gate and the flip-flop operate with 1.8V, but the gate voltage can be up to 5V. The 5V transistors (with 5V bulk voltage) forwarding the gate voltage cannot be fully opened by 1.8V. Therefore, a buffer is necessary to scale the 1.8V control signal from the flip-flop to 5V. This buffer is shown in Figure 4.6 b) and carried out as two inverter stages replicating the truth value of the 1.8V input. In the first stage, the 1.8V signal controls an inverter which supply voltage is 3.3V. This intermediate 3.3V signal then can control the second inverter stage with 5V supply voltage. This buffer then controls a 5V transistor.

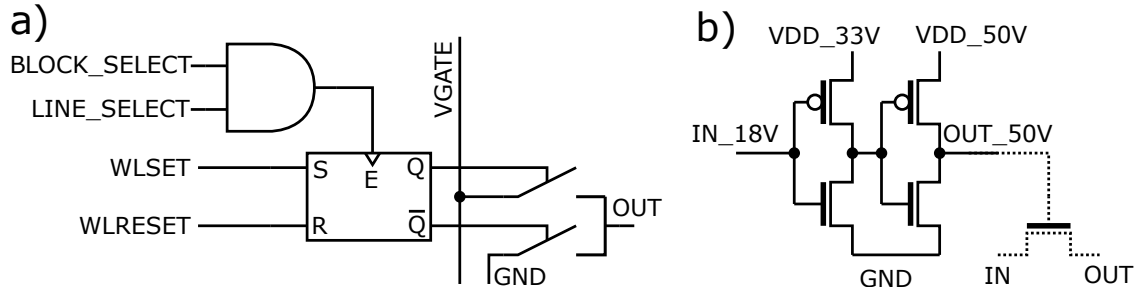

Figure 4.6.: **Word line control circuit** a) The word line control circuit employs a flip-flop which can be either set or reset if the respective line and block are selected. It controls an analog buffered switch, forwarding the applied gate voltage. b) Analog buffer circuit scaling a control signal from 1.8V to 5V with an intermediate of 3.3V. In combination with a 5V transistor (in dashed lines), this circuit is used as an analog buffered switch, e.g., in a).

Source line control circuit

The source line control circuit depicted in Figure 4.7 a) follows the same functional idea as the word line control circuit. Instead of connecting the respective source line to either ground or an external voltage, the source line can be connected to one of two externally generated voltage signals $VIN1$ or $VIN2$ or the source line can float. By activating respective $BLOCK_SELECT$ and $LINE_SELECT$ signal, two flip-flops are enabled. By now activating the signal $SLSET1$, one flip-flop can be set. The same applies for the signal $SLSET2$ and the second flip-flop. Via two OR gates, the set of one flip-flop causes a reset of the other. This makes sure that not both

externally generated signals are connected which would cause a short cut. Both flip-flops can be reset together by applying the $SLRESET$ signal. The flip-flops control buffered analog switches as described above and depicted in Figure 4.6 b) forwarding the external voltage signals $VIN1$ or $VIN2$. If neither flip flop is set both switches are closed and the output terminal OUT at high impedance, causing the connected source line to float.

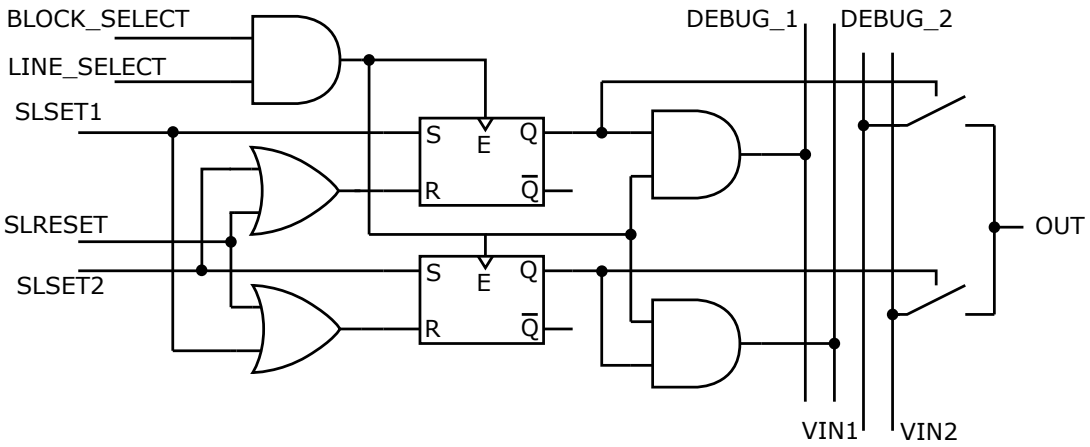

Figure 4.7.: **Source line control circuit.** If appropriate $LINE_SELECT$ and $BLOCK_SELECT$ are applied, the two flip-flops can be set by either setting the $SLSET1$ or $SLSET2$ signal. Activating one of these signals will reset the other flip flop to avoid a short cut. The $SLRESET$ signal deactivates both flip-flops, leading to a high impedance at the output.

Bit line control circuit

All 32 bit lines of the three arrays are connected to IO pins. To make sure that arrays do not influence each other, the signals are gated with transmission gates which are activated by the respective $BLOCK_CONTROL$ signal as shown in Figure 4.8 a). The analog buffers transforming the 1.8V control signal to 5V are carried out as described above. Transmission gates are chosen here instead of single transistors because the bit lines act in two different roles. They can be outputs at which current or voltage is measured. In this case a (virtual) ground is connected to the bit line. On the other hand, a bit line can become a voltage input if a negative effective voltage is applied over a device as it is the case for the SET. Since both signals, zero volts or a high voltage for SET, must be transmitted with high fidelity, a single transistor switch would not be sufficient, as it would disturb the signal in either direction by the transistor's threshold voltage.
The transmission gates are carried out with parallel n-type and p-type transistors shown in Figure 4.6b). To allow high currents, this structure is multiplied four times

in parallel. Since these transistors connect directly to the IO pins, all transistors are IO types.

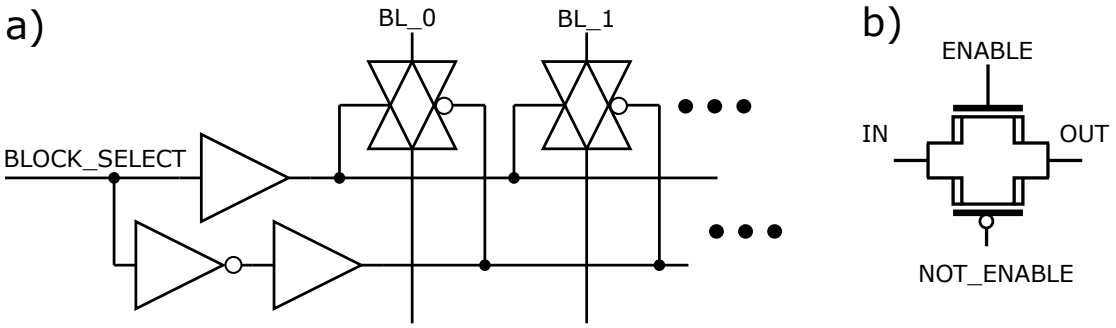

Figure 4.8.: **Bit line control circuit.** a) The $BLOCK_SELECT$ signal controls which array's bit lines are forwarded to the chips I/O pins by controlling transmission gates at each bit line. b) The employed transmission gates consist of an n-channel and a p-channel transistor. Both transistors are designed as I/O transistors because they directly attach to I/O pins of the chip.

Measurement setup

The peripheral measurement system described here was developed by Tobias Ziegler, Tyler Hennen, and Stefan Wiefels (IWE2, RWTH Aachen University).

To interface with the developed chip, a measurement system is set up, as depicted in Figure 4.9. This system consists of a routing printed circuit board (PCB), a LabJack data acquisition board, and a combination of a switch matrix and a micro controller. The whole system is controlled by a MATLAB environment.

The LabJack is used to generate the digital control signals for the chip and the analog gate and input voltages. Via the switch matrix it is possible to flexibly route each bit line output and the two input voltages $VIN1$ and $VIN2$ to either this analog input voltage, the global ground potential, or to a current measurement output. This current measurement is performed by the μC micro controller module of the Aixacct Tool. This module also controls the switch matrix.

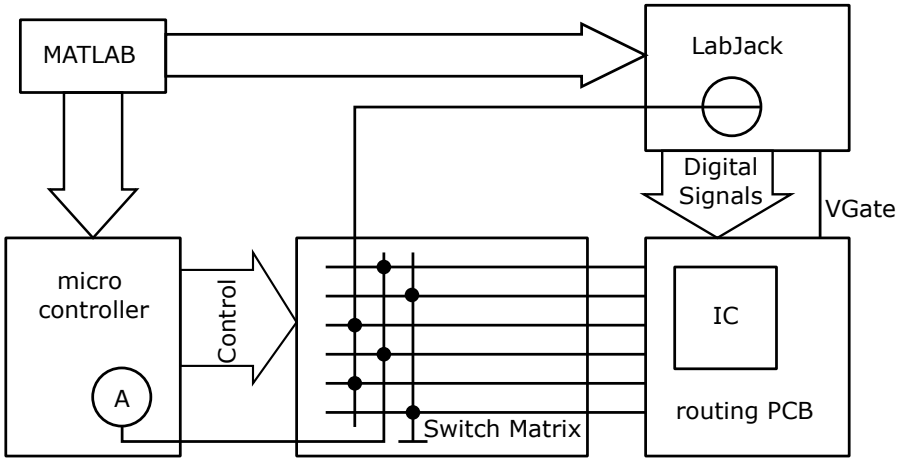

Figure 4.9.: **Peripheral measurement setup** A MATLAB script controls a LabJack board and the Aixacct Tool consisting of a micro controller and a switch matrix that allows flexible connection of signal generation and measurement ports.

4.2. Device characterizations

4.2.1. Quasi-static switching

The most essential switching experiments performed on memristive switching ReRAM devices are quasi-static current voltage sweeps as they are described in Section 3.1.1. In Figure 4.10, current voltage sweeps for memristive devices on the MAD200 memristive demonstrator are displayed. The experiments consist of three cycles of, firstly, one positive voltage sweep from zero volts to $V_{\mathrm{pos}} = 2V$ and back to zero which sets the device followed by a negative sweep to $V_{\mathrm{neg}} = -1.5V$ to reset the device again. During the positive SET sweep, the gate voltage of the series transistor is set to $V_{\mathrm{gate}} = 0.9V$ while during the RESET sweep the gate is completely opened with $V_{\mathrm{gate}} = 5.0V$. In the positive voltage sweep, the voltage is applied to the source line the device-under-test is connected to and the current read at the respective bit line, which is on GND potential. The application of negative voltages is not possible because the bulk contact of the access transistors is connected to ground. Therefore, the negative voltage sweep is executed by applying a positive voltage to the bit line and reading the current at the source line. This results in a negative voltage across the device.

The plot in Figure 4.10 displays five cycles of IV measurements. The black line is the average of all shown measurements. In the positive sweep direction between $0.5V$ and $1V$ all devices undergo a sharp SET event, switching from a low to a high conductive state. The current then saturates between $70\mu A$ and $80\mu A$ due to the transistor's saturation current. After reaching V_{pos}, the voltage decreases again toward zero. The current-voltage relationship becomes linear below $0.5V$. In the negative voltage sweep,

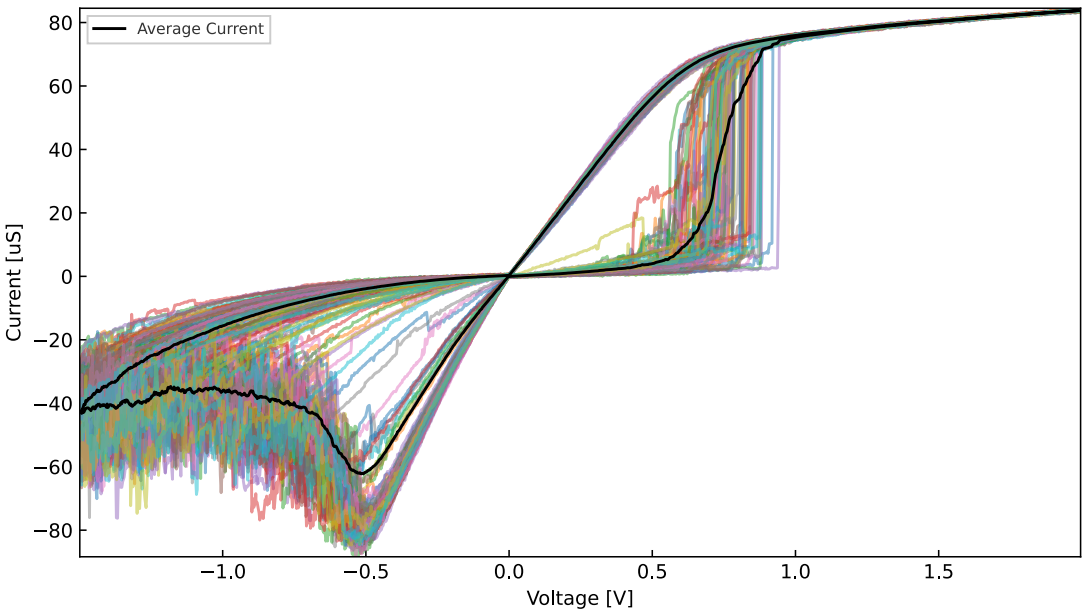

Figure 4.10.: **Quasi-static current voltage sweeps.** Measurement data is displayed for 36 devices for five cycles each. The black line indicates the average over all measurements.

this linearity initially persists before at around $-0.5V$ the RESET process starts, and the current magnitude decreases despite the voltage increasing. After passing V_{neg}, the current is significantly lower than in the high conductive state. Furthermore, the relationship between current and voltage exhibits a strong non-linearity in this region indicating that the potential barrier of the metal oxide interface dominates the conductance here.

The different colors indicate different measurement cycles. It can be seen that the SET voltage at which the abrupt SET event occurs in the positive sweep varies between $0.5V$ and $1V$, while most cycles show a switching at around $0.75V$. Furthermore, in the negative RESET direction, a variability of the low resistive states and of the RESET voltages can be seen.

Initialization with quasi-static voltage sweeps

In this work, quasi static current voltage sweeps are only used to initialize devices to a desired defined conductive state. In order to initialize different conductances, the current compliance in the SET voltage sweep can be used. In the 1T1R structure, this compliance is created by the saturation current of the transistor which depends on the applied gate voltage.

In Figure 4.11 the initialized conductances after current sweeps with equal V_{pos} and V_{neg} but different V_{gate} in a final SET sweep is shown. Between $0.6V$ and $0.68V$ gate voltage, the initialized conductance increases only slightly and the conductances

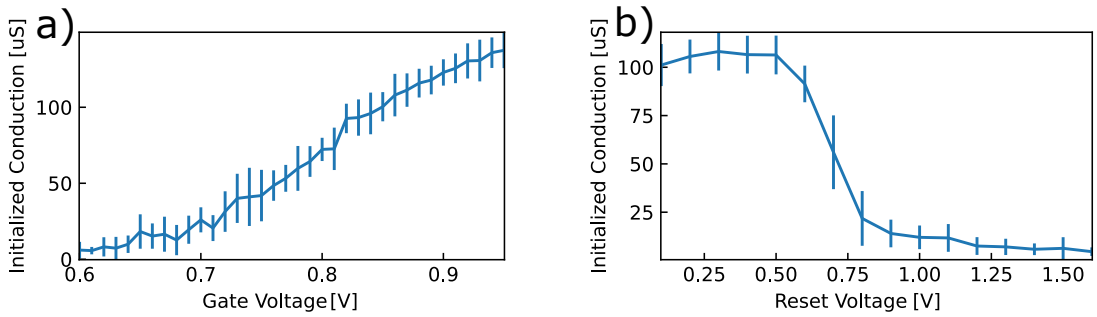

Figure 4.11.: **Device initialization with quasi-static voltage sweeps.** a) During the SET sweep the transistor gate voltage can be used to precisely initialize the device conductance. b) RESET voltage programming.

are low between $5\mu S$ and $20\mu S$. This indicates that here the gate voltage is too low in general to switch the device. From $0.7V$ to $0.9V$ gate voltage the initialized conductance increases from $20\mu S$ to around $130\mu S$ linearly. Above $0.9V$ the slope of conductance increase is lower.

This measurement depicts data from 36 different devices on different rows and columns of the array. The error bars show the standard deviation across these 36 devices. The device-to-device variation is in an acceptable range. Therefore, the variation of the gate voltage renders a practical method to initialize devices to a desired conductive state.

Another possibility is to initialize the conductive state by adjusting the maximum voltage in the last RESET sweep V_{neg}. Figure 4.11b) shows the resulting conductance after RESET sweeps with different V_{neg} for another 36 devices. Before, all devices are swept three times with $V_{\text{pos}} = 2.5V$ and $V_{\text{gate}} = 0.9V$ in the set sweep and $V_{\text{neg}} = -1.5V$ with $V_{\text{gate}} = 5.0V$. In the last RESET sweep, V_{neg} is changed to the values indicated in Figure 4.11b). For V_{gate} below $0.5V$ the initialized conductance plateaus at $100\mu S$ which is the ON state conductance. Here, no RESET takes place. From $0.5V$ to $0.75V$ the initialized conductance on average decreases sharply before saturating for higher V_{gate} below $25\mu S$. Compared to the initialization with the gate voltage discussed above, the standard deviation in the transition region in which the conductance can be controlled, is larger with the RESET voltage control. Therefore, in this work the initialization of conductances via the gate voltage is employed.

4.2.2. Potentiation & Depression Measurements

To investigate the gradual switching behavior of the fabricated devices, potentiation and depression measurements are conducted and the influence of different electrical parameters investigated. For the potentiation, these are the pulse length and the pulse amplitude. For the depression, additionally the influence of the gate voltage during pulsing is tested. For the potentiation direction a constant gate voltage of $0.9V$ is chosen. As shown in section 4.2.1, the gate voltage has a strong influence

on the maximum conductance that can be achieved during quasi-static switching. It can therefore be expected that a lower gate voltage would lead to in general lower conductive devices. However, for the sake of protection of the devices from thermal damage, no higher voltage than $0.9V$ is chosen.

All of the following measurements are performed on single devices to exclude a possible interference or effects from operations on larger arrays. Initially, the device is prepared with quasi-static voltage sweeps between $-2V$ and a gate voltage of $0.9V$ to limit the SET current, and $1.5V$ and $5.0V$ gate voltage in the RESET direction. Finally, the device is either reset with a sweep in preparation of a potentiation experiment or set to prepare it for a depression measurement. The potentiation or depression measurements each consists of 100 consecutive pulses with the same length and amplitude. Each experiment is conducted for five different devices.

Potentiation

For the potentiation direction, first the influence of the pulse length is investigated. Starting with the measurements setup's minimum pulse length of $30\mu s$, pulse lengths of $60\mu s$ and $135\mu s$ are investigated. The pulse length therefore approximately doubles from one measurement set to the next. The results are shown in Figure 4.12. The five different measurements are indicated by different colors. It can be seen that for a pulse train with $30\mu s$ pulses of $0.7V$ amplitude (shown in Figure 4.12a)), some devices switch immediately at the beginning of the measurement to a high-conductive state with only few intermediate conductance levels. The devices then saturate at the high conductance level. Other devices show no conductance increase. Only at the end of the measurement one more device switches to the high conductive state. The black line indicates the average of those devices that switch. When increasing the pulse length by a factor of 2 to $60\mu s$, as displayed in Figure 4.12b) a similar behavior can be observed with some devices switching immediately, some with delay and other not at all. When doubling the pulse length again to $135\mu s$ (see Figure 4.12c)), all devices switch over the course of the pulse train measurement but also here some are delayed. It should be noted that the conductance at which the devices saturate is similar for all pulse lengths at around $90\mu s$. In these measurements shorter pulses do not lead to more intermediate states during the potentiation, especially when comparing the results for the shortest pulses in 4.12a) and the longest pulses in 4.12c). These results indicate a stochastic switching mechanism with the probability of switching depending on the pulse length.

Next, the pulse voltage as a parameter is evaluated. Figure 4.13 shows results for three different pulse voltages ($0.5V$, $0.6V$, $0.7V$) with a common pulse length of $60\mu s$. In Figures 4.13a) and b) it can be seen that $0.5V$ and $0.6V$ pulse voltage are not sufficient to switch the devices. The conductance remains in its low initial state. For $0.6V$ in 4.13b) even a decline of the average conductance is observed which is due to some devices initially showing higher conductances than the initialized $5\mu S$. For a pulse voltage of $0.7V$ however, an abrupt immediate switching of some devices can be seen. At least in the investigate voltage range the step size of the conductance could

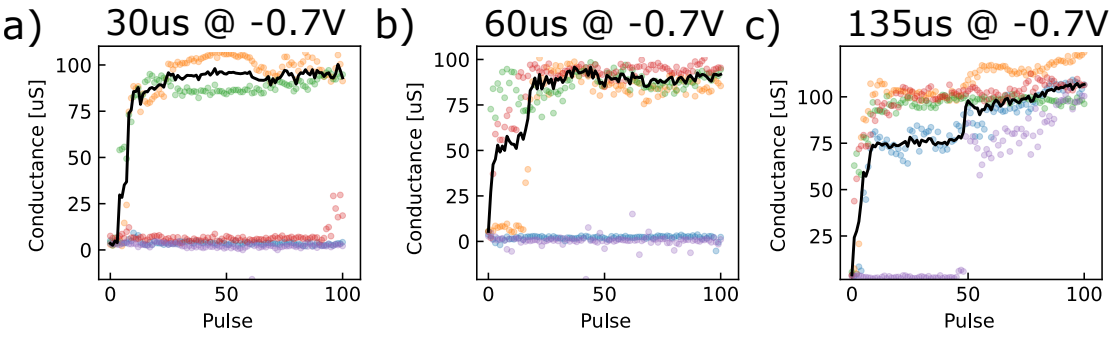

Figure 4.12.: **Potentiation Pulse Length Series** With increasing pulse length of $30\mu s$ (a)), $60\mu s$ (b)), and $135\mu s$ (c)) an increasing fraction of devices switches abruptly during the experiment. After a sharp SET all devices saturate at about the same high conductance, irrespective of the pulse length.

therefore not effectively be modulated by the pulse voltage.

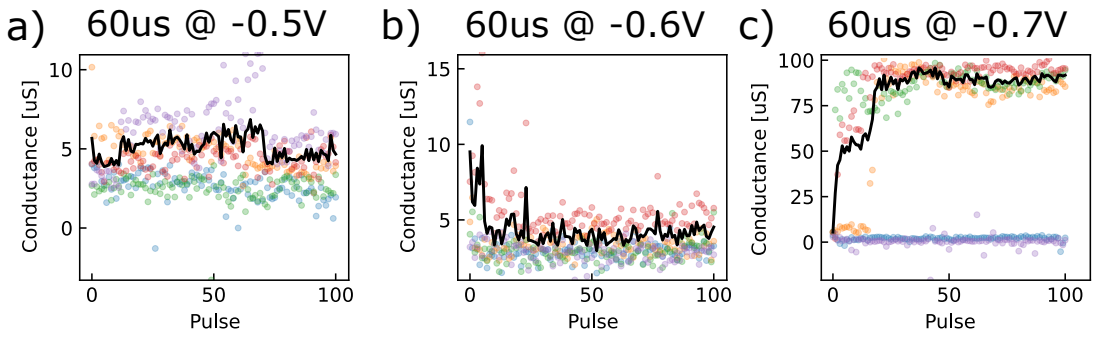

Figure 4.13.: **Potentiation Pulse Voltage Series** For the low voltages $0.5V$ (a)) and $0.6V$ (b)) no switching can be observed. With $0.7V$ (c)) pulse voltage the switching is stochastic and abrupt.

When analyzing the dependency of the average ON-OFF ratio and non-linearity as defined in Section 3.1.2, an effect of both the pulse length and pulse amplitude can be observed. The value for the ON-OFF ratio here also includes devices that did not switch during the measurement. As shown in Figure 4.14a) with increasing amplitude of the potentiation voltage, the average ON-OFF ratio increases. This seems to be in contrast to the measurements shown above. But for other pulse lengths some devices show switching at $0.6V$. However, at $0.5V$ pulse voltage no switching whatsoever can be observed, leading to an average ON-OFF ratio of 1. With the ON-OFF ratio, also the non-linearity increases gradually with the potentiation pulse voltage.

When looking at the influence of the pulse length on the ON-OFF ratio in Figure 4.14b) also a nearly linear correlation can be seen. It should be recalled that the saturation value the devices converge to during the potentiation is almost equal for all pulse lengths. But the average value here also takes the conductance of not switched

devices into account. As their fraction increases with the pulse length as discussed above, the average increases as well. Interestingly, the non-linearity has a maximum at a pulse length of $60\mu s$ and decreases again toward $135\mu s$. A possible explanation is that with $135\mu s$ long pulses, all devices switch during the measurement. Since some switch significantly later than the few that switch at the beginning, this increases the average conductance more linearly than for measurements where some devices do not switch at all, and the average conductance is dominated by those devices that switch right in the beginning.

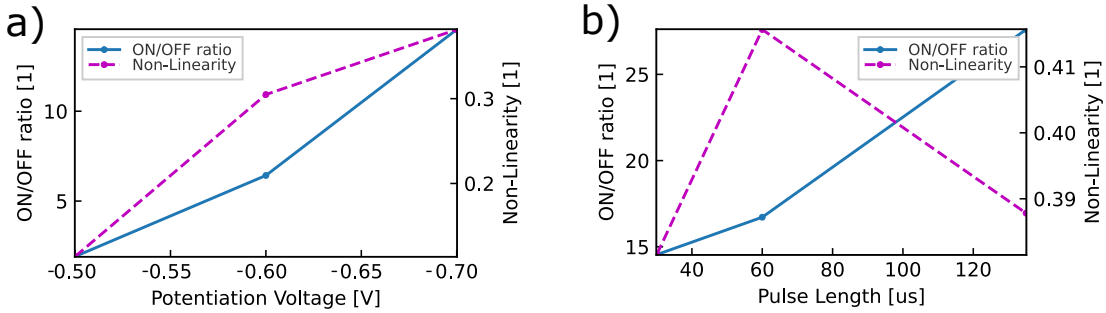

Figure 4.14.: **ON-OFF ratio and non-linearity for potentiation measurements** Influence of a) voltage and b) pulse length on ON/OFF ratio and non-linearity of potentiation pulse train measurements.

Depression

For the depression direction also at first the influence of the pulse length is investigated and measurements again with $30\mu s$, $60\mu s$, and $135\mu s$ pulse length and a constant voltage of $-0.5V$ are investigated. The gate voltage is always set to $5.0V$. In Figure 4.15a) an almost linear gradual decrease of the conductance can be observed for most devices. This linearity persists also for $60\mu s$ long pulses as shown in Figure 4.15b). With even longer pulses of $135\mu s$, the steepness of the conductance decreases toward the end of the measurement (see Figure 4.15c)). On the other hand, a higher total decrease of conductance is observed here. Also visible are the strongly differing traces of the different devices here, which are signs of a large device-to-device variability.

To test the influence of the depression pulse voltage, experiments with $-0.5V$, $-0.75V$, $-1.25V$ are conducted. For the comparison of the pulse voltages, $60\mu s$ pulses are applied while the transistor gates are completely opened with a gate voltage of $5.0V$. The results of the measurements are displayed in Figure 4.16. At $-0.5V$ pulse voltage in 4.16a), an almost linear decrease of the conductance over 100 pulses can be observed. All devices behave similar here in contrast to the potentiation direction, where a stochastic behavior was found. However, the conductance decreases from the initial average conductance of about $140\mu S$ only by about $40\mu S$ no even reaching an ON-OFF ratio of 2. This changes for a depression pulse voltage of $-0.75V$, as

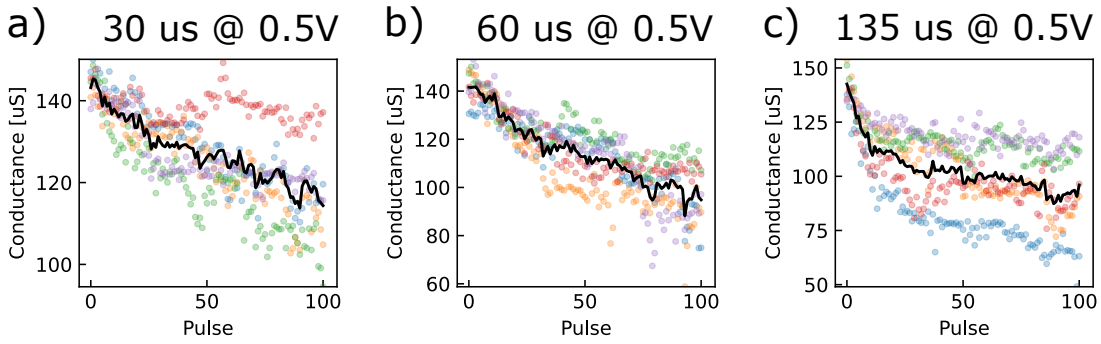

Figure 4.15.: **Depression Pulse Length Series** An almost linear gradual depression switching can be observed for $30\mu s$ (a)) and $60\mu s$ (b)) pulse length. For longer pulses of $135\mu s$ (c)) the switching is still gradual but steep in the beginning than at the end.

depicted in Figure 4.16b). Here, a sharp RESET of all devices is observed followed by a small gradual decrease. The ON-OFF ratio here is about 6 as the conductance decreases to about $25\mu S$ from the initial $140\mu S$. This small gradual continuation of the conductance decrease cannot anymore be seen for a depression pulse voltage of $1.25V$ in Figure 4.16c). All devices here switch immediately to a low conductance and remain there. The ON-OFF ratio is again higher than for $-0.75V$ depression pulse voltage.

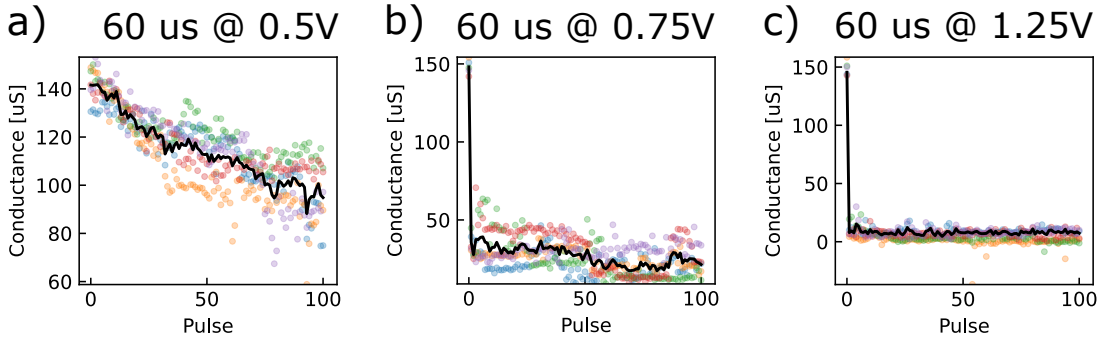

Figure 4.16.: **Depression Pulse Voltage Series** For a low depression pulse voltage of $-0.5V$ (a)) the depression is gradual, while for the higher voltages $-0.75V$ (b)) and $-1.25V$ (c)) an abrupt switching upon the first pulse can be seen.

For the depression measurements, also the gate voltage during is varied with values of $5.0V$, $2.5V$, and $1.5V$. The pulse parameters are fixed at a voltage of $-0.75V$ and a pulse length of $60\mu s$. For the two larger gate voltages no significant difference can be observed, when comparing Figure 4.17a) and b). A sharp immediate RESET is followed by a small gradual further decrease of the conductance. For a gate voltage of $1.5V$ in 4.17c) however, a more gradual depression switching is observed. At the

same time the conductance traces of the single devices show higher amounts of cycle-to-cycle noise.

