# Digital Implementation of Neuromorphic systems using Emerging Memory devices

Tifenn Hirtzlin

université PARIS-SACLAY

# Digital Implementation of Neuromorphic systems using Emerging Memory devices

**Thèse de doctorat de l'université Paris-Saclay**

École doctorale n°575: Electrical, Optical, Bio-Physics and Engineering
Spécialité de doctorat:
Electronique et Optoélectronique, Nano-et Microtechnologies
Unité de recherche : Université Paris-Saclay, CNRS,
Centre de Nanosciences et de Nanotechnologies, 91120, Palaiseau, France.
Référent : Faculté des sciences d'Orsay

**Thèse présentée et soutenue à Palaiseau, le 18/11/2020, par**

**Tifenn HIRTZLIN**

## Composition du Jury

| | |
|---|---|
| **Delphine MARRIS-MORINI**<br>*Professeure, C2N, CNRS, Université Paris-Saclay* | Présidente |
| **Ian O'CONNOR**<br>*Professeur, École Centrale de Lyon, Université de Lyon* | Rapporteur & Examinateur |
| **Benoît MIRAMOND**<br>*Professeur, LEAT, Université Côte d'Azur* | Rapporteur & Examinateur |
| **Olivier SENTIEY**<br>*Professeur, ENSSAT, Université de Rennes 1* | Examinateur |
| **François ANDRIEU**<br>*CEA-Leti, Université Grenoble-Alpes* | Examinateur |
| **Jean-Michel PORTAL**<br>*Professeur, IM2NP, CNRS, Aix-Marseille Université* | Examinateur |
| **Damien QUERLIOZ**<br>*C2N, CNRS, Université Paris-Saclay* | Directeur de thèse |
| **Marc BOCQUET**<br>*Professeur, IM2NP, CNRS, Aix-Marseille Université* | Invité |

Thèse de doctorat

NNT : 2020UPAST071

# Acknowledgements

The research results being presented here would not be what they are without the help of a large number of people whom I would like to thank for their involvement, enthusiasm, help, and advice.

First and foremost, I would like to sincerely thank Prof. O'CONNOR and Prof. MIRAMOND for their careful study of the manuscript and the time spent in writing their pre-submission report. I am also grateful to the examiners of the jury who devoted time to read this document for the defense. I have a special thought for Delphine, who advised me in my choice of orientation in the Master Nanosciences program at the University of Paris-Saclay she coordinates with Arnaud, as well as in my first research internship where I was able to approach the world of research for the first time.

I would also like to thank very warmly Jean-Michel and Marc who were almost my thesis co-supervisors. I was able to exchange new ideas, discuss, and of course, work hand in hand to both design this magnificent Bayesian circuit and to have been able to test it in Marseille despite a very unfavorable context. I have always truly enjoyed working with you and I hope to continue in the future with exciting joint projects.

Of course, I am extremely grateful to Damien for all the time devoted to my thesis, the resources given to bring all these projects to their fruition, and the advice and exchanges that were numerous during these three years. I can say that I couldn't have dreamed of a better thesis supervisor, he was always there, was able to push for the completion of the projects. Damien is not only a brilliant scientist with incredible knowledge, but he is also able to tackle new research topics very quickly to advise his students. The research group, as well as the different collaborations you have been able to set up, is extraordinary. As soon as I arrived in the group for my internship I felt in your team like in a family! For that, I thank Adrien, Damir, Alice, Chris, and Nicolas who welcomed me, made me feel at home, and also gave me the first advice for the research. Adrien taught me electronics from the bachelor's degree and he is one of the people who contributed to the special affection I have for the field. The brainstorming with Damir on the board in our office in building 220 will remain etched in my memory. Alice, whom I was able to work with to bring to fruition this magnificent project of population coding of super-paramagnetic junctions, as well as your involvement in the organization of collaborative meetings between the C2N and the CNRS-Thales. And Chris and Nicolas with whom I was also able to exchange. I hope to meet you again soon, Chris in Albuquerque if I ever pass by and

Nicolas in Palaiseau and for sure I will pass by there often.

Then I would like to thank all the members of the IntegNano team, first of all, Maxence with whom I started my thesis and who is someone I care about, both for the scientific exchanges (sometimes a bit rough because we don't always agree), and from a friendly point of view, we could share very nice moments (I will always remember the missed boat in Roscoff after a jog around the island of Batz because you wanted to drink a coke!).

Next, I would like to thank Axel for his enthusiasm and his ability to tackle a variety of subjects. In the beginning, you were doing micro-magnetic simulations, then you had to do clean room and manufacturing, but in the end, given all the problems in the lab, you were able to tackle different subjects. I almost feel like you were there at the beginning of my thesis, your presence is so calming.

Thanks also to Bogdan, who was able to get the group to adopt the Deep Learning tools to the whole team, your arrival in the group did a lot of good to modernize the tools. But also for your friendliness, your fruitful exchanges, your critical and constructive remarks as well as the sharing of your innovative ideas. I enjoyed working with you.

I would also like to thank the new generation of doctoral students on the team. Kamel is an extremely calm and nice person, able to take the deadlines without too much stress while the whole team seems to be in a panic. I thank you very much for the work you have done for the design of the Bayesian chip, it is largely thanks to you that this chip could be designed. I also thank you for the soccer games that we've been able to play recently, it was a great time for me and it allows us to create moments with the members of the group. Atreya, I also thank you for the exchanges we had, the very interesting discussions, and I trust that you will take over the learning of binarized neural networks, it is a project I deeply believe in and I am sure that you will be able to carry it through to the end. Xing, you are brilliant, I wanted to thank you, I believe very much in the project you are doing, I think you don't realize the impact of your work but these are extremely important results that will have a huge impact in the field of numerical simulations in the whole field of physics! I also hope we will have the opportunity to share a moment to play table tennis or badminton soon! Clément, you are starting your thesis with a great team and I hope that you will take care to maintain this family atmosphere. With Kamel, you're taking over the chip design, and I think you make a great team for the next tap outs. I'm looking forward to seeing your next projects, which I'm sure will be very interesting.

More generally, I would like to thank all the other people on the team with whom I was able to interact. Mamour, we will surely have the opportunity to meet again since we will both work in the same institution. At both ends of France but I'm sure we will meet again! Thanks also to Liza, I am happy to have been able to discuss with you, and congratulations for all that you do both in the laboratory but also for your beautiful paintings which I am a big fan of.

Many thanks also to Julie and her entire team. Students, doctoral and post-doctoral students should be happy to have her as a supervisor. I have enjoyed interacting on the many occasions that have been joint meetings, conferences, and events such as BioComp. This stay

in Roscoff was for me the best opportunity to get to know the field of Neuromorphic, with an unequaled quality of speakers, a perfect organization which will remain a reference for me. I am also impressed by Julie for her vision of the field, her knowledge, her talents in writing papers, and above all her ability to identify an extremely innovative research topic and bring it to fruition. The collaboration of your CNRS-Thalès team with Damien's group is truly exceptional and very unique with a very high international reputation. I hope to have other opportunities to collaborate in the future.

I would also like to thank Dafiné Ravelosona, Jacques-Olivier Klein, Laurie Calvet, Nicolas Vernier, Rohit Pachat, Subhajit Roy, Guanda Wang, Guillaume Hocquet, Gyan van der Jagt, Marie Drouhin for the different discussions we had.

I would also like to thank Elisa Vianello for her help in getting the MAD 200 technology and for taking me on a postdoc in Grenoble, I'm sure we'll do great projects together.

A big thank you also to the entire Hawai.tech team for their discussions about the BM1: Pierre Bessière, Emmanuel Mazer, Jacques Drouez, Raphael Laurent, Raphael Frish, Marvin Faix, Karim Cherkaoui and Jean Simatic.

I would also like to thank all my friends who made me think of something else during my thesis, but especially the Triad: with Anthony and Amaury that support me for many years now, as well as the members of my athletic club that I leave with felt of pinching with the heart but for new adventures in Grenoble.

I would also like to thank my friends from the Master Nanosciences, Pierre, Léa, Raquel for the all-you-can-eat sushi dinners.

Finally, I would like to thank my family and especially my mother who has always been there to guide me in my orientation for my studies and who has always pushed me and my brothers to work. Also, my father who gave me a computer with internet, relatively young, in a naive way maybe but in the end, it didn't have so many drawbacks. Also for giving me the taste of the sport.

My two little brothers as well, Killian who has changed a lot and is now extremely accomplished and still has the smile and the word for laughs. My little Quentin with your big heart and a lot of sensitivity has also changed now that you have big biceps. I am really lucky to have you.

Finally, I would like to thank from the bottom of my heart my little Laura who supported me throughout my thesis and even before, and to all your family that I love, especially your father Laurent for the trail jog. Laura, you have changed me, I am happy to live with you and to start a new life in Grenoble made of nature and sport!

# Contents

**Introduction** **1**

**1 The brain as an inspiration** **5**
1.1 The fundamental limitations of Electronic hardware . . . . . 6
1.2 Advancing electronics another way: enhance communication . . . . . 10
1.3 The special case of memory . . . . . 14
1.4 Integration of Non-Volatile-Memory (NVM) for In-Memory Computing (IMC) . . 20
1.5 Bridging the gap between memory and computing . . . . . 24
1.6 Existing implementation of neuromorphic hardware . . . . . 29

**2 Low Energy Inference Neuromorphic System** **37**
2.1 Bayesian principle . . . . . 39
2.2 Quantization of Neural Networks . . . . . 43
2.2.1 Neural Networks : background . . . . . 43
2.2.2 Reducing the resolution of neural network . . . . . 46
2.2.3 Binarized Neural Network . . . . . 48
2.2.4 Quantized and Binarized Neural Network : a comparison . . . . . 50
2.2.5 Specificities of Binarized Neural Network . . . . . 51
2.3 Connection between Bayesian Reasoning and Neural Network . . . . . 52
2.4 Emerging memory devices for NN and Bayesian models . . . . . 54

**3 Hardware implementation of Bayesian Machine** **57**
3.1 Theoretical Bayes Inference . . . . . 58
3.2 Naive Bayes Assumption . . . . . 62
3.3 Computing Bayesian Inference with Stochastic Computing . . . . . 64
3.4 The Bayesian Machine : hardware description . . . . . 68
3.4.1 General overview of the Architecture . . . . . 68
3.4.2 Global Architecture . . . . . 71
3.4.3 Random Number Generator . . . . . 72
3.4.4 Weighted Binary Generator . . . . . 73
3.5 The boat localization example . . . . . 74

3.6 The final Chip Design with RRAM . . . 77
3.6.1 The Design Flow . . . 78
3.6.2 The RRAM memory array . . . 79
3.6.3 Digital Logic Blocks . . . 86
3.6.4 The Bayesian Machine full architecture . . . 87
3.6.5 Final Design and small array test structures . . . 89
3.7 Conclusion . . . 90

**4 Hardware implementation of Binarized Neural Networks 93**
4.1 Differential Memory Array for In-Memory Computing . . . 97
4.2 Design of In-Memory Binarized Neural Network Based on the Differential Memory Building Block . . . 100
4.3 Memory Operation at Reduced BER . . . 104
4.4 Do All Errors Need to Be Corrected? . . . 109
4.4.1 Impact of Errors on Binarized neural Network performances . . . 109
4.4.2 Reducing Error Impact using adapted learning . . . 111
4.5 Projection at the System Level . . . 112
4.5.1 Impact of In-Memory Computation . . . 112
4.5.2 Impact of Binarization . . . 112
4.5.3 Comparison with Analog Approaches . . . 113
4.5.4 Impact in Terms of Programming Energy and Device Aging . . . 114
4.6 Stochastic Computing for Binarized Neural Networks . . . 115
4.6.1 Background . . . 115
4.6.2 Stochastic Computing with Regular Training Procedure . . . 116
4.6.3 Adapted Training Procedure . . . 117
4.6.4 Energy and Area, comparison using stochastic computing . . . 117
4.6.5 Partially Binarized Convolutional Neural Network . . . 119
4.7 Conclusion . . . 120

**5 Stochastic components 123**
5.1 Stochastic device behaviours for MCMC Sampling . . . 124
5.2 Neural-like computing with populations of superparamagnetic basis functions . . . 127
5.3 Tuning curve of a superparamagnetic tunnel junction . . . 129
5.4 Population coding with superparamagnetic tunnel junctions . . . 130
5.5 A computing unit that can learn . . . 133
5.6 Design of the full system . . . 137
5.6.1 Methods . . . 138
5.6.2 Results . . . 139
5.7 Discussion . . . 140

**6 Analog RRAMs** **143**
6.1 Ternarized Neural Network . . . 144
6.1.1 Using the same amount of device to implement ternary synapse without memory overhead . . . 145
6.1.2 Binarized Neural Network vs Ternarized Neural Network . . . 147
6.2 Synaptic Metaplasticity in Binarized Neural Networks . . . 148
6.2.1 Catastrophic forgetting . . . 148
6.2.2 Interpreting the Hidden Weights of Binarized Neural Networks as Metaplasticity States . . . 149
6.2.3 Learning two tasks with Metaplastic Binarized Neural Networks . . . 150
6.2.4 Metaplasticity using emerging nanodevices . . . 151
6.3 Hybrid Analog-Digital Learning with Differential RRAM Synapses . . . 152
6.3.1 Principle of learning . . . 152
6.3.2 A 2T2R complementary programming strategy . . . 154
6.3.3 Reprogramming check strategy to avoid saturation . . . 156
6.3.4 Device simulation . . . 157
6.3.5 Device imperfection, impact on accuracy . . . 159
6.3.6 Benefits of Operating in a Weak RESET Regime . . . 160
6.3.7 Comparison with other approaches for RRAM-Based Learning . . . 161
6.3.8 Next requirement for a final on-chip learning implementation . . . 162

**Conclusions and future work** **167**

**List of publications** **173**

**Appendix A Link between neural network and Bayesian models** **177**

**Appendix B Practical example** **181**

**Appendix C Synthèse en Français** **187**

**Bibliography** **226**

# Introduction

"Listen to the technology; find out what it's telling you."

Carver MEAD "Steve Jobs and the Economics of Elitism" by Steve Lohr, in NYtimes, January 30, 2010.

"THE *digital revolution of the 21st century has profoundly changed society. Access to information is now extremely easy, telecommunications are extremely efficient, and today's computing capacities are such that they are an indispensable tool for almost everything, including scientific research.* "

WHILE electronics has prospered inexorably for several decades, its leading source of progress will stop in the next coming years, due to the fundamental technological limits of transistors. However, the current state of microelectronics is not entirely satisfactory. Today, we can build extremely powerful computers, very flexible in the tasks that they can solve, but they are far from optimal in terms of energy. Smartphones are more powerful than the computers of the 90s, but their energy autonomy is relatively limited: one to two days of battery life without intense use. To add new interesting features to today's electronic circuits, it is necessary to increase their computing efficiency.

In particular, performing tasks for the Internet of Things (IoT) using extremely low energy is an exciting challenge. The number of potential applications is considerable: all the wearables and connected objects in the IoT are concerned. One of the most exciting applications concerns brain-computer interface and implants. Such types of equipment have some practical and useful applications in medicine for rehabilitation with intelligent prostheses, post-hospital monitoring at home, prevention for aged people, detection, and care for epileptic seizures, strokes, heart attacks, etc.

Currently, to have such services, data transfer through the Internet network is required, but it raises many issues:

- The energy consumption is very high.
- The reduced mobility when there is no signal.
- Privacy issues (Data on users may be collected).
- Security concerns (data manipulation, denial of service attack...).

To overcome all these challenges, a solution is to work at the edge, i.e. with autonomous and intelligent equipment. Therefore, electronics has to be able to meet all the constraints of embedded systems in terms of weight, size, integration of sensors, on-chip memory, and being able to process on-chip data with very low energy. Unfortunately, so far, running artificial intelligence algorithms required a high computing power which resulted in large energy consumption. A very important part of this energy consumption using classical computer architecture is due to the data exchange between the memory and the computation, conceptually, and physically separated. We might believe that continuous progress in microelectronics will enable the design of low energy circuits able to run such algorithms, however, we are coming to the end of Moore's law, which has governed the scheme of scaling Complementary Metal-Oxide-Semiconductor (CMOS) transistor size year after year.

Nevertheless, microelectronics is currently offering a major breakthrough: in recent years, memory technologies have undergone incredible progress, opening the way for multiple research venues in embedded systems. Additionally, a major feature for future years will be the ability to integrate different technologies on the same chip. For this reason, in this thesis, we explore the use of new emerging memory devices that can be embedded in the core of the CMOS, such as Resistive Random Access Memory (RRAM), Spin Torque Magnetic Tunnel Junction (ST-

MTJ) memory, based on naturally intelligent in-memory-computing architecture.

In this thesis, inspired by neuroscience research and taking into account the latest advances in artificial intelligence, we have designed a hybrid RRAM/CMOS circuit implementing algorithms for In-Memory-Computing. Three brain-inspired algorithms are carefully examined: Bayesian reasoning, binarized neural networks, and an approach that further exploits the intrinsic behavior of components, population coding of neurons.

The first chapter of the thesis is introductory. We discuss the many opportunities available from a technological perspective for embedded system: first, we present the main limitations of today's electronics and then we investigate what are the new emerging memory technologies and what they enable. We also explore potential next developments of In-Memory-Computing using these memory devices, including those of bio-inspired algorithms and the recent literature on neuromorphic electronics.

The second chapter defines the differences and links between Bayesian models and neural networks, as well as some ideas about the importance of a hardware implementation of these models using innovative memory components.

The third chapter presents both a detailed presentation of a Bayesian inference algorithm, and more importantly, a complete description of the implementation of an In-Memory-Computing hybrid CMOS/RRAM chip (from the algorithm to the tape out) designed during this thesis.

Similarly, Chapter 4 focuses on the hardware implementation of an inference algorithm, this time based on neural networks. An in-depth study of the impact of the properties of memory devices is presented, in particular, the effect of errors on system performance.

To go even further in the use of memory devices, Chapter 5 presents how the stochastic character of magnetic tunneling junctions can be used to reproduce the population coding of neurons in the brain.

Finally, the chapter 6 shows how to go further in the use of the complex behaviour of memory devices. We mention our latest advances in the use of these memory devices, where we aim at developing new algorithms related to devices behaviour and/or adapt classical machine learning algorithm existing in the literature to hardware constraints.

# Chapter 1

# The brain as an inspiration for the future of electronics.

> It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in 5 years.
>
> John VON NEUMANN in 1949

"FROM *the first computer to the present day, the evolution of the performance of electronic circuits has followed a major rule: improving the performance of transistors. Even if this component is extraordinary, because when its dimensions are reduced all its characteristics are improved, this rule is at its limits. Today, many nanotechnologies with exciting characteristics are emerging and their use can solve some of the limitations of transistors. One route of research to use them efficiently is the inspiration of the biological brain, which is composed of a very large number of complex features and which allows accomplishing complex and varied tasks.* "

ELECTRONIC chips are no longer extraordinary products as in our daily lives, electronic is present everywhere, in our cars, our coffee machines, our fridges, not to mention our telephones, which are almost a physical extension of ourselves. The large amount of data they provide requires processing, analysis, and decision-making. Advances in telecommunications have enabled massive data transmission over the cloud involving data centers where the data is processed. Even if new telecommunications technologies try to provide technical answers to the energy consumption of networks and the Internet of Things (IoT), energy consumption due to data sent by IoT will remain a major issue. Without a very large battery capacity, it is not certain that they will have a long autonomy.

Another approach would be to limit the data transmitted, so that only those data that are truly necessary would be transferred. Such an approach consists of integrating native intelligence into systems in order to select important information. But this data analysis is challenging; indeed, to be truly interesting, the associated energy consumption has to be lower than the energy consumption related to communication.

For now, electronic circuits have not been designed for this purpose. From the most complex computer to the smallest microcontroller in a watch, the operating principle remains the same. Rethinking the computer architecture paradigm is an opportunity to answer to some issues of modern electronics.

In this chapter, we will begin by detailing how the operating principles of today's electronics are coming to the end, detailing its limits, and how to reduce energy consumption by working on both technology and architectural level. We will highlight the important role of memory for the future of electronics by presenting bio-inspired architecture where memory plays a crucial role, in the brain.

## 1.1 The fundamental limitations of Electronic hardware

In 1960, at the Bell Lab, the first MOSFETs (Metal-Oxide-Semiconductor-Field-Effect-Transistors) were invented [1]. This invention paved the way for a great adventure with the beginning of modern electronics. Between the first microprocessors composed of a few thousand transistors and today's multi-billion transistors [2], the implementation procedures are quite different. Today, highly complex tools are employed to be able to integrate all these transistors [3], but also because the constraints of today's technologies are much more demanding than in the past [4], when a design could be done by hand. At the same time, system architectures have become more and more complex, and most of them are owned by large companies that have been working on successful system architectures for decades with thousands of engineers.

Currently, from server and personal computer, to mobile processors, electronic circuits have reached incredible performance. It is possible to perform simulations extremely quickly and at low cost, to play video games with exceptional graphic quality even on mobile phones, to do augmented/virtual reality with VR headsets only limited by movement in a confined space.

What is quite interesting is that all these advances in the field of electronics are linked to a single principle: the scaling of transistors, with the reduction of the dimension of a single transistor. The transistor scaling was theorized by Robert Dennard in the early 1970s (Figure 1.1 (a) & (b)).

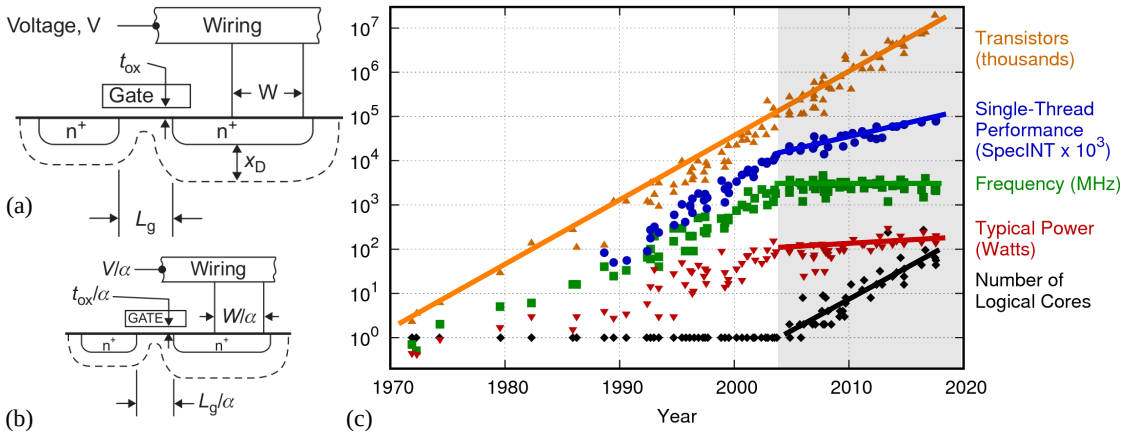



Figure 1.1: Dennard scaling & Moore's law [5] adapted from [6], [7] & [8]: (a) Schematic illustration of a classic CMOS (Complementary Metal Oxide Semiconductor) Bulk transistor technology , (b) transistor following Dennard scaling –dimension reduced by factor alpha– and (c) Moore's law trend, around 2005, the Dennard scaling stops.

The operation principle of a transistor is very simple. It consists of controlling the current flowing under the gate of the transistor between the drain and the source by applying a voltage at the gate. By decreasing all its dimensions –including the thickness of the gate oxide – by a factor $\alpha$, and decreasing the voltage applied at the gate by the same factor, it is possible to maintain the same electric field value under the gate oxide. Since the main source of power consumption of the leak-free CMOS (Complementary Metal Oxide Semiconductor) transistors is the charging and discharging of the gate, the resulting energy consumption is $E = CV^2$. Likewise, with the scaling of transistors, the maximal operating frequency defined by the gate delay $\tau = CV/I$, with I proportional to $(1/tox)(W/L)\mathrm{V}^2$ is increased by a factor $\alpha$. Traditional scaling rules (but no longer valid), imply that the energy consumption of a single transistor can be decreased by a factor $\alpha^3$, the frequency increased by a factor of $\alpha$, the transistor density increased by a factor of $\alpha^2$, while keeping the power density constant. Thereby, to perform the same calculation, between 1971 and 2012, the number of transistors has not only exponentially increased, but also the individual performance of each transistor has been significantly improved. To perform the same logical operation, between 1971 and today, energy consumption is more than 400,000 times lower.

What about the coming decades? As can be seen in Fig. 1.1, a change in the scaling trends occurred in the mid-2000s as Dennard's scaling met some difficulties. The number of transistors still increased exponentially, but the operating frequency has been stuck around a few GHz. The reason for this stop in progress is the significant increase in variability, noise, and leakage of nanometer-scaled transistors. To rectify it, a simple solution has been used: reduc-

ing the supply voltage much slower than suggested by Dennard scaling. However, this trend has the consequences of increasing the temperature and, to the point that heat dissipation may no longer be possible with conventional techniques. Water cooling techniques exist but are very expensive [9]. As the size of the transistors continues to decrease, the energy density rises inexorably, until we are not able to power them anymore. However, the systems architectures are now composed of heterogeneous cores, with GPU (Graphical Processing Unit), CPU (Central Processing Unit), or more complex systems where the full architecture is not used 100% at the same time. Thus, the number of cores has been increased but most of them are used at half of their maximum performance. Remained cores will be "dark" or dimmed, this phenomenon is called Dark Silicon [10].

Despite these limitations, the electronics industry is making more and more efforts to continue to follow Moore's law with incredible sophistication of transistors [11]. The two main technologies used today to reduce transistor dimensions below 28nm are FDSOI [12] (Fully-Depleted Silicon-On-Insulator) and FinFET (Fin field-effect transistor) [13]. These two-transistor structures are very different from the classical structure of the Bulk CMOS transistor. The manufacturing techniques of these transistors is very complex, and the cost of production machinery has become extremely high [14]. Ultimately, the true limit of transistor scaling will be a physical limit. If we imagine the ultimate electronic device, it would work with a single electron. So, to make a binary switch, the state of the electron would have to be stable enough to not be activated by thermal energy. According to basic physics, a fundamental limit for this energy barrier is $E_b = log(2)\mathrm{k}_b T$ and is 0.003aJ at room temperature. However, following basic physical derivations, if we have $10^{10}$ devices in our microchip and we do not want errors for ten years then the energy barrier required is 0.24aJ [15]. Today, we have technologies that use around 20aJ per basic switch. Physics, therefore, allows possible improvements by a factor 100, which is not negligible, but this margin compared to the 400,000 factor that was obtained over the past 50 years.

What can we do next? The constraints that have just been set: one error every ten years among $10^{10}$ devices is a minimum standard in the electronics industry, but in reality, it is extremely challenging to achieve this type of reliability, especially when it comes to memory. Already today, for many memory technologies, formal error correction algorithms are used [16]. By relaxing this constraint, performance could be much better, but system architectures were never designed to work in regimes where devices have errors. Innovative algorithms for electronics is therefore highly relevant today, with new nanodevices that promise fantastic opportunities.

Given the many limitations of competing devices, CMOS has so many advantages that finding a successor to it has never been successful. Traditionally these devices proposed as alternatives to CMOS are very complicated to use, they have a high variability, a low reliability, and are not scalable as easily as CMOS. To be useful in modern electronics, new devices must have strong qualities. They have to be competitive in terms of energy consumption or speed, but

also in slightly more diverse areas such as: a very high compactness (e.g.: nanotubes...), a complex behaviour (e.g., molecular electronics) or novel features (e.g., sensing, negative differential resistance, non volatility...). In most cases, CMOS is and will remain extremely competitive, so it is very critical before embarking on the use of new nanodevices, to understand its qualities. It is often difficult to develop devices with sufficient reliability for large-scale use. If the number of devices is very large, and a single device is defective, and this causes the entire chip to be discarded, it is certain that it will be impossible to market such a product in view of the costs involved. Adapting circuit design to system and device reliability is now essential. An example of a case study, is the retention time of memories, for 10 years retention an energy barrier 8000 times greater than a system of one hour retention is required. Working with low retention memory device would result in a much lower energy consumption. At the system level, we should consider new computational paradigms, such as approximate computing [17].

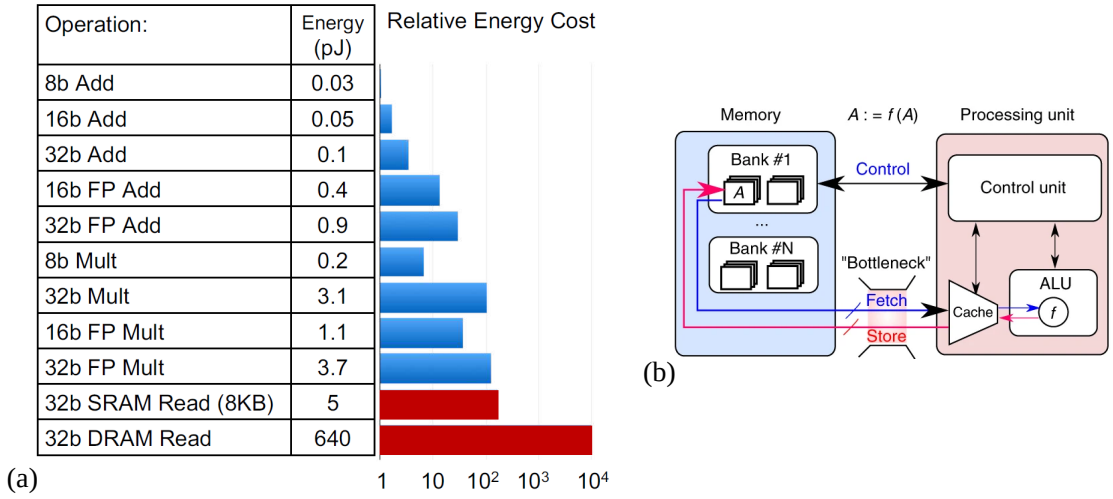



Figure 1.2: (a) Energy consumption for various operations in 45nm 0.9V CMOS technology, taken from [18], originally adapted from [19] & [20] and (b) Conventional operating principle of a processor based on the von Neumann architecture. In this type of architecture, the memory and the calculation are physically separated. A calculation operation f requires data A which is in memory, and at the end of the calculation the result f(A) will go to the same memory unit. The data therefore needs to be continuously moved back and forth between the computing unit and the memory unit, which leads to a communication bottleneck called von Neumann bottleneck. Taken from [21]

Until now, we have been talking about the power consumption of individual electrical devices, but what about the energy consumption of a whole system, for real applications? In Figure 1.2 (a), we can see the relative energy cost of various operations. An obvious observation is that the energy consumption related to the data movement is very high compared to the operations performed by transistors. When we look more closely, we can see that the dominant energy is the access to read DRAM (Dynamic Random Access Memory) data with 640pJ

for 32bit, which corresponds to the data in the central memory of a computer. This high energy consumption is related to the physical distance that separates the calculation from the memory which is considerable in comparison to the transistor level. In principle, energy consumption related to memory access would not be a concern if it did not happen very often. However, this data access is inherent to today's computer architecture, which is based on von Neumann's architecture as shown in Figure 1.2. To perform a calculation a computer needs data, e.g., if it wants to add x with y it will need to fetch the values x and y from the RAM of the computer. Then, the processing unit needs to store these data in the temporary memory called "cache memory", composed of SRAM (Static Random Access Memory) cells, in order to perform the calculation. Once the calculation has been performed, the result is returned to RAM. It is this continuous coming and going of data between different parts of the computer that is actually the dominant source of power consumption in today's computers.

## 1.2 Advancing electronics another way: enhance communication

The understanding of the high power consumption due to the physical distance between the memory and the computing cores explained in the previous section is not recent. It was already remarked in 1977 by John Backus [22]. Now, this information is therefore well known to integrated circuit designers. Despite considerable progress in reducing these data exchanges with architectures that reduce data flow by integrating more and more levels of memory cache, the energy consumption for cache access and power dissipation in electrical interconnect is still a big concern. Connections are indeed a main source of heat dissipation in electronic chips, and their scaling does not come without certain difficulties especially concerning the electric interconnects. The propagation time of a specific signal is limited by a time constant defined by the resistance and the capacity of the interconnection. When we reduce the size of a wire, the resistance increases. It means that for a wire going through the whole chip, the capacitance does not change a lot when the dimensions of the wires are reduced but the resistance is increased as well as the propagation time constant. In order to reduce the energy consumption of the connecting wires within a chip, there is a whole hierarchy of techniques from system level to circuit level [23]. Due to the use of all these techniques, the bandwidth between computation and memory is limited, which implies the use in modern computers of branch prediction [24] to speed up the execution speed of computations. This implies an important energy overhead at the computing core level since part of the performed computations is useless.

To measure the progress of future technologies, and highlight the increase importance of interconnections Wong et al. [25] propose a density metric to evaluate the performance of future developments. They propose to use a density metric that is decomposed into three parts, a transistor density DL for Logic transistor Density, DM for main Memory bit Density and DC

for main memory/logic Connection Density. In today's computers, the main memory is the DRAM, which is mainly composed of very finely integrated capacitors, but which cannot be integrated with the same manufacturing process as the computing core. So, even though both the transistor technology and the DRAM technology are very advanced the connection between the two is limited, so the margin for progress is quite large. This Logic Memory Connectivity (LMC) is very important to define the performance of a circuit. This idea is a guiding line in all our research work on neuromorphic electronics, and therefore, of this thesis.

Today, memory and computing densities are already very high, so an interesting question to ask is therefore how to improve the communication metric. In a conventional computer, a data bus allows the communication between the processor and the memory (cf. Figure 1.2 (b)). For this, the pipeline principle [26] is used, i.e., the data are not sent all at once but sequentially, which allows the calculation to be carried out even though not all the data have been received. The main reason for this sequential approach is that it allows large data processing even though the data bus is limited in size.

Moreover, in a conventional synchronous design of electronic circuit, one of the input signals is the clock that manages the other signals and their timing. The goal of this clock is to share within the circuit the same notion of time: two independent operations can be done at the same time, and then their result can be used at the next clock cycle, which allows a very good efficiency of calculation. Unfortunately, this clock, which by its nature switches continuously, is a very important source of energy consumption. The first reason is that it is always active, even when there is little or no demand on the circuit. Therefore, one of the tricks to reduce this source of unnecessary energy consumption is the use of clock gating [27]. It consists of turning off this clock signal on a part of the electronic circuit when it does not need it. The other reason for the high power consumption of the clock signal is that it has to be shared across the chip, and managing synchronization is very complex on large circuits. Indeed, two registers located at two opposite places in the chip must receive the same signal at the same time, which implies complex circuitry.

An interesting approach without clocks are asynchronous circuits [28] [29]. Asynchronous circuits do not have shared clocks. To perform calculations and communication of data, these circuits use the principle of handshaking. This communication protocol consists of communicating information between two entities by a process of waiting for a response, i.e., one entity performs an operation and transmits a control signal that tells that the calculation is accomplished. Thus, the performance of these circuits can be much higher than synchronous circuits because they are not limited to the longest critical path of the circuit. If a calculation is quick to perform, the result is quickly transmitted. Unfortunately, asynchronous circuits are difficult to design, and most of the time implies an overhead in terms of silicon area due to the handshaking implementation. Despite the growing interest in asynchronous circuits, their popularity is still limited today [30].

To improve the communication between the computing unit and the memory, it is also pos-

sible to avoid conventional communication by electronic wire. The use of light communication instead of electrical communication has been explored a lot in recent years. The growing interest in optical communications is certainly driven by the speed offered by optical fibers. Optical fibers allowed the broadband internet revolution of the 2010s. But when it comes to reducing the energy cost of long-distance communication, we are in a completely different field from the approach of reducing the energy related to communication between memory and computing. In order to replace short-range electrical communication with optics, it is not only important to have a low loss in optical communications but also to have all its devices with low power consumption such as photon generators (i.e., lasers), modulators (which convert information electrical signal into optical signal), and photodetectors [31]. Great progress has been made in this field and is starting to be commercialized but the communications that are replaced are not the ones that concern us in this thesis, in general, these optical circuits are used for communication in Datacenters [32] between the different motherboards of the different servers which are a significant source of the global energy consumption.

Optical circuits are objectively not easy substitutes for conventional electronic circuits at the transistor level. The first reason is that optical communications use waveguides that have relatively large dimensions (micrometers) compared to the dimensions of electrical communications. Nevertheless, the idea of working with silicon photonic optical circuits [33] would be very promising in the future if it becomes possible to integrate optical communications circuits closely to conventional electronic circuits. The exceptional bandwidth, multiplexing [34], and the fact that waveguides can cross each other without interfering are three properties that are largely underestimated for the future of electronics. To overcome the issue related to the size of optical waveguide, plasmonics [35] combines the qualities of both electronics and optics. The transmission losses and the difficulties to co-integrate this technology with conventional electronic are the two main issues that limit their use for the next generation of circuits.

Enhancing communications between processor and memory can be performed by reinventing communication techniques and technology. In my opinion, the idea of adding additional technology to the current CMOS technology seems very promising for future embedded systems. By pursuing this idea, we can then go further than Moore's Law.

The idea would be to reduce the energy consumption related to communication by increasing integration like Moore's law but using new technologies. This approach is therefore often called More than Moore [36]. The first benefit of using new technologies is that it adds new and complementary features to CMOS. CMOS is mostly powerful to perform simple binary operations. Even though it is possible to work with transistors in an analog regime their precision is very limited to perform real analog calculations, especially since scaling is not as easy in the analog regime. On the other hand, for features that cannot be present within a CMOS core, hybrid circuits that include several technologies with new features are a great opportunity for the future low-energy electronic circuits.

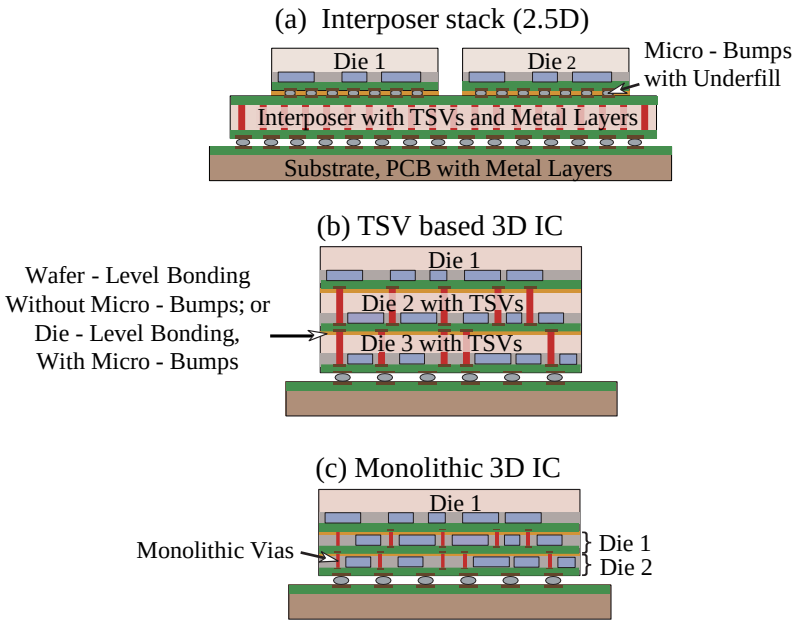



Figure 1.3: Overview of 3 different solutions to integrate different technologies on the same chip from [37] (a) Integration using an interposer, two different technologies placed side by side that communicate with each other through the interposer. Also called 2.5D. (b) 3D integration based on TSVs, the different technologies are superimposed on each other but during the manufacturing process the technologies were initially separated on different wafers, during the final assembling the TSVs pass through the whole chip to connect the different chips together. (c) Monolithic 3D integration approach, the different technologies are manufactured one after the other, there is no final assembly and the technologies are connected to each other through conventional metal connections.

In order to achieve the integration of different technologies within the same chip, different methods summarized in Figure 1.3, exist. With the advent of Through-Silicon Vias (TSVs) [38], even when manufacturing techniques are not compatible between the different technologies, 3D integration has become widely possible and is even being developed industrially. There is 3 types of 3D integration structures. The first one, often called 2.5D [39], is presented in Figure 1.3 (a) it use an interposer (i.e. an electrical connection interface) using TSVs to connect two dies together. The second one (Figure 1.3 (b)), called TSV based 3D-IC consists of directly connecting the different technologies manufactured on different dies by superimposing and connecting them using TSVs. This technique is very promising but the number of stacks is limited. First, because of the alignment problems and secondly because of the thermal heating involved by the technologies integration. The last approach, called monolithic 3D-IC or pure 3D (Figure 1.3 (c)), involves directly integrating different technologies on the same die using manufacturing processes that are compatible with each other. This technique seems to be the most appropriate, but it is the most difficult to implement, since it implies the compatibility of the different manufacturing processes. The more advanced the technologies are, the more it

becomes an issue.

While historically, memory and computing were physically very far from each other, using 3D and 2.5D integration technology has led to the emergence of High Bandwidth Memory (HBM) [40] (widely used in modern graphics cards (GPUs) [41]) where memory and computing core are very close physically. Despite recent advances in this type of integration, there remains a considerable challenge for this type of integration process, which concerns the thermal implications of such structures. In addition to the influence of temperature on its own structure, a stack of one technology can influence another stack. Despite the growing interest in 3D integration, it remains very difficult to implement. When designs are made with redundant patterns, such as memory arrays, it remains relatively simple to make designs. But when it comes to integrating different technologies with different features to perform different operations as one would do with a conventional CMOS circuit, it becomes much more complicated since there is no fully automatic placement routing flows, or they are very complex to set up. There is considerable industrial interest in 3D technologies and a lot of research is being done in this direction, but there are very important limitations for mass production. Reliability, testing possibilities, and high cost are difficulties that are rarely presented in pure academic research but are very important for industry.

To sum up, to limit the energy on communication, either we work on pure communication between the computing units, the memory, the inputs/outputs, the sensors, or we make it disappear by integrating the different technologies together as much as possible [42–44]. Especially, that it is now possible to integrate the different technologies with each other.

## 1.3 The special case of memory

Until very recently, the industry has focused on optimizing either memory technologies or high performance logic technologies independently. As these technologies have progressed, the manufacturing processes have become more complex. The monolithic 3D integration of memory with compatible high performance logic manufacturing processes is therefore very difficult. By being a little less demanding on the performance of the devices, but being able to integrate them with other devices that do not have the same function, it is possible to make very innovative chips. Some very impressive achievements have been made by research teams recently [44], and the industry seems to have embraced the importance of integration in the future chips [45, 46].

We have seen in Figure 1.2 that the communication between the CMOS computing core and the DRAM represents a large part of the power consumption in a full system. But the energy consumption of the memory in the CMOS core, i.e., for registers as well as SRAM, is also large, especially since it is used for the cache memory and is therefore very intensively used.

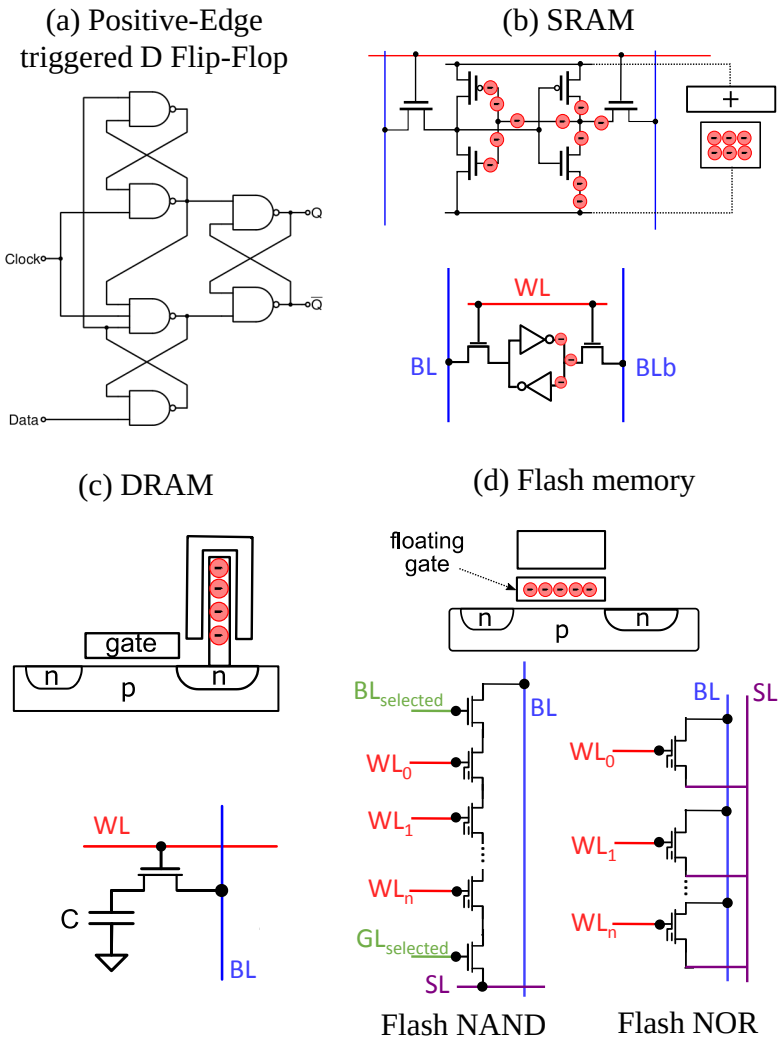



Figure 1.4: (a) Schematic of a Positive-Edge triggered D Flip-Flop acting as the classical register in CPUs, (b) Static Random Access Memory (SRAM): Two CMOS inverters connected back to back. The charge is confined within the barriers formed by FET channels and gate insulators. (c) DRAM, Capacitor connected in series to a transistor (d) Flash memory, charged are trapped in a floating gate influencing the current that flows in the transistor channel & the two configurations NAND and NOR that setup many devices.

An SRAM single array element is composed of 6 transistors (Figure 1.4 (b)): one flip-flop cell and two selecting transistors. The charge is confined in a feedback wire loop between the two barriers formed by the FET channels and the gate insulator. By contrast to a memory register, which by construction does not need a read circuit, an SRAM needs a slightly complex periphery to be read and programmed. To read an SRAM cell, the Bit Line ($BL$) & the complementary Bit Line Bar ($BLb$) are initially floating high, when the Word Line (WL) is raised. In the configuration of the Figure 1.4 (b), $BLb$ is pulled down, and $BL$ is pulled up, or the opposite to have the other binary value. In reality in large memory arrays, the load of the electrical wires

being relatively important, a more complex reading is implemented. It consists of charging $BL$ & $BLb$ to Vdd and when WL is active a slight voltage difference between $BL$ & $BLb$ appears and is detected by a sense amplifier. To write an SRAM as for a latch, BL is loaded to Vdd and BL to the ground or vice versa, and then WL is activated.

As shown in Figure 1.4 (c), the DRAM memory itself consists of a capacitor that stores the electrical charge rather than a feedback wire-loop connected to the drain of a transistor. It is more compact than SRAM, but slower, and needs to be refreshed regularly since reading destroys the content of the capacitance, and it is subject to capacitor leakage losses. The reading is achieved by first charging BL and then activating WL. According to the charge contained in the capacitor, a voltage change appears on BL and is compared with a voltage threshold by a sense amplifier. The SRAM manufacturing processes are compatible with high-performance CMOS, but DRAM does not have this compatibility, so DRAM is typically implemented in a dedicated chip. However, some foundries provide some embedded memory e-DRAM that can be implemented within high-performance CMOS [47, 48].

Flash memory, (Figure 1.4 (d)) relies on a special transistor, which contains an extra gate suspended in an oxide that can contain trapped charges. The reading is made by detecting the current flowing under the floating gate. When applying a voltage to the grid, if the floating gate is discharged, electrons can flow in the channel, but when the floating gate is charged, the electric field is not sufficient to let the electrons flow. Flash memories can be multilevel-cell, depending on the possible level of charge in the floating gate. In this case, several gate voltages are required to sense the current. To write data, an electric current must flow between the source and the drain and a higher voltage must be applied to the control gate. Some of the electrons pass between the electrodes and will tunnel towards the floating gate, through the oxide. Erasing a cell is done in the same way, but by passing a negative voltage across the control grid. The electrons then tunnel from the floating gate to the substrate. The writing process tends to damage flash memories, resulting in a relatively low endurance. Reading is faster than writing or erasing, because the floating gate does not have to be filled or emptied with electrons.

There are two main categories of Flash memory: NAND Flash memory and NOR Flash memory, shown in Figure 1.4 (d). NOR memory is the conventional Flash memory where each device can be addressed separately; one end is connected to the source line and the other end is connected to the bit line. Flash memory is called NAND when several memory devices are connected in series. In NAND memory the access is sequential while in NOR Flash the access can be random. In modern process, the great advantage of NAND memory is that it can be stacked in 3D [49], and therefore its memory capacity is much larger than NOR Flash. In addition to being non-volatile memory, i.e., it is not erased when the power is turned off. Flash memory is an inexpensive and high-density memory. However, it has some drawbacks: an asymmetrical read/write speed performance, memory retention is poorer when its dimensions are reduced, the endurance is low, and decreases sharply when we increase the number of levels per cell.

Flash memory, therefore, requires the use of complex error-correcting codes, capable of correcting multiple errors. Finally, it is not Back-End-Of-Line compatible with high-performance CMOS core.

To work correctly a computer must continuously fetch data from memory, but this data is often very large and therefore needs to be stored in a memory space outside the computing chip, otherwise, it would saturate all the registers and the SRAM that the chip contains. Even when using the best caching strategies, a classical processor will sometimes fetch some data from the DRAM, even though it could store this data within the computing core to reuse it the next fraction of a second. The memory access speed, memory capacity, and compatibility of the technology with a CMOS core are interrelated and interdependent. The current solution is to make distinctions between each of the categories.

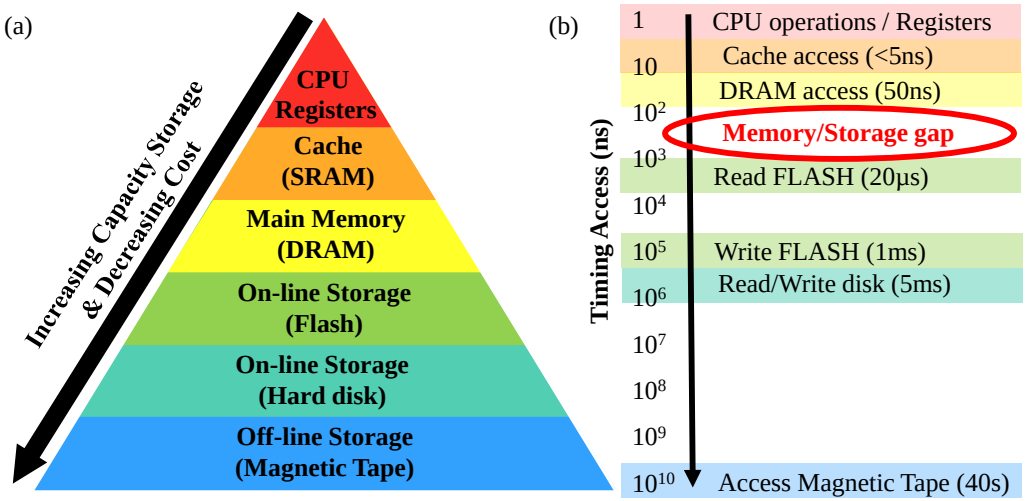



Figure 1.5: (a) Hierarchy of the different memory technologies in use today, there are two categories of memory, working memory, which is composed of registers, SRAM and DRAM, and storage memory, which is composed of flash memory, hard disks or magnetic tapes. The larger the storage capacity the lower the cost. (b) Corresponding the different memory technologies with their access and write times, it can be seen that there is a technology gap between DRAM and flash memory, and even more so if we consider writing time.

Each of these memories has different characteristics and performance, so they are not used for the same functions. The registers are very fast memories and are part of the computing core, they are used to synchronize data as well as to briefly store data between two clock cycles. The SRAM is also very fast, it is organized as a memory array, and data access is done by addressing. It is on the same chip as the computing core and is often used as cache memory to speed up data transfer between computing and DRAM. DRAM, on the other hand, is the computer's main memory, which is relatively compact and has a relatively short access time. Figure 1.5 (a) shows the memory hierarchy present in electronic computing systems and Figure 1.5 (b) highlights the access times required for different technologies. It can be seen that there is a clear

separation between two types of memory capacities, working memory and data storage, which do not have the same access times. The gap is quite clear, especially if we look at the write time of Flash memory compared to DRAM. NAND Flash can offer high-density storage, increasing as devices scale down to smaller dimensions; but while prices are dropping, the speed gap between memory and storage remains significant. Additionally, the device endurance of Flash is low and is not likely to improve.

Modern computers have long been designed to limit this constraint and hide this gap in memory access time. In data centers, the speed of DRAM and its high power consumption are a big concern, but the non-volatility of Flash memory, and its relatively long access time does not seem to be the ideal technology either. So very recently, to fill this gap in memory access time, new technologies have been explored and are now being commercialized [50]. This range of memories is called Storage-Class Memory [51]. There are a multitude of candidate technologies to fill this empty space in the memory hierarchy. The main ones are shown in Figure 1.6. These technologies come with two new features compared to DRAM: the non-volatility of the stored information and the Back-End-Of-Line compatibility. Therefore, open the way to new possibilities of circuit architecture.

The origin of these memories can be tied to the works of Leon Chia[52], who theorized a fourth elementary passive device, called the memristor M, in addition to the resistor R, the inductor L, and the capacitor C. A paper entitled "The missing memristor found" [53] published in 2008 claimed to have found this theoretical device. Many controversies appeared since theoretical guarantees were not provided [54]. There are serious doubts about the existence in the real world of the memristor as initially proposed. For this reason, we will not use this word in this thesis and prefer the more generic term resistive memory. But this is not because non-volatile memories do not correspond to the precise characteristics of an ideal "memristor" that they are not fascinating. As mentioned earlier, these memory devices are outstanding candidates to fill the gap between DRAM and storage memory. Each resistive memory technology has pros and cons and the main ones are listed in Figure 1.6.

The first memory device shown in Figure 1.6 (a) is the Spin Transfer Torque Magnetic RAM (STT-MRAM). This device consists of a free magnetic layer whose direction of the magnetization can be changed and a pinned magnetic layer which is fixed. By current injection, it is possible to detect the state of the free magnetic layer, the current being greater when both magnetic layers have their magnetization in the same direction. A spin transfer of the electrons passing through the junctions can reverse the direction of magnetization of the free layer. Before the end of the 2000's, magnetic junctions did not use spin transfer, and the direction of magnetization of the free layer had to be changed by a magnetic field, which made it poorly scalable. Today, this technology has made huge progress and manufacturing companies have their technologies ready to use [45] [46] [55].

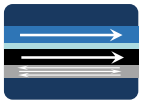
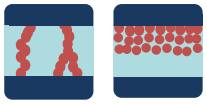
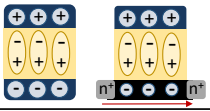
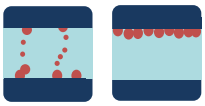
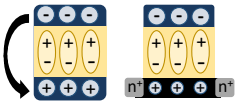
| | (a) STT-MRAM | (b) RRAM Filamentary / Interfacial | (c) PCM | (d) FeRAM + FeFET |
|---|---|---|---|---|
| **LRS « 1 »** |  |  |  |  |
| **HRS « 0 »** |  |  |  |  |
| **Physical Principle** | A free magnetic layer is controlled by injected spin-polarized electrons that flow throw a pinned magnetic layer by spin transfer torque | Formation & dissipation of oxygen-vacancy or metallic filament/interfacial | Phase transition between a crystalline phase (low resistivity) and an amorphous phase (high resistivity) | Ferroelectricity material have electrical polarization in the spontaneous state, polarization can be reversed with external electric field |
| **Strengths** | - Mature Technology<br>- Very fast<br>- Very good endurance<br>- Radiation-tolerant<br>- Back-End-Of-Line compatible | - Good retention at high temperature<br>- Simple structure (low cost)<br>- Fast & low current switching<br>- Back-End-Of-Line compatible<br>- Possible Co-Integration with Access Device<br>- Possible 3D integration | - Mature Technology<br>- Large (HRS-LRS)/LRS ratio<br>- Good Endurance<br>- Highly scalable<br>- Back-End-Of-Line compatible<br>-Very fast | - Fast<br>- High density<br>- High Endurance<br>- Use of Ferroelectric for FeFET as embedded DRAM : compatible with high performance transistor<br>- Opportunities with Ferroelectric Tunnel Junction |
| **Weakness** | - High current switching<br>- Small (HRS-LRS)/LRS ratio<br>- Scaling strongly affect retention<br>- Fast switching & reliability tradeoff | - Highly Immature technology<br>- Low Endurance<br>- Switching Reliability<br>- Initial “forming” step<br>- High Intra device variability | - Power-Hungry RESET<br>- Lower retention at high temperature<br>- Device failure over time<br>- Resistance Drift | - Destructive read<br>- Poor density<br>- FeFET low endurance like flash |

Figure 1.6: (a) Energy consumption for various operations in 45nm 0.9V CMOS technology, taken from [18], originally adapted from [19] & [20] and (b) Classic operating principle of a processor based on the von Neumann architecture. In this type of architecture the memory

The second category of memory shown in Figure 1.6 (b) is a oxide based resistive memory (RRAM). The basic idea is that a dielectric material, which is normally insulating, can be forced to conduct through a filament or conduction path after a sufficiently high voltage is applied. The formation of the conduction path can result from a variety of mechanisms, including defects, and metal migration. The switching process can be filamentary as well as over the entire contact surface with the electrode (interfacial). Once the device has been programmed to a low resistance, it may be reset (broken, resulting in high resistance) or set (re-formed, resulting in lower resistance) by an appropriately applied voltage. In this thesis, we will present a chip design that uses this technology using hafnium oxide (HfO2) [56].

The third technology shown in Figure 1.6 (c) is Phase Change Memory (PCM). This device is made of a nanometer volume of phase change material (that can switch between amorphous and crystalline phase) between two electrodes. The amorphous phase features a high electric resistance, while the crystalline phase is conductive. By applying a voltage to the electrodes, the increase in temperature at the material can induce a phase change, which then changes the resistance of the device.

The last memory device shown in Figure 1.6 (d) is the ferroelectric memories. Ferroelectric

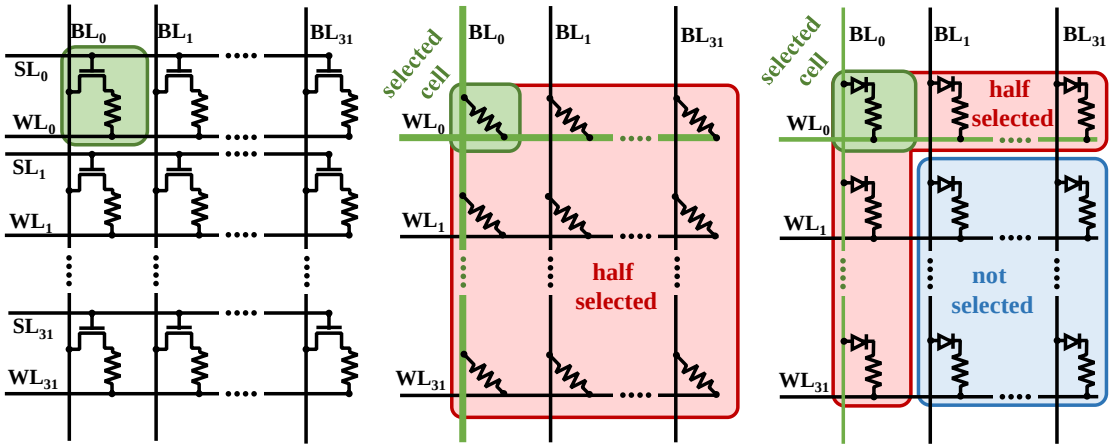
memories are different from the three devices presented earlier, since to read their state, they rely on the electric charges located at the electrodes and not on the state of resistance. The circuits used for reading therefore look more like the circuits used for DRAM. The principle of FeRAM is similar to that of DRAM except that the capacitance of the device can be modulated according to the ferroelectric material state between the two electrodes of the capacitor. Due to the effect on the electric field of ferroelectric materials, it is possible to use it similarly to the flash memory at the gate of a transistor to affect on the electrons flowing under the gate: this memory device is called FeFET. There are also the Ferroelectric Tunnel Junction memories, which combine the tunnel effect with ferroelectric materials but which are still in very preliminary stages [57]. Moreover, these devices are very promising because their strong non-linearity allows matrix products to be produced without selectors [58].

## 1.4 Integration of Non-Volatile-Memory (NVM) for In-Memory Computing (IMC)

In order to integrate the different memory technologies mentioned in section 1.3, it is important to consider the method of integration, i.e., how to assemble them with each other. Since these are memory devices, it seems quite obvious at first glance to integrate them similarly to SRAM cells. In SRAM, two access transistors per bit are needed because the change of state can only be done by forcing the state of the SRAM cell from one side or the other. This constraint is specific to SRAMs and does not apply to memory devices based on their resistance state. That means that each device, needs to be only associated with one access transistor to be addressed sequentially (1T1R see Figure 1.7 (a)) to be programmed and read. It is this method that is classically used to use these devices digitally. The problem with this method is that it uses a transistor that is below the first metal level 1 to address devices that may be as high as metal level 4-5. Firstly, it means that all metal levels and transistors under the memory array are dedicated to it and therefore cannot be used for anything else and secondly, due to the dimensions of metals that increase at each level, the device itself can be smaller than the device access metal.

To increase the memory density per unit area, it is possible to do without access transistors, i.e., to design a memory array only consisting of memory devices (1R configuration 1.7 (b)). Only two levels of metals are therefore required to access the devices. To program a device, or to read it, it is enough to apply a voltage difference between the word line (WL) and the bit line (BL). Unfortunately, in this configuration, all devices will receive half of this voltage difference, which will induce leakage currents that will flow through all half-selected devices. The last configuration presented in Figure 1.7 (c) has associated a bipolar diode-type device that stops low positive and negative current, which eliminates the effects of sneak paths in a large part of the memory array. This method transfers the constraints on the manufacturing of

a selecting device other than CMOS. Many technologies are candidates and can be combined with the memory device technology [59]. Some are even already marketed by manufacturers. The technology is not public, but according to multiple news reports, it seems to be Ovonic Threshold Switch (OTS) [60] [61].




(a) 1T1R configuration (b) 1R configuration (b) 1S1R configuration

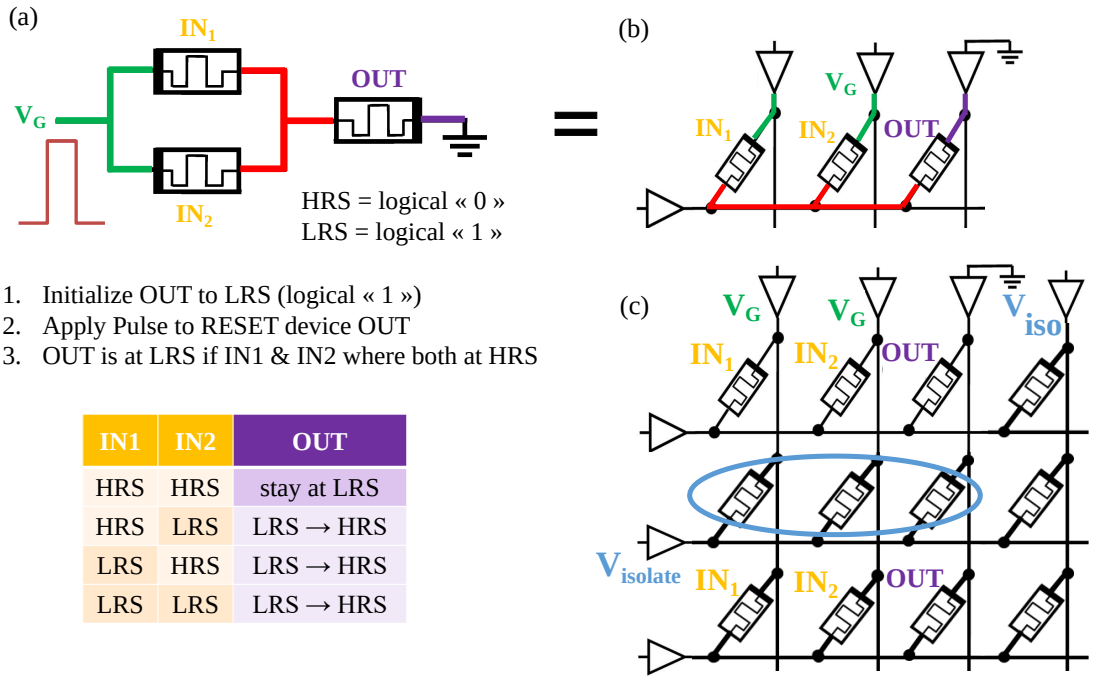
Figure 1.7: (a) Memory cell array using 1T1R configuration, one cell gather one Transistor and one Resistor (1T1R). (b) Memory cell array without access transistor, when a voltage difference between $BL_0$ & $WL_0$ is applied, the device in green is programmed put all the other also receive a voltage difference (c) Configuration which associates a resistor with a selector, avoiding sneak paths (1S1R).