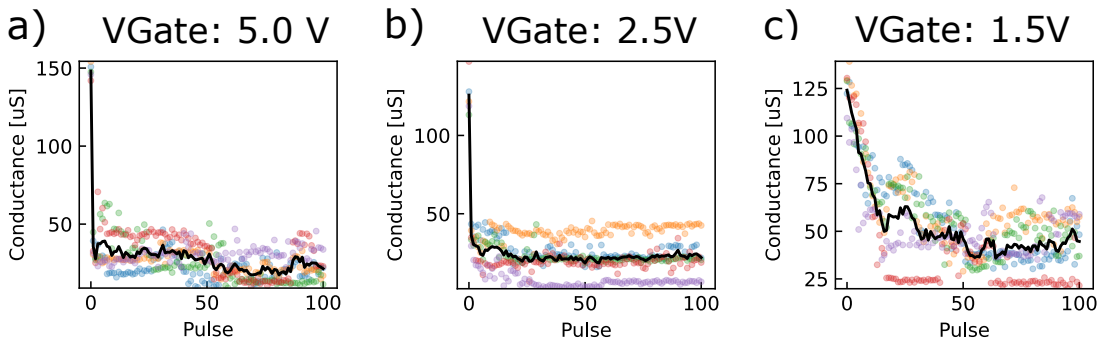

Figure 4.17.: **Depression Gate Voltage Series** By decreasing the gate voltage during depression pulse train experiments, the switching turns from abrupt and immediate for $5.0V$(a)) and $2.5V$(b)) to gradual for $1.5V$(c)).

With higher depression voltage magnitudes, higher ON-OFF ratios can be achieved, as shown in Figure 4.18a). Similar to the potentiation direction this goes together with an increase of the non-linearity of the pulse train switching. The pulse length can also be used to modulate the ON-OFF ratio and the non-linearity of the depression measurement. Here the same increasing trend can be observed as displayed in Figure 4.18b).

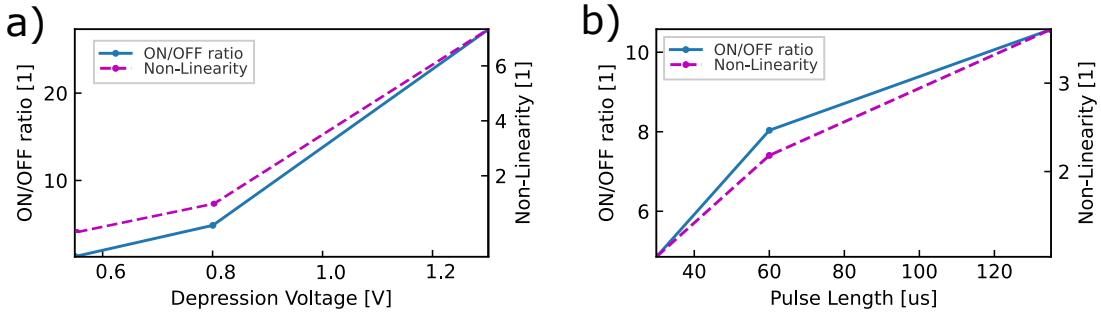

Figure 4.18.: **ON-OFF ratio and non-linearity for depression measurements** Influence of a) voltage and b) pulse length on ON/OFF ratio and non-linearity of depression pulse train experiments.

For a more detailed view on the influence of the pulse parameters on the ON-OFF ratio, the final ON-OFF ratio of all parameter sets is plotted in Figure 4.19 with pulse voltages on the x-axis, pulse lengths on the y-axis and the ON-OFF ratio color-coded. For both the potentiation and the depression it can be seen that the highest ON-OFF ratio can be achieved with the highest pulse voltage magnitude and the longest pulse duration. Respectively for the highest voltage a gradual increase of the ON-OFF

ratio with the pulse length is visible, when looking at Figures 4.19a) and b). On the voltage side a certain minimum voltage must be retained to still produce switching. Both trends can also be seen for the depression measurements with closed gates (at a gate voltage of $1.5V$). However, here the ON-OFF ratio in general is much smaller than with an open gate.

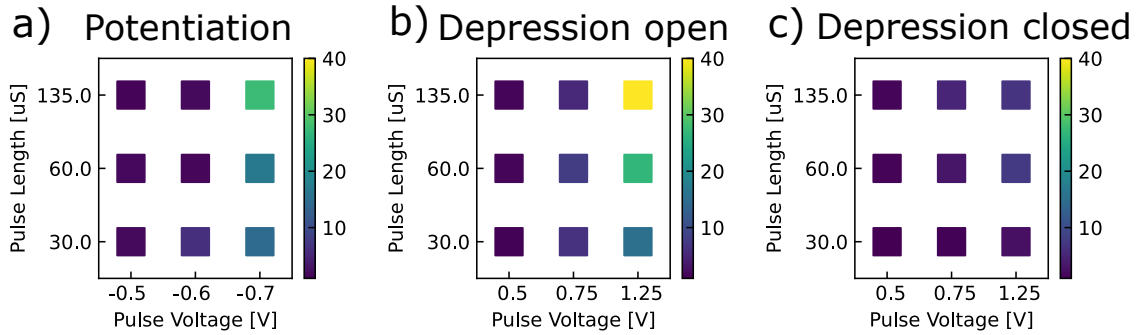

Figure 4.19.: **Pulse Parameter Dependency of ON/OFF ratio** Overview of the ON-OFF ratio by all combinations of pulse length and pulse voltage for the potentiation direction (a)), depression with open gates (b)), and depression with partially open gates (c)).

4.3. Array operations

The main benefit of matrix-like organization of memristive devices is the ability to perform compute-in-memory (CIM) operations which have the potential of increasing latency and energy efficiency in contrast to single device operations and computations in a von-Neumann style computer. CIM operations can either be used to read out the conductance matrix or to alter it. In the following, read-out and program operations are discussed and it is shown how they can be performed on the vertical 1T1R array of the MAD200 demonstrator chip.

4.3.1. Multiply and Accumulate (MAC)

The most essential CIM operation are matrix-level read-out, which essentially resembles a vector-matrix multiplication (VMM), and array-wide write operations, quasi-static sweep experiments as well as pulse-train mediated potentiation and depression measurements. The performance of these operations will be investigated in the following.

Parallel read-out

Parallel matrix read-out is performed by applying a read-voltage to the source lines of a (sub-)array with the transistor gates of this (sub-)array opened to the maximum by

applying $5V$ to the word lines. Then the memristive device at position (i, j) within this (sub-)array produces a current $i_{\mathrm{i,\,j}}$ due to the applied read voltage v_i and their conductance $g_{\mathrm{i,\,j}}$ according to Ohm's Law

$$i_{\mathrm{i,\,j}} = g_{\mathrm{i,\,j}} * v_i \tag{4.1}$$

These currents are then accumulated on the bit lines according to Kirchhoff's Law

$$i_{\mathrm{out,\,j}} = \sum_i \left(i_{\mathrm{i,\,j}}\right) \tag{4.2}$$

This operation can be expressed as a VMM of the input voltage vector $\underline{v}_{\mathrm{in}}$ and the conductance matrix C, yielding the output current vector $\underline{i}_{\mathrm{out}}$.

Issues specific to the VMM-like read-out of memristive crossbar arrays are the influence of line resistances and sneak paths. Read-out perturbations due to line resistances can occur if the voltage on the bit lines is not zero. In this case the voltage drop over a memristive device is not the applied read voltage but the difference between the read and the bit lines voltage. If the bit line voltage origins from the bit line's resistance, the bit line voltage will depend on the current flowing and the position of the respective device. In this case the perturbation due to line resistances cannot be estimated beforehand.

Sneak paths describe the phenomenon that in arrays of devices without rectifying behavior, i.e., without diode-like current-voltage curves, a device has at least one parallel resistance path if multiple source and bit lines are conductive. If a high resistance has multiple small resistances in parallel, the total resistance on the read-out large resistance will appear smaller than the three low resistances in parallel. In contrast to the line resistance issue, sneak path problems can be met with specific read-out schemes, especially in crossbars with 1T1R devices.

To investigate the impact of different read-out schemes a high conductive triangle of devices is programmed in a 10-by-10 subarray in the vertical-1T1R array of the MAD200 demonstrator, as depicted in Figure 4.20a). The read-voltage of $0.3V$ is applied in parallel to all ten source lines. First, a reference measurement is conducted, where each word line is opened separately, and the corresponding bit line is read out. In this scheme, no sneak paths can exist because only one device per source line is activated. The right graph in Figure 4.20a) shows the read out currents for the different bit lines. An almost linear increase of the current with an increasing number of high-conductive devices per bit line can be observed. The output currents with seven and nine high-conductive devices however overlap with their next lower neighboring measurement. The reason is that despite of equal initialization schemes for all high-conductive devices, the devices on the bit lines with seven and nine high-conductive devices have on average lower conductances than their next lower neighbors. The lowest read-out is around $20\mu A$ and the highest around $160\mu A$. With this each high-conductive device adds around $16\mu A$ to the read-out current. This fits well when looking at the increase of output current from bit line zero (without high-conductive devices) to bit line 1 (with 1 high-conductive device).

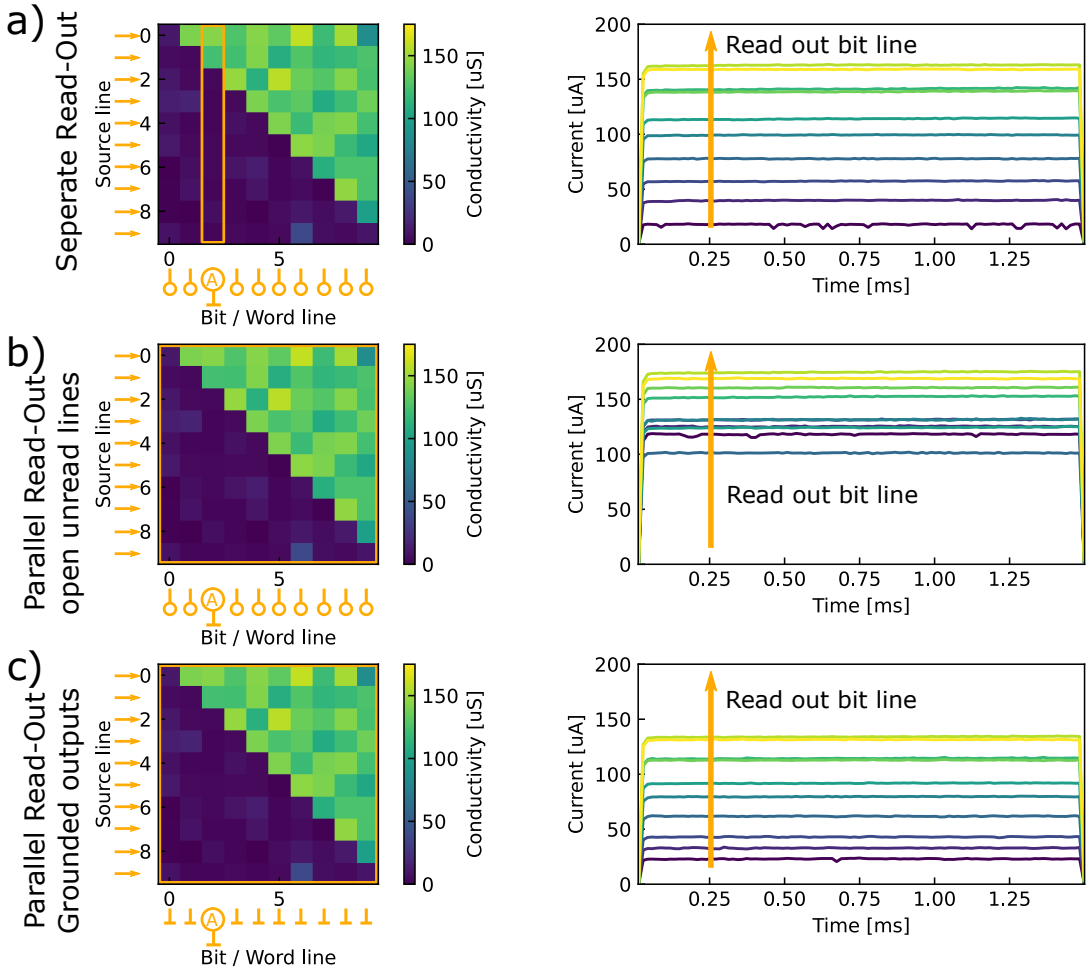

Figure 4.20.: **Sneak path effect in the matrix read out**a) Reference measurement with sequentially opened word lines and separate read out bit lines. Output currents increase almost linearly with the number of high-conductive devices per bit line. b) Unread bit lines are terminated with high impedance. High offset due to sneak paths and lower dynamic. c) All outputs grounded. Linear increase of output currents with number of high-conductive devices. Slightly reduced currents leading to low reduction of output dynamic.

The two read-out schemes that are investigated differ in the termination of the bit lines. While in the first case, all bit lines are open except for the read out bit line, the second case reflects a parallel read-out of all bit lines for which the bit lines are connected to (virtual) GND. In both cases, all transistor gates are open to the maximum irrespective of the read out bit line.

The case with open bit lines is depicted in Figure 4.20b). In the read-out current graph on the right, it can be seen that the field of read-out currents is shifted toward higher currents. This is a direct result of the series resistances due to sneak paths

described above. The lowest current, corresponding to the bit line with no high-conductive devices, is now around $100\mu A$ while it was only about $20\mu A$ in the reference measurement. The highest current is slightly higher than in the reference with $175\mu A$, which leads to a dynamic of about $8\mu A$ per additional high-conductive device on a bit line. This is only half of the reference dynamic. In the read-out scheme with all grounded output lines, the impact of sneak paths is significantly reduced. As depicted in Figure 4.20c), the increase of current with increasing number of high-conductive devices is still about linear. The lowest current corresponding to no high-conductive devices on the bit line is at the same value of $20\mu A$ as in the reference. The highest current however is only at $140\mu A$ which is $20\mu A$ lower than in 4.20a). The dynamic therefore is about $13\mu A$. Therefore, the decrease of the read-out dynamic is significantly lower with grounded output lines than with open unread bit lines.

4.3.2. Parallel write

In the previous section the read-out of multiple memristive devices in a VMM fashion was investigated. In this section now the parallel programming of multiple devices is investigated. This operation is the second fundamental CIM operation. Applications can be imagined in the parallel initialization of multiple devices or for gradual switching as it is employed later in Section 6.2. Therefore, here at first an artifact of the parallel initialization of multiple devices is investigated before the parallel potentiation and depression with voltage pulse trains is discussed.

Array-size dependent parallel initialization

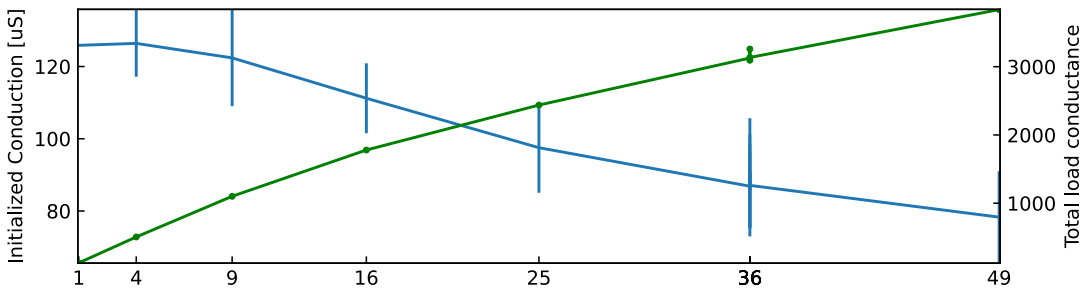

Figure 4.21.: **Array size dependency for parallel initialization** The same quasi-static voltage sweeps are applied to sub arrays for a gate voltage defined initialization of different size and the resulting initialized conductances measured (blue line is the average with the error bars the standard deviation). The reason for the decreasing initialized conductance is an increasing conductive load to the line access transistors (green line) with increasing parallel device count.

When applying quasi-static current voltage sweeps to initialize devices to a specific

conductance state, it is possible to activate multiple source and word lines at the same time to initialize a sub array of devices in parallel. However, when doing so, a dependency of the initialized conductance on the number of parallel initialized devices can be observed. in Figure 4.21 the average conductance after the initialization of different numbers of devices in parallel is shown. The initialization procedure consists of quasi-static triangular voltage shapes. A SET sweep is followed by a RESET sweep and both sweeps repeated three times. Finally, another SET sweep is applied with a gate voltage of $V_{\text{gate}} = 0.9V$. As described in Section 4.2.1 this method can be used to precisely initialize the conductance of memristive devices. However, when the method is applied to multiple devices at the same time, a decline of the resulting average conductance can be observed. Since all devices in the sub-array receive the same voltage stimulus and the measurement is quasi-static, sneak paths can be excluded as the main origin of this behavior. Instead, the limited driver capability of the source line access transistors can be a reason. If all open bit lines are grounded, the sub-array of stimulated devices forms a network of parallel conductances, which the access transistors have to drive. With an ON resistance of G_{on}, the total load conductance on a source line is $G_{\text{line}} = n_{\text{WL}} \cdot G_{\text{on}}$ with the number of open word lines n_{WL} increases to almost $4mS$ for an array size of 49-by-49. This means that each source line access transistor has to drive $I = 1V * 4mS = 4mA$ of current which is about the saturation current of the transistor at the gate voltage 0f $0.9V$ during the SET sweep.

Parallel potentiation

Beside quasi-static current voltage sweeps, also potentiation and depression measurements can be performed array wide. To investigate the potentiation direction, a 6-by-6 sub array with 36 devices in the vertical 1T1R array of the MAD200 demonstrator chip is initialized in a low conductance state with quasi-static current voltage sweeps as described above. Then the source lines to the 6-by-6 sub array are connected to "VIN1" and the transistor gates to the sub array opened with $V_{\text{gate}} = 0.9V$. "VIN1" is then put to GND potential. During the experiment, an $l_{\text{pulse}} = 15\mu s$ long pulse with different voltages is applied for 20 times to all bit lines of the sub-array in parallel. The applied voltage is positive, but because it is applied at the bit line side and the source lines are grounded, the effective voltage over the 1T1R devices is negative. In the following discussion, V_{pot} means this effective voltage over the devices. The applied voltage has the same magnitude but the opposite (positive) polarity. In between each pulse, all conductances are read out separately with a READ voltage $V_{\text{read}} = 0.2V$.

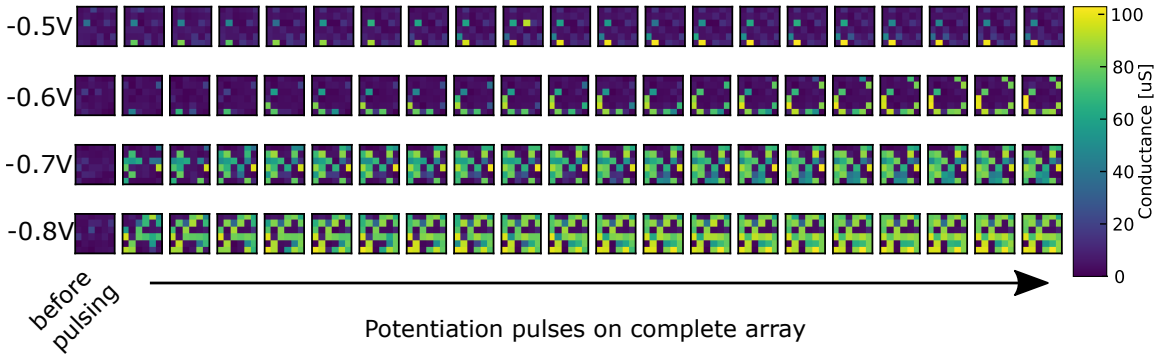

Figure 4.22.: **Parallel array-wide potentiation** A 6-by-6 sub-array is initialized to a low conductance before potentiating pulse trains are applied in parallel to all source lines of the array with different potentiation voltages (as indicated). It can be seen that devices switch abruptly at random pulse numbers to high conductances. With increasing potentiation voltage, the probability of a device to switch increases.

The experiment is conducted for $V_{\text{pot}} = \{-0.5V, -0.6V, -0.7V, -0.8V\}$. The evolution of the conductance matrix is displayed in Figure 4.22. Figure 4.23 on the left side shows for each pulse which fraction of the devices reaches a certain ON-OFF ratio compared to the initial conductance. The plots in the right side show the accumulated amounts of devices that have reached a certain ON-OFF ratio after a certain pulse.

In Figure 4.22 the left most plot in each row is the initialized conductance matrix before the first potentiation pulse is applied. For the lowest voltage $V_{\text{pot}} = -0.5V$ it can be seen that only two devices increase their conductance significantly. Most devices only undergo a small conductance increase. In Figure 4.23a) on the left it can be seen that the probability for devices to reach an ON-OFF ratio of 2 is distributed over all 20 pulses. only rarely, higher ON-OFF ratios are achieved. On the right side it can be seen that the number of devices reaching an ON-OFF ratio of 2 increases steadily during the measurement. Ratios of 20 are not reached at all.

For higher $V_{\text{pot}} = \{-0.6V, -0.7V\}$ it can be seen in Figure 4.22 that more devices strongly increase their conductance. Not all devices increase their conductivity equally, but some show an abrupt increase. For the highest potentiation voltage $V_{\text{pot}} = -0.8V$ most device switch to a high conductive state within the 20 pulses of the experiment. When comparing the conductances to which the devices switch, there is no difference between the four potentiation voltages. All measurements show in the end devices which have switched to a range of conductances between $60\mu S$ and $100\mu S$. However, when looking at the switching statistics of the measurement with the highest $V_{\text{pot}} = -0.8V$, most switching activity can be observed right on the first pulse. 50% of the devices reach already here an ON-OFF ratio of 2 and at least 10% even a ratio of 20. During the following pulses further devices reach all different ON-OFF ratios. This behavior is also reflected int he accumulated switching statistics in

the right side plot. The sharpest increase for all ON-OFF ratios can be seen on the first pulse. After 18 pulses, all devices have reached an ON-OFF ratio of 2 and after the measurement 20% of the conductances are increased by a factor of 20.

The array wide potentiation shows an abrupt and stochastic switching from the initial low conductance to a very high conductance with an ON-OFF ratio well above 10. Nevertheless, also devices which do not show an abrupt switching increase their conductance.

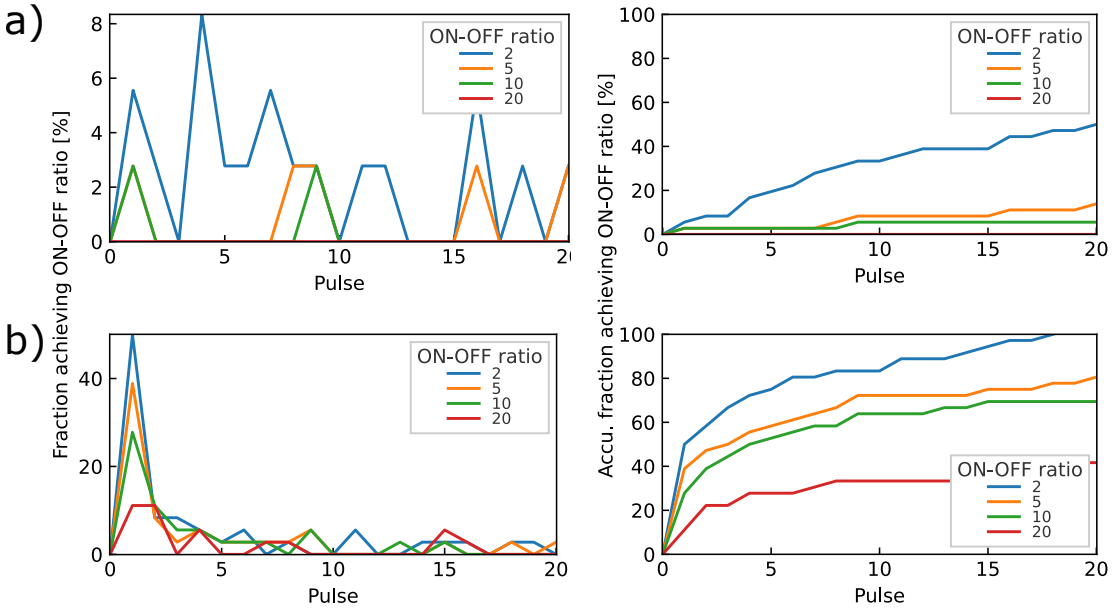

Figure 4.23.: **Parallel potentiation switching statistics** a) Low voltage ($-0.5V$) potentiation probability. Devices achieve an ON-Off ratio of 2 distributed across all 20 pulses. Higher ratios are rarely reached. b) Accumulated low voltage ($-0.5V$) switching probability. The number of devices reaching different low ON-OFF ratios increases almost linearly. c) High voltage ($-0.8V$) potentiation probability. The highest switching probability for all ON-OFF ratios is on the first pulse. 50% of the devices already here change their conductance by a factor of 2. Afterward the probability is evenly equally distributed. d) Accumulated high voltage ($-0.8V$) switching probability. Most switching takes place on the first pulse with the fraction of devices reaching specific ON-OFF ratios slowly increasing afterwards.

Parallel Depression

In the depression direction, the same sub-matrix of 6-by-6 devices is initialized first, but in this case in a high conductive state. Therefore, quasi-static current voltage

sweeps with $V_{\text{pos}} = 2.5$ and $V_{\text{gate}} = 0.9V$ in the SET direction followed by $V_{\text{neg}} = -1.5V$ and $V_{\text{gate}} = 5.0V$ in the RESET direction are applied three times, while in the last sweep the RESET direction is omitted to achieve a high conductance. Again, 20 depression pulses are applied to the complete array in parallel via the respective source lines while the bit lines are grounded. The read-out after each depression pulse is the same as in the potentiation measurements above.

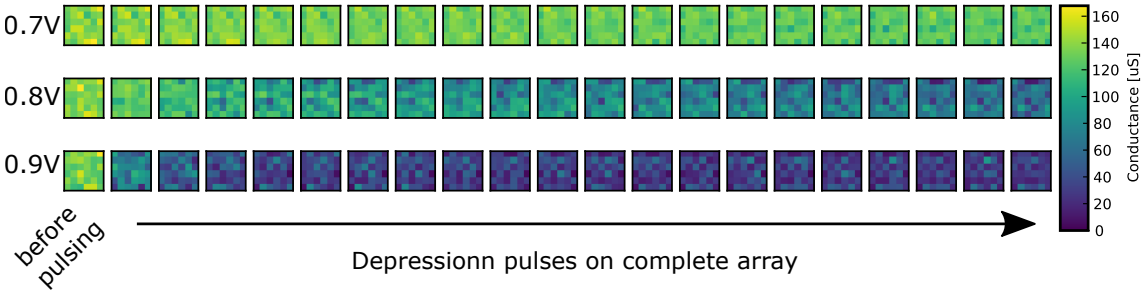

Figure 4.24.: **Parallel array-wide depression** A 6-by-6 sub-array is initialized to high conductance before depression pulse trains are applied to all source lines in parallel with the indicated voltages. For all voltages the conductance of the devices decreases gradually and homogeneously across the whole sub-array. Higher voltages lead to a steeper decrease and thus a lower final conductances at the end of the measurement.

In Figure 4.24 the conductance matrices evolution for three different depression voltages $V_{\text{dep}} = \{0.7V, 0.8V, 0.9V\}$ are displayed. The first matrix in each row is the initialized conductance matrix. Qualitatively, it can be seen that for the lowest voltage $V_{\text{pot}} = 0.7V$ the conductances barely change over the 20 pulses of the measurement. A small homogeneous overall decrease however can be seen. This is also reflected in the switching statistics of this particular depression experiment which are displayed in Figure 4.25a). On the left side one can see that from the second pulse on after each depression pulse about $5 - 10\%$ of the devices change their conductance by a factor of 2, in this case a halving of the conductance. Higher ON-OFF ratios are rarely reached after later pulses. In the accumulated switching statistics on the right, a linear increase of the fraction of devices is visible that reach an ON-OFF ratio of 2. After 15 pulses all devices have halved their conductance. Only 5% of the device conductances have changed by a factor 5 after 20 pulses, and even less by a factor of 10. Higher ON-OFF ratios cannot be observed.

When looking at higher depression voltages in Figure 4.24, it can be seen that also here the change of the conductances is homogeneous across the complete sub array. Furthermore, especially for the measurement with a depression voltage of $V_{\text{dep}} = 0.8V$ a gradual change of the conductance is visible. An even higher depression voltage of $V_{\text{dep}} = 0.9V$ leads to a strong decrease of the conductance right upon the first depression pulse. Yet, this decrease is still homogeneous over the whole array. The switching statistics at this higher depression voltage, depicted in 4.25b), show that most devices reach the low ON-OFF threshold of 2 already within the first 3 pulses.

In the first and second pulse each around 40% of the devices half their conductance. The switching rate to higher ON-OFF ratios is then more equally distributed over the rest of the measurement. In the following pulses always about 10% of the devices reach and ON-OFF ratio of 5 and 5% a ratio of 10. For the accumulated switching statistics in the right-hand plot, this leads to an almost linear increase of all device counts.

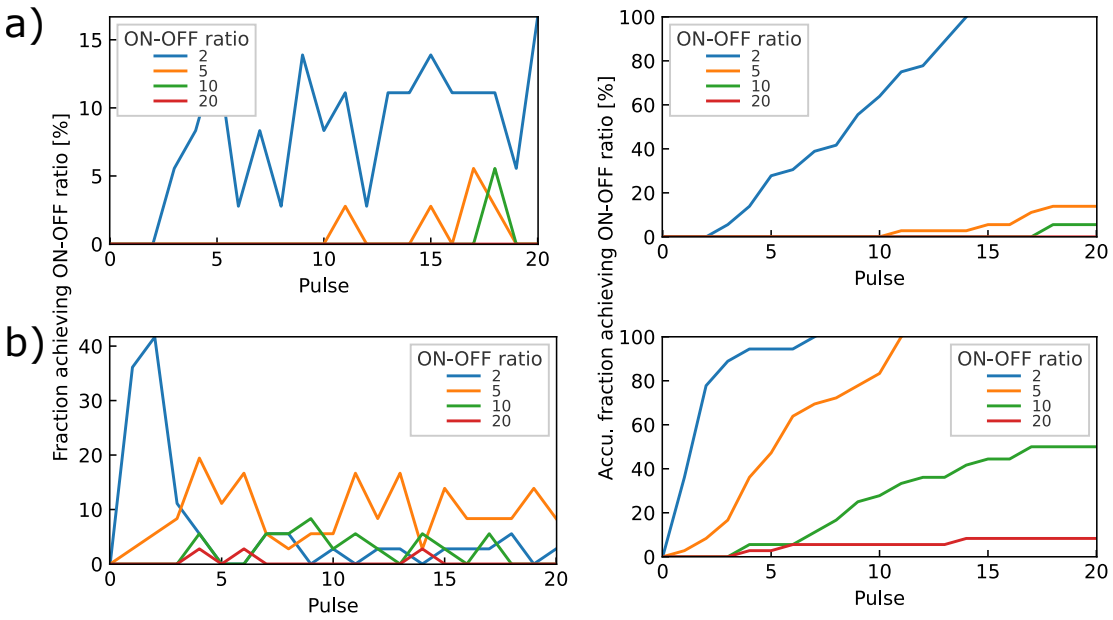

Figure 4.25.: **Parallel depression switching statistics** a) Low voltage ($0.7V$) depression probability. The switching probability for certain ON-OFF ratios is about constant after a delay where no switching is observed (2 pulses for ratio 2, 10 pulses for ratio 5). b) Accumulated low voltage ($0.7V$) switching probability. The number of devices reaching different low ON-OFF ratios increases linearly after the respective delay. c) High voltage ($0.9V$) potentiation probability. During the first few pulses, devices reaching an ON-OFF ratio of 2 dominate. Later the fraction of devices reaching higher ratios is higher and about constant. d) Accumulated high voltage ($0.9V$) depression switching probability. The fractions of devices reaching all ON-OFF ratios increase almost linearly with higher steepness for lower ratios.