Given the advance of memory technologies, which now seems inevitable, it is fascinating to wonder what they can allow us to achieve from a system point of view. First, as we have seen, these memories can be very densely integrated within a CMOS core. It is therefore possible to envision systems that rely much more on memory.

First, an advantage of the new memory devices that are shown in Figure 1.6, is that they are non-volatile, whereas all memories that are usually used in processors are volatile. This feature offers possibilities to replace some memory registers with non-volatile ones [62] [63]. The possibility of using this non-volatility to replace registers may seem a little curious for today's architectures as these registers are continuously loaded. In reality, for many applications, –e.g. low utilization processors in satellites– these registers are not so much used, and finally maintaining the data in the register that needs to be continuously supplied with power can be an issue. A trade-off between the on-time and off-time of a register may therefore require the use of non-volatile memory within the CMOS itself for applications using a classical von Neumann architecture [64].

The possibilities offered by such memory devices are not only related to the registers or memories classically found in a processor. It is also possible to perform operations directly with

these memory devices. The first method for performing calculations with memory devices is to use the read circuit of the binary value of the memory device and to augment it with logic functions. A circuit classically used to know the binary value of a memory device is a PCSA (Pre-Charge Sense Amplifier) [65]. The difference in resistance between two memory devices is detected using this PCSA circuit. In fact, it resembles to a pre-charged SRAM cell, which is configured in such a way that, depending on whether the current is higher in one branch or another, the SRAM cell ends up in one state or another. By adding transistors in the different branches where the memory devices are located, it is possible to perform resembles between the binary value stored in the memory and an input logic value [66]. The detailed operating principle and the circuits will be presented in detail in the chapter 4 of the thesis, since they are basic elements of our research.




| IN1 | IN2 | OUT |
|---|---|---|
| HRS | HRS | stay at LRS |
| HRS | LRS | LRS → HRS |
| LRS | HRS | LRS → HRS |
| LRS | LRS | LRS → HRS |

Figure 1.8: (a) Basic element of Memristor-Aided Logic, first the output memory device is initialized to low resistance state coding for a logical "1" then a pulse $V_G$ is applied at the input of the gate which is such as it is able to RESET to high resistance state. If the two devices IN1 & IN2 are at high resistance state the main voltage difference is seen by the device two input devices and then the device is kept at LRS, otherwise if IN1 or IN2 is at low resistance state, the current mainly flow throw LRS devices and then the device OUT see the voltage difference and is RESET to HRS. (b) Same basic element but arranged in such a way that it match a memory array. (c) Full Memory Array with Magic basic element, to perform the NOR operation an isolation voltage is applied. The devices circled in blue do not see any voltage difference. Adapted from: [67]

The second method using memory devices for the calculation aims at relying only on mem-

ory devices without any CMOS overhead. The inventor S. Kvatinsky of this principle describes it as true In-Memory-Computing, in the sense that all operations are done only with memory devices. The operating principle is based on the memory arrays previously shown in Figure 1.7, but the logical inputs and outputs of the system are directly the resistance values stored in the memory devices. This method is called Memristor Aided LoGIC (MAGIC) [67]. The logical inputs and outputs of the system are directly the resistance values stored in the memory devices. The operating principle is shown in Figure 1.8.

The principle of Magic is therefore very ingenious, the inputs are always programming pulses, and depending on the state of resistance of the various memory devices, a logical operation is stored in a memory device. Since the NOR logic gate presented in Figure 1.8 is a universal gate it means that any logical operation can be done by repeating this logic gate. However, for technological development, this approach is relatively difficult to implement due to the imperfection of devices and the sensitivity of Magic to errors. Moreover, since the devices are continuously programmed, the voltages involved are relatively large, which implies higher energy consumption and relatively rapid aging of the devices.

Beyond this extreme approach, the possibilities offered by memory integration are considerable and numerous have been developed in recent years to take advantage of it. Four examples of systems using various types of In-Memory computing are listed and briefly introduced below:

- In the paper called "Solving matrix equations in one step with cross-point resistive arrays" [68], Daniele Ielmini et al. show that by using a crossbar array of memory devices, it is possible to solve a system of linear equations and find the eigenvectors of the matrix in a single step using Ohm's law and Kirchoff's laws. This calculation is carried out by reading the voltage at the output of a current-voltage converter where the memory device acts as a feedback resistor.

- Resistive memory can also be used for cryptographic purposes. With a goal of low power consumption, lightweight cryptography has been introduced. The work [69] presents an implementation and a power/area analysis using STT-MRAM devices.

- Another approach is to use the input data and compare it with a memory content called Ternary Content Addressable Memory (TCAM). This technology is already used to perform very high-speed SRAM research, but has recently been developed with RRAM [70]. In addition to having industrial applications with classical computer architectures, a whole field is being opened up for the use of this type of memory in the field of neuromorphic with associative memory.

- The approach of hyperdimensional computing using memory devices is presented in both [71] and [72]. Hyperdimensional computing is about representing semantic, holographic, spatter code, or other high-dimensional information and comparing it with data

patterns for analysis. This type of algorithm requires both high-dimensional data processing and a large amount of memory, which seems particularly suitable for the use of our memory nanodevices.

Given the technological advances in the field of memory, a wide variety of applications are possible. Some authors advocate that memory could replace everything, including logical operations performed with transistors [67]. In our research, and in this thesis, we express the idea that it is important to mix CMOS and new memory devices to be energy efficient. CMOS being dedicated to computing and memory devices that save data. Thus some questions appear and some problems that we should try to address in the next few years: given the human capacity to perform complex cognitive tasks, is there a way to take inspiration from it? And is there a connection between its memory-oriented structure and its low energy consumption?

## 1.5 Bridging the gap between memory and computing: a brain inspired vision

It is commonly accepted to define as the von Neumann architecture, the architecture used in today's computer that physically separates memory from computation. This architecture was developped for the EDVAC1 project, designed for functioning with a program stored in memory. Despite the fact that von Neumann imagined the computer at the time as being composed of two distinct units, one of calculation and one of memory, one of his primary motivations was to take his inspiration from the brain. He also began a book shortly before his death describing the brain as a calculating machine: "The computer and the brain" [73]. Even if the knowledge of the time was not as developed as it is today, his interest in the brain in his book brings out the fundamental differences that exist between the brain and a computer. In Table 1.1, a non-exhaustive comparison of the main characteristics of the brain and the computer is made.

A computer, as it works today, is very fast and can perform very sophisticated mathematical operations in a very short period of time. But it struggles on most cognitive tasks, not even some tasks that a small mouse performs naturally. Even though we have been able to develop very powerful algorithms to perform cognitive tasks, especially with the recent progress of artificial intelligence, there is still a long way to go to have intelligent and low-energy consuming systems. What makes biological inspiration so attractive? The list of differences between the working principle of the brain and a computer is much larger than the list of similarities. What should be highlighted is the energy consumption. When we look at the brain, its energy consumption to perform cognitive tasks is several orders of magnitude lower in comparison to that of a computer. This low energy consumption is largely due to the fact that in the brain, memory and calculation are co-located, and therefore, the high energy consumption related to communication (cf. 1.2) is avoided.

| | Computer | Brain |
|---|---|---|
| | **Differences** | |
| Tasks | Arithmetic | Cognitive |
| Speed | ≈ 1 ns | 1 ms |
| Devices | transistor deterministic | noisy & stochastic synapses |
| Exactness | 32/64 floating point (real numbers) | imprecise real numbers |
| Temperature | 5°C – 70°C | 36°C – 38°C |
| Operations | Sequential | Massively parallel |
| Communication | synchronous, clock based | asynchronous |
| Memory | **separated from calculation** | **co-localized** |
| Energy | $\approx 10^3$ W | 20W |
| Structure | 2D | 3D |
| | **Similarities** | |
| | Evolution: search for **energy optimization** | |
| | Communications based on **binary electric signal** | |
| | Structure composed of **nanoscale elements** | |
| | **Going further?** | |

Table 1.1: Non-exhaustive comparison table between the main characteristics of the brain and a computer.

Our research on neuromorphic electronics aims at trying to get closer to the functioning of the brain, while keeping in mind that some features of the electronics are still very attractive. Many computer architects will say that we should not foolishly copy biology, by using as a metaphor the working principle of an airplane compared to a bird flapping its wings. To this objection, I often point out that their metaphor is a bit shaky, since it compares two totally radically different systems, the plane sometimes weighing more than 200 tons and a bird that weighs at most a few kilograms. Moreover, the Wrights Brothers', two famous aviation pioneers, were largely inspired by the flight of birds to design the firsts airplanes [74]. In fact, the neuromorphic can finally be brought closer to this comparison because electrons are used in electronics but not ions like the brain. So the level of difference between airplanes and birds is comparable with the neuromorphic approach.

Being convinced of the effectiveness of biological inspiration to design energy-efficient electronic systems, it seems critical to take a closer look at how the brain works in more detail. The brain is made up of about 10,000 billion interconnected neuronal cells. There are different categories of neurons [75], which all consist of the same basic elements; it is the arrangement of these elements that allows us to categorize neurons. Figure 1.9 shows a biological neuron and its main components. There is a large number of such schematics of neurons in the literature describing these devices, but we have chosen to use this representation to highlight two important points. The first point is that the brain is often seen as a multitude of neurons

communicating with each other, when in fact each neuron has close to 10,000 synapses, which means that the brain can rather be seen as an ensemble of synapses with a few neurons nested inside it, with a multitude of wired connections! The brain can therefore be considered as an extreme realization of in memory computing.

The second point is the importance of dendrites, since most neural network models are based only on connections between two neurons via synapses but are not influenced by other dendrite connections at the neuron level. This consideration of dendrites is increasingly used to understand how our brain is able to learn [76–79], but this aspect of the brain topology is not kept as an inspiration in this thesis.

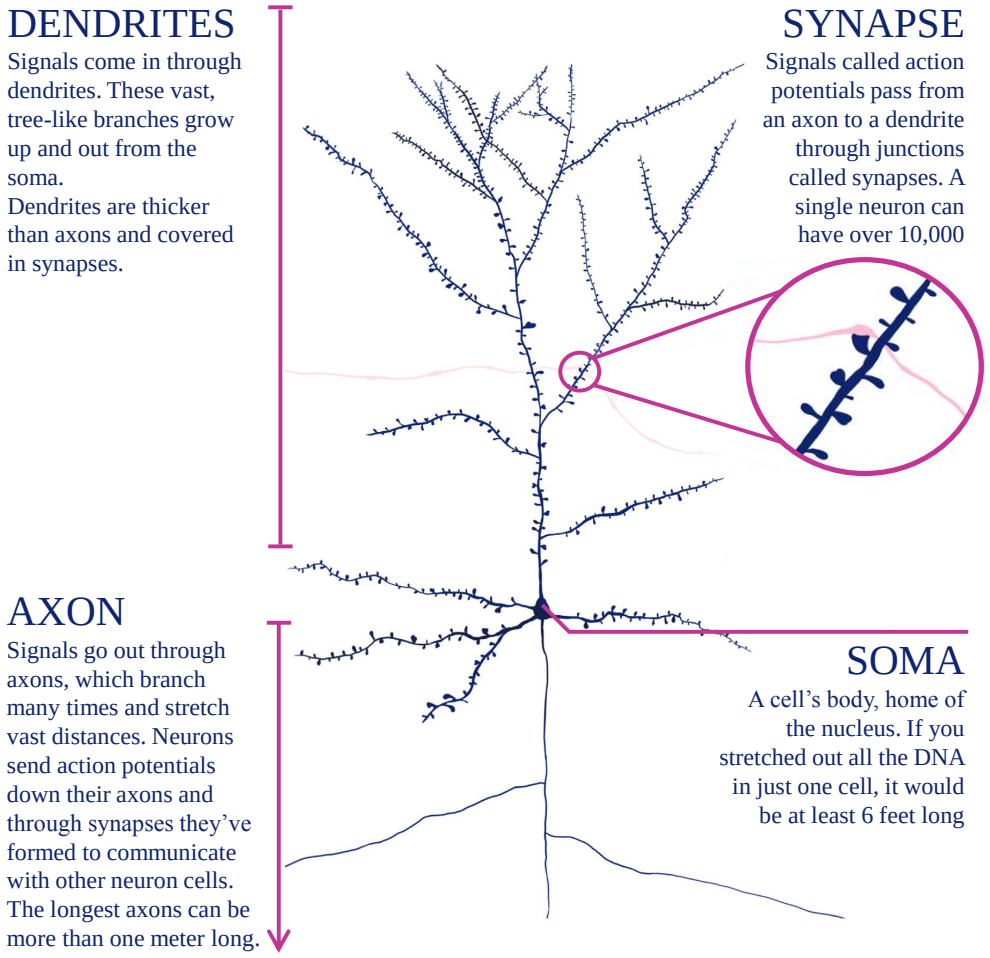



Figure 1.9: Biological Neuron and his main devices. The electrical signal from pre-synaptic neurons comes to the dendrites throw synapses that are integrated at the soma. In the picture the electrical signal propagates from the top to the bottom.

Neurons are the main brain cells; they communicate with each other through long fibers called axons. The axon of a neuron transmits nerve impulses, called action potentials or spikes, to other neurons or specific target cells located in more or less distant regions of the brain. The connection between two neurons takes place via synapses between the axon of the pre-

synaptic neuron and the dendrites of the post-synaptic neuron. Each neuron integrates information from other neurons through dendrites within the soma. Many models describe the behaviour of neurons in relation to the action potentials they receive. The simplest model is the integrate and fire model: its principle is that the soma sums up the information received in the form of spikes at the dendrites, thus increasing its membrane potential. Once a threshold is reached, the neuron transmits an action potential too. This model being a bit too simplistic, another model called leaky integrate and fire is more commonly used to simulate biological neurons. It is similar to the integrate and fire model in that the soma sums up spikes but an extra term of leakage is added, which tends to bring the membrane potential down to zero over time. Thus, to declare an action potential, a neuron must receive typically multiple spikes in a short period of time. In any case, this model remains simple and does not reflect the physical reality of the neuron. On the other hand, a model called Hodgkin-Huxley model [80] was awarded a Nobel Prize in Physiology or Medicine in 1963. This model uses a series of differential equations to describe how action potentials are transmitted. Unfortunately, the complexity of the equations is such that it is very difficult to simulate large neural networks with this model. Concerning the synapses, their size being very small, it is highly complex to make measurements on individual synapses, and therefore to fully understand how they work. Nevertheless, we know that there exist two categories of synapses, excitatory synapses, and inhibitory synapses, which will either increase the membrane potential or inhibit the potentiation [81]. It is also difficult to extract information about the value of synaptic weights, i.e., they are considered to be imprecise and noisy [82].

Concerning the communication between neurons, we know that communication is asynchronous, i.e., there is no global signal between neurons that controls the transmitted information. Additionally, communication is intrinsically binary, since the only signals transmitted are spikes [83]. Some theories suggest a supplementary form of not entirely binary communication might be possible, involving spike bursts [84].

That being said, the coding of the transmitted information is not well understood, and there are different theories [85]. The first theory, rate coding [86], as its name indicates, corresponds to coding the information transmitted between neurons by a frequency, in other words, a number of spike per unit of time. This type of coding is very interesting as it directly suggests how the brain is coding real values.

However interesting experiments performed by Thorpe & al. [87] have shown that neurons can communicate information with one and only one spike. To do it, he analyses the distance between the visual system and the part of the brain dedicated to image recognition, with a comparison to the reaction time interval needed to recognize an image. He concludes that only one spike could be transmitted. The temporal coding theory could explain such an experiment. We will see in this thesis, that the use of binarized neural networks, which will be extensively described in Chapter 4 can also be directly linked to explain this phenomenon. The idea here is to code information in a time interval, for example, a large real value would be coded in a small

time interval, and a small real value in a long time interval [88].

Another possibility of coding is to use a set of neurons to code information, i.e., rather than looking at what is happening at the individual level, we look at the collective level. In neuroscience, population coding [89, 90] is a method that involves the use of the joint activity of neurons to represent information. In Chapter 5, we will study this type of coding in more detail and show that the assembly of superparamagnetic nanodevices embedded in a CMOS core can encode information and represent complex functions. A final theory is sparse coding [91], where the main idea is to code the information as a linear combination of a set of small simple patterns and thus represent more complex patterns. This type of coding allows unsupervised learning [92], i.e., a data item does not need a label.

The challenges of understanding how the brain works are therefore manyfold. In addition to the difficulties in understanding the underlying mechanisms of the various devices of our brain, linked to their noisy and imperfect character, understanding its functioning in its entirety is very difficult. Let us imagine someone who looks at a microprocessor. Even if he or she perfectly understands the working principle of the transistor, understanding what the microprocessor does in its entirety simply by looking at the arrangement of the transistors is almost impossible. This is why many neuroscience studies do not focus on individual devices but rather at the system level, observing both human behaviour and imaging the brain as a whole. Many theories concerning reasoning have thus been developed [93], an interesting one which will be addressed in this thesis with a hardware implementation presented in Chapter 3 is the Bayesian theory [94, 95].

To understand the learning mechanisms in the brain, some neuroscience studies have examined how the modification of synaptic weights is done. At the level of an individual synapse, the major local learning rule that has been extensively studied, particularly for neuromorphic electronics is Spike Timing Dependant Plasticity (STDP) [96]. This learning rule is very simple, and it is local as it needs only the information of the post-synaptic spike event and the pre-synaptic spike event. If at a synapse level, a post-synaptic spike appears just after a pre-synaptic spike, the synaptic weight is increased, i.e. , reinforced. If it is the opposite, i.e., a post-synaptic spike occurs just before a pre-synaptic spike the synaptic weight is reduced. Despite the great interest in this learning law, which is both unsupervised and local, and the rich learning outcomes with hardware implementations using non-volatile memory devices [97], it does not seem to scale up to learn very complex tasks and deep neural networks like the brain. As a matter of fact, some recent neuroscience studies [98] suggest that the STDP learning rule would not really be a learning rule as such, but would rather result from a more inherent learning mechanism [99, 100].

While research has been done on a more comprehensive understanding of the brain, the advance of artificial intelligence especially through "Deep Learning" [101] has been so remarkable in recent years that it may help to understand how the brain works. The learning principle of artificial neural networks is mainly based on the calculation of a gradient of an error, which

will be used to change the value of synaptic weights, through a process called backpropagation [102]. In the next section 1.6, we will extensively present artificial neural networks, discuss the challenges, present the main ideas of the learning algorithm, and their first hardware implementations. There is no direct evidence that the brain uses backpropagation for learning, but since backpropagation has proven to be a very effective method for optimizing neural networks, I personally think that the brain might use some kind of backpropagation.

Recent works by AI pioneers like Yoshua Bengio also reinforce this idea [99]. To study how the brain works, a lot of research has been done on artificial neural networks for a long time. One of the earliest attempts were Hopfield networks [103], which are completely connected neural network, which means that all neurons are connected with the others. The values the neurons can take are 1, -1, or 0. The synaptic weight that connects the neurons together makes it possible to store and recreate disrupted patterns as an associative memory. Of the same kind but a bit more complex, a Boltzmann machine [104] is a stochastic Hopfield network with hidden units, i.e., some neurons are used as input to the system and others are used for the dynamics of the system (whereas in Hopfield networks all neurons are used as input). They are very commonly used to estimate the probabilistic distribution of a dataset. Boltzmann machines are very difficult to train because the training time increases exponentially with the size of the network. For this reason, other studies have suggested that to make training easier, it is advisable to limit connections in layer connection with Restricted Boltzmann Machines [105, 106]. They were then extended to deeper networks through the stacking of the restricted Boltzmann machines, called Deep Belief Networks [107].

More recent research is making some sense in bridging the gap between artificial and biological neural networks by looking at how the brain can encode, calculate and propagate an error signal to optimize a function [108, 109]. A first idea is that the synapses within dendrites would have very different importance and roles depending on their position on the dendritic tree [78, 79, 84]. Another idea called Equilibrium Propagation is that the error signal can implicitly be propagated by a difference between alternating phases of signal propagation, free phases where the system evolves to a steady state of equilibrium, and phases nudged by an error signal that evolves to another state of equilibrium. Even if this idea is at an early stage with only preliminary results for the moment, and features some limitations, it seems very promising especially as it can be applied to deep Convolutional Neural Networks [100] and very recently to solve complex tasks [110].

## 1.6 Existing implementation of neuromorphic hardware

Bio-inspired computing became very popular only recently; it is no longer a marginal topic as it nowadays possesses a topic with an industrial dimension. Scientific publications and conferences in electronics are very often oriented in the neuromorphic perspective. For this reason, hardware implementations of neuromorphic circuits are very diverse and all of them can not

be listed here. In this section, I will present my vision on the neuromorphic field with a non-exhaustive list of neuromorphic implementations, that I believe to be the most promising for the next decades. Given the major progress in artificial intelligence and the development of the gradient backpropagation bio-plausibility (see previous section 1.5), we include hardware implementations of artificial neural networks as parts of the neuromorphic category.

The flagship technique used for training neural networks is gradient backpropagation [111]; it is an optimization method to approximate a function using a derivative calculation. In the case of neural networks, this function can be very complex. However, an artificial neural network is no more than a function. This function takes an input x and provides an output y. This function $f$ is parameterized, i.e., its behavior can be modified by changing the parameters, which we will call here the weights $w$. The function $f(x, w)$ outputs a prediction y. From this prediction, a loss can be defined – usually, cross-entropy is used for classification and root mean square error for regression–. From this error, it is also possible to define the gradient of this loss as a function of the parameters $w$. By stochastic gradient descent, –that means that in each example the parameters $w$ will be modified by a small step in the opposite direction of the calculated gradient–, it is possible to move closer to the parameters $w$ where the loss is low. This stochastic gradient descent is illustrated in Figure 1.10, the global loss landscape of the task is presented in brown, while the loss landscape of the current example is presented in blue. To reach the global minimum $w^*$, a successive presentation of different examples and a gradient descent allows approaching the ideal $w^*$ parameters.

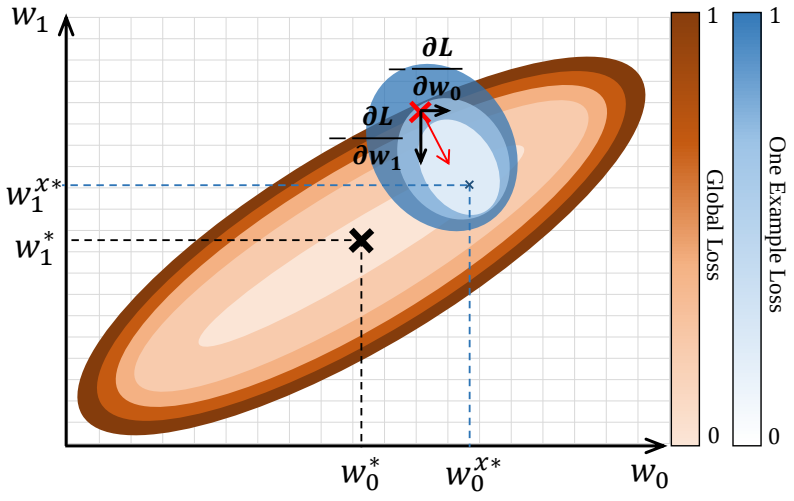



Figure 1.10: Landscape of the global loss in brown as function of two parameters $w_0$&$w_1$, the global minimum is at $(w_0^*, w_1^*)$, landscape of one example loss as function of two parameters in blue, the example loss is at $(w_0^{x*}, w_1^{x*})$. The optimization in stochastic gradient descent consists of calculating the gradient of a multitude of examples, by accumulation of the gradients we get closer to the global minimum.

A simple artificial neuron is presented Figure 1.11 (a), it is a simple computation of a weighted

sum of inputs $x$ with weights $w$. An artificial neuron performs a linear regression, and by adding a nonlinear function at the output of the neuron, the complexity of the representation space becomes higher. It then becomes beneficial to cascade layers of neurons, to obtain a network capable of highly complicated tasks. In this way, a neural network is composed of a multitude of neurons connected to each other, the output of one neuron is the input of another. In general, even if it is possible to have other neural network architectures, the simplest approach is to have a succession of interconnected layers, also called a fully connected neural network as presented Figure 1.11 (b).

The principle of learning a pristine neural network is presented in the algorithm of Figure 1.11 (c), which also highlights the data exchange between the memory and the processing unit in this process. It is possible to decompose the learning of a neural network in three parts:

1. The inference. It consists of the calculation of the function $f(x, w)$ and thus to make a prediction. The only operations to be performed are those performed by the neurons as in Figure 1.11 (a), that is a weighted sum between an input $x$ and weights $w$ and then a non-linear function. When there is a multitude of neurons, the operation to be performed becomes a matrix-vector product: $w.x$ followed by non-linear function $\sigma$.

2. The backpropagation of the gradient. It begins by determining the loss that corresponds to the difference between the output of the neural network and the expected label. Then, we want to determine the derivative of this loss with respect to the weights ($\partial L/\partial w$). For this, we need at each layer of the network the input X and the derivative of the loss with respect to the weighted sum $\partial L/\partial w$ calculated during the inference step. The first calculation is the derivation of the loss with respect to the weighted sum $\partial L/\partial z$. Then a product between the vector row $\partial L/\partial z$ and the input vector column $x^T$ gives us a matrix of derivative of the loss with respect to the weights. To carry out the calculation of the other layers, the derivative of the loss with respect to the inputs X must be determined, for that a product between the transposed weight matrix and the derivative of the loss with respect to the weighted sum $z$ is sufficient.

3. The update of the parameters. In a pristine neural network, the only parameters are the weights, so they are the only parameters to be optimized by gradient descent, but in more complex neural networks, it can be possible that other parameters need to be optimized. Once the calculation of the gradient is done, we just have to modify the $w$ parameter by subtracting from it the gradient re-scaled by an alpha parameter called the learning rate. The smaller it is, the slower but more precise the learning is. No matter what type of neural network architecture is used, these steps are always performed.

In Figure 1.11, a very simple neural network architecture was presented. However, more complex architectures all rely on this basic functioning. We often talk about Deep Learning, but these are simply neural networks that have a large number of successive layers. The principles that we just discussed were identified as early as 1974 [112], unfortunately, deep learning did

not work for many years. But in 2012, deep Convolutional Neural Networks (CNN) [113] that allowed to extract redundant features inside inputs Data by sharing weights at each layer, outperformed all others machine learning algorithm for image recognition. This incredible performance did not come through a conceptual breakthrough as backpropagation was already proposed to train neural networks around 30 years before [112, 114], and CNNs already existed, but rather through the possibility of having a very large amount of data and high-performance graphics cards.

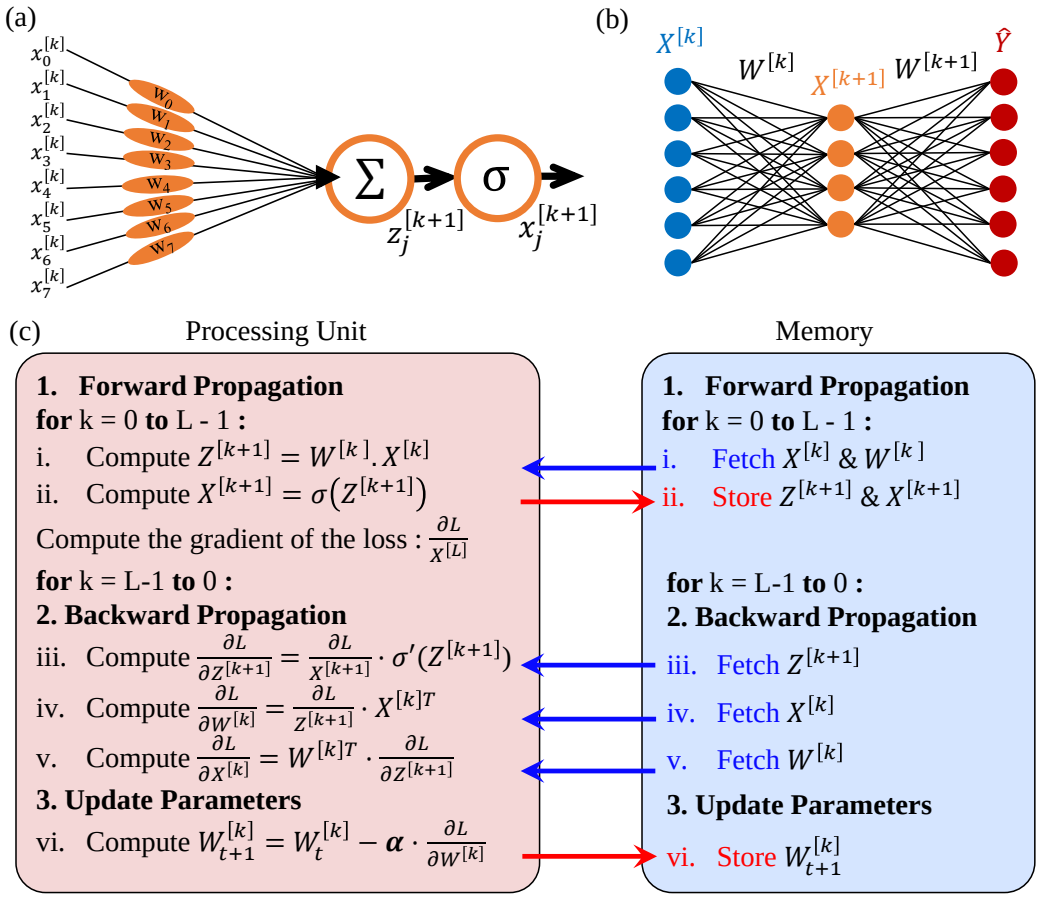



Figure 1.11: (a) Artificial Neuron, simple model the ouput of a neuron is the weigted sum of inputs x followed by a non-linear function $\sigma$ (b) Two hidden layers artificial neural network

Figure 1.11 (c) presents the algorithm to learn a feed-forward neural network in the situation of memory and computation being distinct. During the forward pass, the parameters of the neural network weights must first be loaded, and then the neuron outputs that are required for the backpropagation step must be stored. During the backpropagation, the memory request is still present since the data of the forward pass and the weights must be loaded, then the new weights are stored in the memory, and this at each layer of the network and at each

example.

As mentioned in sections 1.1 and 1.2, the energy consumption related to data exchange is dominant over this type of arithmetic calculation, especially since here the mathematical operations that are carried out are very simple but need to be performed many times. To perform these calculations, GPUs have proven to be very fast, they have a very large number of computing cores and access to a very large dedicated DRAM featuring a high bandwidth. They can therefore both load weights and store in buffer memory the various data required for learning. The market for GPUs specifically optimized for deep learning is growing very fast [115]. But there are also other accelerators for Deep Learning. GPUs are very fast for performing matrix computations but they are not designed to perform only these operations. There is therefore further research to optimize the calculations involved in neural networks. The research in this area is very intensive, we have decided to categorize the different research concerning Deep Learning accelerators into four main categories:

- The first category, composed of CMOS with non-embedded memory, are accelerators that are very reminiscent of GPUs and that are already available on the market. They mainly use DRAM to store the parameters of the neural network. The most famous one is the Tensor Processing Unit (TPU) [116] developed by Google, which allows the acceleration of tensor operations. The first generation of TPUs was only used for the forward propagation. The new generation of TPU is composed of two kinds of TPUs, TPU Edge, dedicated for ultra-low power consumption at inference [117] and the higher-precision TPU is dedicated for the learning.

  Other accelerators were developed for high-speed inference like the Movidius Stick [118]: a flash drive that we can plug into a computer to perform neural network inference. Much other research focuses on designing ASIC with low energy consumption, especially for CNN. We will not mention them because they rely on a separation between memory and computing, which is not relevant for this thesis. Most of these works are focusing on the optimization of the memory access [119].

- The second category had embedded-memory with SRAM [120]. This category is particularly interesting for this thesis, as most of the challenges at the system level are the same as the ones we try to address with our emerging memory devices. Therefore, the important common issues are parallel or sequential data management [121], how to move data to perform convolution operations, and input/output management.

- The third category is the one we study in this thesis. It is based on CMOS and Non-volatile Memory embedded at the technology level. To perform the calculation of the neural network based on matrix product, a classical approach is to use a crossbar array and the Kirchoff laws presented in Figure 1.12. In the forward pass, the calculation to be performed is the product $w \cdot x$, by presenting the input as voltage Figure 1.12 (a) $(V_0, V_1)$, the product is performed by Ohm and Kirchoff laws and the result is obtained as electric current.

Similarly, for the calculation of the backward pass, the product $w^T \cdot \delta$ is performed by applying the voltage Figure (b) $(\delta_0, \delta_1)$, the result is also obtained as an electric current. Updating the weights is more complex. To do it, the $\delta$ and $V$ voltages must be applied at the same time, each device then sees a potential difference, which will have the effect of programming or not the device. In the ideal case, we would like to have a modification of the conductance of the device proportional to the product of the voltage. Most of the non-volatile memory components do not have this behaviour. In order to overcome this limitation, some works have proposed different approaches [122, 123], but it remains a challenging question. The main difficulties encountered are: the asymmetry between potentiation –an increase in conductance– and depression –a decrease in conductance– of the devices, non-linearity, programming noise, or resistance drift in some memory technologies. Some proposed solutions are using a capacitor that is almost linear [124], shows very promising results but it requires significant technological overhead.

- The last category concerns system-level research. This literature does not consider the technical challenges associated with devices. We consider perfect crossbars that can perform calculations at very low energy. It mainly presents architectures that focus on the hardware implementation of convolutional neural networks. The important question that these works try to answer is largely based on the management of non-volatile memory when the weights are shared. They are also studying the impact of different neural network architectures and how to have systems that can be used in a variety of ways. The main circuits at the state of the art are: ISAAC [125] and PipeLayer [126]

Neuromorphic research is not limited to the implementation of artificial neural networks trained by backpropagation. As mentioned in section 5, the complexity, the dynamics of biological neurons, of synapse learning is not limited to matrix product operations. Some more complex hardware implementation taking into account some biological aspects exists. As an example, the SpiNNaker circuit for (Spiking Neural Network Architecture) is a massively parallel system composed of one million of ARM cores (featuring a von Neumann architecture) but, which communicate with each other using spikes to simulate the brain's operating principle [127]. Similarly, the LoiHi chip [128] developed by Intel is composed of 130,000 artificial neurons and 130 million synapses that are simulated with great complexity including dendritic compartments, all communicating with spikes.

The other major technological achievement in neuromorphic circuits that mimic the brain is TrueNorth. As LoiHi does, TrueNorth [129] aims to emulate the functioning of neurons. In these implementations, the synapses are in SRAM and are therefore volatile. This volatility implies that at boot time, all the synaptic weights must be loaded, and when the circuit is not useful it consumes static power.

Other neuromorphic implementations using non-volatile memory devices and biological learning approaches exist. There is an extensive literature on hardware implementations of

STDP with innovative memory devices. The implementation of STDP using resistive devices is presented in [130], and the implementation of stochastic STDP using magnetic memory devices is presented in [131]. In both cases, a particular pulse shape is applied to approximate the STDP. This curve is matched with the device physics; when a post-synaptic spike appears very quickly after a pre-synaptic spike the synapse is potentiated and if it is the opposite the synapse is depreciated.

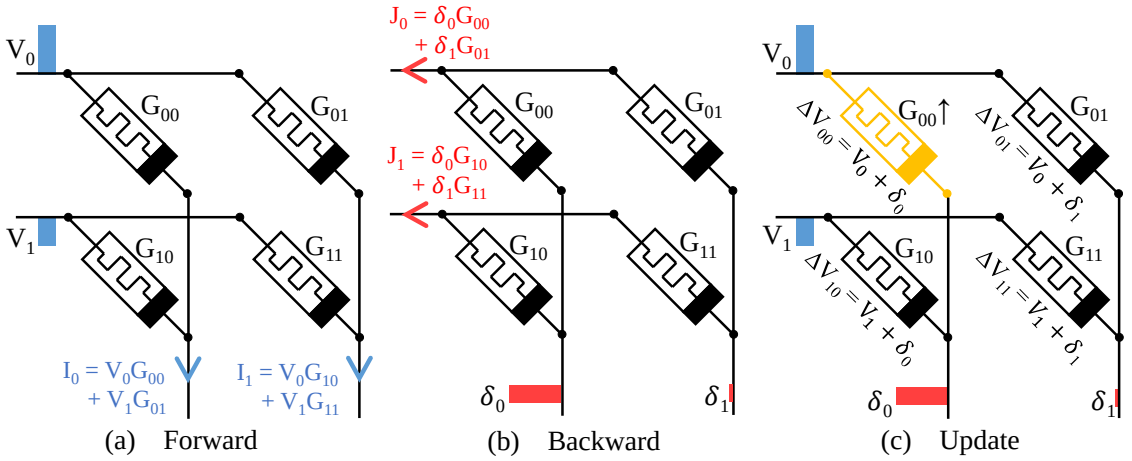



Figure 1.12: (a) Configuration of the crossbar array to compute the forward pass of the neural network, it computes the product of the voltage with the conductance of the devices by Ohm's law and the sum is obtained as current by Kirchoff's law. (b) The backward pass is performed similarly as the forward pass, but in the opposite direction, a voltage $\delta$ is applied where the current was previously read and a current is read where a voltage was previously applied (c) The weight update is obtained by applying a voltage difference at each device proportional to the previous voltage $V$ and $\delta$, if the voltage difference is sufficient the devices are programmed, here the yellow one see a high voltage difference and its conductance is increased

The integration possibilities are not only limited to the use of innovative memory devices, some nanotechnologies such as photonics, MEMS, sensors can be used and benefit from the advances of neuromorphic. For instance, the paper [44] not only combines a multitude of technologies in 3D monolithic integration (CNFET logic and sensors, RRAM, CNFET logic, and silicon logic) but also an accelerated classification module based on a support-vector-machine. Even if it is an artificial intelligence module far from biological inspirations, the idea of integrating sensors, calculations, and memory within the same chip is an approach that should be used for future neuromorphic chips

Neuromorphic optical chip research is a recent development that is beginning to be developed. Some research teams are working on the hardware implementation of deep neural networks with photonic circuits. A first approach is to use a set of programmable Mach-Zehnder interferometers to perform the neuron function [132], performing both the weighted sum operation and the non-linear function for the neuron output. The approach is very interesting es-

pecially since it could be implemented in 3D, as recent work has shown that it is possible to integrate waveguides in 3D [133]. Synaptic weights, so the memory devices are at the center of the neuromorphic approach as discussed in the previous section. Photonic circuits have recently demonstrated the possibility of integrating memory devices and more particularly PCM [43] which can be programmed by laser heating. This phenomenon is well known since it is used to rewrite optical disks [134]. However, it is only recently that the first photonic circuits implementing deep neural networks with integrated memory devices have been demonstrated [135]. The main attractions of photonic circuits for neural network integration are: multiplexing, low-loss, and low cross-talk crossings waveguides [136], low power consumption, and speed.

The other interesting neuromorphic approach that focuses more on the physical behaviour of a system is reservoir computing. The concept of reservoir computing is based on the fact that a physical system can have a very rich behavior, so by presenting certain inputs to the system it can produce interesting measurements. The first interesting property of reservoir computing is that it allows modeling very complex functions, the other interesting property is that it allows producing outputs that are temporally correlated. In other words, the output will not only depend on the input state of the system at time $t$, but also on the input states at previous time steps. This approach has been widely explored in photonics [137–140], and with memory devices [141] or oscillating magnetic devices [142]. These oscillating magnetic devices have been used both for reservoir computing but also because of their very rich dynamics it has been demonstrated both theoretically [143, 144] and experimentally [145, 146] that it is possible to use them to perform complex tasks in the manner of a neural network by the phenomenon of devices synchronization.

Given all these neuromorphic literature, the possibilities offered by both the physics of devices and the ability to produce reliable memory devices for digital implementation, how does this thesis contribute to the field?

To understand the choices made here, it is helpful to make a few remarks on what are the guiding threads of our work. The first remark to be mentioned is that CMOS technology is very efficient for logic operations and that its improvement in the next few years should be only minor. Therefore, in our work, we will not seek to replace CMOS where it performs best, but rather to provide answers to situations where it is not effective. Secondly, we believe that the implementation of non-volatile memory devices at the core of CMOS is the future of electronics to increase both energy performance and computing performance. And that finally, the brain is a great source of inspiration to efficiently integrate memory and calculation to have efficient electronic systems.

# Chapter 2

# Low Energy Inference Neuromorphic System: Bayesian Machine and Low precision Neural Network inference

I have always been convinced that the only way to get artificial intelligence to work is to do the computation in a way similar to the human brain. That is the goal I have been pursuing. We are making progress, though we still have lots to learn about how the brain actually works.

Geoffrey HINTON

"BRAIN *inspiration is complex to implement, especially because all the underlying mechanisms are not known. Nevertheless, advances in artificial intelligence in recent years have made tremendous progress. It is now possible to solve so many remarkable cognitive tasks exceeding human performance. One of the challenges is how these solutions can be embedded in systems. In this thesis, the main approach is to consider that learning is done off-chip (i.e. on a computer) and then, the final implementation of the solution that solves some very specific problems is done on the chip for inference only.* "

RECENT years may have highlighted the limitations of transistor scaling, but new emerging technologies are paving the way for new algorithms. Brain inspiration seems promising as it could reduce energy consumption by bringing memory and calculations closer together, while being intelligent.

There are a variety of bio-inspired algorithms. Some are far from real applications, mainly belonging to the category of bio-mimicry, whereas some are only partially inspired by biology, such as artificial neural networks. The work presented in this thesis relies on just a relative inspiration on biology. Biological functioning and its direct inspiration can be highly successful approaches to create intelligent systems, nevertheless, we still believe that it is preferable to use it just as an inspiration, as long as we are not able to reproduce most of the complex behaviour of the neural system. We preferred to focus on application-oriented work rather than on naive mimicry.

In view of all these developments, our position is quite strong on the question of energy consumption. In this thesis, we mainly study two methods for designing low-energy architectures for in-memory-computing: Bayesian reasoning and neural networks. These two methods differ fundamentally, the first one is inspired by the brain as a whole, in its way of thinking and reasoning, while neural networks are rather inspired by the basic devices of the brain: the synapses and neurons.

Three ideas connect the different works presented in this thesis:

- The use of emerging memory nanodevices to create innovative In-Memory-Computing brain-inspired architectures.
- The reduction of energy consumption for inference – inference is the use of the system without learning, i.e. the system will respond to input but will not be modified from it.
- The use of approximate computation to reduce this energy consumption. For the Bayesian model, approximate computation is achieved using stochastic computing and for neural networks by quantifying the values that neurons and synapses can take.

The objective of the present chapter is to present the basics of Bayesian reasoning and the quantization of neural networks and the connections between the two fields.

## 2.1 Bayesian principle

The concept of intelligence is particularly difficult to define. Nevertheless, it is certain that a large part of intelligence is the ability to recognize when we do not know something. For many intelligent systems, such as robots, autonomous vehicles, or even medical diagnostics, it is important to take uncertainty into account. The description of this uncertainty is critical when working with small amounts of data, noisy sensors, or safety-critical systems. How to model this uncertainty? For Bayesian practitioners, the universal mathematical language for uncertainty are the laws of probabilities. As is the case for humans, the world in which an artificial intelligence operates is uncertain. The data it will encounter will sometimes be noisy, sometimes provide only partial information, sometimes be biased. Integrating probability theory into the design of systems presented as intelligent machines is therefore highly promising.

In recent years, artificial intelligence has been driven mainly by advances in artificial neural networks. Nevertheless, the most complex artificial intelligence system in the world "has less common sense than a rat", said Yann LeCun [147]. Animals are able to learn from very few examples, which is very difficult for artificial neural networks. For example, for a child, from a single drawing of a giraffe, he is able to recognize a live giraffe when he sees one for the first time. No artificial intelligence is capable of such a feat. To recognize a giraffe, a neural network needs to be trained on giraffe data, i.e., it will need to see thousands of giraffe images, mixed with other data, to be able to discriminate whether it is a giraffe image or not. In addition, the output of a neural network will produce almost certain answers –it optimizes the negative log-likelihood–, whereas when the child asks for confirmation, maybe he was 50% confident of his prediction, he is uncertain.

The Bayesian approach is based on Bayes' theorem :

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y\in Y} P(x,y)}. \tag{2.1}$$

This equation is obtained from the sum rule and the product rule that underlie probability theory :

$$\textbf{sum rule: } P(x) = \sum_{y\in Y} P(x,y) \tag{2.2}$$

$$\textbf{product rule: } P(x,y) = P(x)P(y|x). \tag{2.3}$$

This approach is radically opposed to neural networks and the conventional gradient descent optimization techniques. In neural network type optimization problems, we try to optimize the parameters to match a prediction. Once these parameters have been optimized, the system is considered to have learned, and the parameters are no longer modified. On the other hand, in Bayesian models, learning and inference are two problems that can be treated in an equivalent manner.

Probabilistic theory and Bayesian principle are well explained in a Nature review [148], the

book of Daphne Koller [149], and her Coursera's course [150], as well as in the book [151]. A main idea of the Bayesian approach is that instead of looking only at the most likely set of parameters of a model (maximum likelihood or maximum a posteriori), we can consider all the possible settings of parameters of the model, and we try to see for each of that possible settings what probability they correspond according to the data we observe. In other words, we will observe data $D$ and calculate for each possible value of the parameters $p$ the corresponding probability of observing these data : $P(p|D)$.

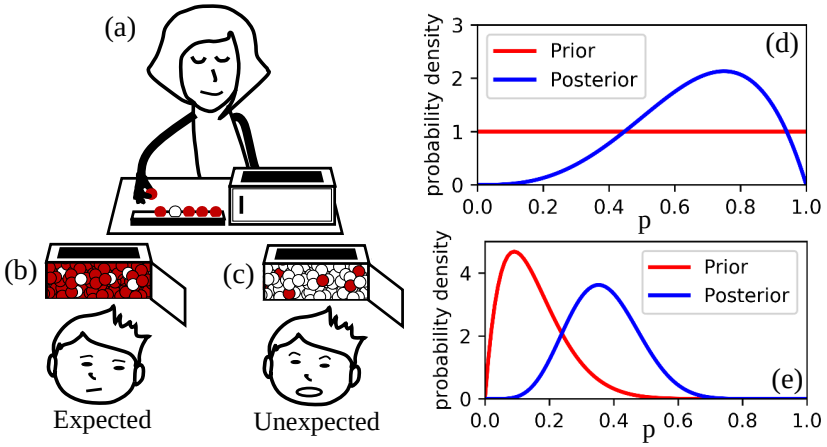



Figure 2.1: (a) Drawing of the experimental demonstration of the Bayesian reasoning on an eight-months child. Once the box is open we observe the degree of surprise of the child (b) when the box turns out to be filled mostly with red balls, (c) when the box turns out to be filled mostly with white balls. (d) Interpretation of the results using Bayesian Reasoning (the initial distribution is plotted in red, and in blue is plotted the distribution after a draw): with a uniform prior on the content on the box and (e) with a different prior on the content on the box.

In addition to the possibility to model uncertainty with Bayesian approach, over the past few years, neuroscience research has shown that our brain unconsciously carries out Bayesian reasoning. Two full years of courses at the "College de France" in Experimental Cognitive Psychology by S. Dehaene are dedicated to Bayesian studies of human learning [94, 95]. Here, we will present only one experiment that seems quite illustrative of our brain's cognitive ability to do Bayesian reasoning. The experiment is presented in Figure 2.1 (a), (b) and (c) and is taken from the studies [152, 153]. An opaque box is presented to an eight-months-old child, and one after the other, balls are taken out of the box in front of him. The balls can be white or red. A trick makes sure that most of the balls that come out are red, and only a few are white. Once this has been done, the box is opened, its contents are presented to the child, and the child's degree of surprise is observed from the time the child looks at it: the longer the time, the more surprised the child is considered to be. If the content is consistent with the sample (a lot of red, little white) (Figure 2.1 (b)), the child is not surprised. But if the content is in contradiction with the sample (a lot of white, little red) Figure 2.1 (c), the child shows a long fixation time (he is

surprised). That is evidence that he had managed to make a hypothesis about the content of the box from the sample presented.

The interpretation of these results can be made from the curve in Figure 2.1 (d). The red curve considered that the child has a prior uniform probability density on p (proportion of red balls in the box), i.e., it is just as likely that there are only red balls, white balls, or any other proportion. The blue curve represents the posterior probability density on p after the draws of five red balls and one white ball. It is obtained from Bayes' theorem, equation 2.1. It is also possible to have a different prior to the experience, e.g., if we have already seen a first draw with 10 white balls and one red ball, we will have the prior of the red curve in Figure 2.1 (e). And after the drawing of 5 red balls and 1 white ball, the posterior probability distribution will be different from the case where there has been no observation beforehand. The probability of having a box with a very large majority of white balls and few red balls is higher in the case of Figure 2.1 (e) than in the case of Figure 2.1 (d).

So we intuitively make Bayesian deductions without even realizing it. But, despite our apparent pragmatism, sometimes our intuition can be wrong. A remarkable example is testing for disease. "Let's say we are tested for a serious illness and the test turns out to be positive. The doctor tells that the test is 99% reliable, what is the probability to be sick? 99%? In reality, there is one missing information which is the prior probability of being sick. From Bayes' theorem – Equation 2.1 – we have:

$$P(sick = 1|test = 1) = \frac{P(test = 1|sick = 1)P(sick = 1)}{P(test = 1|sick = 0)P(sick = 0) + P(test = 1|sick = 1)P(sick = 1).}$$

If we assume that the 99% represents both likelihoods $P(sick = 1|test = 1)$ and $P(sick = 0|test = 0)$ and that the prior probability of being sick is $1/10000$, the posterior probability of being sick is a bit less than 1%. It turns out that, if our prior to being sick was 50%, after the test we would have had that 99% chance of being sick.

The prior probability is therefore very important to make a successful deduction. The beauty of the Bayesian theory is that this prior can be enhanced, even though it was initially very vague. When we see some data, we combine our prior distribution with a likelihood term to get a posterior distribution that will be our new prior. The likelihood term takes into account how probable the observed data is given the parameters of the model. It favors parameter settings that make the data likely. It fights the prior. With enough data, the likelihood terms always win. And we can obtain a precise estimate of the posterior distribution using smaller sample sizes when we use a more informative prior.

To make an estimate of $P(y|x)$, it is common to use sampling: starting from Equation 2.1, we draw $x$ randomly, we look at the conditional probability $P(y|x)$ which we multiply by the prior probability $P(x)$ and we start again. In the example of the young child (Figure 2.1), we started from a uniform distribution. After each ball is drawn by the experimenter, the posterior distribution changes. After the first draw of a single red ball the posterior distribution on $p$ – the

probability of drawing a red ball – increases. Whereas when a white ball is fired, this probability $p$ decreases.

The challenge is that it is not always possible to observe all the possibilities of $x$. If $x$ is a binary image consisting of only 28x28 pixels there are $2^{784}$ possibilities to encode all possible images. Since there is necessarily a limited amount of observable data, the problem has to be modeled in such a way that the number of parameters is reduced. In practice, strong assumptions on the structure of the model have to be made. In the example of the child, the model was very simple, it was based on a single parameter $p$ of dimension 1 which is the probability of drawing a red ball with the probability of drawing a white ball being simply $(1-p)$. But if we had 3 colors (red, white, and blue) we would then have a parameter $p$ of dimension 2 to learn: $p = [p0, p1]$ (where $p0$ the probability of drawing a red ball, $p1$ a white ball and $(1-(p0+p1)$ to draw a blue ball).

Therefore, a model has parameters $p$ that can be learned by presenting $D$ data, and a prior on these parameters noted $P(p|m)$. The prior of the child for a random draw from a box was a uniform distribution on the single parameter $p$. Over the experience, the child changes his perception and determines a posterior probability $P(p|D,m)$. It is therefore through the different experiences or data $D$ that the adjustment of the parameters is made following the equation above:

$$P(p|D,m) = \frac{P(D|p,m)P(p|m)}{P(D|m)}. \tag{2.4}$$

Interestingly $P(D|m)$ does not depend on $p$, so learning the optimal $P(p|D,m)$ can be done using only $P(D|p,m)$ and $P(p|m)$, avoiding the calculation of the normalization term. Once the parameters are learned, the posterior probability distribution $P(p|D,m)$ of the model is used to make predictions. To do it, we use this learned probability distribution, as well as the sum and product rule to obtain the following equation:

$$P(D_{test}|D,m) = \sum_{p} P(D_{test}|p,D,m)P(p|D,m). \tag{2.5}$$

It provides a mathematical description of the prediction on unseen data $D_{test}$. Once the model is learned, the probability distribution $P(D_{test}|D,m)$ is obtained by integrating all the values of the vector space of the parameters $p$. Summing out over all the parameters in the model is the main challenge in computing the Bayesian approach. When the space of parameter vectors is large, it is indeed very difficult to get all the values of the space. Nevertheless, there are methods to move within this space randomly but with a bias allowing to sample the parameters proportionally to their posterior probability distribution such as Monte-Carlo methods [154]. It means that by sampling a large number of parameters we can have a good approximation of the posterior probability.

The main subject of this thesis is to perform low-energy inference once the model has learned all its parameters. In the case of Bayesian models, this involves performing the com-

putation of Equation 2.5, but as we said, such an inference requires complex integration calculations. As a consequence, in Chapter 3 we will present the approximations made and a in-memory design to perform this inference.

## 2.2 Quantization of Neural Networks

### 2.2.1 Neural Networks : background

Traditionally, the training of a neural network is done by a method differing fundamentally from Bayesian learning: gradient descent by backpropagation. The loss used for this backpropagation is generally the maximum of likelihood [155] or the maximum a posteriori. This method does not attempt to determine the full posterior distribution but only has a point estimate of the parameters of the neural network, i.e. its synaptic weights. Unfortunately, a neural network trained with backpropagation is difficult to implement when little data is available.

The training of a neural network for classification can be summarized as follows. We have an input variable $X$ with very high entropy (for example an image), and we try to assign a label $y$ to it. This label $y$ is very simple, and can often be encoded in only one bit which denotes the corresponding class, e.g., the image represents a dog or a cat. This new variable is therefore much less complex than X. However, the difficulty is that this bit is not determined by X in an obvious fashion: there is not a particular pixel in an image that says whether it is a dog or a cat. This information is very highly distributed throughout the image, it is necessary to identify areas corresponding to the characteristics classically present in all images of a specific label.

What does a neural network do to determine the corresponding label of an image? We said in the previous chapter that neural networks are very complex parametric functions. The number of parameters in a neural network is very high and gradient backpropagation is a particularly effective method of changing these parameters to decrease the loss and assign the corresponding label to the input. In general, the architecture of neural networks is made of layers that are connected one after the other.

While it has been shown that a single hidden layer can be sufficient to approximate any function [156], why are multi-layers used in all current architectures? First, to model a very complex function, it appears that a single-layer architecture requires a considerable amount of intermediate neurons. Second, the layered architecture is a series of new representations of the image, at each layer-level, the representation can only be calculated from the previous representation and only affects the next one.

Using the principle of bottleneck information, recent work has made possible a detailed analysis of what happens in the successive representation of neural network layers [157]. By using the mutual information between layers and inputs and outputs, it is indeed possible to have an intuitive understanding of learning in a feed-forward neural network. The information available on the input $X$ input and the label $y$ can only decrease over the layers. Ultimately,

we want the information on the last layer to be structured, i.e., we lose information on $X$ but keep the information on the label $y$. This mutual information can be visualized during the learning process. During the learning, the authors of this work identified two phases: a first one that will learn features very quickly but keeping a lot of information on $X$ and a second one of compression where the information on $X$ decreases. Thus, a neural network can be seen as consisting of two parts, an encoder that transforms input $X$ into a sub-representation $\hat{X}$ and a decoder that extracts the label $y$ from this sub-representation $\hat{X}$.

At the beginning of the training, the encoder will arbitrarily project the $\hat{X}$ sub-representation, and the decoder will try to assign it the label $y$. Thus, initially, the encoder is very simple and the decoder is very complicated, then, as the learning process goes on, things are reversed. During training, the neural network will extract a very large number of features from the different input examples and then forget those that are not essential for the classification. At the end of the training, the encoder becomes very complicated and the decoder very simple since it is generally enough to make a linear classification / a perceptron to obtain the corresponding label.

Neural networks generally feature a very large number of parameters compared to the number of inputs presented. This high complexity provides them with a particularly important power of representation. What is astonishing is that they generalize incredibly well. Moreover, the representation capacity of a neural network is such, that in [158] the authors were able to overfit the CIFAR-10 training dataset [159] by setting random labels instead of real ones. This means that the neural network can store an important part of the features of the dataset and to assign to it a label without real meaning. –CIFAR-10 is a relatively complicated image classification task since it concerns images like dogs, cats, or cars.

Nevertheless, when we have a sufficient amount of data and are well labeled, neural networks are very powerful machines to model very complex functions $f(D,\theta)$. But we have to keep in mind that it is surprising that this function is good to generalize to examples that the network has never seen. For this reason, for researchers working on Bayesian models, reasoning, and explanation, neural networks are seen as black boxes. The reasons for the efficient generalization of neural networks, as interesting as they may be, will not be discussed in this thesis.

The interest of neural networks and their use within this thesis is for a low-energy computing implementation. Chapter 4 will present a low-energy implementation but only inference and in Chapter 6, we will talk about learning. To implement low-energy on-chip artificial neural networks, we are interested here in decreasing the number of parameters (e.g. synaptic weights $w$) of the neural networks. There are different ways to reduce the number of parameters: by reducing the number of neurons and layers, reducing the neuron connectivity, or reducing the resolution of synaptic weights.

Reducing the number of neurons and the number of layers is not an easy task. In general, they are hyperparameters of the model that we train, i.e., they are the prior parameters of the

model and will not be modified during the learning phase. There are methods for optimizing hyperparameters by gradient descent [160], but this is not the case for those parameters that define the architecture of the model. Different methods exist to find these hyperparameters optimally. Classically we do grid-search, i.e. we test a lot of values and choose the optimal one. But when the number of hyperparameters is large, this method cannot be used. A very interesting and biologically inspired method is to use genetic algorithms to search for hyperparameters [161]. With this method, we sometimes find quite astonishing architectures and the definition of the evolutionary environment is fundamental to obtain interesting results.

Another method to reduce the number of parameters is to reduce connectivity, i.e., rather than having a fully connected neural network between each layer, some weights will be shared. Convolutional neural networks are this type of neural network that share the same weights [162]. Sharing weights has a great advantage in reducing the number of parameters in the model, but they were not initially invented for this reason. The idea of weight sharing in convolutional neural networks is to detect translation invariant and redundant features inside an image, such as a vertical or horizontal line, a slightly rounded shape, or much more complex shapes when going into deeper layers. By this simple neural network structuring, the progress of GPUs and the amount of available data it was possible to perform image classification with incredible performance. It was in 2012, with the classification of the ILSVRC competition on the Imagenet dataset consisting of more than one million images and 1,000 categories that neural networks became popular [163].

The current neural network structure for image classification is strongly associated with convolutional neural networks. The advantage of structuring neural networks with convolutions is that it makes it possible to identify shapes in an image by filter translations and that these feature detectors can be easily learned by backpropagation since a filter will receive much more signals than a conventional feed-forward neural network layer. What happens when training a deep neural network without structuring is that it tends to overfit, the parameter space is excessively large compared to the amount of data available, and there is no spatial consistency per layer. Other research has been done very recently to add structuring to existing neural networks (Both structured neural networks for Sequence to Sequence Learning achieves the state of the art results: Long Short-Term Memory[164] and Transformers [165]).

According to Geoffrey Hinton, the convolutional neural networks for image recognition are missing something fundamental, by adding structuring to neural networks he hopes, to be able to improve the angular variations of an image. Some work involving capsules have already shown promising results on rotating images or overlapping digits [166–168]. A capsule is a group of neurons whose activity is expressed as a vector representing a specific entity or part of an object. The length of this vector represents the probability of the presence of the entity and its orientation the estimated pose parameters of the entity. The classic example used to describe capsules intuitively is that of two geometric entities, a triangle, and a rectangle, and depending on their positioning we can see a house or a boat. At a higher dimension, it is a mat-

ter of making the different entities coincide with each other. In a classical neural network, the activity vectors disappear in the scalar product with the weight vector. By working on activity vectors rather than scalars, we can look at the scalar product of two activity vectors and see if they coincide. The addition of such a structure in neural networks aims at better generalization and we can hope that they also allow reducing the number of parameters. The interest of such an approach is that it makes it possible to locate the orientation and the size of the objects whereas in the convolutional neural networks the orientation of the objects cannot be held by the convolution layers, as for the size of the objects, it is indirectly extracted by the max-pooling layers. Hinton was not satisfied with the max-pooling layers, so he invented the capsules. We also mentioned these works because they are inspired by the cortical microcolumns of the human visual system [169, 170]. The biomimetic of cortical columns for neuromorphic has already been studied in some research work [171]. Adding structure in neural networks is of great interest to facilitate generalization and in some cases to reduce the number of parameters but sometimes leads to some difficulties of implementation. First, it can lead to challenges for data movement and leads to a specialization of the architecture and as a consequence less representation capacity.

Finally, the most widespread approach to reduce the size of parameters in a neural network is to reduce the resolution of synaptic weights. More precisely, this approach means that instead of having high-precision synaptic weights, they are approximated by less precise values. Performing such an approximation has two major advantages, the first one is that it may require less memory, and the second one is that the mathematical operations required to perform the calculations required for the network can be simplified. A multiplication and addition operation on a few bits is much less expensive to implement than a precise operation on 64 bits, with fewer transistors involved, reduced area occupation, and less energy consumed. This approach is used in this thesis to reduce the energy consumption of a neural network. Decreasing the resolution of neural networks is a key aspect of the energy efficiency of deep learning for embedded systems. Decreasing resolution can be done for both, the inference, and the learning.

In the next section, we will present challenges for quantized neural network inference. Similarly, in Chapter 4, we will present the challenges for the hardware implementation of low-energy inference of quantized neural networks [172] with emerging memory nanodevices, and more specifically Binarized Neural Networks. While in the last chapter of the thesis, we will discuss existing approaches to perform quantized learning.

### 2.2.2 Reducing the resolution of neural network

In recent years, research to limit the energy consumption of neural networks has made enormous progress. The first motivation for such research is that the energy consumption in the data-center of algorithms is very high, and, therefore, the cost is very significant. The other motivation is that these algorithms have a real interest within embedded systems. Being able to

process data, analyze results, and even make decisions without having to systematically send data to the cloud could revolutionize the Internet of Things. We mentioned in the previous section what the choices could be to reduce the number of parameters of a neural network: synaptic weights are not the only objects of neural networks limiting energy consumption, a study on neurons is also interesting since mathematical operations are carried out between neuron activations and synaptic weights. A mathematical operation between a low-resolution synaptic weight and a high-resolution neuron activation will not have the same impact as if both are low-resolution.

First, I illustrate whether it is possible to decrease the synaptic resolution of an already trained neural network without degrading performance. Fig. 2.2 (a) shows the recognition rate on the MNIST [173] test set as a function of the number of resolution bits of the synaptic weights and activation. The neural network considered here is a backpropagation driven feedforward neural network with a single 512 neurons hidden layer, the activation functions are hyperbolic tangents, and a batch-normalization [174] is performed at the level of each neuron. In this Figure, we can see that reducing the resolution has an important effect on the performance of the neural network only if we go below 4 bits of resolution of weights or activation, i.e., 16 values. Studies on more complex tasks such as ImageNet found that the resolution could be reduced to 8 bits.

The observation made that a few bits are sufficient to have almost as good test accuracy as full precision neural network is quite interesting, indeed, it means that to perform inference, it is not necessary to use processors or graphics cards operating on a very large number of bits, typically 32 or 64. Moreover, the coding type is different, in our experiment we considered that we have linearly distributed the values, with a fixed point coding type [175], whereas in a classical computer floating-point coding is used [176]: one part of the bits are used to code the exponent and the other part to code the mantissa of the considered number. Mathematical operations on a fixed point coding are relatively simple to perform in hardware with a small number of transistors compared to floating-point coding.