Comparison of parallel potentiation and depression

When comparing the switching behavior in the potentiation and depression direction, a fundamental difference can be seen. The potentiation is in the main part abrupt and stochastic. This means that devices change their conductance suddenly, and

not necessarily on the first pulse, immediately from a low to a high conductance which then does not significantly increase further. For devices that do not switch abruptly, only a small gradual increase of the conductance takes place. The depression in contrast is gradual and homogeneous. The change of the device conductance is stepwise until a saturation conductance is reached. The step size of the conductance after each depression pulse is similar for all devices in the array.

4.4. Summary

In this chapter the design and characterization of a memristive-CMOS co-integrated CIM accelerator for neural network applications is discussed. The focus of the design is to create a versatile yet functional chip. Therefore, an analog / mixed-signal routing and address logic is devised which allows the control of large 512-by-32 arrays with only a limited number of I/O pads and no on-chip co-processor necessary. The measurement of output signals is performed completely off-chip. The goal of all these decisions is to push all complexity of voltage pulse generation tasks and the need for precision of analog current and voltage measurements to external of-the-shelf hardware which can be fit to the specific needs of an experiment.

The memristive switching of single devices and complete sub-arrays is investigated. The focus of these measurements is placed on the precise initialization of the conductance of the memristive devices and pulse train measurements. Both these operations are crucial for neural network tasks. The initialization can be used to precisely write pre-trained networks to perform inference workloads, while pulse train measurements resemble the step wise adjustment of weights in e.g., gradient descent learning.

5. Memristive devices as weights for Machine Learning

Machine Learning (ML) in Artificial Neural Networks (ANN) has recently shown great success in creating artificial systems that mimic human intelligence at a level never seen before, with the latest example being the conversational AI system ChatGPT [1]. The vast energy consumption however of the server farms necessary for the training and inference of these models leads to high monetary and environmental costs. But even less complex neural network applications are mostly executed within centralized servers, which again raises issues in terms of energy consumption for data transfer and data privacy. Therefore, ANN accelerators which can be used locally, even in mobile systems, are an ongoing object of investigation.

Memristive devices are a possible solution to realize large weight layers for ANNs due to their small spatial form factor and low power operation modes. There are however some drawbacks when using memristive devices as weights for neural networks. In this thesis two crucial aspects are investigated among these. These are the limited conductance range and the non-linear and asymmetric update behavior. For both approaches for a solution are proposed in the following chapter. For both issues solutions are proposed in the following.

5.1. Initialization of memristive weight layers

Weights in ANNs usually can assume any floating-point value. Memristive devices in contrast are restricted to a certain range of conductance values usually in the nano-siemens nS or micro-siemens μS regime. Therefore, it is necessary to reflect this constraints in the architecture of a memristive ANN. Furthermore, the currents produced during VM operations in memristive crossbar arrays are limited due to the limited conductance range of the memristive devices, which can only be positive, and also the constraints on the input voltages. These must be small enough not to influence the conductance state of the devices. It is however not practicable to scale each memristive conductance to match another range of values, such as the interval $(-1, 1)$, because this would mean reading out devices separately and performing the multiplication of input vector and weight layer with a separate computing system instead of a direct VMM with a memristive crossbar array. In this case the memristive devices would only be a pure weight data storage and take no part in the computation.

[1] `https://help.openai.com/en/collections/3742473-chatgpt`

To exploit the benefits of VMM computation in memristive crossbar arrays, the range of conductances provided by memristive devices and the limited range of output (current) values from this memristive VMM must be dealt with, and other network parameters must be found to compensate these limits.

One approach is to find a proper scaling of the activation function after each weight layer and combine each memristive devices with a bias device. The currents of both devices are subtracted which makes it possible to realize effectively negative weights. This approach is explained in the following section, and it is shown how the memristive conductance range can be considered to create network initializations that benefit the training performance, i.e., to realize the Xavier and Kaiming initializations.

5.1.1. Naive memristive initialization

When initializing a memristive weight layer, all devices are formed and potentially switched multiple times. Depending on the type of devices and the switching parameters, all devices will assume conductance values within a certain range. For simplicity, we here assume a normal distribution of the conductance values g_+ around the mean conductance g_{mean} with a certain standard deviation σ. The difference between g_+ and a constant bias conductance g_{bias} gives the weight w.

$$\begin{gathered} w = g_+ - g_{bias} \\ g_+ \in N(g_{mean}, \sigma) \text{ with } \sigma = \alpha \cdot \frac{g_{max} - g_{min}}{2} \end{gathered} \tag{5.1}$$

g_{min} and g_{max} are the maximum and minimum conductances and $\alpha < 1$ a scalar parameter to ensure that most conductances are withing the interval (g_{min}, g_{max}). It has to be noted that $g_{max} > 0$ and $g_{min} > 0$ which makes also all $g_+ > 0$. When assuming that weights in an ANN weight layer distribute symmetrically around zero, the bias conductance can be chosen as $g_{bias} = g_{mean}$ to pull the mean of the distribution of w to zero.

Secondly, to cope with the memristive conductance range, a scaling factor is introduced as part of the activation function. Furthermore, as in other initialization schemes, the input and output size of the weight layer is considered, yielding a scaling factor of

$$scale = \frac{2}{g_{max} - g_{min}} \frac{1}{N^{L-1} + N^L} \tag{5.2}$$

This approach can also be used for memristive layers with a uniform initial conductance distribution.

The implementation of the constant bias conductance can be realized as proposed in [142] and the scaling factor as an amplifier before or after the activation function.

However, this initialization method is a naive reference. It suffers from poor signal conservation and therefore, from the Vanishing or Exploding Signal respectively Gradient problem described in Section 2.1.4. It is only used here as a reference for more advanced initialization methods.

5.1.2. Kaiming initialization

Kaiming He and others showed that the variance of the gradient in a neural network scales with the variance of the weights in a layer L and proportional to the layer size N^L [27]. Demanding that this scaling must be 1 for each layer

$$\frac{N^L}{2} Var(w^L) = 1 \tag{5.3}$$

for *ReLU* activation functions they propose to initialize a weight layer by drawing the weights from a normal distribution around zero with the width depending on the layer size like

$$w^L \sim \mathcal{N}\left(0, \frac{2}{N^L}\right) \tag{5.4}$$

This shows great improvement of training especially for deep neural networks with many subsequent layers.
Memristive devices only have a specific and limited resistance range $[g_{min}, g_{max}]$ that cannot be adjusted freely. When assuming that a memristive weight layer can be initialized normally distributed within this range, this distribution is

$$g \sim \mathcal{N}\left(\frac{g_{max} + g_{min}}{2}, \alpha \cdot \frac{g_{max} - g_{min}}{2}\right) \tag{5.5}$$

where α is a scaling factor with $\alpha < 1$ that depends on the initialization scheme, making sure that a large percentage of the g lies within $[g_{min}, g_{max}]$.

However, due to the shaping of the normal distribution in equation 5.4, the weights can assume arbitrary values. Moreover, g is not centered around zero as memristive conductances cannot be negative. Therefore, each memristive weight is combined with a reference conductance g_{ref} to the total conductance g_{total} with

$$g_{total} = g - g_{bias} \tag{5.6}$$

where g_{bias} is equal for all conductances. Doing so, a negative g_{total} can be achieved for $g < g_{bias}$ Other approaches like from Prezioso et al. [143] propose having two anti-parallel conductances g_+ and g_-, yielding a total conductance of $g_{total} = g_+ - g_-$, where both conductances can be changed during training.
One can assume a linear scaling relationship between the memristive weights g and the weights of a theoretical network layer w. With this scaling factor being $A_{Kaiming}$, the relationship between the variances is

$$Var(A_{Kaiming} * g) = Var(w) \tag{5.7}$$

putting this relation in to equation 5.3 yields

$$\begin{aligned} \frac{1}{2} N^L Var(A_{Kaiming} \cdot g) &= 1 \\ \frac{A_{Kaiming}^2 N^L}{2} Var(g) &= 1 \\ A_{Kaiming} = \sqrt{\frac{2}{N^L \cdot Var(g)}} &= \sqrt{\frac{2}{N^L}} \cdot \frac{1}{std(g)} \end{aligned} \tag{5.8}$$

where $std(g)$ is the standard deviation of g which in equation 5.1 was defined as $\frac{g_{max}-g_{min}}{2}$.

The factor $A_{Kaiming}$ that scales the conductance values can potentially be applied at different states of the network. Obviously, one could scale each individual output current of each device individually, which would generate an immense circuit overhead. Bit since the VMM in a memristive crossbar array is a linear operation and $A_{Kaiming}$ a linear scaling factor, it is also possible to scale the sum of each output by $A_{Kaiming}$. this would be reflected in the input scaling of the activation function which follows the weight layer that performs the VMM.

5.1.3. Xavier initialization

The idea of the Kaiming initialization was derived from the method proposed by Xavier Glorot and Yoshua Bengio [26]. Instead of assuming ReLU-type activation functions, they used linear functions. ReLU, as assumed by He et al. is more realistic in terms of actual neural network architectures as it is a non-linearity.
However, the main difference in the derivation of the Xavier initialization can be found in the scaling condition of the weight variance in formula 5.3, where there is no $\frac{1}{2}$ factor. Following the same train of thoughts outline above, the scaling factor A_{Xavier} for the activation functions can be determined.

$$\begin{aligned} Var(W) &= \frac{6}{N^{L-1}+N^{L}} = Var(A_{Xavier}\cdot g) = A_{Xavier}^{2}\cdot Var(g) \\ A_{Xavier} &= \sqrt{\frac{6}{(N^{L-1}+N^{L})\cdot Var(g)}} = \sqrt{\frac{6}{(N^{L-1}+N^{L})}}\cdot\frac{1}{std(g)} \end{aligned} \tag{5.9}$$

5.1.4. Comparison of initialization methods for memristive ANN layers

To evaluate what impact the initialization method has on the networks training performance, a simple two-layer network with $(784, 100)$ weights in the first and $(100, 10)$ weights in the second layer and ReLU activation functions is trained with SGD on the MNIST data set for pattern recognition. Scaling factors and bias conductances are chosen as described above for a naive initialization with normally and uniformly distributed memristive layers, the Xavier, and the Kaiming initialization. The network is trained with the same learning rate for 10 epochs.

In Figure 5.1 the distribution of the recognition error after 10 epochs for 100 instantiations of the respective initialization method is displayed. The naive method leads to high errors around 40% and 30%. With the Xavier and Kaiming initialization the median error decreases below 20% with the Kaiming initialization even reaching to only 10% error.

It can be seen that the initialization scheme has large impact on the training performance. Therefore, for all further ANN training simulations displayed in this thesis, the Kaiming-style initialization approach described above is used.

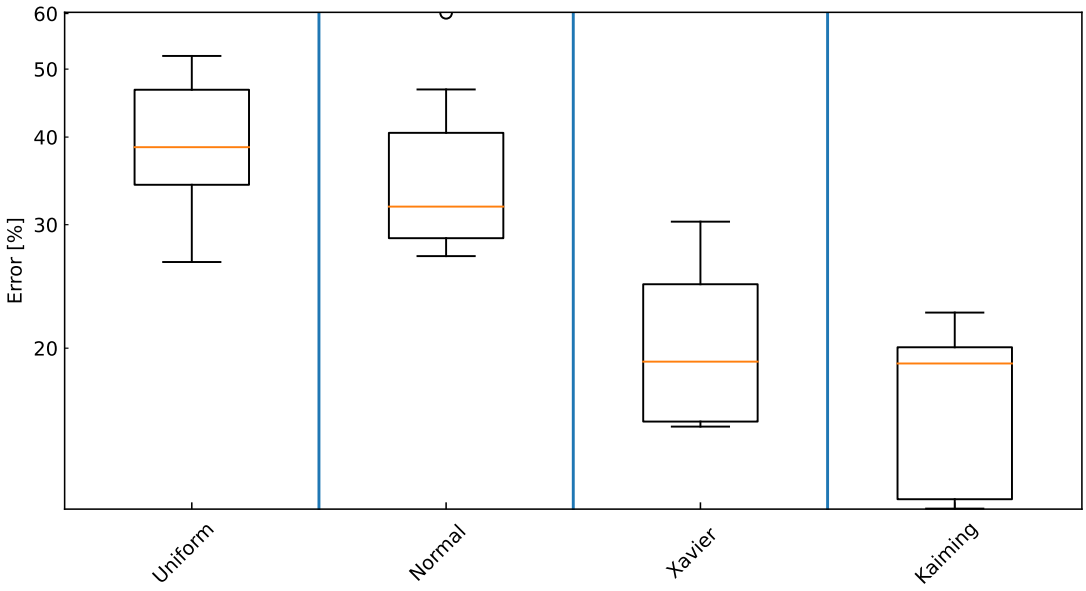

Figure 5.1.: **Comparison of different initializations for memristive ANN layers.** Recognition error after 10 epochs of training on the MNIST data set is displayed. Uniformly and normally distributed initializations without tailored activation functions yield high errors. With the activation function's scaling factor derived from Xavier initialization, a significant gain in accuracy is observed. Derivation from Kaiming initialization yields a similar average error, but the distribution of errors is shifted to lower values.

5.2. Memristive update dynamics

When training ANNs with gradient-based algorithms, the gradients are usually only multiplied with a learning rate which is the same for all layers and weights.

However, when applying gradients to memristive weight layers by applying voltage pulses to the devices, the actual conductance change is not necessarily linear to the gradient and the same for all devices and conductance values. The reason is the specific memristive update dynamics which perturb the update gradient.

The effect of these dynamics can exemplarily be observed in the potentiation and depression measurements (see Section 3.1.2a)). Figure 5.2a) shows five cycles of a potentiation depression measurement on a $Ti/SrTiO_3$ device. The opaque dots symbolize the conductance after the respective pulse with the colors differentiating the measurement cycles. The dark blue and orange lines represent the averages of the potentiation and depression cycles. It can be seen that in the beginning of both measurement directions the conductance changes faster than at the end. When plotting the magnitude of the conductance change $delG$ over the absolute conductance (see

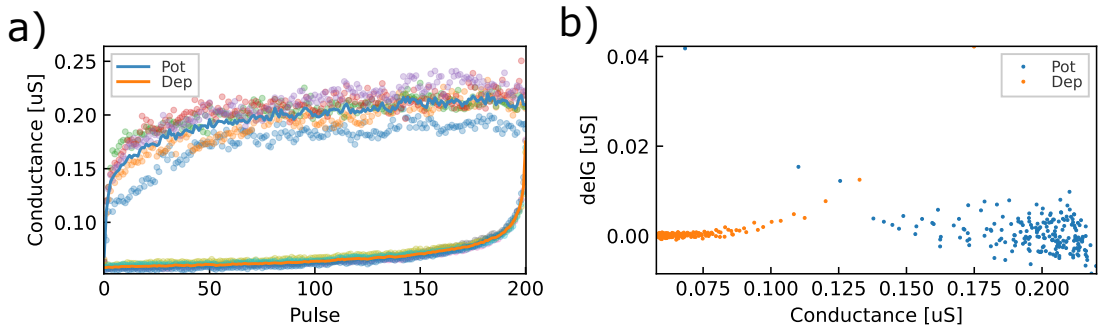

Figure 5.2.: **Exemplary combined potentiation and depression measurement for Ti/STO devices.**
a) Evolution of device conductance after a number of consecutive equal pulses. Starting from an initially low conductance, a pulse train with a positive polarity causes gradual potentiation followed by a gradual depression by a pulse train with negative polarity.
b) Dependency of the conductance step on the conductance for potentiation and depression. For potentiation, the highest conductance increase due to a single pulse is at low device conductance. At high conductances, the increase reduces to almost zero, corresponding to the conductance saturation in a). For the depression the situation in vice-versa.

Figure 5.2b)) for the potentiation it can be observed that for low conductances around $75nS$ the step size is larger and converges toward zero when approaching $200nS$. The opposite trend is seen for the depression direction where high conductance changes are found at higher conductances and vice versa. Especially for the depression measurement it can be seen that the correlation between step size and conductance is not linear.

For the application of gradients on memristive weight layers this means that it depends on the current conductance state of the memristive device how strongly a computed gradient is applied. This is called non-linearity in the weight update. Furthermore, it depends on the direction of the conductance change, if a high or a low conductance leads to a higher conductance change and vice versa. This is the asymmetry of the weight update. The latter aspect is especially problematic for the training of memristive ANNs [142].

5.3. Training ANNs with memristive update dynamics

5.3.1. Update schemes

There are two possible ways how to convert a calculated gradient to an update pulse, the first being a simple Gradient Sign Rule and the second being a Gradient Magnitude Modulation. Both resemble the actual update with different accuracy and therefore

lead to different accuracies. Also, the complexity of circuitry to possibly calculate and apply the update pulses differs.

Gradient Sign Rule

Parts of this section have been published in [102]. The displayed measurement data on PCMO memristive devices was acquired by Alexander Gutsche (PGI-7, Forschungszentrum Jülich GmbH).

Using the Gradient Sign Rule, the voltage polarity V_{update} of an update pulse depends on the sign of the calculated gradient δw. The magnitude in either polarity (V_{pot} or V_{dep}) and length of the update pulse l_{update} are fixed (but not necessarily equal):

$$V_{update} = \begin{cases} V_{pot} & , \delta w < 0 \\ V_{dep} & , \delta w > 0 \end{cases} \text{ and } l_{update} = const. \tag{5.10}$$

This leads to the issue that the magnitude of the gradient plays no role anymore in the definition of the respective update pulse. A small gradient, as it can be expected when the training is completed, would cause the same update pulse as a high gradient, as observed for weights far from their converged state at the beginning of the training. Potentially, weights could oscillate which would impede the training convergence.

To avoid this, a selection of gradients must be made that cause an update. A naive approach would be the introduction of a threshold magnitude below which gradients are not applied. This however imposes the problem that this threshold would be strongly dependent on the task and network and moreover, it could be necessary to adapt it during training because gradients tend to become smaller with progressing training.

Therefore, here a strict yet simple criterion is chosen as to which only the one highest positive and highest negative (in absolute terms) gradients are applied. This leads to a sparse update matrix with only two non-zero elements per layer.

This update rule was investigated in [102] for PCMO based memristive devices with different interlayer materials.

Figures 5.3 a), b), and c) show the potentiation and depression curves of three PCMO based memristive devices with an Al (a)), TaO_x (b)), and WO_x (c)) interlayer between the PCMO and the platinum top electrode. Fabrication details and static switching curves are available in [102]. The ON/OFF ratio of samples with an Al interlayer and with a $TaOx$ interlayer are similar, while with a WOx interlayer a slightly larger ratio is achieved. The Al interlayer sample on the other hand shows a faster saturation in the potentiation measurement compared to the TaO_x and WO_x interlayer samples, while the WOx sample saturates fastest in the depression measurement. To investigate how these devices would behave as weights in an ANN, the potentiation and depression curves were fitted with a logistic function as described in [102] and the results implemented in the TensorFlow framework [144]. An MLP with a 784 node input layer, two hidden layers (with 250 and 125 nodes respectively), and 10

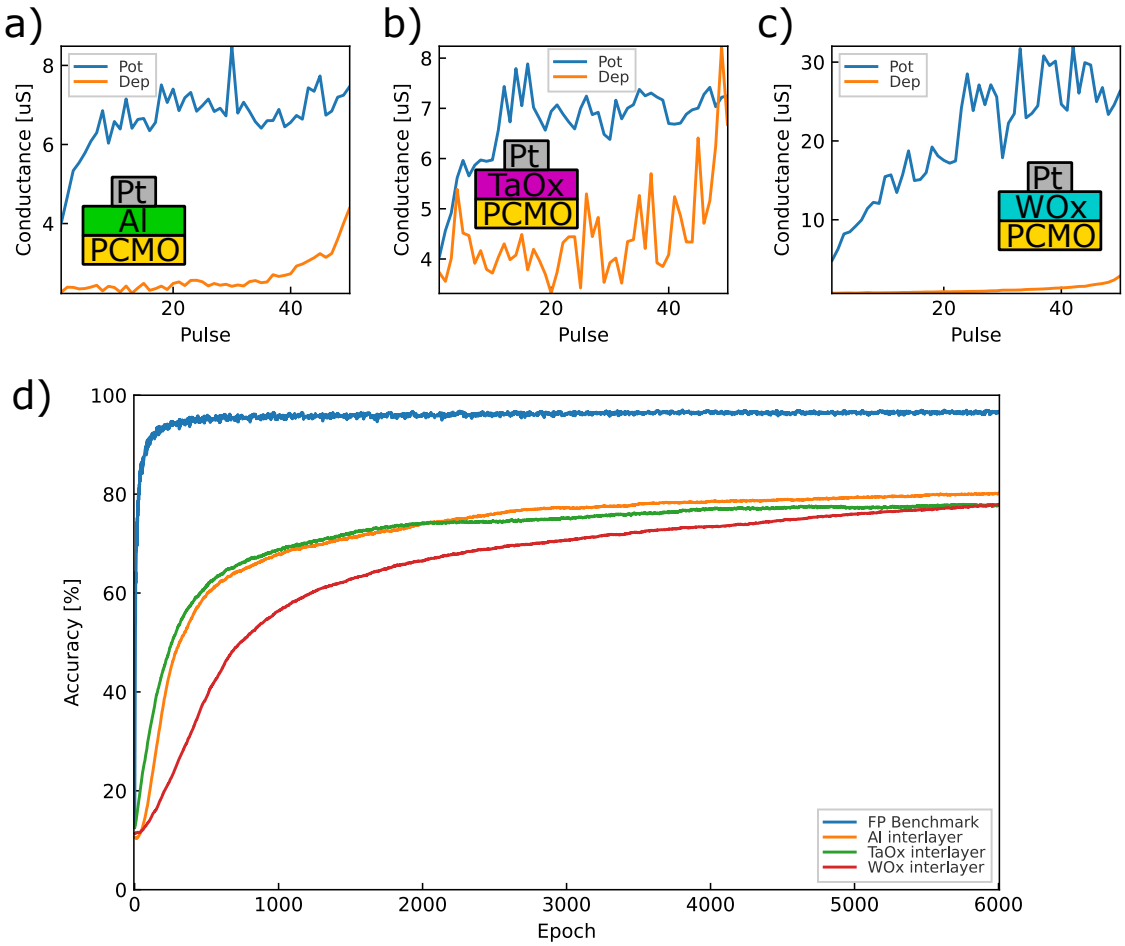

Figure 5.3.: **Sign Rule training with different PCMO memristive devices** a), b), and c) show measured potentiation (blue) and depression (orange) curves for PCMO based ReRAM devices with different interface layers (Al, TaOx, and WOx respectively). d) When training a three layer network with the Gradient Sign Rule and modeling the memristive update dynamics after the respective displayed potentiation / depression curves, for all devices stacks the test recognition accuracy increases strongly within the first 1,000 epochs and then saturates. The maximum accuracy is similar for all devices stacks and significantly lower than the training with the same update rule but linear floating point weights.

output nodes was trained on the MNIST data set using the above described Gradient Sign Rule for 6,000 epochs. The resulting test accuracy for each investigated PCMO device stack can be found in Figure 5.3 d). To check if possible losses in accuracy are due to the network structure or training algorithm, a floating-point benchmark was performed (also displayed in d)). The maximum accuracy of the benchmark is 96.7% on the test set. All training simulations using PCMO device update dynamics show a similar significant loss in accuracy with the Al interlayer samples (yellow) reaching only 80.2%, the TaO_x interlayer devices (green) 77.8%, and the WO_x interlayer devices (red) 77.9% accuracy. Furthermore, the accuracy of the memristive device dynamics increases slower than the floating-point benchmark. The WO_x interlayer samples exhibit the slowest increase of the three memristive device stacks yet reaches about the same test accuracy at epoch 6,000.

The investigation in [102] suggest that the limited ON/OFF ratio of the memristive devices is one cause of the accuracy loss. Another factor is the steepness of the initial increase of the potentiation curve. To this combination of factors, the slightly better accuracy with *Al* interlayer PCMO samples is attributed, because even though they have a smaller ON/OFF ratio than the WOx interlayer samples, but a steeper initial increase of the potentiation curve (compare Figure 5.3 a) and c)).
The trend of an increased training performance with a steeper, more non-linear, potentiation curve seems in contrast to previous observations [106], [145], where a more linear potentiation and depression behavior was beneficial to the learning performance. However, this difference can be attributed to the special Gradient Sign Rule for the update resulting in extremely sparse update matrices. With this, devices get updated only rarely and this makes an initially faster change beneficial.

In Figure 5.3 d) another effect of these sparse update matrices can be observed. The number of epochs becomes very large. But this leads to 48,000 updates over the curse of the whole 6,000 training epochs for all devices and layers in the network. In the same network with every gradient being applied in each update step, 1,169 updates must be performed per calculated gradient. For a training batch which comprises the whole training data set, therefore, after 41 epochs the same number of updates would have been applied. With Stochastic Gradient Descent training and a smaller batch size of 64 samples, the training set of MNIST of 60,000 samples could not be progressed completely before 48,000 update pulses are applied. Another benefit of the sparse update matrices produced by the Gradient Sign Rule therefore is the comparably low number of update pulses which saves training time on the one hand, because inference in an ANN with memristive crossbar layers would be a one-shot action while updates must, in the worst case, be applied sequentially. On the other hand, it saves device lifetime, as memristive devices only have a limited switching endurance.

Gradient-magnitude modulation

Instead of modulating the update pulse only by the sign of the gradient, the height or length can also be considered. This relieves the issue that small gradients can

potentially have the same effect as large.

When using rectangular voltage pulses to apply the update, there are two parameters which can be used to reflect different gradient magnitudes: the height (magnitude) or the length of the applied update pulse. In this work only the second approach is investigated. The reason is that memristive update dynamics show a strong non-linearity with the pulse voltage [146]. While this can be beneficial for the storage of the weight data itself, it changes the applied gradient value in a strongly non-linear fashion and can therefore hinder the training, because it disturbs the gradient which is applied.

The pulse length on the other hand scales the conductances change linearly (not considering general memristive update dynamics). A gradient magnitude is linearly transformed to a pulse length while the pulse voltage is a fixed positive or negative voltage depending on the sign of the gradient.

This approached is investigated in the remainder of this chapter.

5.3.2. Loss of training accuracy due to memristive update dynamics

In order to investigate the impact of memristive update dynamics using the gradient magnitude modulation approach, it is necessary to have a model for the memristive switching dynamics that can capture arbitrary gradient values. This can hardly be provided by experimental data, because only a limited number of experiments for a limited number of gradients to reflect can be conducted.

There are various models for the memristive switching dynamics, ranging from physically plausible yet computationally costly compact models like in [147] to fit models that are fast yet only capture the range of the measured data. For the purpose of ANN simulations where numerous memristive weights are to be investigated, a fast model is necessary. Therefore, here a fit model described by Tayfun Gokmen and Wilfried Haensch is employed [148]. This model describes the perturbation of the update due to the memristive dynamics as follows:

$$\begin{aligned} \Delta w_{i,j}^{act} &= \eta \left(\Delta w_{i,j} F(w_{i,j}) + |\Delta w_{i,j}| G(w_{i,j}) \right) \\ &= \eta \Delta w_{i,j} \left(F(w_{i,j}) + sign\left(\Delta w_{i,j}\right) G(w_{i,j}) \right) \end{aligned} \tag{5.11}$$

The terms $F(w_{i,j})$ and $G(w_{i,j})$, describe the symmetric and asymmetric part of the perturbation, respectively. They follow both a simple linear relationship.

$$\begin{aligned} F(w_{i,j}) &= \alpha_s + \beta_s \cdot w_{i,j} \\ G(w_{i,j}) &= \alpha_a + \beta_a \cdot w_{i,j} \end{aligned} \tag{5.12}$$

The α and β factors (with index s for the symmetric part and a for the asymmetric contribution) can be extracted from potentiation and depression measurements

described in [148].The parameters α and β with their respective subindices can be calculated from fits of the potentiation and depression measurement data like

$$\begin{aligned} \alpha_s &= (b_p + b_m)/2 \\ \beta_s &= (m_p + m_m)/2 \\ \alpha_a &= (b_p - b_m)/2 \\ \beta_a &= (m_p - m_m)/2 \end{aligned} \tag{5.13}$$

where b_p and m_p are the fit parameters of a linear fit with the formula

$$delG = m_i \cdot p + b_i$$

of the conductance step size $delG$ to the pulse number p during the potentiation and b_m and m_m for the depression measurement. The subindices p stand for a positive conductance change, the potentiation, and m for the negative conductance change, the depression. The fit parameters for the respective material systems and pulse voltages employed in this thesis can be found in Table 5.4.

As usual in Gradient Descent learning, the gradient $\Delta g_{i,j}$ is multiplied with the learning rate η before it is subtracted from the device conductance $g_{i,j}$. According to [142], the asymmetry of the memristive update dynamics creates the largest impact on the training accuracy in ANN.

As described in Section 3.2, this model is implemented in the update path of the ANN training simulator. As an example, the training of an MLP with three neuron layers on the MNIST data set for number recognition is demonstrated here, while the weight update is subjected to the dynamics of resistive switching $SrTiO_3$ devices [92]. The switching behavior in potentiation and depression measurements is shown above. When training the network with these weight update dynamics, the accuracy on the test set develops as shown in Figure 5.5 for ten iterations with the dark line being the average accuracy at the respective epoch and the shaded area spanning between the maximum and minimum value. Toward the end, the accuracy saturates at around 85% max. This is a loss of more than 10% compared to the accuracy that can be reached with exact updates of floating point weight.

This loss of accuracy is commonly observed for various material systems like HfO [133], [149], Ta_2O_5 [150] and PCMO based devices.

5.4. Compensation of memristive update disturbance

For an effective application of memristive devices in ANN systems, the accuracy loss due to memristive update dynamics must be compensated. There are different possible stages at which such a compensation mechanism can be applied. The most obvious would be either the neuron model or the update of the memristive device. For both stages, approaches are presented and discussed here.

Device stack	V_{pulse}	m_p	b_p	m_m	m_p
$Al/PCMO$	$-1.5V$	-0.132	$8.17e-7$		
	$-1.6V$	-0.135	$9.46e-7$		
	$-2.0V$	-0.131	$9.49e-7$		
	$1.6V$			0.0188	$-7.51e-8$
	$1.8V$			0.0712	$-2.17e-7$
	$2.0V$			0.118	$-2.81e-7$
	$2.2V$			0.137	$-3.23e-7$
	$2.3V$			0.105	$-2.08e-7$
$WO_x/PCMO$	$-2.0V$	-0.0603	$2.46e-7$		
	$-2.2V$	-0.0358	$3.76e-7$		
	$-2.4V$	-0.0417	$1.23e-6$		
	$-2.6V$	-0.0332	$3.32e-7$		
	$1.0V$			0.0333	$-3.34e-8$
	$1.2V$			0.0779	$-8.33e-8$
	$1.4V$			0.0945	$-8.54e-8$
	$1.6V$			0.0868	$-6.354e-8$
$TaO_x/PCMO$	$-1.0V$	-0.122	$8.64e-7$		
	$-1.5V$	-0.166	$1.11e-6$		
	$0.8V$			0.0763	$-3.480e-7$
	$1.2V$			0.119	$-4.66e-7$
	$1.4V$			0.120	$-4.27e-7$
	$1.6V$			0.166	$-6.90e-7$
	$2.0V$			0.113	$-3.16e-7$
$Ti/SrTiO_3$	$2.3V$	-0.186	$6.79e-8$		
	$-2.4V$			0.148	$-1.24e-8$

Figure 5.4.: **Fit parameters for PCMO and STO devices for network simulations**

5.4.1. Compensating neuron model

The concept of the compensating neuron model is to make the neuron "aware" of the disturbance caused by the memristive update dynamics. Then, the neuron can adapt its output signal which is propagated in the forward pass to the next layer to account for memristive update disturbance in the backward pass and the following update of its incoming weights by gradient descent. In the remainder of this thesis this approach is however not followed further, because of multiple issues that occur in the real application of the model. Nevertheless, the mathematical approach will be described here, and the problematic issues discussed.

As a reminder, each neuron j performs two operations: the accumulation of the activation a_j as the weighted sum of inputs from the previous layer $w_{i,j} \cdot x_i$ and the

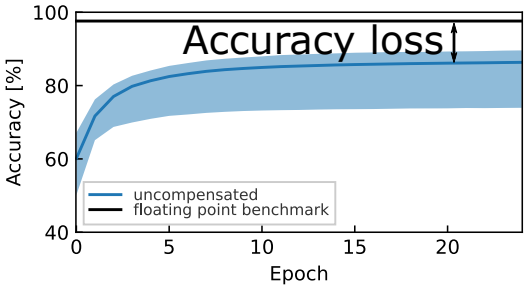

Figure 5.5.: **Exemplary training with Ti / $SrTiO_3$ device dynamics** When assuming the memristive switching dynamics of Ti / $SrTiO_3$ ReRAM devices /displayed in Figure 5.2 for weights in an ANN training on the MNIST data set, a significant loss of recognition accuracy can be observed.

generation of an output y_j from this activation with the activation function $\sigma(a_j)$ as written in Equation 5.14.

$$\begin{aligned} a_j &= \sum_i^{N^L} (w_{i,j} \cdot x_i) \\ y_j &= \sigma(a_j) \end{aligned} \tag{5.14}$$

In this approach we employ the model proposed by Tayfun Gokmen and Wilfried Haensch [148] described in Section 5.3.2.

For simplicity, first only the compensation of a purely symmetric memristive perturbation is described.

Symmetric case

In the symmetric case, the term $G(w_{i,j}) = 0$. This simplifies the calculation of the memristively perturbed update with Equation 2.3 to

$$\begin{aligned} \Delta w_{i,j}^{act} &= \eta \left(\Delta w_{i,j} F(w_{i,j}) \right) \\ &= \eta \left(\frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) \frac{\partial a_j}{\partial w_{i,j}} \right) F(w_{i,j}) \\ &= \eta \frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) x_i F(w_{i,j}) \end{aligned} \tag{5.15}$$

The output of the previous layer $y_i^{(L-1)}$ in Equation 2.3 is here and in the following the input to the current layer x_i to simplify the notation. In equation 5.15 it can be seen that the usual three factors for the gradient calculation $\left(\frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) x_i \right)$ are multiplied by the perturbation term $F(w_{i,j})$ which depends in the incoming weights.

the idea of the compensating neuron model is to make the forward pass formulation of the neuron aware of this factor. This means that when calculating the gradient of the output y_j of the neurons with respect to the weight $w_{i,j}$, the result should contain $F(w_{i,j})$ as a multiplicative factor. This can be achieved by changing the formulation of the neuron in the following way:

$$\begin{aligned} y_j &= \sigma\left(a_j\right) \\ a_j &= \sum_i^I \left(w_{i,j} \cdot x_i \cdot F\left(\frac{1}{2} w_{i,j}\right)\right) \\ &= \sum_i^I \left(x_i \cdot \left(\alpha_s \cdot w_{i,j} + \frac{1}{2}\beta w_{i,j}^2\right)\right) \end{aligned} \tag{5.16}$$

The activation function σ remains the same, but the perturbation $F(\frac{1}{2}w_{i,j})$ is included in the accumulation of the activation a_j. Of course, this would increase the computational costs drastically, because now the accumulation of a_j no longer can be carried out by an analog VMM, which makes it impossible to perform this step directly in a memristive crossbar array. However, for from a mathematical perspective, this approach leads to a gradient of the weight in the backward pass like:

$$\begin{aligned} \Delta w_{i,j} &= \eta \left(\frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) \frac{\partial a_j}{\partial w_{i,j}} \right) \\ &= \eta \frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) \frac{\partial}{\partial w_{i,j}} \left(\sum_i^I \left(w_{i,j} \cdot x_i \cdot F\left(\frac{1}{2} w_{i,j}\right)\right)\right) \\ &= \eta \frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) \frac{\partial}{\partial w_{i,j}} \left(\sum_i^I \left(x_i \cdot \left(\alpha_s w_{i,j} + \frac{1}{2}\beta_s w_{i,j}^2\right)\right)\right) \\ &= \eta \frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) \cdot x_i \cdot (\alpha_s + \beta_s w_{i,j}) \\ &= \eta \frac{\partial \mathcal{L}}{\partial y_j} \sigma'(a_j) \cdot x_i \cdot F(w_{i,j}) \end{aligned} \tag{5.17}$$

It can be seen that the result now is the same as in Equation 5.15. This means that the compensated forward pass formulation is aware of the memristive perturbation that would be caused if the gradient is calculated with the uncompensated forward pass.

Asymmetric case

The asymmetric case is more complex than the symmetric one. For the symmetric case, the neuron is made aware of the memristive update perturbation such that when calculating the gradient of the output of the neuron with respect to the a weight in Equation 5.17, the differentiation of the activation a_j as given in the formulation of the compensating neuron 5.16 with respect to the weight yields the symmetric perturbation term $F(w_{i,j})$. This cannot easily be done for the asymmetric perturbation,

because the perturbation of the update for the asymmetric case contains the sign of the calculated gradient as listed in Equation 5.11. This causes an issue because there is no differentiable function with the sign function as derivative.

Therefore, a naive approach is taken, and the neuron formulation split into two cases: one for the compensation of a positive gradient and one for a negative gradient. Since both terms F and G are linear functions, they can be superimposed to another linear function H for both cases separately.

$$\begin{aligned} H(w_{i,j}) &= \begin{cases} H_+(w_{i,j}) & ,\Delta w_{i,j} >= 0 \\ H_-(w_{i,j}) & ,\Delta w_{i,j} < 0 \end{cases} \\ &= \begin{cases} F(w_{i,j}) + G(w_{i,j}) = (\alpha_s + \alpha_a) + w_{i,j} \cdot (\beta_s + \beta_a) & ,\Delta w_{i,j} >= 0 \\ F(w_{i,j}) - G(w_{i,j}) = (\alpha_s - \alpha_a) + w_{i,j} \cdot (\beta_s - \beta_a) & ,\Delta w_{i,j} < 0 \end{cases} \\ &= \begin{cases} \alpha_+ + w_{i,j} \cdot \beta_+ & ,\Delta w_{i,j} >= 0 \\ \alpha_- + w_{i,j} \cdot \beta_- & ,\Delta w_{i,j} < 0 \end{cases} \end{aligned} \tag{5.18}$$

When discriminating these two cases, the compensation can be calculated following the same approach as for the symmetric case. However, the problem remains that the gradient must be known during the forward pass in order to decide which case should be applied. This however changes the output and at least for multi-layer networks, it can influence the gradient and even flip its sign. This is a so-called chicken-and-egg problem that prohibits the direct application of the above described compensating neuron approach in a practical network.

Potentially, the sign of the gradient depends on the signs of all three factors in Equation 2.3. Since every activation function that is used in ANNs is monotonically increasing, the derivative of the activation function, which is the second factor, will always be positive. When choosing a ReLU activation function 2.1.2, it can even be guaranteed that the output of each neuron, which is the input to the next weight layer x_i, is also always positive. With this the sign of a gradient is solely dependent on the derivative of the loss.

$$\Delta w^{act} = \eta \Delta w \left(F(w) + G(w) sign\left(\frac{\partial \mathcal{L}}{\partial y} \right) \right) \tag{5.19}$$

With this information, it is possible to think of approaches to estimate the sign of the gradients in the forward pass. The idea is that even if not all estimates are correct, an effect of the compensation and therefore, an increase of the accuracy could be observed. Some approaches are explained, and results reported for a three-neuron-layer MLP with 784, 100, and 10 neurons per layer with two weight layers, training on the MNIST data set in the following:

- **Fixed-gradient assumption** The simplest approach is to choose one sign for each gradient before the forward pass. For statistical reasons it can be assumed that 50% of the gradient signs are estimated correctly. In theory, it is possible

to choose the same gradient sign for all gradients, but for balancing reasons, the signs are randomly picked with the same number of positive and negative signs. It is furthermore possible to choose once before the start of the training or before each forward pass. The last approach of course is more computationally costly. However, all methods for fixed-gradient assumption show no effect on the maximum accuracy that can be reached.

- **Double pass** In this method, two forward and backward passes are performed. In the first forward and backward pass, the uncompensated neuron model is used to determine the sign of the gradients. In the backward pass, the weights are not updated but the gradient signs stored. Subsequently, a second forward pass is executed with the compensated neuron model and the signs that have been determined in the first pass are used. Then the backward pass is used to actually update the weights. Also, this method does not lead to increased accuracies. The reason can be found when comparing the estimated gradient signs (calculated with the uncompensated model) with the signs of the gradient s in the second pass (calculated with the compensated model). It can be observed that for the investigated two weight layer network, the estimates for the second 100-10 layer are correct in 55% to 65%, but for the first 784-100 layer, only 18% to 25% of the estimated gradients are correct. This shows how strongly the compensation can alter the signals in the network, both in the forward and the backward pass.

- **Direct-Random Target Projection** Direct-Random Target Projection (DRTP) [151] is a method to calculated gradients already in the forward pass, making a separate backward pass obsolete for training. It was shown that the loss of accuracy with this method is small when training on common classification benchmark data sets like MNIST or CIFAR-10. While in standard backpropagation, the loss with respect to the last layer's output is propagated backward through the entire network, in DRTP for each layer separately a loss is calculated from the output of this particular layer and the trainer signal. Since the dimensions of the trainer signal and an arbitrary neuron layer's output may not match, a random projection matrix is set up for each layer to achieve equal dimensions for a loss calculation.

  In the compensating neuron approach presented here, DRTP is used as follows: The first neuron layer performs a forward pass with the compensating neuron model and the network inputs. The compensation needs an estimate of the gradient signs. For the first pass, this is randomly chosen. Before the output of the neuron layer is propagated to the next layer, the loss is calculated with DRTP. From this local loss it is possible to calculate $\frac{\partial \mathcal{L}}{\partial y_i}$ for each neuron i in this layer. As described above, this is sufficient to get the sign of the gradient of the weights. These calculated signs are now compared to the gradients that have been assumed before. Each falsely estimated gradient sign is flipped. Then the forward pass of this layer is performed again. This step is repeated until all

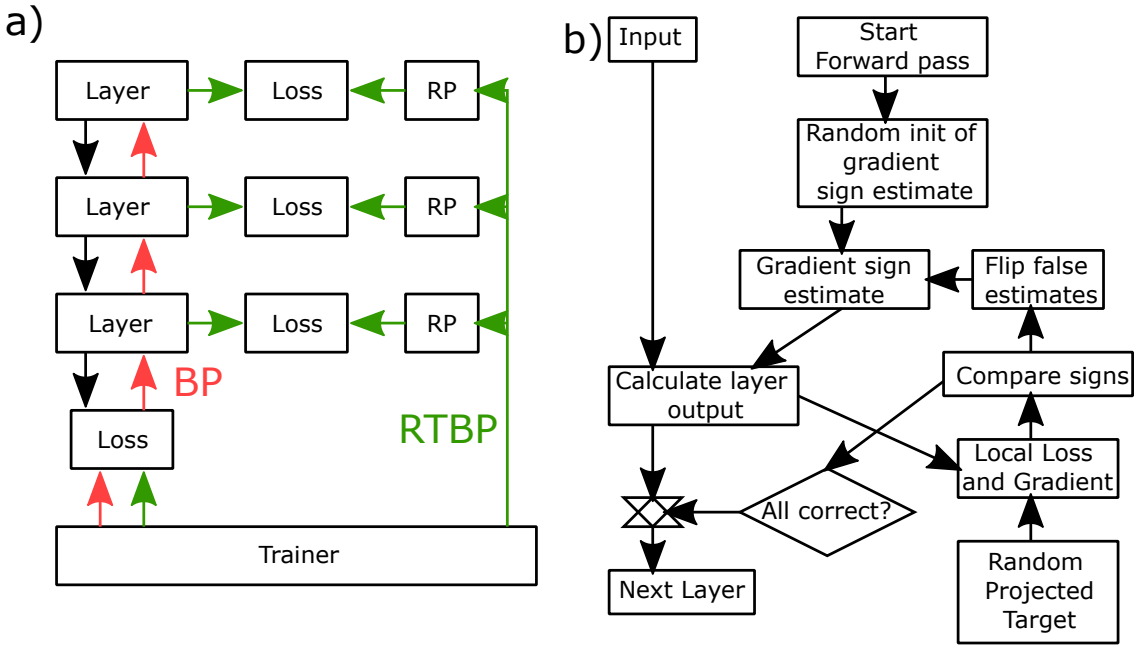

Figure 5.6.: **Direct Random Target Projection for forward determination of the gradient sign** a) Principle of DRTP. In conventional backpropagation (red), the loss is propagated backward from layer to layer in a serial fashion after the complete forward pass (black). For DRTP (green), each layer receives a random projection (RP) of the trainer signal and generates a loss signal directly after calculating its own output. b) For the compensating neuron approach the estimate of the gradient sign is tested by comparing the estimate with the sign of the local gradient. False estimates are flipped.

gradients have been correctly estimated. Only then, the gradients are applied, and the output of the neuron layer is propagated to the next layer.

With this approach in some trials indeed a certain increase of accuracy can be observed compared to the uncompensated neuron model. However, in many cases the issue occurs that the signs of gradients keep flipping back and forth because they influence each other via the compensating neuron function. This causes an endless loop. Since the forward pass has to be performed for each input sample separately, it is very likely that one sample leads to such an endless loop which causes the training to fail completely. Furthermore, even if the convergence to one gradient sign estimate is successful, the many repetitions of the forward pass create a large computational burden. In conclusion the use of DRTP is promising to be used together with the compensating neuron model, but the large computational complexity and the possibility that the training can fail due to oscillating gradient sign estimates, render this method not useful for practical application.

In conclusion for the implementation of a compensation in the neuron model which would compensate the memristive update dynamics already in the forward pass, it is shown that on a mathematical level a solution can be found, but at least for the asymmetric memristive dynamics case, the chicken-and-egg problem of the gradient signs which have to be known already in the forward pass constitutes an obstacle for practical implementation. Therefore, in the following the compensation of the memristive update dynamics in the backward pass by the synapses will be discussed.

5.4.2. Compensation of the update asymmetry

One other approach to compensate the update asymmetry is presented by Tayfun Gokmen and Wilfried Haensch with the TikiTaka algorithm [148]. In this approach, gradients from multiple gradient steps are accumulated on an auxiliary memristive array. The resulting net gradient sign then determines the direction of one single update pulse applied to the actual device array which is executed in a precise manner.

Here, a different approach for the compensation of the memristive update dynamics is presented which does not need an additional auxiliary memristive array. The memristive switching dynamics are modeled in the same fashion as described above.

As discussed in [142] the asymmetry between potentiating and depressing update steps has the largest impact on the loss of accuracy in the training with memristive devices. Therefore, the approach presented here is to mitigate this asymmetry by adjusting the pulse length of the applied update pulse.

When looking at the model in Equation 5.20, the gradient perturbed by the memristive dynamics is

$$\Delta g_{i,j}^{mem} = \Delta g_{i,j} \left(F(g_{i,j}) + sign(\Delta g_{i,j}) G(g_{i,j}) \right) \tag{5.20}$$

The asymmetry stems from the second part $sign\left(\Delta g_{i,j}\right) G(g_{i,j})$ of the factor, the gradient is multiplied with, which depends on the sign of the gradient. When the perturbed update is now multiplied with a factor $(F(g_{i,j}) - sign(\Delta g_{i,j}) G(g_{i,j}))$ it yields

$$\begin{aligned} \Delta g_{i,j}^{comp} &= \Delta g_{i,j}^{mem} \cdot (F(g_{i,j}) - sign(\Delta g_{i,j}) G(g_{i,j})) \\ &= \Delta g_{i,j} \cdot (F(g_{i,j}) + sign(\Delta g_{i,j}) G(g_{i,j})) \cdot (F(g_{i,j}) - sign(\Delta g_{i,j}) G(g_{i,j})) \\ &= \Delta g_{i,j} \cdot \left(F(g_{i,j})^2 - G(g_{i,j})^2 \right) \end{aligned} \tag{5.21}$$

The dependency on the compensated gradient has vanished due to the application of the third binomial formula. Therefore, this would lead to a symmetric update. This compensation approach is coined Binomial Compensation. However, it cannot be established that $F(g_{i,j})^2 > G(g_{i,j})^2 \forall g_{i,j} \in (g_{min}, g_{max})$. Therefore, the compensated gradient is further multiplied with $sign\left(F(g_{i,j})^2 - G(g_{i,j})^2\right)$ thus yielding

$$\begin{aligned} \Delta g_{i,j}^{comp} &= \Delta g_{i,j}^{mem} \cdot (F(g_{i,j}) - sign(\Delta g_{i,j}) G(g_{i,j})) \cdot sign\left(F(g_{i,j})^2 - G(g_{i,j})^2 \right) \\ &= \Delta g_{i,j} \cdot \left(F(g_{i,j})^2 - G(g_{i,j})^2 \right) \cdot sign\left(F(g_{i,j})^2 - G(g_{i,j})^2 \right) \end{aligned} \tag{5.22}$$

This equation however still leads to a dependency of the update step size on the current conductance. Therefore, the potentiation / depression measurement curves will still be non-linear. In the following, the effect of the compensation method on these measurements and on the training of ANNs is investigated.

Effect of binomial asymmetry compensation

Potentiation / depression measurement The gradient modulates the pulse length of applied update pulses. Therefore, one can investigate what effect the Binomial Compensation has on the pulse length. In Figure 5.7 a compensated and an uncompensated potentiation and depression experiment are compared. The measurements are conducted on $Ti/SrTiO_3$ devices. While the position of each point indicates the conductances before each pulse, the colors indicate the pulse length which was applied at this conductance. Figure 5.7a) shows the uncompensated measurement. Here potentiation and depression have a constant pulse length throughout the complete measurement of $10\mu s$ for the potentiation and $9.5\mu s$ for the depression direction. The difference results only from the measurement setup. From this measurement, the parameters for the F and G terms of the Gokmen & Hänsch model 5.20 are extracted. With these, the compensation term $(F(g_{i,j}) - sign(\Delta g_{i,j})G(g_{i,j})) \cdot sign\left(F(g_{i,j})^2 - G(g_{i,j})^2\right)$ can be calculated each time before the application of the update pulse. Besides shorter pulses in general, it can be seen that at the beginning of both the potentiation and the depression measurement, where the conductance steps are the largest, the pulses are shorter than at the end, where the conductance saturates. Looking for example at the high conductance region, this means that the conductance steps of the potentiation measurement become larger while the uncompensated steps are very small. At the same time, the pulse length of the depression measurement decreases which leads to smaller conductance steps in a region where normally the conductance steps are the largest. With both effects combined the conductance steps for the same conductance value therefore become more equal, which leads to a reduction of the asymmetry.

ANN training To investigate the effect of the binomial compensation and this asymmetry reduction on the training performance of ANNs with memristive layers, the same two-layer network as described in 5.1.4 is trained on the MNIST data set for 10 epochs. Again, four different initialization methods are investigated. The comparison of the recognition errors for compensated and uncompensated training for 100 instantiations of the network each can be found in Figure 5.8. For the naive initializations with both a uniform and a normal initialization distribution, the compensation has no effect. It can be assumed that the reason here is still the poor initialization which to compensate was not the target of the binomial approach. For the Xavier initialization, a significant increase of accuracy can be observed with the recognition error dropping from a median value of 20% to about 6%. With the Kaiming-style initialization the training with compensation achieves a median recognition error of below 4% and thereby an accuracy gain of more than 16%. Even more, the median accuracy of 96.7% reached here is equal to the floating-point accuracy. The combina-

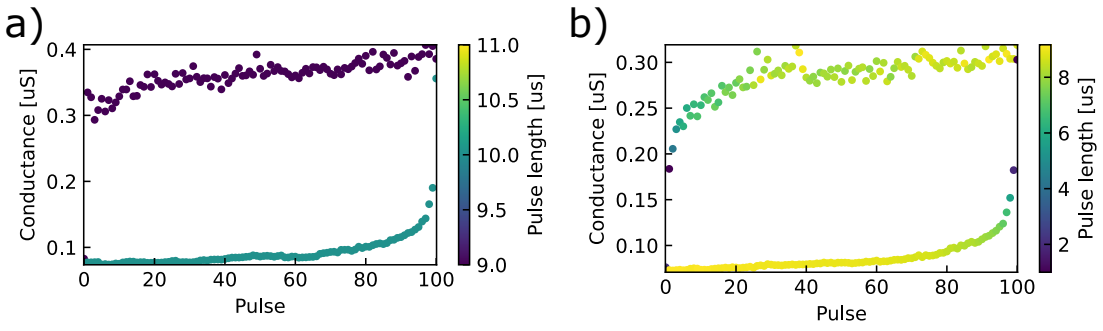

Figure 5.7.: **Effect of binomial compensation on pulse length**
a) Without compensation, all pulses have the same length.
b) When applying the binomial update compensation, pulses are shorter in regions where the conductance change is large, e.g., at the beginning of the potentiation and depression. This allows to reach more intermediate states in this region and decrease the steepness of the plasticity curves. The higher (for the potentiation) or lower (for the depression) the conductance gets, the longer the pulses become. In these regions, the conductance saturates. Longer pulses here maintain a certain change of the conductance while it stalls in the unsaturated case.

tion of Kaiming-style initialization and Binomial Compensation is therefore able to completely compensate the memristive update perturbation.

The compensation method can also be applied to different material systems and works for a range of pulse parameters. Figure 5.9 shows memristive ANN training results with update dynamics of $PCMO$ devices [2]. Figure 5.9a) shows the results for devices with a tungsten oxide interlayer. For depression voltages from $1V$ to 1.6 in combination with potentiation voltages from $2V$ to $2.6V$ accuracies between 88% and 94% for the uncompensated case are achieved. With the lowest potentiation and the lowest depression voltage slightly lower maximum accuracies can be seen than with higher voltages. However, with the Binomial Compensation applied, for all combinations of potentiation and depression voltage, an accuracy close to the floating-point accuracy is achieved. This results in a gain in recognition accuracy of 5% to 12%. For devices with a tantalum oxide interlayer, displayed in Figure 5.9b), also the accuracy with the compensation method is higher than without. Here floating point accuracy is not achieved. For devices with an aluminum interlayer, however in Figure 5.9 it can be seen that only for a small number of devices a better performance is achieved with the compensation than without. However, the cases where a lower accuracy is reached are those which even without compensation perform poorly. Similar to the case of the naive uniform and normal initializations, the compensation algorithm cannot compensate an anyway too low training performance.

[2]The measurement data for these experiments was provided by Alexander Gutsche, Forschungszentrum Jülich GmbH, Jülich, Germany.

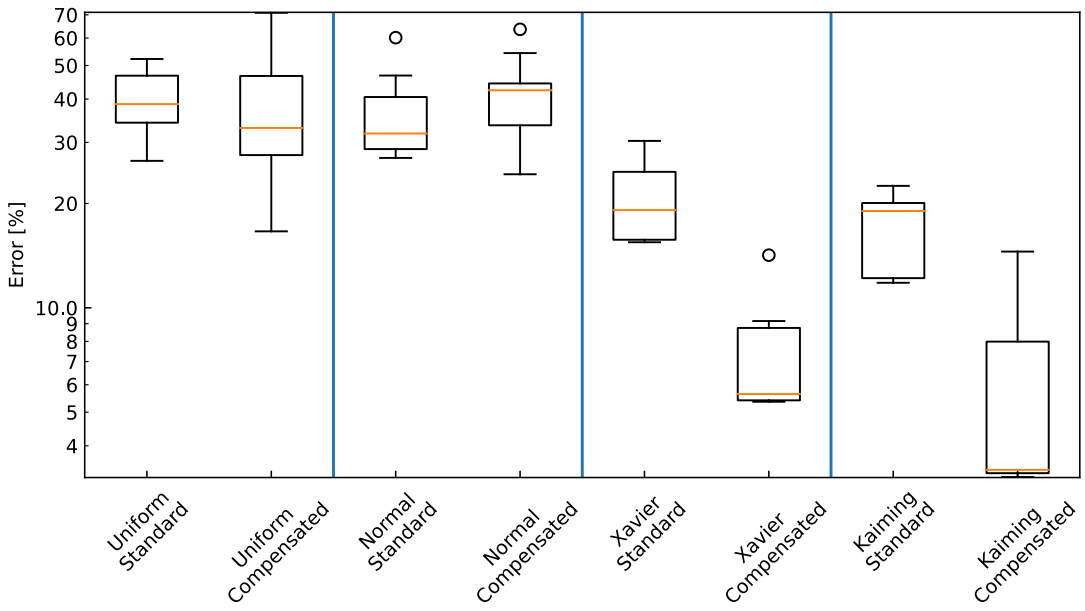

Figure 5.8.: **Effect of binomial compensation on networks with different initializations** For networks whose initializations are derived from Xavier or Kaiming initialization, a significant gain in accuracy (here on the MNIST data set after 10 epochs of training) can be observed. For the pure uniform and normal initializations which already initially showed poor recognition performance, no improvement is achieved.

5.5. Summary of Memristive ANNs

The conductances and update dynamics of memristive devices differ in many ways from the number representations that are used in conventional computers for weights in ANNs. These differences can significantly perturb the training process and hinder memristive ANNs from achieving the same performance in terms of accuracy as ANNs with floating-point weights. In this chapter solution strategies for two of these specific properties are proposed, namely, the finite and only positive conductance values range and the memristive update non-linearity and asymmetry.

The fact that memristive devices can only assume positive conductance values in a limited range and can only be initialized with limited precision makes it impossible to directly use advanced initialization methods like Xavier or Kaiming initialization which are crucial for a fast convergence of the training. Here, it is discussed how the neuron's activation function can be used to achieve the same scaling of forward and backward signals that was proposed in the respective works to preserve the magnitude of the signals in the network and avoid vanishing or exploding signals. It is shown that the use of these methods benefits the training success strongly.

The non-linearity and especially the asymmetry of the update of memristive weights

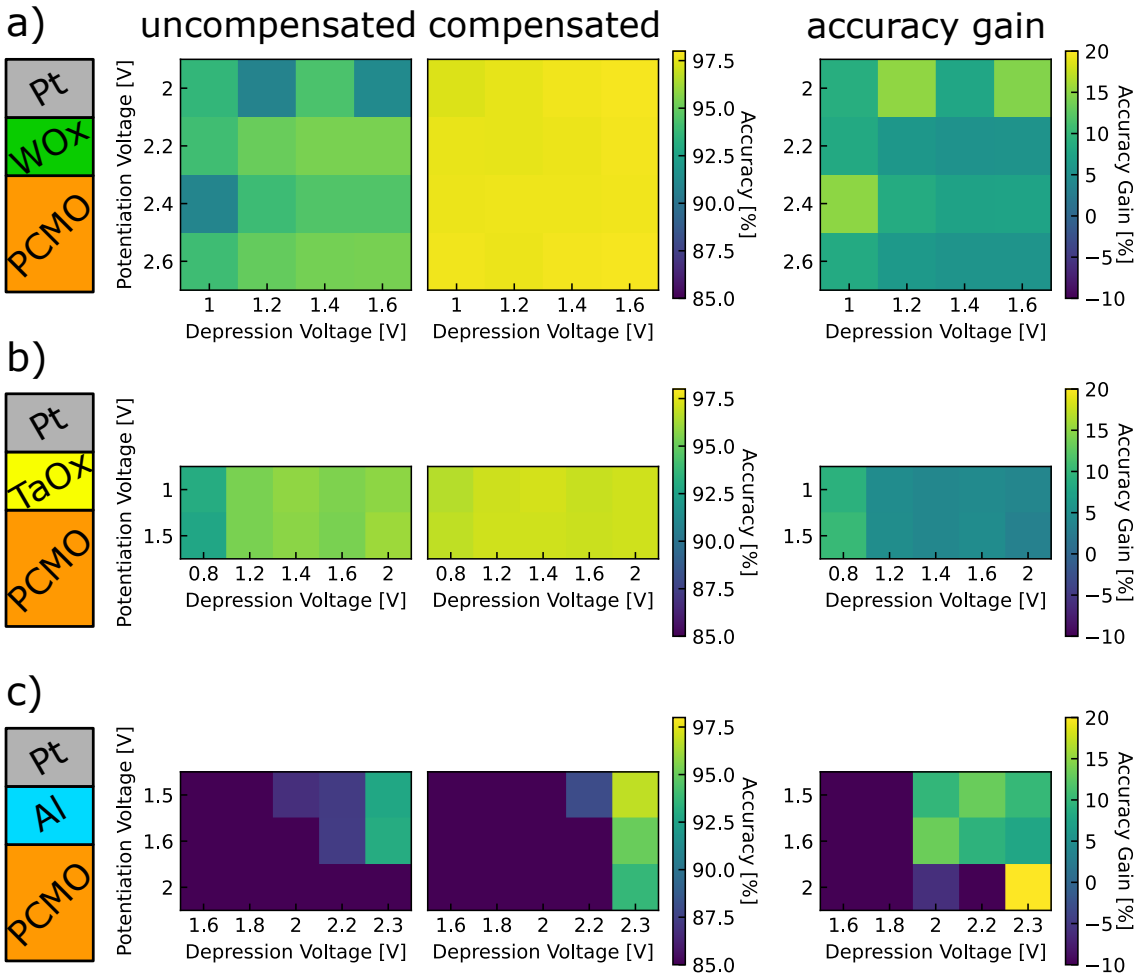

Figure 5.9.: **Binomial Backward Compensation for PCMO devices** With the update dynamics of three different PCMO material stacks(with a WOx (a)), TaOx (b)), and Al (c)) interlayer) and various V_{pot} / V_{dep} combinations ANN training simulations with the MNIST data set are conducted. Comparing the training with and without compensation, a gain of accuracy can be observed for most switching dynamics.

is another obstacle to achieving maximum network performance with memristive ANNs. To mitigate these special dynamics, compensation methods at the neuron and the synapse update level are investigated. Of these, the compensation of the synapse update in the backward pass yields the best results restoring the recognition accuracy that can be reached with floating-point devices.

6. Memristive Spiking Neural Networks

Artificial Neural Networks show promising performance when it comes to tasks that can hardly be solved by conventional deterministic algorithms. The implementation of memristive devices moreover can enable a more energy efficient execution. However, the underlying principles of these ANNs are still not those of the biological brain, both in terms of operation and training as they are derived from mathematical optimization. Neuromorphic computing in contrast aims at the integration of biological working principles into technical systems.

In this chapter it is therefore investigated if and how memristive devices can benefit also in bio-inspired networks and together with biological learning paradigms. Therefore, the example of a biologically inspired algorithm for sequence learning is taken. It makes sense to investigate this particular task because it is considered a core functionality of the brain. There are however many neuromorphic algorithms proposed that solve benchmark tasks for conventional ANNs like classification of the MNIST data set. This data is static as it consists of single frames that have no temporal correlation to their neighbors. These proposals aim at solving a non-biological problem with bio-plausible algorithms which is a questionable attempt that also produces significantly lower accuracies compared to what can be achieved with conventional static ANNs. Solving a static task with neuromorphic algorithm therefore seems not to be the best application of this approach.

6.1. Spiking Temporal Memory with memristive synapses

An exemplary task which is fundamental to the way the human brain perceives its environment and solves complex tasks is sequence learning [46]. Sequences in this context are temporally separated occurrences of events from a finite event space. In the case of the brain, events can be sensory impressions when perceiving the environment or reference frames when examining complex ideas and trains of thought [46]. In this work, different events are symbolized by alphabetic characters (A, B, C...) and can represent any kind of event space.