Figure 2.2 illustrates that normal training is not efficient enough to reduce to a lower resolution. To go further, it is necessary to adapt the training process. Recently, various works emerged to reduce the number of bits required for the inference of a neural network. The main advances have been achieved in Y. Bengio's group in 2015 and 2016 with the publication of BinaryConnect [177] and then Binarized Neural Networks [178]. In the first work, only the weights are binarized, while in the second it is both the weights and the neuron activations that are binarized. At the same time another group published in 2016, showed how to train Binarized Neural Networks [179].

### 2.2.3 Binarized Neural Network

In a conventional neural network with $L$ layers, the activation values of the neurons of layer $k$, $a_i^{[k]}$, are obtained by applying a non-linear activation function $f$ to the matrix product between real-valued synaptic weight matrix $W^{[k]}$ and the real-valued activations of the previous layer of neurons $a^{[k-1]}$:

$$a_i^{[k]} = f\left(\sum_j W_{ij}^{[k]} \cdot a_j^{[k-1]}\right). \tag{2.6}$$

By contrast, in a Binarized Neural Network, excluding the first layer, neuron activation values as well as synaptic weights assume binary values, meaning +1 and −1. The products between neuron activation $a$ and weight $w$ values in Equation (2.6) then simply become logic XNOR operation:

| $a_j$ | $W_{ij}$ | XNOR |
|---|---|---|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |
| 1 | 1 | 1 |

The sum in Equation (2.6) is replaced by the popcount operation, the basic function that counts the number of ones in a data vector. The resulting value is then converted to a binary value by comparing it to a trained threshold value $\mu_i^{[k]}$. Equation (2.6) therefore becomes:

$$a_i^{[k]} = \text{sign}\left(\underset{j}{\text{popcount}}\left(\text{XNOR}\left(W_{ij}^{[k]}, a_j^{[k-1]}\right)\right) - \mu_i^{[k]}\right), \tag{2.7}$$

where sign is the sign function.

The hardware implementation of such Binarized Neural Network is clearly attractive for low power consumption in edge computing, since it is based only on logical functions that can be implemented with only a few transistors. Regarding learning, this remains an open question that will be addressed in the last chapter of this thesis.

The method for training a Binarized Neural Network is presented in Algorithm 1 adapted from Courbariaux et al. [178]. And the inference phase corresponds to the forward propagation only. The first important point to know is that during the learning process, synapses actually have two distinct weights: a first binary weight which is used to make the inference and a second non-binary weight which is the one that is modified during the update phase of the weight. Without this real-valued hidden weight, the weight update is not possible directly on the binary weight. The binary weight is then the sign of the realed-value weight.

**Algorithm 1** Conventional BNN training model

**Require:** training data : $X_{train}$, targets output $y_{train}$, previous binarized and real weights $W$ and $W_a$, and previous threshold values $\mu$

**Ensure:** updated weights $W_{t+1}$ and $W_{a,t+1}$, updated BatchNorm parameters $\mu$ and $\sigma$

**1. Forward propagation**

**for** $k = 1$ to $L$ **do**

$W^{[k]} \leftarrow \text{sign}(W_a^{[k]})$

$z^{[k]} \leftarrow W^{[k]} \cdot a^{[k-1]}$

$\widehat{z}^{[k]} \leftarrow \text{BatchNorm}(z^{[k]}, \mu^{[k]}, \sigma^{[k]})$

**if** $(k < L)$ **then**

$a^{[k]} \leftarrow \text{sign}(\widehat{z}^{[k]})$

**else**

$a^{[k]} \leftarrow \text{softmax}(\widehat{z}^{[k]})$

**end if**

**end for**

Compute gradient of softmax cross entropy loss : $g_{a^{[L]}} = \dfrac{\partial C}{\partial a^{[L]}} = a^{[L]} - y$

**2. Backward propagation**

**for** $k = L$ to $1$ **do**

**if** $(k < L)$ **then**

$g_{a^{[k]}} \leftarrow g_{a^{[k-1]}} \circ 1_{|a_{k<1}|}$

**end if**

$g_{\widehat{z}^{[k]}} \leftarrow \text{BackBatchNorm}(g_{a^{[k]}}, \widehat{z}^{[k]}, \mu^{[k]}, \sigma^{[k]})$

$g_{z^{[k]}} \leftarrow {W^{[k]}}^T g_{\widehat{z}^{[k]}}$

$g_{W_b^{[k]}} \leftarrow g_{\widehat{z}^{[k]}}\ a_{k-1}^T$

**end for**

**3. Update parameters**

**for** $k = 1$ to $L$ **do**

$W_{a,t+1}^{[k]} \leftarrow \text{Clip}(\text{UpdateAdam}(W_{a,t+1}^{[k]}, g_{W_b^{[k]}}), -1, 1)$

$(\mu^{[k]}, \sigma^{[k]})_{t+1} \leftarrow \text{MovingAverage}(\mu_B^{[k]}, \sigma_B^{[k]})_t$

**end for**

Regarding the activation of neurons, it is necessary to make an approximation of the derivative of the activation function to be able to back-propagate the gradient through it. As the sign function has no derivative, generally the derivative of a hardtanh is used as an approximation, i.e. the STE (Straight-Through Estimator) [180, 181].

Two other key ingredients are to be taken into account for learning Binarized Neural Networks, the first one was batch-normalisation as mentioned above. This normalization aims to have neurons that are not always in the same state whatever the input provided. The second key ingredient is the momentum on the gradients [182], to perform a correct weight update we will take into account the dynamics of the gradients in the form of momentum. In most cases the Adam method [183] is used, but this is not the only one that works well.

### 2.2.4 Quantized and Binarized Neural Network : a comparison

Using Binarized Neural Networks (BNNs) is very interesting, because it both reduces the number of bits needed and facilitates the mathematical operations involved in a neural network. Nevertheless, Binarized Neural Networks have a lower power of representation than classical neural networks, which implies that it is necessary to increase the number of neurons at each layer to obtain equivalent performance. If this increase in the number of neurons is too important, the interest of Binarized Neural Networks can be questioned.

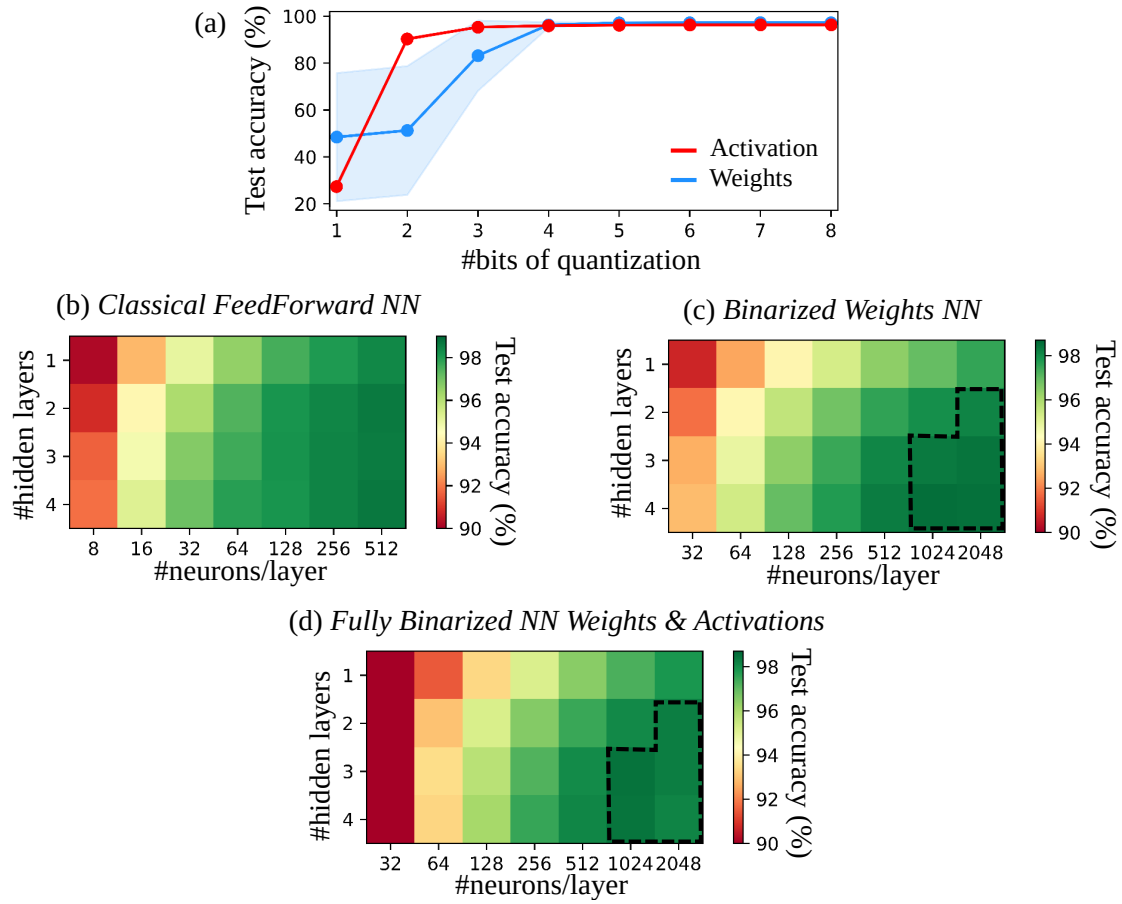



Figure 2.2: Impact of quantization on the performance of neural networks. (a) Test accuracy on MNIST data-set for various bits size quantization for the activation and the weights on a two hidden 512-512 neurons. Accuracy on the MNIST test-set for (b) a classical neural network, (c) a binary weight neural network, (d) a fully Binarized Neural Network for various size.

To address this question, we have plotted three colormaps in Figure 2.2. Each of these colormaps represents the test accuracy as a function number of the number of layers and neurons per layer. The colormap of Figure 2.2 (b) represents the performances for a classical feeedforward network with real values, Figure 2.2 (c) with binary weights but real activations and Figure 2.2 (d) with both activations and weights that are binary. For the case of binary weights, col-

ormaps (c) and (d), an inflation of the number of neurons by a factor of 4 is performed, and which brings equivalent performances with real-valued neural networks. In both colormaps (c) and (d), a black dashed surrounded area corresponds to the configurations where the number of bits required for the binary weights is greater than the 8-bit fixed-point coding colormap (b). To ensure that binary neural networks are interesting regarding the amount of bits required to encode all the synapses, we should accept to have a little drop in accuracy.

Intermediate quantized neural networks can also be used, e.g. Ternarized Neural Networks (TNN), but also require special training and special hardware implementation [184]. The implementation of such a TNN is attractive as it has a greater expressivity than Binarized Neural Networks which are sometimes constrained to be in a state incompatible to the value they would like to be, indeed, a binary weight will be necessarily -1 or 1 and there is no intermediate value, the gap between these two values is significant, adding a simple representation value increases drastically the expressivity and therefore the resulting performances can also be improved. In Chapter 6, we will briefly present the circuit used to perform a low-energy TNN product operation.

This comparison was strictly in terms of the amount of required memory. However, BNNs have further advantages when taking into account the arithmetic operations that are performed in a neural network. Doing an operation between two bits is much less complex than an 8-bit operation. The advantage of Binarized Neural Networks is that the mathematical operation of multiplication becomes a simple XNOR logic gate and then the additions required to perform the scalar product between the activation vector and the weight vector become a simple bit count. In a classical neural network, the operations involved are the same, multiplication and addition, but the number of coding bits is much higher than in the binary case involving both greater circuit complexity and greater associated power consumption. In Chapter 4, based on practical design, we were able to present a comparison between binarized neural network and classical neural network in terms of energy consumption. The other consideration is the data movement, even if the calculation is relatively localized in a neural network, the data movement can have a significant impact on energy consumption, by opting for a Binarized Neural Network, only one wire is needed to transmit the information between two neurons, whereas in a case of 8-bit coding, at least 8 times more would be needed.

These considerations strongly motivate our study on the implementation of the binarized neural network with RRAM, presented in chapter 4.

### 2.2.5 Specificities of Binarized Neural Network

An important remark about the training of Binarized Neural Network, is that it usually requires more iterations than for classical neural network. The reason for this observation was understood recently. When we train a classic neural network, we have two phases, the first one that will learn features very quickly and the second one of compression that will learn to forget the useless features [157]. In binarized neural networks, as we have less representation capacity,

the binary network performs both phases at the same time! It is not able to learn a lot of features at the beginning, so the choice is partial, it will forget the uninteresting features and then build new ones at the same time [185].

We have just mentioned Binarized Neural Networks for feedforward architectures, but this approach also works for a wide range of neural networks. In Chapter 4, the results of Convolutional Binarized Neural Networks will be described, but some authors have also been able to train binarized autoencoders [186] / restricted Boltzmann machines and recurrent neural networks. For recurrent architectures, like Long Short-Term Memory neural network (LSTM) [164], weights can be easily binarized but not activations which need a sufficiently precise probabilistic coding. In order to have probabilities at the output of layers, we have to rescale the activation functions. Otherwise, they will saturate at maximum and minimum values, the sigmoid functions will be re-scaled with a small value parameter $\alpha$ that can be learned by backpropagation or tuned as a hyper-parameter.

The special feature of restricted Boltzmann machines is that they have a local learning rule, i.e. there is no need to transmit information from the output of the neural network to the input. This makes them particularly interesting candidates for the implementation of on chip learning. These different architectures will not be presented in this thesis as the principle is very similar to feedforward neural networks, so we mainly focused on the simple hardware implementation of feedforward Binarized Neural Networks.

## 2.3 Connection between Bayesian Reasoning and Neural Network

Today, deep learning and probabilistic reasoning are the two main areas of research in artificial intelligence. Deep learning has the ability to create very complex and efficient models for image classification[163], time series [187], physics simulations [188] and an incredible range of other applications [189, 190]. On the other hand, the probabilistic approach to reasoning makes it possible to integrate information in a progressive manner, to infer on a set of parameters and to make decisions. Probabilistic reasoning makes possible to work with uncertainty, and also allows to make a choice of models. Each of these two areas has its own advantages and drawbacks, a list of which is presented in the Table 2.1 reproduced from [191].

Neural networks feature a very large number of parameters. At each layer of a deep network, there can be hundreds of thousands of parameters, so a complete network can feature millions of parameters or even hundreds of billions of parameters for the most complex models [192]. This large number of parameters allows deep learning to create extremely complex and very non-linear models. On the other hand, Bayesian models are very limited in the number of parameters, the inference being made by integrating on all possible parameter configurations (see Equation 2.5), when there are more than a few parameters, we quickly end up with a prob-

lem complicated to solve in a reasonable amount of time. In Bayesian models, the sampling weight vector space is difficult to simulate whereas neural networks do not need it. For neural networks, stochastic gradient descent and inference are conceptually very simple.

| **Deep Learning** | **Bayesian Reasoning** |
|---|---|
| Rich non-linear models for classification and sequence prediction. | Mainly conjugate and linear models |
| Scalable learning using stochastic approximation and conceptually simple. | Potentially intractable inference, computationally expensive or long simulation time. |
| Easily composable with other gradient-based methods. | Unified framework for model building, inference, prediction and decision making |
| Only point estimates. | Explicit accounting for uncertainty and variability of outcomes |
| Hard to score models, do selection and complexity penalisation. | Robust to overfitting; tools for model selection and compositiong |

Table 2.1: Advantages and drawbacks of the two main areas of machine learning research: Deep Learning and Bayesian Reasoning. Reproduced from [191].

Concerning the mixture of models, probabilistic reasoning is very powerful as it provides a unified framework for inference, prediction, and decision making. Where deep learning lacks efficiency is in estimating the uncertainty of its prediction. It is difficult to score models, to do selection and complexity penalization. This difficulty emerges from the fact that neural networks optimize the maximum likelihood (they only have point estimates) and thus estimate the model on a single set of parameters. This aspect makes the big difference with Bayesian reasoning, which explicitly takes into account uncertainty, variability, is robust to overfitting, and is a perfect tool for model selection and model composition.

Even though neural networks have revolutionized artificial intelligence, they still have a lot of limitations. They are very data-intensive: without several million examples, some models are not capable of good performance. They are also very demanding in calculations, both for training and for the deployment of solutions in embedded systems. This thesis tries to answer this question: how to reduce the energy consumption related to neural networks for inference in Chapter 4 and for learning in Chapter 6. But it should not be forgotten that they are not very efficient to represent uncertainty, can be fooled by adversarial examples [193], are very tricky to optimize: the choice of architecture, the learning procedure, the initialization of parameters require expert knowledge. For all these reasons and because they are often seen as uninterpretable black boxes that lack transparency and are difficult to trust, we are interested in bringing them closer to the Bayesian models.

Unfortunately, for Bayesian models, and for all graphical models in general, it is difficult to generate their architectures automatically. In the example in section 2.1, we described a very

simple example of the child and the random draw of a ball in a box. This example features only one hidden variable representing the probability of having a red or white ball. In fact, we unconsciously created our model from prior information. In some cases, we don't know how many variables exist and how they influence each other. As this example, early graphical models used experts to define the graph structure where nodes have meanings. And the conditional probabilities are typically stored in conditional probabilities tables made by hand. The graphs were initially quite simple with sparse connections. Most of the research focused on doing correct inference and not learning.

Research to link neural network models to Bayesian models has now become a broad area of research. To go further in this study, Appendix A provides two principles linking neural networks to Bayesian models. Similarly, in Appendix B, we present some very recent and unpublished work on the use of Bayesian inference to interpret the last layer of a binary neural network.

## 2.4 The use of emerging memory devices for Neural Network and Bayesian models

As mentioned in the previous chapter, memory devices are the key to the low-power implementation of AI electronic circuits. Memory devices have various characteristics depending on the technology used. Magnetic memories (MRAM) are intrinsically binary and cannot encode analog values. RRAMs are very noisy during programming, so it can happen that when a pulse is applied to increase the conductance of the component, it eventually decreases. PCMs are subject to thermal drift, their conductance value is not stable over time, making it necessary to reprogram them from time to time. The choice of technology therefore depends very strongly on the target application.

In this thesis, we have mainly studied a specific technology, the resistive memories (RRAM) to implement the Bayesian machine (Chapter 3), the implementation of the inference of a Binarized Neural Network (Chapter 4), and for the learning process of the BNN (Chapter 6). In Chapter 5, we also studied the use of stochastic memory devices for the implementation of neuromorphic algorithms using the intrinsic stochastic characteristics of MRAM devices. The use of other technologies for the implementation of Bayesian circuitry and Binarized Neural Networks using the approaches proposed in this thesis is entirely realizable. Concerning the learning of the Binarized Neural Network and the studies within chapter 5, we had to focus on the characteristics of a specific technology and to face its defects, these studies exploit (and sometimes suffer from) the very specific features of the devices. In addition to the collaborations that we have, which allow us to have access to the fabrication of chip using this technology, the choice was made to work with an RRAM technology because it is a simple structure, low cost, mature and back-end-of-line compatible.

The first hardware implementation is a Bayesian model, which is a probabilistic model that performs operations on probabilities. The mathematical operations of such a system are very simple, under the conditions we fixed, i.e. assuming conditional independence, the mathematical operations are simply products of probabilities. The circuit implementation in Chapter 3 shows how to perform these probability products using stochastic computing. Each probability is coded in conventional binary fixed point, the circuit generates a sequence of random bits proportional to the value stored in memory, and a stochastic probability product is performed by a simple AND logic gate. The idea of this hardware implementation is to associate a memory matrix to each set of likelihood, each cell will therefore associate one single probability corresponding to one specific observation. Errors in such a system can have an impact on the final result since an error on the most significant bit impacts the totality of the probability and thus the final result. The implementation of such systems cannot have a large number of errors occurring in the memory devices.

In Chapter 4, we will see that the implementation of Binarized Neural Networks is different, each synapse is associated with a value, a 1 or a -1 which corresponds to the value by which the binary input will be multiplied. An error will have a very different impact on the final result compared to a system coding the values in a classical way. The classical method of implementing neural networks to perform multiplication operations between the input vector and the weight vector is the use of Kirchhoff's laws. The use of this principle is very interesting conceptually but can lead to some difficulties and hidden costs in the implementation especially because of ADCs [194] or current-voltage converters [195]. For these reasons in Chapter 4 we do differently using fully digital implementation.

In Chapter 5, we will see how to go beyond the use of memory devices as simple binary storage values, since we will see also how to use all the complexity of the devices and especially their stochastic behaviour to perform learning, inspired by the population coding of neurons in the brain. But this work is not the only one that seeks to go beyond the classic use of memories. The approach that we will propose in Chapter 5, is relatively different from the two other subjects addressed in this thesis that can be linked together. Nevertheless, the foundations remain the same, and other works propose to use the same type of approach to implement algorithms that bring neural networks and Bayesian models closer together.

Finally Chapter 6, we explore all the opportunities for using the analog characteristics of RRAM type devices. Increasing the number of values that can be encoded per synapse allows to increase the performance of neural networks, but also to obtain new features, such as long term memory. Finally, the use of these analog devices paves the way for extensive on-chip learning.

# Chapter 3

# Hardware implementation of Bayesian Machine

> The human brain is still far from being reproduced by machines. What they lack is the ability to formulate scientific theories, as toddlers do. Machines remain highly specialized, they are not able to think in multiple fields and gather their knowledge in a symbolic form. But things are changing very quickly, and I wouldn't be surprised if, in ten or fifteen years, much more advanced machines emerge. All the key elements of human learning [...] (attention, active engagement, error feedback, consolidation) are being modelled by artificial intelligence.
> *(translated from French)*
>
> Stanislas DEHAENE

*"One of the solutions for low-energy inference involves a high-level inspiration of the brain, around the principle of probabilistic reasoning. To work with probabilistic reasoning, we mainly use Bayes' theorem, which allows us to deduce a probability prediction from an observation. The purpose of this chapter is to present a hardware implementation of a circuit that implements this Bayesian reasoning."*

THE BIOLOGICAL inspiration from the brain can take place at several levels. Since the pioneering work of Carver Mead, researchers are naturally inclined to replicate the underlying mechanisms: mainly synapses and neurons. By contrast, another approach is to look at how reasoning is done, by trying to interpret the characteristics of human reasoning. Many neuroscientists and cognitive psychologists study the brain by doing experiments on children's and adults' reasoning, by showing them images/objects but also on adult subjects. As reported in the previous chapter, many of these experiments suggest that the brain might be doing a form of Bayesian reasoning. We also discussed the links with neural networks in Appendix A and Appendix B.

My contribution is mainly focused on the design of a hardware accelerator for Bayesian inference. I was able to implement a hybrid CMOS/RRAM chip that allows "In-Memory-Computing" with the innovative paradigm of Bayesian reasoning. The theoretical notions presented here have been extensively studied in the literature.

In this chapter, we will therefore present these notions, showing the main equations of Bayesian inference and study the naive hypothesis, which is the working hypothesis for our hardware implementation. Then we will detail the hardware implementation using stochastic computation and finally, we will detail the complete architecture of our taped out.

I designed the full architecture of the circuit implementing Bayesian inference from the theoretical level to the final taped out. The layout design was largely conducted by Kamel-Eddine Harabi.

## 3.1 Theoretical Bayes Inference

Bayesian programming is a discipline in its entirety and it would be quite presumptuous to pretend to be able to summarize this discipline in a few pages! To have a global understanding of it, to fully apprehend its challenges, and to see the wide variety of applications that can use it, the *"Bayesian Programming"* [151] and *"Probabilistic Graphical Models"* [149] books are a gold mine of information. In these few pages, we will therefore limit ourselves to considering probabilistic inference. In the previous chapter, we introduced the use of probabilities to describe machine learning problems. The whole approach of probabilistic models is based on the Bayes theorem (Equation 3.1) :

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y \in Y} P(x,y)}. \tag{3.1}$$

The calculation of Bayesian inference seeks to determine the posterior distribution $P(y|x)$ of a set of hypothesis $y$ from observations $x$. As mentioned in the previous Chapter 2. It is decomposed into three factors, the prior distribution $P(y)$, which corresponds to the initial probability of the hypothesis before any observation has been obtained, the conditional probability $P(x|y)$, which corresponds to the probability of the observation given the hypothesis, and the marginal likelihood $\sum_{y \in Y} P(x,y)$, which corresponds to the sum of the probabilities

over all hypotheses.

From one observation $x$, we try to determine information on $y$ using Bayes' rule. In Equation 3.1, the variables $X$ and $Y$ are continuous variables that can take their value in the set of real numbers $\mathbb{R}$. As a matter of fact, for the calculation of probabilities, it is not useful to know the value of the variables, it is sufficient to associate a probability at each value. Most models can be described using discrete variables, but when this is not the case, the set of variables will be discretized to perform the calculation on a computer. Most of the time, the performed discretization seeks to approximate simple functions like Gaussian distributions. For this type of computation, some Application Specific Integrated Circuit (ASIC) can be used. In this chapter, we will describe an ASIC architecture using non-volatile memory devices focusing on computation from discrete variables. Being limited in the amount of memory that can be allocated, these discrete variables may take a relatively small number of different values.

In a wide range of cases, it is not the full posterior probability over all possible input that we try to estimate, but rather from an observation $x$, what are the relative probabilities of the different hypotheses $y$. The advantage of such an approach is that the marginal probability does not depend on the hypothesis $y$, as it is the same for each one of the hypotheses, it can be ignored to infer the best hypothesis given an input. We then have the following proportionality equations:

$$P(y|x) \propto P(x|y)P(y). \tag{3.2}$$

In the various equations presented above, only one variable $x$ influencing the distribution was considered. It is according to the value of this unique variable that we determine the posterior probability on the hypotheses $y$. In more real-life situations, there is not just one variable that can influence the posterior distribution, but several. In the case of a multivariate problem, the Bayes' theorem becomes:

$$P(y|x_0, x_1, \cdots x_n) = \frac{P(x_0, x_1, \cdots, x_n|Y = y)P(y)}{P(x_0, x_1, \cdots, x_n)} = \frac{P(\bigcap_{i\in I} x_i|Y = y)P(y)}{P(\bigcap_{i\in I} x_i)}. \tag{3.3}$$

As for Equations 3.2, the marginal probability does not depend on the hypothesis, so when looking at an observation set to determine which hypothesis is the most likely, the term can be removed. Proportionality equations for multivariate problems consequently become:

$$P(y|\bigcap_{i\in I} x_i) \propto P(\bigcap_{i\in I} x_i|Y = y)P(y). \tag{3.4}$$

The most important term in the previous Equation 3.4 is the likelihood factor $P(\bigcap_{i\in I} x_i|Y = y)$, which contain all the information about the different observed variables, the prior corresponding only to the knowledge about the hypothesis before making the observation. For a given variable $x_i$, different observations can be made. For instance, for a six-sided dice, there are six possible observations. If we use several variables, e.g. four dices, and with the same

number of observations, the dimension of the likelihood term is $6^4 = 1296$. For a relatively small number of variables, it is possible to count all these cases and assign a probability to them, but when the number of variables becomes large this is no longer possible.

When all the variables depend on each other, (equation 3.3) the model is impractical, as the amount of available memory is limited. As a consequence, when building a Bayesian model, it is extremely important to use the independence between variables and more particularly conditional independence to reduce the dimension of the likelihood term.

Two variables $a$ and $b$ are independent from each other if and only if $P(a \cap b) = P(a) \cdot P(b)$. From the product rule, this condition is also equivalent to $P(a|b) = P(a)$ and $P(b|a) = P(b)$.

Similarly, two variables $a$ and $b$ are conditionally independent to a third one, $c$, if and only if $P(a \cap b|c) = P(a|c) \cdot P(b|c)$. From the product rule, this condition is also equivalent to $P(a|b \cap c) = P(a|c)$ and $P(b|a \cap c) = P(b|c)$.

Be aware that strict independence does not imply conditional independence, neither conditional independence does imply strict independence. We have indeed these following two relationships: $P(a,b) = P(a)P(b) = P(a|c)P(c)P(b|c)P(c)$ if $a$ is strictly independent from $b$ and $P(a,b) = P(a,b|c)P(c) = P(a|c)P(b|c)P(c)$ if $a$ is independent from $b$ given $c$. Therefore, even if two variables are independent in the prior, they can be dependent to a third variable which they both influence.

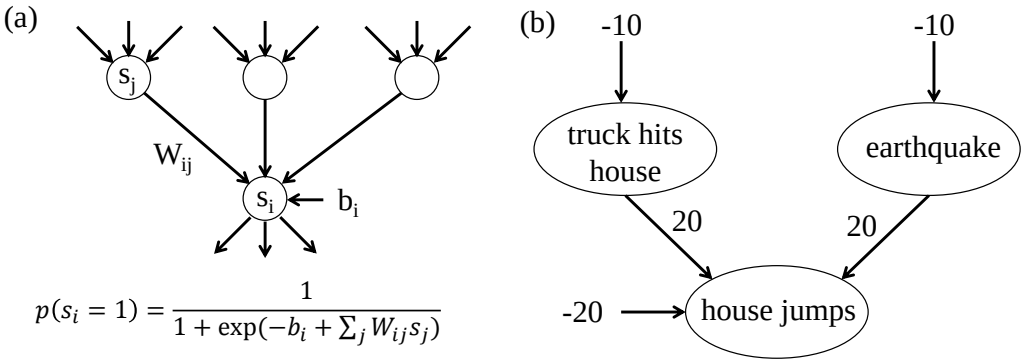



Figure 3.1: (a) Sigmoid belief net illustration, this probabilistic graphical model is a directed graph where the probability of the variable is given as function of the parents state. (b) Example of a particular belief network, the fact that a house jumps is very unlikely unless a truck hits the house or there is an earthquake. This example was used by Geoffrey Hinton in [180] to illustrate the "Explaining away" phenomena that lead to difficulties to train sigmoid belief nets one layer at a time.

This phenomena, called "explaining away" [196], is illustrate in the classic example of a simple belief network of Figure 3.1. The example in Figure 3.1 (b) shows three binary variables taking value 0 or 1, two hidden causes $a$ = *"truck hits house"* and $b$ = *"earthquake"*, and an effect that they can both influence $c$ = *"the house jumps"*. The network presented here is a belief network where the two variables $a$ and $b$ are independent but dependent given $c$. From

the model we obtain the following probabilities of the variables $a$, $b$ and $c$:

$$P(a) = \frac{1}{1+exp(-10)} \;\&\; P(b) = \frac{1}{1+exp(-10)} \tag{3.5}$$

$$P(c|a,b) = \frac{1}{1+exp(-20+20(s_a+s_b))} \tag{3.6}$$

The posterior distribution on the hidden variables is obtained from the observation on $c$. The mathematical relation 3.7 below is obtained from independence between $a$ and $b$, and from the equations 3.5 and 3.6.

$$\begin{aligned} P(a,b|c) =& \frac{P(c|a,b)P(a,b)}{P(c)} \\ =& \frac{P(c|a,b)P(a,b)}{P(c|a,b)P(a,b)+P(c|\overline{a},b)P(\overline{a},b)+P(c|a,\overline{b})P(a,\overline{b})+P(c|\overline{a},\overline{b})P(\overline{a},\overline{b})} \\ =& \frac{P(c|a,b)P(a)P(b)}{P(c|a,b)P(a)P(b)+P(c|\overline{a},b)P(\overline{a})P(b)+P(c|a,\overline{b})P(a)P(\overline{b})+P(c|\overline{a},\overline{b})P(\overline{a})P(\overline{b})} \end{aligned} \tag{3.7}$$

From the different coefficients presented in Figure 3.1 (b), we obtain the posterior probabilities on $a$ and $b$ given $c$ as follows:

| |
|---|
| $P(a=0,b=0\|c=1)=0.00005$ |
| $P(a=0,b=1\|c=1)=0.49995$ |
| $P(a=1,b=0\|c=1)=0.49995$ |
| $P(a=1,b=1\|c=1)=0.00005$ |

If the house jumps, it is about 50% likely that it is due to the earthquake and about 50% that it is due to a truck hitting the house. But it is highly unlikely that this is both reasons at the same time or neither at all. This means that if it is learned that there was an earthquake, this information greatly reduces the probability that the house jumps because of a truck. In this very common configuration, it is therefore important to keep each of the conditional probabilities given $c$, otherwise, a sensor measuring several factors would be useless.

As we want to build dedicated hardware that is energy efficient and consequently with a small amount of memory, we will try to build Bayesian models that reduce the number of parameters. Thus, we will try to avoid working with observational variables depending on several hidden causes. The type of model that will be preferred is therefore the opposite of the one presented in Figure 3.1. It will be composed of several observable variables for the one hidden variable. The goal is therefore to work on the conditional independence between the observa-

tions with respect to the hidden variable we want to determine.

## 3.2 Naive Bayes Assumption

The model can be constructed in such a way that only conditionally independent observations are taken into account. The advantage of such a technique is that it allows information from different sensors to be merged to estimate the state of a given phenomenon. For example, in Figure 3.2 (a), we try to estimate the probability corresponding to each stat e $k$ of the variable $s$ from observations $\omega$.

The statement of the problem is the following: we want to obtain information on a given phenomenon, this phenomenon influences directly sensors, which state can be read, and from this reading, we try to estimate the probability of each state of the phenomenon.

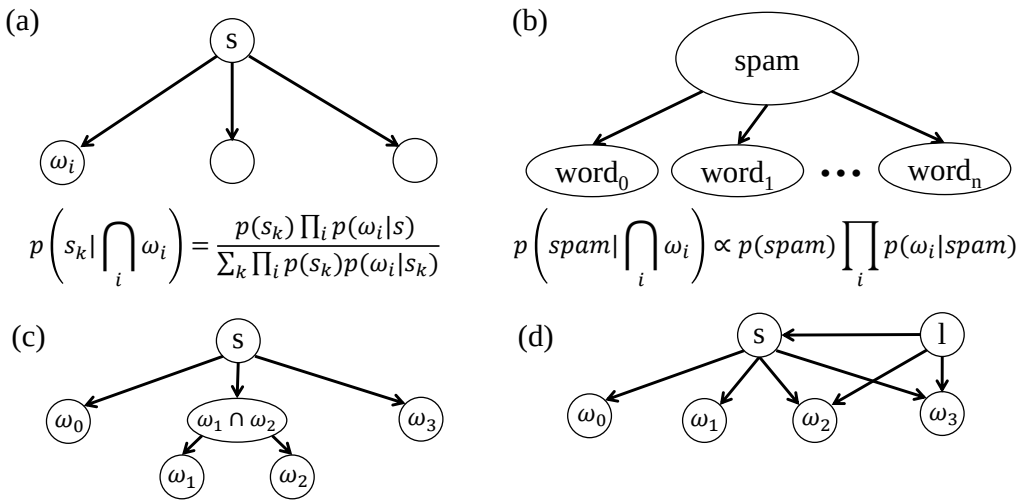



Figure 3.2: (a) "Naive Bayes Model", all the observations $\omega_i$ are conditionally independent one to another given the hidden phenomena $s$. The equation below describes the probability of the hidden phenomena $s$ given the observations $\omega$, as a function of the prior knowledge of the hidden phenomena $s$ and the conditional probability of the observation $\omega$ given $s$. This kind of model are the easiest to implement Bayesian Inference, and will be the one that our hardware design implement. (b) A simple example of Naïve Bayes Inference to detect if an email is a spam or not. The posterior probability that an email is a spam or not is proportional to the product of the conditional probabilities. (c) Model in which we have added a dependency between the observations $\omega_1$ and $\omega_2$. (d) Model in which we have added a dependency between the observations $s$, $\omega_2$ and $\omega_3$ to a hidden phenomenon $l$.

Obviously, in reality, the different observations are not always conditionally independent given the hidden variable. The first case is when two observations may not be conditionally independent of each other given the hidden variable 3.2 (c). In that case, the conditional inde-

pendence simplification leads to error compared to the exact Bayesian inference.

When we assume that all the observations are conditionally independent given the hidden variable we call it *"the naive assumption"*. This hypothesis is very strong, but it turns out that it is particularly effective in a very large number of cases [197].

The other issue mentioned in the previous section is the phenomenon of explaining away when observations are linked to another hidden variable. In the real world, there are always factors other than the searched variable that is correlated with the sensor readings. It is important to decide whether or not these correlations can be neglected. If modeling all these correlations requires a very large amount of computation and resources, while they have little impact on the final result, there is no point in modeling them.

In the spam detection example of Figure 3.2 (b) knowing whether a text is a spam or not is far from being enough to explain all the correlations between words in the text. This model neglects both the direct correlations between variables that can exist as 3.2 (c) depicted or the correlations of the observations to another hidden variables $t$, that we do not model as 3.2 (d) depicted. Despite these very strong simplifications, the results of the *"Naive Bayes Model"* turns out to be very efficient for this example of spam classifications [198].

However, in some cases, the naive assumption is not sufficient and leads to large errors compared to the exact Bayesian inference. When there are strong dependencies between variables, (Figure 3.2 (c)) it is possible to use the intersect of these variables making the number of observations higher. Bayesian inference with the naive hypothesis is then still possible by not taking the variables $\omega_1$ and $\omega_2$ independently anymore, but by taking as observable variable the intersect of the two. This approach is relatively simple to implement, the mathematical operations remain the same, as it is sufficient to add more memory to the system.

When there is a hidden variable that influences only the observations and not the search phenomenon $s$, it is possible to perform Naive Bayesian inference. It only requires to relax the constraints on the naive hypothesis exactly as it is done in Figure 3.2 (c), no longer assuming the conditional independence of single variables but that of tuples of variables given the phenomenon.

On the other hand, when there is a hidden variable that also influences the hidden search phenomenon, the problem is more complicated. In the example in Figure 3.2 (d) a hidden phenomenon $l$ influences both observations $\omega_2$, $\omega_3$ and the variable $s$. In this configuration multiple phenomena can explain a specific observation, so the naive Bayes assumption is not anymore valid.

Let us take the following example: the variable $s$ is a binary variable corresponding to the presence or not of a storm and the four observations $\omega_0$, $\omega_1$, $\omega_2$ and $\omega_3$ are three binary wind speed sensors, taking the values 0 for weak wind and 1 for strong wind, the hidden variable $l$ could correspond to the presence of lightning during the storm or not. Unfortunately, when lightning strikes the $\omega_2$ and $\omega_3$ sensors are defective and give mostly random responses. Knowing the hidden variable *"lightning"* would be very useful, not because it would allow determin-

ing directly if there is a storm but especially to detect if the observations $\omega_2$ and $\omega_3$ can be used to detect if there is a storm or not. In some problems, it is essential to determine these hidden variables. For this purpose, there is a field of statistics that works to identify latent variables from data called "Latent Class Analysis" [199]. When a latent variable $t$ is identified, it is mandatory to model the joint distribution $p(s,t|\omega_0,\omega_1,\omega_2,\omega_3)$, looking at the two variables $s$ and $t$ at the same time and not only $p(s|\omega_0,\omega_1,\omega_2,\omega_3)$, looking at the only variable $s$ .

This chapter will focus on the hardware implementation of naive Bayesian inference. When naive inference is not possible, we will model the problem from the hidden variables using joint probabilities –increasing significantly the number of memories required–. Nevertheless, this naive hypothesis is relatively efficient for a wide variety of applications. It is indeed one of the most efficient methods for sensor fusion [200]. The naive hypothesis of Bayesian inference allows data from multiple sources to be combined meaningfully.

## 3.3 Computing Bayesian Inference with Stochastic Computing

In classical computers, the mathematical operations are performed with high precision, in a fully deterministic manner. Given the digital nature of computers, calculations with real values are not exact, but what we mean by exact calculation is that once real value data is converted into digital data, the mathematical operations on this converted digital data are exact. If the discretization of the analog data was done accurately, the digital result at the output of the digital computer will be very close to the true real calculation.

The classical coding scheme for precise calculation is 64-bits (full precision) and 32-bits (single precision) floating-point. In CPUs and GPUs, the hardware implementation optimized the speed of the mathematical operations leading to very intensive energy consumption, especially for multiplication. Recently the 16-bits (half precision) floating-point coding scheme has been shown to be useful for the training of neural network [201]. For this reason, the new GPUs optimized for deep learning now include half-precision coding. This conversion to half-precision is sometimes very brutal for most applications and requires rethinking program code. Nevertheless, some tools and techniques are available to perform this conversion [202].

It is possible to go even further in floating-point bit reduction by using "mini float" coding with less than 16 coding bits [203]. It makes hardware implementations even simpler. This is a coding scheme to take into account when working with data that may be imprecise but requires a wide range of values.

For many applications, the exact calculation is not necessary, and rather acceptable results can be obtained by using such extensive approximations. Moreover, by accepting to not perform exact calculations, it is possible to relax the constraints of digital circuits by making them faster and less energy-consuming [204] [205].

In recent years, this field of *"approximate computing"* has gained renewed interest in research. The goal is to analyze the trade-off between the accuracy, the energy efficiency, and speed of the calculation. In general, when applications do not require extraordinary precision, it is possible to work with fixed-point coding with a relatively small number of bits. Digital operations become simpler and lower in power-consuming. But it is possible to go further, by using approximate operators for the calculation.

That being said, in the literature [206], a detailed comparison between fixed-point arithmetic architecture and approximate operators shows that even if theoretically, the use of approximate operators seems competitive, the advantages can be lost at the application level. An approximate operator does not reduce the size of the output data and therefore has no positive impact on the operator that follows. When operating a succession of approximated operators, the result can be quite different from the expected output. Nevertheless, a system able to overcome these successive approximations could be a method of reviving this approach.

Rethinking data coding is very important to reduce the energy consumption of microchips. Most machine learning algorithms (e.g. neural networks) are based on multiplication and accumulation (MAC) operations. The academic and industrial research for a simplified coding scheme and associated dedicated hardware to optimize these operations is very intensive and very competitive. On the other hand, for the Bayesian approach, hardware optimization is not yet highly developed. Bayesian Inference mainly involves the successive product of probabilities.

In this succession of products necessary for Bayesian computation, the main potential influences on energy consumption are: the access to the probabilities data, the multiplication operations between probabilities, and the data movement. It is given these challenges that we decided to use stochastic computing, whose main qualities will be detailed in the following paragraphs.

Figure 3.3 (a) and (b) present a classical architecture to perform respectively the addition and the multiplication operations of floating-point coding [207]. Each small building block represents a consequent number of logic gates and thus a large number of transistors in the final design. This remark should be balanced by the fact that the architectural complexity of a floating-point multiplier is relatively simple when working with few mantissa bits (it is dominated by the fixed point addition on the exponent bits).

By contrast, the stochastic computing approach consists in coding the data in the form of a bitstream – a sequence of successive bits of 1s and 0s whose ratio of 1 to the total size of the bitstream corresponds to the encoded data between 0 and 1–. By construction, the coded value can only be between 0 and 1. All the data are coded this way; it is a different paradigm since, for instance, the addition presented in Figure 3.3 (c) does not sum the two values but performs a weighted sum with another factor also coded in the form of a bitstream. In Figure 3.3 (c) and (d), we can see that the implementation of the calculation is exceptionally simple. The multiplication operation in stochastic computing is therefore performed by a simple AND

logic gate, the output precision of this multiplication depending only on the averaging time. When a high degree of precision is required, the time needed to obtain it becomes very important. To get as good precision as floating-point coding the time needed is considerable and the energy required is higher than performing floating-point multiplication. On the other hand, the implementation area remains unchanged.

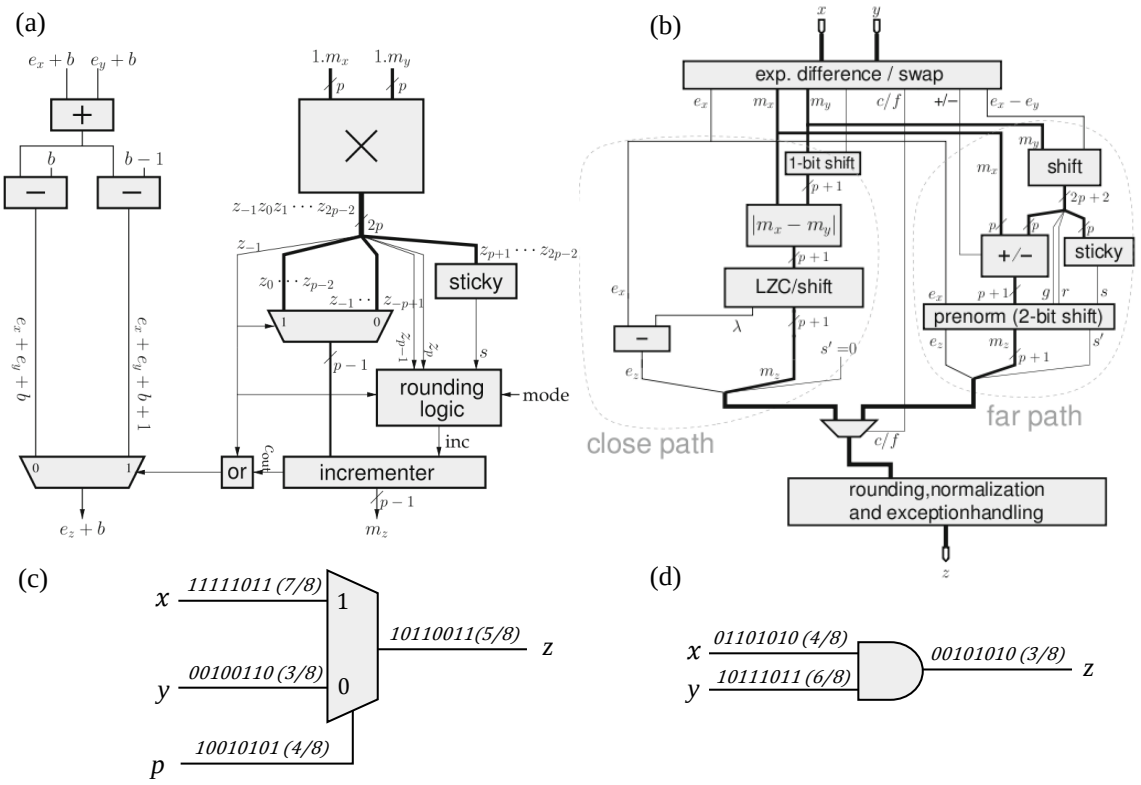
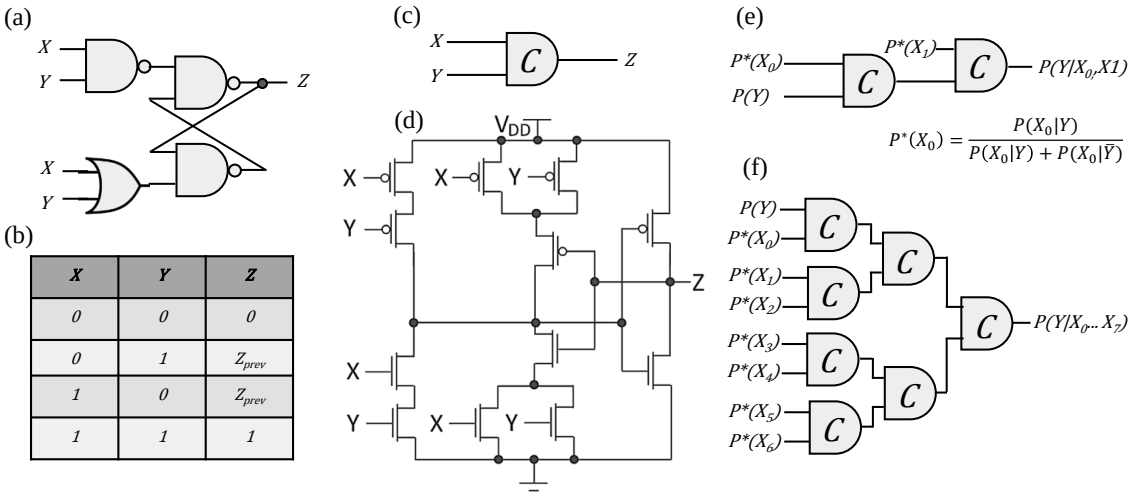



Figure 3.3: (a) Basic architecture of a floating-point adder. (b) Basic architecture of a floating-point multiplier. Both taken from [207]. (c) A multiplexer can be used to perform the sum in stochastic computing, the output is : $z = p\text{x} + (1-p)\text{y}$ if $p = 1/2 : z = (x+y)/2$. (d) A logical AND gate can perform directly the stochastic multiplication between two bit-streams.

In the idea of building an *"in memory computing"* circuit, where the computation and the memory are co-located, the possibility to replicate the multiplication circuit with stochastic computing gives the ability to perform a large number of parallel computations at the same time. This is in GPUs with high precision operators, where the arithmetic and logical units (ALU) are replicated for each core. But the size and power consumption of a hybrid CMOS/Non-Volatile-Memory (NVM) ASIC performing floating-point coded probability product operations would be way too high in comparison to the memory units.

Another advantage of stochastic computing concerns the data movement. The number of clock cycles required will be very large compared to a more classical calculation approach.

However, the fact that the data is coded on a single wire allows us to significantly reduce the data size and therefore the data movement. The other advantage is the possibility of coding a very large number of probability values, each wire corresponding to a probability value: the amount of data that can be processed simultaneously is considerable. This is one of the reasons that make the brain very powerful, transmitting only one specific signal per axon.




| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $Z_{prev}$ |
| 1 | 0 | $Z_{prev}$ |
| 1 | 1 | 1 |

Figure 3.4: (a)Design of a C-Element circuit using standard cell. (b) Muller C-Element Truth table. (c) Symbol of the C-element. (d) Efficient implementation of the C-Element with CMOS transistors. (e) Two input cascaded C-element give the output $P(y|X_0, X_1)$. (f) Tree structure of C-element give the output of the Bayesian Inference for seven inputs $P(y|X_0, ...X_6)$.

In stochastic computing, the key elements of the operations concern the coherence of the bit-streams. To obtain accurate results, the input bit-streams of the logic gates must be independent and uncorrelated otherwise the output will not correspond to the expected result.

Stochastic computing seems to be particularly well suited to perform calculations based on probabilities. For instance, the basic equation of Bayes' theorem can be calculated from the basic Müller C-elements logical circuit presented in Figure 3.4 (c) taken from [198].

From the truth table Figure 3.4 (b) and for uncorrelated input signals, the output probability $P(Z)$ of a C-Element is:

$$P(Z) = \frac{P(X)P(Y)}{P(X)P(Y) + (1 - P(X))(1 - P(Y)).} \tag{3.8}$$

By replacing P(X) by $P^{\star}(X) = \frac{P(X|Y)}{P(X|Y)+P(X|\overline{Y})}$, we obtain the Bayesian Inference Equation that can be directly compute by the C-Element logic gate:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X|Y)P(Y) + P(X|\overline{Y})P(\overline{Y})} \tag{3.9}$$

By adding different inputs, still with the same type of probability coding P*, it is possible to

perform a Bayesian inference by stochastic computing by cascading the C-elements logic gates if the inputs $X_i$ are conditionally independent given $Y$. As shown in Figure 3.4 (e), the Bayesian inference $P(Y|X0,X1)$ is calculated from two C-Elements gates :

$$\begin{aligned} P(Y|X_0,X_1) &= \frac{P(X_1|Y,X_0)P(Y|X_0)}{P(X_1|Y,X_0)P(Y|X_0)+P(X_1|\overline{Y},X_0)P(\overline{Y}|X_0)} \\ &= \frac{P^\star(X_1)P(Y|X_0)}{P^\star(X_1)P(Y|X_0)+(1-P^\star(X_1))(1-P(Y|X_0))}. \end{aligned} \tag{3.10}$$

In general, it is possible to extend the Bayesian inference with a very large number of conditionally independent variables given Y by rearranging the succession of logic gates C-elements in the tree form.

However, even if the logical gate C-element is very interesting as it allows to directly perform the mathematical operation of the Bayes' theorem, it can only work for binary variables, i.e., variables that can only take two values. When we want to fuse sensors, it is more convenient to have sensors that provide more information, with more possibilities, i.e., the values that the variables can take are greater than two. Moreover, when two observations are correlated, we explained earlier that the joint probabilities have to be represented, which is the same as if we have several possible values for a variable.

Taking into account the limitations of pure C-Element architecture, a new Bayesian Machine has been developed [208] [209] and studied for source localisation [210] [211]. This architecture will be presented in the next section 3.4, then a hardware implementation with RRAM technology will be presented 3.6.

## 3.4 The Bayesian Machine : hardware description

### 3.4.1 General overview of the Architecture

The hardware implementation of the Bayesian machine aims at implementing the calculation of the probability of an event $y$ from observations $\omega$ for discrete variables. The working hypothesis of this machine is that it performs naive Bayesian inference. i.e. we will work with observations $\omega$ that are independent given $y$. If two observations are not independent given $y$, the choice will be to model the joint distribution of these observations. Similarly, if there is more than one hidden variable, we will use their joint distribution. As in Figure 3.2 (a), under naive assumption, the equation to compute is:

$$P(y|\bigcap_{i\in I}\omega_i) = \frac{P(y)\prod_{i\in I}P(\omega_i|y)}{\sum_{y\in Y}\prod_{i\in I}P(\omega_i|y)P(y)}. \tag{3.11}$$

As mention in the section 1.1, when looking at an observation set to determine which hy-

pothesis is the most likely, the term at the denominator can removed:

$$P(y|\bigcap_{i\in I}\omega_i) \propto P(y)\prod_{i\in I}P(\omega_i|y). \tag{3.12}$$

For more clarity, the equation above can be written in the form:

$$P(y|\omega_0,\cdots\omega_n) \propto P(y)P(\omega_0|y)\cdots P(\omega_n|y). \tag{3.13}$$

This Bayesian machine uses stochastic bit-streams. Each probability $P(\omega_i|y)$ is a bitstream and a logical $AND$ gate will be used to perform the multiplication. It is possible to perform a succession of stochastic multiplication with a cascade of logical AND gates.

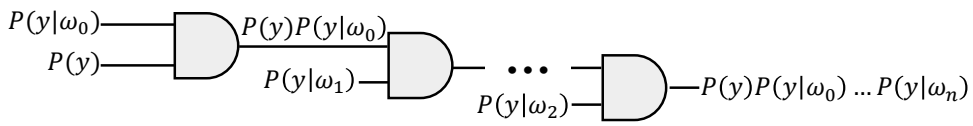



Figure 3.5: Succession of $AND$ gate that perform for one configuration of $y$ the succesive multiplications of the likelihood $P(\omega_i)$. This circuit is replicated as many as the possible values for y.

In Figure 3.5 the stochastic multiplication between the bitstreams $P(y)$ and all the probabilities $P(y|\omega_i)$ is performed by a succession of $AND$ gates. It is important to notice that here, only one configuration of $y$ is calculated. For instance, a thermal sensor will be able to give different values and not only give a binary value 0 or 1. These likelihood values can be determined beforehand during the learning process, then at the Bayesian inference we will look for these values and multiply them with each other according to Equation 3.12 using stochastic computing. Imagine that $y$ can take 16 values, we will need 16 times the circuit presented in Figure 3.5 to compute the complete posterior distribution.

An example of the complete Bayesian machine is shown in Figure 3.6 for 4 possibles values of $y$. Each row of the Bayesian machine then performs the product between the prior $P(y_k)$ of one of the possible values of $y_k$ with the likelihoods $P(\omega_i|y_k)$. As each row performs product probabilities with bit-streams, at each likelihood $P(\omega_i|y_k)$, there is a logical $AND$ gate.

The strength of this architecture relies not only on the efficiency in the calculation of the product of probabilities logical $AND$ gate but more importantly, on its particularly suitable architecture for In-Memory-Computing. As a matter of fact, each blue box in Figure 3.6 corresponds to a probability that can be stored in a memory array.

Even if it would have been possible to input each bitstream corresponding to $P(\omega_i|y_k)$ from outside the architecture, and to have a circuit dedicated to the stochastic product, the in-memory computation approach seems highly attractive as it avoids massive data-movement.

Instead of presenting at the top column a bit-stream for each likelihood, this architecture only requires the configuration of the observation (e.g. the temperature value). Consequently, $\omega_i$ corresponds to an address that is shared along the whole column, each memory array in the light blue cells addressed by $\omega_i$ gives the corresponding likelihood $P(\omega_i|y_k)$.

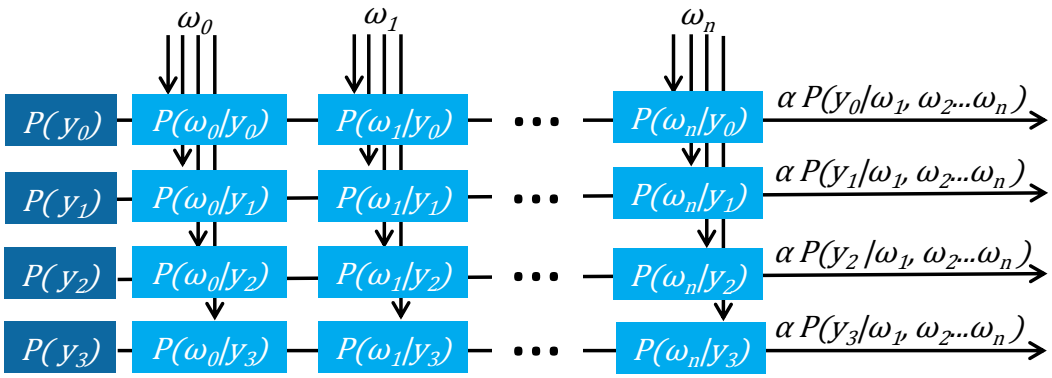



Figure 3.6: Schematic of the implemented Bayesian machine basic principle. Each row perform the calculation of the posterior distribution $P(y|\omega)$

The system works as follows:

- The dark blue blocks –*the priors*– generate a random bit-stream with probability $P(y_k)$ at each row level. This bitstream can be generated intra-architecture or out-of-architecture, as these priors can have been generated either previously by another system or come from the posterior distribution of the same system.

- The light blue blocks correspond to likelihoods modules. Each of it consists of a memory array that stores the different likelihoods $P(\omega_0|y_0)$ with precise value. For instance, for a thermal sensor, each temperature will have a different likelihood. As a consequence, each column receives the same information $\omega_i$ but assigns different values to the likelihood. Therefore one column is a map of the likelihoods for a given observation.

- When the various observations are presented to the system, the system chooses the corresponding probability from the set of probabilities available. Each likelihood block will then generate random bits proportional to the selected probabilities in the array.

- The bitstream at the output of the system is finally proportional to the product of the likelihood and the prior probabilities

In practice, to speed up the calculation, the probabilities stored in the system do not correspond to the true probability, they are normalized, by the highest value for each column.

### 3.4.2 Global Architecture

Figure 3.6 presents a schematic overview of the machine, while Figure 3.7 shows the complete design. As mentioned above, the architecture is mainly composed of memory arrays storing probability values. Each memory array is used to save the different likelihoods of the different possible observations given a single configuration of the hidden variable. By combining all probability products in rows, the last column gives the complete posterior distribution of the hidden variable given the observations.

The complete design, Figure 3.6 (a) consists of a pattern (the likelihood cell) repeated many times. As shown in Figure 3.6 (b), each "likelihood cell" will therefore be composed of a memory matrix, a random number generator, a circuit able to transform these random bits into a bit-stream proportional to the probability value read from the memory array and an *AND* gate that performs the probability product.

To control each of these cells, but also to manage the inputs and outputs of the system, other blocks are added. The memory controller has the task to manage the addressing of the memories for both writing and reading. When writing the memories each cell will be addressed independently, while when reading, each probability of the same column uses the same address. Input/output controllers are used to manage the data. The input controller is used to manage the writing of memories, the generation of stochastics bit-stream for the prior, and the loading of seeds necessary for the generation of pseudo-random numbers.

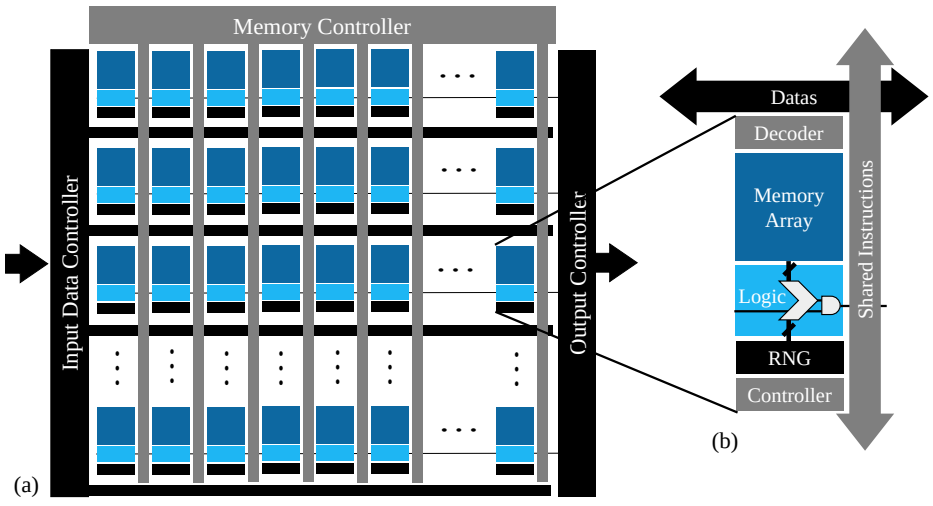



Figure 3.7: (a) Schematic of the fully implemented Bayesian machine formed of a repetition of basic cells. (b) A basic cell is composed of a memory array that stored all the likelihoods for a given observation, a Random Number Generator (RNG) –it can be True RNG of pseudo-periodic RNG– both are working for the Bayesian Inference, and a logic module composed of a comparator that makes the comparison between the generated random number and the stored probability, generating a bit-stream according to this probability.

The architecture presented allows Bayesian inference and not learning, i.e. likelihoods are fixed and no longer change. The use of non-volatile technologies seems particularly suited to the implementation of such a system.

Each technology requires different methods to be programmed, e.g. different voltages. Sometimes, level-shifters [212] are needed to manage the programming voltage levels. For this reason, more detailed design with all the inputs/outputs of the system is presented in section 3.6, showing all the control signals needed for both memory programming/reading RRAM memory technology as well as signals for random number generation.

### 3.4.3 Random Number Generator

The quality of random number generators is one of the most important considerations in the use of bitstream encoding data. Each variable must be independent and decorrelated in time to ensure a consistent result on the mathematical operations. There are two main categories of random number generators: PRNGs (Pseudo Random Number Generator) and TRNGs (True Random Number Generator). TRNGs are obtained from physical sources that have intrinsic entropy, such as thermal noise. Pseudo-random number generators are obtained by a deterministic system with a specific algorithm.

The presented architecture is incredibly interesting for in-memory-computing. The freedom concerning the generation of random numbers is therefore specific to the designer but also to the algorithmic constraints of the system.

One of the recent approaches to generating random bits is the use of memory technology, either by playing with the dimensions of the device[213] or by using the stochastic behaviour when it is programmed[214] [215]. The use of such an approach has a real advantage, in addition to being TRNGs, they can be implemented using the same technology as the one used for the memory array. However, this approach usually requires the addition of circuitry, either because the bits can be self-correlated in time, or because it involves programming devices that can even cause power overhead. This approach is fundamentally exciting but requires an in-depth study to be implemented.

Consequently, for the implementation of our machine, we used a pseudo-random number generator, the principle can be easily simulated on a computer, the result being deterministic, it is also possible to make an exact comparison with the implemented system.

The pseudo-random number generation system used is a Linear-Feedback Shift Register (LFSR). It is a pseudo-random generator that is inexpensive in terms of surface area and energy consumption as the implementation only requires flip-flops and $XOR$ gates. The physical structure of an LFSR is often presented in polynomial form, for example $x^8 + x^6 + x^5 + x^4 + 1$ for a 8-bits LFSR. This polynomial corresponds to the different shift register bits which are connected one after the other by XOR logic gates. For each size of LFSR, a configuration maximizes the periodicity of it, passing in all states except the null state.

This random number generator has three major drawbacks for our system. Firstly, the zero

state is never reached which decreases the number of values reached by the generator by one. Secondly, this generator is pseudo-random, this means that the sequence of generated bits is periodic, which results in a limitation on the length of the output bitstream configuration. Moreover, the number of LFSRs of the same physical structure that can be put in series for the stochastic product is limited. Indeed, two LFSRs must be different, otherwise, the bitstreams output from the different likelihoods will be highly correlated with each other. And thirdly, to generate different values, each LFSR needs a seed value that must be programmed, increasing the complexity of the system, and also requiring to be seeded every time we turn off the system.