During the learning, different sequences of experiences are presented to the system. The goal is to learn the temporal connections between different events. An especially challenging sequence learning task are so-called high-order sequences. If two sequences

begin and end with different characters but share a common sub-sequence of characters in between, the trained network must learn that this common sub-sequence can occur in different contexts, which are the different initial characters, and result in different final characters. An example of two high-order sequences are {A, B, C, D} and {E, B, C, F}. Both share the sub-sequence {B, C} but depending on the first character, either A or E, the sequence terminates with D or F.

A trained network can solve different tasks. The most prominent are prediction, error detection and sequence replay. Prediction in this context means that after a number of characters has been presented to the network, it can indicate the next character before it is presented. In the case of the simple sequence {A, B, C, D} this means that after {B} has been shown to the network, it predicts C as the next character. For the above mentioned high-order sequences, the prediction is more complex. If {B, C} have been shown to the network, it must have remembered, if before this sub-sequence an A or E have been shown, and then predict either D or F accordingly. This is called a context-sensitive prediction. The sub-sequence {B, C} can occur either in the context of A or D. If a system can predict sequence elements, it can also detect errors in the presented sequence if this does not fit to any of the predictions.

6.1.1. Principles of the Hierarchical Temporal Memory (HTM) and biologically plausible SpikingTM

There are numerous algorithm and networks in the machine learning field tackling sequence learning [152] like Recurrent Neural Networks (RNN) [153], [154], Long Short term Memory (LSTM) [22], [155], [156] and Transformer Networks [157] which are proven to solve complex tasks. However, these systems are derived from mathematical control theory and lack deep biological inspiration.

Since sequence learning is so fundamental to the way the biological brain works, researchers are looking into bio-inspired algorithms to solves these tasks and get deeper understanding of the brain's functioning principles. One proposed algorithm is the Hierarchical Temporal Memory (HTM) [44].

The idea behind the HTM is that events are represented in Sparse Distributed Representation (SDR) which are sparse activations of neurons that are characteristic not only for a specific immediate event but also for its context. This sparsity is achieved by two mechanisms that limit the activity of neurons, the Spatial Pooling, and the Temporal Pooling. Spatial Pooling limits the activity of all neurons at the same point in time. The Temporal Pooling on the other hand limits the activity of the same neuron in time. Both together lead to a distribution of different events and contexts over all available neurons.

The original HTM however is a high level concept and does not explicitly rely on biological synaptic plasticity mechanisms for learning and spiking neurons. Therefore, Bouhadjar et al. devised a version of the Temporal Memory (TM) of the HTM which comprises these features, rendering the biologically plausible SpikingTM algorithm for

sequence learning. It was shown that the system can successfully learn, predict, and replay sequences within the boundaries of realistic parameters of biological neurons and synapses.

6.1.2. Memristive weight dynamics for synapses in SpikingTM

Parts of this section have been published in [141]. The discussed simulation were performed by Younes Bouhadjar (INM-6 / PGI-15, Forschungszentrum Jülich GmbH).

Due to the simplicity and the relatively low number of parameters, the SpikingTM algorithm is promising for deployment on a research-scale dedicated neuromorphic hardware system with memristive devices. The idea here is to use memristive devices as synaptic weights. Therefore, in a first step, it is investigated if memristive update dynamics are suitable to be used instead of the biological synaptic update plasticity.

Models for memristive switching dynamics

In order to do so, abstract fit models for two distinct types of memristive switching dynamics are defined, namely gradual and abrupt switching. The models give, in case of the gradual model, the change of the device conductivity $\Delta G_{i,j}$ connecting neuron j to i upon a potentiation or depression pulse or for the abrupt switching model the device conductance $G_{i,j}$ which depends on a hidden permanence value $P_{i,j}$. With the device conductance $G_{i,j}$ the synaptic current can be calculated following Equation 6.1. Here, a read noise term Z is added to the conductance $G_{i,j}$, yielding the noisy conductance $\overline{G}_{i,j}$. Multiplied with the read voltage V_{read}, the synaptic current is calculated. in this work, the read voltage can only assume one constant value or zero.

$$I_{i,j}(t) = (G_{i,j}(t) + Z) \cdot V_{\text{read}}(t) = \overline{G}_{i,j}(t) \cdot V_{\text{read}}(t), \tag{6.1}$$

For the gradually switching device, the model for the conductance change per plasticity event can be found in Equation 6.2, with the maximum / minimum conductivity of the memristive device $G_{\text{max/min}}$, the learning rates $\lambda_{+/-}$ for potentiation and depression, the non-linearity factors $\mu_{+/-}$, and a write noise term X. An exemplary evolution of the memristive device conductance can be found in Figure 6.1a). Starting from a low initial conductance, the conductance increases steeply during the first 25 potentiating / SET events and saturates afterwards around G_{max}. In the depression / RESET direction starting at event number 100, a less steep decrease of the conductance can be observed which saturates around G_{min}. The steepness of the potentiation and depression can be independently controlled by the $\mu_{+/-}$ parameters and the write noise parameter X is responsible for the noise on the conductance values.

$$\Delta G_{i,j} = \begin{cases} G_{\max} \cdot \lambda_+ \cdot \left(1 - \dfrac{G_{i,j}}{G_{\max}}\right)^{\mu_+} + X & \text{for potentiation} \\ -G_{\max} \cdot \lambda_- \cdot \left(\dfrac{G_{i,j}}{G_{\max}}\right)^{\mu_-} + X & \text{for depression} \end{cases} \tag{6.2}$$

The abrupt switching model in Equation 6.3 switches the device conductivity between G_{max} and G_{min} upon crossing a threshold θ_P to the permanence value $P_{i,j}$. The formula for the change of this permanence 6.4 has a similar shape as the conductivity of the gradual switching device with a minimum and maximum permanence $P_{max/min}$ instead of conductance. In this sense, both models contain a gradually changing value. For the gradually switching model, this value, the conductance, is directly observable, while the permanence of the abrupt switching model cannot be seen directly except for its effect on the conductivity in equation 6.3. The formulation of the abrupt switching memristive device resembles the model for the synaptic structural Spike Timing Dependent Plasticity (STDP) in [158].

$$G_{i,j}(t) = \begin{cases} G_{\max} & \text{if } P_{ij}(t) \geq \theta_P \\ G_{\min} & \text{if } P_{ij}(t) < \theta_P. \end{cases} \tag{6.3}$$

$$\Delta P_{i,j} = \begin{cases} P_{\max} \cdot \lambda_+ \cdot \left(1 - \dfrac{P_{i,j}}{P_{\max}}\right)^{\mu_+} + X & \text{for potentiation} \\ -P_{\min} \cdot \lambda_- \cdot \left(\dfrac{P_{i,j}}{P_{\min}}\right)^{\mu_-} + X & \text{for depression,} \end{cases} \tag{6.4}$$

In Figure 6.1b) the light gray curve represents the development of the permanence upon repeated potentiation / SET events for the first 100 events and depression / RESET events for the next 100. It resembles the evolution of the conductance in a) for the gradual switching memristive device. Upon the permanence crossing the threshold at 10, the conductance (in dark colors) switches between two distinct levels, except for a noise contribution by the read noise Z.

While the update dynamics of the synaptic weights are changed to memristive update dynamics, the learning rule remains a Hebbian plasticity rule. As indicated in Figure 6.1c) a potentiation / SET event is triggered if a synapse experiences a POST synaptic pulse shortly after a PRE synaptic pulse. Each PRE synaptic pulse alone triggers a small depression / RESET event. The depression events have less strong effect on the conductance because the learning rate λ_- for the depression direction is chosen smaller than λ_+ for the potentiation. Therefore, in the case of a POST synaptic pulse directly following a PRE synaptic pulse, an effective increase of the conductance can be observed.

Pulses are generated by spiking neurons which are sparsely and randomly connected. Excitatory neurons are organized in subpopulation and recurrently connected with inhibitory neurons leading to a winner-take-all (WTA) behavior where only a small number of excitatory neurons can generate a spike. This happens if a neuron receives somatic input, which is essentially an externally generated stimulus. If the

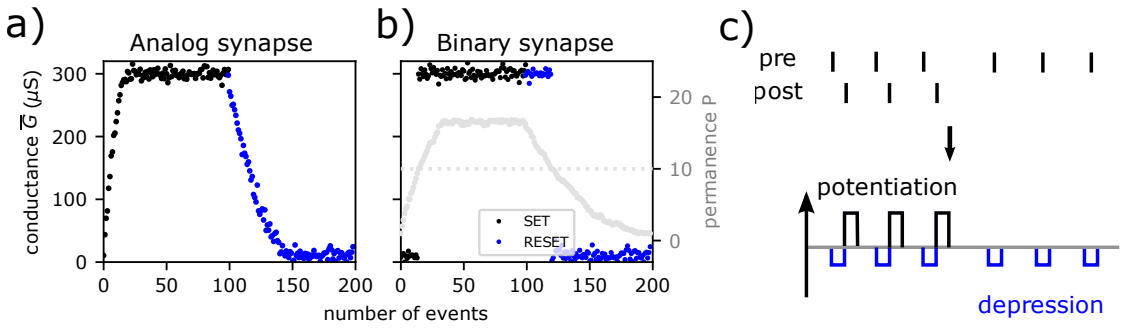

Figure 6.1.: **Memristive synapse models**. a) The conductance of the analog synapse changes gradually from a low conductive to a high conductive state upon prolonged application of potentiation pulses. Depression pulses decrease the conductance vice versa. b) In the binary synapse a hidden permanence value changes gradually with pulsing. The observable conductance changes suddenly upon the permanence crossing a threshold. Both models show cycle-to-cycle variation from read and write noise. c) In the SpikingTM algorithm each pre-synaptic pulse causes a small depression step. Only if it is followed by a post synaptic pulse, a strong potentiation is triggered. Figure adapted from [141]

neuron receives significant input from other neurons, called dendritic action potential (dAP), this neuron spikes earlier upon a somatic stimulus and inhibits other excitatory neurons in the same subpopulation from spiking via the inhibitory neuron. More details on the SpikingTM network and the neuron formulation can be found in [141], [158].

Sequence learning with memristive update dynamics

The network with memristive update dynamics is simulated in a NEST [159] environment in Python. The network is trained with four high-order sequences {A,D,B,E,I}, {F,D,B,E,C}, {H,L,J,K,D}, and {G,L,J,K,E} for up to 400 epochs or until zero prediction error of the final characters of all sequences is achieved. To ensure that the SpikingTM algorithm can cope with a wide range of memristive switching characteristics, parameter variation over the ON/OFF ratio, the learning rate, the non-linearity of potentiation and depression, and the write and read noise are performed. The results can be found in Figure 6.2 for the gradual and abrupt switching memristive synapse. In 6.2a) the average prediction error over multiple training runs for various ON/OFF ratios and potentiation learning rates can be found. The depression learning rate was also investigated but shows no effect within realistic boundaries. An error of zero means that all simulations are successful while an average error of one indicates that no training was successful after 400 training epochs. Intermediate values show that some simulations succeeded while other failed. For both synapse models, a lower threshold for the learning rate exists below which the training never succeeded, 15 for

the gradual switching and 10 for the abrupt switching synapse. Above these values the training converges for low learning rates of the potentiation λ_+. For higher values, the training can fail due to oscillations.

Update non-linearity of the potentiation and depression does not have a large impact on the training convergence for neither the gradual not the abrupt switching memristive synapse, as one can see in Figure 6.2b). Only for a high potentiation and no depression non-linearity the training performance of the abrupt switching synapse is impeded. Since a high non-linearity in the model goes together with a decreased learning rate of the respective update direction if the dynamics approach the saturation, this means that the potentiation increment becomes smaller than the depression decrement of the conductance. As Figure 6.1c) indicates, a potentiation can only happen together with a depression event. Therefore, if the non-linearity of the potentiation is high and for the depression low, it can be that the actually small depression cancels out the usually stronger potentiation and no effective increase of the permanence happens. In the case of the abrupt switching synapse this can mean that the threshold in Equation 6.3 is never crossed and the conductance of the synapse remains low. For the gradual switching synapse this effect is not so critical because here the observable conductance is increased and causes the synapse to generate a higher current than others anyway. An abrupt switching synapse with increased permanence, which is below the threshold, generates the same current as a synapse with minimal permanence. Therefore, in this case no selection process in the network can trigger and the training fails.

Write and read noise can be accepted up to certain levels, as shown in Figure 6.2. σ_w and σ_r here indicate the standard deviation of the Gaussian distributions the write and the read noise values for each update and read process are drawn from.

A common device failure mechanism that can be observed in endurance measurements of real devices are stuck-at faults where a device assumes a very high (stuck ON) or very low (stuck OFF) conductance and cannot be switched anymore. In order to test the robustness of the SpikingTM algorithm against such failures, a network is trained for 150 epochs. Then a fraction of all devices in the network is switched to the maximum or minimum conductance and its plasticity turned off. The training is then continued for another 150 epochs. In Figure 6.3a) it can be seen that the training converges with zero prediction error before the failures are injected. If 10% of all devices now go to a stuck OFF fault, the network can quickly recover by forming new connections bypassing any failed originally high-conductive synapse. For higher percentages (20% and 30%) most simulations cannot recover because too little connections exist to train the network. The results for stuck OFF faults are similar for the gradual and the abrupt switching synapse.

A similar experiment is performed for stuck ON faults where after 150 epochs a certain percentage of synapses becomes high conductive and fixed. One can see in Figure 6.3b) how the networks with both synapse types can recover after prolonged training from these errors for fault percentages of 10% and 20%. For a fault rate of 30% however, the network with gradual switching synapses does not recover anymore. Most of the simulations of the network with abrupt switching synapses however con-

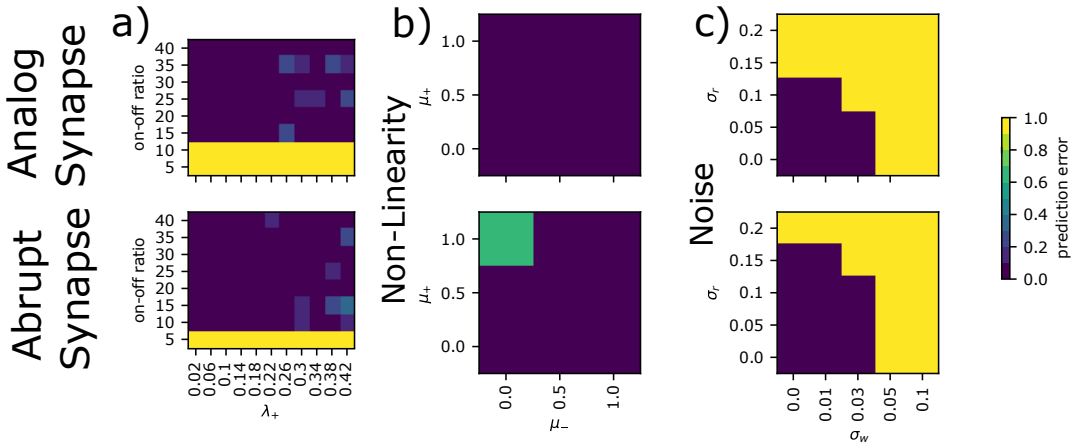

Figure 6.2.: **Influence of ON/OFF ratio, learning rate, non-linearity, and noise on the training with memristive synapses** The upper figure in each column shows the results for the analog synapse. Results for the abrupt switching synapse can be found in the lower plots. a) For both synapse models a lower threshold exists for successful training (15 for the analog and 10 for the abrupt synapse). A too high learning rate can impede the training success for both models. b) For most investigated combinations of potentiation and depression non-linearity the model converges. Only for the abrupt switching synapse a highly non-linear potentiation together with a linear depression causes training failures because the non-linearity indirectly influences the learning rate, and the switching threshold of the permanence cannot be reached in this case. c) The training with both synapse models can withstand certain amounts of read and write noise. Figure adapted from [141]

verge ever for this higher failure rate. This synapse type is more robust against stuck ON faults.

Summary of the investigation of memristive switching dynamics as synaptic update dynamics in the SpikingTM algorithm

These simulations show that memristive switching dynamics are indeed applicable as the synaptic update dynamics in the SpikingTM algorithm and a successful sequence learning can still be achieved with certain boundaries of memristive non-idealities such as update non-linearity and noise. The algorithm can also cope with high rates of stuck-at faults which makes it interesting to deploy the algorithm to real memristive neuromorphic hardware.

However, the results here are acquired using general purpose computers simulating complex biological neuron dynamics and synapse plasticity. Emulation of these computations on dedicated neuromorphic hardware would render complex circuitry and potentially a large computation overhead. Furthermore, above's simulations assume

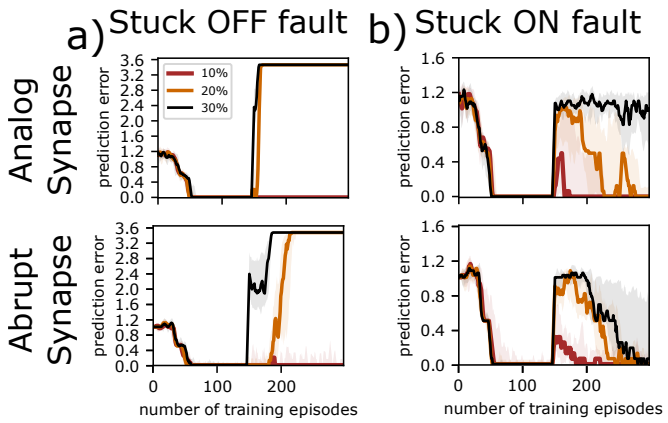

Figure 6.3.: **Influence of memristive device failure on SpikingTM** After training of the network for 150 epochs, stuck-at faults are injected into the network. An indicated fraction of devices is set to a very low (Stuck-OFF) or very high (Stuck-ON) conductance and its plasticity turned off. a) With both synapse models, the network can accept up to 10% of stuck-OFF devices. Higher amounts lead to persistent prediction errors. b) While with the analog synapse only up to 20% of stuck-ON devices can be compensated, the abrupt switching synapse model also allows 30% of the devices to be stuck-ON. Figure adapted from [141]

standalone single memristive devices and do not take the benefits and drawbacks of a complete memristive memory organization into account, such as a memristive crossbar array, on which the algorithm could potentially be performed more efficiently. Therefore, it is necessary to adapt the SpikingTM algorithm the specific properties of such a memory system.

6.2. MemSpikingTM - Spiking Temporal Memory on memristive neuromorphic hardware

Parts of this section have been published in [137].

In the previous section it is shown that the switching dynamics of memristive device can be used as synaptic update dynamics in the Spiking Temporal Memory algorithm for sequence learning.

In the next section a dedicated neuromorphic hardware system based on a memristive crossbar array is devised for the execution of an adapted version of this algorithm. The goal is to achieve an effective implementation exploiting or at least mitigating the specific switching dynamics of the employed memristive devices and efficiency in terms of energy consumption.

In a first step, the system is simulated in a SPICE environment using the commercial Spectre simulator. For large-scale simulations, the system is implemented on a pure logic level in MATLAB. The JART VCM model can be found on the EMRL website [1] and in the respective references.

The overall structure of the neuromorphic hardware system is depicted in Figure 6.4a) with the memristive crossbar array surrounded by peripheral neuron circuits. In contrast to the original SpikingTM, the system devised here is a synchronized one with global clock and epoch signals. The clock sets the operation time steps with each first section of the clock being a READ phase and the second phase an Update phase. The epoch signal indicates the beginning of each training epoch, in which all sequence that are supposed to be learned are presented once to the network via external stimuli.

Neurons are ordered in subpopulations of neurons. Here those neurons connected to the memristive crossbar array are called excitatory neurons, while one neuron per subpopulation is an inhibitory neuron. The exact functionality of all parts of the system will be explained in the following.

6.2.1. Network and algorithm

Memristive crossbar array

At the heart of the neuromorphic hardware system lies a memristive crossbar array. The structure of the crossbar array used here is a so-called vertical 1T1R array where the transistor gates are connected along the vertical columns in parallel to the vertical output lines. The input lines of the 1T1R devices are connected horizontally. In this work, these horizontal input lines are called source lines while vertical output lines are referred to as bit lines and transistor control lines as word lines.

The fundamental 1T1R cell of the crossbar array consists of one memristive device in series with a transistor, as shown in Figure 6.4b). The memristive device is modeled by the JART VCM v1b model [147] with the physical device parameters for HfO/TiO (HOTO) based memristive devices from [97]. The transistor model of a commercial 130 nm CMOS technology is used to model the access transistor. One source / drain terminal of the transistor connects to the bottom electrode of the HOTO device which is the active electrode (see Figure 6.4c)). The other terminal of the memristive device is the output of the 1T1R base and connects to a bit line. The second source / drain terminal of the transistor forms the input of the 1T1R device and goes to a source line while the gate contact of the transistor connects to a word line.

During the operation of the memristive crossbar array, voltage pulses are applied to the horizontal source lines. By opening or closing transistor gates via the vertical word lines, these pulses can be applied to specific memristive devices. During a READ process, all transistor gates are opened to the maximum by applying $5V$ gate voltage. Due to the configuration of the 1T1R device with the transistor connecting to the active electrode and because the HOTO devices switch in a counter eight-wise mode,

[1] `http://www.emrl.de/JART.html#Artikel_4`

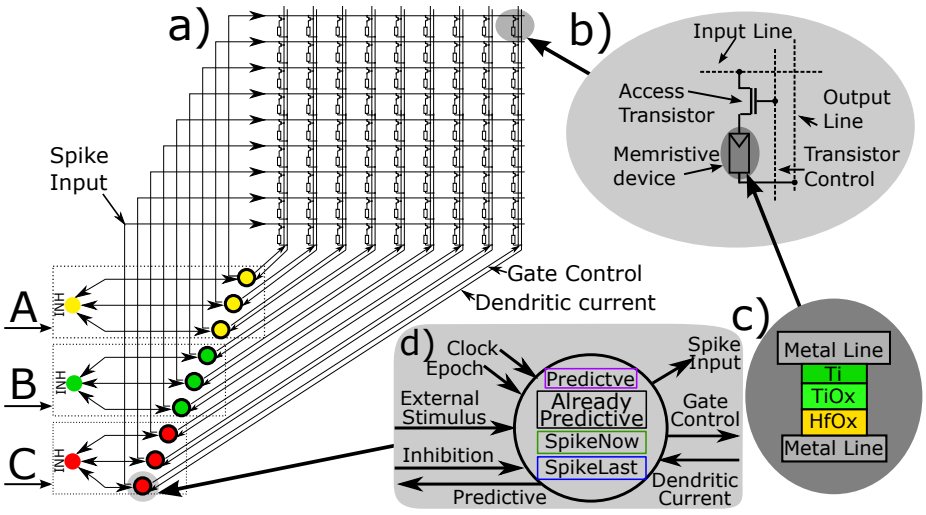

Figure 6.4.: **System overview of the MemSpikingTM** a) The core of the system is the memristive crossbar array. Peripheral neuron circuits apply inputs to the horizontal source line of the array, control the vertical word lines and receive input from the vertical bit lines. Neurons are ordered in subpopulations which are each affiliated with one character (A, B, C, D). Each subpopulation shares one inhibitory neuron and receives the respective external stimulus. Global clock and epoch signals are propagated to all neurons. b) The memristive crossbar array consists of 1T1R devices where one memristive device is in series with a silicon transistor. The input to each 1T1R device is connected to a source line, the output to a bit line, and the transistor gate is controlled via a word line. c) The memristive devices in sections 6.2 and 6.4 are hafnium oxide based devices or modeled after such devices. The device's material layer stack is sandwiched between two metal electrodes. It consists of a HfO layer with a TiOx and Ti layer on top.

a positive voltage can cause a depression of the device's conductance. If a potentiation is applied during the update phase, the transistor gate is not completely opened to form a current compliance and control the SET process.

If during a READ pulse all transistor gates are open to the maximum, the 1T1R array acts as a passive crossbar array and performs a vector matrix multiplication of the binary (V_{read} or $0V$) input vector and the memristive weight matrix. The result are the currents on the bit lines.

Neuron circuits and functionality

The generation of READ and update pulses and the analysis of the resulting bit line currents is performed by the peripheral neuron circuitry. Each excitatory neuron connects exclusively to one bit, word, and source line. Furthermore, each neuron receives an external stimulus, representing the somatic input. The neuron saves different

states, for example in SRAM or flip-flop logic. These states indicate if the neuron is predictive, was already predictive during the current epoch, if a spike is triggered, and if a spike was triggered in the last time step. How these states are set or reset will be explained in the following paragraphs.

The functionality of a neuron is sketched in Figure 6.5. The current on the connected bit line during a READ phase is called "Dendritic input current" to the neuron. As shown in 6.5a), if this current is above a certain threshold and the neuron was not yet in a predictive state in the current epoch, the neuron becomes predictive. Furthermore, if the neuron receives an external stimulus during the READ phase, it can trigger a spike, if it is not inhibited by the inhibitory neuron of its subpopulation. If the neuron is inhibited but at the same time became predictive in the previous time step, the inhibition is bypassed, and a spike is triggered anyway (see 6.5b)).

During the READ phase, all neurons apply a high voltage ($5V$) to their connected word lines and thereby open all access transistors completely. Those neurons that are triggered to spike send a READ pulse with V_{read} magnitude, which is positive, to their connected source lines. On the one hand, this creates the input vector for the above mentioned vector-matrix-multiplication, resulting in currents which can again predict neurons. On the other hand, due to the positive polarity of the READ pulse and its magnitude, a small RESET or depression of the conductance of all memristive device is caused which are horizontally connected to the spiking neurons.

Figure 6.5c) shows the neuron function during the Update phase. If a spike was triggered in the READ phase of the current time step before an Update phase, the neuron applies a gate voltage to the connected word line and partially opens all transistor gates on this word line. These transistors can now let a potentiating update pulse pass. Such pulse is applied by neurons which spiked in the previous time step to their connected source lines. This way, memristive devices are potentiated if they connect a neuron that spiked in the previous time step to a neuron which spikes in the current time step. Previously spiking neurons apply potentiation pulses on horizontal input line and currently spiking neurons open their vertically connected transistors, enabling the 1T1R devices for update. This implements a simple Hebbian learning rule.

Inhibition and Homeostasis

Two mechanisms limit the number and frequency of neurons becoming predictive.

Each subpopulation of neurons contains one inhibitory neuron. If a certain number of neurons become predictive at the same time, the inhibitory neuron triggers an inhibition signal. This hinders all other neurons in the same subpopulation from becoming predictive. This mechanism is responsible for the sparsity of connections between different subpopulations. In the original SpikingTM algorithm, inhibitory neurons share dynamics with excitatory neurons. In the hardware implementation presented here it is possible to realize the inhibitory neuron as a simple counter of the predictions in its subpopulation.

While the inhibition causes sparse activity in the lateral dimension, a homeostasis

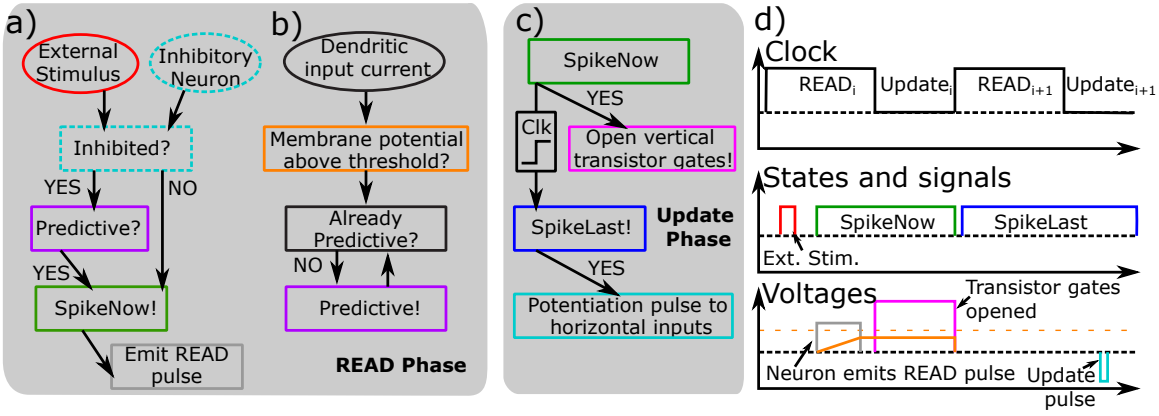

Figure 6.5.: **Neuron model for MemSpikingTM** a) In the READ phase at the beginning of each time step, the neuron senses the dendritic input current from the crossbar array on its connected bit line. If the input current overcomes a threshold and the neuron has not yet been predictive in the current epoch, it becomes predictive. b) If the neuron is externally stimulated and is not inhibited by the inhibitory neuron of its subpopulation, it emits a spike. An inhibition is overruled if the neuron is predictive. c) If a spike was emitted in the current time step's READ phase, the neuron opens its connected transistor gates in the Update phase. If a spike was emitted in the previous time step, the neuron emits a potentiation pulse to horizontal connected source line.

function limits the prediction activity of each neuron in time by letting it become predictive only once per epoch. If a neuron becomes predictive, this event is saved and prohibits it from becoming predictive again. This block is reset when the global epoch signal is triggered. The homeostasis can be compared with the refractory period of biological neurons. It was shown that the implementation of homeostasis can significantly benefit the training in artificial hardware SNNs with Hebbian-style update plasticity [160].

Training procedure

The described neuron and array operations in combination with the inhibition and homeostasis functions implement the complete learning system. Externally, only the stimuli must be applied. Therefore, in the following a walk-through of the training procedure of a simple training sequence on a small array will be explained.

The network consists of 6 excitatory neurons in 3 subpopulations and is displayed in Figures 6.7 and 6.8. The memristive crossbar array therefore has a size of a 6-by-6 matrix. The neurons input to the horizontal source lines and the output to the neurons is generated on the vertical bit lines. The word lines controlling the transistor gates are in parallel to these bit lines but are not drawn for improved visibility. In both subfigures a) and b), one epoch with the external stimulus of the training sequence

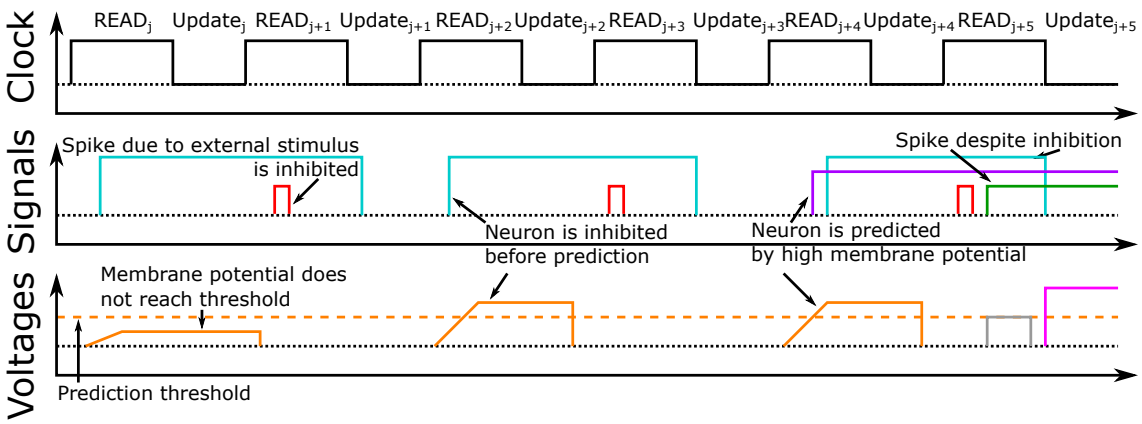

Figure 6.6.: **Prediction and inhibition functionality** The inhibition caused in time step j hinders the neuron from spiking in j+1 despite of an external stimulus. The same happens in time steps j+2 and j+3 even though here the membrane potential exceeds the predictions threshold, but after the inhibition is triggered. Only as in time steps j+4 and j+5 when the neuron is predicted before the inhibition is triggered, its effect is overruled.

{A, B, C} is shown. Here, a) displays the training on a pristine network and b) after some epochs of training. The first row in each subfigure explains the READ and the second row the Update phase of the training algorithm.

Stimuli on pristine network In the beginning, all memristive devices have a low conductance. When subpopulation A receives external stimulus in the READ phase, all neurons in A spike, because none was inhibited before as there was no previous time step. All synapses colored in red in Figure 6.7(upper left) receive a READ voltage pulse which causes them to produce an output current and slightly reduce their conductance. The READ current is, due to the initially low conductances, too low to cause any predictions.

In the following update phase those neurons that spiked in the recent READ phase now open their connected transistor gates. But since there was no previous spiking activity, no neuron emits an Update pulse.

In the following time step, subpopulation B is stimulated and all neurons spike, similar to the previous time step where A was stimulated. Also, during the Update phase all neurons in B open their transistor gates, as shown in the lower center figure in Figure 6.7. However, now the neurons in A emit a potentiating update pulse to their horizontal inputs because these neurons spiked in the previous time step. The synapses at the intersection of these horizontal input lines and open transistor gates along vertical word lines receive this potentiating pulse and increase their conductances. These synapses are those connecting subpopulation A to B.

In the third time step, displayed in the right-hand figure in 6.7, the READ and update phase after the external stimulus of C are shown. The situation here is similar

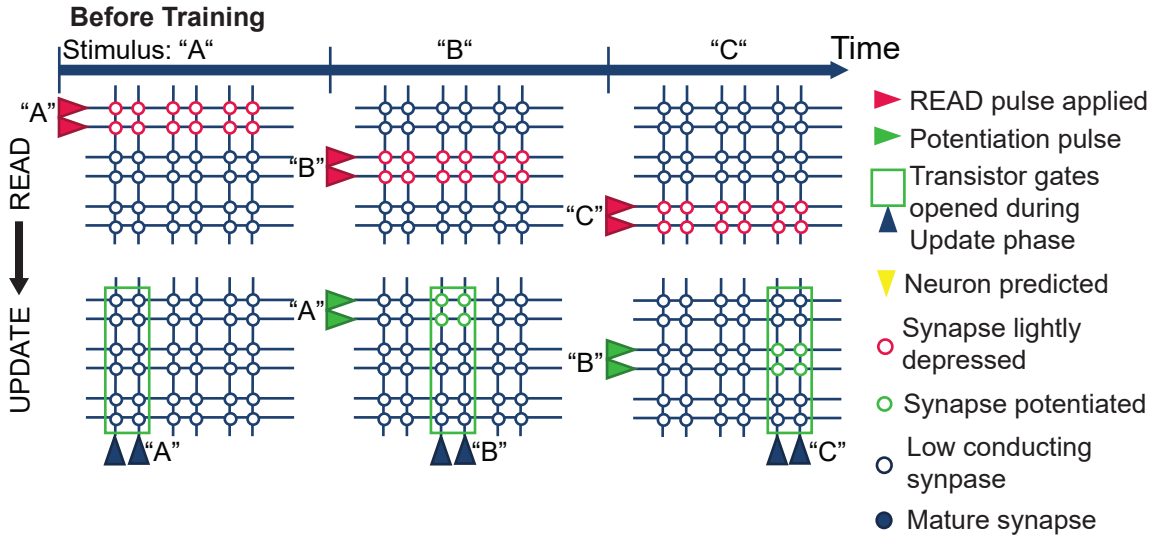

Figure 6.7.: **Exemplary walk through of the MemSpikingTM algorithm at the beginning of the training** External stimuli cause the complete respective subpopulations to spike. This causes a depression of all outgoing synapses (upper row). In the update phase (lower row), an effective update only takes place when opened transistor gates of the currently spiking subpopulation meet a potentiating update pulse from previously spiking neurons. Since all conductances are initialized low, READ pulses cause no high currents and therefore no predictions.

to the previous time step.

When superimposing all potentiation and depression events during this epoch (red and green cycles), it can be seen that all devices connected to spiking neurons experience a (small) depression while those connecting subpopulation that spiked immediately after another, receive a strong potentiation pulse. Taken all pulses into account, those synapses are potentiated, while all others are depressed. Therefore, the learning rule follows a Hebbian-style plasticity.

Stimuli on trained network This process is repeatedly executed. Potentiated synapses will increase their conductance, but, due to the inherent variability of the update in ReRAM devices, not homogeneously. After some epochs, arbitrary synapses will become mature, meaning upon a READ pulse they produce a current large enough to cause a prediction in another neuron. in Figure 6.8 these synapses are marked with blue filled circles.

If now A is externally stimulated and all neurons in A spike and emit a READ pulse, the synapse connecting the first neuron in A to the first neuron in B, which is mature, will produce a high current in the first neuron in B which then becomes predictive (see upper left figure in 6.8). With an inhibition threshold of one predictive neuron, the inhibitory neuron of subpopulation B now inhibits the second neuron in

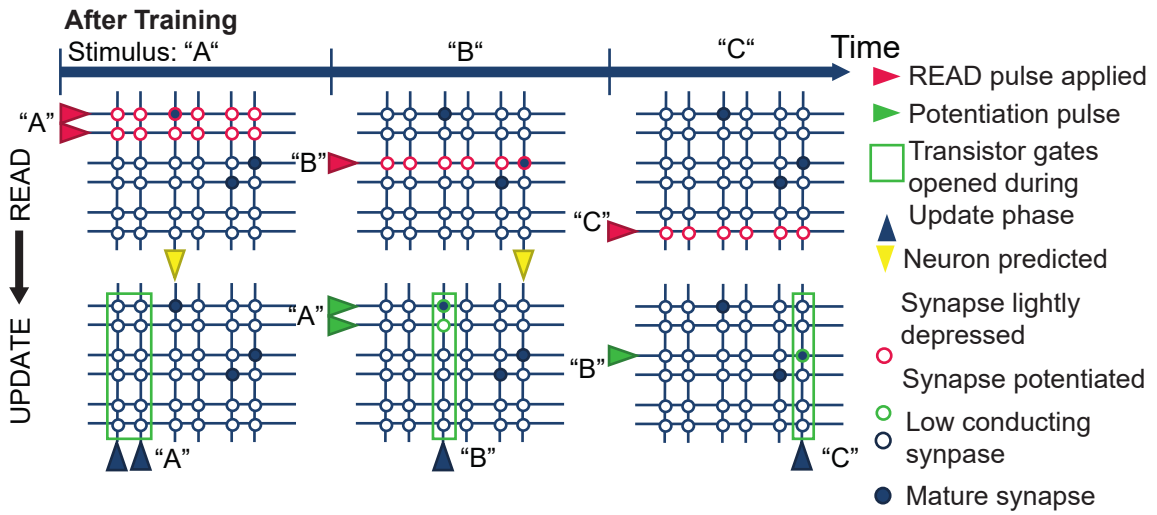

Figure 6.8.: **Exemplary walk through of the MemSpikingTM algorithm on a mature network** After some epochs of training, the spiking of neurons will by high conductive devices in the crossbar array cause high currents and therefore predictions in other neurons. In the upper left figure, the spiking of A causes a prediction in one neuron in B. Due to the inhibition, only predicted neurons spike. This causes sparse spiking and update activity in the network.

its subpopulation. The update phase of this time step is the same as for the untrained network.

When now subpopulation B is externally stimulated, the inhibited neuron does not spike and only the first neuron in B emits a READ pulse. This READ pulse again causes, in this cause, a prediction in a neuron in C. During the following Update phase, only the first neuron in B which just spiked opens its transistor gates, enabling them for update. This way, only connections to a sparse subset of neurons (in this case only a single neuron) can be increased while the others only receive the small depression during the READ phase and are slowly decreased. The situation after the stimulus of C is similar. This means that after some epochs of training and as soon as predictions are caused by mature synapses, the spiking activity in the network becomes sparse and with it the update activity. Only synapses that promoted the sparse activity are potentiated further while all other are depressed again.

Implementation in VerilogA and SPICE

Them memristive crossbar array is implemented as a matrix of unit cells of the type `TwoSel`, where each unit cell consists of one instance if the JART VCM model [2] in series with two analog gates, which can be either implemented as transistors from a

[2] `https://www.emrl.de/JART.html#Artikel_4`

foundry library or as simple switches. One of the gates is always open during simulation, effectively resulting in a 1T1R structure. for the MemSpikingTM simulations a structure is used where the bottom electrode terminal of the JART VCM device connects to a bit line terminal `BL` and the top electrode to an internal node `te`, which connects to the analog gates. The analog gates then connect to the source line terminal `SL`. With this orientation of the device, a positive read voltage from `SL` to `BL` cause a decrease of the JART VCM objects conductance. A voltage of the opposite polarity, more explicitly, if a positive voltage is applied at the `BL` terminal with the `SL` grounded, causes an increase of the conductance. This fits to the operation of the MemSpikingTM algorithm, where a positive read voltage is applied to the horizontal source lines, while the positive Update pulse is applied at the vertical bit line side.

The matrix is created in a `TwoSelArray` object. Each memristive device is initialized from a normal distribution of initial oxygen vacancy concentrations in the gap area `Ninit`. For all simulations shown here, the mean of this distribution has the value `0.05`, meaning a concentration of $5 \cdot 10^{24}/m^2$ oxygen vacancies. The standard deviation of the normal distribution is set to $5 \cdot 10^{25}/m^2$. With this, the resulting initial oxygen vacancy concentrations belong to a high resistive regime of the device's conductance.

The neuron functionality described above is realized in a `Neuron` object. The contact between the neuron circuits and the array is established by a `PulseGate` which would resemble the routing logic in a real hardware implementation. While the homeostasis function is implemented within each neuron by itself, the inhibition is organized by an `Inhibitor` which essentially implements a digital counter of the prediction values of all neurons within its subpopulation.

The global clock and epoch signals as well as the external stimuli are defined in a SPICE script which can be executed by a simulator, in this case the commercial SPICE simulator Spectre.

6.2.2. Context-specific prediction - Training of high-order sequences

Training of {A, B, C} {D, B, E}

To test the algorithms capability of context-sensitive prediction, a MemSpikingTM is trained in simulation with the two sequences {A, B, C} and {D, B, E}. These sequences share a common central character B but begin and end with different characters. Therefore, the prediction of the final character cannot be based on the second last alone but must keep into account all previous characters as well. These kinds of sequences are called high-order sequences.

In this experiment the system consists of a crossbar with 36-by-36 neurons and 36 excitatory neurons divided into 6 subpopulations (A to F) with one inhibitory neuron each. All memristive devices are initialized in a low conductive state. The input sequences {A, B, C} and {D, B, E} are externally stimulated sequentially once in every epoch.

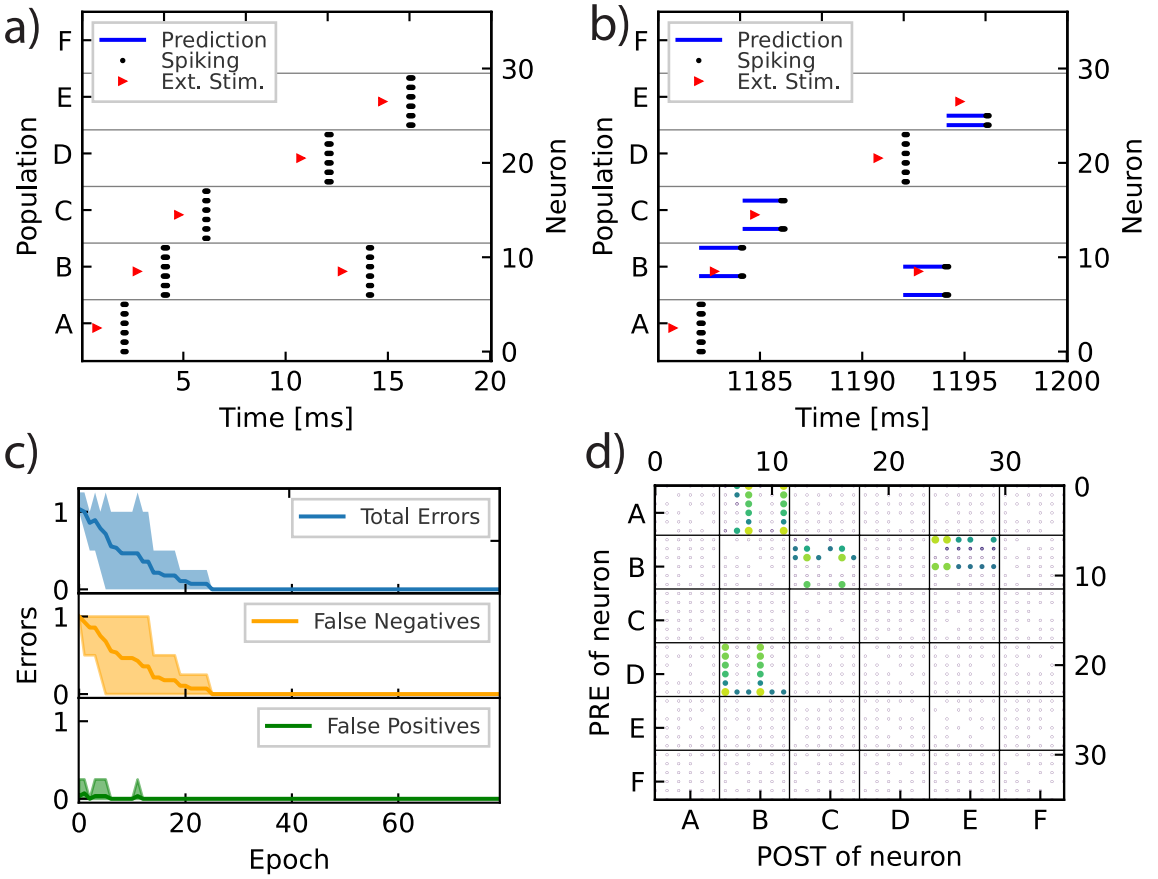

Figure 6.9.: **Training of the simple high-order sequences {A, B, C} and {D, B, A}** a) Initially, when the trained sequences are externally stimulated (red triangles) the complete respective subpopulations spike without generating prediction in any other neurons. b) After the training, the external stimulus of the first character of each subsequence still causes its complete subpopulation to spike. this spiking generates predictions in subpopulation B, which are different depending on the first character. The spiking of these predicted neurons then predicts either neurons in C or E. c) For ten different initializations, the training converges between five and 25 epochs of training with the majority of errors being missing (false negative) predictions. d) After the training, the conductance matrix shows a sparse pattern of strongly increased conductances while the majority remains low conductive as initialized.

Initially, the externally stimulus of a subpopulation causes the complete subpopulation to spike (see Figure 6.9a)). However, after 50 epochs of training, as depicted in Figure 6.9b), only the stimulus of the respective first character of each sequence leads to spiking of the complete subpopulation. The spikes emitted by subpopulation A cause a high current and therefore prediction in two neurons in subpopulation B before this character is externally stimulated. Since the inhibition threshold is set to 2, these two predictions will hinder all non-predicted neurons from spiking, when the character B is externally stimulated. Only the predicted neurons emit a spike and thereby cause prediction of two neurons in subpopulation C. The spiking of these neurons causes no predictions.

When now character D is externally stimulated, again, two neurons in B are predicted. Looking carefully, it becomes apparent that these are a different subset within subpopulation B than the two neurons which were predicted by the spiking of A at the beginning of the epoch. When now B is externally stimulated, the predicted neurons spike and cause prediction in subpopulation E. It is apparent that it depends on the specific subset of neurons in B that are predicted and spike, if their spike predicts either C or E.

Training metrics

In the beginning of the training, it can be seen in Figure 6.9 that none of the spikes causes a prediction in another neuron. Since the inhibition threshold is set to 2, for successful training it is expected that at least two neurons are predicted in the correct next subpopulation. Therefore, in the beginning the prediction error consist mainly of missing predictions which are called **false negatives**. During the course of the training, it is possible that false connections between neurons are established which causes predictions of neurons in the wrong subpopulation, i.e., in those which are not stimulated next. These predictions are called **false positives**. The sum of false positives and false negatives yields the total prediction error. For better comparability, the error is normalized with the number of errors in the first epoch.

In Figure 6.9c) the evolution of the prediction errors over the training epochs is displayed. For statistics, 10 simulations of the training with different initializations of the memristive conductances of the sequences {A, B, C} and {D, B, E} have been performed. The dark line in each plot represents the average error in the respective epoch while the light area spans between the maximum and minimum error. It can be seen that the majority of errors are false negatives. In the first 10 epochs, also some false positives can be observed but they never make more than 10% of the relative error. The average total error decreases gradually over 25 epochs where it reaches zero prediction error, indicating successful training at this point. For some instances of the training, success can already be observed after 5 epochs.

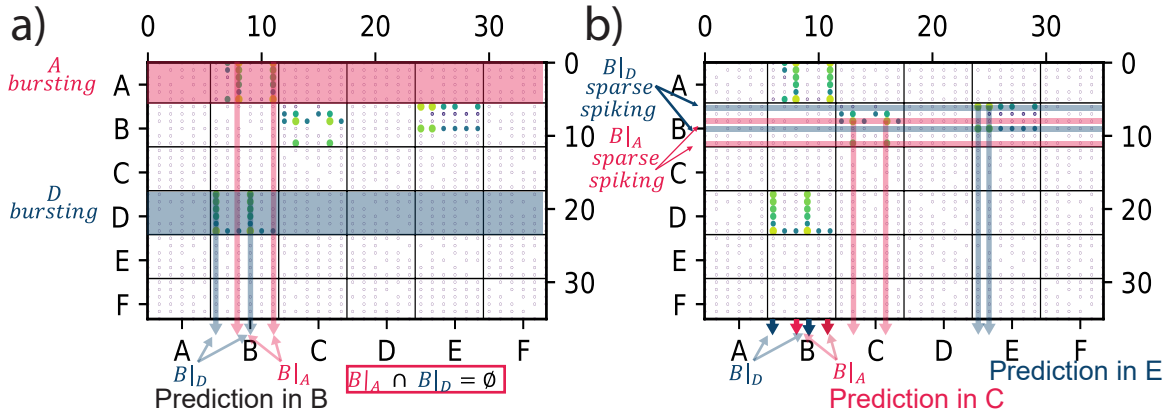

Figure 6.10.: **Origin of context specific prediction** a) When all neurons in A spike, the red colored synapses receive a READ pulse and generate a current according to their conductance. Only two branched in the submatrix leading from A to B are highly conductive and cause a prediction in two neurons in B which are called $B|_A$. If D spikes, the blue colored synapses receive a READ pulse which leads to a prediction in two other neurons in B ($B|_D$) via two conductive columns in the submatrix from D to B. b) If B is externally stimulated, only the predicted neurons spike due to the inhibition. The spiking subsets in B are disjoint. Their spiking either produces large currents by single high conductive devices in the submatrix B-to-C or B-to-E.

Origin of context

The key to context-dependent predictions lies in the prediction of different subsets of neurons within the same subpopulation by different spiking sets of neurons. The reason why different subsets of neurons in B are predicted by A and D can be found in the conductance matrix after training, which is displayed in Figure 6.9d). This matrix can be read that way that pulses are applied to horizontal lines and currents caused accumulate on vertical output lines. Most devices have a similarly low conductance as before the training. Looking at all output lines leading to neurons in the subpopulation B, it can be seen that there are two groups of highly conductive synapses in the sub-matrix leading from A to B and two groups leading from D to B.

When subpopulation A or D are externally stimulated and spike completely, either the highly conductive synapses leading from A to B or from D produce significant currents by the READ voltage. These currents accumulate on the vertical output lines and cause prediction in neurons in B. As indicated in Figure 6.10a) depending on whether A or D spike, these predictions are caused in different neurons. More explicitly, the subsets of neurons $B|_A$ and $B|_D$ predicted by different previous characters are disjoint.

When now B is externally stimulated, either $B|_A$ or $B|_D$ spike. If A was stimulated initially, $B|_A$ spikes and produces high currents in the sub-matrix leading from B to

C (see Figure 6.10b)), which causes neurons in C to become predictive. In this cause, B predicts C. If D was stimulated before B, it is $B|_D$ that spikes and predicts neurons in E via highly conductive synapses in the sub-matrix leading from B to E, meaning that in this case B predicts E.

Training of {A, B, C, D} {E, B, C, F}

The tasks of learning high-order sequences can be extended from a single shared character to a shared sequences of characters in the center of the sequences. in this case the context must be preserved over multiple stimuli. As an examples, the training for the two sequences {A, B, C, D} and {E, B, C, F} is performed in the same framework as above.

All memristive conductances are initialized with a low value. Therefore, in the beginning of the training, there are no predictions caused by the spiking of the complete subpopulations when the training sequences are stimulated externally (see Figure 6.11a)). However, after the training, it can be seen in Figure 6.11b) that after the initial spiking of A and the shared subsequence {B, C} predictions in D are caused, while with the initial character E the character F is predicted.

The training for these more complex high-order sequences takes longer on average as displayed in Figure 6.11c). The epochs-to-success range from 20 to 50 with an average of 45 epochs. It can also be seen that false positive prediction errors play a larger role for the more complicated learning task. Nevertheless, the majority of errors are false negatives.

A trained conduction matrix is shown in Figure 6.11d). Similar to Figure 6.9d) for the simple high-order sequences, there are high conductive columns in the submatrices leading from A to B and from E to B, with the columns leading to different subsets of neurons in B. When looking at the submatrix B-to-C, no complete columns of conductive devices can be found. Instead, the connection from B to C for the context of a previous stimulus of A is realized by single highly conductive synapses in the first , second, and fifth row of the submatrix B-to-C, leading to the first and last neuron in C. The first, second, and fifth neuron in B are exactly those neurons addressed by the highly conductive columns of the submatrix A-to-B and are the neurons that are predicted i the trained system (compare Figure 6.11b)). Similarly, high conductive devices in the fourth and sixth row of the submatrix B-to-E produce predicting currents when the fourth and sixth neuron in B spike after a previous stimulus of E. These different subsets of neurons in C again produce high current either in the submatrix leading from C to D or from C to F, differentiating the final prediction.

6.2.3. Parameterization of the MemSpikingTM network

Assuming a minimum device conductance of g_{min} and a maximum device conductance of g_{max}, the ON/OFF ratio $r_{ON/OFF}$ is defined as $\frac{g_{max}}{g_{min}}$. However, g_{max} will not be reached in a practical run, because even if a potentiation pulse drives a device to g_{max}, subsequent reading pulses will lower the conductance. Therefore, the effective

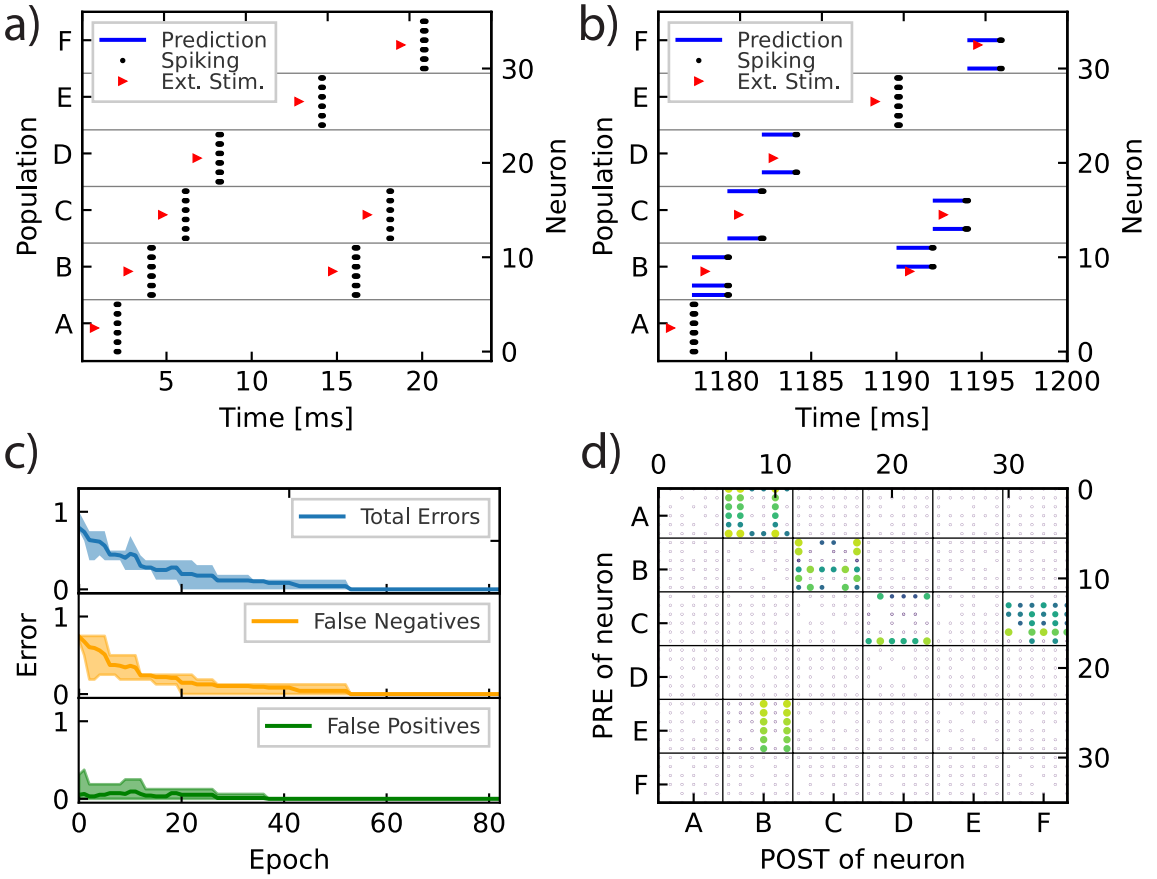

Figure 6.11.: **Training of the more complex high-order sequences {A, B, C, D} and {E, B, C, F}** a) External stimulus of the trained sequences initially causes all neurons in the respective subpopulations to spike. b) Only the respective first character of each subsequence spikes completely. The predictions in B and C are different depending on the context of either A or E spiking first which leads to a different prediction by C of either D or F. c) the training of these more complex sequences over ten instantiations takes between 20 and 50 epochs to converge with a larger number of false positive predictions during training. d) Also, for the more complex sequences a sparse matrix of high conductive synaptic devices leads to the context-specific prediction capability. Similarly, to the simper sequences, conductive pathways for either context can be found.

maximum conductance $g_{max,eff}$ depends in the READ depression step size δg_{read} and how often this device is read. $g_{max,eff}$ can therefore conservatively estimated to

$$g_{max} - \delta g_{read} * l_{sequence} \leq g_{max,eff} \leq g_{max} \tag{6.5}$$

Condition for the prediction current threshold

If assuming an array size of $n_{neurons}$ and a number of $n_{subpopulation}$ this yields, for equally sized subpopulations, a size of $n_{neuron,sub} = \frac{n_{neuron}}{n_{subpopulation}}$ per subpopulation. If a complete subpopulation spikes and all synapses connecting this subpopulation to a neuron have g_{min} conductance, the input current to this neuron is $g_{min} * V_{read} * n_{neuron,sub}$. A current threshold value I_{pred} indicating a prediction of this neuron must be higher than this value.

After learning, only predicted neurons spike if their subpopulation is stimulated, because the others are inhibited. This means that only th_{inhib} neurons emit a READ pulse. The connecting synapses from these neurons to the next-to-predict neurons are mature and have a conductance of $g_{max,eff}$. Therefore, the maximum current to each neuron which is supposed to be predicted is $g_{max,eff} * V_{read} * th_{inhib}$. The current threshold I_{pred} must be chosen smaller than this value. To summarize, the prediction threshold must fulfill the following condition:

$$g_{min} * V_{read} * n_{neuron,sub} < I_{pred} < g_{max,eff} * V_{read} * th_{inhib} \tag{6.6}$$

This condition can of course only be fulfilled if the lower boundary is smaller than the upper boundary for I_{pred}.

Relating device parameters to subpopulation and task

$$\begin{aligned} g_{min} * V_{read} * n_{neuron,sub} &< g_{max,eff} * V_{read} * th_{inhib} \\ g_{min} * n_{neuron,sub} &< g_{max,eff} * th_{inhib} \\ \frac{n_{neuron,sub}}{th_{inhib}} &< \frac{g_{max,eff}}{g_{min}} \end{aligned} \tag{6.7}$$

This results in a relation between the effective ON/OFF ratio $r_{ON/OFF,eff} = \frac{g_{max,eff}}{g_{min}}$ and the parallel capacity of the network $\frac{n_{neuron,sub}}{th_{inhib}}$. This capacity determines how many parallel conductive pathways can form through a subpopulation of a given size with a given inhibition threshold. The practical implication of this capacity is that for each occurrence of the same sequence element in a training sequence, at least th_{inhib} possible conductive pathways must be present. If a character appears k_{rep} times in the sequences to learn, the parallel capacity must be larger than this number:

$$\frac{n_{neuron,sub}}{th_{inhib}} > k_{rep} \tag{6.8}$$

	Counter Task	Advanced Counter	Occluder Task
$r_{ON/OFF}$	21		
$g_{init,Mean}$	$5\mu S$		
$g_{init,std}$	$1\mu S$		
δg_{READ}	$-0.12\mu S$		
δg_{Update}	$1\mu S$		
$n_{subpopulation}$	6	6	9
$n_{neurons,subpopulation}$	20	20	15

Figure 6.12.: **Parameters for large scale simulations**

Network layout process

The result leads to a trade-off: The inhibition threshold can be raised to account for a lower effective ON/OFF ratio, but this lowers the parallel capacity. This even allows a relationship between the parallel capacity and the effective ON/OFF ratio with

$$\frac{g_{max,eff}}{g_{min}} > k_{rep} \tag{6.9}$$

This relates the device parameters g_{max}, g_{min}, and δg_{read} directly to the parameters $l_{sequence}$ and k_{rep} of the task to learn.

If the device properties fulfill the task's demands, the size of the subpopulations and the inhibition threshold must be chosen according to the inequality 6.7. Then, condition 6.6 can be satisfied and a threshold value can be chosen accordingly. Finally, the number of different sequence elements determines the number of subpopulations.

6.2.4. Large-scale sequence learning benchmarks

SORN benchmark

To investigate the functioning of MemSpikingTM on a larger scale, a simplified version of the algorithm is implemented in MATLAB. To this extend, a linear potentiation and depression behavior with a saturation at a minimum conductance g_{min} and g_{max} for the memristive device conductance is assumed. The conductance values are stored in a matrix C.

To calculated output currents of the network (respectively input currents to the neurons) i_{array} at time step t, the conductance matrix C is multiplied with a vector s_t where all spiking neurons result in V_{read} entries while all other vector values are zero.

$$\begin{pmatrix} s_0 \\ \dots \\ s_{n_{neurons}} \end{pmatrix}, \text{ with } s_i = \begin{cases} V_{read} & \text{, neuron i spiking} \\ 0 & \text{, } neuron \text{ i not spiking} \end{cases} \tag{6.10}$$

Results above a $I_{prediction}$ threshold can lead to the respective neuron becoming predictive. If more neurons than the inhibition threshold $n_{inhibit}$ per subpopulation

receive input current above the prediction threshold, only the $n_{inhibit}$ neurons with the highest input become predictive. Predictive neurons are listed in the group p. The next spiking vector s_{t+1} is the intersection of the groups p of the predictive neurons and neurons receiving external stimulus in time step $t + 1$. The network parameters used for the following simulations are listed in Table 6.12.

Counter Task A common challenge for sequences learning algorithms is the counting task as described by Lazar et al. [161]. In this task the network must learn the spatio-temporal relation between two characters over the repeated occurrence of another character. Exemplary, the sequences {A, B, B, ..., B, C} and {D, E, E, ..., E, F} provide this kind of task. The number of repeating central characters defines the task's complexity n.