### 3.4.4 Weighted Binary Generator

To perform stochastic computation, in particular the multiplication between each likelihood, we need to generate a stream of random bits whose ratio of 1 to 0 is proportional to a probability. Probability coding is done by a classical binary coding with a value between 0 and 1. When all the bits are at 1, the probability is 1, and when all the bits are at 0, the probability is zero. A digital cell is needed to convert the value of a probability in fixed-point coding into a bit-stream encoding this value. The digital circuit Figure 3.8 (a) was previously shown in the global architecture Figure 3.7 (b). Two implementations options are presented in Figure 3.8 (b) and (c).

In both cases, random numbers and probability must be encoded on the same number of bits. Otherwise, the generation of a stochastic bit stream proportional to the desired probability is not ensured. For the generation of the stochastic bitstream, a conventional comparator is usually used (Figure 3.8 (b)). If the random number generated is less than or equal to the probability shown, the output bit is 1, otherwise, it is 0.

Another method, proposed by Gupta in 1988 [216] [217] and presented in Figure 3.8 (c) consists in generating the proportional bitstream by comparing each bit of the probability with the random number generated by the LFSR. The system is based on the principle that each bit of the LFSR has a probability $1/2$ of being a 1 and a probability $1/2$ of being a 0. The *AND* gates are used to weight each bit generated by the LFSR by the probability loaded in the register. The logical *OR* gate is used to add these different options.

We designed this circuit using the SystemVerilog Hardware Description Language (HDL) and made a comparison on the energy consumption, area and speed of these two circuits. The circuit in Figure 3.8 (c) is around 10% better in both energy consumption and surface area than the circuit in Figure 3.8 (b), but these results depend on the number of bits considered.

The architecture of the stochastic Bayesian circuit is relatively little dependent on the choices of this circuit. If we work with memory arrays, the peripheral circuits are relatively large compared to the logic. As we will see in the next sections, in a full architecture with a large quantity of memory, the logic largely comes for free. This architecture allowing Bayesian inference is to a large extent composed of small memory arrays with little logic.

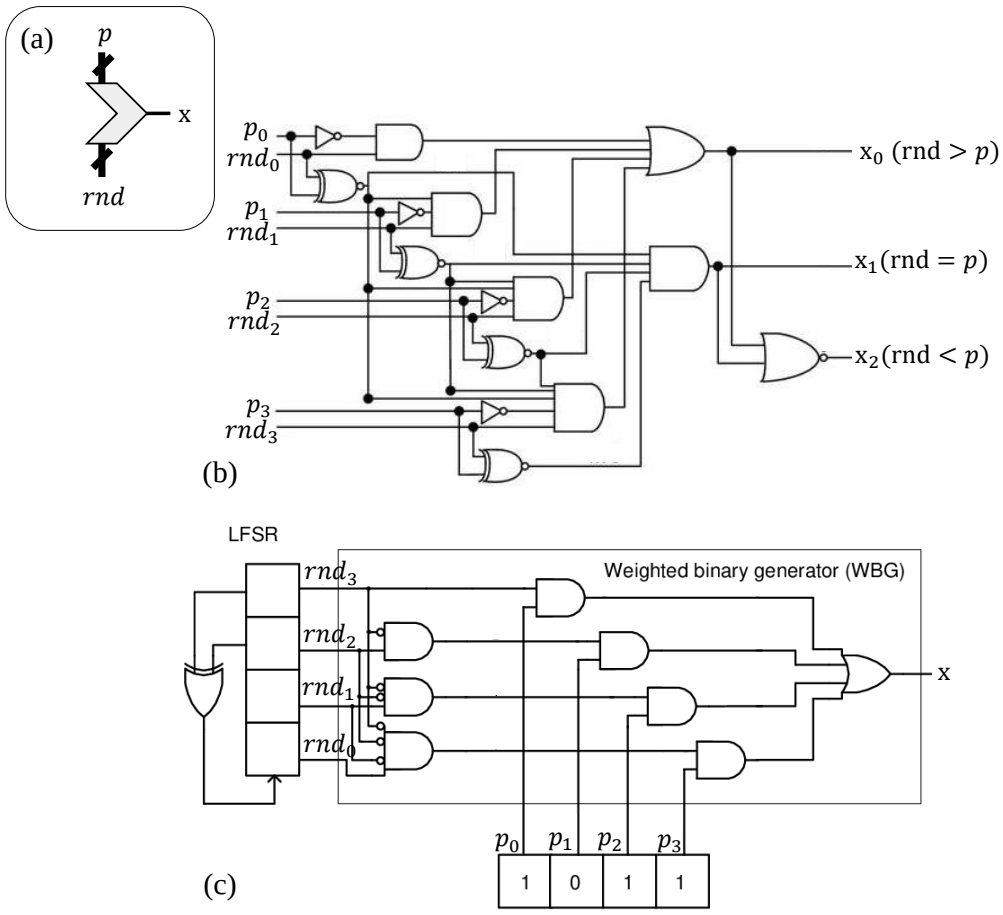



Figure 3.8: (a) Logical gate required to generate a bit-stream proportional to the value of a probability $p$, it requires the input $p$ encoding the probability in fixed point coding and $rnd$ which is a generated random number. (b) Classical circuit used to make the comparison between two numbers, both encoding in 4 bits fixed point, it can have 3 outputs, but to generate the bit-stream only $x_2$ is used. (c) Alternative circuit to generate this bit-stream, it is a simplified form in comparison with the comparator.

## 3.5 The boat localization example

For the implementation of the Bayesian inference model, we need to create an example model that we can test. The example requires specific observations and to draw inferences from them. The example that will be described in this section is about localizing the position of an object from sensor data, for instance, the position of a ship in the ocean from position sensors. This example was first introduce in the book *"Bayesian Programming"* [151].

This proposed example is relatively simple to understand, but its hardware implementation is relatively complex because it requires a large amount of memory. The hybrid CMOS/RRAM microchip that was fabricated does not have enough memory to make this full Bayesian inference. The study of Bayesian inference on our chip will therefore be relatively simple, with a

typical simpler task, which is to classify sleep phases from Electroencephalography (EEG) temporal data previously transformed in the frequency domain [218] [219]. Our chip will also be able to perform the Bayesian inference of the last layer of a neural network as mentioned in Appendix B.

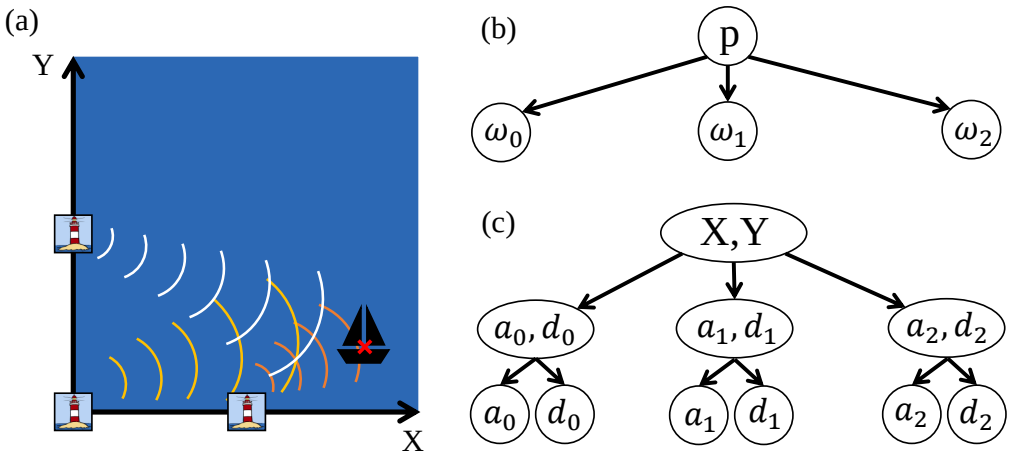



Figure 3.9: Description of the model for the boat localization. (a) Drawing presenting the problem, a boat is on a map and three sensors try to find the position of the boat. (b) The simplified form of the model, the position of the boat is obtained from three sensors that are conditionally independent given the position. (c) Full description of the model, the position of the boat is described by the joint variable $(X, Y)$ and each sensor gives an estimated position using polar coordinates, consequently, they can be decomposed into two terms, a distance and a position. All the observations of angles and distances are conditionally independent given the position of the boat $(X, Y)$. To determine the position of the boat, the Bayesian Inference simply consists into the computation of Equation 3.12 $P(p|\bigcap_{i\in I}\omega_i) \propto P(p)\prod_{i\in I}P(\omega_i|p)$

Figure 3.9 (a) describes the basic problem of the localization of a boat in a map. The position of the boat $p$ is determined by each sensor observation $\omega_i$ Figure 3.9 (b). The variable $p$ can be described by the two parameters $X$ and $Y$ that are not independent, therefore we need to describe the joint distribution $(X, Y)$. Each observation of sensor gives a specific position using polar coordinates, but instead of modeling the joint distribution $(a_i, d_i)$ which can be very large, it is possible to describe each parameter $a_i$ and $d_i$ as conditionally independent given the joint distribution $(X, Y)$ (Figure 3.9 (c)). This model finally results in a naive Bayesian inference between the different likelihoods of angles and distances of sensors.

To obtain the values of likelihoods, it is possible to perform a sampling based on a very large number of examples, in which we know the position of the boat, (i.e. the distribution $(X, Y)$) and we look at the sensors' response and then determine the different probability values. However, this is not always possible to do: either there is too little data available, or the experiment is far too complex to execute. Nevertheless, it is possible to add human knowledge to model these sensors. This is one of the major strengths of Bayesian models.

To model the behavior of the sensors, it is typical to work using Gaussian functions. The likelihoods correspond to the response of a sensor given the position of an object at a specific position $(X, Y)$. When an object is at position $(X, Y)$ the angle and the distance sensors will determine the corresponding angle and distance with a certain error. It is this error that is determined in a Gaussian way. If the object is at the position $(X, Y)$ it is simply a conversion from Cartesian coordinates to polar coordinates with the sensor as a reference to get the corresponding distance and angle. A Gaussian distribution centered on the exact values is then added.

It can be required to add complexity to the model, for instance, a sensor can be less accurate when the target is at a long distance. In this case, the standard deviation of the Gaussian distribution can be increased according to the distance of the sensor.

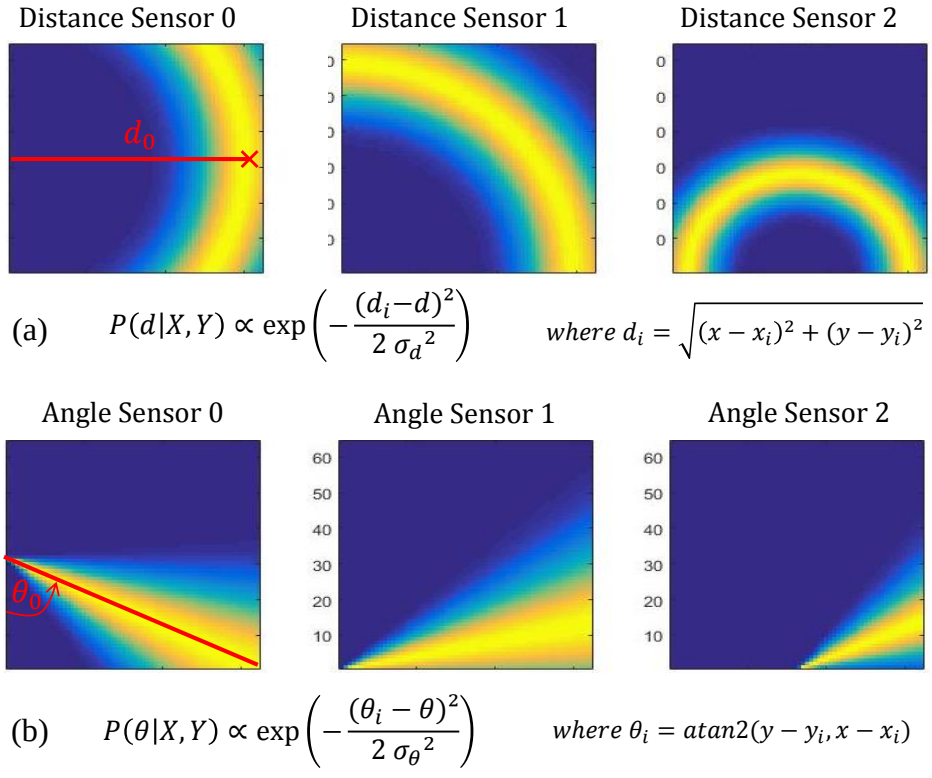
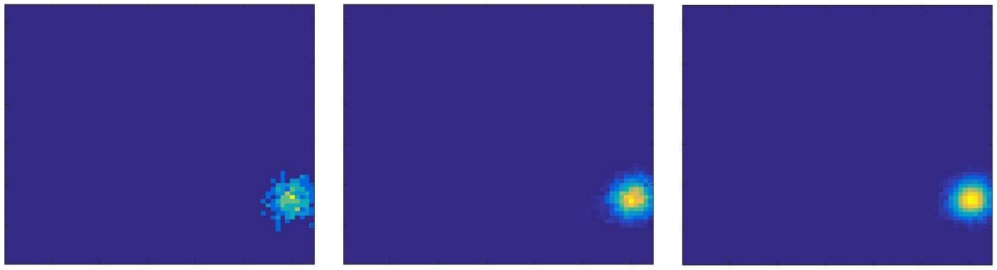



Figure 3.10: Sensor likelihood model. (a) Three distance sensors give an estimation of the position of the boat. For each sensor at a given position $(x_i, y_i)$, $N$ maps are stored for different values of $d$, generated from the Equation on $P(d|X,Y)$. When doing the Bayesian inference the sensors will chose one of this $N$ stored maps as "likelihood". (b) Likelihoods maps for three angle sensors obtained from the Equation on $P(\theta|X,Y)$.

To evaluate the performance of the system, we created a cycle-accurate model that simulates the operation of the electronic circuit. The system perfectly reproduces the behavior of the circuit and is interfaced with the hardware description in SystemVerilog using the same

probability coding files, random generator seeds, and using the same logic functions. This interfacing allows us to perform fast simulations and to evaluate both system parameters and the likelihood values stored in memory.

Figure 3.11 shows different inferences made after different duration: after 10 (Figure 3.11 (a)), 100 (Figure 3.11 (b)), and 1000 (Figure 3.11 (c)) clock cycles corresponding to the size of the bit-stream. The greater the number of clock cycles, the greater the accuracy of the inference, but even after 10 clock cycles, the prediction is accurate. It should be noticed, however, that the precision of the system is limited by the periodicity of the random generators. If this periodicity is too low, the same cycle will repeat the same inference result over and over. To avoid this concern in the future, TRNGs will be preferred.



(a) 10 cycles (b) 100 cycles (c) 1000 cycles

Figure 3.11: Naive Bayes Inference for the toy problem of boat localization obtain after (a) 10 clock cycles, (b) 100 clock cycles, and (c) after 1000 clock cycles

To evaluate the strengths of such a model, we also added complexity to the system by increasing the complexity of sensors, changing their position or tracking the temporal movement of the boat on the map. Adding sensors makes the model more complex, but the accuracy of the inference is increased. This accuracy is increased if the sensors add information about the position of the ship. For instance, a sensor close to the boat will have higher accuracy. Changing the position of the sensors requires changing the values of the likelihoods stored in memory. This is simply a pre-process of the model. To track the position of the boat over time, its prior position can be considered as the posterior position at the previous time step.

## 3.6 The final Chip Design with RRAM

The implemented system is the one presented in the previous sections but on a smaller scale. This is a proof-of-concept circuit and not a perfect design for future applications. The goal of this microchip is to create an intelligent memory where memory and calculations are co-located resulting in very little energy consumption for inference.

The fabrication of a hybrid CMOS/NVM chip is performed in two phases. A classical silicon implementation using a 130nm commercial technology and an implementation of RRAM cells whose architecture and technology will be described in the following parts. This design has

been realized in collaboration with Jean-Michel Portal and Marc Bocquet from the *"Institut Matériaux Microélectronique Nanosciences de Provence"* (IM2NP) at Aix-Marseille Université and CNRS, and manufacturing is currently performed under the supervision of Etienne Nowak and Elisa Vianello at CEA-Leti in Grenoble.

Being able to experiment the fabrication of a hybrid CMOS/RRAM chip is a true opportunity to test at an early stage the future of microelectronics, to implement original algorithms, and to understand what are the constraints that occur in real chips in contrast with high-level description.

### 3.6.1 The Design Flow

The circuit architecture that we will implement uses ASIC circuit integration tools. An ASIC *(Application-specific integrated circuit)* is a circuit designed for a particular use and, therefore, dedicated to specific tasks.

To describe the architecture of such an electronic circuit, designers use a hardware description language (HDL), which describes the functionalities of the chip. The code can be interpreted and synthesized using system design tools (such as the tools provided by Cadence or Synopsys), a library of logic gates (usually provided by the design kit of a foundry). These gates are optimally designed to limit power consumption, delays, and surface area.

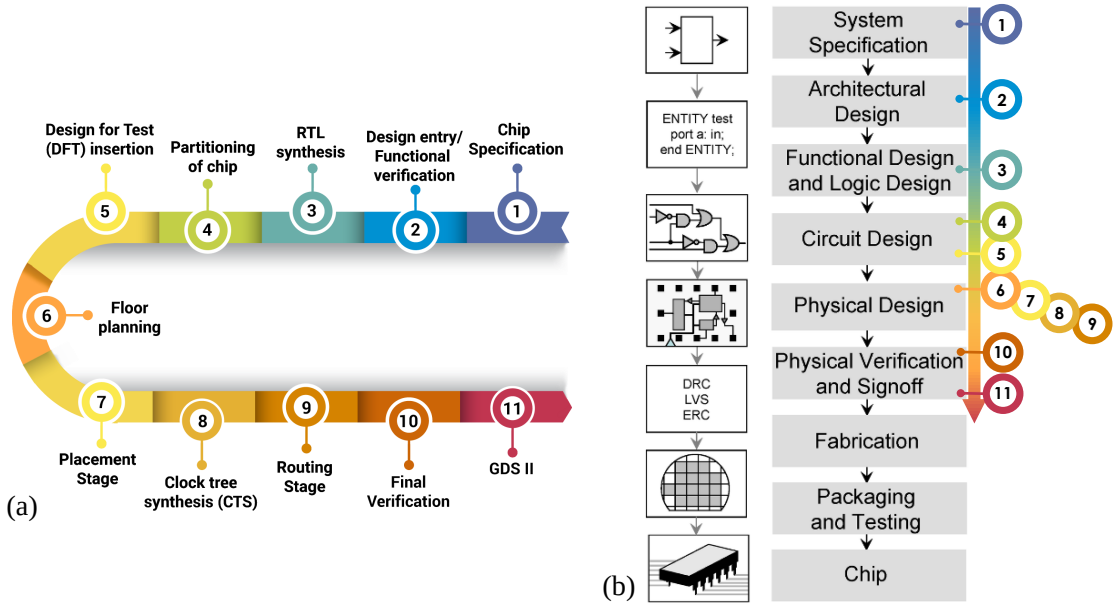



Figure 3.12: Overview of an ASIC Design Flow. (a) Simulation steps that are done from Design Tools, only simulation based. At the end of these steps GDS masks are obtained and used for the manufacture process. (b) All main steps from system specification to ready to use final chip.

To design the Bayesian system, we used the SystemVerilog hardware description language.

Once the system structure is written, the model is simulated to confirm that the design will work correctly.

When setting up a design flow most of the steps can be automated. Nevertheless, considering the particularity of our design integrating RRAM at the core of CMOS, we have performed a semi-automated design. The layout of the devices and peripheral circuits were placed and routed by hand, while the digital part was pre-placed and routed and then assembled in the final design.

### 3.6.2 The RRAM memory array

In this section, we will present in detail the memory array used as the foundation of our design. First, we will explain the technology used and how it is programmed, and then, we will present the full implementation of the memory array including an explanation of the peripheral circuits required to program it.

#### Presentation of the RRAM device used

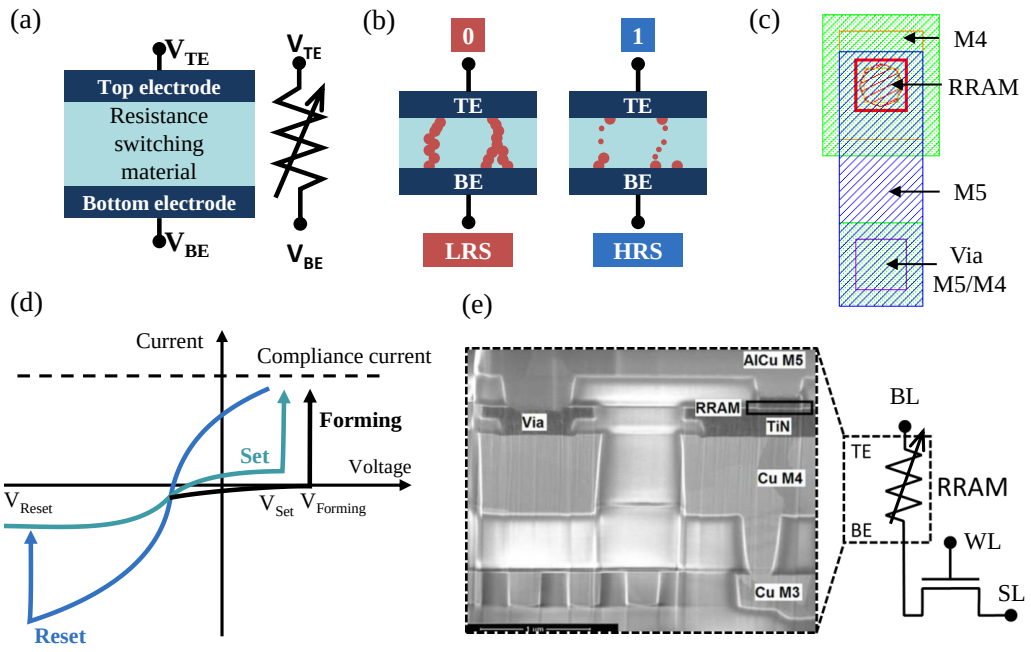



Figure 3.13: (a) Basic principle of all RRAM technology, a material resistance switch occurs under a specific applied voltage. (b) Low Resistance State (LRS) is coding for a binary 0, this state corresponds to the presence of the filament & High Resistance State (HRS) is coding for a binary 1, this state corresponds of the absence of the filament (c) Mask of the device used in our HfO2-based OxRAM integrated with a 130 nm CMOS logic process, the active region is located between metal-4 and metal-5. (d) Schematic of the Bipolar switching behaviour for Forming/Set/Reset process. (e) SEM cross-section of the device –TiN/HfO2/Ti/TiN–

To implement our circuit, we, therefore, use a resistive RAM technology. RRAM technologies are technologies based on the changing resistance value of a material when voltages are applied to it Figure 3.13 (a). The memory component used is a $HfO_2$ OxRAM based RRAM. It is a oxygen vacancies filamentary-based technology presented in Figure 3.13 (e), the mask of the device is presented in Figure 3.13 (c), the active region is located between metal-4 and metal-5 presented in Figure 3.13 (c) & (e).

Before being programmed, these resistive memories will endure a forming process. To achieve this process, a high voltage $V_{forming}$ is applied to their terminals to form the initial filament. The physics of these devices is relatively complex, but the basic principle is as follows [220] [221]. $HfO_2$ based OxRAM– have SET and RESET voltages of opposite signs 3.13 (d). When applying a positive voltage, for the forming process and SET process, oxygen ions drift to the anodic interface of the high electric field, where they are discharged as neutral oxygen. Thus, the current flows through the oxide via the conductive filaments of oxygen vacancy, resulting in a low resistance state (Figure 3.13 (b)). During the RESET process, the electrode/oxide interface behaves like a reservoir of oxygen ions. Ions migrate to the oxide to recombine with the filament of oxygen vacancy, the memory cell is then in the high resistance state (HRS). In [222] & [223], a deep study of the physical $HfO_2$ principle is made, detailing the principle of migration of oxygen ions.

In general, all RRAM technologies are difficult to model, as their underlying physical processes are generally dependent on a large number of parameters with a significant amount of noise [224]. However, there are well-defined programming methods that allow each device to be binary programmed as 0 (LRS) and 1 (HRS), as described in Figure 3.14.

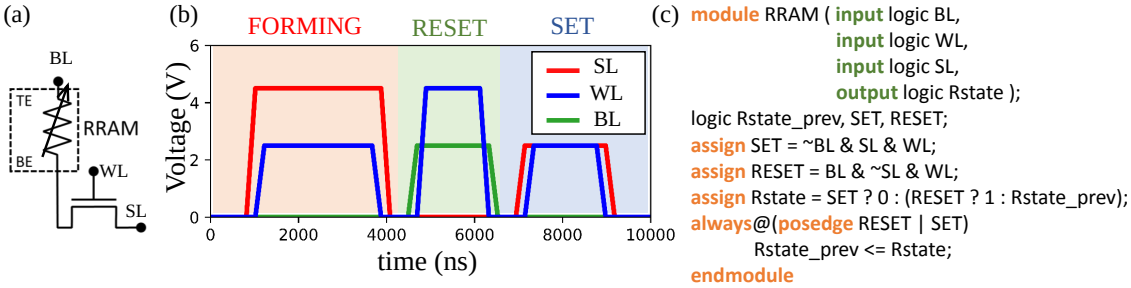



Figure 3.14: (a) 1 Transistor 1 Resistor structure with the corresponding terminals BL, SL and WL. (b) Programming schemes for the Forming/RESET and SET process, the voltages and timing presented are not the typical ones used for our OxRAM. (c) SystemVerilog behavioural model of the 1T1R structure used for the simulations.

The Forming/RESET/SET programming procedures are represented in Figure 3.14 (b) with the voltage applied to the different terminals of a 1T1R architecture (Figure 3.14(a)). For the forming process, a high voltage is applied at the Sense Line (SL) and the Bit Line (BL) is set to ground, while the access transistor is partially turn-on, limiting to the compliance current

with a specific voltage on the Word Line (WL). The SET process is the same as the forming process but with lower voltage and shorter programming time. For the RESET process, it is the opposite: SL is set to ground while a voltage is applied to BL. The access-transistor is fully open. Accordingly, we wrote a SystemVerilog behavioral model of the device that replicates the binary operation of the device under stereotypical programming conditions (Figure 3.14 (c)), which will be used to model the full behaviour of our Bayesian architecture.

The values of the voltages used are defined and can be adjusted when testing the chip. In a previous run on the same technology, different voltages have been tested. In the implementation of our chip, we chose a simple design where all voltages are generated outside the die and applied directly to it. This way, different values of voltage and programming time can be easily defined.

### 2T2R array

In the implementation of our final circuit, we employed a 2T2R structure, i.e. instead of using a single component to encode a binary value, we use two RRAM and two access transistors. To encode a binary value 0, the LRS/HRS configuration is used and the opposite HRS/LRS configuration is used to encode binary value 1. Thorough investigation of this configuration, which reduces the number of errors, will be presented in the next Chapter 4, in the context of Binarized Neural Network.

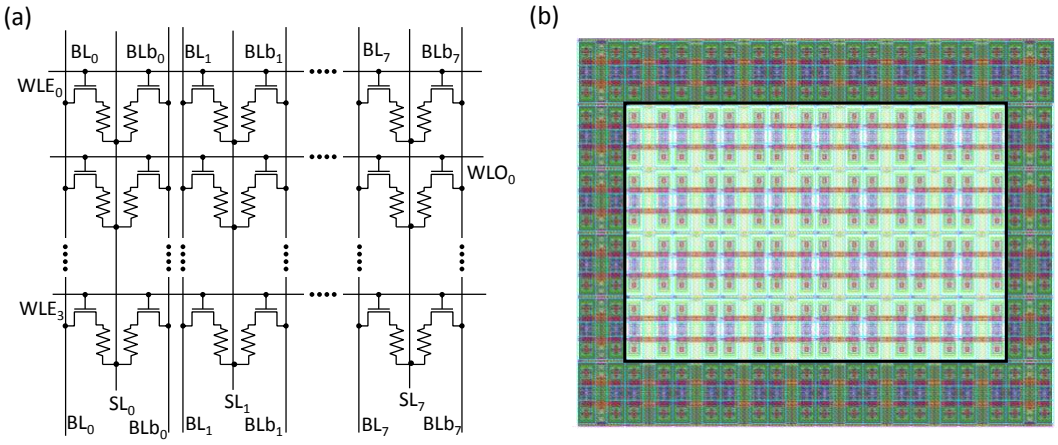



Figure 3.15: (a) Schematic of a 2T2R structure with all the control lines (b) Layout of the 2T2R array, with dummy devices all around

The programming principle of the 2T2R configuration, presented in Figure 3.15 (a), is the same as in the 1T1R one. However, the SL and WL of both devices are shared, and each one possesses its own Bit line, BL and BLb. To prevent sneak path through the complementary device, when performing the programming steps, it is necessary to apply complementary voltages to the complementary device Bit Line to manage these different voltages.

The memory array is composed of a set of devices configured in 2T2R structure, the line control signals allow managing the transistor access (WL) while in the column, the controlled signals are those that allow the programming and reading of the devices (SL and BL).

Additional devices are located all around the main memory array (see Fig (b) shaded area), this way all devices in the core of the memory array have the same number of devices at their interface, reducing surface effect variability.

### Drivers

CMOS transistors optimized for numerical computation do not allow to control the high programming voltages of RRAM devices. To manage these high voltages, it is necessary to use bigger transistors with thick gate oxide. In our design, we have therefore used two types of transistors, small transistors thin-oxide for the digital computation and bigger thick-oxide transistors in the memory array periphery. The bigger transistors are only used in driver circuits. These drivers are composed of small transistors that work using the nominal voltage $V_{DD}$ controlling the bigger transistors able to work with higher supply voltage to program the RRAM devices.

Row and column drivers are used on all four sides of the memory array. The control line drivers are arranged as follows: column driver on WLE on the left, row driver on WLO on the right, BL and BLb driver on top, and SL driver on the bottom.

These drivers control the voltages applied to the various terminals of the 2T2R structure from digital circuits supplied in VDD or GND. For the power transistors to operate and apply the correct voltages to these terminals, two additional power supply voltages pulled to the input pads are added: a row supply voltage VDDR and a column supply voltage VDDC.

The driver on the BLs is a bit different from the others when writing we will have to apply a high voltage, hence the need for the driver. However, when we want to read, we have to leave these lines at high impedance to discharge the current in the reading circuit, so an additional control signal CBLEN is added to control if we are going to apply voltages on the BLs or leave this value at high impedance.

### The PCSA reading circuit & near-memory logic circuit

Once the devices have been programmed, the binary value that they store must be read. To do this, the memory array features a dedicated peripherally circuit shown in Figure 3.16 (a). It was first introduced in 2009 [65] and is called Pre-Charge Sense Amplifier (PCSA). The operating principle is as follows. The first phase consists of charging the RRAM devices and the comparison latch to the supply voltage. To do it, SL is charged at supply voltage and the SEN signal is set to the ground. With SEN signal set to ground, the PMOS transistors are turned on, and the supply voltage is applied at the two states of the latch. The second phase consists of discharging the voltages at the devices through the SL. This is achieved by setting the SEN signal to the supply voltage and SL to the ground. Considering that the latches are both at supply voltages, the NMOS transistors are turned on, letting current flow through the branches of the RRAM de-

vices; but since the two devices have different resistances, the discharge speed is not the same in each of the branches. Because the current is greater in the low-resistance branch, the low-resistance branch discharges faster. The state of the branch of the latch with low resistance will decrease faster than the other with high resistance, this disequilibrium will be amplified until the state of the high current branch discharges to the ground controlling the PMOS transistor of the complementary branch. As a consequence, the PMOS transistor of the other branch will be charged to VDD. The state of the latch is then used as the binary value stored in the RRAM devices.

The advantage of this circuit is that it also allows operations to be carried out directly in the reading circuit by adding a few NMOS transistors. This approach presented in [66], is very interesting to reduce the energy consumption of the calculations. In Figure 3.16 (b), the 4 NMOS transistors are controlled by a DIN input data which will define in which branch of the sense each resistive device will correspond. If DIN is at logic level 1, it is a classical PCSA, if on the opposite the DIN signal is at 0 it is always the same PCSA but the branches are inverted. The output is switched only if DIN is 0. The operation performed in this way is an XNOR between the binary value stored in RRAM memory and the DIN input data.

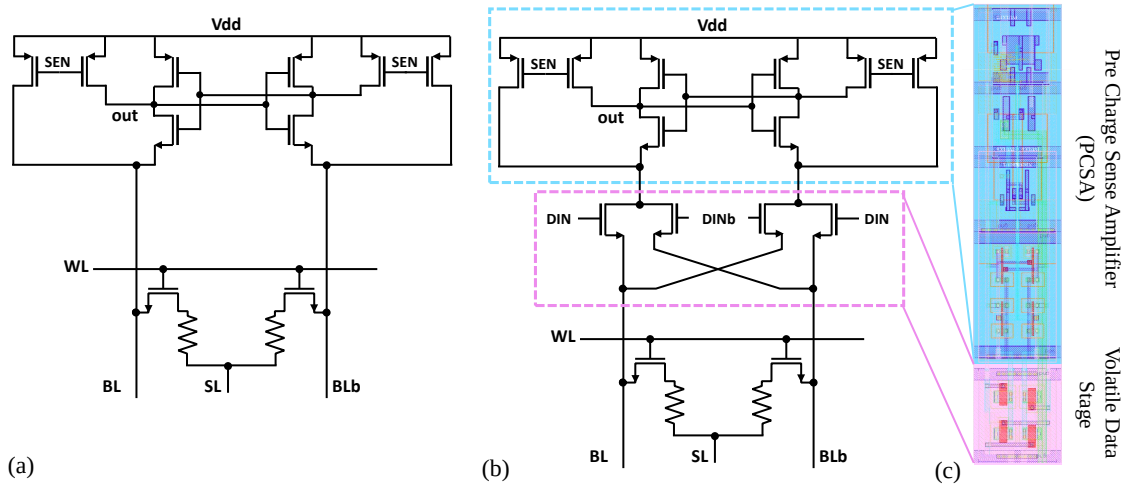



Figure 3.16: (a) Schematic of the Pre-Charge-Sense Amplifier. (b) Schematic of the Pre-Charge-Sense Amplifier with additional volatile data stage that perform an XNOR operation between the RRAM binary state and the DIN input. (c) Layout of the Sense amplifier used in our final design.

As long as the addition of transistors and their resistance is negligible compared to the resistances of RRAM devices, it is possible to make complex operations by adding new transistors. Nevertheless, it should be remembered that the output remains a 1-bit binary output, so complex operations can be performed, for example, AND gates with three inputs but not more complex logic functions. To carry out more complex logic gates, a new differential pair must be added. With this technique, for instance, in [66], it has been possible to create a non-volatile full adder.

In our final circuit implementing the Bayesian machine, we do not use this Volatile Data Stage input to perform digital operations. Nevertheless, these transistors are used to isolate the reading circuit during the programming step. When programming the devices, DIN and DINb are set to the logic zero states, so no current that could damage the sense amplifier flows through it. The layout of the sense amplifier used in our final design is shown in Figure 3.16 (c).

### Full memory Array

The basic elements of our Bayesian design are the RRAM components. The principle of programming and reading devices has been presented in the previous paragraphs. These devices are arranged in the form of a memory array of size 8x8bits. For the implementation of our model, we used very small arrays. The surface of the complete system circuit is therefore dominated by the periphery. The entire memory array is shown in Figure 3.17. To power the circuit, there are 4 power supply rings all around the memory array. VDD is the voltage to power the logic transistors that control the drivers and the sense amplifier. GND is used for the transistors and to discharge the currents. VDDR is the row supply voltage, it is the one used for the WL transistors. VDDC is the column supply voltage, this is the one used for the BL & SL lines when writing.

The drivers are all around the memory array, the read circuit (PCSA) is at the bottom. The top drivers are bigger than the others because they are composed of a double command, CBL, and CBLEN. CBL is used to address which component of the differential pair is selected and CBLEN decides the application of voltage to the device.

We can see that the memory array is mainly composed of wires. The 5 levels of metals are used in the RRAM array and very loaded in the drivers but as for the rest, there is space to add logic transistors below.

In this presented array, all devices on the same row can be programmed and read in parallel from the input command lines (CBL, CBLEN, CWL, and CSL). In the final design, due to the small number of components, each device is programmed sequentially but is read in parallel.

Each of the control lines is going to be controlled by decoders but, in our final design, we still have direct control over these rows. It is a waste of energy, as we need to load all the lines from the PAD to transistors, but it is easier to test the array and to test different programming conditions. If we perform the test on this basic cell efficiently, all the other designs are simple to test as they are derived from this memory array.

The control of all these lines thus passes through a logic circuit whose implementation method will be described in the following sub-section.

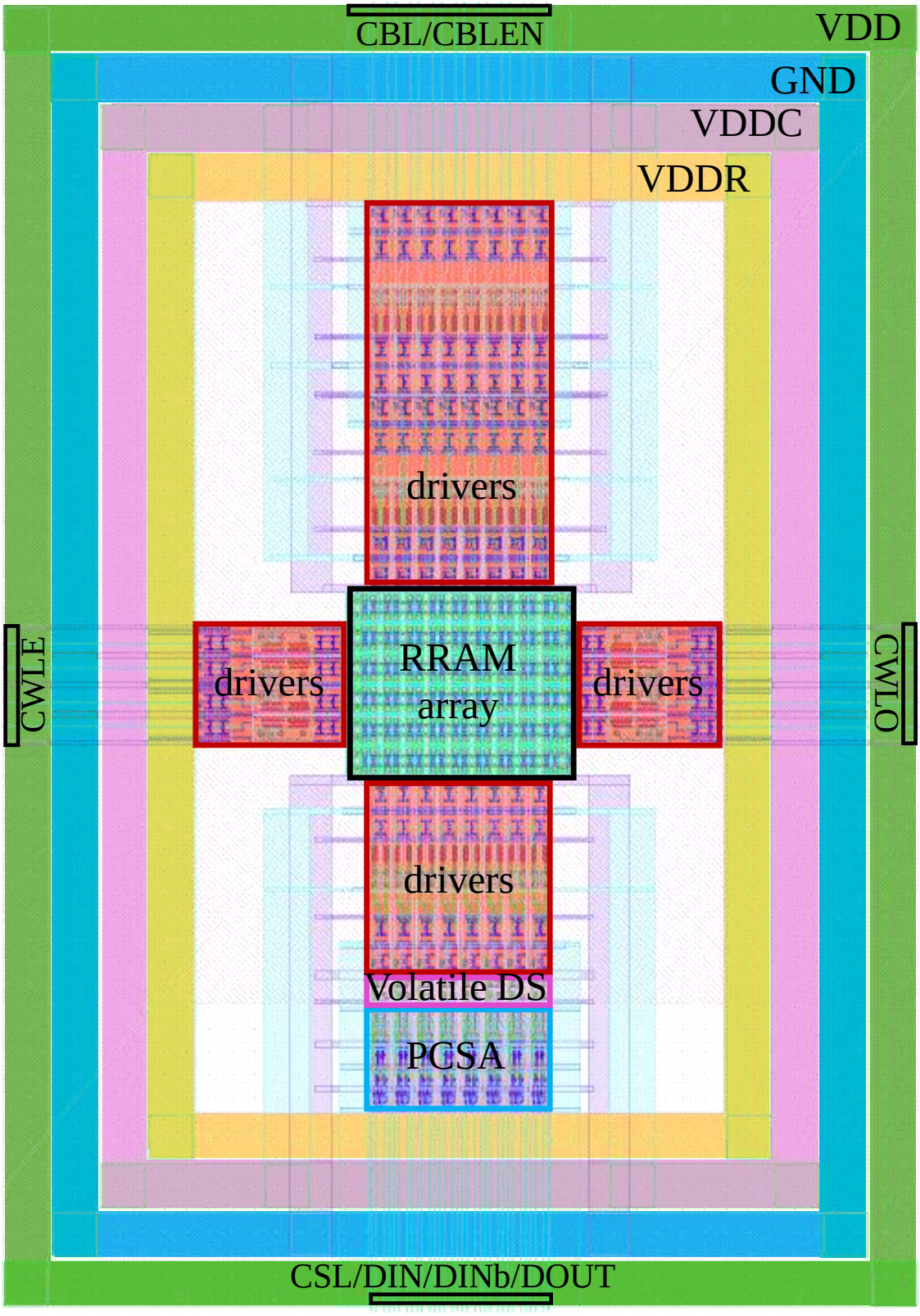



Figure 3.17: Memory array composed of 16x8 readable and programmable RRAM devices allowing 8x8 binary bits to be encoded in the 2T2R configuration. The dimensions of the circuit are 200x290um.

### 3.6.3 Digital Logic Blocks

In this section, we detail the use of the 130nm digital CMOS flow. It will be presented in part as a working document. The memory blocks have been made full-custom, i.e. the transistors, RRAMs, and wires have been arranged by hand using the layout Virtuoso tools provided by Cadence. The implementation of the digital blocks was done automatically with a place and route flow. On the other hand, the memory array was assembled by hand. To use both in our design, we used a behavioural model to do the simulation, and the memory array was used only for the final assembly to obtain the GDS file for fabrication. A complete flow that integrates a place & route of both, memory blocks (pre-designed in full-custom) and logic have been implemented in the IM2NP lab and recently reproduced in our laboratory for our future designs, but for the implementation of the whole die assembly everything was done in full-custom.

As an example, we present the logic block called *complex_decoder_top* in Figure 3.18 (d), which is the basic element of our Bayesian circuit. Inside it, there are the random number generators, one per column, the numerical commands to control the decoders, as well as different instructions to define the different steps of the system. We used CADENCE® Encounter RTL compiler to generate the layout of the circuit.

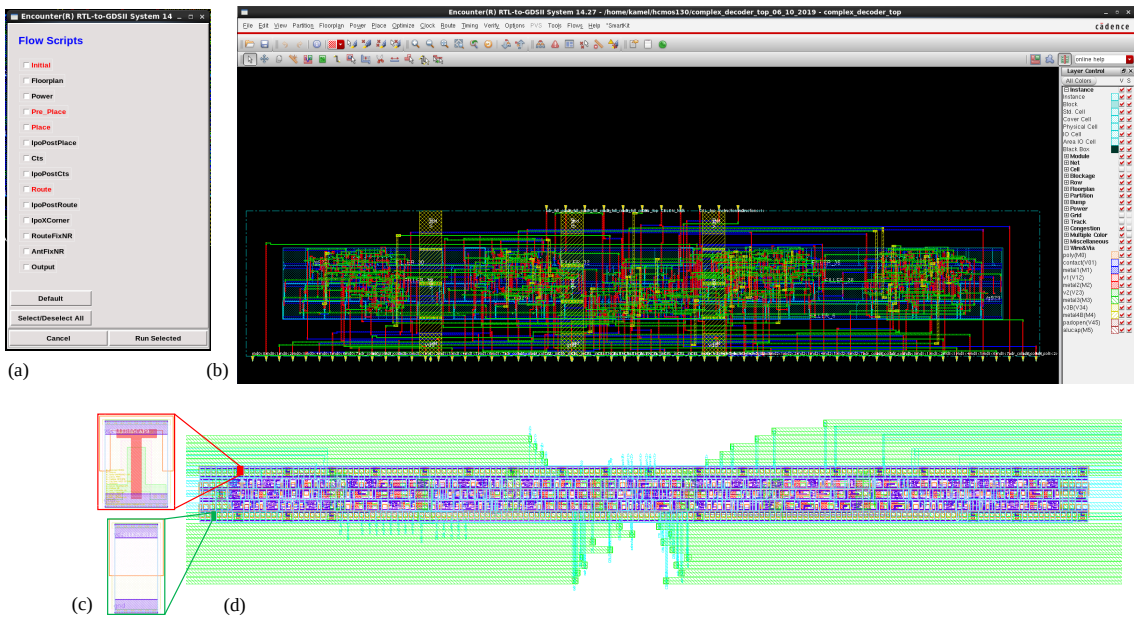



Figure 3.18: (a) Tools used to place & route the digital part of the design. (b) Example of the final placed & routed digital block *complex_decoder_top*. (c) Filler Cap & Fill Cells placed by hand all around (d) a the digital block, input and output pins are also placed by hand to match the full design.

The place & route flow uses an automated flow provided by the CMOS foundry (Figure 3.18 (a)). Once all the steps are finished, we can take the layout and add some wires to connect the logic block with the rest of the design. In Figure 3.18 (d) we can see in green that we added

some wires to connect this logic block to the memory array. To have continuity between the logic gates and logic blocks and that the design respects all the VLSI rules, fillers caps, and fillers cells are used (Figure 3.18 (c)). Usually, the routing placement tool fills the empty spaces with fillers caps and fillers cells automatically so it is not a consideration in most cases. In our case because our design is fully custom we need to add these extra cells. Filler cells are used to have power supply continuity as well as NWELL and PWELL continuity. Filler caps are used to avoid voltage drop and ground bounce.

### 3.6.4 The Bayesian Machine full architecture

In the previous sections, in Figure 3.6 and 3.7, we briefly showed the principle of the Bayesian Machine architecture. It simply consists of a generating stochastic bit-stream proportional within each likelihood and multiplying it with the neighbor's one. Thus, by propagating the probability product, we obtain at the end of the row the naive Bayesian inference. The digital part of a cell is therefore very simple, it is simply a random number generator, an "AND" gate, and control signals for the 8x8 bits memory array.

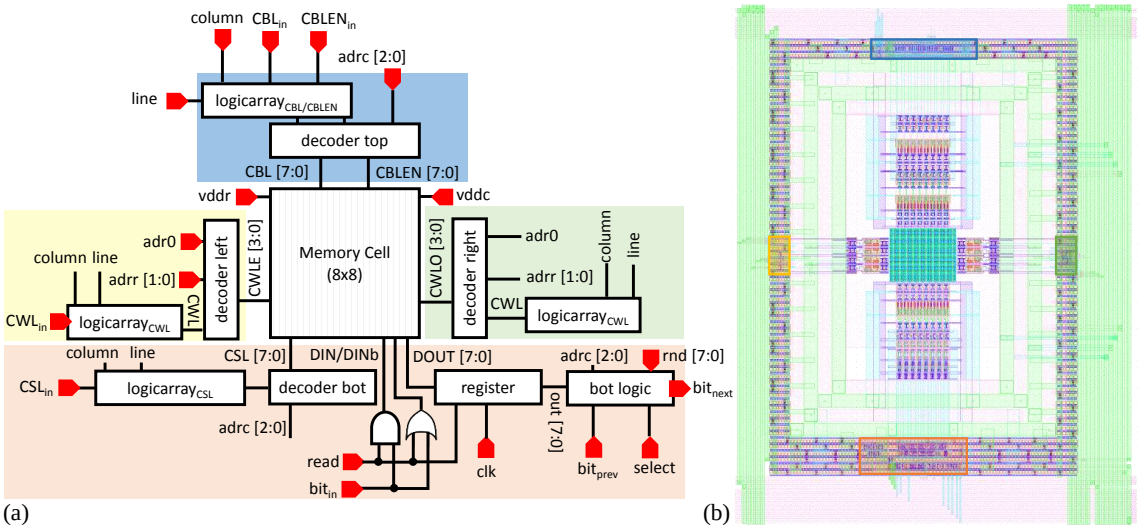



Figure 3.19: (a) Details of the basic likelihood cell which is repeated 4x4 times to make the complete system. (b) Layout of the cell with wires all around that allow the connections between the different arrays.

The architecture we have designed physically follows these principles, with slight differences: instead of having one random number generator per cell, we use one random number generator per column, i.e. each cell in the same column will receive the same random number to compare to. The implementation of such a method is not optimal from a data movement perspective, since the randomly generated numbers will be made far away from the comparison cell. However, this method greatly simplifies the design because these random number generators need to receive a seed, i.e. an initial value at the start. Having only one random

number generator per column is therefore a way to simplify this initialization. This basic likelihood is very simple but receives a lot of input signals.

Details of the basic cell are shown in Figure 3.19 (a) and its associated layout in (b). The complete design consists of 4x4 times the basic cell, plus two fully digital blocks. The overall system is presented in Figure 3.20. It is composed of a module called *"complex_decoder_top"* which is used both to manage the inputs and outputs of the circuit and to generate the random bits using LFSRs. Another digital module called *"complex_decoder_left"* is simpler, and it is only used to manage the row signals of the memory array. The architecture of the complete system, therefore, appears in the form of a matrix of repeated memory arrays.

This handmade design is not optimal in terms of the used area. Nevertheless, we have been able to make a design with a maximum release on constraints, even if the circuit can be slower than a fully automated design, normally, it should work with low energy consumption.

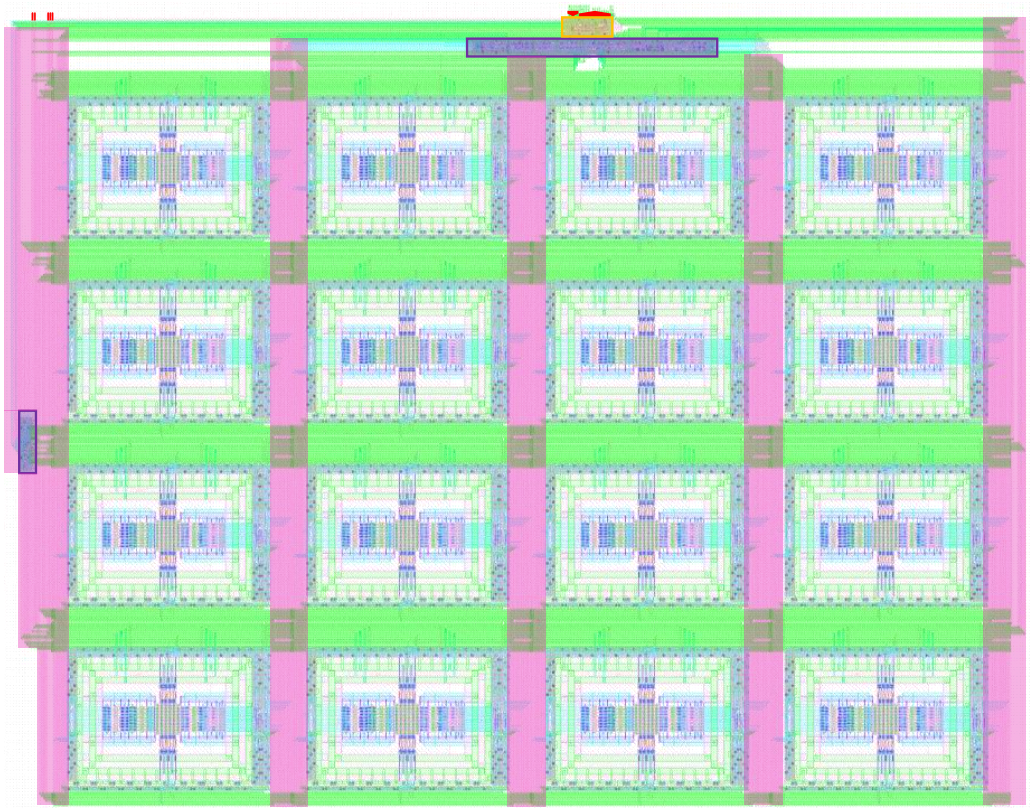


Figure 3.20: Complete design of a 4x4 array of the Bayesian Machine, composed of memory array of 8x8bits. Input/Output are connected to buffers (orange) then go through the *complex_decoder_top* in purple and *complex_decoder_left* that control the whole system. On top left, (in red) is presented input pins for the different voltages used to control the devices programming.

The major advantage of our design is that in addition to being able to perform the Bayesian inference operation, we designed it so that it can also read every bit stored in memory. The outputs of the system allow reading one bit in a row line. Such test capability is critical when

working with emerging technologies, as it can be important to make experiments on the programming conditions of the devices (see Chapter 4).

To verify and prepare the testing, we wrote a behavioural model of our system in HDL (SystemVerilog). The principle is as follows. We will program each bit of the matrix sequentially by applying the right control signals (address & instructions) and programming commands on WL, SL, BL, and BLEN as shown in Figure 3.14 (BL is used to select which device of the pair will be selected and BLEN the applied signal).

Once each device has been programmed and then checked, the inference can be made. To do this we initialize the LFSRs by programming bit by bit the values of the seeds. Once it is done the system is ready to work for inference. The output is in the form of 4 bit-streams.

Each memory array has 8 sense amplifiers, so when reading, 8 bits are read simultaneously. The specificity of the system is that the address on a column is the same for each of the memory arrays. Therefore, when reading the probability, we will be able to update the likelihood on a whole column of the system, which will change the posterior distribution accordingly. The step of addressing the columns is therefore done very quickly since it is a matter of addressing one address per column. This means that in the inference phase, the response of the sensors can be directly modified and the response of the system is determined accordingly.

### 3.6.5 Final Design and small array test structures

Our system will be tested with probe-cards featuring 25 pads. Therefore, to test the design we have 25 inputs/outputs including supply voltages and ground. We have optimized our system to work with this relatively small number of pads. The first design to be tested (Design 3) is the simplest design: it is simply the basic memory array with a few control signals. This design is detailed in Figure 3.21, with all corresponding pads on the layout and architectural description of the system with a correspondence on the layout.

This design consists mainly of 4 decoders, which allow addressing the different RRAM components and the programming control signals CBL, CBLEN, CWL, and CSL. The read signal directly controls the volatile stage for programming, if the read control signal is at logic state 1, the operations of the XNOR in memory can be carried out whereas if it is at 0 it isolates the memory array from the sense-amplifier. The $bit_{in}$ signal is the input signal of the logic data stage XNOR. A clock and an 8 bits register are used to read the output signal.

All programming is therefore done by external software that controls the addressing and programming signals. This design will allow both to prepare the test of the complete system, but also to test the technology. It also allows us to optimize RRAM programming conditions and is the first thing to be tested before testing more complex designs.

The layout of all our designs is shown in Figure 3.22, both designs 1 and 2 are very similar to design 3: they also feature a single 8x8 memory matrix, to which we added a little more digital complexity. Design 4 corresponds to the complete system of the 4x4 Bayesian Machine.

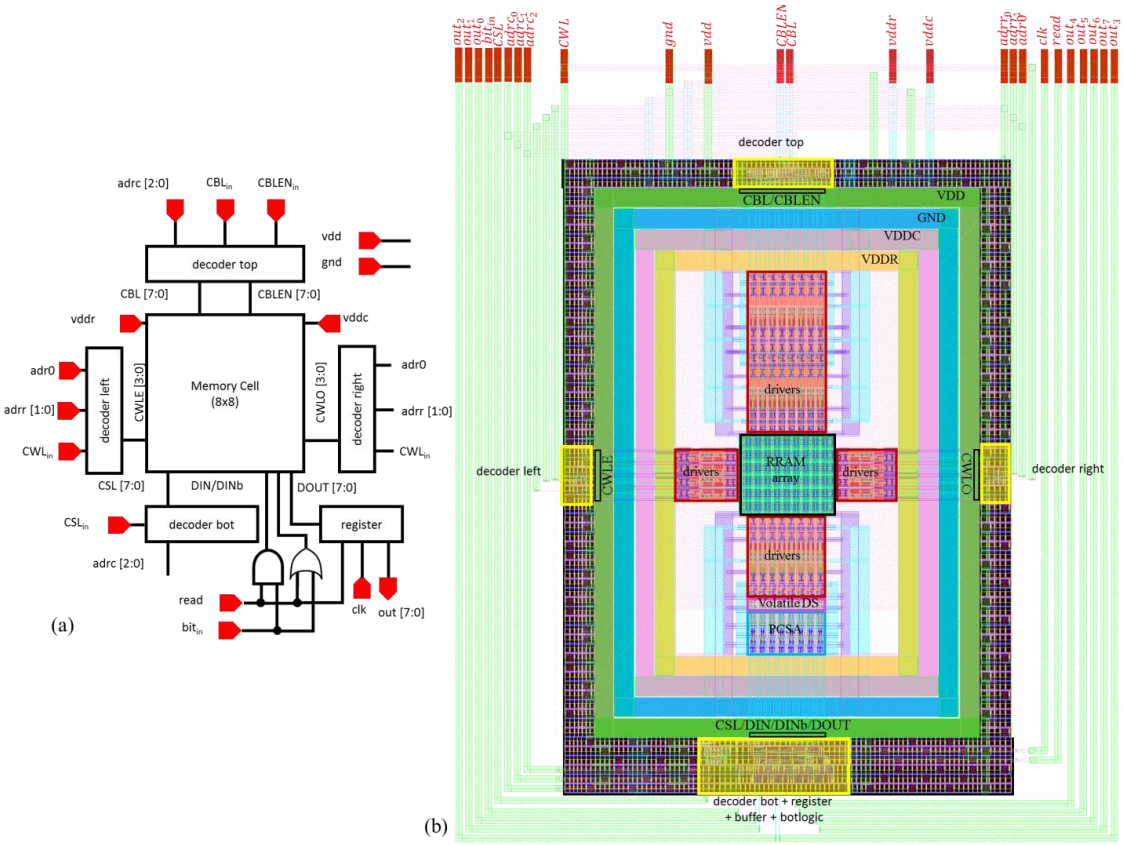



Figure 3.21: (a) Schematic of the implemented design, the decoders are used to address each device independently with control signal CBL, CBLEN, CWL, and CSL. A clocked register is used to store the value read by the sense amplifier and is directly connected to the output PADs, read is used to isolate the sense from the programming step and $bit_in$ is the input of the XNOR operation. (b) The layout of the design where the different logic blocks are associated with their corresponding area.

Design 1 is made of an LFSR that allows verifying its operation for the complete design but also to estimate the difference in energy consumption between a system where the LFSR is quite far from the different memory arrays and a design where this random number generator is very close to the useful data.

Design 2, on the other hand, is a design that has the digital popcount operation that sums the output bits of the XNOR operation. This design is a basic element of a Binarized Neural Network which will be described in the next Chapter 4.

## 3.7 Conclusion

The design that we have implemented using RRAM technologies is relatively simple, as it features only 1024 bits of RRAM. Nevertheless, it is a first basic structure that could be scaled up to a much larger scale if the tests prove to be conclusive. We hope that by testing this chip, we

will obtain significant energy results. This work has also allowed us to set up a design flow for future implementations.

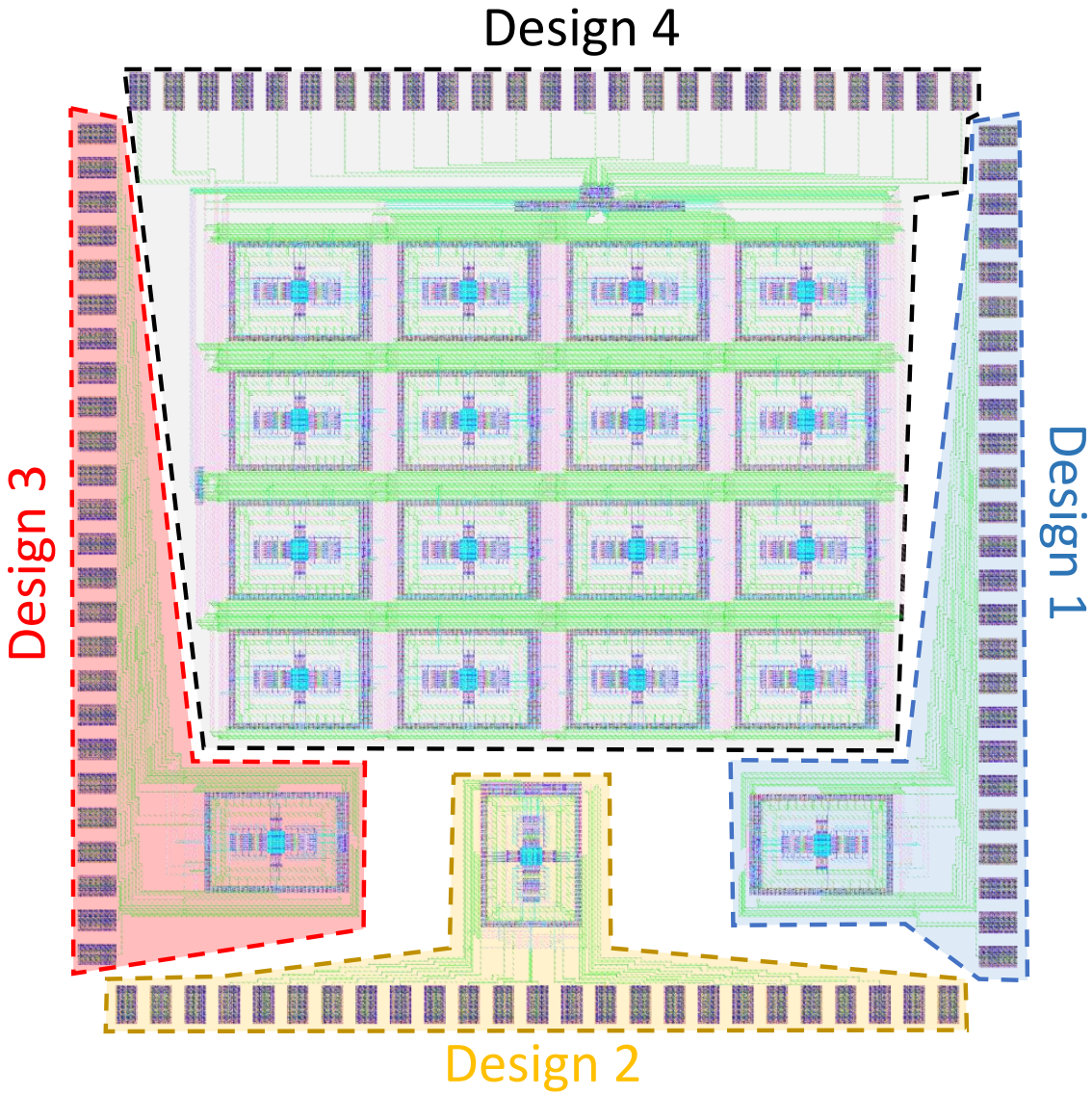



Figure 3.22: Final Layout, each design has is corresponding PADS, 25 for each, they will be tested independently. All designs have a special interest, but design 3 is mainly a control test design for all the others. Designs 1 and 2 are complementary designs that allow to check some digital functions and design 4 is the most interesting design implementing the Bayesian Machine.

For the design of this chip, we have worked on innovative paradigms: in-memory-computing, stochastic computing... It is not certain from an algorithmic point of view that each of our design choices are the best options. We are convinced of the interest of in-memory computing but we should remember that some of our signals in the implementation are not local, many signals travel a long distance to reach their final destination. This is the case for memory ad-

dressing signals but it is a relatively not very frequent signal, the random numbers generated are a real problem, they are propagated over a whole column and require a lot of wires.

It would be preferable, in an approach that would gather memory and computation, to have random number generators within each cell. The reason we did not do it is that the seed of the LFSRs is difficult to address. By using random number generators with memory components [213–215], we could do without seed programming and we would have TRNGs instead of periodic random number generators.

A more fundamental problem is the clock signal, it is the most important signal in digital circuits, but our design seems to only need this clock very superficially. By using the asynchronous paradigm [28], we could implement the same circuit but reduce the circuit connectivity and therefore reduce the power consumption.

The stochastic approach is really interesting when we look at the complexity of the digital circuits needed to do operations. However, when we designed our system we quickly realized that the logic circuits are not dominant in terms of area. Moreover, stochastic implementations are relatively imprecise and timing dilution is a big challenge. An alternate road could be to do approximate computing using logarithmic probability coding rather than floating-point coding, which makes product operations very simple: they become fixed point additions which are very simple, low power, and surface consuming. An implementation of such a system could be very interesting to compare the advantages and drawbacks of stochastic computation against a more accurate coding scheme.

# Chapter 4

# Digital Biologically Plausible Implementation of Binarized Neural Networks With Differential Hafnium Oxide Resistive Memory Arrays

> I think the brain is essentially a computer and consciousness is like a computer program. It will cease to run when the computer is turned off. Theoretically, it could be re-created on a neural network, but that would be very difficult, as it would require all one's memories.
>
> Stephen HAWKING

"THE BRAIN *performs intelligent tasks with extremely low energy consumption. This work, published as [225, 226], and reproduced here, takes inspiration from two strategies used by the brain to achieve this energy efficiency: the absence of separation between computing and memory functions, and the reliance on low precision computation. The emergence of resistive memory technologies indeed provides an opportunity to co-integrate tightly logic and memory in hardware. In parallel, the recently proposed concept of Binarized Neural Network, where multiplications are replaced by exclusive NOR (XNOR) logic gates, offers a way to implement artificial intelligence using very low precision computation. We therefore propose a strategy to implement low energy Binarized Neural Networks, while retaining energy benefits from digital electronics. We design, fabricate and test a memory array, including periphery and sensing circuits, optimized for this in-memory computing scheme. Our circuit employs hafnium oxide resistive memory integrated in the back end of line of a 130 nanometer CMOS process, in a two transistors - two resistors cell, which allows performing the exclusive NOR operations of the neural network directly within the sense amplifiers. We show, based on extensive electrical measurements, that our design allows reducing the amount of bit errors on the synaptic weights, without the use of formal error correcting codes. We design a whole system using this memory array. We show on standard machine learning tasks (MNIST, CIFAR-10, ImageNet and an ECG task) that the system has an inherent resilience to bit errors. We evidence that its energy consumption is attractive compared to more standard approaches, and that it can use the memory devices in regimes where they exhibit particularly low programming energy and high endurance. However, such neural networks are generally not entirely binarized: their first layer remains with fixed point input. This appears to be a challenge for low-energy hardware implementation. For this reason, in this chapter, we also propose a stochastic computing version of Binarized Neural Networks, where the inputs are also binarized. Simulations on the example of the Fashion-MNIST and CIFAR-10 datasets show that such networks can approach the performance of conventional Binarized Neural Networks. We conclude the work by discussing how it associates biologically-plausible ideas with more traditional digital electronics concepts. "*

As presented in Chapter 1, through deep learning, artificial intelligence has made tremendous achievements in recent years. Its energy consumption on graphics or central processing units (GPUs and CPUs) remains, however, a considerable challenge, limiting its use at the edge and raising the question of the sustainability of large scale artificial intelligence-based services. Artificial intelligence algorithms indeed require large amounts of memory access, which consume most of the energy compared to actual arithmetic operations [227]. Brains, by contrast, manage intelligent tasks with highly reduced energy usage by colocating neurons – which implement most of the arithmetic – and synapses – which are believed to store long term memory –. In Chapter 1, we have seen that, in the literature, researchers imitate this strategy and design non-von Neumann systems where logic and memory are merged [228–231], especially with the advent of novel nanotechnology-based non-volatile memories [124, 228, 232–236].

Moreover, in brains, most of the computation is done in a low precision analog fashion within the neurons [237, 238], resulting in asynchronous spikes as an output, which is therefore binary. Therefore, in this Chapter, we also explored a second idea for cutting the energy consumption of artificial intelligence, which is to design systems that operate with low precision computation.

In Chapter 1, we presented how neural network computation can be done using analog electronics: weight/neuron multiplication can be performed out of Ohm's law, and addition can be natively implemented with Kirchoff's current law [124, 125, 232, 233, 239, 240]. But a challenge of this implementation is that it requires to be associated with relatively heavy analog or mixed-signal CMOS circuitry such as an operational amplifier or Analog to Digital Converters, resulting in significant area and energy overhead.

In parallel, the novel class of – Binarized Neural Networks (or the closely related XNOR-NETs) [179, 241] – that we presented in Chapter 2 have limited memory requirements, and also rely on highly simplified arithmetics –multiplications are replaced by one-bit exclusive NOR (XNOR) operations–. Despite this extremely low precision, Binarized Neural Network can achieve near state of the art performance on vision tasks [179, 241, 242], and are therefore extremely attractive for realizing inference hardware.

Moreover, the binary nature of neurons – which is reminiscent of biological neurons spikes – also endows them with biological plausibility: they can indeed be seen as a simplification of spiking neural networks.

Great efforts have been devoted to developing hardware implementations of Binarized Neural Networks. Using nanodevices, one natural intuition would be to adopt the same strategy proposed for conventional neural networks presented in Chapter 2, where arithmetic is performed in an analog fashion using Kirchoff's law [228, 243]. However, Binarized Neural Networks are very digital in nature and are multiplication-less. These networks can therefore provide an opportunity to benefit at the same time from bioinspired ideas and from the achievements of Moore's law and digital electronics. Here we propose a fully digital implementation of

binarized neural networks incorporating CMOS and nanodevices, and implementing the biological concepts of tight memory and logic integration, as well as low precision computing. As memory nanodevices, we use hafnium oxide-based resistive random access memory (OxRAM), a compact and fast non-volatile memory cell fully compatible with the CMOS process [244].

However, one significant challenge to implement a digital system with memory nanodevices is their inherent variability [245, 246], which causes bit errors. Traditional memory applications employ multiple error correcting codes (ECCs) to solve this issue. ECC decoding circuits have a large area and high energy consumption [247], and add extra time to data access due to syndrome computation and comparison. Moreover, the arithmetic operations of error syndrome computation are actually more complicated than those of a Binarized Neural Network. This solution is difficult to implement in a context where memory and logic are tightly integrated especially when part of the computation is performed during sensing. It is one of the main reason for which the state of the art of RRAM for in-memory computing does not correct errors and is not compatible with technologies with errors [248, 249].

In this chapter, we introduce our solution. My main contribution of this work was on the algorithmic part concerning the implementation of neural networks with a study of their intrinsic resilience to error, and the design at the system level of a neural network chip using non-volatile memory. On the other hand, Jean-Michel Portal was in charge of the design of the test chip composed of the differential oxide-based resistive memory array, including all peripheral and sensing circuitry. E. N. and E.V. managed the fabrication of the die at CEA-Leti. The intensive test of the chip was performed by Marc Bocquet.

## 4.1 Differential Memory Array for In-Memory Computing

The memory cell used in this chapter, relies on hafnium oxide ($HfO_2$) oxide-based Resistive Random Access Memory (OxRAM). The main strengths and weakness of RRAM technology were mentioned in Chapter 2. The stack of the device is composed of a $HfO_2$ layer and a titanium layer. Both layers have a thickness of ten nanometers, and they grow between two titanium nitride (TiN) electrodes. Our devices are embedded within the back-end-of-line of a commercial 130 nanometer CMOS logic process (Figure 4.1(a)), allowing tight integration of logic and non volatile memory [244]. The devices are integrated on top of the fourth (copper) metallic layer.

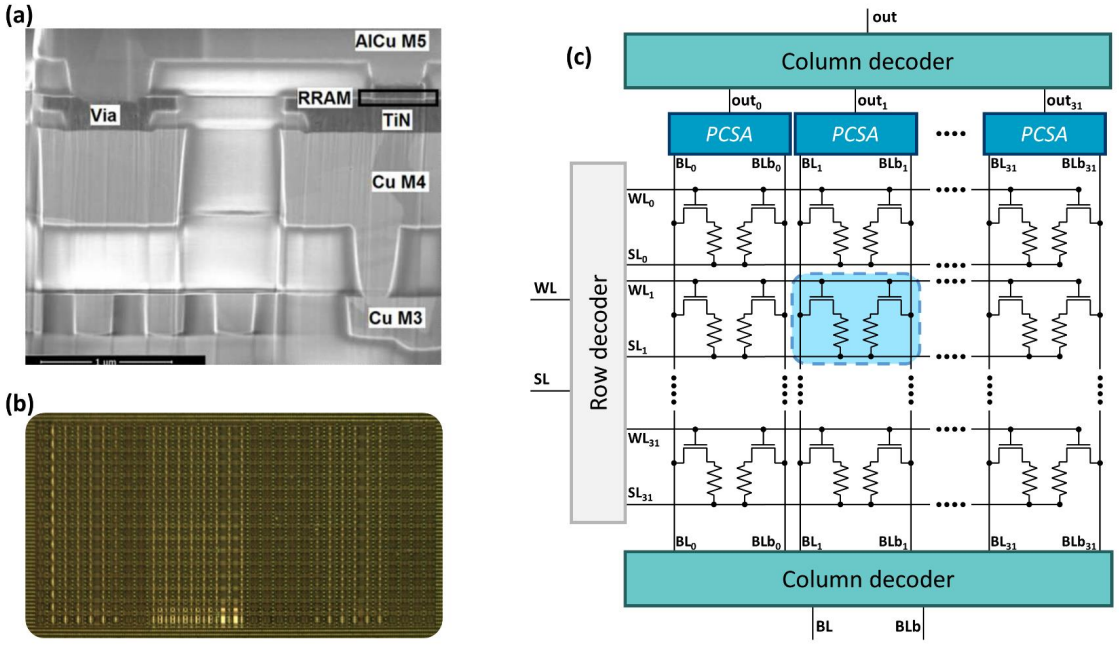



Figure 4.1: (a) Scanning Electron Microscopy image of the back-end-of-line of the CMOS process integrating an OxRAM device. (b) Photograph and (c) simplified schematic of the one kilobit in-memory computing-targeted memory array characterized in this work.

Hafnium oxide OxRAMs are known to provide non-volatile memories compatible with modern CMOS process, and only involve foundry-friendly materials and process steps.

After a one-time forming process, such devices can switch between low resistance and high resistance states (LRS and HRS) by applying positive or negative electrical pulses respectively.

Our work could be reproduced with other types of memories. NOR flash cells, which are readily available in commercial process could be used, and their potential for neuromorphic inference has been proven [250]. However, they suffer from high programming voltages (higher

than ten volts) requiring charge pumps, limited endurance, and they are not scalable to the most advanced technology nodes [251]. Emerging memories such as phase change memory or spin torque magnetoresitive memory could also be used using the strategies presented in this Chapter. These technologies do not require a forming process and they can bring enhanced reliability with regards to OxRAMs, but come with an increased process cost [252].

Conventionally, OxRAMs are organized in a "One Transistor - One Resistor" structure (1T1R), where each nanodevice is associated with one access transistor [252]. The LRS and HRS are used to mean the zero and one logic values, or inversely. The read operation is then achieved by comparing the electrical resistance of the nanodevice to a reference value intermediate between typical values of resistances in HRS and LRS. Unfortunately, due to device variability, OxRAMs are prone to bit errors: the HRS value can become lower than the reference resistance, and the LRS value can be higher than the reference resistance. The device variability includes both device-to-device mismatch, as well as the fact that within the same device, the precise value of HRS and LRS resistance changes at each programming cycle [253].

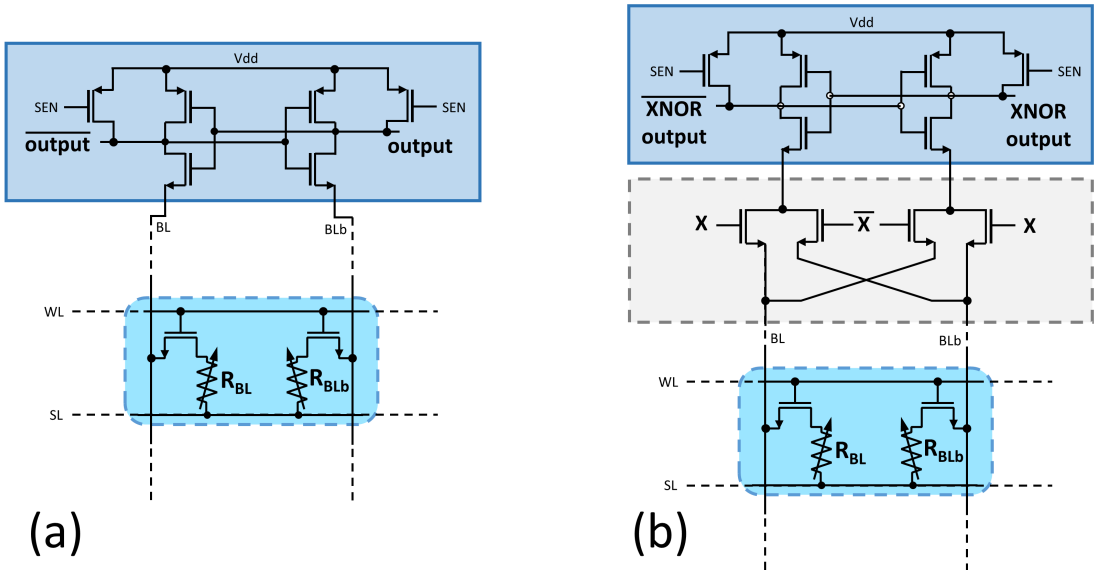



Figure 4.2: (a) Schematic of the precharge sense amplifier used in this work to read 2T2R memory cells. (b) Schematic of the precharge sense amplifier augmented with a XNOR logic operation.

To limit the amount of bit errors, in this work, we fabricated a memory array with a "Two Transistors - Two Resistors" structure (2T2R), where each bit of information is stored in a pair of 1T1R structures. A photograph of the die is presented in Figure 4.1(b) and its simplified schematic in Figure 4.1(c). Information is stored in a differential fashion: the pair LRS/HRS means logic value zero, while the pair HRS/LRS means logic value one. In this situation, readout is performed by comparing the resistance values of the two devices. We therefore expect bit errors to be less frequent, as a bit error only occurs if a device programmed in LRS is more resis-

tive than its complementary device programmed in HRS. This concept of 2T2R memory arrays has already been proposed, but its benefit in terms of bit error rate has never been demonstrated until this work [254, 255].

The programming of devices in our array is made sequentially, i.e. on a device-by-device basis. The first time that the memory array is used, all devices are "formed". To form the device of row $i$ and column $j$, the bit line $BL_j$, connected to the bottom electrode of the memory device, is set to ground, and the word line $WL_i$ is set to a voltage chosen to limit the current to a "compliance value" of $200\mu A$. A voltage ramp is applied to the sense line $SL_i$ connected to the top electrode of the memory device, increasing from 0 to $3.3V$ at a ramp rate of $1000V/s$. This forming operation is performed only once over the lifetime of the device. To program a device into its LRS (SET operation), the bit line $BL_j$ is set to ground, while the sense line $SL_i$ is set to $2V$. The word line $WL_i$ is again set to a voltage chosen to limit the current to a compliance value, ranging from $20\mu A$ to $200\mu A$, depending on the chosen programming condition. To program a device into its HRS (RESET operation), a voltage from opposite sign needs to be applied to the device, and the current compliance is not needed. The sense line $SL_i$ is therefore set to the ground, while the word line $WL_i$ is set to a value of $3.3V$, and the bit line $BL_j$ to a "RESET voltage" ranging from $1.5V$ to $2.5V$, depending on the chosen programming condition. For both SET and RESET operations, programming duration can range from $0.1\mu s$ to $100\mu s$. During programming operations, all bit, select and word lines corresponding to non-selected devices are grounded, to the exception of the bit line of the complementary device of the selected device: this one is programmed to the same voltage as the one applied to the sense line, to avoid any disturb effect on the complementary device.

In our fabricated circuit, the readout operation is performed with precharge sense amplifiers (PCSA) [65, 256] (Figure 4.2(a)), the same read circuits as used in chapter 3. These circuits are highly energy efficient due to their operation in two phases, precharge and discharge, avoiding any direct path between supply voltage and ground. First, the sense signal (SEN) is set to ground and SL to the supply voltage, which precharges the two selected complementary nanodevices as well as the comparing latch at the same voltage. In the second phase, the sense signal is set to the supply voltage, and the voltages on the complementary devices are discharged to ground through SL. The branch with the lowest resistance discharges faster, and causes its associated inverter output to discharge to ground, which latches the complementary inverter output to the supply voltage. The two output voltages therefore represent the comparison of the two complementary resistance values. In our test chip, the read time is approximately 10 $ns$ and is due to the high capacitive load associated with our probe testing setup. Without this high capacitive load, the switching time would be determined by the time to resolve the initial metastability of the circuit. This switching time can be as fast as $100ps$ in a scaled technology [256].

We fabricated a differential memory with 2048 devices, therefore implementing a kilobit memory. Each column of complementary nanodevices features a precharge sense amplifier,

and row and columns are accessed through integrated CMOS digital decoders. The pads of the dies are not protected for electrostatic discharge, and the dies were tested with commercial 22-pads probe cards. In all the experiments, voltages are set using a home made printed circuit board, and pulses voltages are generated using Keysight B1530A pulse generators. In the design, the precharge sense amplifiers can optionally be deactivated and by-passed, which allows measuring the nanodevices resistance directly through external precision source monitor units (Keysight B1517a).

## 4.2 Design of In-Memory Binarized Neural Network Based on the Differential Memory Building Block

This work aims at implementing Binarized Neural Networks in hardware already mentionned in Chapter 2, section 2.2.3. In these neural networks, the synaptic weights, as well as the neuronal states, can take only two values, +1 and −1. The equation to compute is therefore:

$$A_j = \text{sign}\left(\underset{i}{\text{POPCOUNT}}\left(XNOR\left(W_{ji}, X_i\right)\right) - \mu_j\right). \tag{4.1}$$

In this equation, $\mu_j$ is the so called threshold of the neuron, and it is learned during training. POPCOUNT is the function that counts the number of ones in a series of bits, and sign is the sign function.

As explained in the second chapter, the training process of binarized neural networks differs from conventional neural networks, already presented in Algorithm 1. During training, the weights assume real weights in addition to the binary weights, which are equal to the sign of the real weights. Training employs the classical error backpropagation equations, with several adaptations. The binarized weights are used in the equations of both the forward and the backward passes, but the real weights change as a result of the learning rule [241]. Additionally, as the activation function of binarized neural networks is the sign function and it is not differentiable, we consider the sign function as the first approximation of the hardtanh function

$$\text{Hardtanh(x)} = \text{Clip(x}, -1, 1), \tag{4.2}$$

and we use of the derivative of this function as a replacement for the derivative of the sign function in the backward pass. This replacement is a key element for training Binarized neural Network successfully. The clip interval in equation (4.2) is not learned and is chosen to be between -1 and 1 for all neurons. Using a larger interval would indeed increase vanishing gradient effect, while using a smaller interval would lead to derivatives higher than one, which can cause exploding gradient effects.

Finally, the Adam optimizer is used to stabilize learning [183]. A technique known as batch-normalization is employed at each layer of the neural network [174]. Batch-normalization

shifts and scales the neuronal activations over a batch during the training process. This method is optionally used in conventional neural networks to accelerate and stabilizing learning. Using this technique becomes essential when training binarized neural networks to reach high accuracies, as it ensures that neuronal activations utilize both +1 and −1 value. At inference time, batch-normalization is no longer necessary and the threshold learned by this technique can be used directly as neuronal threshold in equation (4.1).

With this learning technique, binarized neural networks function surprisingly well. They can achieve near state of the art performance on image recognition tasks such as CIFAR-10 and ImageNet [242]. After learning, the real weights serve no more purpose and can be discarded. This makes binarized neural networks exceptional candidates for hardware implementation of neural network inference. Not only their memory requirements are minimal (one bit per neuron and synapse), but their arithmetic is also vastly simplified. Multiplication operations of classical neural network are expensive in terms of area and energy consumption, and they are replaced by one-bit exclusive NOR (XNOR) operations in eq. 4.1. Additionally, the real sums are replaced by POPCOUNT operations, which are equivalent to integer sums with a low bit width.

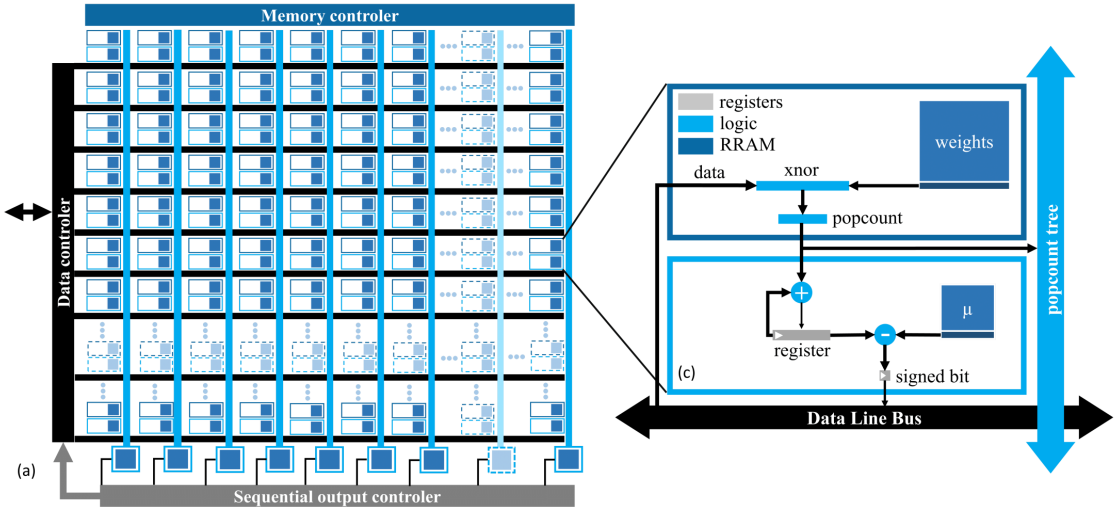



Figure 4.3: (a) Design of an RRAM based fully connected binarized neural network, computing in the "parallel to sequential" configuration. The system assembles a memory block surrounded by logic circuits and moves minimal data between the blocks. The architecture is presented with 32 rows and 32 columns of basic cells (b-c) that behaves as a neuron if the input is sequential, or each column behaves as a neuron if the input is parallel. In each basic cell, a kilobit memory block (i.e., n = 32) is used to store the weights and a smaller memory block is used to stored the threshold to compute the neurons output.

It is possible to implement ASIC Binarized Neural Networks with solely CMOS [121, 257]. However, a more optimal implementation can rely on emerging non-volatile memories, and associate logic and memory as closely as possible. This approach can provide non volatile

neural networks, and eliminate the von Neumann bottleneck entirely: the nanodevices can implement the synaptic weights, while the arithmetic can be done in CMOS. In the Chapter 2, we have seen that the literature proposes the use of emerging memories as synapses relying on analog electronics with an ingenious technique to perform the multiplications and additions: the multiplications are done with Ohm's law, and the addition with Kirchoff current law [124, 243]. This analog approach can be transposed directly to binarized neural networks [228, 258–260]. However, binarized neural networks are inherently digital objects that rely, as previously remarked, on simple logic operation: XNOR operations and low bit-width sums. Therefore, here, we investigate their implementation with purely digital circuitry. This concept has also recently appeared in [261, 262]. We are the first one to explore binarized neural network with measurements on a physical memory array, that includes the effect of bit errors.

A first realization is that the XNOR operations can be realized directly within the sense amplifiers. For this, we follow the pioneering works of [256], which shows that precharge sense amplifier can be enriched with any logic operation. In our case, we can add four additional transistors in the discharge branches of a precharge sense amplifier (Figure 4.2(b)). These transistors can prevent the discharge and allow implementing the XNOR operation between input voltage $X$ and the value stored in the complementary OxRAM devices in a single operation.

Based on the basic memory array with PCSAs enriched with XNOR, we designed the whole system implementing a Binarized Neural Network. The overall architecture is presented in Figure 4.3 (a). It is inspired by the purely CMOS architecture proposed in [121], adapted to the constrains of OxRAM. The design is made of the repetition of basic cells organized in a matrix of $N$ by $M$ cells. These basic cells presented in Figure 4.3 (b-c) incorporate a $n \times n$ OxRAM memory block with XNOR-enriched PCSAs, POPCOUNT logic and adds the threshold values $\mu$ in a memory array. The signed bit of the difference between the popcount value saved in the register and $\mu$ gave the activation value. The whole system, which aims at computing the activation of neurons (equation (4.1)), features a degree of reconfigurability to adapt to different neural network topologies: it can be used either in "parallel to sequential" or in a "sequential to parallel" configuration.

The parallel to sequential configuration can deal with layers with up to $n \times N$ input neurons, and up to $n \times M$ output neurons. In this situation, at each clock cycle, the system computes the activations of $M$ output neurons in parallel. At each clock cycle, each basic-cell reads an entire row of its OxRAM memory array, while performing the XNOR operation with input neuron values. The results are used to compute the POPCOUNT operation over a subset of the indices $i$ in equation (4.1), using fully digital five bits counters embedded within the cell. Additional logic, called "popcount tree"and only activated in this configuration, computes the full POPCOUNT value operation over a column by successively adding the five bits-wide partial POPCOUNT values. The activation value of the neuron is obtained by subtracting the complete POPCOUNT value at the bottom of the column with a threshold value, stored in a separate memory array; the signed bit of the result gives the activation value. At the next clock cycle, the next rows in the

OxRAM memory arrays are selected, and the activations of the next $M$ neurons are computed.

The sequential to parallel configuration, by contrast, can be chosen to deal with a neural network layer with up to $n^2$ inputs neurons, and up to $NM$ output neurons. In this configuration, each basic cell of the system computes the activation of one neuron $A_j$. The input neurons $X_i$ are presented sequentially, by subsets of $n$ inputs. At each clock cycle, the digital circuitry therefore computes only a part of equation (4.1). The partial POPCOUNT is looped to the same cell to compute the whole POPCOUNT sequentially After the presentations of all inputs, the threshold is subtracted, the binary activation is extracted and equation (4.1) has been entirely computed.

This whole system was designed using synthesizable SystemVerilog. The memory blocks are described in behavioral SystemVerilog. We synthesized the system using the 130 nanometer design kit used for fabrication, as well as using the design kit of an advanced commercial 28 nm process for scaling projection.

All simulations reported in the section 4.5 were performed using the Cadence Incisive simulators. The estimates for system-level energy consumption were obtained using the Cadence Encounter tool. We used Value Change Dumps (VCD) files extracted from simulations of practical tasks so that the obtained energy values reflect the operation of the system realistically.

## 4.3 Differential Memory Allows Memory Operation at Reduced Bit Error Rate

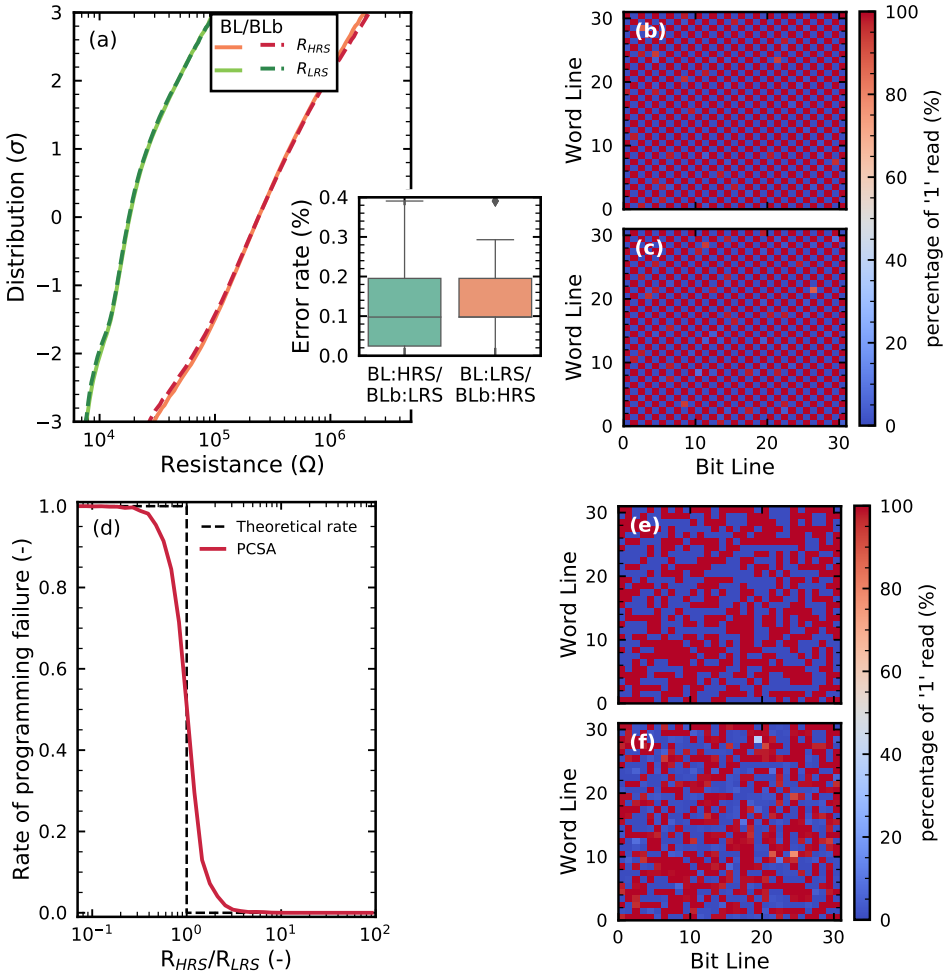



Figure 4.4: (a) Distribution of the LRS and the HRS of the OxRAM devices in an array programmed with a checkerboard pattern. RESET voltage of $2.5V$, SET current of $55\mu A$ and programming time of $1\mu s$. (b-c) Proportion of 1 values read by the onchip precharge sense amplifier, over 100 whole-array programming of a memory array, for the two complementary checkerboards configuration. (d) Rate of programming failure indicated of the precharge sense amplifier circuits as a function of the ratio between HRS and LRS resistance (measured by a sense measure unit), in the same configuration as (a-c). (e-f) Proportion of 1 values read by the onchip precharge sense amplifier, over 100 whole-array programming of a memory array, for the last layer of a binarized neural network trained on MNIST (details in body text)

This section first presents the electrical characterization results of the differential OxRAM arrays. We program the array with checkerboard-type data, alternating zero and one bits, using

programming times of one microsecond. For programming devices in HRS (RESET operation), the access transistor is fully opened and a reset voltage of 2.5$V$ is used.

For programming devices in LRS (SET operation), the gate voltage of the access transistor is chosen to ensure a compliance current of 55$\mu A$. Figure 4.4(a) shows the statistical distribution of the LRS and HRS of the cells, based on 100 programmings of the full array. This graph is using a standard representation in the memory field, where the $y$ axis is expressed as number of standard deviations of the distribution [246]. The Figure superimposes distributions of left (BL) and right (BLb) columns of the array, and no significant difference is seen between BL and BLb devices.

The LRS and HRS distributions are clearly separate but overlap at a value of three standard deviations, which makes bit errors possible. If a 1T1R structure was used, a bit error rate of 0.012 (1.2%) would be seen with this distribution. By contrast, at the output of the precharge sense amplifiers, a bit error rate of 0.002 (0.2%) is seen, giving a first suggestion of the benefits of the 2T2R approach. Figure 4.4(b) and 4.4(c) show the mean error (using the 2T2R configuration) on the whole array, for the two types of checkerboards. We see that all devices can be programmed in HRS and LRS. A few devices have increased bit error rate. This graph highlights the existence of both cycle to cycle and device to device variability, and the absence of "dead" cells.

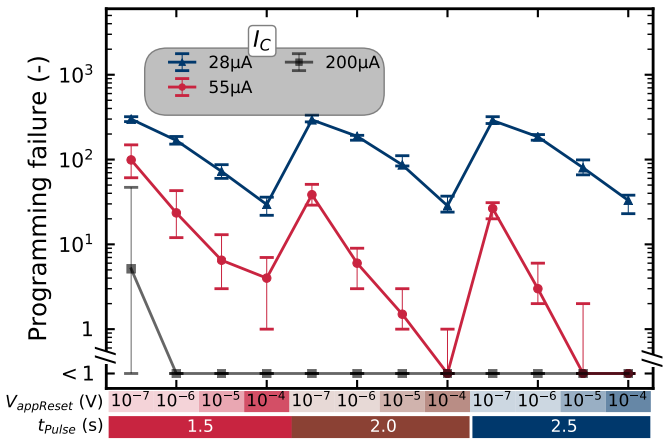



Figure 4.5: Number of errors for different programming conditions, as measured by the precharge sense amplifier, for 2T2R configuration on a kilobit memory array.The "< 1" label means that no errors were detected. The error bars present the minimum and maximum number of detected errors, over five repetitions of the experiments.

We now validate in detail the functionality of the precharge sense amplifiers. The precise resistance of devices is first measured by deactivating the precharge sense amplifiers, and using the external source monitor units. Then, the precharge sense amplifiers are reactivated and a sense operation is performed. Figure 4.4(d) plots the mean measurement of the sense amplifiers as a function of the ratio between the two resistances that are being compared, su-

perimposed with the ideal behavior of a sense amplifier. The sense amplifiers show excellent functionality, but can make mistakes if the two resistances differ by less than a factor five. Finally, Figs. 4.4(e-f) repeat the experiments of 4.4(b-c) in a more realistic situation and on a different die. We trained a memory array 100 times with weights corresponding to the last layer of a binarized neural network trained on the MNIST task of handwritten digit recognition. As in the checkerboard case, no dead cell is seen, and a similar degree of cycle-to-cycle and device-to-device variation is seen.

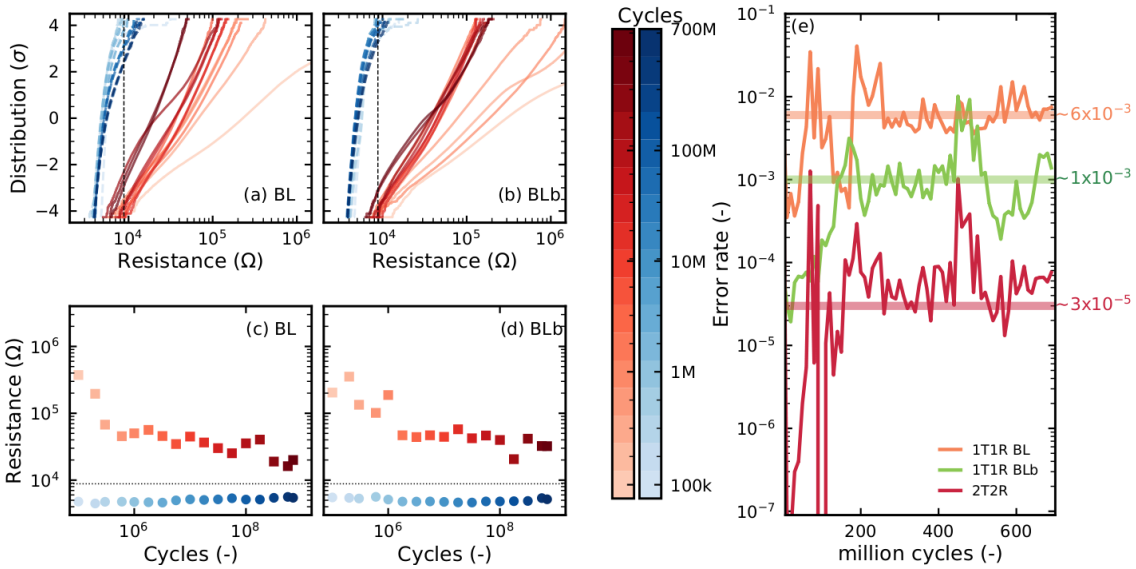



Figure 4.6: (a-b) Distribution of the resistance values, (c-d) mean resistance value and (e) mean bit error rate over 10 million cycles measured by the precharge sense amplifier, in the 2T2R configuration, as function of the number of cycles that a device has been programmed. RESET voltage of $2.5V$, SET current of $200\mu A$ and programming time of $1\mu s$.

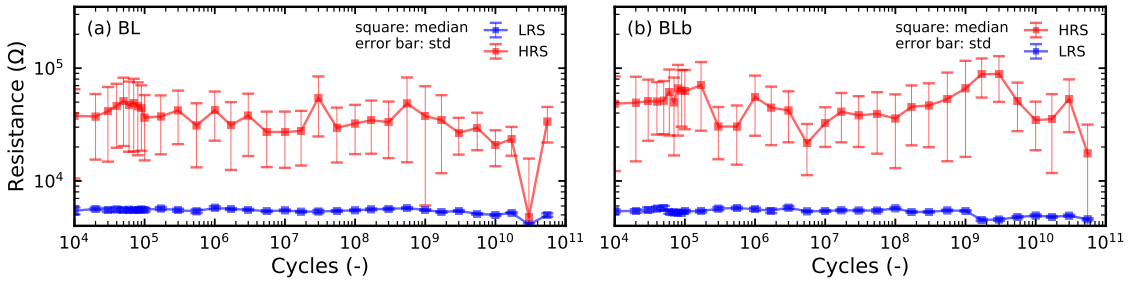



Figure 4.7: (a-b) Mean resistance value of the BL and BLb device over 10 thousand cycles for measurements of a device pair over $5 \times 10^{10}$ cycles. RESET voltage of $1.5V$, SET current of $200\mu A$ and programming time of $1\mu s$.

The programming rates are strongly dependent on the programming conditions. Figure 4.5

shows the mean number of incorrect bits on a whole array for various combination of programming times (from 0.1$\mu s$ to 100$\mu s$), RESET voltage (between 1.5 and 2.5 Volts), and SET compliance current (between 28 and 200$\mu A$). We observe that the bit error rate depends extensively on these three programming parameters, the SET compliance current having the most significant impact.

In Figure 4.6, we look more precisely at the effects of cycle to cycle device variability and device aging. A device and its complementary device were programmed 700 million cycles. Figs. 4.6(a) and 4.6(b) show the distribution of LRS and HRS of the device under test and its complementary device, after different number of cycles ranging from the first one to the last one.

We can observe that when the devices are cycled, LRS and HRS distributions become less separated and start to overlap at lower number of standard deviations. This translates directly on the mean resistance of the devices in HRS and LRS (Figs. 4.6(c) and 4.6(d)), which become closer when the device ages. More importantly, the aging process impacts the device bit error rate (Figs. 4.6(e)): the bit error rate of the device and its complementary device increase of several orders of magnitudes over the lifetime of the device. The same effect is seen on the bit error rate resulting from the precharge sense amplifier (2T2R), but it remains at much lower level: while the 1T1R bit error rate goes above $10^{-3}$ after a few million cycles, the 2T2R remains below this value over the 700 million cycles. This result highlights that the concept of cyclability depends on the acceptable bit error rate, and that the cyclability at constant bit error rate can be considerably extended when using the 2T2R structure. It should also be highlighted that the cyclability depends tremendously on the programming condition. Figure 4.7(a-b)) shows endurance measurements with a reset voltage of 1.5$V$ (all other programming conditions are identical to Figure 4.6(a-e)). We can see that the device experiences no degradation along more than ten billion cycles. Over that time, the 2T2R bit error rates remains below $10^{-4}$.

We now aim at quantifying and benchmarking more precisely the benefits of the 2T2R structure. We performed extensive characterization of bit error rates on the memory array in various regimes. Figure 4.8(a) presents different experiments where the 2T2R bit error rate is plotted as a function of the bit error rate that would be obtained using using a single device programmed in the same conditions. The different points are obtained by varying the compliance current Ic during SET operations, and the graph associates two type of experiments:

- The points marked as “Low Ic” are obtained using whole array measurement where devices are programmed with low SET compliance current to ensure high error rate. Each device in the memory array is programmed once (following the checkerboard configuration), and all synaptic weights are read using the on-chip precharge sense amplifiers. The plotted bit error rate is the proportion of weights for which the read weight differs from the weight value targeted by the programming operation.

- The points marked as “High Ic” are obtained by measurements on a single device pair.

A single 2T2R structure in the array is programmed ten million times by alternating programming to +1 and −1 values. The value programmed in the 2T2R structure is read using an on-chip precharge sense amplifier after each programming operation. The plotted bit error rate is the proportion of read operation for which the read weight differs from the targeted value.

We can see that the 2T2R bit error rate is always lower than the 1T1R one. The difference is larger for lower bit error rate, and reaches four order of magnitudes for a 2T2R bit error rate of $10^{-8}$. The black line presents calculation where the precharge sense amplifier is supposed to be ideal (*i.e.* to follow the idealized dotted characteristics of Figure 4.4(c)).

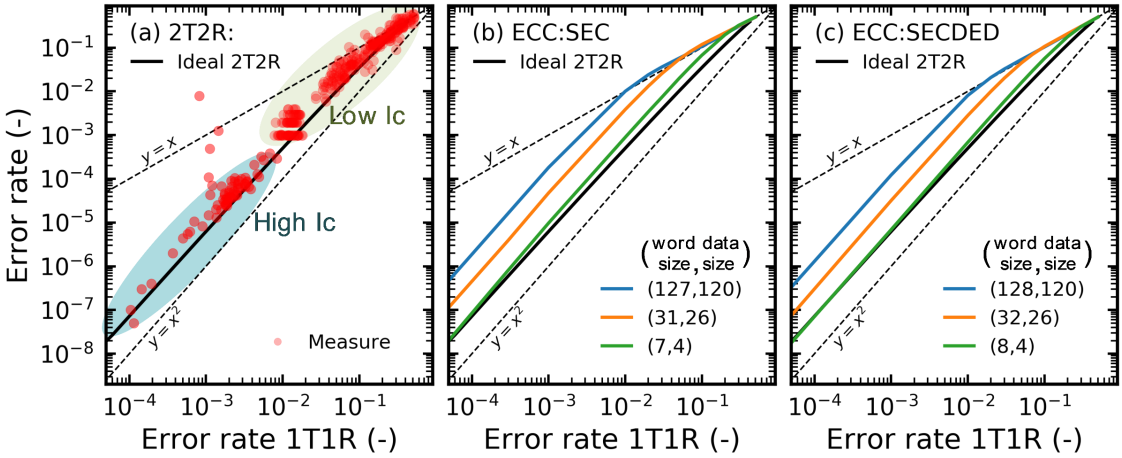



Figure 4.8: Experimental bit error rate of the 2T2R array, measured by the precharge sense amplifiers, as a function of the bit error rate obtained individual (1T1R) RRAM devices in the same programming conditions. The detailed methodology for obtaining this graph is presented in the body text. Bit error rate obtained with (b) Single Error Correcting (SEC) and (c) Single Error Correcting Double Error Detection (SECDED) ECC as a function of the error rate on the individual devices.

To interpret the results of the 2T2R approach with more perspective, we benchmark them with standard error correcting codes. Figs. 4.8(b) and 4.8(c) show the benefits of two codes, using the same plotting format as Figs. 4.8(a): a Single Error Correction (SEC) and a Single Error Correction Double Error Detection (SECDED) code, presented with different degrees of redundancy. These simple codes, formally known as Hamming and extended Hamming codes, are widely used in the memory field. Interestingly, we see that the benefit of these codes are very similar to the benefit of our 2T2R approach with ideal sense amplifier, at equivalent memory redundancy (*e.g.* SECDED(8,4)), although our approach uses no decoding circuit and the equivalent of error correction is performed directly within the sense amplifier. By contrast, ECCs can also reduce bit errors, to a lesser extent, using less redundancy, but the required decoding circuits utilize hundreds to thousands of logic gates [247]. In a context where logic and

memory are tightly integrated, these decoding circuits would need to be repeated many times, and as their logic is much more complicated than the one of binarized neural networks, they would be the dominant source of computation and energy consumption. ECC circuits are also incompatible with the idea of integrating XNOR operations within the sense amplifiers, and cause important read latency.

## 4.4 Do All Errors Need to Be Corrected?

### 4.4.1 Impact of Errors on Binarized neural Network performances

Based on the results of the electrical measurements, and before discussing the whole system, it is important to know the OxRAM bit error rate levels that can be tolerated for applications. To answer this question, we performed simulations of binarized neural networks on four different tasks:

- MNIST handwritten digit classification [162], the canonical task of machine learning. We use a fully connected neural network with two 1024-neurons hidden layers.
- The CIFAR-10 image recognition task [263], which consists in recognizing $32 \times 32$ color images spread between ten categories of vehicles and animals. We use a deep convolutional network with six convolutional layers using kernels of $3 \times 3$ and a stride of one, followed by three fully connected layers.
- The ImageNet recognition task, which consists in recognizing $224 \times 224$ color images out of 1000 classes. This task is considerably more difficult than MNIST and CIFAR-10. We use the historic AlexNet deep convolutional neural network [163].
- A medical task involving the analysis of electrocardiography (ECG) signals: automatic detection of electrode misplacement. This binary classification challenge takes as input the ECG signals of twelve electrodes. The experimental trial data are sampled at $250Hz$ and have a duration of three seconds each. To solve this task, we employ a convolutional neural network composed of five convolutional layers and two fully-connected layers. The convolutional kernel (sliding window) sizes are decreasing from 13 to 5 in each subsequent layer. Each convolutional layer produce 64 filters detecting different features of the signal.

Fully binarized neural networks were trained on these tasks on Nvidia Tesla GPUs using Python and the PyTorch deep learning framework. Once the neural networks were trained, we ran them on the datasets validation sets, artificially introducing errors in the neural networks weights (meaning some $+1$ weights are replaced by $-1$ weights, and reciprocally). Using this technique, we can emulate the impact of OxRAM bit errors. Figure 4.9 shows the resulting

validation accuracy as a function of the introduced bit error rate for the four considered tasks. In the case of ImageNet, both the Top-1 (proportion of validation images where the right label is the top choice of the neural network) and the Top-5 (proportion of validation images where the right label is within the top five choices of the neural network).

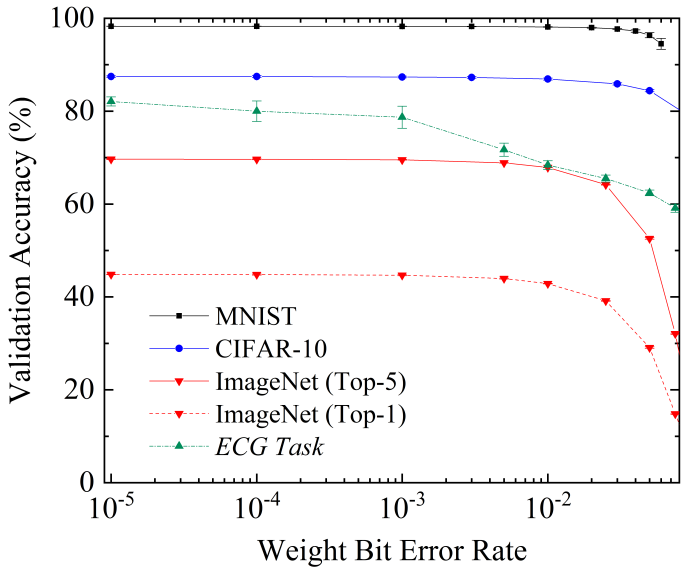



Figure 4.9: Recognition rate on the validation dataset of the fully connected neural network for MNIST, the convolutional neural network for CIFAR10, and AlexNet for ImageNet (Top-5 and Top-1) accuracies and the ECG analysis task, as a function of the bit error rate over the weights during inference. Each experiment was repeated five times, the mean recognition rate is presented. Error bars represent one standard deviation.

On the three vision task (MNIST, CIFAR-10, ImageNet), we see that extremely high levels of bit errors can be tolerated: up to a bit error rate of $10^{-4}$, the network performs as well as with no errors. Minimal performance reduction starts to be seen with bit error rates of $10^{-3}$ (the Top-5 accuracy on ImageNet is degraded from 69.7% to 69.5%). At bit error rates of 0.01, the performance reduction becomes significant. The reduction is more substantial for harder tasks: MNIST accuracy is only degraded from 98.3% to 98.1%, CIFAR-10 accuracy is degraded from 87.5% to 86.9%, while ImageNet Top-5 accuracy is degraded from 69.7% to 67.9%.

The ECG tasks also shows extremely high bit error tolerance, but bit errors have an effect more rapidly than in the vision tasks. At a bit error rate of $10^{-3}$, the validation accuracy is reduced from 82.1% to 78.7%, and at a bit error rate of 0.01 to 68.4%. This difference between vision and ECG tasks probably originates in the fact that ECG signals carry a lot less redundant information than images. Nevertheless, we see that even for ECG tasks high bit errors rates can be accepted with regards to the standards of conventional digital electronics.

### 4.4.2 Reducing Error Impact using adapted learning

The already high robustness seen in Fig. 4.9 can be further enhanced if an appropriate training method is used. These results were published in [264] For this purpose, we retrain the BNNs, but this time including bit errors *during the training process*, and not only during the testing phase. This way, the training process takes into account the fact that the BNN will be implemented on error-prone RRAM-based systems. The devices subject to errors are chosen independently in training and testing phase: the training phase assumes that the RRAM-based systems will have errors, but does not know which devices will be affected.

More precisely, to train the neural network, we added errors at each iteration. The forward-pass is computed with errors on weights, e.g. 10% are changed from the original weights $W$ to $W_{error}$. During the backward-pass, we reuse the same value of weights $W_{error}$ instead of $W$ to backpropagate through the whole depth of the neural network.

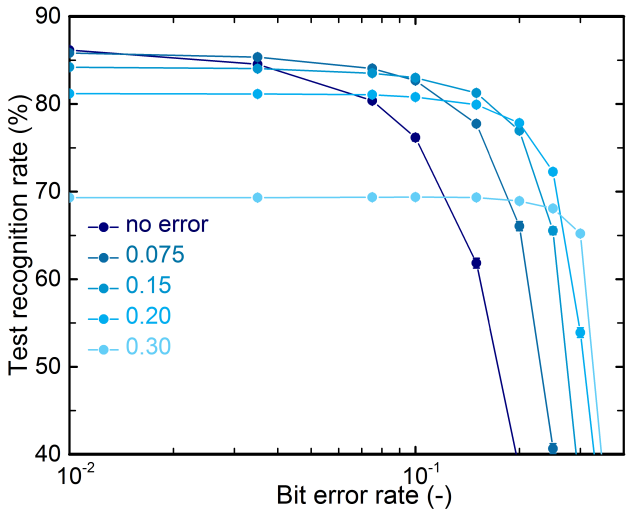



Figure 4.10: Recognition rate on the test dataset of the convolutional neural network for CIFAR10 as a function of the bit error rate over the weights during inference. Navy curve: no weight error considered during training. Other curves: the adapted training was used, each curve corresponding to a different bit error rate on the weights during training.

Fig. 4.10 shows the results of the same study on the CIFAR10 dataset. Using this procedure, extremely high amount of bit errors can be tolerated. Bit error rates up to $4 \times 10^{-2}$ do not affect the recognition rate. For a bit error rate of 0.15, the recognition rate is 81.5%, instead of 62.2% if errors had not been taken into account during training (navy line).

When using RRAM devices, using the adapted training procedure therefore allows use weak programming conditions despite its high bit error rate. This can allow us to benefit from its low programming energy, cell area and high endurance.

## 4.5 Projection at the System Level

### 4.5.1 Impact of In-Memory Computation

We now use all the chapter results to discuss the potentials of our approach. Based on our ASIC design, using the energy evaluation technique described in section 4.2, we find that our system would consume 25 $nJ$ to recognize one handwritten digit, using a fully connected neural networks with two hidden layers of 1024 neurons. This is considerably less than processor based options. For example, [265] analyses the energy consumption of inference on CPUs and GPUs: operating a fully connected neural network with two hidden layers of 1000 neurons requires 7 to 100 millijoules on a low power CPU (from Nvidia Tegra K1 or Qualcomm Snapdragon 800 systems on chip), and 1.3 millijoules on a low power GPU (Nvidia Tegra K1).

These results are not surprising due to the considerable overhead for accessing memory in modern computers. For example, in Chapter 1 we showed that accessing data in a static RAM cache consumes around fifty times more energy than the integer addition of this data. If the data is stored in the external dynamic RAM, the ratio is increased to more than 3000. Binarized Neural Networks require minimal arithmetic: no multiplication, and only integer addition with a low bit width. When operating a Binarized Neural Networks on a CPU or GPU, the almost entirely of the energy is used to move data, and the inherent topology of the neural network is not exploited to reduce data movement. Switching to in-memory or near-memory computing approaches has therefore the potential to reduce energy consumption drastically for such tasks. This is especially true as, in inference hardware, synaptic weights are static and can be programmed to memory only once if the circuit does not need to change function.

### 4.5.2 Impact of Binarization

We now look specifically to the benefits of relying on Binarized Neural Networks rather than real-valued digital ones. Binarized Neural Networks feature considerably simpler architecture than conventional neural network, but also require an increased number of neurons and synapses to achieve equivalent accuracy. It is therefore essential to confront the binarized and real-values approaches.

Most digital ASIC implementations of neural networks inference function with eight-bit fixed point arithmetic, the most famous example being the tensor processing units developed by Google [116]. At this precision, no degradation is usually seen for inference with regards to 32 and 64 bits floating point arithmetic.

To investigate the benefits of Binarized Neural Networks, Figure 4.11 looks at the energy consumption for inference over a single MNIST digits. We consider two architectures: a neural network with a single hidden layer (Figure 4.11(a)) and another one with two hidden layers (Figure 4.11(b)), and we vary the number of hidden neurons. Figs. 4.11(a) and Figs. 4.11(b) plot on the $x$ axis the estimated energy consumption of a Binarized Neural Networks using our archi-

tecture based on the flow presented in the Methods section. It also plots the energy required for the arithmetic operations (sum and product) of a eight bit fixed point regular neural network, neglecting the overhead that is considered for the Binarized Neural Network. For both types of networks, the *y* axis shows the resulting accuracy on the MNIST task. We see that at equivalent precision, the Binarized Neural Network always consume less energy than the arithmetic operations of the real-valued one. It is remarkable that the energy benefit depends significantly on the targeted accuracy, and should therefore be investigated in a case by case basis. The highest energy benefits, a little less than a factor ten, are seen at lower targeted precision.

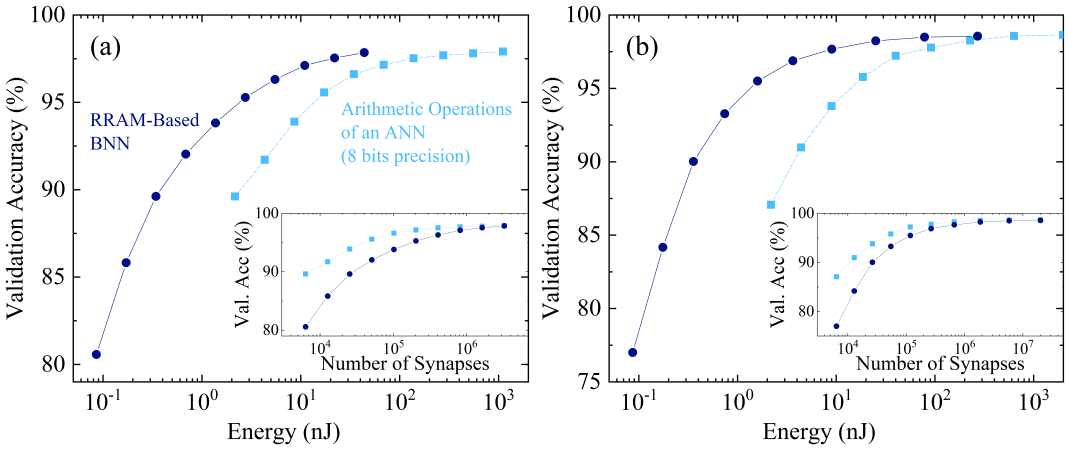



Figure 4.11: Dark blue circles: MNIST validation accuracy as a function of the inference energy of our Binarized Neural Network hardware design. Light blue square: same, as function of the energy used for arithmetic operation in a real valued neural networks employing eight bits fixed point arithmetic. The different points are obtained by varying the number of hidden neurons in (a) a one hidden layer neural network and (b) a two hidden layers neural network. Insets: number of synapses in each situations.

Binarized Neural Networks have other benefits with regards to real valued digital networks: if the weights are stored in RRAM, the programming energy is reduced due to the lower memory requirements of Binarized Neural Networks. The area of the overall circuit is also expected to be reduced due to the absence of multipliers, which are high area circuit.

### 4.5.3 Comparison with Analog Approaches

As mentioned in the introduction, a widely studied approach for implementing neural networks with RRAM is to rely on an analog electronics strategy, where Ohm's law is exploited for implementing multiplications, and Kirchoff's current law for implementing additions [124, 232, 233, 239, 240]. The digital approach presented in this Chapter cannot be straightforwardly compared to the analog approach: the detailed performance of the analog approach depends tremendously on its implementation details, device specifics and size of the neural networks.

Nevertheless, several points can be raised.

First, the programming of the devices is much simpler in our approach than in the analog one: one only needs to program a device and its complementary one in LRS and HRS, which can be achieved by two programming pulses. It is not necessary to verify the programming operation, as the neural network has inherent bit error tolerance. Programming RRAM for analog operation is a more challenging task, and usually requires a sequence of multiple pulses [232], which leads to higher programming energy and device aging.

For the neural network operation, the analog approach and ours function differently. Our approach reads synaptic values using the sense amplifier, which is a highly energy efficient and fast circuit that can operate at hundreds of picoseconds in advanced CMOS nodes [256]. This sense amplifier inherently produces the multiplication operation, and then the addition needs to be performed using low bit-width digital integer addition circuit. The ensemble of a read operation and the corresponding addition typically consumes fourteen femtojoules in our estimates in advanced node. In the analog approach, the read operation is performed by applying a voltage pulse, and inherently produces the multiplication though Ohm's law, but also the addition though Kirchoff law. This approach is attractive, but in the other hand requires the use of CMOS analog overhead circuitry such as operational amplifier, which can bring high energy and area overhead. Which approach is the most energy efficient between ours and the analog one will probably depend tremendously on memory size, application and targeted accuracy.

Another advantage of the digital approach is that it is much simpler to design, test and verify, as it relies on all standard VLSI design tools. On the other hand, an advantage of the analog approach is that it may, for small memory size, function without access transistors, resulting in higher memory densities [232].

### 4.5.4 Impact in Terms of Programming Energy and Device Aging

A last comment is that the bit error tolerance of binarized neural networks can have considerable benefits at the system level. Table 4.1 summarizes the measured properties of RRAM cells in different programming conditions, chosen out of the ones presented in Figure 4.5. We consider only the conditions with bit error rates below $10^{-3}$ (i.e. corresponding to a "$< 1$" data point in Figure 4.5), as this constrains makes them appropriate for use for all tasks considered in section 4.4. The "strong" programming conditions are the ones presented in Figure 4.6. They feature low bit error rate before aging, but high programming energy. The other two columns correspond to two optimized choices. The conditions optimized for programming energy are the conditions of Figure 4.5 with bit error rates below $10^{-3}$ and the lowest programming energy. They use a lower RESET voltage ($2.0V$) than the strong conditions, and shorter programming time ($100ns$). The cyclability of the device – defined as the number of cycles a cell can be programmed while retaining a bit error rate below $10^{-3}$ – remains comparable to the strong programming conditions. The conditions optimized for endurance are by contrast the conditions of Figure 4.5 with bit error rate below $10^{-3}$ and the highest cyclability: more than $10^{10}$ cycles,

as already evidenced in Figure 4.7. These conditions use a low RESET voltage $1.5V$ but require a programming time of $1\mu s$.

| **Programming condition** | **Strong (Figure 6)** | **Optimized endurance (Figure 7)** | **Optimized programming energy** |
|---|---|---|---|
| SET compliance current | $200\mu A$ | $200\mu A$ | $200\mu A$ |
| RESET voltage | $2.5V$ | $1.5V$ | $2V$ |
| Programming time | $1\mu s$ | $1\mu s$ | $100\,ns$ |
| 2T2R Bit error rate (before aging) | $< 10^{-7}$ | $< 10^{-4}$ | $< 10^{-5}$ |
| Programming energy (SET/RESET) | $200 \sim 400\,pJ$ | $150 \sim 400\,pJ$ | $20 \sim 30\,pJ$ |
| Cyclability (number of cycles at BER $< 10^{-3}$) | $> 10^8$ | $> 10^{10}$ | $> 10^8$ |

Table 4.1: RRAM Properties with Different Programming Conditions

## 4.6 Stochastic Computing for Binarized Neural Networks

### 4.6.1 Background

Binarized Neural Networks are not entirely binarized: the first layer input is usually coded as a fixed point real number. This fact is not a significant issue for operating Binarized neural Networks on graphical processor units (GPUs) [241], as they feature extensive arithmetic units. Research aimed at implementing binarized neural network on Field Programmable Gate Arrays (FPGAs) [266] has also not specifically investigated the question of the non-binarized first layer: these works usually use the Digital Signal Processors (DSPs) of the FPGA to process the associated operations. However, in an application-specific integrated circuits (ASIC) implementation, the non-binarization of the first layer implies that this layer needs a specific design, which is more energy consuming and uses more area than the design used for the purely binary layers.

For this reason, in this section, we introduce a stochastic computing implementation of Binarized Neural Networks, which allows implementing them in an entirely binarized fashion. The network functions by presenting several stochastically binarized versions of the images to the Binarized neural Network, in a way reminiscent to the historic concept of stochastic computing [267]. Our work focuses on the inference part of Binarized Neural Networks.

To evaluate the stochastic computing approach, we use the Fashion-MNIST dataset, which has the same format as MNIST, but presents grayscale images of fashion items [268], and constitutes a harder task. The canonical MNIST dataset would not be appropriate for this study, as it consists in images that are mostly black and white. As in the MNIST dataset, each image

in Fashion-MNIST has 28x28 pixels, and can be classified within ten classes. The dataset contains 60,000 training examples, 10,000 test examples. Conventional Binarized neural Networks (non-binarized first layer and no use of stochastic computing), perform very well on this task. With a fully connected Binarized neural Network with first layer coded with eight bit fixed point real numbers, with two hidden layers of 1024 neurons each and dropout, a classification accuracy of 90% can be obtained after 300 epochs. This result is comparable with the test accuracy of 91% obtained by a conventional real-valued neural network with the same architecture.

### 4.6.2 Stochastic Computing with Regular Training Procedure

A first approach to design a stochastic computing Binarized neural Network is to reuse the synaptic weights of a conventional Binarized neural Network, trained with grayscale picture. However, in the inference phase, we approximate the computation of the first layer by using stochastic images presentation instead of grayscale images.

The quality of the results depends on the number of image presentation $T$. In Figure 4.12, the navy blue curve shows the network test error as a function of $T$. We can see that after 100 stochastic image presentation, the accuracy is nearly equivalent to the use of grayscale images. With eight image presentation, the test accuracy is reduced to 88% instead of 90.1%. With a single presentation, the test accuracy is only 76%.

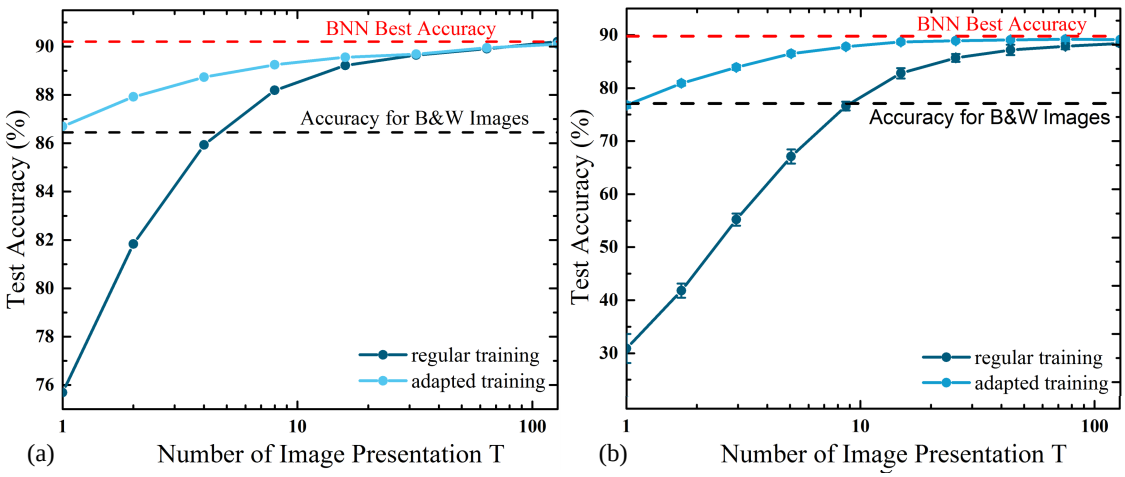



Figure 4.12: Accuracy on the (a) Fashion MINIST and (b) CIFAR-10 classification task as function of the number of stochastic image presented for the two training methods. Navy blue curve: training of the neural network with grayscale images. Light blue curve: training with presentation of stochastic binarized images. Dashed black line: accuracy when training with a black and white image (i.e. pixels with a value greater than 0.5 are white and pixels that are smaller are black). Dashed red line: best accuracy when the binarized neural network is trained with grayscale images.

### 4.6.3 Adapted Training Procedure

We now try a second strategy, where we train the neural network with binarized stochastic image presentation instead of grayscale images. To do this, during training, we use the conventional Binarized neural Network training technique, but instead of using the normal grayscale Fashion-MNIST images, we use stochastic binarized ones, with the same number of presentation $T$ as will be used during inference. The inference technique then remains identical to the the regular one. In Figure 4.12, in cyan color, we plotted the test error rate as a function of the number of presentation of the same image with this scheme. We see that the test accuracy is equivalent to the one obtained with grayscale images for high numbers of image presentation. On the other hand, with few stochastic presentation (one to five), the adapted input training technique allows reaching a quite high accuracy. If a single presentation is used at inference time, the network test accuracy is 86%. This test accuracy is equivalent to the one obtained when training a Binarized neural Network with non-stochastic black and white versions of the Fashion-MNIST dataset (dashed black line in Figure 4.12) (a). If three image presentation are used, the network test accuracy increases to 88.7%.