When training two sequences with complexity $n = 10$ to a network with six subpopulations and 20 neurons per subpopulation and repeating this for 100 instantiations, the training is always successful and converges to zero errors after around 100 epochs of training as can be seen in Figure 6.13a). The total number of errors (in blue, dark line is the average and light areas span between maximum and minimum errors) increases at the beginning. This is mainly due to the occurrence of false positive predictions (green). The number of false negative predictions decreases gradually over the course of the training.

Advanced Counter Task The Counter Task can be made even more complicated by demanding the network to learn the relationships between first and last characters if the central repeating character is the same in both sequences, like {A, B, B, ..., B, C} and {D, B, B, ..., B, E} where the measure of the complexity of the task is the number of repeating characters in each sequence. If B repeats n times, the complexity is n.

This task is tested also for 100 instantiations with the same network as described above. The more complicated task can also be solved in all trials, as visible in Figure 6.13b). Similar to the simple Counter Task, the number of false negative predictions decreases gradually over the training. But in contrast, the false positive predictions around epoch 50 make up for a significantly higher fraction of the total error than the false negative. The training also on average takes longer to converge (up to 120 epochs). The variability of epochs-to-solution is higher than with the simple task.

Occluder Task The network's ability to learn spit-temporal connections can be demonstrated by another task suggested in [161], the so-called Occluder Task. This task consists of the training of four sequences. Two are series walking through all subpopulations but one in opposite direction like {A, B, C, D, E, F, G, H} and {A, B, C, D, E, F, G, H}. Furthermore, two series are trained in parallel that occlude the connection between the first and last character in the first two sequences by repeating a so far unseen character. These sequences in this case can be {A, I, I, I, I, I, I, H} and {H, I, I, I, I, I, I, A}. These four sequences are trained to a MemSpikingTM network

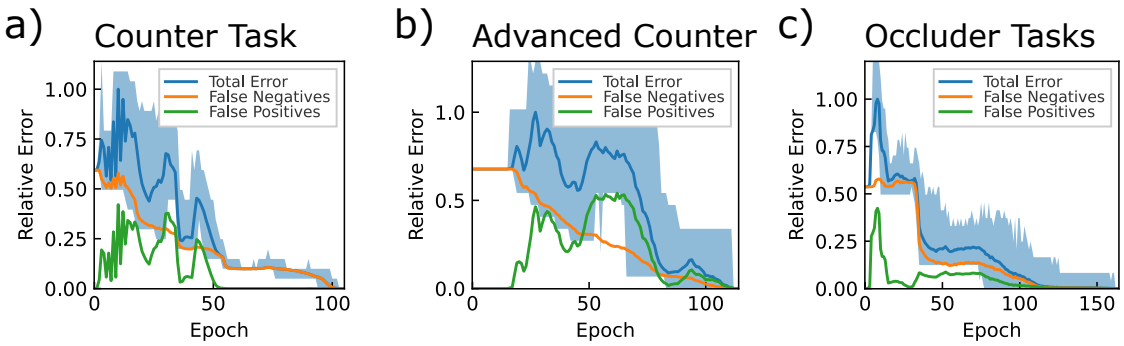

Figure 6.13.: **Sequence learning benchmark tasks** a) Counter Task[161]. The system must be able to count the number of occurrences of a character. b) Advanced Counter Task. In addition to counting the number of events, the system must remember different contexts. c) Occluder Task[161]. Two sequences running through all sequence characters (A to H and H to A) are occluded by two sequences with the same first and last characters.

with nine subpopulations with 15 neurons each. The resulting errors over the course of the training can be found in Figure 6.13c). At the beginning, the number of false positive errors increases up to 50% of the total error. The number of false negatives stalls up to epoch 40 before sharply dropping and then declining gradually. Most instantiations converge around epoch 110, but some take more than 160 epochs.

Summary of benchmark simulations These examples show that the MemSpikingTM algorithm can successfully solve complex sequence learning tasks also on a larger scale. Compared to the results for the Counter Task and the Occluder Task by the MEMSORN algorithm [162] and the SORN algorithm [161], a better performance in terms of higher accuracy with a smaller network and faster convergence time can be observed. While the MEMSORN algorithm with 200 neurons still remains a prediction error, MemSpikingTM converges always successfully with a network of only 135 neurons. On the Occluder Task, the SORN algorithm training converges after 6.000 epochs and 200.000 time steps while MemSpikingTM solves the same task in 110 epochs with 440 time steps.

6.3. Energy-efficient sequence learning

A key goal for the implementation of memristive devices with ANNs and neuromorphic algorithms is the prospect of energy savings during training and operation compared to the use of conventional computing paradigms and memory technologies. These two aspects however need to be distinguished. Energy efficiency can one the one hand be achieved by using beneficial algorithms and compute substrates and

computer architectures and on the other hand by implementing these with low power technologies.

In terms of the sequence learning discussed here, the first aspect would mean the usage of the SpikingTM algorithm instead of a conventional ANN system like a transformer network for sequence learning and executing the SpikingTM algorithm on dedicated neuromorphic hardware instead of simulating it with NEST on a conventional von-Neumann computer. For these systemic differences it is hard to quantify a difference in energy consumption because it would require the solution of the same tasks and objective energy metrics for which it must be possible to quantify the energy cost of each operation in a compute system and if and how it can be attributed to the solution of the task or another process. For example, it is hard to say if a memory refresh in a DRAM block belongs completely to the execution of a particular algorithm.

Therefore, in this thesis we look at a more fundamental level at the difference of energy consumption by using different memory technologies with the same algorithm. in particular, it is investigated how the use of memristive devices benefits the energy consumption in the operation of the MemSpikingTM algorithm in comparison to conventional memory technology. The benchmark for comparison here is floating gate transistor memory because it shares basic properties like non-volatility with the memristive devices. For simplicity, it is assumed that each memristive device can be replaced by one floating gate device.

Metrics for energy consumption

The power consumption of the system is extracted from the simulation of the high-order sequence training described above. For the comparison however only the contribution of the memristive crossbar array is considered. This has multiple reasons: Since no actual implementation of the neuron functionality is proposed, the power consumption of the periphery would strongly depend on the actual circuit and moreover on the technology node it is realized with. Furthermore, the periphery is the same for both memristive and floating gate memory implementations. Adding its contribution would therefore only add an offset to the power consumption of the actual memory block.

The electrical power that is input to each terminal k of the memristive array can be calculated by multiplying the transient current and voltage at this terminal $p_k(t) = v_k(t) \cdot i_k(t)$. The current $i_k(t)$ and the voltage $v_k(t)$ can be extracted from the simulation data. The output terminals during the simulation are always put to (virtual) ground. Integrating this value over the time of an epoch p from $t_{start,p}$ until $t_{end,p}$ and summing up these values for all input terminals yields the total energy consumption during this epoch.

$$P_{total}(p) = \sum_{k} \int_{t=t_{start,p}}^{t_{end,p}} p_k\left(t'\right) dt \tag{6.11}$$

The operation of floating gate memory is in a simplified way divided into a read and

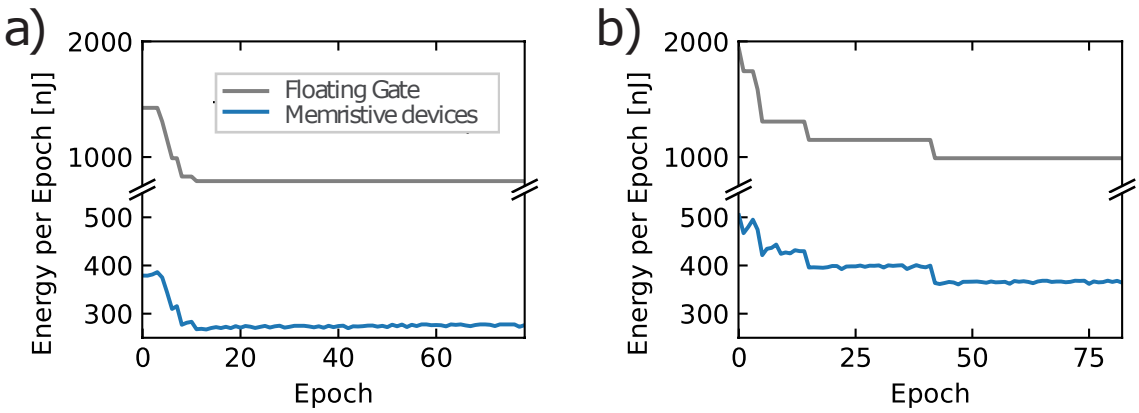

Figure 6.14.: **Energy consumption during training of high order sequences** a) Sequences {A, B, C} and {D, B, A}. The use of memristive devices shows a 66% energy benefit over the same algorithm executed with floating gate memory devices. b) Sequences {A, B, C, D} and {E, B, C, F}. Also, for the more complex high order sequences, a 60% lower energy consumption compared to an equivalent floating-gate implementation is observed.

a write operation. For these operations respectively, the energy consumption can be estimated from literature [163], [164]. For each read of a device 100pJ/bit and for each write 1nJ/bit is assumed. A read event is triggered when the memristive device which is then replaced by the floating gate transistor is read out with the read voltage and an update even is triggered each time the respective memristive device changes its conduction. The energy consumption is accumulated for each epoch p.

Energy-consumption during high-order sequence learning

The energy consumption per epoch is investigated for both high-order sequence learning tasks. In Figure 6.14a) the blue line shows the energy consumption per epoch for the simulated realization with memristive devices. It can be seen that initially about $400nJ$ are consumed per epoch. This value drops to $300nJ$ until epoch 10. This is also the epoch at which the prediction error decreases to zero and the training converges. the decrease of the energy consumption stems from the activity of the network becoming sparse. At the beginning, all neurons of each stimulated subpopulation spike which causes all connected synaptic devices to produce a read current. In the update phases, this leads to an update of the complete sub-matrices connecting subsequently spiking subpopulations. A soon as prediction start to arise, only a small number of neurons per subpopulation spike. This decreases the read and update activity. The effect of sparse activity can be observed even stronger in the training of the more complex high-order sequences depicted in Figure 6.14b). Here, distinct steps at epochs 13 and 40 indicate where the activity of subpopulations becomes sparse.

In gray the energy consumption of the floating-gate equivalent is drawn. It can be

seen that here the same steps exist which indicates that the algorithm itself decreases the system's energy consumption during training. However, for both training sequences, the energy consumption of the floating-gate equivalent is significantly higher than for the memristive device implementation, for both cases about a factor of three to four. Considering that for this comparison only a 1bit precision of the floating gate devices is assumed and according to [163] the energy consumption per operation scales with the number of represented bits, it can be assumed that a practical implementation with floating-gate devices would consume even more energy.

In conclusion this analysis of the energy consumption during sequence learning shows on the one hand how the MemSpikingTM algorithm decreases the energy consumption during training by making the neuron activity sparse. Furthermore, the application of memristive devices as storage devices for the synaptic weights renders beneficial in comparison to another non-volatile memory technology.

6.4. MemSpikingTM on a real memristive crossbar array

MemSpikingTM is designed to implement SpikingTM on a memristive crossbar array based hardware system. But the previous studies were based on simulations on different abstraction levels. The final step is therefore the execution of the Mem SpikingTM on real hardware, in this case on crossbar arrays fabricated on the MAD200 neuromorphic computing demonstrator chip.

In this section two tasks with increasing complexity are demonstrated. Firstly, a simple sequence learning task is performed to test if the general functionality of the MemSpikingTM algorithm. Secondly, the context-sensitive learning capability is demonstrated by training two high order sequences.

Training

To confirm functionality of MemSpikingTM for sequence learning tasks, the algorithm is executed on a fabricated memristive crossbar array. Details of the technology and the design of the integrated crossbar array can be found in Chapter 4. For the experiments discussed here, the vertical 1T1R array is employed. All crossbar operations are executed on this array as described, while the functionality of the peripheral circuitry is simulated in a MATLAB script. This implementation follows the description above in Section 6.2.4.

As a first step, the training of the simple sequence {A, B, C} is explained. After that the training of two high-order sequences is investigated. The electrical parameters for the training on the demonstrator chip can be found in Table 6.15.

V_{Pot}	$-2V$	Potentiation voltage during Update Phase
$V_{Gate,Pot}$	$0.9V$	Gate voltage during Update Phase
t_{Pot}	$15\mu s$	Length of potentiation pulse
V_{Read}	$0.5V$	Read voltage during Read Phase
$V_{Gate,Read}$	$5V$	Gate voltage during Read Phase
t_{Read}	$1ms$	Length of read pulse
$n_{subpopulation}$	4	number of subpopulations
$n_{neurons,subpopulation}$	2	neurons per subpopulation
G_{init}	$10\mu S$	target initialization conductance
$V_{prediction}$	$0.4V$	prediction voltage

Figure 6.15.: **Electrical parameters for MemSpikingTM on the demonstrator chip**

Training of ABC on a 6-by-6 device crossbar array

In the beginning of every time step, the respective external stimulus, if any, is triggered. This leads to a spiking of all neurons in this very subpopulation. In Figure 6.16, the external stimulus is indicated with a red triangle. In the first epoch, each stimulus leads to a spiking of the complete subpopulation it is connected to. Spiking neurons are marked with black circles. These neurons send a READ pulse to their connected source line. To this extent, the respective source lines on the MAD200 chip are connected to the input voltage signal $V_{in,1}$. Then, the READ pulse is generated externally by the LabJack controller and forwarded via the chip's internal routing to all addressed source lines at the same time. The stimulus and spiking of the training sequence {A, B, C} can be seen in the first epoch. The background colors indicate the input current each neuron receives by the READ pulse in the very time step via the array. A value of $40\mu A$ (represented by a white patch) is defined as the prediction current threshold I_{pred}. During the time steps 0 to 2, the input current is always between $10\mu A$ and $30\mu A$, well below I_{pred}. In time steps 3 and 4, the input current is zero because no neurons spike in these time steps. The MemSpikingTM learning rule fosters connections between neurons that spike directly after one another, in a Hebbian fashion. Therefore, the connections between all neurons in A and all neurons in B as well as all neurons in B and all neurons in C are strengthened during the first epoch. In the second epoch, the subpopulation A spikes completely when it is externally stimulated. It can be seen that these spikes lead to high currents to the neurons in subpopulation B. The first neuron in B receives the greatest current and becomes predictive. Therefore, when in the next time step subpopulation B is externally stimulated and because the inhibition threshold is set to 1, only the one predicted neuron spikes. The current received by the neurons in subpopulation C by this one spike is already greater than the current it received in the previous epoch by the combined spiking of both neurons in B. Yet, it is still below the prediction threshold I_{pred}. Therefore, no neuron in C is predicted and this complete subpopu-

lation spikes upon the external stimulus of C in the next time step. However, the learning rule again leads to an increase of the synaptic strength from the one spiking neuron in B to both spiking neurons in C. Finally, in the last epoch, the spiking of the predicted neuron in B leads to a prediction in one neuron in C. With this, the sequence {A, B, C} is correctly predicted and the learning task achieved.

The function of the learning algorithm can not only be seen in the evolution of predictions. When looking carefully at the input current that is received by subpopulations which are not the respectively next character, a decrease from one epoch to the next can be observed. When comparing for example the input current received by the neurons in subpopulation A in the third time step of every epoch, when C is externally stimulated, the currents are significantly smaller in the last epoch than in the first. Besides the strengthening of connections to trigger prediction, the decay of "false" connections is important to increase the dynamics range between predicting and not predicting neurons.

Training of high-order sequences in 8-by-8 array

The final test in this chapter is the demonstration of context-sensitive learning on a fabricated memristive crossbar array. To this extend, the MemSpikingTM is executed on an 8-by-8 1T1R subarray on the MAD200 demonstrator. Each subpopulation consists of two neurons and is attributed to one character A, B, C, or D. The inhibition threshold $th_{\text{inhibition}}$ is set to 1. The function of the peripheral circuitry is again emulated in a MATLAB program. Only the electrical signals to and from the array are performed on chip. To investigate the context-sensitive learning capability, the high-order sequences {A, B, C} and {D, B, A} are shown to the network after another for four epochs.

In Figure 6.17 the training activity of the MemSpikingTM system is displayed. As in the previous paragraph, external stimuli are indicated with red triangles and spikes with black dot. It can be seen that in each epoch the sequences {A, B, C} and {D, B, A} are externally stimulated in subsequent time steps. After each subsequence of three characters, for two time steps no external stimulus is applied to make sure that all potential predictions vanish. The color coding in the background shows the dendritic current level which is received by an excitatory neuron by the spiking in the respective time step. The threshold current $I_{\text{prediction}} = 40\mu A$ corresponds to white background color. If this current is overcome, the excitatory neuron can become predictive what is indicated by a blue line. During the first three time steps, when the first sub sequence is stimulated for the first time, always both excitatory neurons in each stimulated subpopulation spike, because the memristive devices still have their initial low conductance. The resulting dendritic currents are in the range between $10\mu A$ and $30\mu A$. However, following the Hebbian learning rule of the MemSpikingTM algorithm, synapses between subsequently firing excitatory neurons are enhanced.

In the original MemSpikingTM algorithm, this potentiating update in the Update phase is facilitated by opening the transistor gates vertically connected to neurons which spike in the present time step and applying a voltage pulse with a potentiating

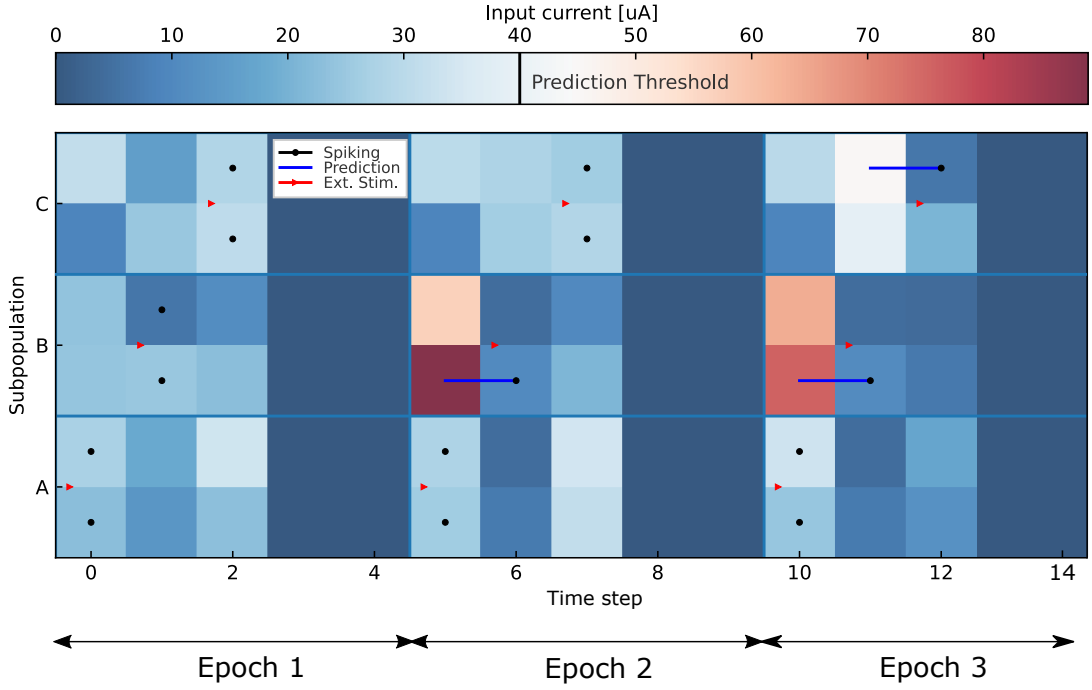

Figure 6.16.: **Training of the sequence {A, B, C} on the demonstrator chip** Initially, in the first epoch the stimulus of the sequence {A, B, C} (red triangles) lets all neurons in the respective subpopulations spike (black circles). The resulting currents to the neurons (color coding in the background) are below the prediction threshold. in the second epoch the spiking of A causes high currents in B, but due to the inhibition, only the highest leads to a prediction. In the third epoch, in both B and C a prediction is caused by the spiking of a neuron in the respective previous subpopulation.

amplitude to horizontal input lines which are connected to excitatory neurons that spiked in the previous time step. For the given 1T1R devices this would mean the application of a negative pulse voltage at the source lines, which are the horizontal input lines here. Since the body contacts of the access transistors in the 1T1R array is connect to GND, this could potentially lead to high currents from the source gate of the access transistors into the body contact if the potential barrier of the pn junction at the source-body interface (at the transistor channel) is overcome. This could lead to thermal damage to the chip. To avoid this issue, the update pulse voltage is applied as a positive voltage to the vertical bit lines of the memristive array, which are the bit lines of the MAD200 demonstrator. The word lines of the same horizontal lines are activated to activate the respective transistors. Before, the line access gates of those horizontal source lines are connected to GND. In this way, an effective negative voltage is applied over the addressed memristive devices.

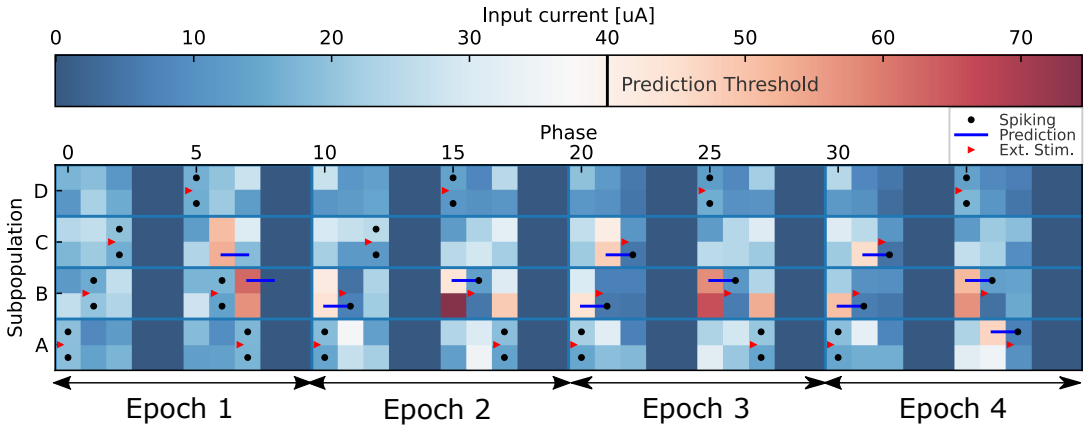

Figure 6.17.: **Training of high-order sequences {ABC} and {DBA} on an 8-by-8 memristive array** In the first epoch, only two false predictions are generated. In the second, the spiking of A and D respectively causes the prediction of one neuron each. Due to the homeostasis condition, these neurons are not the same. In the third epoch then at least for the first subsequence, the spike of this previously predicted neuron causes a prediction in the next character C until in the last epoch all sequence elements are correctly predicted.

In the first three time steps, the connections from subpopulation A to B and from B to C are enhanced. When now the next subsequence {D, B, A} is externally stimulated one can see that again all excitatory neuron of the respective subpopulations spike. The spiking of the subpopulation D only leads to low dendritic currents in all excitatory neurons. But the spiking of both neurons in B, because the synapses from B to C have been enhanced during the stimulus of the first subsequence, now produce high dendritic currents in both neurons in C as can be seen by the red background color of these neurons in time step 6. Both currents are above the prediction threshold $I_t extprediction$, but since the inhibition threshold is set to 1, only the highest of both becomes predictive. The other excitatory neuron is inhibited. However, in the next time step 7, the subpopulation A is externally stimulated. Therefore, none of the excitatory neurons in C spikes. The prediction caused by the spiking of B was a false positive prediction. Similarly, the dendritic currents due to the spikes from the subpopulation A in time step 7 lead to a prediction in subpopulation B. This is due to high conductive synapses leading from A to B which were enhanced in time step 1 when B spiked after A. Here in time step 7, the prediction in B is also a false positive, because no external stimulus is applied in the next time step 8.

The second epoch again starts with the external stimulus of the subsequence {A, B, C}. Now, the spiking of the excitatory neurons in A causes a high current in both excitatory neurons in B. Due to the inhibition, only the highest of them becomes predictive. In this case, the prediction is correct, because in the next time step

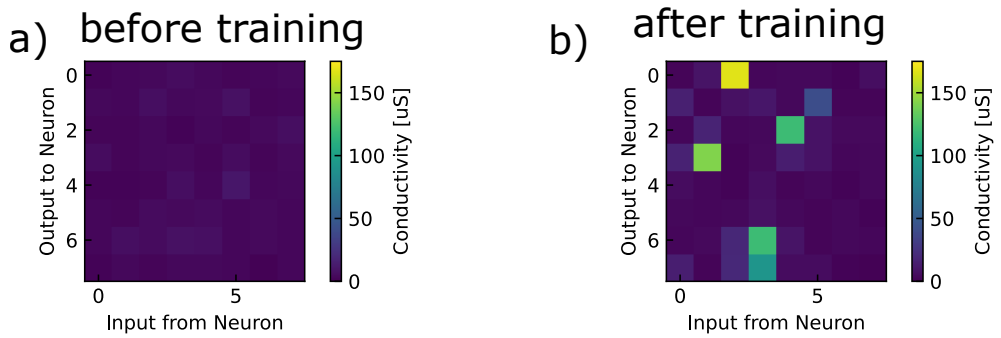

Figure 6.18.: **8-by-8 conductance matrix before and after high order sequence training** a) The matrix is initialized with low conductances between 1μS and 7μS. b) After the training a small number of devices shows an increased conductance between 100μS and 150μS.

number 11, B is externally stimulated. only the predicted neuron spikes because the other i inhibited. The spike of this single excitatory neuron however causes no high dendritic current in any excitatory neuron. Therefore, when C is externally stimulated, all excitatory neurons in this subpopulation emit a spike. When the second subsequence {D, B, A} is stimulated, both excitatory neurons in D spike which causes high dendritic currents in the neurons in subpopulation B. From the color coding it is evident that the dendritic input current to one neuron is significantly higher than to the other. However, the neuron receiving the lower input gets predictive in this case because the other was already predictive in the current epoch and is therefore hindered from becoming predictive again by the homeostasis condition. The prediction is again correct as B is stimulated i the 16th time step. But the spiking of the predicted neuron again causes no significant dendritic currents in other neurons. Therefore, when A is stimulated in the 17th time step, all neurons of A spike.

In the third epoch, the prediction of one neuron in B persists after the spiking of all neurons in A in the 20th time step. Furthermore, it can be seen that the dendritic current to the neuron in B which is not predicted is now below the prediction threshold, result of the small depression due to the pulse in the previous time step. The predicted neuron in B emits a spike in the 21st time step when B is stimulated. This time, this spike generate sufficient dendritic current in one neuron in subpopulation C which causes this neuron to become predictive and solely spike after the stimulus of C in time step 22. The situation during the stimulus of the second subsequence {D, B, A} is the same as in the second epoch with the sole spiking of one neuron in B not being sufficient to predict any other neuron. This however changes in the fourth epoch, where the spike by this neuron generates a high dendritic current in one neuron in subpopulation A. In this fourth epoch, both sequences {A, B, C} and {D, B, A} are correctly predicted and no false predictions are made. The learning is therefore successful at this stage.

The conductance matrix is initialized by sweeping each device three times, finally

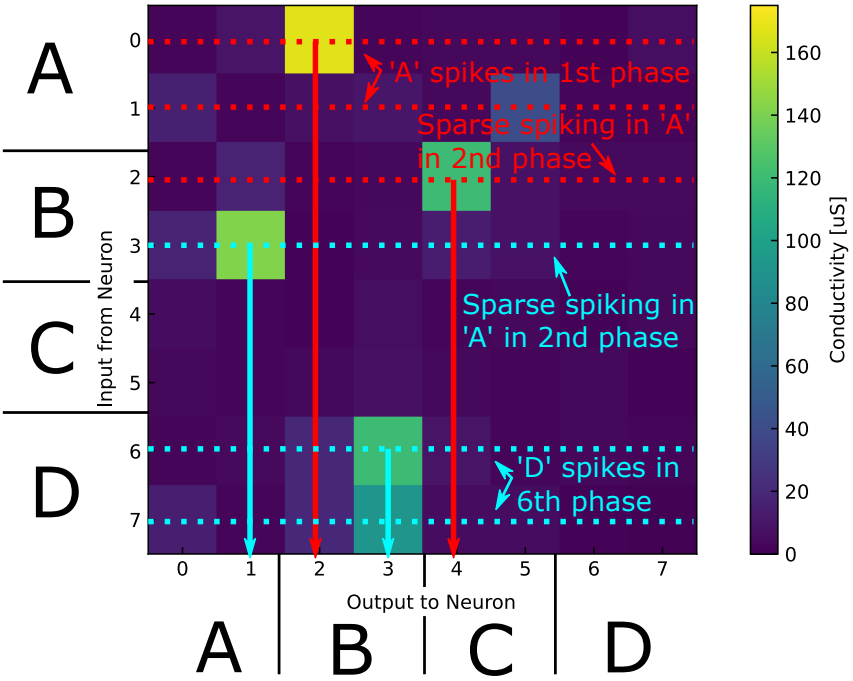

Figure 6.19.: **Conductive pathways for context sensitive learning** During the stimulus of the sequence {A, B, C}, all synapses in lines 0 and 1 (upper two red dashed lines) receive a READ pulse from which the yellow colored synapse generates a large current to neuron 2 in B. The spiking of this neuron leads to a READ pulse to all synapses in line 2 (lower red dashed line) which causes a large current to neuron 4 in C. If D spikes, a predicting current is caused in neuron 3 in B which itself leads to a prediction in neuron 1 in A.

resetting it to a low conductive state. The initial conductances are depicted in Figure 6.18a). They all have values around $5\mu S$. When comparing the conductance matrices of the memristive crossbar array before and after the training (depicted in Figure 6.18b)), five distinctly increased conductances can be seen with conductance values between $70\mu S$ and $165\mu S$. With this ON-OFF ratio of 14 to 33 on the fabricated hardware a larger dynamics range of conductances is achieved than in simulation for the larger array and task complexity. Other devices only exhibit minor changes in conductance.

The devices with increased conductance can be interpreted as a conductive pathway that represents the two trained sequences. In Figure 6.19 these are marked in red for the sequence {A, B, C} and turquoise for the sequence {D, B, A}. All synaptic devices in subpopulation A which are indicated with red dotted lines receive this READ pulse. The high conductance (in yellow) produces a large current upon this spike and thereby the excitatory neuron 2 in subpopulation B is predicted. On the external stimulus of

B, now only this neuron emits a spike. Of all synaptic devices experiencing this pulse, only the device leading to excitatory neuron 4 in subpopulation C produces a large current, predicting this neuron. When in contrast subpopulation D spikes first, two high conductive synapses leading to excitatory neuron 3 in subpopulation produce a high current and neuron 3 is predicted. A spiking of excitatory neuron itself after B is externally stimulated reaches only one high conductive synapse which predicts neuron 1 in subpopulation A.

It is remarkable that a complete spiking of D predicts the next neuron via two synapses, one per spiking neuron in D, while A predicts the next neuron in B via a single very high conductive synapse. The reason is that A in this set of sequences is not only initial character, but also final character of a sequence. Since in the sequence {D, B, A}, neuron 1 in A is predicted, this neuron has a context and can therefore not participate in the prediction of another sequence anymore.

6.5. Summary of memristive spiking neural networks

In this chapter the complete journey from a biologically plausible algorithm for sequence learning to the demonstration of a derived neuromorphic system on a CMOS-ReRAM co-integrated chip is shown.

The Spiking Temporal Memory (SpikingTM) is a biologically plausible version of the Temporal Memory of Jeff Hawkin's Hierarchical Temporal Memory. After showing that memristive switching dynamics can be used to replace biological synaptic switching dynamics in SpikingTM, a system model for an adapted neuromorphic version of this algorithm is adapted that can efficiently operate with the functional principles of ReRAM devices. The ability of this system to learn complex sequences is demonstrated with a lower energy consumption compared to conventional memory technologies. Finally, the adapted system is validated on a CMOS-ReRAM co-integrated chip.