We also apply this strategy to the more advanced CIFAR-10 dataset, presented in Figure 4.12) (b). Each RGB channel pixel presents a value of zero or one. This value is chosen randomly with a probability equal to the RGB value of the corresponding pixel of the image.

These results show that when using the stochastic computing version of Binarized neural Network, the adapted training procedure should be used.

### 4.6.4 Energy and Area, comparison using stochastic computing

To study the hardware implementation of Binarized Neural Networks, we projected the implementation with modern technology available today. Therefore, we designed a system using the design kit of a commercial 28 nanometer technology and MRAM technology [45] [46] [55] instead of RRAM[1]. MRAM are mature technology and already available for commercial production whereas RRAM are still in the research phase. Digital circuits were described in synthesizable SystemVerilog description. MRAM memory arrays are modeled in a behavioral fashion, and their characteristics (area, energy consumption) are inspired by [269]. The system was synthesized to estimate its area and energy consumption. For energy consumption, we employed Value Change Dumps extracted from a Fashion-MNIST inference task, and estimated it using the Cadence Encounter tool.

Figure 4.13(a) shows the area of a basic cell of our architecture already presented in Figure 4.3(b-c), but now using MRAM instead of RRAM.

In the case of binary input (one operating bit), and in situations where the input is coded in Fixed Point representation (two, four and eight operating bit), as is required in the first layer of a conventional Binarized Neural Network.

[1] This study was performed before having access to the RRAM technology, it is also a reason explaining this choice.

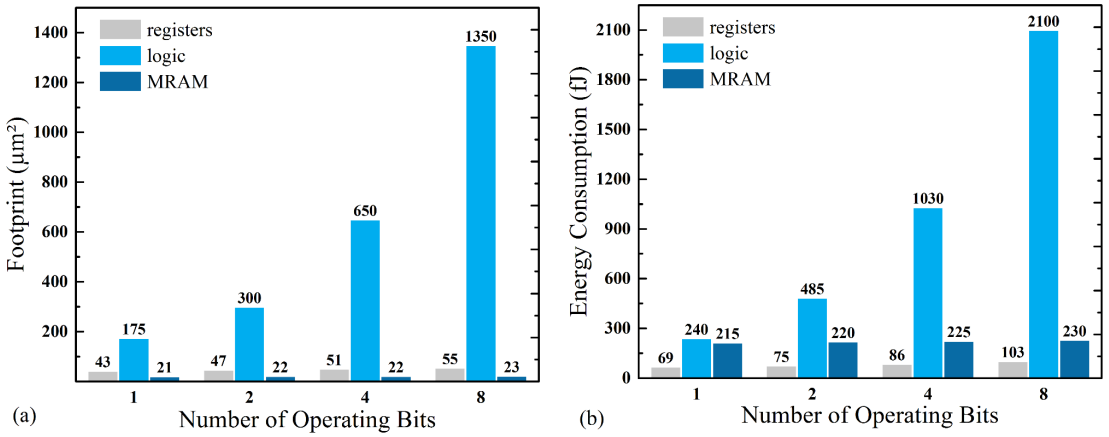



Figure 4.13: (a) Area of the basic cell (Figure 4.3 (b-c)) of our ASIC architecture, implemented in a 28 nm CMOS technology, as function of the number of operating bit for a fixed point binary architecture. One-bit corresponds to our stochastic fully binarized architecture. (b) Corresponding energy consumption, per clock cycle.

This Figure separates the area used by registers, logic and MRAM. A cell with binary input uses six times less area than a cell designed for eight bit input. Interestingly, the difference is mostly due to the popcount circuits, which need more depth when the input is non-binary. Similarly, as seen in Figure 4.13(b), a cell with binary input uses 4.5 times less energy per cycle than the corresponding one with eight bits input. Again, the difference is mostly due to the popcount circuits.

Using our architecture, a full Binarized with eight bit first layer occupies 1.95 $mm^2$, while the Binarized with stochastic binarized first layer occupies 0.73 $mm^2$, a 62% saving in area. These area values were extracted from a system designed for a $T$ value of eight.

Figure 4.14 plots the energy consumption for recognizing an image with our ASIC architecture, as a function of the number of presented stochastic images. This is compared with the energy cost of the same architecture, but using a non stochastic first layer, with eight bit input. We see that the system with stochastic first layer is more energy efficient than the system with non-binary first layer if less than eight presentation are used.

The previous curves do not include the cost of random bit generation. If we use a simple eight-bit Linear Feedback Shift Register (LFSR) pseudo random number generator, the added energy is $0.52nJ/cycle$, and the added area is $48,000\ \mu m^2$. Both are therefore negligible. It has also been shown that Spin Torque MRAM technology can be adapted to provide very low energy true random numbers [213]. If such a technology was used, based on the numbers of [213], the energy cost of random bit generation would be $0.125nJ/cycle$, and the area much smaller than LFSR. The energy cost of random number generation is therefore negligible with regards to the consumption of the system seen in Figure 4.14.

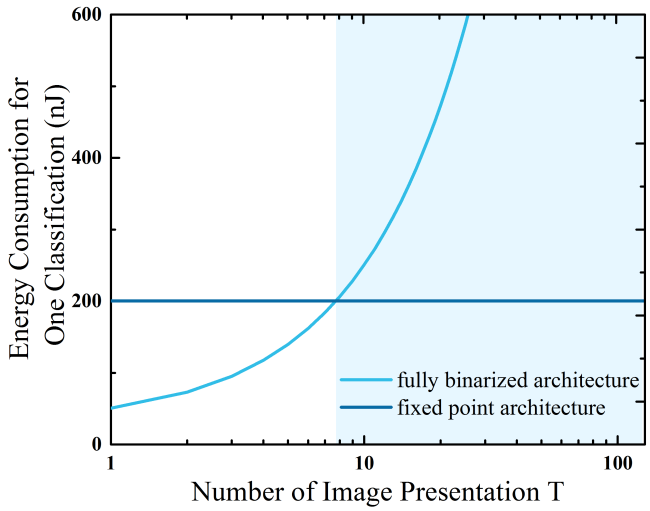



Figure 4.14: Energy consumption of the full Fashion-MNIST classifier systems, for the classification of one image. Light blue: stochastic fully binarized binary architecture. Navy blue: Conventional BNN with non binary (8 bit fixed point) first layers. The neural networks have two layers with 1024 neurons each. The light blue area indicate the regime where the non-binary first layer is more energy efficient thant the fully binarized system.

These energy numbers are very attractive with regards to non binarized implementations at equivalent recognition rate. Non binarized neural networks require less neurons and synapses than BNNs to achieve equivalent recognition rate. However, in an ASIC, the non binarized neural network requires energy-hungry 8-bits multiplications and addition (0.3 $pJ$ and 0.04 $pJ$ per operation in our 28 $nm$ technology). Taking into account only these arithmetic operations, the energy consumption is 220 $nJ$ for recognizing a Fashion-MNIST image with the same accuracy as the stochastic BNN with three image presentations. This stochastic BNN requires only 90 $nJ$ (Fig. 4.14), taking into account the whole system.

### 4.6.5 Partially Binarized Convolutional Neural Network (only Binarized Classifier)

In a study performed with Bogdan Penkovsky, we consider a variation of this scheme, a partially binarized convolutional neural network presented in [270]. Fully connected layers of neural networks are particularly adapted for in-memory Binarized implementation [228, 271], as these layers involve large quantities of memories. Convolutional layers are less memory intensive, and thus benefit less from binarization, while requiring increasing the number of channels when binarized [241]. In a hardware implementation, it can therefore be attractive to binarize only the classifier (fully connected) layers. In that case, the input of the classifier is real, and is processed with the stochastic BNN approach. This is also of special interest as the first fully connected layer in a convolutional neural network is usually the layer that involves the highest

number of additions, and can therefore benefit significantly in a hardware to be implemented with the stochastic approach.

We consider a neural network with the same architecture as the fully binarized one. Without the stochastic approach, this neural network has the same CIFAR-10 recognition rate than the fully binarized one (90%). If the classifier weights are retrained with the stochastic binarized inputs to the classifier, the stochastic results are very impressive. Even with a single image presentation $T$, the network approaches the performance of the non stochastic network. The stochastic BNN approach therefore appears especially effective in this situation.

| Task | Real-weight NN | BNN | Bin. Classifier |
|---|---|---|---|
| EEG | 88% [272] | 84.6% (1×) \| 86% (11×) | 87% (1×) |
| ECG | 96.3% | 92.1% (1×) \| 94.9% (7×) | 95.9% (1×) |
| ImageNet Top-1 | 70.6% [273] | 54.4% (4×) [274] | 70% (1×) |
| ImageNet Top-5 | 89.5% [273] | 77.5% (4×) [274] | 89.1% (1×) |

Table 4.2: Accuracy comparison of CNN with real weights, binarized CNN (BNN), and CNN where only the fully-connected part was binarized. In parentheses, number of filter augmentations.

To extend this study we were able to study other neural network architectures on other datasets and verify these performances, on electrocardiography (ECG) and Electroencephalography (EEG) and ImageNet data-set using MobileNet architecture [274]. In the work published in [270], we investigate the memory-accuracy trade-off when binarizing whole network and binarizing solely the classifier part, as well as the impact in overhead memory due to the increase in size of binarized neural networks compared to traditional neural networks. For both ECG and EEG models, most of the weights reside in fully-connected classifier layers, and the strategy of binarizing only the classifier has therefore high memory benefits.

## 4.7 Conclusion

This work proposes an architecture for implementing binarized neural networks with RRAMs, and incorporates several biological-plausible ideas:

- Fully co-locating logic memory,
- Relying only on low precision computation (through the Binarized Neural Network concept),
- Avoiding multiplication all-together,
- The acceptance of some errors without formal error correction.
- The stochastic computing approach is attractive in terms of area occupancy and energy efficiency, if a relatively small number of presentation is used ($T < 8$).

At the same time, our approach relies on conventional microelectronics ideas that are non-biological in nature:

- Relying on fixed point arithmetic to compute sums, whereas brains use analog computation,
- The use of sense amplifiers circuits, which are not brain-inspired ,
- And the use of a differential structure to reduce errors, a traditional electrical engineering strategy.

Based on these ideas, we designed, fabricated and tested extensively a memory structure with its peripheral circuitry, and designed and simulated a full digital system based on this memory structure. Our results show that this structure allows implementing neural networks without the use of Error Correcting Codes, which are usually used with emerging memories. Our approach also features very attractive properties in terms of energy consumption, and can allow using RRAM devices in "weak" programming regime where they have low programming energy and outstanding endurance. These results highlight that although in-memory computing cannot efficiently rely on Error Correcting Codes, it can still function without stringent requirements on device variability if a differential memory architecture is chosen.

When working on bioinspiration, drawing the line between bio-plausibility and embracing the differences between the nanodevices of the brain and electronic devices is always a challenging question. In this project, we highlight that digital electronics can be enriched by biologically-plausible ideas. When working with nanodevices, it can be beneficial to incorporate device physics questions into the design, and not to necessarily target the level of determinism that we have been accustomed to by CMOS.

This works opens multiple prospects. On the device front, it could be possible to develop more integrated 2T2R structures to increase the density of the memories. The concept of this work could also be adapted to other kind of emerging memories, such as phase change memories and spin torque magnetoresistive memories. At the system level, we are now in a position to fabricate larger systems, and to investigate the extension of our concept to more varied forms of neural network architecture such as convolutional and recurrent ones. In the case of convolutional networks, a dilemma appears between carrying the in-memory computing approach to its fullest, by replicating physically convolutional kernels, or implementing some sequential computation to minimize resources, as works have started to evaluate already. These considerations open the way for truly low energy artificial intelligence for both servers and embedded systems.

# Chapter 5

# Exploiting Stochastic Devices Behaviour for Brain Inspired Computing

We need to group the neurons [...] that do a lot of internal computation and then output a compact result.

Geoffrey HINTON

"FABRICATING *efficient hardware computing devices leveraging stochasticity could be an alternative use for emerging nanodevices. In neuroscience, population coding theory demonstrates that neural assemblies can achieve fault-tolerant information processing. Mapped to nanoelectronics, this strategy could allow for reliable computing with scaled-down, noisy, imperfect devices. Doing so requires that the population components form a set of basis functions in terms of their response functions to inputs, offering a physical substrate for computing. Such a population can be implemented with CMOS technology, but the corresponding circuits have high area or energy requirements. Here, we show that nanoscale magnetic tunnel junctions can instead be assembled to meet these requirements. We demonstrate experimentally that a population of nine junctions can implement a basis set of functions, providing the data to achieve, for example, the generation of cursive letters. We design hybrid magnetic-CMOS systems based on interlinked populations of junctions and show that they can learn to realize non-linear variability-resilient transformations with a low imprint area and low power.* "

THE CHALLENGES to reduce the area and increase the energy efficiency of microelectronic circuits are increasing dramatically. The size of transistors is reaching the nanoscale, and decreasing their dimensions further, or using emerging nanometer-scale devices, leads to stochastic behaviors, large device-to-device variability, and failures [230, 275].

Our current computing schemes are not able to deal well with noisy, variable, and faulty components. Entire processor chips are rejected based on single component failure. However, we have seen in the previous chapter that Binarized Neural Networks can be extremely resilient to errors. But we simply verified that these components had no impact on the performance of our neural networks. In this chapter, we are going further by directly exploiting their seemingly bad characteristics to make calculations.

To introduce this chapter, we will present a very innovative work from the literature that uses stochastic components as synapses to implement a learning algorithm particularly adapted to the intrinsic non-volatile nature of RRAM devices [276].

Then we will present a work done in collaboration with Alice Mizrahi at CNRS-Thales, which involves the use of the stochastic nature of nanodevices to implement the neurons of a neuroscience-inspired model. My main role in this collaboration was the design of the hardware implementation of the system as well as its energy evaluation.

## 5.1 Exploiting stochastic device behaviours for Markov Chain Monte Carlo Sampling

Operating at the thermal limit, our brain seems to have found an optimal tradeoff between low-energy consumption and computational reliability [237]. It carries out amazingly complex computations even though its components, neurons, are very noisy [277, 278].

In Chapter 3, the use of device stochastic behaviour has been shown to be very appealing for Bayesian computation as a random number generator. From a similar perspective, the use of stochastic devices can be exploited as synapse in a neural network. In our work presented in Chapter 4, the stochastic behaviour of synapses is mainly a concern for binarized neural networks. However, in the literature some neural networks are exploiting the stochastic behaviour of synapses: they are called Bayesian Neural Network [279].

A first interesting work exploiting the complex behaviour of components and more particularly MRAM to build a Bayesian network was proposed in [280]. The idea is to use magnetic tunnel junctions as a tunable random number generator to emulate a Bayesian network. One of the main advantages is that they use stochastic units called "p-bits" which can be interconnected and which are invertible. It is called invertible because in the "direct" mode, the input is fixed and the network gives the correct output response and in the "inverted" mode, the output is fixed and the network fluctuates according to all the input possibilities, consistent with the output [281]. The use of this type of approach seems to be an interesting idea to explore for

sampling in Bayesian Neural Networks. As a consequence, the use of innovative memory devices is a key element for in-memory computing, not only because it allows values to be stored in memory for energy-efficient calculation, but also because their behaviour can be exploited for calculation.

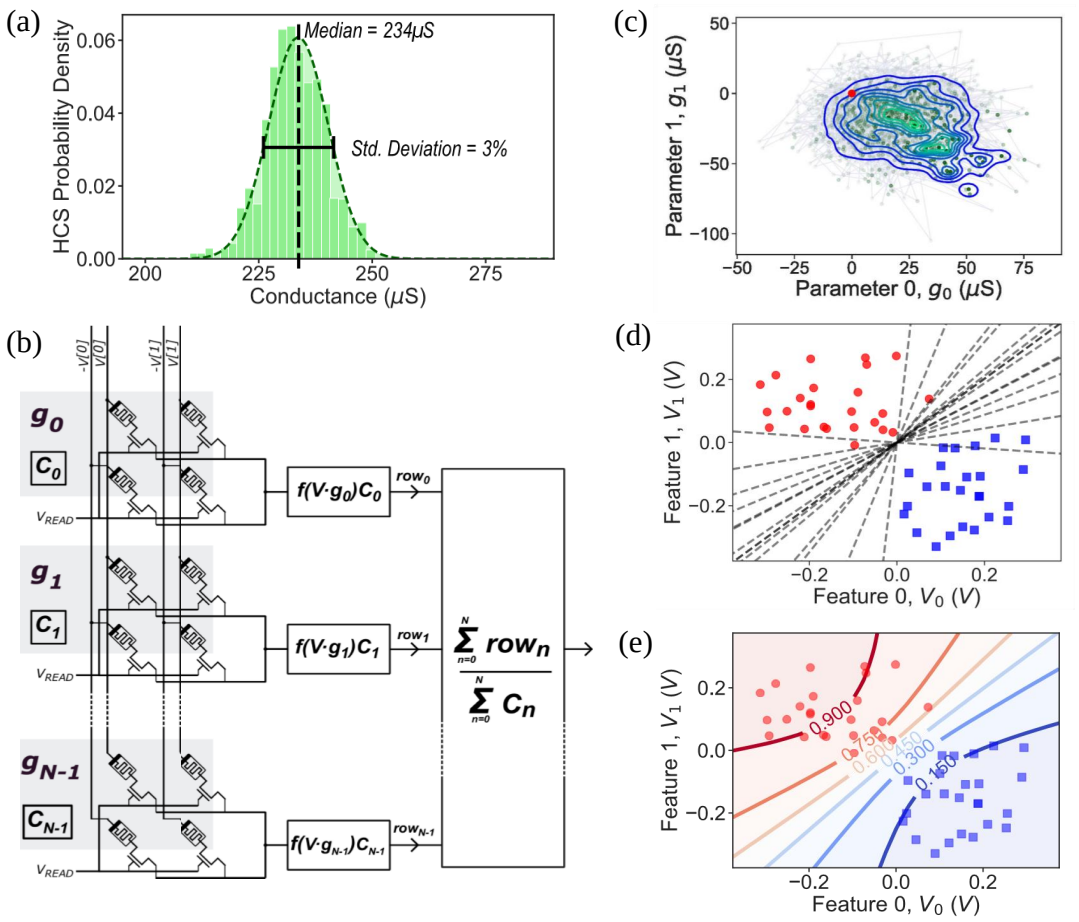



Figure 5.1: Using RRAM devices variability for Markov Chain Monte Carlo Sampling, taken from [276]. (a) Distribution of conductance for one specific device in the high conductance state when applying multiple SET programming pulses. (b) Memory array used to store the multiple parameters SET, each row correspond to one specific set of parameter, a counter is used to add a coefficient for the row that corresponds to the number of the rejection by the Metropolis-Hastings algorithm. (c) The posterior distribution for a multiple set of parameters, the initial parameter set is the redpoint, each accepted model corresponds to one green point and the corresponding level lines show the probability value. (d) Input vectors are presented as two categories, red point or blue point, each dashed black line corresponds to one set of parameters. (e) The probabilistic boundary of the posterior distribution, the value of the probability to belong to the red category is written on each line of the boundary.

One of the interesting features of new memory devices is that they can be integrated into the core of the CMOS and use little energy, but this almost forgets that the main feature of these

memory components is their non-volatility. How to exploit the non-volatility of a component and its intrinsic physical behaviour? Indeed, if we use its physical behaviour, most of the time the non-volatile characteristic is no longer of interest. In my opinion, one of the most interesting works exploiting both the physical behavior of memory devices and their non-volatility for implementing synapses is *"In-situ learning harnessing intrinsic resistive memory variability through Markov Chain Monte Carlo Sampling"* [276].

The idea is to use the stochastic behaviour of RRAM devices when applying SET pulses: when SET pulses are applied to the devices, for each device, a Gaussian conductance distribution with a well-defined mean $\mu$ and standard deviation $\sigma$ in conductance is obtained (see Figure 5.1 (a)). As in a more classical approach, the conductance of an RRAM pair is used to code a parameter of the model [123] and the Kirchhoff laws to perform the dot product between the input vector and the parameters vector. Here, the system is presented for a canonical problem with two parameters, but in the paper, the study is also done with the evaluations on a malignant breast tissue recognition task and a reinforcement learning task.

As its name suggests, the idea of the paper is to perform a sampling of the parameters by the Metropolis-Hastings Monte-Carlo method. To do it, a set of parameters must be selected by the algorithm and then saved to obtain the full posterior distribution. In the paper, the authors saved for the canonical example 2048 sets of parameters as a memory array of size 2048x2 (Figure 5.1 (b)).

The learning is done in the following way: initially, the first row is selected and a first random sampling of the parameters is performed, then the model is evaluated. Thereafter, the next row is selected and sampling is again performed until the Metropolis-Hastings algorithm accepts the set of parameters. The system then proceeds the next row, and the one after it, until it has gone through the whole matrix. At each row, a counter will determine the number of rejections of samples in order to assign a coefficient to each line parameters. The presence of this coefficient is due to the generation of the new parameter set that is not done directly from the previous parameter set. Once the training is finished, a whole set of parameters is used to evaluate the full posterior distribution of the model. Figure 5.1 (c) shows the posterior distribution after training of all parameter sets stored in the memory array. Figure 5.1 (d) shows the data used to evaluate the model, it is a classification task where we are trying to separate the blue dots from the red dots. Each black dotted line corresponds to a different set of parameters while Figure 5.1 (e) shows the probabilistic boundary of the posterior distribution.

Gathering a set of parameters to perform Bayesian inference with memory technologies is a very innovative idea that could also be used for Bayesian neural networks. The technology used here is an RRAM technology, but this approach could be valid for a whole range of technologies. Unfortunately, regarding learning, the Metropolis-Hastings MCMC algorithm does not seem to be effective for neural networks with a very large number of parameters [282].

Moreover, we have seen in the section 2.1 that when we perform training with gradient descent, during the second training stage – that is, once the neural network starts to forget the

useless features– the exploration of the parameters is almost done following random-walks, indeed the mean of the gradients is very small compared to the standard deviation. Thus, it is possible to exploit these random-walks by saving several sets of parameters of the same model that have changed during this last phase. Then operating on a combination of these different sets of parameters can increase performance during inference.

## 5.2 Neural-like computing with populations of superparamagnetic basis functions

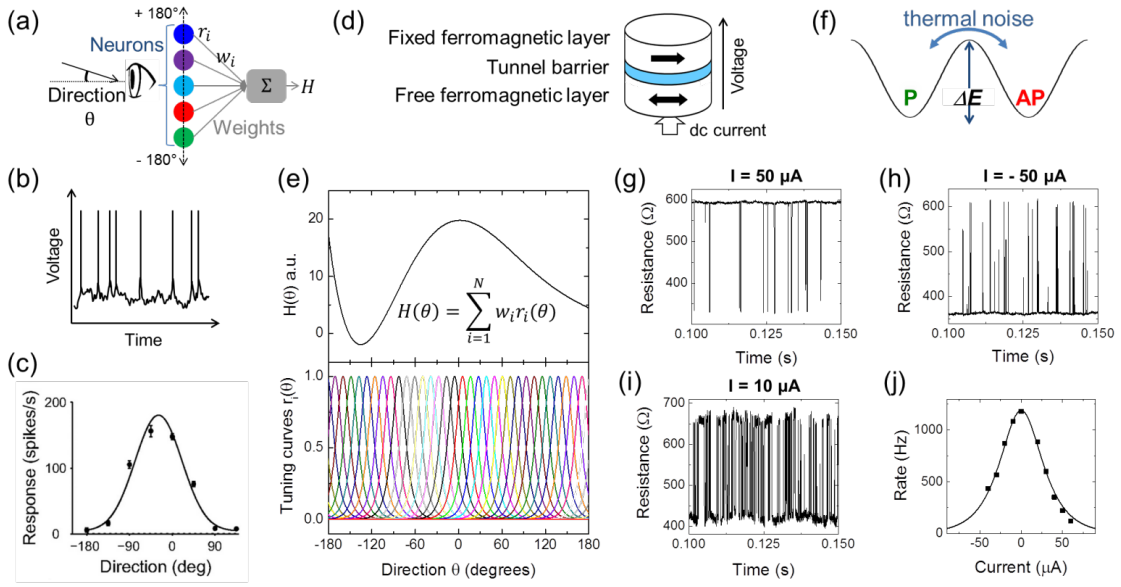



Figure 5.2: Neural and superparamagnetic tuning curves for population coding. (a) Schematic representing information reconstruction from a population of neurons. Each neuron (color dots) senses a specific range of stimuli (orientations). The represented function H is computed as a weighted sum of the rates of each neuron. (b) Sketch of a typical neuron firing pattern. The emitted voltage is plotted versus time. (c) Tuning curve of a neuron: spiking rate versus direction of the observed target, reproduced with data from [283]. Experiment (symbols) and Gaussian fit (solid lines) are shown. (d) Schematic of a superparamagnetic tunnel junction. (e) Polynomial function constructed from a weighted sum of the tuning curves of the population of neurons. (f) Energy landscape of the magnetic device. (g)–(i) Experimental measurements of the resistance versus time of a superparamagnetic tunnel junction for injected currents of $50\mu A$ (g), -$50\mu A$ (h), and $-10\mu A$ (i). (j) Rate of the superparamagnetic tunnel junction versus current. The experimental results (symbols) and analytical fit (solid line) are shown

The stochastic behavior of devices can not only be exploited for synapses, but also for the behavior of neurons. It is interesting to observe the important similarity between the behaviour

of a neuron in the brain and those of a specific type of nanodevices that we will study in this section. This work was published in [284] and is reproduced in the following sections of this chapter.

A key reason for the resilience of the brain seems to be redundancy. Measurements of neuronal activity in diverse parts of the brain such as the retina [285], the midbrain [286], the motor cortex [287] or the visual cortex [288] indicate that these parts encode and process information by populations of neurons rather than by single neurons. This principle of population coding and its benefits for the brain have been investigated in numerous theoretical works [89, 90]. In electronics, mimicking population coding has been proposed and shown to be effective in circuits using conventional transistors, but leads to circuits with high area costs due to the large size of the artificial neurons [289, 290]. It is therefore attractive to take inspiration from this strategy and compute with populations of low-area nanoscale electronic devices, even when they exhibit stochastic or variable behaviors. This approach has recently inspired pioneering studies of the dynamical response of ensembles of emerging nanodevices [291, 292]. However, showing that actual computations can be realized using the physics of population of nanodevices remains an open challenge.

Neuroscience studies indicate that, for this purpose, elementary devices mimicking neurons should have certain properties [89]. In particular, a neuron that is part of a population should possess a tuning curve: on average, it should spike more frequently for a narrow range of input values, to which it is tuned [293, 294]. Figure 5.2 (c) shows data from [283] corresponding to spike rate measurements of a single neuron in vivo. The corresponding tuning curve has a bell-shape dependence on the drift direction of the input visual stimulus. The measured neuron spikes more frequently when the drift direction is around -20°: it is in charge of representing the input over a narrow range of angles. In general, all neurons in a given population have similar tuning curves of rate versus amplitude. However, the tuning curves are shifted and distributed in order to cover the whole range of input amplitudes. The ensemble of tuning curves in the population then forms a basis set of functions (bottom panel of Fig. 5.2 (e)), similar to the sines and cosines of a Fourier expansion [89, 295].

In the present work, we show that a nanodevice –the superparamagnetic tunnel junction– naturally implements neurons for population coding, and that it can be exploited for designing systems that can compute and learn. The behavior of the nanodevice directly provides a tuning curve and resembles a spiking neuron. Without the use of explicit analog-to-digital converters it transforms an analog input into a naturally digital output that can then be processed by energy-efficient digital circuits, resulting in a low area and low energy system. The spiking nature of the neurons gives a stochastic character to the system, which appears a key element of its energy efficiency and a source of robustness.

After having studied and modeled the tuning curve provided by superparamagnetic tunnel junctions, we demonstrate experimentally that they can be assembled to implement a physical basis set of expansion functions and carry out computations. We simulate larger systems

composed of several populations of superparamagnetic junctions and show that they can be combined in order to learn complex non-linear transformations, and that the resulting systems are particularly resilient. We propose and evaluate an implementation associating the nanodevices with conventional CMOS (complementary metal oxide semiconductor) circuits, highlighting the low area and energy consumption potential of the approach.

## 5.3 Tuning curve of a superparamagnetic tunnel junction

Magnetic tunnel junctions, schematized in Fig. 5.2 (d), are devices composed of two ferromagnets: one with a fixed magnetization and the other with a free magnetization that can be either parallel (P) or antiparallel (AP) to the fixed magnet. Large junctions are stable and used today as non-volatile memory cells in spin-torque magneto-resistive random access memories (ST-MRAM) [296]. However, when the junctions are scaled down, the energy barrier confining the magnetization in the P or AP states (ΔE in Fig. 5.2(f)) is reduced. For very small lateral dimensions of the junctions (typically below a few tens of nanometers), thermal fluctuations can destabilize the magnetic configuration, generating sustained stochastic oscillations between the P and AP states [297–299] (Fig. 5.2 (f)). This phenomenon, called superparamagnetism, leads to telegraphic signals of the resistance as a function of time through magneto-resistive effects. These stochastic junctions have recently attracted interest for novel forms of computing [298, 300, 301]. Here, we experimentally study superparamagnetic junctions with a $Co_{27}Fe_{53}B_{20}$ magnetic switching layer of thickness $1.7nm$, and an area of $60x120nm^2$ Figure 5.2(g)-(i) shows experimental time traces of a superparamagnetic junction resistance as a function of time. The thermally induced random resistive switches follow a Poisson process [297, 299, 302]. This phenomenon presents similarities with the highly stochastic neural firing illustrated in Fig. 5.2(b), also often modeled as a Poisson random process [298, 299].

We propose to combine the thermally induced resistive switches arising in nanoscale magnetic tunnel junctions with spin-torque phenomena to emulate the tuning curves of stochastic spiking neurons. Indeed, when a direct current is applied across a superparamagnetic tunnel junction, the escape rates of the Poisson process are modified through spin-transfer torque (STT) [297, 303]. As observed in Fig. 5.2 (g), a positive current stabilizes the anti-parallel state while a negative current stabilizes the parallel state (Fig. 5.2 (h)), resulting in reduced switching rates in both cases compared to the case where I is close to zero (Fig. 5.2 (i)). As a consequence, the rate of the stochastic oscillator varies with the value of the applied dc current. From such measurements, we extracted the rate r of the junction at various current values. The resulting experimental rate versus current curve $r(I)$ is shown in Fig. 5.2 (j). With its bell-shape, it accurately mimics the neural tuning curve schematized in Fig. 5.2 (c). Spin transfer torque theory [299] allows deriving the analytical expression of the rate of a superparamagnetic tunnel junction as a function of current:

$$r(I) = \frac{r_0}{cosh(\frac{\Delta EI}{k_B T I_c})} \tag{5.1}$$

In Eq. 5.1, $k_B T$ is the thermal energy, $I$ the applied current, and $I_c$ the critical current of the junction. As shown by the solid line in Fig. 5.2 (j), this equation fits well the experimental result, with $\frac{\Delta E}{k_B T} \approx 13$, and a critical current $I_c$ of $300\mu A$. The natural rate $r_0 = \varphi_0 exp(-\frac{\Delta E}{k_B T})$ (with an attempt frequency $\varphi_0$ of $1GHz$) is the peak frequency at zero current, of the order of a few thousand Hertz in the case of the junction of Fig. 5.2 (j). Superparamagnetic tunnel junctions therefore have a well-defined tuning curve $r(I)$, which allows them to sense a narrow range of currents around zero current (here around $\pm 50\mu A$). The shape of the superparamagnetic tuning curve approximates a Gaussian function, which is favorable for population coding, as the ensemble of Gaussian functions with all possible peak positions forms a well-known basis set [89].

## 5.4 Population coding with superparamagnetic tunnel junctions

Following the basic principles of [89], for our approach, we need to produce a population of superparamagnetic tunnel junctions that can construct non-linear functions H of its inputs through a simple weighted sum of the nanodevice non-linear tuning curves $r_i$:

$$H(\theta) = \sum_{i=1}^{N} w_i r_i(\theta) \tag{5.2}$$

Non-linear transformations underlie a wide range of computations such as pattern recognition, decision making or motion generation [295, 304–308]. For example, navigating in a crowded room requires generating complex trajectories to avoid obstacles. The top panel of Fig. 5.2 (e) displays an instance of such a trajectory produced through Eq. 5.2 using the basis set formed by the tuning curves in the bottom panel. These outputs are generated by the ensemble of the neural responses. Therefore, having a full population rather than a single superparamagnetic tunnel junction allows for parallel processing of each neuron, as well as resilience to failure of the devices. In addition, the population outputs correspond to time averages of the stochastic neural firing patterns, which make them robust to noise. Good approximations of these output curves can be obtained quickly and at low energy by averaging the first few observed spikes, whereas more precision can be gained by increasing the measurement length.

To build a population, we need to tune each junction to different ranges of input currents. An elegant solution for this purpose is to leverage a spintronic effect called spin-orbit torques [309–311]. However, shifting the tuning curves can also be achieved by applying individual current biases $I_{bias}$ to each junction, so that the effective current $I_{eff}$ flowing in a junction is shifted compared to the common applied current $I_{app}$: $I_{eff} = I_{app} - I_{bias}$. This

method has been used in CMOS-only hardware implementations of population coding [289]. Figure 5.3(a) shows the normalized rates $r/r0$ of an experimental population of nine junctions obtained with this method (symbols) and the corresponding fits with Eq. 5.1 (solid lines). We have chosen the shifts so that the junctions in the population cooperate to sense a large range of currents between $-300$ and $+300\mu A$. As can be observed in Fig. 5.3(a), the junctions are not identical due to the polycrystalline nature of the free ferromagnetic layer. This variability affects both the critical current $I_c$ and the energy barrier $\Delta E$, resulting in the width variations of the tuning curves in Fig. 5.2(a), but also in the variation of natural rates that for this set of junctions span from a few Hertz to $70kHz$.

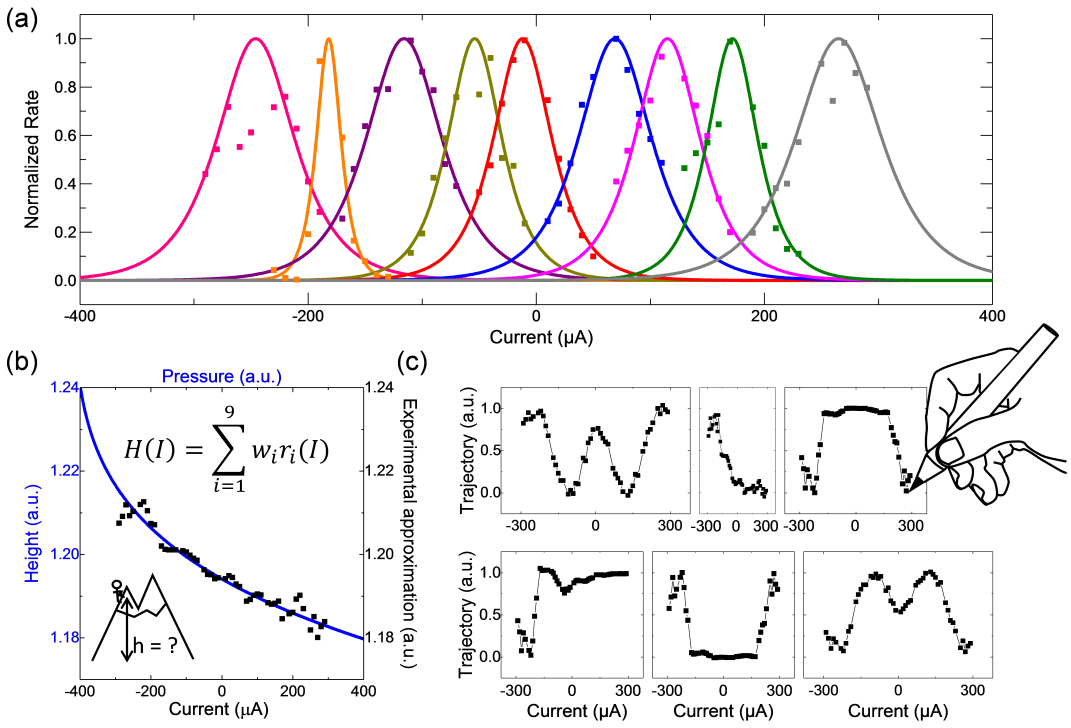



Figure 5.3: Representing non-linear functions with superparamagnetic tunnel junctions. (a) Rates versus current for nine superparamagnetic tunnel junctions with shifted tuning curves. Symbols correspond to experimental data while solid lines are analytical fits with Eq. 5.1. The switching rate of each junction is normalized by its natural rate $r_0$. (b) Example of the altimeter sensor. The solid blue line corresponds to the barometric formula, converting an air pressure measurement into the local height. The black symbols correspond to the experimental approximation of this function generated with Eq. 5.3, using the basis set data from (a) and performing the weighted sum with a computer. (c) Six examples of cursive letters (w, i, n, r, u, m) generated from the experimental junction tuning curves of (a) following the same procedure as in (b)

Despite this variability, the experimental basis set of nine superparamagnetic tuning curves

can be used to perform useful computations. We encode the input to process in the current applied to the junctions. We use the junctions measured output rates $r_i(I)$. Then this data is used to achieve the transformation to the output function $H$ by performing a weighted sum through:

$$H(I) = \sum_{i=1}^{9} w_i r_i(I) \tag{5.3}$$

where the optimal weights in Eq. 5.3 for the desired function H are obtained through matrix inversion on a computer.

Non-linear transformations of inputs as in Eq. 5.3 are essential in many applications. A first field of applications is sensors, which generally require converting a measured quantity into the sought-after information through a complex equation. For instance, a thermometer will convert the height of a column of liquid into a temperature. Similarly, an altimeter measures the local air pressure that is then converted into the corresponding height through the barometric equation shown in solid line in Fig. 5.3 (b). We have used our experimental basis set to implement this equation. As can be seen in Fig. 5.3 (b), the output reconstructed from the experimental data using Eq. 5.3 (symbols) reproduces the desired function. Another application making substantial use of non-linear transformations is motor control. Indeed, directing robotic arms, guiding vehicles or moving biological fingers require the generation of complex trajectories. For instance, we use here our superparamagnetic basis set to create handwriting. Figure 5.3 (c) shows that we can successfully output six letters, which means that our small experimental system of nine junctions could potentially guide a robot's arm to write. These results constitute the proof of concept of computing with electronic nanodevices through population coding.

## 5.5 A computing unit that can learn

We have seen that the benefit of representing a value, such as the current I, by a basis set population is that non-linear transformations on this value, $H(I)$, can be conducted by operating only linear operations. However, in order to realize multi-step computations, series of non-linear transformations are necessary. As a consequence, the result $H(I)$ of the first transformation should be represented by a basis set as well, implemented by an output population.

For this purpose, we can take inspiration from biology, where neurons in different populations are densely connected through synapses which control the strength of the connection. This configuration has indeed multiple advantages. In particular, the weight values can be learnt from example data, and the high degree of interconnection provides a high resilience to noise and variability in the synapses and neurons. In neuroscience models, the rates of an output population are linked to the input rates through linear weights $w_{ij}$ [90, 301].

$$r_j^{OUT} = \sum_{i=1}^{N} w_i r_i^{IN} \tag{5.4}$$

The encoded value $Y$ can then be determined by counting the switching rates of the output population: $Y$ is equal to the mean of the values of the stimulus to which the neurons are tuned, weighted by the spiking rates of the corresponding neurons [89, 305].

$$Y = \frac{\sum_{j=1}^{N} I_{j_{bias}} r_j^{OUT}}{\sum_{j=1}^{N} r_j^{OUT}} \tag{5.5}$$

The error of the system is then the distance between $H(I)$ and $Y$.

To evaluate this approach before designing the full system, we perform numerical simulations of transformation learning with two populations of superparamagnetic tunnel junctions. We choose parameters for the junctions that reflect the experimental values and variability of their energy barrier and their critical current.

We first focus on an example of a sensory-motor task (illustrated in Fig. 5.4(a)) to explain our system and demonstrate the transfer of information between two basis sets, implemented by two different populations. A robot observes an object with a visual sensor and attempts to grasp it with a gripper. The input population of junctions receives a current I encoding for the orientation of the object. The output population represents the orientation $Y$ of the gripper. We want to find the weights $w_{ij}$ allowing for the orientation $Y$ of the gripper to match the orientation of the object, and show how they can be learned. For this purpose, we follow an error and trial procedure, similar to the one described in [312]. Originally, the weights are random. At each trial, the object is presented at a different orientation and the weights are modified depending on the success of the grasping:

1. If the gripper succeeds—i.e., if its orientation is close enough to the orientation of the object to be in the catch zone—the weights are unchanged.

2. If the gripper strikes in the up zone, the synaptic weights connecting the sensor network to motor junctions which are tuned to orientations above (resp. below) of the gripper are decreased (resp. increased).

3. If the gripper strikes in the down zone, the opposite is implemented.

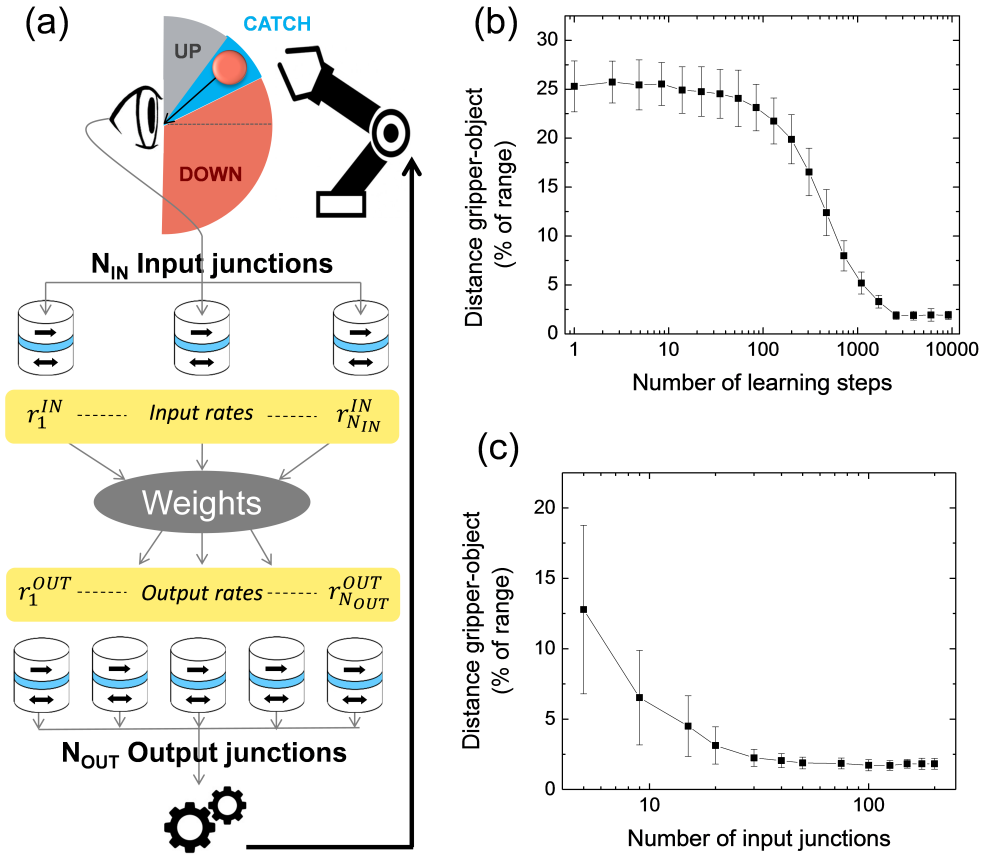



Figure 5.4: Learning to transfer information between two interconnected populations. (a) Schematic of the system and associated learning process. (b) Distance between the gripper and the object (i.e., grasping error) versus the number of learning steps (populations of 100 junctions). (c) Distance between the gripper and the object after 3000 learning steps as a function of the number of junctions in the input population. The output population has 100 junctions. For all figures, each data point corresponds to the average over 50 trials and the error bar to the associated standard deviation

The key advantage of this learning rule is its simplicity: there is no need to perform a precise measurement of the error (here distance between the gripper and the object) as required by most learning methods in the literature [313] [314]. Note that the proposed system is independent of this learning rule and that different algorithms could be used to perform more complex tasks. Figure 5.4 (b) shows that the distance between the object and the gripper is

progressively decreased through repeated learning steps. After 3000 learning steps, the mean error is below 2.5% of the range: learning is successful. As can be seen in Fig. 5.4 (c) the grasping error decreases as the number of junctions in the input population increases. The precision of the result indeed improves as the population grows, better approximating an ideal, infinite basis set. Figure 5.4 (c) also demonstrates that transfer of information between populations of different sizes can be achieved, allowing changes of basis if needed.

The example of the gripper in Fig. 5.4 shows how we can transfer information without degradation from one population to a different one performing a basis change. Now we show that our system and our simple learning procedure can also transform information during the transfer between populations, in other words, realize more complex functions than the identity of Fig. 5.4. In Fig. 5.5 (a), we illustrate increasingly more complicated transformations: linear but not identity (double), square, inverse, and sine of the stimulus. Each can be learned with excellent precision, similar to the identity.

Furthermore, by adding another matrix of synaptic weights and another population of junctions after the output of our system, we can realize transformations in series. An example of this is shown in Fig. 5.5 (a), as indicated by the label Series, where the square of the sine is performed.

The system can also be adapted for learning and performing tasks involving several inputs. A possible solution to process multiple inputs with a population is to combine them in a single input that can then be presented to the superparamagnetic tunnel junctions, consistently with the approach recently presented in [315]. Here we propose a different approach where each input is sent to a different input population, and the rates originating from these separate populations are combined into a single neural network. In this way, by using several populations as inputs and outputs, multi-input multi-output computations, and therefore transformations in several dimensions can be learned. In particular, we used this approach to learn the conversion of coordinates from polar to Cartesian system. The results corresponding to this task are labeled 2 inputs in Fig. 5.5 (a).

The excellent precision of these transformations, obtained with junction parameters and variability extracted from experiments, demonstrates the resilience of our system to variability. Additional simulations indicate that variability of the critical current barely affects the system. Figure 5.5 (b) shows the distance between the object and the gripper as a function of the variability on the energy barrier (and thus on the natural rate). The level of variability corresponding to experiments is indicated. We observe that even larger levels of variability can be tolerated by the system, which is promising for realizing population coding with ultra-small junctions despite lithographic defects.

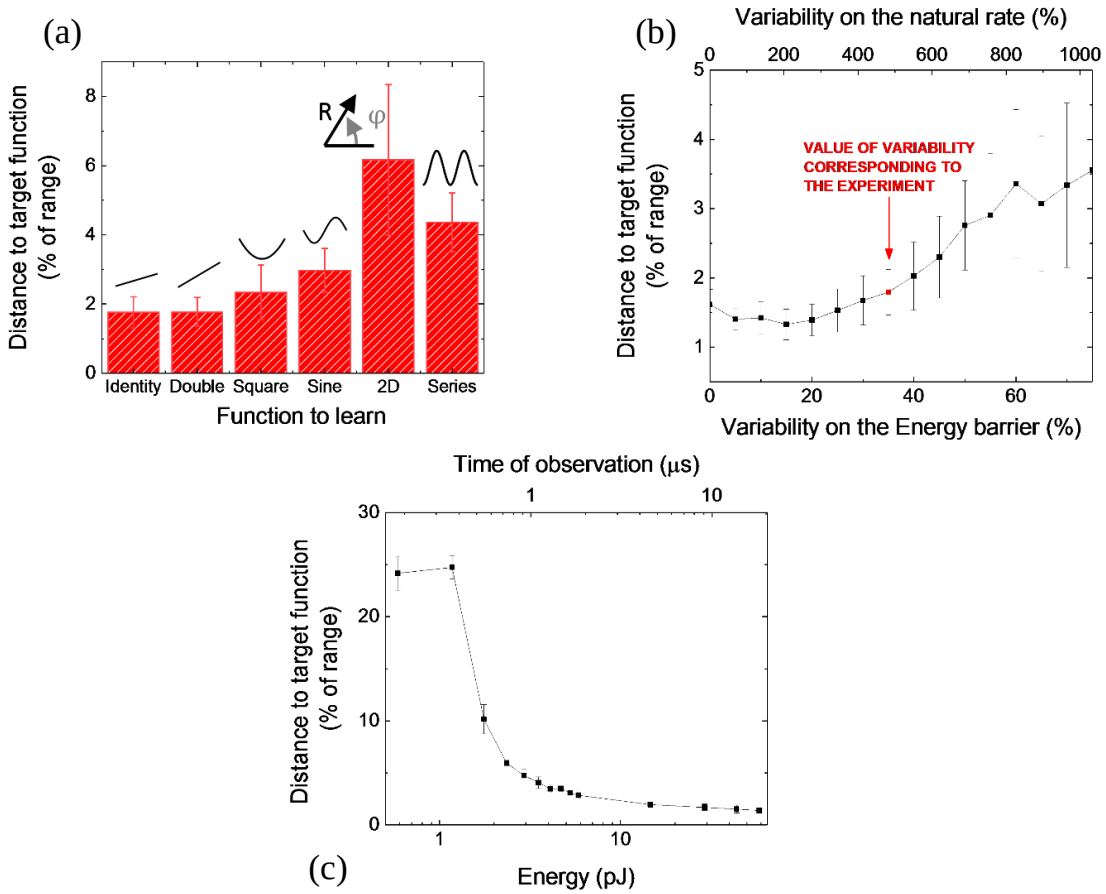



Figure 5.5: Evaluation of stochastic population coding with superparamagnetic tunnel junctions. (a) Performance of several transformations, including non-linear. The 2 inputs label corresponds to transformation from polar to Cartesian coordinates. The series label corresponds to two transformations in series implementing the function $sin^2(x)$. (b) Distance to target versus variability of the energy barrier (bottom axis) and variability of the natural frequency (top axis). The experimental variability is indicated in red. (c) Distance to target for different times of observation during which switching rates are recorded, leading to different energy dissipated by the junctions. Longer acquisition time allows better precision of the transformation, but leads to higher energy consumption. Each population is composed of 100 junctions and 3000 learning steps are used. Each data point corresponds to the average over 50 trials and the error bar to the associated standard deviation

Finally, it should be noted that scaling down the junctions allows decreasing the energy consumption of a population to tens of picoJoules, as show on Fig. 5.5 (c). Furthermore, as typical in stochastic computing systems [217], the precision of the system is directly dependent on the observation time and thus on the consumed energy, allowing choice in a precision-energy tradeoff.

## 5.6 Design of the full system

To evaluate the viability of the approach, we designed a full system associating superparamagnetic tunnel junctions as input neurons, CMOS circuits, and standard magnetic tunnel junction used as ST-MRAM to store the synaptic weights $w_{ij}$ . These stable junctions can be fabricated using the same magnetic stacks as the superparamagnetic junctions (but a different sizing). The CMOS parts of the circuit were designed using standard integrated circuit design tools and the design kit of a commercial 28nm CMOS technology. A representation of the system is shown in Fig. 5.6 (a).

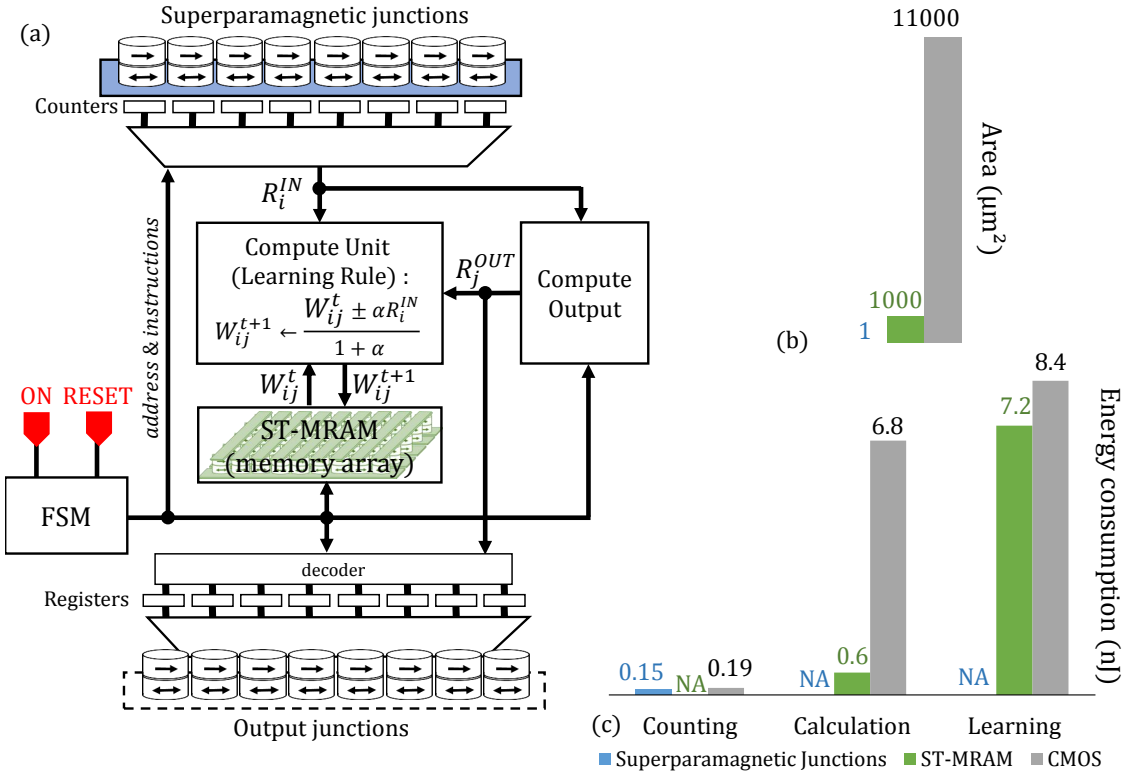



Figure 5.6: Design of the full system for the gripper task. (a) Schematic illustrating the data path of the designed system, associating CMOS circuits, superparamagnetic tunnel junctions, and stable magnetic junctions used as MRAM. (FSM finite state machine). (b) Circuit area occupied by the superparamagnetic tunnel junctions, CMOS, and MRAM. (c) Energy consumption of the superparamagnetic tunnel junctions, CMOS, and MRAM required to perform one system operation: running the junctions, computing the output, and adjusting the weight. The system has 128 inputs and 128 outputs

### 5.6.1 Methods

The CMOS digital parts of the system were designed with the SystemVerilog description language at the register transfer level, and synthesized to the standard cells provided with the design kit with Cadence RTL Compiler. Overall, the circuits were optimized for low-area and low-energy consumption, and not for high speed computation. Their area was estimated using the Cadence Encounter tool. For estimating their energy consumption, value change dumps files corresponding to the gripper task were generated using Cadence ncsim and the power consumption was estimated using Cadence Encounter.

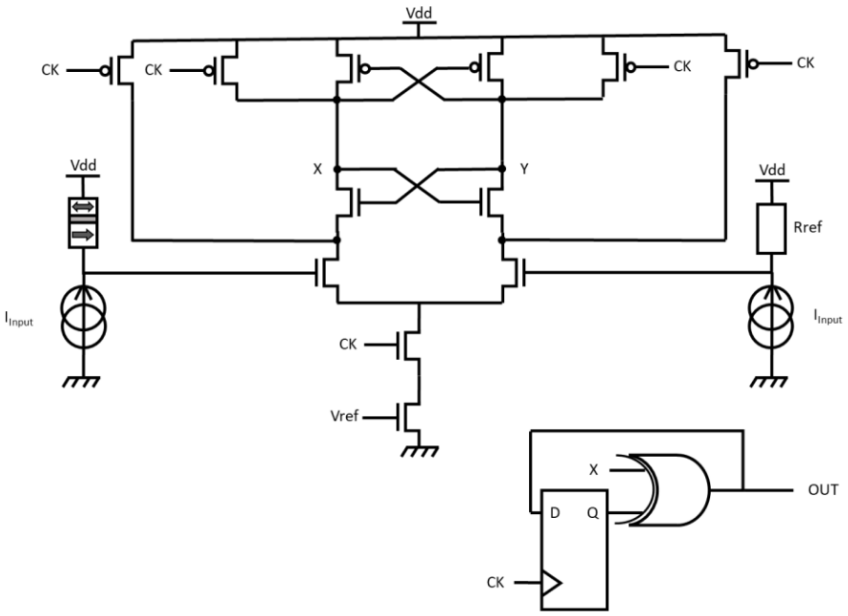



Figure 5.7: Circuit for converting the switching events of a superparamagnetic tunnel junction to a CMOS digital signal. In addition to the superparamagnetic tunnel junction, the stimulus current is applied to a reference resistor $R_{ref}$, whose resistance is intermediate between the parallel and anti-parallel state resistance of the superparamagnetic tunnel junctions, and at each clock cycle, the voltage at the junction and at the reference resistor is compared by a low power CMOS comparator. Simple logic comparing the result of the comparison to the same result at the previous clock cycle allows detecting the junction switching events, which are counted by an eight-bit digital counter.This design is not able to detect multiple switching occurring during a single clockcycle. We saw on system-level simulations that this particularity has no impact on the full application

The superparamagnetic junctions were modeled assuming $d = 11nm$ diameter, a size that has been demonstrated experimentally [316]. The energy consumption for the detection of the spikes was based on Cadence Spectre simulation of a simple circuit, presented in Figure 5.7, and based on the stimulus value corresponding to the highest energy consumption. The stimulus is applied to reference resistors whose resistance is intermediate between the par-

allel and anti-parallel state resistance of the superparamagnetic tunnel junctions, as well on the superparamagnetic tunnel junction. At each clock cycle, the voltage at the junction and at the reference resistor is compared by a low-power CMOS comparator (Figure 5.7). Simple logic comparing the result of the comparison to the same result at the previous clock cycle allows detecting the junction switching events, which are counted by an eight-bit digital counter. (Each junction is associated with one counter).

At the end of the counting phase, the system then computes Equation 5.4 in a sequential manner, controlled by a finite state machine (Figure 5.6 (a)). The synaptic weights are stored in eight-bit fixed point representation in an ST-MRAM array. Computation is realized in fixed point using integer addition and multiplication circuits. The ST-MRAM array was modeled using assumptions in terms of area and energy consumption as expected for a $28nm$ technology [317]. ST-MRAM read and write circuits are modeled in a behavioral fashion, using results of [256] for evaluating their area and energy consumption.

The learning circuit can be activated after the computation phase optionally. Based on the learning rule described above, computed in fixed point representation, the ST-MRAM array is reprogrammed. In order to save energy, ST-MRAM cells are read before programming, so that only bit that actually changed are reprogrammed (a standard technique for resistive memory [318])

### 5.6.2 Results

The system features an ensemble of superparamagnetic tunnel junctions, to which the stimulus is applied using the current shift method introduced earlier. But, it is also a possibility to design the system using a single superparamagnetic junction, and to implement the population response through time multiplexing. This approach would allow avoiding the effects of device variability. However, it would also increase conversion time by the number of input neurons, giving a very low bandwidth to the system. As the superparamagnetic junctions have low area and the system features a natural resilience to device variability, we propose to physically implement the population with an actual population of junctions.

Figure 5.6 (b) shows the circuit area occupied by superparamagnetic junctions, CMOS, and ST-MRAM on chip, for a system with 128 inputs and 128 outputs. The total area is very low ($12,000\mu m^2$) showing that the concept is adapted to be used in low-cost intelligent sensor applications. The area is dominated by the CMOS circuits, while the area occupied by the superparamagnetic junctions is negligible.

Figure 5.6 (c) shows the energy consumption to perform the gripper task, for one operation of the gripper, separating the three phases (observation of the stimulus, computation of Eq. 5.4, learning), and the three technologies present in the system. The total energy is very low: $23nJ$ during the learning phase, and $7.4nJ$ when learning is finished.

It is instructive to compare these results with solutions where neurons would have been implemented with purely CMOS circuits. A natural idea is to replace our junctions and their

read circuitry by low-power CMOS spiking neurons, such as those of [319], which provides features similar to our nanodevices (analog input and spiking digital output). This strategy works but has high area requirements (higher than $1mm^2$), and would consume more than $330nJ$ per operation. Alternative options rely on analog computation, for example exploiting neurons such as [290]. Such solutions require the use of an explicit analog to digital conversion (ADC), which actually becomes the dominant source of area and energy consumption. Even extremely energy efficient ADCs [320] require a total of $20nJ$/conversion and an area of $0.2mm^2$. Finally, a more conventional solution, using a generic processor and not an application-specific integrated circuit would have naturally used order-of-magnitudes more energy.

The low-energy consumption of our system arises from a combination of three major factors. The superparamagnetic junctions consume a negligible energy ($150pJ$), and allow avoiding the ADC bottleneck present in other approaches by implementing a form of stochastic ADC in a particularly efficient manner. The use of a stochastic approach and of integer arithmetic in the CMOS part of the circuit is particularly appealing in terms of energy consumption. Finally, associating both CMOS and spintronic technology on-chip limits data transfer-related energy consumption.

## 5.7 Discussion

In this work, we show that superparamagnetic tunnel junctions are promising nanodevices for computing in hardware through population coding. We experimentally demonstrate that these components intrinsically mimic the tuning curve of neurons through their non-linear frequency response to input currents. We realize a basis set of expansion functions in hardware from a small population of junctions, and show how they can encode information and compute by generating complex functions such as letters. Using a physical model of the superparamagnetic tuning curves, we demonstrate that combined populations of junctions can learn non-linear transformations with accuracy, even with substantial device-to-device variability. Our system acts as a stochastic computing unit that can be cascaded to perform complex tasks. The design of the full system associating the junctions with CMOS circuits and ST-MRAM shows the potential of the approach for extremely low-area and low-energy implementation.

Our work reproduces the essence of population coding in neuroscience, with some adaptations for implementation with nanoelectronics. In population coding theory, neuronal correlation [90, 321], the meaning of the time [90], as well as decoding techniques [321] are contentious topics. In our system, these aspects were guided by the properties of the nanodevices and by circuit design principles. The input neurons spike in an uncorrelated fashion, as their noise originates from basic physics. The time is divided into discrete phases, allowing the use of counters, and finite state machines in the system. The information is decoded by counting spikes using simple unsigned digital counters.

It is also important to note that in our system, the junctions act as a form of spiking neurons

that employ rate coding, similarly to several population coding theories [89, 90]. The spiking nature of the neurons offers considerable benefits to the full system: it naturally transforms an analog signal into easy-to-process digital signals. The stochastic nature of the neurons is one of the keys of the energy efficiency and of the robustness of the system. It also gives the possibility for the system to provide an approximate or precise answer depending on the time and energy budget, similarly to stochastic computing [217, 267]. The rest of the system is rate based, which allows learning tasks in a straightforward manner. Another possibility would have been to perform the entire operation in the spiking domain, as is common in the neuromorphic engineering community [129, 322, 323]. However, learning in the spiking regime remains a difficult problem today [130], and involves more advanced concepts and overheads [323]. Therefore, our system is designed to take benefits from both the spiking and the rate-coding approaches.

In summary, our system mixes biological and conventional electronics ideas to reach low-energy consumption in an approach that might presage the future of bioinspired systems. Our results therefore open the path to building low energy and robust brain-inspired processing hardware.

# Chapter 6

# Are Analog RRAMs required for Neuromorphic Computing?

There's no sense in being precise when you don't even know what you're talking about.

–Unsourced quote– John VON NEUMANN

"EXPLOITING *the analog properties of RRAM cells is a compelling approach for Neuromorphic Computing as it increases integration and calculation capabilities. Considering the performances of low precision neural network, using analog RRAM might seem unnecessary. Nevertheless, real weights are always required in the learning process of these low precision neural networks and are of particular interest to prevent catastrophic forgetting. Besides, learning raises important challenges in terms of CMOS overhead, the impact of device imperfections, and device endurance. For these reasons, in this chapter, we show that the analog behaviour of RRAM can be a key for both increasing neural network performances and implementing learning at the edge. We investigate a learning-capable architecture, based on the concept of Binarized Neural Networks, which addresses the three issues of on-chip learning with RRAM. It exploits the analog properties of the weak RESET in hafnium-oxide RRAM cells and requires no refresh process.* "

This last chapter presents the latest advances in our work on neural networks. The main idea is to look at how we can go further in the hardware implementation of binarized neural networks and more particularly how we can use the analog nature of RRAM devices and their analog nature to implement learning. In the preceding chapters, we have only worked on the inference.

A first step towards exploiting the analog nature of RRAM is to increase the capacity of binary weights for the inference using a third value. A very recent work on Ternarized neural network, mainly performed by Axel Laborieux and published in [184] opens the way to understanding what is necessary from a nanodevice perspective. The main idea is to use a configuration not used in our previous work, the HRS/HRS state, to encode this third weight value.

We also explored approaches using this analog nature much more deeply. First by suggesting the utility of the underlying analog value, and paving the way for metaplasticity, a memory variable, allowing the learning of several data sets at the same time. Then, we will see that this analog value can be used for learning and the various challenges of the implementation of the learning algorithm of binarized neural networks.

## 6.1 Ternarized Neural Network

The design of systems implementing low precision neural networks with emerging memories such as resistive random access memory (RRAM) is a significant lead for reducing the energy consumption of artificial intelligence, as already presented in chapter 4. To achieve maximum energy efficiency in such systems, logic and memory should be integrated as tightly as possible. The work presented here and published in [184] focuses on the case of ternary neural networks, where synaptic weights assume ternary values. We use the same architecture as in Chapter 4: a two-transistor/two-resistor memory architecture employing a precharge sense amplifier, where the weight value can be extracted in a single sense operation. Based on neural network simulation on the CIFAR-10 image recognition task, we show that the use of ternary neural networks significantly increases neural network performance, with regards to binarized ones.

Ternary neural networks (TNNs) are an extension to Binarized Neural Network where neuronal activations and synaptic weights $A_j$, $X_i$, and $W_{ji}$ can now assume three values: +1, −1, and 0 instead of binarized one. Neuronal activation then becomes:

$$A_j = \phi\left(\sum_i GXNOR\left(W_{ji}, X_i\right) - \mu_j\right). \tag{6.1}$$

$GXNOR$ is the "gated" XNOR operation that realizes the product between numbers with values +1, −1 and 0 (Table 6.1). $\phi$ is an activation function that outputs +1 if its input is greater than a threshold $\Delta$, −1 if the input is lesser than $-\Delta$ and 0 otherwise.

| $W_{ji}$ | $X_i$ | $XNOR$ |
|---|---|---|
| $-1$ | $-1$ | 1 |
| $-1$ | 1 | $-1$ |
| 1 | $-1$ | $-1$ |
| 1 | 1 | 1 |

| $W_{ji}$ | $X_i$ | $GXNOR$ |
|---|---|---|
| $-1$ | $-1$ | 1 |
| $-1$ | 1 | $-1$ |
| 1 | $-1$ | $-1$ |
| 1 | 1 | 1 |
| 0 | $X$ | 0 |
| $X$ | 0 | 0 |

Table 6.1: Truth Tables of the XNOR and GXNOR Gates

### 6.1.1 Using the same amount of device to implement ternary synapse without memory overhead

The architecture, where synaptic weights are stored in a differential fashion, and the used RRAM technology, are the same as presented in Chapter 4 and [324]. Each bit is implemented using two devices programmed either as low resistance state (LRS) / high resistance state (HRS) to mean weight +1 or HRS/LRS to mean weight −1. Additionally, we use the HRS/HRS configuration to mean synaptic weight 0, while the LRS/LRS configuration is avoided. The ternary synaptic weights are read using on-chip precharge sense amplifiers (PCSA) with the addition of a XOR logical gate, presented in Figure 3.16.

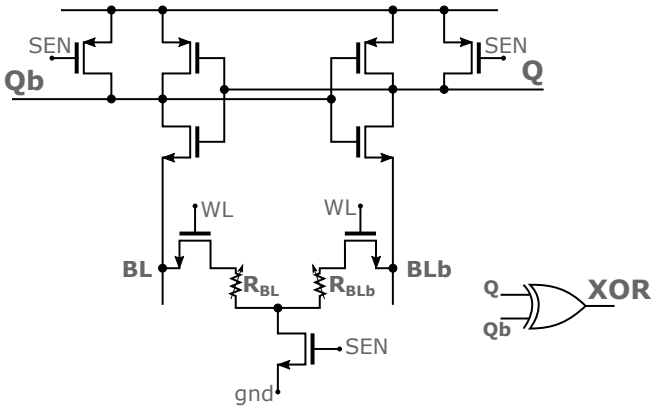



Figure 6.1: Schematic of the precharge sense amplifier with an additionnal XOR gate between output Q and Qb.

Figure 6.2 (a) shows an electrical simulation of this circuit to explained its working principle, using the Mentor Graphics Eldo simulator. As explain in the chapter 3, in the first phase (SEN=0), the outputs Q and Qb are precharged to the supply voltage $V_{DD}$. In the second phase (SEN=$V_{DD}$), each branch starts to discharge to the ground. The branch that has the resistive

memory (BL or BLb) with the lowest electrical resistance discharges faster and causes its associated inverter to drive the output of the other inverter to the supply voltage. At the end of the process, the two outputs are therefore complementary and can be used to tell which resistive memory has the highest resistance and therefore the synaptic weight.

We observed that the convergence speed of a PCSA depends heavily on the resistance state of the two resistive memories. Figure 6.2(b) shows a simulation where the two devices, BL and BLb, were programmed in the HRS. The two branches (BL or BLb) discharges slowly to the ground and causes to drive the inverters' output slowly.

We see that the two outputs converge to complementary values in more than $200ns$, whereas less than $50ns$ were necessary in Figure 6.2(a), where the devices are programmed in complementary LRS/HRS states.

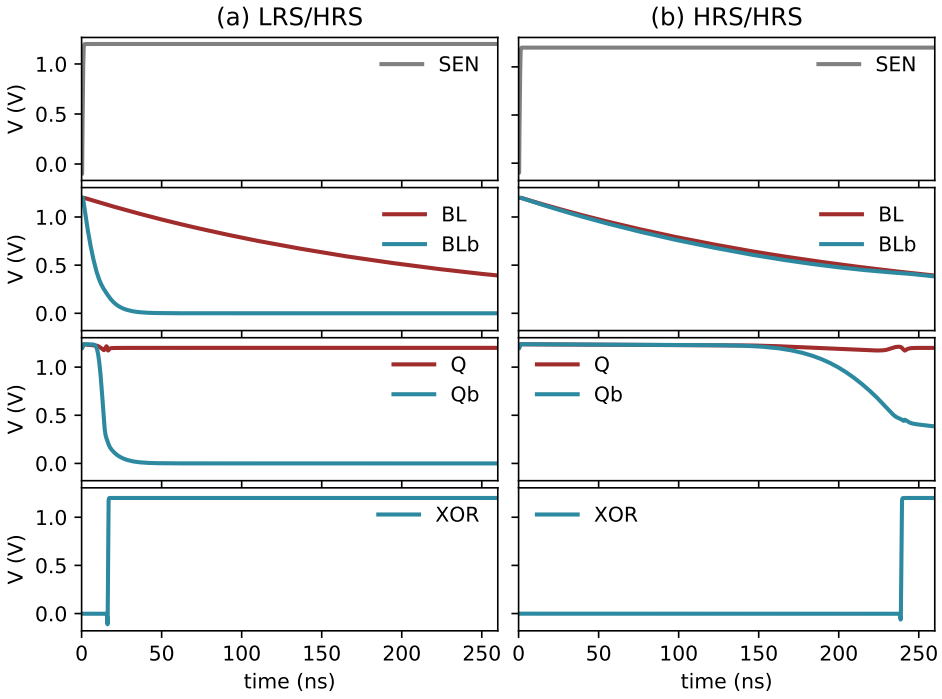



Figure 6.2: Circuit simulation of the precharge sense amplifier of Figure 6.1, if the two devices are programmed in an (a) LRS / HRS ($5k\Omega/350k\Omega$) or (b) HRS/HRS ($320k\Omega/350k\Omega$) configuration.

The sense operation is performed during a duration of $50ns$. If at the end of this period, outputs Q and Qb have differentiated, causing the output of the XOR gate to be 1, output Q determines the synaptic weight (1 or $-1$). Otherwise, the output of the XOR gate is 0, and the weight is determined to be 0. In [184], we verified this behavior experimentally.

By thinking RRAM cells as more than binary switches, this works makes a first step toward exploiting the analog features of RRAMS.

### 6.1.2 Binarized Neural Network vs Ternarized Neural Network

To confirm the strength of ternarized neural network implementation, we investigate the accuracy gain when using ternarized instead of binarized networks. We trained BNN and TNN versions of networks with Visual Geometry Group (VGG) type architectures [325] on the CIFAR-10 task of image recognition [263].

Table 6.1.2 lists the impact of weight ternarization for different types of activations (binary, ternary, and real activation). All results are averaged over five training procedures. We observe that for BNNs and TNNs with quantized activations, the accuracy gains provided by ternary weights over binary weights are 0.84 and 0.86 points and are statistically significant over the standard deviations. This accuracy gain is more important than the gain provided by ternary activations over binary activations, which is about 0.3 points. This bigger impact of weight ternarization over ternary activation may come from the ternary kernels having a better expressing power over binary kernels, which are often redundant in practical settings [241]. The gain of ternary weights drops to 0.26 points if real activation is allowed (using rectified linear unit, or ReLU, as activation function), and is not statistically significant considering the standard deviations.

| **Activations** / **Weights** | Binary | Ternary | Full Precision |
|---|---|---|---|
| Binary | $91.19 \pm 0.08$ | $91.51 \pm 0.09$ | $93.87 \pm 0.19$ |
| Ternary | $92.03 \pm 0.12$ | $92.35 \pm 0.05$ | $94.13 \pm 0.10$ |
| *Gain of ternarization* | *0.84* | *0.86* | *0.26* |

Table 6.2: Comparison of the gain in test accuracy for a $N = 128$ model size on CIFAR-10 obtained by weight ternarization instead of binarization for three types of activation quantization.

## 6.2 Synaptic Metaplasticity in Binarized Neural Networks

While we have studied binarized neural networks as especially adapted for performing inference with low computational and memory cost [225, 257, 326], and excellent accuracy on multiple vision [179, 242] and signal processing [270] tasks, they might feature more fundamental properties. In this work published in [327], we made a connection between the hidden weights and a neuroscience concept called metaplasticity [328, 329].

### 6.2.1 Catastrophic forgetting

While deep neural networks have surpassed human performance in multiple situations, they are prone to catastrophic forgetting: upon training a new task, they rapidly forget previously learned ones [330, 331]: synaptic weights optimized during former tasks are not protected against further weight updates and are overwritten, causing the accuracy of the neural network on these former tasks to plummet [332, 333] (see Figure 6.3).

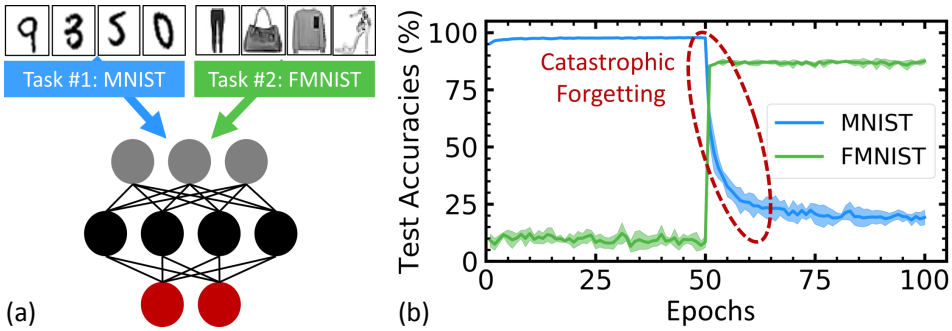



Figure 6.3: (a)Problem setting: two training sets (here MNIST and Fashion-MNIST) are presented sequentially to a fully connected neural network. (b) When learning MNIST (epochs 0 to 50), the MNIST test accuracy reaches 97%, while the Fashion-MNIST accuracy stays around 10%. When learning Fashion-MNIST (epochs 50 to 100), the associated test accuracy reaches 85% while the MNIST test accuracy collapses to $\sim$ 20% in 25 epochs: this phenomenon is known as “catastrophic forgetting”.

Neuroscience studies, based on idealized tasks, suggest that in the brain, synapses overcome this issue by adjusting their plasticity depending on their history. However, such “metaplastic” behaviours do not transfer directly to mitigate catastrophic forgetting in deep neural networks.

The neuroscience literature provides insights about underlying mechanisms in the brain that enable task retention. In particular, it was suggested by Fusi et al. [328, 329] that memory storage requires, within each synapse, hidden states with multiple degrees of plasticity. For a given synapse, the higher the value of this hidden state, the less likely this synapse is to change: it is said to be consolidated. The plasticity of the synapse itself being plastic, this behaviour is named “metaplasticity”. The metaplastic state of a synapse can be viewed as a criterion of im-

portance with respect to the tasks that have been learned throughout and therefore constitutes one possible approach to overcome catastrophic forgetting.

### 6.2.2 Interpreting the Hidden Weights of Binarized Neural Networks as Metaplasticity States

The training procedure of binarized neural networks involves a real value associated to each synapse which accumulates the gradients of the loss computed with binary weights. This real value is said to be "hidden", as, during inference, we only use its sign to get the binary weight. In this work, we interpret this hidden weight as a metaplastic variable that can be leveraged to achieve multitask learning. Based on this insight, we develop a learning strategy using binarized neural networks to alleviate catastrophic forgetting.

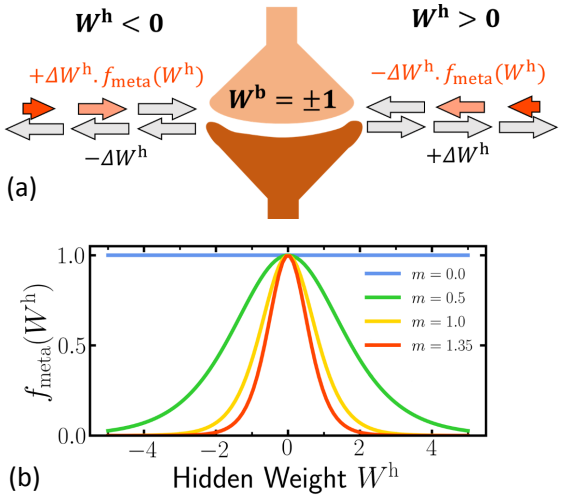



Figure 6.4: (a) Illustration of our approach: in a binarized neural network, each synapse incorporates a hidden weight $W^h$ used for learning and a binary weight $W^b = \text{sign}(W^h)$ used for inference. Our method, inspired by neuroscience works in the literature [328], amounts to regarding hidden weights as metaplastic states that can encode memory across tasks and thereby alleviate forgetting. With regards to the conventional training technique of binarized neural network, it consists in modulating some hidden weight updates by a function $f_{meta}(W^h)$ whose shape is indicated in (b). This modulation is applied to negative updates of positive hidden weights, and positive updates of negative hidden weights. $f_{meta}(|W^h|)$ being a decreasing function, this modulation makes the hidden weights signs less likely to switch back when they grow in absolute value.

An important benefit of our synapse-centric approach is that it does not require a formal separation between datasets, which also allows the possibility to learn a single task more continuously. Traditionally, if new data appears, the network needs to relearn incorporating the new data into the old data: otherwise, the network will just learn the new data and forget

what it had already learned. Through the example of the progressive learning of datasets, we show that our metaplastic binarized neural network, by contrast, can continue to learn a task when new data becomes available, without seeing the previously presented data of the dataset. This feature makes our approach particularly attractive for embedded contexts. The spatially and temporally local nature of the consolidation mechanism makes it also highly attractive for hardware implementations, in particular using neuromorphic approaches.

This approach takes a remarkably different direction than the considerable research in deep learning that is now addressing the question of catastrophic forgetting. Many proposals consist of keeping or retrieving information about the data or the model at previous tasks: using data generation [334], the storing of exemplars [335], or in preserving the initial model response in some components of the network [336]. These strategies do not seem connected to how the brain avoids catastrophic forgetting, need a very formal separation of the tasks and are not very appropriate for embedded contexts.

The mechanism that we propose is illustrated in Figure 6.4 (a), where $W^{\mathrm{h}}$ is the hidden weight and $\Delta W^{\mathrm{h}}$ is the update provided by the learning algorithm. We introduce a set of functions $f_{\mathrm{meta}}$, parameterized by a scalar $m$ and depending on the hidden weight to modulate the strength of updates prescribing to switch the sign of the binary weight. The specific choice of this set of functions is motivated by the conceptual properties that we want our model to share with the cascade model [328]. First, the functions $f_{\mathrm{meta}}$ should be chosen so that the switching strength of the binary weight decreases exponentially with the amplitude of the hidden weight. On the other hand, the switching ability should remain unaffected when the hidden weight is close to zero, making the learning process of such weights analogous to the training of a conventional binarized neural network. We therefore choose a set of functions plotted in Figure 6.4(b) that decrease exponentially to zero as the hidden weight $|W^{\mathrm{h}}|$ approaches infinity, while being flat and equal to one around zero values of $W^{\mathrm{h}}$:

$$f_{\mathrm{meta}}(m, W^{\mathrm{h}}) = 1 - \tanh^2(m \cdot W^{\mathrm{h}}). \tag{6.2}$$

The parameter $m$ controls the speed at which the decay occurs and constitutes the only hyper-parameter in our approach. All experiments in this work use adaptive moment estimation (Adam) [183]. Momentum-based training and root mean square propagation showed equivalent results. However, pure stochastic gradient descent leads to lower accuracy, as usually observed in binarized neural networks, where momentum is an important element to stabilize training [179, 225, 241].

### 6.2.3 Learning two tasks with Metaplastic Binarized Neural Networks

To test the ability of our binarized neural network to learn several tasks sequentially, we sequentially train a binarized neural network on two tasks in a difficult situation. We trained a binarized neural network with two hidden layers of 4,096 units to learn sequentially the MNIST

dataset and the Fashion-MNIST dataset [268] which consists of fashion items images belonging to ten classes. Figure 6.5 (b) shows the result of the training of a $m = 1.5$ binarized neural network, with 50 epochs on MNIST and 50 epochs on Fashion-MNIST. Figs. 6.5 (a) also show the result for the conventional binarized neural network ($m = 0$). Baselines define the accuracies the binarized neural network would have obtained had it been trained on each of these tasks separately. We observe that the metaplastic binarized neural network can learn both tasks sequentially with baseline accuracies. Our full paper [327] includes a much more in-depth study on multitask learning, and also an analysis of the possibility of stream learning.

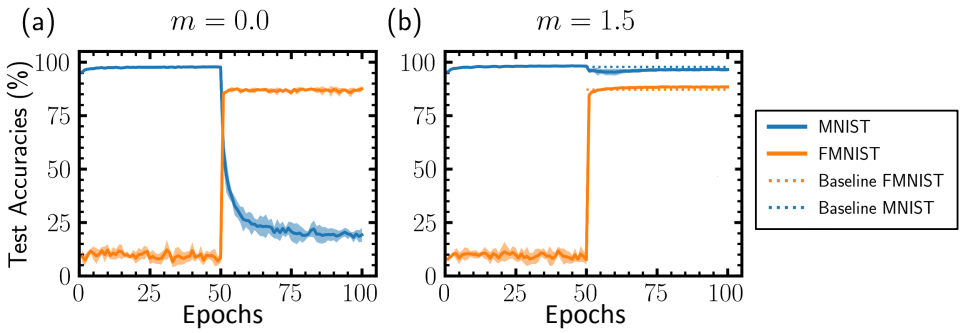



Figure 6.5: **MNIST/Fashion-MNIST sequential learning.** Binarized neural network learning MNIST and Fashion-MNIST sequentially for two values of the metaplastic parameter $m$. $m = 0$ corresponds to a conventional BNN (a), $m = 1.5$ is a metaplastic BNN (b). Curves are averaged over five runs and shadows correspond to one standard deviation

### 6.2.4 Metaplasticity using emerging nanodevices

The fact that metaplastic approaches build on synapses with rich behaviour resonates with the progress of nanotechnologies, which can provide compact and energy-efficient electronic devices able to mimic neuroscience-inspired models, employing analog memristive technologies [124, 145]. Many works in nanotechnologies have shown that a single nanometer-scale device can provide metaplastic behaviour [337–341]. The metaplasticity features of these nanodevices vary greatly depending on their underlying physics and technology, but their complexity is analogous to our proposal here. Typically, metaplasticity occurs by transforming the shape of a conductive filament continuously. These changes make the device harder to program, and therefore provide a feature that can be analogous to our continuous metaplasticity function $f_{\text{meta}}$. Our proposal could therefore be an outstanding candidate for nanotechnological implementations, as it provides rich features at the network level while remaining compatible with the constraints of technology.

Additionally, taking inspiration from the metaplastic behaviour of actual synapses of the brain resulted in a strategy where the consolidation is local in space and time. This makes this approach particularly suited for artificial intelligence dedicated hardware and neuromorphic

computing approaches, which can save considerable energy by employing circuit architectures optimized for the topology of neural network models, and therefore limiting data movements [229].

This work also evidences the benefit of taking inspiration from biology with regards to purely mathematically-motivated approaches: they tend to be naturally compatible with the constraints of hardware developments and can be amenable for the development of energy-efficient artificial intelligence.

Here we have implemented long-term memory into binarized neural networks by modifying the hidden weight update of synapses. It highlights that binarized neural networks can be more than a low precision version of deep neural networks, as well as the potential benefits of the synergy between neurosciences and machine learning research, which for instance aims to convey long-term memory to artificial neural networks.

## 6.3 Hybrid Analog-Digital Learning with Differential RRAM Synapses

In the work presented in this section, already published in [342], we investigate a learning-capable model of Binarized Neural Network which exploits the analog properties of the weak RESET in hafnium-oxide RRAM cells, but uses exclusively compact and low power digital CMOS. This approach requires no refresh process, is more robust to device imperfections than more conventional analog approaches, and we show that due to the reliance on weak RESETs, the devices show outstanding endurance that can withstand multiple learning processes.

### 6.3.1 Principle of learning

We have already presented in Chapter 1 and Chapter 4 the benefit of these RRAM devices for the implementation of neural networks for inference. In addition, a particularly attractive idea is to exploit their analog properties for learning [122, 124]: during training, following a learning rule, the synaptic weights, implemented by RRAM conductance, are adjusted sequentially until the neural network reaches maximum accuracy. This technique is compelling but faces multiple challenges. First, it requires analog CMOS circuitry or conversion between analog and digital signals, which brings high area and energy overheads [343]. Second, it is highly sensitive to RRAM imperfections such as asymmetry between SET and RESET process, non-linearity and noise [124]. Third, devices optimized for analog operation may suffer from reduced reliability and endurance [233].

As already mentioned in this Chapter, during the training of binarized neural network, a hidden real weight is also associated with binary synapses, which is adjusted by the learning

rule. This real value is encoded by the difference between the log resistance of the two devices:

$$W^h = log(R_{BL}) - log(R_{BLb}). \tag{6.3}$$

The binarized weight used by the neural network in all arithmetic operations is the sign of this hidden real weight. Once training is finished, the hidden real weight is of no use and can be discarded. For RRAM-based learning hardware, BNNs do not seem ideal at first sight, as synapses need to be associated with real weights. **However, the BNN learning process does not need to know the value of the hidden real weight, as only the sign is used in arithmetic operations.** The network only needs the ability to increase or decrease it. We show here that this is feasible in a simple manner employing RRAM cells with only an adapted programming sequence and purely digital CMOS.

For the binary value, we use the 2T2R structure already used to reduce the bit error rate in Chapter 4: if the BL cell has higher resistance than the BLb one, the 2T2R structure implements synaptic weight +1, and otherwise -1. The binary weight can therefore be obtained by comparing the resistances of the two devices using a pre-charge-sense-amplifier.

For learning, we exploit weak RESET pulses [344], which exhibit a progressive effect (Figure 6.6): when such a pulse is applied to an RRAM cell, it increases the resistance of the device slightly.

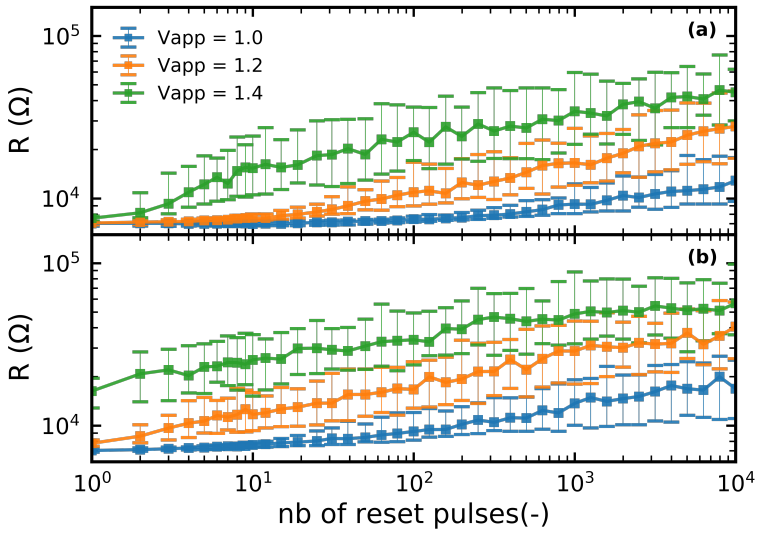



Figure 6.6: Statistical measurements of the weak RESET process over 64 devices. Error bar is one standard deviation, for different RESET voltages.
(a) tRESET=1 µs. (b) tRESET=100 ns.

Figure 6.6 shows this progressive effet of RESET pulses as statistical measurements over 64 devices. This effect is strongly dependent on the programming voltage and duration and is subject to significant noise and variability due to the atomic size of filaments in the RRAM devices. The conductance vs. pulse number is also highly nonlinear. It is however seen on all

devices that we measured. In section 6.3.4, this high noise and variability will be shown and compared with a device model.

### 6.3.2 A 2T2R complementary programming strategy

On the other hand, the SET process does not feature a progressive effect. For this reason, we mainly exploit the RESET effect for learning. Whenever the learning rule of BNNs predicts to increase a synaptic weight, we apply a weak RESET to the BL device, whereas if the learning rule predicts a decrease in the weight, we apply a weak RESET on the BLb device (Figure 6.7).

To determine the number of pulses to be applied for training, it is first of all necessary to calculate the value of the gradient. This gradient calculation is done by the backpropagation algorithm where the value of the gradient is precise. It is this gradient value that will then be transformed into several pulses to be applied. This value will be positive or negative, which will determine the choice of the device to which the pulses are applied.

This technique is particularly effective for learning in binarized neural networks, indeed, it is not the specific value of the weight that is important but the relative difference. The real value coded in the devices is therefore the number of pulses received on each of the devices, if the BL device has received more pulses than the BLb device the weight will be +1 and if it is the opposite the weight will be −1.

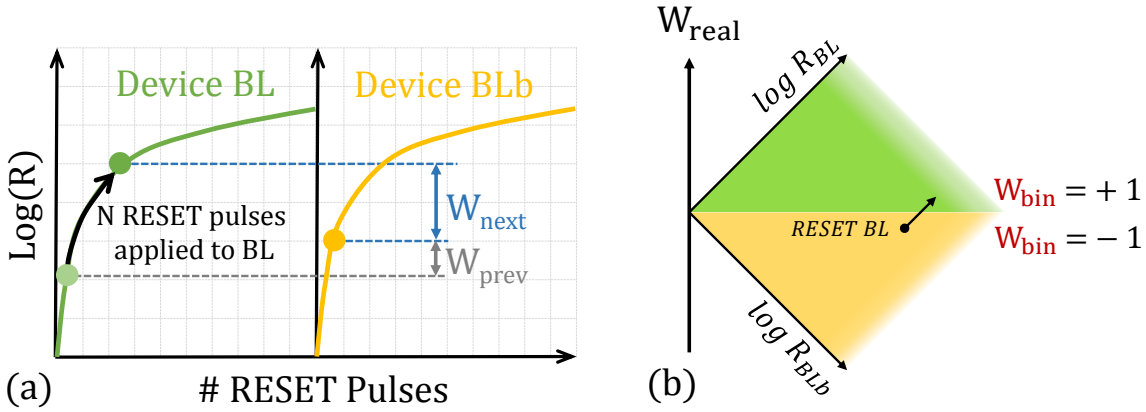



Figure 6.7: Learning procedure for an artificial synapse using a 2T2R structure in binarized neural networks. (a) The two resistances states of the two devices are presented as function of the number of pulses they received. The weight is coded as the difference between the log of the resistance of the two devices. (b) The two resistance states are presented in the form of a diamond, a dot presents the position of the actual pair of resistances. When a RESET pulse is applied to one device the position of the dot change. If the dot is in the green area the binary weight is +1, if it is in the yellow area the binary weight is -1.

To verify the learning compatibility of RRAM devices with on-chip learning, we simulated the training of a neural network with 784 inputs neurons, a hidden layer of 1024 neurons, and

10 output neurons on the MNIST handwritten recognition task. The RRAM cells were modeled with an experiment-matched compact model that will be described in section 6.3.4 and CMOS circuits with cycle-accurate modeling.

During the learning process, the two complementary devices receive RESET pulses that gradually increase their resistances. Unfortunately, this approach has a limitation, after a certain number of pulses the resistances saturate and no longer increase, which has several consequences. Firstly, when these resistances become high, they are no longer able to increase and the binary value is therefore no longer mobile. And secondly, the devices are subject to noise, when the resistance values are high the binary values will be completely stochastic.

Figure 6.8 (a) shows the effect of the saturation in term of accuracy. During the first 100 epochs, the resistance increases (or the conductance decreases) until it reaches the maximum resistance, where devices begin to saturate (Figure 6.8 (b)).

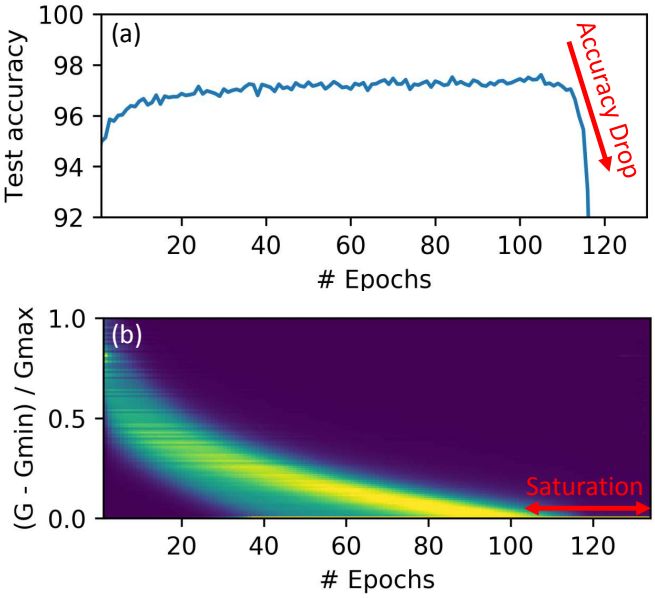



Figure 6.8: (a) Test accuracy on the MNIST task and (b) distribution of the conductance of the RRAM cells, in the neural network trained using the complementary 2T2R strategy for the learning process.

In [345], the authors solved the saturation issue by performing refresh operations after a certain number of iterations. But, this approach is time and energy-hungry and require complex programming circuits. It is indeed necessary to follow the number of updates performed from the beginning of the learning process and also because during the refresh phases it is not sure that the sign of the binary weight remains unchanged. Moreover, despite theses refreshments, performances decrease after a while.

### 6.3.3 Reprogramming check strategy to avoid saturation

Here, we propose a simpler approach based on an adapted programming sequence implemented only with pure digital circuitry (Figure 6.9). Before any write operation, we check the binary value of synapse. After the progressive RESET, we check if the binary value of the synapse has switched. If this has happened both devices are reprogrammed with a full SET and one with a weak RESET to recover an opportunity for progressive RESET.

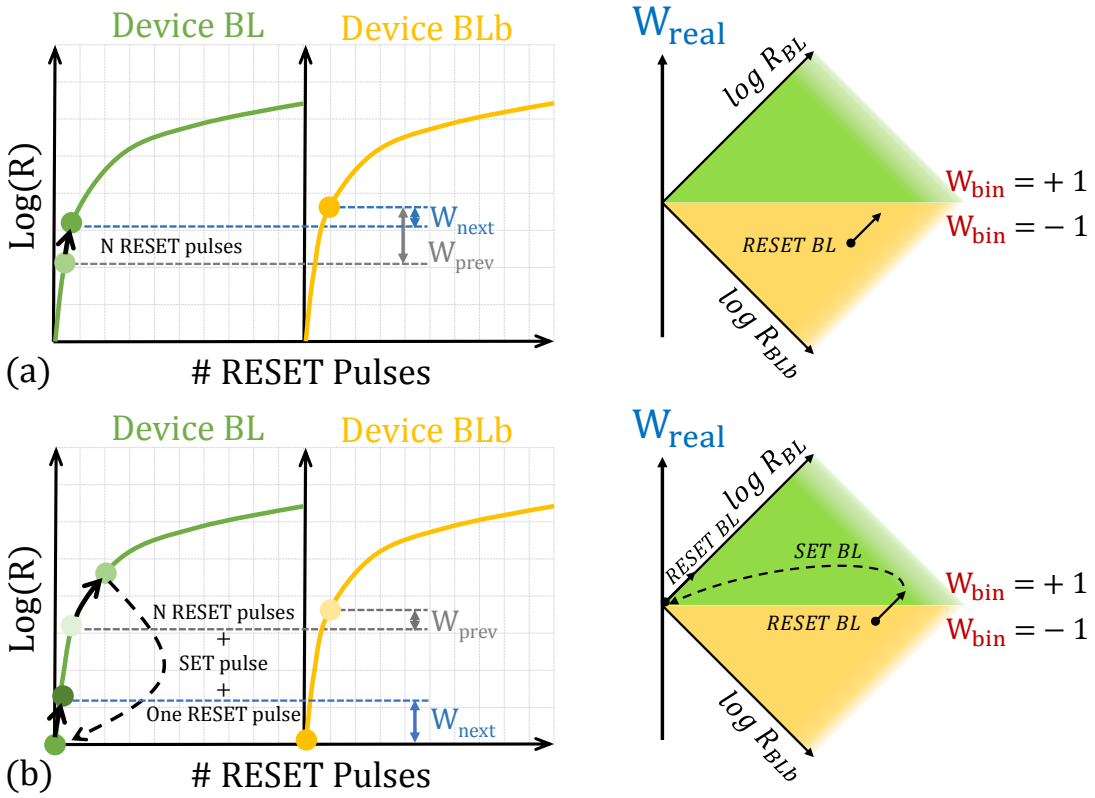



Figure 6.9: Adapted learning procedure for an artificial synapse using a 2T2R structure in binarized neural networks that require no refresh process. (a) After $n$ RESET pulses applied to the device BL, the resistance change but the sign of the binary value remains unchanged. (b) After $n$ RESET pulses applied to the device BL, the resistance change and the sign of the binary value change. We detect this change by operating a double read, a first one before the applying pulses, and a second one after the $n$ pulses applied. When then sign change happens, we apply a strong SET pulse to put the two devices in low resistance state and then we apply one pulse to the device corresponding to the device selected by the new sign of the binary weight.

We check the efficiency and robustness of our technique in practical simulation using the same neural network architecture as used in the previous section and the same learning rate resulting in the same order of magnitude of the number of pulses applied. In Figure 6.10 (a), we can see that the neural network does learn to recognize digits with 98% accuracy without any

drop in accuracy even after a large number of iterations. We can also see that the distribution of conductances is much wider than before. It is also interesting to note that when the devices begin to saturate, under the effect of the inherent noise of the devices, the sign switch occurs naturally, resulting in a strong SET pulse applied in which devices return to low resistance state.

After several iterations, the different pairs of devices tend to configure themselves in a certain order. We have seen in section 6.2.4 that hidden values can have some importance. What happens is that the real values are limited to a certain range, for W maximum $R_{BL}$ must be maximum and $R_{BLb}$ must be minimum. This means that the dynamics of the weight evolution are limited, we can see that the distributions tend to be polarised, some devices are low resistance and some devices are high resistance. The advantage of such polarisation is that the errors of the reading circuit are much lower in this case. The weakness is that the weights are less mobile and therefore, as mentioned in section 6.2.4, learning a new task becomes more complex.

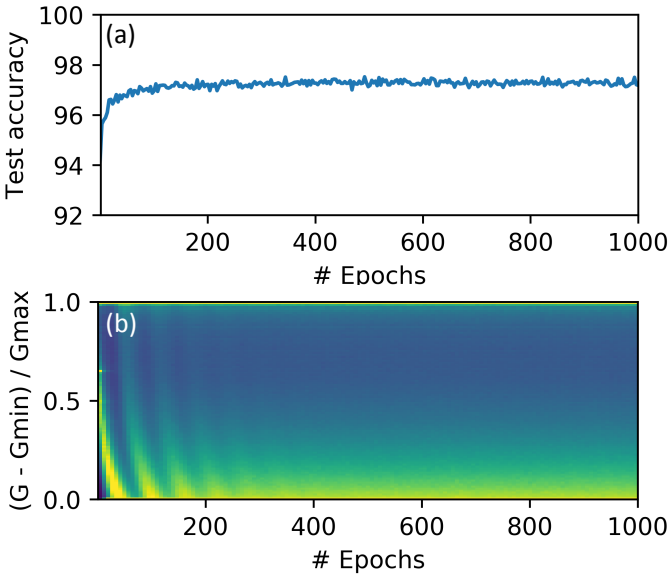



Figure 6.10: (a) Test accuracy on the MNIST task and (b) distribution of the conductance of the RRAM cells, in the neural network trained using the reprogramming check strategy. The plots are obtained with the same learning rate as Figure 6.8 but with much more iterations.

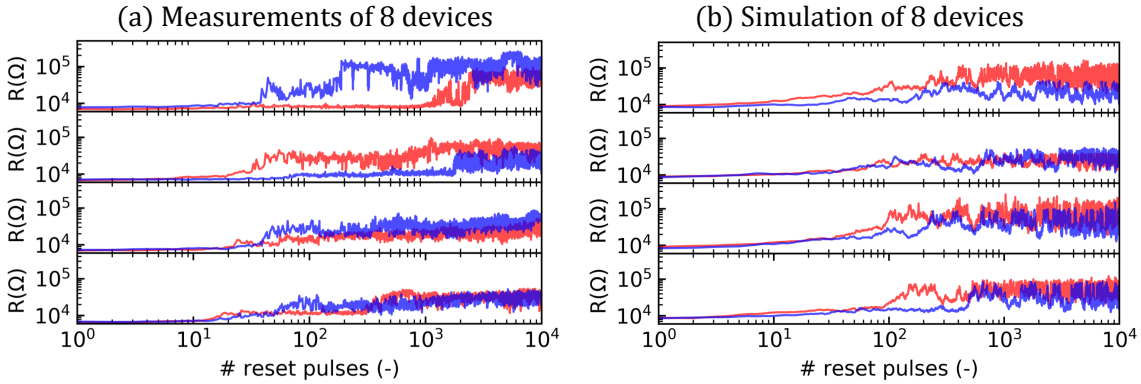
### 6.3.4 Device simulation

Our neural network simulations are performed using a behavioural RRAM device model. We used the analog progressive behaviour of RRAM devices under RESET pulses. The SET pulse does not exhibit this progressive behaviour, and we model the SET as a refresh process: under SET pulses, the resistance goes back to low resistance state under a distribution specific to each device.

The progressive RESET pulses behaviour is strongly dependent on the programming volt-

age and duration, and is subject to significant noise and variability due to the atomic size of filaments in the RRAM devices (Figure 6.11 (a)). The conductance also features a highly non-linear relationship with the number of pulses applied. This behaviour is seen in all devices that we measured.

From the measurements made on the devices, we extracted a behavioral model. Figure 6.11 (a) shows the behavior of 8 devices when RESETs are applied pulse by pulse. During programming, we can see the high noise level. The model used is inspired by [346, 347], where a noise term was added to each RESET pulse applied, resulting in the equation of Figure 6.11 (c). As illustrated in Figure 6.11 (c), this model reproduces relatively correctly the measurements made on the devices. To go further in this device modeling it would be necessary to extend this modeling study by linking different terms with physical principles, as well as to understand the nature of this noise.




(c) $$\frac{dG}{dt} = -C\exp\left(-\beta\frac{G_{max} - G(t)}{G_{max} - G_{min}}\right) + \eta Noise(t)$$

$G$ : *Device conductance* — $\eta$ : *Noise factor*
$\beta$ : *Nonlinearity of the device* — $C$ : *RESET strength*
$G_{max}/G_{min}$ : *Max/Min device conductance*

Figure 6.11: (a) Measurements of the weak RESET process on 8 randomly chosen devices. (b) Simulations of the weak RESET process. (c) Analytical equations used to model the weak RESET process in our RRAM cells. $Noise(t)$: Gaussian noise

The specific features of this model and its complexity originates from the noisy behavior. Since the equation is very non-linear, the noise does not have the same impact at low conductance as at high conductance. Therefore, we worked either with a single applied pulse or performed an iterative loop in the simulations on the number of applied pulses making the simulations relatively long.

### 6.3.5 Device imperfection, impact on accuracy

Moreover, we can look at the impact of various device issues by adjusting some parameters of the model.

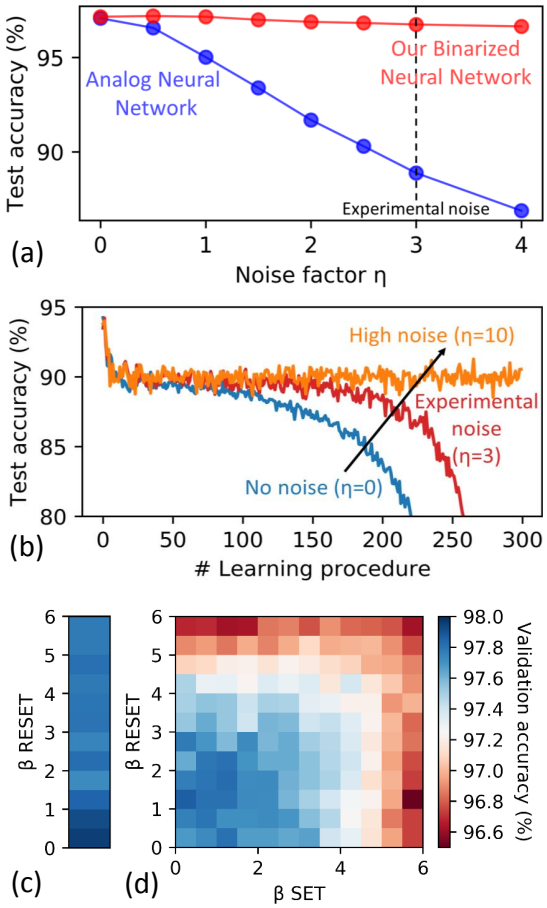



Figure 6.12: (a) Test accuracy on MNIST data-set as function of the noise factor $\eta$ for our approach and for conventional analog neural network implementing real weights with RRAM devices. (b) Test accuracy on MNIST data-set as function of the number of learning procedures. Between each learning procedure, we randomly mixed the devices pairs to see the saturation effect of the devices. When the noise is strong, the device pair do not saturate as there is some sign switch of the binary weight due to the noise. (c) Study of the effect of the $\beta$ value on the test accuracy of MNIST data-set for our reprogramming scheme procedure in binarized neural network and (d) in a classical neural network using RRAM pair of devices.

First, in Figure 6.12 (a), we simulate the neural network training for different levels of noise ($\eta$ in Figure 6.11 (c)) associated with the weak RESET process. We compare our reprogramming check strategy and a control case employing the conventional approach of using RRAM devices in a fully analog fashion. We see that with our approach, unlike the conventional one, noise is tolerated to an outstanding level. The level of noise seen in our experimental devices ($\eta = 3$ in

Figure 6.11 (a)) causes no accuracy degradation, whereas it would cause important degradation in the conventional approach.

Second, we explored the effect of the noise of the devices. Figure 6.11 (b) shows that RESET noise is not only tolerated, in some situations it can even be useful. In this particular situation, several training processes are realized cumulatively, without resetting devices in-between the processes: i.e., in this figure, the x-axis is not a number of epochs or iterations, it is the number of times the network learns. We perform a first learning procedure during one epoch (each training data is presented only once) and then we keep the conductance of the devices. To perform a new learning process, all pairs of devices are randomly mixed. As some device pairs have complementary conductances, e.g. BL is high resistance state and BLb is low resistance state, the neural network has more difficulty to learn than for the first learning process, however, it is still possible; the learning process only needs more iterations.

We see that the re-learning processes are more effective when RESET noise is stronger. This is because when there is more noise, the reprogramming SET step where both devices are programmed in a low-resistance state, occurs more often and therefore the devices do not end up in a not very mobile configuration.

Another study of device imperfection concerns the nonlinearity of the RESET process. Figure 6.12 (c) shows the impact of the nonlinearity of the RESET process ($\beta$ in equation Figure 6.11 (c)) in our approach. We did not have to study the effect of the non-linearity of the SET process for our approach as it is used only to apply strong SET to the device to low resistance state. So the SET operation only needs to be efficient to go to low resistance state but do not exhibit any progressive behaviour. To make a comparison we study the impact of the non-symmetry between the SET and RESET process in the conventional approach. We see an outstanding tolerance of our approach to these effects, whereas the conventional approach is highly sensitive to them.

### 6.3.6 Benefits of Operating in a Weak RESET Regime

For learning, a very important concern is endurance as devices will have to be programmed a large number of times until it reaches a high accuracy. The number of programming steps required for the learning can be very different from one synapse to another. Therefore, specific synapses and consequently specific devices will receive more or less programming pulses. Generally, to learn a simple task like MNIST, a device will be programmed on average several hundred times. This value increases for more difficult tasks as they generally require more iterations of learning.

The reliance of our technique on weak RESET pulses has tremendous benefits in regards to endurance. Figure 6.13 shows cycled measurements alternating strong SET and weak RESET pulses. Figure 6.13 (a) & (b) shows the cycle to cycle distribution variation of two devices. The distribution of the low resistance state does not change dramatically when the devices age, whereas the distribution of the high resistance state tends to shift towards low resistances when

the number of cycles is more than $10^{10}$.

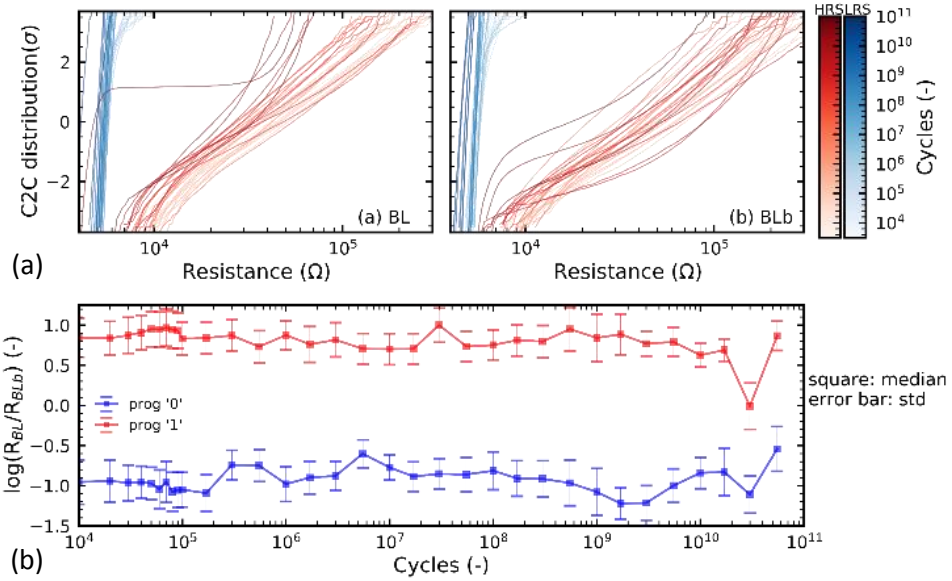



Figure 6.13: Endurance measurement on two complementary devices programmed with weak RESET (VRESET=1.5V), pulse width of 1 $\mu s$ and SET compliance current of 200 $\mu A$. (a-b) Cycle-to-cycle (C2C) distribution of resistance values for 10k cycle. (c) median value resistance ratio (RBL/RBLb), extracted over 10k cycles.

Under test, an outstanding endurance of 55 Billion cycles is seen on the two devices. This result shows that with our approach we could perform multiple learning processes on a single chip. Besides this high endurance, the great quality of our RRAM memory array is that it requires low energy consumption with voltage compatible with high-performance CMOS.

### 6.3.7 Comparison with other approaches for RRAM-Based Learning

The approach presented in this section for learning binarized neural networks is very new. It requires only two devices per synapse using only the RESET pulse programming regime. This approach is by construction resilient to non-linearity which does not affect performance. It is very resilient to programming noise and has the particularity of using only one regime of programming, therefore it is completely resilient to the asymmetry between RESET and SET pulses.

Table 6.3 compares the requirements for on-chip learning of our hybrid RRAM/CMOS reprogramming check strategy with conventional neural network and 8-bits precision approach.

All the approaches mentioned in this table focus on the technological aspect, how to implement the synapses of a neural network. But from a system point of view, the challenges of the hardware implementation of neural networks on a chip are not limited to this single aspect, there are also plenty of additional difficulties.

| | Device/ synapse | CMOS overhead | Resilience to noise | Resilience to non-linearity | Resilience to asymmetry | Requires refresh |
|---|---|---|---|---|---|---|
| Analog RRAM [122, 343] | 1 | High | Low | No | No | No |
| Analog PCM [124] | 2 | High | Low | No | Yes | Yes |
| Fully digital [348] | 8+ | Medium | Medium | Yes | Yes | No |
| [345] | 2 | Low | High | Yes | Yes | Yes |
| **This work** | **2** | **Low** | **High** | **Yes** | **Yes** | **No** |

Table 6.3: Comparison of our approach with approaches of the literature for learning with resistive memories.

### 6.3.8 Next requirement for a final on-chip learning implementation

The idea of our work is that we consider that the 2T2R differential implementation present in chapter 4 is not only interesting to reduce errors but also, as presented in section 6.3, that it allows learning.

There are various reasons to believe that the hardware implementation of binarized neural networks is particularly attractive for learning and simpler than classical neural networks:

- Their implementation for inference is very simple and energy-efficient.
- The update of the hidden value only influences the binary weight under certain conditions.
- The learning of binary neural networks has a high tolerance to noise and non-linearity of the devices unlike classical neural networks implemented with the same devices.

In chapter 4 we studied most of the hardware challenges that may exist for the inference of binary neural networks, but learning raises new questions.

#### Challenge with batch

To perform the learning process on software, the use of batches is usually required to evaluate the loss landscape. The loss landscape can also be evaluated on the whole data-set, but this learning process is very long and requires intensive calculations. To use large batch, a large amount of memory is needed: all neuron activations have to be stored for the backpropagation (see Figure 1.11).

To have a relatively fast learning and less intensive calculations, we usually use mini-batches. Nevertheless, it is possible to train neural networks by making a pure stochastic gradient decent, i.e. by presenting the inputs one by one. But this training is generally longer on GPU as it only deals with one sample at a time and therefore only with vector/matrix operations whereas GPUs function optimally with pure matrix operations. For a low-power hardware implementation, it was seen in chapters 1 and 4 that memory registers have an important role in power consumption. Therefore, for embedded systems it may be preferable to work with these vector/matrix operations rather than pure matrix operations.

In the previous sections of this chapter, we did not focus on the challenges related to batches, as we mainly focused on how to perform the update of the weights using RRAM devices. The weights were updated according to the mini-batch gradient. For a real implementation, it is important to verify that our approach works with stochastic gradient descent. This is a work in progress.

Moreover, all our binarized neural network simulations use batch-normalization, as it is known to efficiently stabilize the training [179, 241]. Batch-normalization is not, however, a fundamental element of the scheme. Normalization technique that do not involve batches, such as instance normalization [349], layer normalization [350], or online normalization [351] provide more hardware-friendly alternatives and works perfectly well with binarized neural network training.

### Quantization of gradient for backpropagation

In binarized neural networks, the inference is completely binary: between each neural network layer, only binary information is transmitted. For learning, it is different because the information transmitted between layers during backpropagation is not binarized; it has a real value. The implementation of binarized neural network is interesting only if the transmitted information is binarized for both inference and learning.

Some recent work has shown that it is possible to quantize gradients for backpropagation [352]. Even if these first results are promising, their implementation requires at least 8 bits for quantization for difficult learning tasks.

To go further, in the same way as we discuss in Chapter 4 where we use stochastic computing for the inference of the first layer of binarized neural network, some researchers have experienced the implementation of classical networks using a similar approach with spikes [353]. They realized that the gradient can be discretized into spike events for training a spiking neural network. They were able to demonstrate that even for deep networks, the gradients can be discretized sufficiently well with spikes if the gradient is properly rescaled. This form of spike-based backpropagation enables to achieve equivalent or better accuracies than comparable state-of-the-art spiking neural networks trained with full precision gradients. These spikes – binary in nature– would make it possible to transmit completely binary information for both the forward pass and the backward pass of neural networks. We will investigate this approach for the future implementation of a binarized neural network able to perform both inference and learning.

### Gradient momentum: a fundamental element and adapted learning procedure

In the learning procedure of binarized neural networks, generally, the use of the classical neural networks optimizer Adam, or momentum, in stochastic gradient descent to update the hidden weight is required. Without these methods, the performance of these neural networks is weaker and tend to have oscillations in accuracy.

As seen in section 6.2.4, where the hidden value of weights is interpreted as a factor of importance, the authors of a very recently published paper [354] also mentioned that the latent value in binarized neural network cannot be treated analogously to weights in real-valued neural networks.

The role of this hidden weight seems to be linked to the momentum (called inertia in the original paper). Making this link between hidden weight and momentum not only allows us to understand that, without momentum, the learning of binary neural networks is subject to oscillations, but also to use only one parameter for the learning process, instead of two until now. Indeed, in the learning presented so far, we did not illustrate that the evaluated gradient was related to momentum, which implied that each synapse had an additional hidden gradient value in addition to the hidden weight value.

Figure 6.14 summarises the approach, which is relatively simple. The gradient $g$ in light green corresponds to the value of the gradient calculated for a specific weight $W$, i.e. $\partial L/\partial W$. For each successive example at step $t$, this gradient is calculated and then summed up in the form of momentum according to the equation below:

$$m^{t+1} = (1-\gamma).m^t + \gamma.g \tag{6.4}$$

If the binary weight $W_b$ and the sign of this value $m$ are of the same sign and $m$ is greater than a threshold $\tau$, the binary value switches.

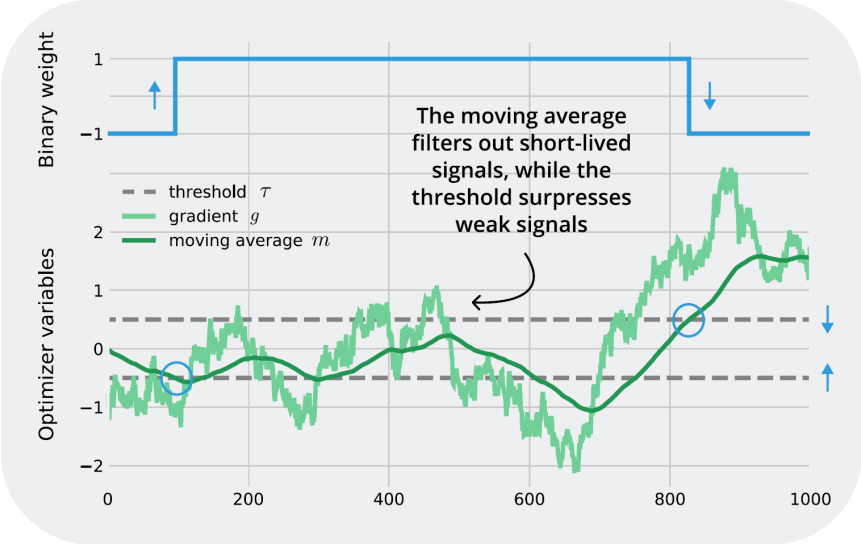



Figure 6.14: Illustration of the optimization method for training binarized neural network with only one inertia parameter $m$. Taken from [354] NeurIPS poster.

For its implementation in neural network simulations, it is very simple as it only requires to rewrite the optimizer (i.e. the update rule method). In frameworks such as Pytorch [355] or Tensorflow [356], this is done in a few lines of code. Modifying the optimizer of these frameworks makes it very easy to add hardware features for learning. When it is well written, it can

be generalized to all kinds of architectures, allowing the training of a wide variety of data sets.

In general, creating optimizers specific to binary neural networks is highly innovative. It is also a very interesting approach because we can extend these optimizers with the specificy of RRAM models.

We are currently working on this type of implementation. We are taking into account all the hardware constraints to be optimized and in particular to adapt this very efficient optimizer for the implementation of binarized neural networks to perform learning using post-programming checks.

By simulating all the dynamics of the real value of the weight with RRAMs we can hope to have a real hardware implementation of the learning process. However, other challenges have to be taken up, in particular concerning neurons activations. For instance, in deep neural networks, propagation times have to be taken into account: in order to calculate the gradient at the first layer, the information must first be transferred through the entire network a first time for the forward pass and a second time for the backward pass. Each layer will require different times to save the activations of neurons for the gradient calculation. This management of cache memory for both feedforward neural network activations and convolutions is a very interesting and difficult topic that deserves to be explored for the hardware implementation. The big advantage of binarized neural networks is that the activations that have to be saved are binary, this way the amount of memory required is relatively small.

# Conclusions and future work

"The profound study of nature is the most fertile source of mathematical discovery."

Joseph FOURIER

"BIOLOGICAL *inspiration is the approach explored in this thesis as a solution to the fundamental limitations of today's electronics circuits. Even if the working principle of the brain is not fully understood, current knowledge still allows researchers to extract useful features to implement intelligent systems for embedded system with relatively low energy. We explored several bio-inspired algorithms, following as a guideline the low energy consumption achieved by bringing calculation and memory close, using emerging memory nanodevices. "*

## Summary

The development of electronics is expected to take a fundamental turn in the next few years, mainly because the scaling laws of transistors are at the end of their lifespan. In Chapter 1, we first presented the fundamental limitations of CMOS transistors: the flagship component of today's integrated circuits. Transistors have exceptional performance in terms of computing, especially for logic functions. Regarding memory, this is not the case: transistors-based memory circuits are volatile, i.e. the memory effect disappears when the circuit is turned-off, and such memories are not very compact. To be more compact, transistor-based memories have to be manufactured using an additional capacitor which is not compatible with an implementation in the core of the high-performance computation.

For this reason, today's hardware architectures are based on the von Neumann principle, with random access memory and computing conceptually and physically separated. One of the main issues with this architecture is the energy consumption related to the data exchange between the memory and the computing core. Especially since transistor technologies have improved, this energy consumption has constituted an increasing part of the total energy consumption of the circuits.

In Chapter 1, we explored different methods for saving this energy by presenting:

- More efficient technologies for data transfer (e.g. photonics and asynchronous communication).
- Advantages and drawback of the different emerging non-volatile memory devices that can be embedded in the core of the CMOS.
- A comparison between the main characteristics of the brain and computer.
- In-Memory-Computing and neuromorphic hardware published recently.

An overview of the different approaches explored in the thesis is introduced in Chapter 2. We described two topics: Bayesian models and neural networks. Even if, these approaches remain focused on a common application, performing an inference calculation (i.e. a prediction) with a minimum of energy, they differ fundamentally. We explained the specificities of both approaches and discussed their implementation using the new emerging memory devices. We also present some recent works that make possible connections between Bayesian reasoning and neural networks in Appendix A. And in Appendix B, we describe an example mixing the two approaches.

The use of innovative memory devices is the guideline for all the chapters of this thesis. One of its the main results, presented in chapter 3, is the hardware implementation of a Bayesian reasoning algorithm, from the algorithm to the tape-out using a hybrid RRAM/CMOS technology. Firstly, we presented a theoretical description of the implemented algorithm and the hypotheses required to create the system with a very low energy cost. Secondly, we presented a new paradigm of calculation in which the final circuit operates: stochastic computing, with its

advantages and drawbacks. Finally, we presented the complete architecture, with its layout, its operating principle, both for the digital part and the programming of the RRAM devices.

The hardware implementation of intelligent algorithms continues in Chapter 4 with a study of binarized neural networks. It mainly focused on the implementation of a 2T2R design (i.e. two transistors and two RRAMs) to encode a neural network's binary-weight. An in-depth study of the impact of errors on the performance of neural networks was presented. A comparison with error-correcting codes was provided. To analyze the energy consumption of the system, a projection at the system level was performed by simulating the implementation of a complete circuit with a perspective on an MRAM-ready technology. Finally, an analysis of the stochastic approach in binary neural networks was presented.

The use of non-volatile devices in a binary fashion is something relatively simple to implement that does not fundamentally challenge all the principles of digital circuit design. In Chapter 5, we wanted to go further in the use of emerging devices by exploiting the stochastic behaviour of MRAMs. After briefly introduced works exploiting the stochastic characteristic for Bayesian neural networks, and thus making a link with our previously presented work, we introduced an alternative neuroscience theory concerning stochastic neurons. An analogy between the stochastic behaviour of superparamagnetic tunnel junctions and a population of neurons allowed us to design a hybrid STT-MRAM/CMOS chip for this new computing paradigm.

Finally, the last chapter of the thesis aimed at going even further in the use of the analog behaviour of devices. Their analog characteristic allows improving the performances of quantized neural networks, but also enables long-term memory effects and to perform on-chip learning. There is still more to be done concerning the full implementation of the learning process, both at system level and at the device level, identifying what the impacts of variability and noise are. Also, some algorithmic aspects of the learning process need to be reworked so that it can be easily implemented at the system level.

## Perspectives

In September 2020, Nvidia presented the new RTX 30 Series graphics cards[1], an announcement that is causing a stir among video game players but also computer scientists. These GPUs will be able to train even bigger neural networks! But what about this end of Moore's Law that has been predicted for a while? It seems that graphics cards keep getting better and better.

One of the reasons for the latest advances is still the improvement of transistors, but it is not the leading reason. In the latest generations of high-performance GPUs, an improvement on high bandwidth memory technology has allowed the increase of the transfer speed between the computing core and memory. This recent technology brings memory and calculation closer one to each other. The GPU architecture is also different from the one used in previous years. The major data transfer issue is therefore today largely studied by the semiconductor industry.

[1] https://www.nvidia.com/fr-fr/geforce/graphics-cards/30-series/

All our insights on how to reduce energy consumption by bringing memory and calculation closer together seem confirmed by the direction major industrial companies are following.

Even if energy consumption is an important issue, the main limitation in the development of high-performance electronic circuits is the thermal dissipation. Interesting innovations at the academic level are making great progress to tackle this issue in recent years. For instance, the use of silicon photonic optics to both increase the transfer speed and reduce the heat dissipated locally at the data bus. And possibly in the longer term, optical calculation as tensor computing cores for machine learning [357]. Another interesting approach is to cool-down microchips by integrating microfluidic cooling systems within the chip itself [358]. Nevertheless, these approaches remain relatively complex to implement, expensive and are not compatible with the prerequisites of embedded systems.

The ability to perform low-energy computing for embedded systems is the main objective of this thesis. Working under these circumstances implies constraints that high-performance electronic circuits do not have. Embedded systems require a large autonomy, a relatively small system, a limited use of external networks, and the ability to process data directly from sensors. Our core idea is to have a large memory capacity within our chip to implement artificial intelligence algorithms where memory plays a primordial role. In this way, our approach allows the reduction of the energy linked to data transfer between the computing core and the memory, but also it limits exchanges with the external network. Assuming that more and more memory can be used, a critical question appears: in which case is it better to perform a calculation explicitly or to use a memory table that has in memory the result of the calculation?

The technologies providing a very large amount of on-chip memory exist; some of them like phase change memory and ST-MRAM are already commercialized. Their characteristics are already interesting for digital uses with classical von Neumann architecture, and they should be used for realizing in-memory-computing in the near-future. To look even further, the most complex characteristics of devices could also be exploited in the future. Perhaps, by integrating more analog behaviours into current electronic circuits, we could significantly improve their performance. According to this approach, we should be able to characterize the behaviour of the devices as well as to work on the algorithm to use them optimally.

For now, most computer scientists with expertise in machine learning do not question the progress of electronic circuits and do not seek to address this type of question. And even the subject of neuromorphic circuits and chips dedicated to the calculation for artificial intelligence has only recently become a topic of discussion at prestigious computer science conferences.

The major publications in the field of artificial intelligence are based on the increase in the size of artificial intelligence models. The strength of neural networks is indeed that increasing significantly the size of models and the amount of data, seems to keep improving their performances infinitely. Today's most complex models can contain hundreds of billions of parameters [359]. In this context, what is the future of embedded systems and how can we make them

fit into AI embedded systems when models have so many parameters?