7. Conclusion

The implementation of memristive devices as weights in the hardware realization of ANNs for machine learning and neuromorphic systems holds great promise to yield energy-efficient systems with a small silicon footprint. This however must not come at the cost of significant performance losses or an extensive computational and circuitry overhead. Therefore, in this thesis it discussed how memristive devices can successfully be integrated in both classical ANNs in machine learning as well as neuromorphic approaches. For **ANNs for machine learning** the limited conductance range and initialization accuracy and furthermore the non-linear and asymmetric update are targeted as large issues for the training accuracy. Solutions are proposed that on the one hand allow a Xavier and Kaiming-like initialization with arbitrarily initialized memristive weight layers by adapting the activation function. The memristive update disturbance is targeted by a simple mathematical approach that compensates the asymmetry. Training simulations show that this method can effectively restore floating point accuracy.

The development of **neuromorphic memristive systems** in this thesis addresses a fundamental learning principle of the human brain: sequence learning. Therefore, a biologically plausible version of the Hierarchical Temporal Memory (HTM) is adapted for execution on a dedicated memristive neuromorphic hardware system after it is shown that memristive switching dynamics are well-suited to replace the biological synaptic dynamics. The adapted algorithm can successfully learn context dependencies in temporal sequences up to solving synthetic sequence learning benchmarks. The combination of an algorithm that develops sparse spiking activity in combination with the low-power operation of memristive devices is shown to improve the energy consumption for this training process.

To support the development of memristive ANNs for machine learning and neuromorphic applications, a **demonstrator chip with memristive 1T1R crossbar arrays** in a commercial 130nm CMOS process node is designed and taped-out. Featuring three different crossbar array organization principles and signal routing logic the chip can be used for various applications ranging from neural networks to Content-addressable Memory (CAM) demonstrations. In the context of this thesis the chip is used to demonstrate the MemSpikingTM algorithm.

With these three key results combined, a next step toward the implementation of neural network algorithms on dedicated memristive hardware systems is done. It is shown that the shortcomings of memristive devices like there imperfect update dynamics can be mitigated and their non-volatile data storage nature holds great promise for continuous neuromorphic learning systems.

Future directions

While the compensation method developed in this thesis that copes with the asymmetry of memristive update dynamics and thereby restores floating-point accuracy is mathematically simple, it still must be discussed how this method can be applied to a memristive hardware system with acceptable computational and circuitry overhead. Furthermore, its application to other and more complex network types like convolutional neural networks and temporal networks like recurrent neural networks can be investigated.

As for the neuromorphic approach of sequence learning with memristive devices, it is necessary to demonstrate the functionality of the MemSpikingTM algorithm on real-world data instead of synthetic data. The impact of noise and methods how to further simplify the algorithm are especially interesting here. Another important question is how to learn without a homeostasis algorithm.

Bibliography

[1] C. Mead, "Neuromorphic engineering: In memory of Misha Mahowald," *Neural Computation*, pp. 1–41, 2022.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[3] J. Alsmeier, M. Higashitani, S. S. Paak, and S. Sivaram, "Past and future of 3D flash," in *2020 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2020, pp. 6–1.

[4] P. McCorduck, M. Minsky, O. G. Selfridge, and H. A. Simon, "History of artificial intelligence.," in *IJCAI*, 1977, pp. 951–954.

[5] A. Turing, "Turing machine," *Proc London Math Soc*, vol. 242, pp. 230–265, 1936.

[6] A. Turing, "Turing," *Computing machinery and intelligence. Mind*, vol. 59, no. 236, pp. 433–60, 1950.

[7] J. R. Sublette, "The Dartmouth conference: Its reports and results," *College English*, vol. 35, no. 3, pp. 348–357, 1973.

[8] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[9] M. Marvin and A. P. Seymour, "Perceptrons," *Cambridge, MA: MIT Press*, vol. 6, pp. 318–362, 1969.

[10] H. Moravec, "Today's computers, intelligent machines and our future," *Analog*, vol. 99, no. 2, 1979.

[11] R. Anyoha. "The history of artificial intelligence." (2017), [Online]. Available: `https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/` (visited on 11/30/2022).

[12] B. Pandolfini, *Kasparov and Deep Blue: The historic chess match between man and machine*. Simon and Schuster, 1997.

[13] X. Huang, J. Baker, and R. Reddy, "A historical perspective of speech recognition," *Communications of the ACM*, vol. 57, no. 1, pp. 94–103, 2014.

[14] M. Blut, C. Wang, N. V. Wünderlich, and C. Brock, "Understanding anthropomorphism in service provision: A meta-analysis of physical robots, chatbots, and other AI," *Journal of the Academy of Marketing Science*, vol. 49, no. 4, pp. 632–658, 2021.

[15] G. Lugano, "Virtual assistants and self-driving cars," in *2017 15th International Conference on ITS Telecommunications (ITST)*, IEEE, 2017, pp. 1–5.

[16] A. Maedche, C. Legner, A. Benlian, *et al.*, "AI-based digital assistants," *Business & Information Systems Engineering*, vol. 61, no. 4, pp. 535–544, 2019.

[17] P. Hamet and J. Tremblay, "Artificial intelligence in medicine," *Metabolism*, vol. 69, S36–S40, 2017.

[18] B. W. Wirtz, J. C. Weyerer, and C. Geyer, "Artificial intelligence and the public sector—applications and challenges," *International Journal of Public Administration*, vol. 42, no. 7, pp. 596–615, 2019.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[20] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[21] A. C. Tsoi and A. Back, "Discrete time recurrent neural network architectures: A unifying review," *Neurocomputing*, vol. 15, no. 3-4, pp. 183–223, 1997.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[24] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[25] S. Hayman, "The mcculloch-pitts model," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, IEEE, vol. 6, 1999, pp. 4438–4439.

[26] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[28] C. Mead, "How we created neuromorphic engineering," *Nature Electronics*, vol. 3, no. 7, pp. 434–435, 2020.

[29] C. Mead and L. Conway, *Introduction to VLSI systems*, 1980.

[30] C. A. Mead and M. A. Mahowald, "A silicon model of early visual processing," *Neural networks*, vol. 1, no. 1, pp. 91–97, 1988.

[31] M. A. Mahowald, "Silicon retina with adaptive photoreceptors," in *Visual information processing: from neurons to chips*, SPIE, vol. 1473, 1991, pp. 52–58.

[32] C. A. Mead, X. Arreguit, and J. Lazzaro, "Analog VLSI model of binaural hearing," *IEEE transactions on Neural Networks*, vol. 2, no. 2, pp. 230–236, 1991.

[33] E. Chicca, M. Schmuker, and M. P. Nawrot, *Neuromorphic sensors, olfaction.* 2014.

[34] C. Bartolozzi, "Neuromorphic circuits impart a sense of touch," *Science*, vol. 360, no. 6392, pp. 966–967, 2018.

[35] M. Maris and M. Mahowald, "Neuromorphic sensory-motor mobile robot controller with pre-attention mechanism," in *Neuromorphic Systems: Engineering Silicon from Neurobiology*, World Scientific, 1998, pp. 126–137.

[36] R. Douglas, M. Mahowald, and C. Mead, "Neuromorphic analogue VLSI," *Annual review of neuroscience*, vol. 18, no. 1, pp. 255–281, 1995.

[37] S. Tharin and A. Golby, "Functional brain mapping and its applications to neurosurgery," *Operative Neurosurgery*, vol. 60, no. suppl_4, ONS–185, 2007.

[38] H.-J. Park and K. Friston, "Structural and functional brain networks: From connections to cognition," *Science*, vol. 342, no. 6158, p. 1 238 411, 2013.

[39] S. Furber, "Large-scale neuromorphic computing systems," *Journal of neural engineering*, vol. 13, no. 5, p. 051 001, 2016.

[40] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[41] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[42] M. Davies, A. Wild, G. Orchard, *et al.*, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[43] S. Scholze, H. Eisenreich, S. Höppner, *et al.*, "A 32 GBit/s communication SoC for a waferscale neuromorphic system," *Integration*, vol. 45, no. 1, pp. 61–75, 2012.

[44] J. Hawkins and D. George, "Hierarchical temporal memory: Concepts, theory and terminology," *Numenta Inc*, 2006.

[45] S. Ahmad and J. Hawkins, "Properties of sparse distributed representations and their application to hierarchical temporal memory," *arXiv preprint arXiv:1503.07469*, 2015.

[46] J. Hawkins, *A thousand brains: A new theory of intelligence*. Hachette UK, 2021.

[47] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay, *et al.*, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

[48] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[49] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[50] W. Schultz, "Behavioral theories and the neurophysiology of reward," *Annu. Rev. Psychol.*, vol. 57, pp. 87–115, 2006.

[51] T. Isomura and T. Toyoizumi, "Error-gated hebbian rule: A local learning rule for principal and independent component analysis," *Scientific reports*, vol. 8, no. 1, p. 1835, 2018.

[52] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in neuroscience*, vol. 11, p. 324, 2017.

[53] R. Waser, *Nanoelectronics and information technology: advanced electronic materials and novel devices*. John Wiley & Sons, 2012.

[54] H.-M. Huang, Z. Wang, T. Wang, Y. Xiao, and X. Guo, "Artificial neural networks based on memristive devices: From device to system," *Advanced Intelligent Systems*, vol. 2, no. 12, p. 2 000 149, 2020.

[55] M. Le Gallo and A. Sebastian, "An overview of phase-change memory device physics," *Journal of Physics D: Applied Physics*, vol. 53, no. 21, p. 213 002, 2020.

[56] G. W. Burr, M. J. Brightsky, A. Sebastian, *et al.*, "Recent progress in phase-change memory technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 146–162, 2016.

[57] H.-S. P. Wong, S. Raoux, S. Kim, *et al.*, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[58] T. Kim and S. Lee, "Evolution of phase-change memory for the storage-class memory and beyond," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1394–1406, 2020.

[59] G. Servalli, "A 45nm generation phase change memory technology," in *2009 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2009, pp. 1–4.

[60] S. La Barbera, D. R. Ly, G. Navarro, *et al.*, "Narrow heater bottom electrode-based phase change memory as a bidirectional artificial synapse," *Advanced Electronic Materials*, vol. 4, no. 9, p. 1 800 223, 2018.

[61] S. Bhatti, R. Sbiaa, A. Hirohata, H. Ohno, S. Fukami, and S. Piramanayagam, "Spintronics based random access memory: A review," *Materials Today*, vol. 20, no. 9, pp. 530–548, 2017.

[62] A. Khvalkovskiy, D. Apalkov, S. Watts, *et al.*, "Basic principles of STT-MRAM cell operation in memory arrays," *Journal of Physics D: Applied Physics*, vol. 46, no. 7, p. 074 001, 2013.

[63] T. Na, S. H. Kang, and S.-O. Jung, "STT-MRAM sensing: A review," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 1, pp. 12–18, 2020.

[64] K. Garello, F. Yasin, and G. S. Kar, "Spin-orbit torque MRAM for ultra-fast embedded memories: From fundamentals to large scale technology integration," in *2019 IEEE 11th International Memory Workshop (IMW)*, IEEE, 2019, pp. 1–4.

[65] Q. Shao, P. Li, L. Liu, *et al.*, "Roadmap of spin–orbit torques," *IEEE Transactions on Magnetics*, vol. 57, no. 7, pp. 1–39, 2021.

[66] S. Ikegawa, F. B. Mancoff, and S. Aggarwal, "Commercialization of MRAM–historical and future perspective," in *2021 IEEE International Interconnect Technology Conference (IITC)*, IEEE, 2021, pp. 1–3.

[67] L. Jiang, B. Zhao, Y. Zhang, and J. Yang, "Constructing large and fast multi-level cell STT-MRAM based cache for embedded processors," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 907–912.

[68] A. Ranjan, S. Venkataramani, Z. Pajouhi, R. Venkatesan, K. Roy, and A. Raghunathan, "STAxCache: An approximate, energy efficient STT-MRAM cache," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, IEEE, 2017, pp. 356–361.

[69] Y.-J. Song, J. H. Lee, H. Shin, *et al.*, "Highly functional and reliable 8mb STT-MRAM embedded in 28nm logic," in *2016 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2016, pp. 27–2.

[70] F. Oboril, R. Bishnoi, M. Ebrahimi, and M. B. Tahoori, "Evaluation of hybrid memory technologies using SOT-MRAM for on-chip cache hierarchy," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 3, pp. 367–380, 2015.

[71] L. Chang, Z. Wang, A. O. Glova, *et al.*, "PRESCOTT: Preset-based cross-point architecture for spin-orbit-torque magnetic random access memory," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2017, pp. 245–252.

[72] R. Dittmann, S. Menzel, and R. Waser, "Nanoionic memristive phenomena in metal oxides: The valence change mechanism," *Advances in Physics*, vol. 70, no. 2, pp. 155–349, 2021. DOI: 10.1080/00018732.2022.2084006. eprint: https://doi.org/10.1080/00018732.2022.2084006. [Online]. Available: https://doi.org/10.1080/00018732.2022.2084006.

[73] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced materials*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[74] J. Park, K. Biju, S. Jung, *et al.*, "Multibit operation of TiOx-based ReRAM by schottky barrier height engineering," *IEEE Electron Device Letters*, vol. 32, no. 4, pp. 476–478, 2011.

[75] A. Marchewka, R. Waser, and S. Menzel, "Physical simulation of dynamic resistive switching in metal oxides using a schottky contact barrier model," in *2015 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, IEEE, 2015, pp. 297–300.

[76] S. Menzel, C. Funck, C. La Torre, and A. Marchewka, "Multiscale simulation of ReRAMs based on the valence change mechanism," in *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*, IEEE, 2018, pp. 1–3.

[77] K. Fleck, C. La Torre, N. Aslam, S. Hoffmann-Eifert, U. Böttger, and S. Menzel, "Uniting gradual and abrupt set processes in resistive switching oxides," *Physical review applied*, vol. 6, no. 6, p. 064 015, 2016.

[78] D. Cooper, C. Baeumer, N. Bernier, *et al.*, "Anomalous resistance hysteresis in oxide reram: Oxygen evolution and reincorporation revealed by in situ tem," *Advanced materials*, vol. 29, no. 23, p. 1 700 212, 2017.

[79] S. Menzel, M. Waters, A. Marchewka, U. Böttger, R. Dittmann, and R. Waser, "Origin of the ultra-nonlinear switching kinetics in oxide-based resistive switches," *Advanced Functional Materials*, vol. 21, no. 23, pp. 4487–4492, 2011.

[80] J. van den Hurk, S. Menzel, R. Waser, and I. Valov, "Processes and limitations during filament formation and dissolution in GeSx-based ReRAM memory cells," *The Journal of Physical Chemistry C*, vol. 119, no. 32, pp. 18 678–18 685, 2015.

[81] Y. Chen, "ReRAM: History, status, and future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[82] J. Yang, M. Rao, H. Tang, *et al.*, "Thousands of conductance levels in memristors monolithically integrated on CMOS," 2022.

[83] C. Baeumer, R. Valenta, C. Schmitz, *et al.*, "Subfilamentary networks cause cycle-to-cycle variability in memristive devices," *ACS nano*, vol. 11, no. 7, pp. 6921–6929, 2017.

[84] S. Bagdzevicius, K. Maas, M. Boudard, and M. Burriel, "Interface-type resistive switching in perovskite materials," *Journal of Electroceramics*, vol. 39, pp. 157–184, 2017.

[85] A. Asamitsu, Y. Tomioka, H. Kuwahara, and Y. Tokura, "Current switching of resistive states in magnetoresistive manganites," *Nature*, vol. 388, no. 6637, pp. 50–52, 1997.

[86] C. Jooss, L. Wu, T. Beetz, *et al.*, "Polaron melting and ordering as key mechanisms for colossal resistance effects in manganites," *Proceedings of the National Academy of Sciences*, vol. 104, no. 34, pp. 13 597–13 602, 2007.

[87] A. Gutsche, S. Hambsch, N. C. Branca, R. Dittmann, S. Scholz, and J. Knoch, "Disentangling ionic and electronic contributions to the switching dynamics of memristive Pr0.7Ca0.3MnO3/Al devices by employing a two-resistor model," *Physical review materials*, vol. 6, no. 9, p. 095 002, 2022.

[88] R. Muenstermann, T. Menke, R. Dittmann, and R. Waser, "Coexistence of filamentary and homogeneous resistive switching in Fe-doped SrTiO3 thin-film memristive devices," *Advanced materials*, vol. 22, no. 43, pp. 4819–4822, 2010.

[89] R. A. De Souza, J. Fleig, R. Merkle, and J. Maier, "Srtio3: A model electroceramic," *International Journal of Materials Research*, vol. 94, no. 3, pp. 218–225, 2022.

[90] K. Szot, R. Dittmann, W. Speier, and R. Waser, "Nanoscale resistive switching in srtio3 thin films," *physica status solidi (RRL)–Rapid Research Letters*, vol. 1, no. 2, R86–R88, 2007.

[91] A. Sarantopoulos, R. Waser, and R. Dittmann, "Eightwise switching mechanism in memristive srtio3 devices and its implications on the device performance," *physica status solidi (a)*,

[92] S. Siegel, C. Baeumer, A. Gutsche, *et al.*, "Trade-off between data retention and switching speed in resistive switching ReRAM devices," *Advanced electronic materials*, vol. 7, no. 1, p. 2 000 815, 2021.

[93] C. Baeumer, C. Schmitz, A. H. Ramadan, *et al.*, "Spectromicroscopic insights for rational design of redox-based memristive devices," *Nature communications*, vol. 6, no. 1, p. 8610, 2015.

[94] J. L. Rieck, F. V. Hensling, and R. Dittmann, "Trade-off between variability and retention of memristive epitaxial srtio3 devices," *APL materials*, vol. 9, no. 2, 2021.

[95] C. Y. Mei, W. C. Shen, C. H. Wu, *et al.*, "28-nm 2t high-K metal gate embedded rram with fully compatible cmos logic processes," *IEEE electron device letters*, vol. 34, no. 10, pp. 1253–1255, 2013.

[96] D. Garbin, E. Vianello, O. Bichler, *et al.*, "HfO2-based OxRAM devices as synapses for convolutional neural networks," *IEEE Transactions on Electron Devices*, vol. 62, no. 8, pp. 2494–2501, 2015.

[97] F. Cüppers, S. Menzel, C. Bengel, *et al.*, "Exploiting the switching dynamics of HfO2-based ReRAM devices for reliable analog memristive behavior," *APL materials*, vol. 7, no. 9, p. 091 105, 2019.

[98] W. Feng, H. Shima, K. Ohmori, and H. Akinaga, "Investigation of switching mechanism in hfox-reram under low power and conventional operation modes," *Scientific reports*, vol. 6, no. 1, p. 39 510, 2016.

[99] A. Sawa, T. Fujii, M. Kawasaki, and Y. Tokura, "Hysteretic current–voltage characteristics and resistance switching at a rectifying ti/ pr0. 7ca0. 3mno3 interface," *Applied Physics Letters*, vol. 85, no. 18, pp. 4073–4075, 2004.

[100] D.-J. Seong, M. Hassan, H. Choi, *et al.*, "Resistive-switching characteristics of $Pr_{0.7}Ca_{0.3}MnO_3$ for nonvolatile memory applications," *IEEE electron device letters*, vol. 30, no. 9, pp. 919–921, 2009.

[101] A. Herpers, *Electrical characterization of manganite and titanate heterostructures*. Forschungszentrum Jülich, 2014, vol. 32.

[102] A. Gutsche, S. Siegel, J. Zhang, S. Hambsch, and R. Dittmann, "Exploring area-dependent Pr0.7Ca0.3MnO3-based memristive devices as synapses in spiking and artificial neural networks," *Frontiers in neuroscience*, p. 723, 2021.

[103] Q. Xia and J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature materials*, vol. 18, no. 4, pp. 309–323, 2019.

[104] W. Qian, P.-Y. Chen, R. Karam, L. Gao, S. Bhunia, and S. Yu, "Energy-efficient adaptive computing with multifunctional memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 2, pp. 191–195, 2016.

[105] M. Mayahinia, A. Singh, C. Bengel, *et al.*, "A voltage-controlled, oscillation-based ADC design for computation-in-memory architectures using emerging ReRAMs," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 2, pp. 1–25, 2022.

[106] F. Cai, J. M. Correll, S. H. Lee, *et al.*, "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," *Nature Electronics*, vol. 2, no. 7, pp. 290–299, 2019.

[107] S. R. Kulkarni, S. Yin, J.-s. Seo, and B. Rajendran, "An on-chip learning accelerator for spiking neural networks using STT-RAM crossbar arrays," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020, pp. 1019–1024.

[108] Z. Li, Z. Wang, L. Xu, *et al.*, "RRAM-DNN: An RRAM and model-compression empowered all-weights-on-chip DNN accelerator," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1105–1115, 2020.

[109] T. Diokh, E. Le-Roux, S. Jeannot, *et al.*, "Investigation of the impact of the oxide thickness and RESET conditions on disturb in HfO2-RRAM integrated in a 65nm CMOS technology," in *2013 IEEE International Reliability Physics Symposium (IRPS)*, IEEE, 2013, 5E–4.

[110] H. Wu, P. Yao, B. Gao, *et al.*, "Device and circuit optimization of RRAM for neuromorphic computing," in *2017 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2017, pp. 11–5.

[111] W. Wan, R. Kubendran, C. Schaefer, *et al.*, "A compute-in-memory chip based on resistive random-access memory," *Nature*, vol. 608, no. 7923, pp. 504–512, 2022.

[112] Y.-W. Chin, S.-E. Chen, M.-C. Hsieh, T.-S. Chang, C. J. Lin, and Y.-C. King, "Point twin-bit RRAM in 3D interweaved cross-point array by Cu BEOL process," in *2014 IEEE International Electron Devices Meeting*, IEEE, 2014, pp. 6–4.

[113] M. Azzaz, E. Vianello, B. Sklenard, *et al.*, "Endurance/retention trade off in HfOx and TaOx based RRAM," in *2016 IEEE 8th international memory workshop (IMW)*, IEEE, 2016, pp. 1–4.

[114] S. Yu, Z. Li, P.-Y. Chen, *et al.*, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in *2016 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2016, pp. 16–2.

[115] A. Grossi, E. Nowak, C. Zambelli, *et al.*, "Fundamental variability limits of filament-based RRAM," in *2016 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2016, pp. 4–7.

[116] A. Valentian, F. Rummens, E. Vianello, *et al.*, "Fully integrated spiking neural network with analog neurons and RRAM synapses," in *2019 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2019, pp. 14–3.

[117] D. V. Christensen, R. Dittmann, B. Linares-Barranco, *et al.*, "2022 roadmap on neuromorphic computing and engineering," *Neuromorphic Computing and Engineering*, vol. 2, no. 2, p. 022 501, 2022.

[118] E. Esmanhotto, L. Brunet, N. Castellani, *et al.*, "High-density 3D monolithically integrated multiple 1t1r multi-level-cell for neural networks," in *2020 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2020, pp. 36–5.

[119] D. A. Robayo, G. Sassine, L. Grenouillet, *et al.*, "Integration of OTS based back-end selector with HfO2 OxRAM for crossbar arrays," in *2019 IEEE 11th International Memory Workshop (IMW)*, IEEE, 2019, pp. 1–4.

[120] T. Dalgaty, F. Moro, A. De Pra, G. Indiveri, E. Vianello, and M. Payvand, "The neuromorphic mosaic: Re-configurable in-memory small-world graphs," 2021.

[121] A. Mizrahi, T. Hirtzlin, A. Fukushima, *et al.*, "Neural-like computing with populations of superparamagnetic basis functions," *Nature communications*, vol. 9, no. 1, p. 1533, 2018.

[122] P. Wijesinghe, A. Ankit, A. Sengupta, and K. Roy, "An all-memristor deep spiking neural computing system: A step toward realizing the low-power stochastic brain," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 345–358, 2018.

[123] C. Sung, H. Hwang, and I. K. Yoo, "Perspective: A review on memristive hardware for neuromorphic computation," *Journal of Applied Physics*, vol. 124, no. 15, p. 151 903, 2018.

[124] M. Payvand, M. E. Fouda, F. Kurdahi, A. M. Eltawil, and E. O. Neftci, "On-chip error-triggered learning of multi-layer memristive spiking neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 522–535, 2020.

[125] J. Doevenspeck, R. Degraeve, A. Fantini, *et al.*, "Temporal sequence learning with a history-sensitive probabilistic learning rule intrinsic to oxygen vacancy-based RRAM," in *2018 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2018, pp. 20–5.

[126] Y. Dan and M.-m. Poo, "Spike timing-dependent plasticity of neural circuits," *Neuron*, vol. 44, no. 1, pp. 23–30, 2004.

[127] Q. Duan, Z. Jing, X. Zou, *et al.*, "Spiking neurons with spatiotemporal dynamics and gain modulation for monolithically integrated memristive neural networks," *Nature communications*, vol. 11, no. 1, p. 3399, 2020.

[128] A. Bhattacharjee, Y. Kim, A. Moitra, and P. Panda, "Examining the robustness of spiking neural networks on non-ideal memristive crossbars," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.

[129] E. Zhou, L. Fang, and B. Yang, "Memristive spiking neural networks trained with unsupervised stdp," *Electronics*, vol. 7, no. 12, p. 396, 2018.

[130] P. Zhou, D.-U. Choi, J. K. Eshraghian, and S.-M. Kang, "A fully memristive spiking neural network with unsupervised learning," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2022, pp. 634–638.

[131] I. Boybat, M. Le Gallo, S. Nandakumar, *et al.*, "Neuromorphic computing with multi-memristive synapses," *Nature communications*, vol. 9, no. 1, p. 2514, 2018.

[132] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "STDP and STDP variations with memristors for spiking neuromorphic learning systems," *Frontiers in neuroscience*, vol. 7, p. 2, 2013.

[133] S. Ambrogio, S. Balatti, V. Milo, *et al.*, "Neuromorphic learning and recognition with one-transistor-one-resistor synapses and bistable metal oxide rram," *IEEE Transactions on Electron Devices*, vol. 63, no. 4, pp. 1508–1515, 2016.

[134] G. Pedretti, V. Milo, S. Ambrogio, *et al.*, "Memristive neural network for on-line learning and tracking with brain-inspired spike timing dependent plasticity," *Scientific reports*, vol. 7, no. 1, pp. 1–10, 2017.

[135] B. Taylor, A. Shrestha, Q. Qiu, and H. Li, "1S1R-based stable learning through single-spike-encoded spike-timing-dependent plasticity," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5.

[136] Z. Deng, C. Wang, H. Lin, and Y. Sun, "A memristive spiking neural network circuit with selective supervised attention algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[137] S. Siegel, Y. Bouhadjar, T. Tetzlaff, R. Waser, R. Dittmann, and D. J. Wouters, "Neuromorphic sequence learning on a memristive crossbar array," *under review*, 2022.

[138] S. Siegel, T. Ziegler, Y. Bouhadjar, *et al.*, "Demonstration of neuromorphic sequence learning on a memristive array," *under review*, 2023.

[139] M. C. Van Rossum, G. Q. Bi, and G. G. Turrigiano, "Stable hebbian learning from spike timing-dependent plasticity," *Journal of neuroscience*, vol. 20, no. 23, pp. 8812–8821, 2000.

[140] S. Brivio, D. R. Ly, E. Vianello, and S. Spiga, "Non-linear memristive synaptic dynamics for efficient unsupervised learning in spiking neural networks," *Frontiers in Neuroscience*, vol. 15, p. 580 909, 2021.

[141] Y. Bouhadjar, S. Siegel, T. Tetzlaff, M. Diesmann, R. Waser, and D. J. Wouters, "Sequence learning in a spiking neuronal network with memristive synapses," *ArXiv*, p. 2211.16592, 2022.

[142] S. Agarwal, S. J. Plimpton, D. R. Hughart, *et al.*, "Resistive memory device requirements for a neural algorithm accelerator," in *2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2016, pp. 929–938.

[143] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[144] M. Abadi, A. Agarwal, P. Barham, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[145] G. W. Burr, R. M. Shelby, A. Sebastian, *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017.

[146] S. Menzel, U. Böttger, M. Wimmer, and M. Salinga, "Physics of the switching kinetics in resistive memories," *Advanced functional materials*, vol. 25, no. 40, pp. 6306–6325, 2015.

[147] C. Bengel, A. Siemon, F. Cüppers, *et al.*, "Variability-aware modeling of filamentary oxide-based bipolar resistive switching cells using SPICE level compact models," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4618–4630, 2020.

[148] T. Gokmen and W. Haensch, "Algorithm for training neural networks on resistive device arrays," *Frontiers in neuroscience*, vol. 14, p. 103, 2020.

[149] J. Wu, F. Mo, T. Saraya, T. Hiramoto, and M. Kobayashi, "A monolithic 3-D integration of RRAM array and oxide semiconductor FET for in-memory computing in 3-D neural network," *IEEE Transactions on Electron Devices*, vol. 67, no. 12, pp. 5322–5328, 2020.

[150] J.-H. Ryu, C. Mahata, and S. Kim, "Long-term and short-term plasticity of Ta2O5/HfO2 memristor for hardware neuromorphic application," *Journal of Alloys and Compounds*, vol. 850, p. 156 675, 2021.

[151] C. Frenkel, M. Lefebvre, and D. Bol, "Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks," *Frontiers in neuroscience*, vol. 15, p. 629 892, 2021.

[152] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[153] M. Jang, S. Seo, and P. Kang, "Recurrent neural network-based semantic variational autoencoder for sequence-to-sequence learning," *Information Sciences*, vol. 490, pp. 59–73, 2019.

[154] M. Sharma, L. Vig, *et al.*, "Automatic chromosome classification using deep attention based sequence learning of chromosome bands," in *2018 International joint conference on neural networks (IJCNN)*, IEEE, 2018, pp. 1–8.

[155] P. Liu, J. Fu, Y. Dong, X. Qiu, and J. C. K. Cheung, "Learning multi-task communication with message passing for sequence learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4360–4367.

[156] B. Ballinger, J. Hsieh, A. Singh, *et al.*, "DeepHeart: Semi-supervised sequence learning for cardiovascular risk prediction," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[157] S. Karita, N. Chen, T. Hayashi, *et al.*, "A comparative study on transformer vs RNN in speech applications," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, IEEE, 2019, pp. 449–456.

[158] Y. Bouhadjar, D. J. Wouters, M. Diesmann, and T. Tetzlaff, "Sequence learning, prediction, and replay in networks of spiking neurons," *PLOS Computational Biology*, vol. 18, no. 6, e1010233, 2022.

[159] M.-O. Gewaltig and M. Diesmann, "NEST (NEural Simulation Tool)," vol. 2, no. 4, p. 1430, 2007. DOI: 10.4249/scholarpedia.1430. [Online]. Available: https://doi.org/10.4249/scholarpedia.1430.

[160] 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.

[161] A. Lazar, G. Pipa, and J. Triesch, "Spontaneous activity in a self-organizing recurrent network reflects prior learning," in *Front. Neurosci. Conference Abstract: Computational and Systems Neuroscience*, 2010.

[162] M. Payvand, F. Moro, K. Nomura, *et al.*, "Self-organization of an inhomogeneous memristive hardware for sequence learning," *Nature Communications*, vol. 13, p. 5793, 2022. DOI: https://doi.org/10.1038/s41467-022-33476-6.

[163] L. M. Grupp, A. M. Caulfield, J. Coburn, *et al.*, "Characterizing flash memory: Anomalies, observations, and applications," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 24–33.

[164] V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M. R. Stan, and S. Swanson, "Modeling power consumption of NAND flash memories using flashpower," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 7, pp. 1031–1044, 2013.