Working on neural network model optimization for embedded systems is crucial. Some architectures are well suited to the hardware implementation of convolutional networks such as the MobileNets [273] architectures, but there is still a lot of research to be done.

In this thesis, we seeked to build AI models optimised for hardware using two different approaches. On one hand, the hardware implementation of Bayesian models presented in Chapter 3 can be very efficient even with relatively few parameters and a relatively low memory overhead. On the other hand, the quantification of neural networks presented in Chapter 4 allows to elegantly reduce the number of bits needed for inference.

These challenges are therefore opening up new perspectives, especially for embedded systems, with the possibility to re-engineer the architecture of electronic circuits, integrating more data processing, new non-volatile memory devices, and sensors. The changes to be made are considerable, the decades of labour optimizing hardware architecture are difficult to reconsider (it is a question of rethinking everything!).

One opportunity to achieve this could be open-source hardware. Innovative architectures could have the same success as RISC-V [360]. This is especially relevant as IoT is emerging, opening up a whole new perspective of hardware for specific applications. Academic projects for open source hardware already exist (such as [361]) and could be massively implemented in the coming years. Open-source hardware could also be beneficial for neuromorphic circuits, providing the opportunity to develop the field very rapidly.

The neuromorphic field, despite having been initiated by Carver Mead, remains in its infancy: who knows what remains to be discovered? In my opinion, the avenues to be pursued are at the forefront of many disciplines:

- Computer science and mathematics, both to carry out simulations and to understand qualitatively the experimental results.
- Neuroscience, to explore the methods used by the brain to reason. It indeed inspires today's artificial intelligence algorithms, as we did to explore the utility of metaplastic synapses in the context of deep-learning.
- Electronics and physics, to try to reproduce certain characteristics of the brain.

For this reason, it is essential, today, to work in hardware/software co-development, otherwise, certain specific research fields could be condemned to disappear [362]. Finally, for neuroscience research, I am convinced that to understand all the underlying mechanisms of the brain we need to reproduce its main mechanisms by developing mimetic electronic circuits.

# List of publications

## Peer-Reviewed Journal Articles

✠ **Tifenn Hirtzlin**[†], Marc Bocquet[†], Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz, "Digital Biologically Plausible Implementation of Binarized Neural Networks With Differential Hafnium Oxide Resistive Memory Arrays" , *Frontiers in Neuroscience*, vol. 13, 2019. [†]: co-first authors.
doi:10.3389/fnins.2019.01383

✠ **Tifenn Hirtzlin**, Bogdan Penkovsky, Marc Bocquet, Jacques-Olivier Klein, Jean-Michel Portal, and Damien Querlioz, "Stochastic computing for hardware implementation of binarized neural networks" , *IEEE Access*, vol. 7, p. 76394–76403, 2019.
doi:10.1109/ACCESS.2019.2921104

✠ Alice Mizrahi, **Tifenn Hirtzlin**, Akio Fukushima, Hitoshi Kubota, Shinji Yuasa, Julie Grollier, and Damien Querlioz , "Neural-like computing with populations of superparamagnetic basis functions" , *Nature communications*, vol. 9, No. 1, p. 1–11, 2018.
doi:10.1038/s41467-018-03963-w

✠ Miguel Romera, Philippe Talatchian, Sumito Tsunegi, Flavio Abreu Araujo, Vincent Cros, Paolo Bortolotti, Juan Trastoy, Kay Yakushiji, Akio Fukushima, Hitoshi Kubota, Shinji Yuasa, Maxence Ernoult, Damir Vodenicarevic, **Tifenn Hirtzlin**, Nicolas Locatelli, Damien Querlioz, and Julie Grollier, "Vowel recognition with four coupled spin-torque nano-oscillators" , *Nature*, vol. 563, No. 7730, p. 230-234, 2018.
doi:10.1038/s41586-018-0632-y

✠ Axel Laborieux, Marc Bocquet, **Tifenn Hirtzlin**, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, Damien Querlioz, "Implementation of Ternary Weights with Resistive RAM Using a Single Sense Operation per Synapse" , *Transactions on Circuits and Systems I*, 2020. ***Accepted (minor revision)***

✠ Axel Laborieux, Maxence Ernoult, **Tifenn Hirtzlin**, Damien Querlioz, "Synaptic Metaplasticity in Binarized Neural Networks" . ***Under Review***

## Peer-Reviewed Conference Proceedings

✠ **Tifenn Hirtzlin**, Marc Bocquet, Maxence Ernoult, J-O Klein, Etienne Nowak, Elisa Vianello, J-M Portal, and Damien Querlioz, "Hybrid Analog-Digital Learning with Differential RRAM Synapses" , *IEEE International Electron Devices Meeting (IEDM)*, p. 22.6. 1-22.6. 4, 2019.

doi:10.1109/IEDM19573.2019.8993555

✠ Marc Bocquet†, **Tifenn Hirtzlin**†, J-O Klein, Etienne Nowak, Elisa Vianello, J-M Portal, and Damien Querlioz, "In-memory and error-immune differential rram implementation of binarized deep neural networks" , *IEEE International Electron Devices Meeting (IEDM)*, p. 20.6. 1-20.6. 4, 2018. †: co-first authors.

***Oral presentation***

doi:10.1109/IEDM.2018.8614639

✠ **Tifenn Hirtzlin**, Marc Bocquet, J-O Klein, Etienne Nowak, Elisa Vianello, J-M Portal, Damien Querlioz, "Outstanding bit error tolerance of resistive ram-based binarized neural networks" , *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, p. 288-292, 2019.

doi:10.1109/AICAS.2019.8771544

✠ Axel Laborieux, Marc Bocquet, **Tifenn Hirtzlin**, J-O Klein, L Herrera Diez, Etienne Nowak, Elisa Vianello, J-M Portal, and Damien Querlioz, "Low Power In-Memory Implementation of Ternary Neural Networks with Resistive RAM-Based Synapse" , *2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, p. 136-140, 2020.

doi:10.1109/AICAS48895.2020.9073877

✠ Bogdan Penkovsky, Marc Bocquet, **Tifenn Hirtzlin**, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, andd Damien Querlioz, "In-memory resistive ram implementation of binarized neural networks for medical applications" , *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, p. 690-695, 2020.

doi:10.23919/DATE48585.2020.9116439

✠ **Tifenn Hirtzlin**, Bogdan Penkovsky, Jacques-Olivier Klein, Nicolas Locatelli, Adrien F Vincent, Marc Bocquet, Jean-Michel Portal, and Damien Querlioz, "Implementing binarized neural networks with magnetoresistive ram without error correction" , *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, p. 1-5, 2019.

doi:10.1109/NANOARCH47378.2019.181300

✠ Philippe Talatchian, M Romera, Sumito Tsunegi, F Abreu Araujo, Vincent Cros, Paolo Bortolotti, J Trastoy, Kay Yakushiji, Akio Fukushima, H Kubota, S Yuasa, Max-

ENCE ERNOULT, DAMIR VODENICAREVIC, **TIFENN HIRTZLIN**, NICOLAS LOCATELLI, DAMIEN QUERLIOZ, JULIE GROLLIER, "Microwave neural processing and broadcasting with spintronic nano-oscillators" , *IEEE International Electron Devices Meeting (IEDM)*, p. 27.4. 1-27.4. 4, 2018.
doi:10.1109/IEDM.2018.8614585

✠ DAMIEN QUERLIOZ, **TIFENN HIRTZLIN**, JACQUES-OLIVIER KLEIN, ETIENNE NOWAK, ELISA VIANELLO, MARC BOCQUET, JEAN-MICHEL PORTAL, MIGUEL ROMERA, PHILIPPE TALATCHIAN, JULIE GROLLIER, "Memory-Centric Neuromorphic Computing With Nanodevices" , *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, p. 1-4, 2019.
doi:10.1109/BIOCAS.2019.8919010

## Conferences Without Acts

✠ **TIFENN HIRTZLIN**, MARC BOCQUET, NICOLAS LOCATELLI, ADRIEN F. VINCENT, J-O KLEIN, J-M PORTAL, AND DAMIEN QUERLIOZ, "Bit error tolerance in MRAM-based binarized neural networks" , *MRAM poster session at IEEE International Electron Devices Meeting (IEDM)*, 2018.
***Poster***

✠ **TIFENN HIRTZLIN**, MARC BOCQUET, NICOLAS LOCATELLI, ADRIEN F. VINCENT, J-O KLEIN, J-M PORTAL, AND DAMIEN QUERLIOZ, "Physics-based stochastic learning in MRAM-based binarized neural networks" , *MRAM poster session at IEEE International Electron Devices Meeting (IEDM)*, 2019.
***Poster***

✠ ALICE MIZRAHI, **TIFENN HIRTZLIN**, AKIO FUKUSHIMA, HITOSHI KUBOTA, SHINJI YUASA, MARK STILES, JULIE GROLLIER, AND DAMIEN QUERLIOZ, "Neural-like computing with populations of superparamagnetic basis functions" , *Ecole d'Hiver Fetch*, 2017.
***Poster***

✠ ALICE MIZRAHI, **TIFENN HIRTZLIN**, AKIO FUKUSHIMA, HITOSHI KUBOTA, SHINJI YUASA, MARK STILES, JULIE GROLLIER, AND DAMIEN QUERLIOZ, "Neural-like computing with populations of superparamagnetic basis functions" , *BioComp*, 2018.
***Poster***

# Appendix A

# Link between neural network and Bayesian models

*"In this appendix, we investigate some of the research work that tries to link neural network and Bayesian models together.*

*Bayesian reasoning does not really belong to a category of model but rather to a category of algorithms. Any well-defined model can then be interpreted in a Bayesian way, including neural networks. Bayesian neural networks seems to be the best of both world. For this reason, a lot of research aims at implementing deep Bayesian learning like "The Evidence Framework Applied to Classification Networks" [363], "The Bayesian interpretation of weight decay" [180], "An introduction to MCMC for machine learning" [364], "Bayesian learning via stochastic gradient Langevin dynamics" [365], "Probabilistic backpropagation for scalable learning of bayesian neural networks" [366], "Dropout as a bayesian approximation: Representing model uncertainty in deep learning" [367]. For more in-depth coverage, we recommend Yarin Gal's thesis "Uncertainty in Deep Learning" [368] and the NeurIPS workshop "Bayesian Deep Learning" [369] that is held every year."*

Here we present two techniques associating ideas from neural networks and Bayesian modeling: Variational Inference and Sum-Product Networks.

## Variational Inference

The variational inference (or variational approximations) method was originally presented by G. Hinton in 1993 [370], and modernized in 2011 by A. Graves [279]. The idea of a Bayesian neural network is to describe neural network variables as Gaussian latent variables. To make the exact but intractable inference of such Bayesian neural network with Gaussian latent variables, it is necessary to do the following. We start with a prior distribution on all the weights of the model. We then construct the correct posterior distribution of each of these parameters by multiplying the prior distribution by the probability of having the output of the training set knowing these weights. Then we normalize to have the complete posterior distribution and make new predictions on new inputs.

To obtain the posterior distribution in a tractable way, we can use a Monte Carlo method which does not impose any hypothesis on the shape of the posterior distribution. Unfortunately, the simulation time can remain very high. The method proposed by G. Hinton [370] is therefore to make a hypothesis on the posterior distribution, but to take into account the parameters distribution during training. The integration on Gaussian distributions can be done exactly and the exact derivative of the weights can be computed. The learning procedure is done by backpropagation minimizing the Kullback-Leibler divergence [371] that measures the disparity between two probability distributions. To calculate the KL divergence, a sum over the whole parameter set using random draw on the weights is necessary. The classical method to perform backpropagation through theses stochastic weights is *"the local reparameterization trick"* [372] described in Figure A.1 (a). The weight is deflected as Gaussian distribution of mean $\mu$ and standard deviation $\sigma$. Each weight has its own $\mu$ and $\sigma$ parameters that are optimized by backpropagation while a standard Gaussian random draw $\epsilon$ is performed but no error is backpropagated through it. To understand the Variational Bayes Method in more detail, we recommend the T.Broderick [373] tutorial given at ICML 2018.

## Sum-Product Networks

*"Sum-Product Networks"* (SPN) [374] is a particular architecture of neural networks. These neural networks only contain indicator variables, sums, and products. Sum-Product Networks have clear semantics, they have some interesting properties from a probabilistic and statistical perspective. Then, they do not suffer from several of the problems that other types of deep neural networks have. Figure A.1 (b) presents a picture of what a sum-product network is: the network is an acyclic directed graph, and, in the leaves, we have indicator variables or more generally univariate distributions.

Such network can be interpreted as a probabilistic graphical model like Bayesian networks [375] and Markov networks [149], but it is a class of tractable probabilistic graphical model. Such network mainly focus on computation and not correlation between variables but it can't represent all the complexity of all Graphical Models.

As every classical neural network, SPNs are also composed of a linear combination of inputs and non-linear activation. At the level of the sum nodes, SPNs perform a linear combination of the inputs. By adding a logarithmic activation function, the network behaves in a log space, which allows the product nodes to make a linear combination of the inputs. At the level of the product nodes, SPNs have an exponential activation function to go out of the log space and return to the initial linear combination space of the sum nodes. With this architecture, there is an alternation of layers with log and exponential activations and therefore can be trained by backpropagation.

Even if at first glance neural networks and Bayesian models are two completely different principles, research linking the two fields has made considerable progress in recent years. In the thesis, the objective was not to explore all the possibilities linking these two spheres, but rather to explore what possibilities are available for the hardware implementation to make an inference at a very low energy cost.

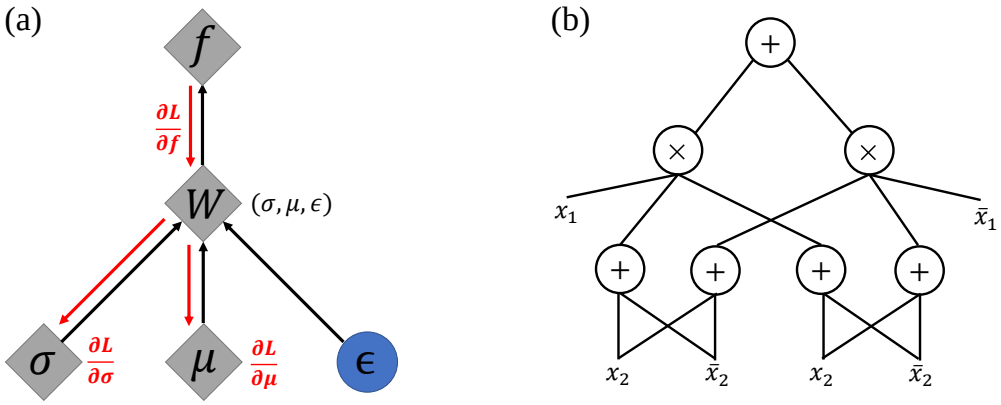



Figure A.1: (a) The local reparametrization trick [372] that allows to perform backpropagation through stochastic weights. (b) Sum-Product Network composed of Sum Nodes "+" and Product nodes "$x$"

# Appendix B

# Mixing Bayesian Inference and Binarized Neural Network: a pratical example

"NEURAL NETWORKS *are nowadays the preferred solution for artificial intelligence. They allow obtaining impressive results as long as the practitioner have a dataset including a lot of data. Even if some studies seek to understand in detail the principles underlying these very complex networks [157], they are still interpreted by many researchers as black boxes. What is impressive about neural networks is their incredible ability to model extremely complex functions. For an input set, to associate a specific answer or set of outputs with a very low error is an operation that can be done extremely efficiently. But these neural networks are very bad when there is very little data because they tend to overfit the training set and therefore do no generalize well. To overcome this weakness, we will generally reduce the complexity of the neural network when we have small amount of data. But fundamentally there is no reason why the amount of data should influence our prior knowledge about the complexity of the model used. It is more interesting to choose a model with a lot of parameters but averaged over all the parameters to know the full posterior distribution. The other question that arises is how much trust can be placed in a neural network... Multiple examples show that they can very easily be fooled [193]. "*

In this appendix, we will consider neural networks as magical black boxes and see how to use the neuron's output to perform Bayesian Inference. The neural networks considered will be Binarized Neural Network. The two main reasons for the choice of Binarized Neural Networks in this study are: first, they seem to be a bit more reliable in their response than full precision neural networks. In particular, they have been shown to be more resistant to adversarial attacks [376] than a conventional full precision neural network. The second reason, which is the focus of this thesis, is that they seem to be effective candidates for making low-energy inference with almost as good accuracy as a classical neural network. Therefore, using them as a simple feature extractor without training and analyzing their output is of very strong interest. Since Binarized Neural Networks have only two possible outputs for each neuron, counting the vectors represented by the output neurons is feasible if the number of neurons is sufficiently small.

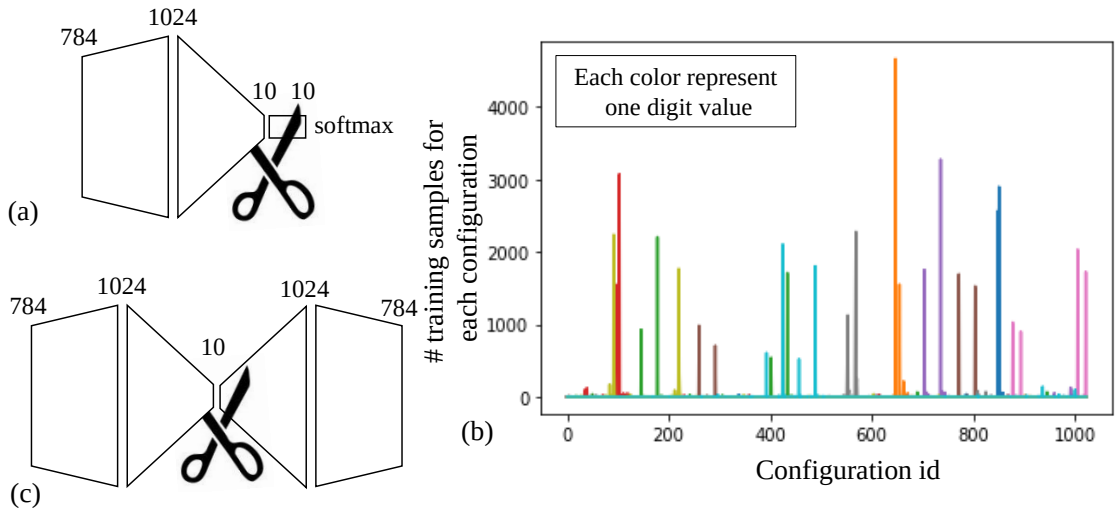



Figure B.1: Using Binarized Neural Networks as a feature extractor to train a Bayesian classifier. (a) We look at the neuron's output in the middle of an autoencoder trained by backpropagation to give from an input $X$ the same output $X$ (b) Same kind of observations but instead of looking at the sub-representation of an autoencoder we look at what the neuron's output just before the softmax layer. (c) For each corresponding configuration of neuron output vector, we plotted the corresponding number of samples of each configuration.

## Principle

The question we try to answer here is if we can use a Bayesian inference as the output of the layer of the Binarized Neural Network rather than softmax. To do so, we trained a Binarized Neural Network with a particular architecture Figure B.1 (a), we created a bottleneck at the end of the network of the same size as the number of digits, i.e. 10 here. The training of the neural network is performed using the usual backpropagation minimizing the log-likelihood

of the softmax cross-entropy loss. The choice of 10 neurons for the bottleneck is arbitrary, we use this value because the number of output configurations of the neurons is relatively small ($2^{10}$ = 1024 configurations).

We used the Binarized Neural Network as a feature extractor, and we observed the configurations in Figure B.1 (b) for each set of the handwritten digits. We can see that the configurations of the digits are very distinct from each other. There are a few configurations that overlap with each other, but in most cases, each digit has a very limited number of possible configurations. To make an exact Bayesian inference, we need to determine different probabilities. The calculation of the exact inference is expressed in Equation B.1.

$$P(label|Configuration) = \frac{P(Configuration|label)\ P(label)}{P(Configuration)} \tag{B.1}$$

The configuration corresponds to the neurons' activation vector of the layer. From the training data, or even from a smaller number of data, the different associated probabilities can be determined. The probability $P(label)$ is the prior of a particular label: if there is the same amount of images for each category of the handwritten digits, we can assume a prior of 1/10. The probability $P(Configuration|label)$ corresponds to a likelihood matrix of 1024 x 10. For each configuration, we will determine the likelihood of each configuration given the labels from the Equation B.2.

$$P(Configuration|label) = \frac{1 + \#Samples\ for\ the\ corresponding\ label}{10 + \#Samples\ for\ the\ Configuration} \tag{B.2}$$

This equation is obtained from the generalized form of Laplace's rule (or Rule of succession) [377]. The probability $P(Configuration)$ is determined by summing over all the likelihoods of each label.

By using this Bayesian inference method instead of using a softmax layer, we can have a real idea of the probabilities corresponding to each of the labels. It also allows us to work with images that change over time (i.e videos) by replacing the prior distribution by the posterior distribution of the previous instant. By looking at the posterior distribution of such a Bayesian inference, it is possible to make our model say that it does not know the answer and therefore to make no decision when it does not know what the answer is.

## Results

By making such an inference we obtain performances similar to the neural network classification with softmax output, i.e. around 97% test accuracy for both. But what is impressive is when we compare the output probability of the softmax with the output probabilities of the Bayesian inference. For instance, it is possible to choose a particular threshold probability that defines in which case the model chooses to make a decision or not. A threshold can be chosen in both cases, the Bayesian inference case and the softmax layer case. The comparison of these two methods is made in Figure B.2. When the threshold is taken in a way that the model gives an answer for 90% of the test set, the Bayesian inference has better accuracy than the softmax layer. During the training, using softmax output, we maximize the log-likelihood of the posterior but not the distribution of the posterior over all the parameter space, resulting in inconsistent probability whereas, with Bayes inference, the probability calculated are consistent.

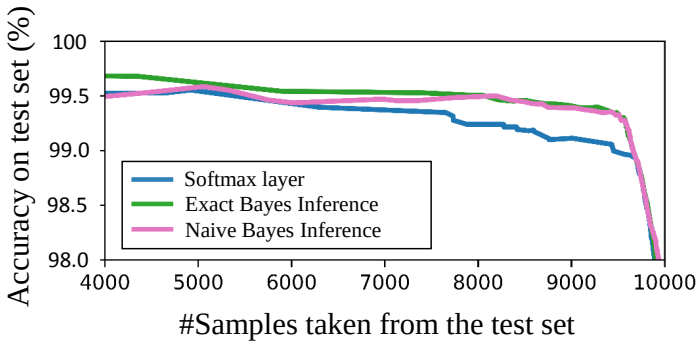



Figure B.2: Accuracy on the subtest set as a function of the number of samples taken from the test set, if the probability is smaller than a particular threshold we do not make the classification for some samples. Three cases are plotted: The softmax layer classification, The Exact Bayes Inference, and the Naive Bayes Inference. The accuracy of the exact Bayes inference is always higher than for the softmax layer. When we make the naive Bayes assumption over the neurons we can see that the accuracy is not much changed from the exact Bayes inference when the threshold is not too high, but for a higher threshold (i.e. for fewer samples taken, it looks like a lot like the softmax layer classification).

We also tried to perform this analysis with an autoencoder Figure B.1(c) rather than a fully connected neural network. The problem with the autoencoder is that it does not really try to maximize probabilities on labels, but tries to reconstruct input X. In the case of a Binarized Neural Network, the size of the latent space of the neurons plays a critical role in the ability of the autoencoder to reconstruct the input. With 10 neurons, it was not possible to have a good reconstruction of the input. Nevertheless, it was possible to have good reconstruction performances with 128 neurons. The problem is that with 128 neurons the exact Bayesian inference is already intractable, so we could only evaluate the naive Bayesian inference and not the full

Bayesian one. The performance was bad with only 79% accuracy on the test set. This low performance is explained by the strong naive hypothesis as it assumes that each neuron reacts independently from the others given the input X. The independency is very challenging as the goal of an autoencoder is to keep a maximum of information on X and can therefore correlate neurons with each other.

The use of a Bayesian interpreter at the output of neural networks can be useful in several particular cases: if a pre-trained neural network is used on data that do not match the deployment solution for a real application, to use a multiplicity of models and combine their answer, and possibly for the use of non-deterministic neural networks with stochastic outputs units and stochastic weights that can use the reparametrization trick.

We have briefly mentioned here the naive hypothesis of Bayesian inference because it allows us to contextualize our work. The details of the naive inference are presented at the beginning of the Chapter 3. We explain what it implies, and why it can be interesting to perform such an approximation. Actually, the Chapter 3 is dedicated to the hardware implementation of a Bayesian machine to perform naive Bayesian inference with a minimum amount of energy.

# Appendix C

# Synthèse en Français

"DES *années soixante-dix à nos jours l'évolution des performances des circuits électroniques a été fondée exclusivement sur l'amélioration des performances des transistors. Ce composant a des propriétés extraordinaires puisque lorsque ses dimensions sont réduites, toutes ses caractéristiques sont améliorées. Du fait de limites physiques fondamentales, la diminution des dimensions des transistors n'est plus possible aujourd'hui. De nombreuses nanotechnologies présentant des caractéristiques différentes voient le jour, ce qui constitue une opportunité pour dépasser cette problématique. L'une des voies de recherche possible est de s'inspirer du fonctionnement du cerveau biologique. Ce dernier peut accomplir des tâches complexes et variées en consommant très peu d'énergie."*

# Introduction et contexte

Les smartphones actuels ont des processeurs puissants mais leur autonomie énergétique est relativement limitée : un à deux jours pour une utilisation relativement modérée. Afin d'ajouter des fonctionnalités d'intelligence dans les circuits électroniques, il convient d'accroître cette efficacité énergétique.

Or réaliser des tâches cognitives à très faible consommation énergétique, c'est ce que fait notre cerveau puisqu'il consomme environ 20 Watts. Lorsqu'en 2016, Lee Sedol, le meilleur joueur mondial de go s'est fait battre par « alphago », un algorithme d'intelligence artificielle, il est rarement précisé que l'algorithme utilisé tournait sur des serveurs consommant plusieurs dizaines de milliers de Watts.

Dans l'optimal, on voudrait être capable d'effectuer ces même calculs sur une puce miniature tenant dans le creu de la main. Mais pour se faire, il faut non seulement être attentif à l'optimisation du calcul informatique, mais aussi à celle des capteurs, des écrans, des télécommunications et à la récupération d'énergie. Certaines applications, nécessitent de travailler sans l'intermédiaire d'une connexion réseau, par exemple les interfaces cerveau-machine en médecine, tels que :

- L'utilisation de prothèses intelligentes pour les amputations et la paralysie.
- Le suivi post-hospitalier à domicile.
- La prévention à destination des personnes âgées.
- La détection et le soin des crises épileptiques, des accidents vasculaires cérébraux ou encore des crises cardiaques.

Actuellement le traitement de ce type de données est encore réalisé sur des datas-centers mais soulève de nombreuses problématiques :

- La consommation énergétique.
- Des questions de confidentialité (Les données des utilisateurs pouvant être interceptées et collectées).
- Des problèmes de sécurité (manipulation des données, attaque par déni de service...).
- La mobilité réduite lorsqu'il n'y a pas de signal de connexion au réseau.

Pour répondre à tous ces défis, une des solutions est de travailler sur des technologies mémoires émergentes. Ainsi, en s'inspirant de la recherche en neurosciences et en prenant en compte les dernières avancées en matière d'intelligence artificielle, nous avons conçu un circuit hybride mémoires/transistors, implémentant des algorithmes de calcul en mémoire. Trois algorithmes bio-inspirés sont explorés: le raisonnement bayésien, les réseaux de neurones binaires, et une approche qui exploite davantage le comportement intrinsèque des composants, le codage en population de neurones.

## Résultats

Les limites fondamentales des transistors sont visibles aujourd'hui par la fréquence maximale de fonctionnement qui est bloquée autour de quelques GHz depuis quelques années déjà. En plus d'être très peu compacts, les circuits mémoires à base de transistors sont volatiles, l'effet mémoire disparaît lorsque le circuit est éteint. Pour être plus compacts, il faut ajouter des condensateurs qui ne sont pas compatibles avec une implémentation au cœur du calcul haute performance.

Les architectures matérielles actuelles cherchent donc à minimiser leurs utilisations, en utilisant le principe de von Neumann, consistant en une séparation conceptuelle et physique entre la mémoire vive et le calcul. Plus les technologies des transistors se sont améliorées, plus la consommation d'énergie liée à l'échange entre la mémoire et le calcul a constitué une part importante de la consommation totale des circuits.

Pour répondre à cette problématique, nous avons travaillé sur une implémentation totalement orthogonale, où mémoires et calculs sont co-localisés. Les deux technologies mémoires utilisées étant les mémoires résistives à base d'oxyde (RRAM) et les mémoires magnétiques (MRAM). Lorsque ces composants sont utilisés de façon binaire, leurs caractéristiques sont relativement semblables, seule change leur méthode de programmation, la valeur des tensions à appliquer et leur fiabilité. Mais leurs comportements physiques particuliers, nous ont également permis d'explorer de nouvelles pistes.

Deux principaux algorithmes ont été utilisés : les modèles bayésiens et les réseaux de neurones. Ces approches sont optimales pour effectuer un calcul d'inférence (c'est-à-dire une prédiction) avec un minimum d'énergie, mais sans apprentissage. Après avoir décrit les spécificités des réseaux de neurones quantifiés et plus particulièrement ceux des réseaux de neurones binarisés, nous avons présenté les différences et les potentielles connexions avec le raisonnement bayésien. Le rôle des composants mémoires émergents est primordial dans l'implémentation finale sur puce de ces deux algorithmes.

L'un des principaux résultats de cette thèse est l'implémentation matérielle de l'algorithme d'inférence bayésienne utilisant une puce hybride mémoire/transistor, on y présente :

- Un détail complet de l'implémentation matérielle de l'algorithme au tape-out.
- Les détails de programmation des composants mémoires.
- Une description théorique de l'algorithme implémenté.
- Des hypothèses nécessaires pour créer un système à très faible coût énergétique.
- L'utilisation d'un nouveau paradigme de calcul : le calcul stochastique. Avec ses avantages et ses inconvénients.

Toutes les architectures que nous avons conçues reposent sur une base commune, tel que sur la Figure C.1. Des matrices mémoires et leurs circuits de périphérie sont juxtaposés les unes avec les autres, entre chacune d'elles on dispose des circuits logiques qui se chargent des

calculs numériques.

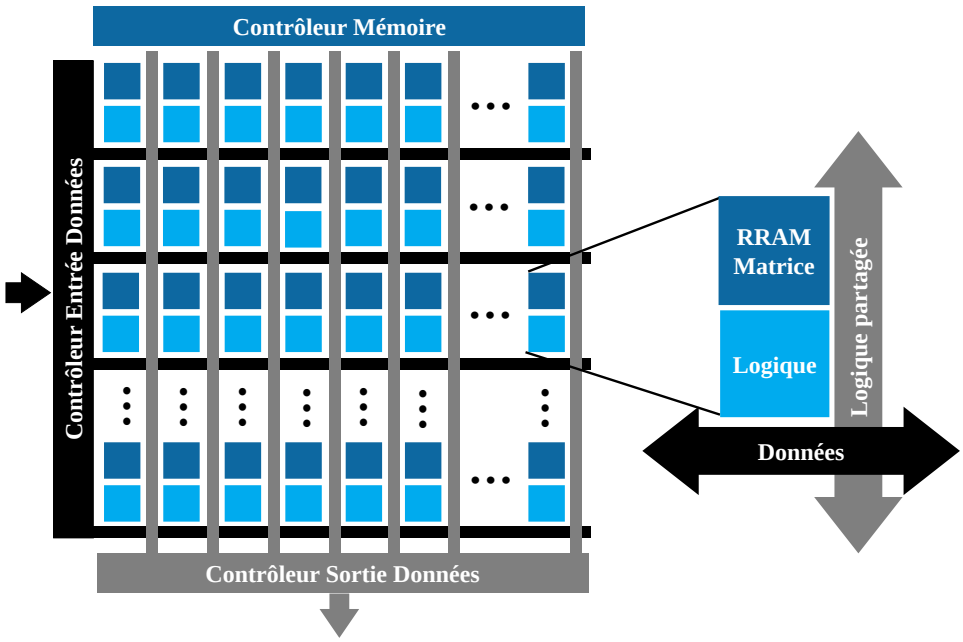



Figure C.1: Architecture classique utilisée pour concevoir un modèle de calcul en mémoire. Des signaux de contrôles se trouvent autour des matrices mémoires, le tout contrôlé par des circuits digitaux partagés entre les différents blocs constituant la puce complète.

La mise en œuvre matérielle des algorithmes intelligents se poursuit avec une étude des réseaux neuronaux binarisés. Nous nous sommes principalement concentrés sur la mise en œuvre d'une conception 2T2R, c'est à dire avec deux transistors et deux composants RRAM pour coder un poids binaire de réseau de neurones. Une étude approfondie de l'impact des erreurs sur la performance des réseaux de neurones est présentée. Tout d'abord, une comparaison de l'effet des erreurs dans la configuration 2T2R vis à vis de la configuration 1T1R. Puis, une comparaison avec les codes correcteurs d'erreurs. Nous avons pu en conclure que notre approche est aussi efficace que l'approche utilisant des codes correcteurs d'erreurs mais ne nécessite pas de circuit supplémentaire pour la détection et la correction de celles-ci.

S'il était nécessaire d'ajouter un circuit de correction pour supprimer les erreurs, dans un calcul en mémoire, il faudrait le répliquer un très grand nombre de fois. Cela nécessiterait un très grand nombre de transistors et donc on aurait un impact à la fois sur la taille du circuit mais aussi sur la consommation énergétique finale.

Afin d'analyser la consommation d'énergie, une projection au niveau du système a été réalisée en simulant la mise en œuvre d'un circuit complet avec une perspective sur une technologie commerciale MRAM. Enfin, une analyse de l'utilisation de l'approche stochastique pour la première couche des réseaux de neurones est explorée; on y montre que jusqu'à 8 présen-

tations stochastiques la consommation énergétique peut être diminuée en utilisant cette approche.

Par la suite nous avons voulu aller plus loin dans l'utilisation des dispositifs émergents en exploitant le comportement stochastique des MRAM. Après avoir brièvement présenté des travaux exploitant la caractéristique stochastique pour les réseaux neuronaux bayésiens, et faisant ainsi un lien avec nos travaux présentés précédemment, nous avons introduit une théorie des neurosciences complètement différente concernant les neurones stochastiques. Une analogie entre le comportement stochastique des jonctions tunnel super-paramagnétiques et une population de neurones nous a permis de concevoir une puce hybride STT-MRAM/CMOS dans un nouveau paradigme informatique.

Enfin, en poursuivant dans l'utilisation du comportement analogique des dispositifs, nous avons non-seulement amélioré les performances des réseaux de neurones quantifiés, mais également obtenu des effets de mémoire à long terme et réaliser un apprentissage sur puce. Il reste encore beaucoup à faire concernant la mise en œuvre complète du processus d'apprentissage, tant au niveau du système qu'au niveau du dispositif, en identifiant les impacts de la variabilité et du bruit. En outre, certains aspects algorithmiques du processus d'apprentissage doivent être retravaillés afin qu'ils puissent être facilement mis en œuvre au niveau du système.

## Perspectives

Les dernières avancées des circuits électroniques concernent toujours l'amélioration des transistors, mais ce n'est pas la seule explication. Dans les dernières générations de cartes graphiques haute performance, une amélioration de la technologie de la mémoire à large bande passante a permis d'augmenter la vitesse de transfert entre le cœur du calcul et la mémoire en les rapprochant physiquement. Toutes les idées relatives à la réduction de la consommation d'énergie en rapprochant mémoire et calcul sont validées par l'industrie.

L'objectif de nos travaux est d'améliorer la capacité à réaliser des calculs à faible consommation d'énergie pour les systèmes embarqués. Travailler dans ces conditions implique certaines contraintes que les circuits électroniques à haute performance n'ont pas. Les systèmes embarqués nécessitent une grande autonomie, un système relativement petit, une utilisation limitée des réseaux externes et la capacité de traiter les données directement à partir des capteurs. L'idée principale de notre travail est donc d'avoir une grande capacité de mémoire dans notre puce. Les technologies mémoires commencent à être commercialisées. Leur comportement non-volatile est déjà intéressant pour l'électronique numérique mais sous exploite la complexité de leurs caractéristiques. L'une des utilisations futures pourrait donc être d'tintégrer davantage d'analogique dans les circuits électroniques actuels. Ainsi, nous pourrions améliorer sensiblement les performances de ces circuits.

Même si elle semble relativement compliquée à mettre en œuvre, utiliser des composants analogiques n'est pas une nouveauté. L'utilisation de transistors dans leur régime analogique

est déjà extrêmement courante. L'intégration de nouveaux composants dans la palette de conception des ingénieurs de circuits analogiques est donc quelque chose qui peut être réalisable. Pour l'instant, les informaticiens spécialisés dans l'intelligence artificielle ne remettent pas en question les progrès des circuits électroniques. Le sujet des circuits et puces neuromorphiques dédiés au calcul pour l'intelligence artificielle n'est devenu que récemment un sujet de discussion lors des prestigieuses conférences d'informatique.

Ce qui est intéressant, c'est que la plupart des avancées dans le domaine de l'intelligence artificielle est basée sur l'augmentation de la taille des modèles. En effet, le grand intérêt des réseaux de neurones est que l'augmentation de la taille des modèles ainsi celle de la quantité de données, semble toujours améliorer leurs performances.

Les modèles les plus complexes d'aujourd'hui peuvent contenir des centaines de milliards de paramètres. Dans ce contexte, quel est l'avenir des systèmes embarqués et comment peuvent-ils intégrer ces modèles ayant tant de paramètres ? Comme les architectures MobileNets qui sont particulièrement bien adaptées à la mise en œuvre matérielle des réseaux convolutionnels, il y a beaucoup de recherches à faire sur l'optimisation des modèles de réseaux neuronaux. Les deux approches étudiées dans cette thèse cherchent à construire de petits systèmes d'IA, notre implémentation matérielle des modèles bayésiens peut être très efficace même avec relativement peu de paramètres et une surcharge mémoire relativement faible. D'autre part, la quantification des réseaux de neurones présentée permet de réduire élégamment le nombre de bits nécessaires à l'inférence.

Ces algorithmes, ont la capacité d'intégrer davantage de traitement des données, de nouveaux composants mémoires non volatiles, en intégrant des capteurs, et notamment en le faisant sur une puce spécifique. Le paradigme informatique utilisé jusqu'à présent centré sur l'architecture von Neumann est difficile à reconsidérer puisqu'il s'agit de repenser tous les circuits développés pendant des dizaines d'années et ce par des milliers d'ingénieurs.

Une opportunité pour y parvenir pourrait être le hardware open-source. Des architectures innovantes pourraient avoir le même succès que RISC-V. Cela est d'autant plus pertinent que l'Internet des Objets est en train d'émerger, ouvrant une toute nouvelle perspective de matériel pour des applications spécifiques. Des projets universitaires existent déjà et pourraient être massivement mis en œuvre dans les années à venir.

Le hardware open-source pourrait également être utile pour les circuits neuromorphiques, ce qui permettrait de développer très rapidement ce domaine. Même s'il est difficile de définir exactement ce qu'est le domaine neuromorphique, notamment, dans notre travail, les approches sont extrêmement différentes les unes des autres. Mais rendre libre les architectures inspirées du cerveau pourraient venir alimenter le domaine de nouvelles idées.

Pour les grandes entreprises technologiques, les réseaux neuronaux artificiels est le principal domaine de recherche dans lequel des progrès dans la mise en œuvre du matériel restent à accomplir. Même si les algorithmes de rétro-propagation du gradient sont largement utilisés depuis des années, les architectures dédiées à un apprentissage efficace sont relativement

peu développées à l'échelle industrielle, à l'exception du TPU de Google, mais ils n'utilisent que la technologie CMOS. C'est surtout dans le domaine des systèmes embarqués qu'il existe des opportunités. Les contraintes sont si importantes qu'elles permettent de réfléchir à des idées novatrices, et le fait que les conceptions ASIC soient spécifiques à une application nous permet de les développer davantage. Le domaine du neuromorphique, bien qu'il ait été initié par Carver Mead, en est encore à ses débuts. Qui sait ce qui reste à découvrir ? Les pistes à poursuivre sont à la frontière de nombreuses disciplines:

- Informatique et mathématiques, à la fois pour effectuer des simulations et pour comprendre qualitativement les résultats expérimentaux.
- Neuroscience, pour explorer les méthodes utilisées par le cerveau pour raisonner dont certains nouveaux algorithmes d'intelligence artificielle en sont inspiré. Nous l'avons également fait pour explorer l'utilité des synapses métaplastiques dans le contexte de l'apprentissage profond.
- Électronique et physique, pour tenter de reproduire certaines caractéristiques du cerveau.

C'est pourquoi il est essentiel, aujourd'hui, de travailler au co-développement matériel/logiciel, faute de quoi certains domaines de recherche spécifiques pourraient être condamnés à disparaître. Enfin, pour la recherche en neurosciences, je suis convaincu que pour comprendre tous les mécanismes sous-jacents du cerveau, nous devons reproduire ses principaux mécanismes en développant des circuits électroniques mimétiques.

# Bibliography


[1] Bo Lojek. *History of semiconductor engineering.* Springer, 2007.

[2] Mohamed Arafa, Bahaa Fahim, Sailesh Kottapalli, Akhilesh Kumar, Lily P Looi, Sreenivas Mandava, Andy Rudoff, Ian M Steiner, Bob Valentine, Geetha Vedaraman, et al. Cascade lake: Next generation intel xeon scalable processor. *IEEE Micro*, 39(2):29–36, 2019.

[3] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. *VLSI physical design: from graph partitioning to timing closure.* Springer Science & Business Media, 2011.

[4] Xiaoqing Xu, Nishi Shah, Andrew Evans, Saurabh Sinha, Brian Cline, and Greg Yeric. Standard cell library design and optimization methodology for asap7 pdk. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 999–1004. IEEE, 2017.

[5] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[6] Shahab Siddiqui, Takashi Ando, Rajan K Pandey, and Dominic Schepis. Limits of gate dielectrics scaling. In *Handbook of Thin Film Deposition*, pages 107–145. Elsevier, 2018.

[7] David J Frank, Robert H Dennard, Edward Nowak, Paul M Solomon, Yuan Taur, and Hon-Sum Philip Wong. Device scaling limits of si mosfets and their application dependencies. *Proceedings of the IEEE*, 89(3):259–288, 2001.

[8] Karl Rupp. 40 years of microprocessor trend data. In *GitHub*, 2018.

[9] Kioan Cheon. Water cooling type cooling system for electronic device, January 15 2004. US Patent App. 10/241,126.

[10] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual international symposium on computer architecture (ISCA)*, pages 365–376. IEEE, 2011.

[11] Kelin J Kuhn. Considerations for ultimate cmos scaling. *IEEE transactions on Electron Devices*, 59(7):1813–1828, 2012.

[12] Nicolas Planes, Oliver Weber, V Barral, S Haendler, D Noblet, D Croain, M Bocat, P-O Sassoulas, X Federspiel, A Cros, et al. 28nm fdsoi technology platform for high-speed low-voltage digital applications. In *2012 Symposium on VLSI technology (VLSIT)*, pages 133–134. IEEE, 2012.

[13] Xuejue Huang, Wen-Chin Lee, Charles Kuo, Digh Hisamoto, Leland Chang, Jakub Kedzierski, Erik Anderson, Hideki Takeuchi, Yang-Kyu Choi, Kazuya Asano, et al. Sub 50-nm finfet: Pmos. In *International Electron Devices Meeting 1999. Technical Digest (Cat. No. 99CH36318)*, pages 67–70. IEEE, 1999.

[14] Mark Neisser and Stefan Wurm. Itrs lithography roadmap: 2015 challenges. *Advanced Optical Technologies*, 4(4):235–240, 2015.

[15] Damien Querlioz. "nanoarchitectures" (circuit and system architectures that use nanodevices). In *2017 Nanosciences and Integrated Circuits Master programs*. Université Paris-Saclay, 2018.

[16] Mu-Yue Hsiao. A class of optimal minimum odd-weight-column sec-ded codes. *IBM Journal of Research and Development*, 14(4):395–401, 1970.

[17] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2013.

[18] Vivienne Sze. Efficient processing of deep neural networks: from algorithms to hardware architectures. In *2019 Conference on Neural Information Processing Systems (NeurIPS), Invited Tutorial*. NeurIPS, 2019.

[19] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

[20] Ardavan Pedram, Stephen Richardson, Mark Horowitz, Sameh Galal, and Shahar Kvatinsky. Dark memory and accelerator-rich system optimization in the dark silicon era. *IEEE Design & Test*, 34(2):39–50, 2016.

[21] Abu Sebastian, Tomas Tuma, Nikolaos Papandreou, Manuel Le Gallo, Lukas Kull, Thomas Parnell, and Evangelos Eleftheriou. Temporal correlation detection using computational phase-change memory. *Nature Communications*, 8(1):1–10, 2017.

[22] John Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[23] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *arXiv preprint arXiv:1401.0765*, 2014.

[24] James E Smith. A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*, pages 202–215, 1998.

[25] H-S Philip Wong, Kerem Akarvardar, Dimitri Antoniadis, Jeffrey Bokor, Chenming Hu, Tsu-Jae King-Liu, Subhasish Mitra, James D Plummer, and Sayeef Salahuddin. A density metric for semiconductor technology [point of view]. *Proceedings of the IEEE*, 108(4):478–482, 2020.

[26] Shahid H Bokhari. Partitioning problems in parallel, pipeline, and distributed computing. *IEEE transactions on Computers*, 37(1):48–57, 1988.

[27] Qing Wu, Massound Pedram, and Xunwei Wu. Clock-gating and its application to low power design of sequential circuits. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 47(3):415–420, 2000.

[28] Jens Spars and Steve Furber. *Principles asynchronous circuit design*. Springer, 2002.

[29] Kees Van Berkel. Handshake circuits: an asynchronous architecture for vlsi programming. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):391–397, 1993.

[30] Barry Pangrle. Asynchronous design is it time yet? In *2015 Semiconductor Engineering*, 2015.

[31] Wim Bogaerts, Martin Fiers, and Pieter Dumon. Design challenges in silicon photonics. *IEEE Journal of Selected Topics in Quantum Electronics*, 20(4):1–8, 2013.

[32] Dessislava Nikolova, Sébastien Rumley, David Calhoun, Qi Li, Robert Hendry, Payman Samadi, and Keren Bergman. Scaling silicon photonic switch fabrics for data center interconnection networks. *Optics express*, 23(2):1159–1175, 2015.

[33] Bahram Jalali and Sasan Fathpour. Silicon photonics. *Journal of lightwave technology*, 24(12):4600–4615, 2006.

[34] DJ Richardson, JM Fini, and Lynn E Nelson. Space-division multiplexing in optical fibres. *Nature Photonics*, 7(5):354, 2013.

[35] Stefan Alexander Maier. *Plasmonics: fundamentals and applications*. Springer Science & Business Media, 2007.

[36] M Mitchell Waldrop. More than moore. *Nature*, 530(7589):144–148, 2016.

[37] Johann Knechtel, Ozgur Sinanoglu, Ibrahim Abe M Elfadel, Jens Lienig, and Cliff CN Sze. Large-scale 3d chips: Challenges and solutions for design automation, testing, and trustworthy integration. *IPSJ Transactions on System LSI Design Methodology*, 10:45–62, 2017.

[38] Makoto Motoyoshi. Through-silicon via (tsv). *Proceedings of the IEEE*, 97(1):43–48, 2009.

[39] Masahiro Sunohara, Takayuki Tokunaga, Takashi Kurihara, and Mitsutoshi Higashi. Silicon interposer with tsvs (through silicon vias) and fine multilayer wiring. In *2008 58th Electronic Components and Technology Conference*, pages 847–852. IEEE, 2008.

[40] Dong Uk Lee, Kyung Whan Kim, Kwan Weon Kim, Hongjung Kim, Ju Young Kim, Young Jun Park, Jae Hwan Kim, Dae Suk Kim, Heat Bit Park, Jin Wook Shin, et al. 25.2 a 1.2 v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 432–433. IEEE, 2014.

[41] Chang-Chi Lee, CP Hung, Calvin Cheung, Ping-Feng Yang, Chin-Li Kao, Dao-Long Chen, Meng-Kai Shih, Chien-Lin Chang Chien, Yu-Hsiang Hsiao, Li-Chieh Chen, et al. An overview of the development of a gpu with integrated hbm on silicon interposer. In *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*, pages 1439–1444. IEEE, 2016.

[42] JH Smith, S Montague, JJ Sniegowski, JR Murray, and PJ McWhorter. Embedded micromechanical devices for the monolithic integration of mems with cmos. In *Proceedings of International Electron Devices Meeting*, pages 609–612. IEEE, 1995.

[43] Carlos Ríos, Matthias Stegmaier, Peiman Hosseini, Di Wang, Torsten Scherer, C David Wright, Harish Bhaskaran, and Wolfram HP Pernice. Integrated all-photonic non-volatile multi-level memory. *Nature Photonics*, 9(11):725, 2015.

[44] Max M Shulaker, Gage Hills, Rebecca S Park, Roger T Howe, Krishna Saraswat, H-S Philip Wong, and Subhasish Mitra. Three-dimensional integration of nanotechnologies for computing and data storage on a single chip. *Nature*, 547(7661):74–78, 2017.

[45] S Aggarwal, H Almasi, M DeHerrera, B Hughes, S Ikegawa, J Janesky, HK Lee, H Lu, FB Mancoff, K Nagel, et al. Demonstration of a reliable 1 gb standalone spin-transfer torque mram for industrial applications. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 2–1. IEEE, 2019.

[46] K Lee, JH Bak, YJ Kim, CK Kim, A Antonyan, DH Chang, SH Hwang, GW Lee, NY Ji, WJ Kim, et al. 1gbit high density embedded stt-mram in 28nm fdsoi technology. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 2–2. IEEE, 2019.

[47] H Ishiuchi, T Yoshida, H Takato, K Tomioka, K Matsuo, H Momose, S Sawada, K Yamazaki, and K Maeguchi. Embedded dram technologies. In *International Electron Devices Meeting. IEDM Technical Digest*, pages 33–36. IEEE, 1997.

[48] Sparsh Mittal, Jeffrey S Vetter, and Dong Li. A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches. *IEEE Transactions on Parallel and Distributed Systems*, 26(6):1524–1537, 2014.

[49] Hang-Ting Lue, Tzu-Hsuan Hsu, Yi-Hsuan Hsiao, SP Hong, MT Wu, FH Hsu, NZ Lien, Szu-Yu Wang, Jung-Yu Hsieh, Ling-Wu Yang, et al. A highly scalable 8-layer 3d vertical-gate (vg) tft nand flash using junction-free buried channel be-sonos device. In *2010 Symposium on VLSI Technology*, pages 131–132. IEEE, 2010.

[50] Micron Technology TM. 3d x-point technology. https://www.micron.com/products/advanced-solutions/3d-xpoint-technology.

[51] Richard F Freitas and Winfried W Wilcke. Storage-class memory: The next storage system technology. *IBM Journal of Research and Development*, 52(4.5):439–447, 2008.

[52] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[53] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[54] Sascha Vongehr and Xiangkang Meng. The missing memristor has not been found. *Scientific reports*, 5:11657, 2015.

[55] WJ Gallagher, Eric Chien, Tien-Wei Chiang, Jian-Cheng Huang, Meng-Chun Shih, CY Wang, Chih-Hui Weng, Sean Chen, Christine Bair, George Lee, et al. 22nm stt-mram for reflow and automotive uses with high yield, reliability, and magnetic immunity and with performance and shielding options. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 2–7. IEEE, 2019.

[56] Binh Q Le, Alessandro Grossi, Elisa Vianello, Tony Wu, Giusy Lama, Edith Beigne, H-S Philip Wong, and Subhasish Mitra. Resistive ram with multiple bits per cell: Array-level demonstration of 3 bits per cell. *IEEE Transactions on Electron Devices*, 66(1):641–646, 2018.

[57] André Chanthbouala, Arnaud Crassous, Vincent Garcia, Karim Bouzehouane, Stéphane Fusil, Xavier Moya, Julie Allibe, Bruno Dlubak, Julie Grollier, Stephane Xavier, et al. Solid-state memories based on ferroelectric tunnel junctions. *Nature nanotechnology*, 7(2):101–104, 2012.

[58] Radu Berdan, Takao Marukame, Kensuke Ota, Marina Yamaguchi, Masumi Saitoh, Shosuke Fujii, Jun Deguchi, and Yoshifumi Nishi. Low-power linear computation using nonlinear ferroelectric tunnel junction memristors. *Nature Electronics*, pages 1–8, 2020.

[59] Geoffrey W Burr, Rohit S Shenoy, Kumar Virwani, Pritish Narayanan, Alvaro Padilla, Bülent Kurdi, and Hyunsang Hwang. Access devices for 3d crosspoint memory. *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, 32(4):040802, 2014.

[60] Fabio Pellizzer and Agostino Pirovano. Phase change memory with ovonic threshold switch, March 30 2010. US Patent 7,687,830.

[61] D Garbin, W Devulder, R Degraeve, GL Donadio, S Clima, K Opsomer, A Fantini, D Cellier, WG Kim, M Pakala, et al. Composition optimization and device understanding of si-ge-as-te ovonic threshold switch selector with excellent endurance. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 35–1. IEEE, 2019.

[62] Jean-Michel Portal, Marc Bocquet, Mathieu Moreau, Hassen Aziza, Damien Deleruyelle, Yue Zhang, Wang Kang, Jacques-Olivier Klein, YG Zhang, Claude Chappert, et al. An overview of non-volatile flip-flops based on emerging memory technologies. *J. Electron. Sci. Technol*, 12(2):173–181, 2014.

[63] Weisheng Zhao, Eric Belhaire, and Claude Chappert. Spin-mtj based non-volatile flip-flop. In *2007 7th IEEE Conference on Nanotechnology (IEEE NANO)*, pages 399–402. IEEE, 2007.

[64] Albert Lee, Chieh-Pu Lo, Chien-Chen Lin, Wei-Hao Chen, Kuo-Hsiang Hsu, Zhibo Wang, Fang Su, Zhe Yuan, Qi Wei, Ya-Chin King, et al. A reram-based nonvolatile flip-flop with self-write-termination scheme for frequent-off fast-wake-up nonvolatile processors. *IEEE Journal of Solid-State Circuits*, 52(8):2194–2207, 2017.

[65] Weisheng Zhao, Claude Chappert, Virgile Javerliac, and Jean-Pierre Noziere. High speed, high stability and low power sensing amplifier for mtj/cmos hybrid logic circuits. *IEEE Transactions on Magnetics*, 45(10):3784–3787, 2009.

[66] Weisheng Zhao, Mathieu Moreau, Erya Deng, Yue Zhang, Jean-Michel Portal, Jacques-Olivier Klein, Marc Bocquet, Hassen Aziza, Damien Deleruyelle, Christophe Muller, et al. Synchronous non-volatile logic gate design based on resistive switching memories. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(2):443–454, 2013.

[67] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Magic—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[68] Zhong Sun, Giacomo Pedretti, Elia Ambrosi, Alessandro Bricalli, Wei Wang, and Daniele Ielmini. Solving matrix equations in one step with cross-point resistive arrays. *Proceedings of the National Academy of Sciences*, 116(10):4123–4128, 2019.

[69] M Kharbouche-Harrari, G Di Pendina, R Wacquez, B Dieny, D Aboulkassimi, J Postel-Pellerin, and J-M Portal. Light-weight cipher based on hybrid cmos/stt-mram: Power/area analysis. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[70] DRB Ly, JP Noel, B Giraud, P Royer, E Esmanhotto, N Castellani, T Dalgaty, J-F Nodin, C Fenouillet-Beranger, E Nowak, et al. Novel 1t2r1t rram-based ternary content addressable memory for large scale pattern recognition. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 35–5. IEEE, 2019.

[71] Tony F Wu, Haitong Li, Ping-Chen Huang, Abbas Rahimi, Jan M Rabaey, H-S Philip Wong, Max M Shulaker, and Subhasish Mitra. Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 492–494. IEEE, 2018.

[72] Geethan Karunaratne, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abbas Rahimi, and Abu Sebastian. In-memory hyperdimensional computing. *Nature Electronics*, Jun 2020.

[73] John Von Neumann and Ray Kurzweil. *The computer and the brain*. Yale University Press, 2012.

[74] Wright brothers' invention process (1899 - 1902) - researched how things fly. https://wright.nasa.gov/researched.htm.

[75] JB Furness. Types of neurons in the enteric nervous system. *Journal of the autonomic nervous system*, 81(1-3):87–96, 2000.

[76] Christopher D Harvey and Karel Svoboda. Locally dynamic synaptic learning rules in pyramidal neuron dendrites. *Nature*, 450(7173):1195–1200, 2007.

[77] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *Elife*, 6:e22901, 2017.

[78] Panayiota Poirazi and Athanasia Papoutsi. Illuminating dendritic function with computational models. *Nature Reviews Neuroscience*, pages 1–19, 2020.

[79] Joao Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic error backpropagation in deep cortical microcircuits. *arXiv preprint arXiv:1801.00062*, 2017.

[80] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[81] Hugh R Wilson and Jack D Cowan. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical journal*, 12(1):1–24, 1972.

[82] William H Calvin and CHARLES F Stevens. Synaptic noise and other sources of randomness in motoneuron interspike intervals. *Journal of neurophysiology*, 31(4):574–587, 1968.

[83] Semir Zeki. A massively asynchronous, parallel brain. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370(1668):20140174, 2015.

[84] Alexandre Payeur, Jordan Guerguiev, Friedemann Zenke, Blake Richards, and Richard Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *bioRxiv*, 2020.

[85] Eric R Kandel, James H Schwartz, Thomas M Jessell, Department of Biochemistry, Molecular Biophysics Thomas Jessell, Steven Siegelbaum, and AJ Hudspeth. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[86] Rufin Van Rullen and Simon J Thorpe. Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. *Neural computation*, 13(6):1255–1283, 2001.

[87] Simon J Thorpe. Spike arrival times: A highly efficient coding scheme for neural networks. *Parallel processing in neural systems*, pages 91–94, 1990.

[88] Gyorgy Buzsáki, Rodolfo Llinas, Wolf Singer, Alain Berthoz, and Yves Christen. *Temporal coding in the brain*. Springer Science & Business Media, 2012.

[89] Alexandre Pouget, Peter Dayan, and Richard Zemel. Information processing with population codes. *Nature Reviews Neuroscience*, 1(2):125–132, 2000.

[90] Martin Boerlin and Sophie Denève. Spike-based population coding and working memory. *PLoS computational biology*, 7(2), 2011.

[91] Bruno A Olshausen and David J Field. Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4):481–487, 2004.

[92] Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[93] Alvin I Goldman et al. Theory of mind. *The Oxford handbook of philosophy of cognitive science*, 1, 2012.

[94] Stanislas Dehaene. Le cerveau statisticien: la révolution bayésienne en science cognitives. In *2011-2012 College de France*. College de France, 2012.

[95] Stanislas Dehaene. Le bébé statisticien : les théories bayésiennes de l'apprentissage. In *2012-2013 College de France*. College de France, 2013.

[96] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. Spike-timing-dependent plasticity: a comprehensive overview. *Frontiers in synaptic neuroscience*, 4:2, 2012.

[97] Damien Querlioz, Olivier Bichler, Philippe Dollfus, and Christian Gamrat. Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Transactions on Nanotechnology*, 12(3):288–295, 2013.

[98] Harel Z Shouval, Samuel S-H Wang, and Gayle M Wittenberg. Spike timing dependent plasticity: a consequence of more fundamental learning rules. *Frontiers in computational neuroscience*, 4:19, 2010.

[99] Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu. Stdp as presynaptic activity times rate of change of postsynaptic activity. *arXiv preprint arXiv:1509.05936*, 2015.

[100] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Continual weight updates and convolutional architectures for equilibrium propagation. *arXiv preprint arXiv:2005.04169*, 2020.

[101] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[102] Geoffrey Hinton. How to do backpropagation in a brain. In *Invited talk at the NIPS'2007 deep learning workshop*, volume 656, 2007.

[103] John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

[104] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines*. New York, NY; John Wiley and Sons Inc., 1988.

[105] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[106] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071, 2008.

[107] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[108] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, pages 1–12, 2020.

[109] James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in cognitive sciences*, 2019.

[110] Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. *arXiv preprint arXiv:2006.03824*, 2020.

[111] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[112] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley & Sons, 1994.

[113] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012.

[114] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[115] NVIDIA. Powering change with ai and deep learning, https://www.nvidia.com/en-us/deep-learning-ai. In *NVIDIA-website*, 2020.

[116] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. ISCA*, pages 1–12. IEEE, 2017.

[117] Google. Edge tpu, https://cloud.google.com/edge-tpu. In *Google*. Google, 2020.

[118] Movidius. Movidius stick, https://www.intel.com/content/www/us/en/support/articles/000033354/boards-and-kits/neural-compute-sticks.html. In *Movidius*. Intel, 2020.

[119] Arthur Stoutchinin, Francesco Conti, and Luca Benini. Optimally scheduling cnn convolutions for efficient memory access. *arXiv preprint arXiv:1902.01492*, 2019.

[120] Avishek Biswas and Anantha P Chandrakasan. Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 488–490. IEEE, 2018.

[121] Kota Ando, Kodai Ueyoshi, Kentaro Orimo, Haruyoshi Yonekawa, Shimpei Sato, Hiroki Nakahara, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, Tadahiro Kuroda, et al. Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w. *IEEE Journal of Solid-State Circuits*, 53(4):983–994, 2017.

[122] Fabien Alibart, Elham Zamanidoost, and Dmitri B Strukov. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nature communications*, 4(1):1–7, 2013.

[123] Geoffrey W Burr et al. Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses). *IEEE Trans. Electron. Dev.*, 62(11):3498–3507, 2015.

[124] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M Shelby, Irem Boybat, Carmelo di Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan CP Farinha, et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60–67, 2018.

[125] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[126] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552. IEEE, 2017.

[127] Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2012.

[128] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[129] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[130] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[131] Adrien F Vincent et al. Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems. *IEEE T. Biomed. Circ. S.*, 9(2):166–174, 2015.

[132] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature Photonics*, 11(7):441, 2017.

[133] Johnny Moughames, Xavier Porte, Michael Thiel, Gwenn Ulliac, Laurent Larger, Maxime Jacquot, Muamer Kadic, and Daniel Brunner. Three-dimensional waveguide interconnects for scalable integration of photonic neural networks. *Optica*, 7(6):640–646, 2020.

[134] Noboru Yamada, Eiji Ohno, Nobuo Akahira, Ken'ichi Nishiuchi, Ken'ichi Nagata, and Masatoshi Takao. High speed overwritable phase change optical disk material. *Japanese Journal of Applied Physics*, 26(S4):61, 1987.

[135] J Feldmann, N Youngblood, C David Wright, H Bhaskaran, and WHP Pernice. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature*, 569(7755):208–214, 2019.

[136] Wim Bogaerts, Pieter Dumon, Dries Van Thourhout, and Roel Baets. Low-loss, low-cross-talk crossings for silicon-on-insulator nanophotonic waveguides. *Optics letters*, 32(19):2801–2803, 2007.

[137] Laurent Larger, Miguel C Soriano, Daniel Brunner, Lennert Appeltant, Jose M Gutiérrez, Luis Pesquera, Claudio R Mirasso, and Ingo Fischer. Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Optics express*, 20(3):3241–3249, 2012.

[138] Kristof Vandoorne, Pauline Mechet, Thomas Van Vaerenbergh, Martin Fiers, Geert Morthier, David Verstraeten, Benjamin Schrauwen, Joni Dambre, and Peter Bienstman. Experimental demonstration of reservoir computing on a silicon photonics chip. *Nature communications*, 5(1):1–6, 2014.

[139] Yvan Paquot, Francois Duport, Antoneo Smerieri, Joni Dambre, Benjamin Schrauwen, Marc Haelterman, and Serge Massar. Optoelectronic reservoir computing. *Scientific reports*, 2:287, 2012.

[140] François Duport, Bendix Schneider, Anteo Smerieri, Marc Haelterman, and Serge Massar. All-optical reservoir computing. *Optics express*, 20(20):22783–22795, 2012.

[141] Chao Du, Fuxi Cai, Mohammed A Zidan, Wen Ma, Seung Hwan Lee, and Wei D Lu. Reservoir computing using dynamic memristors for temporal information processing. *Nature communications*, 8(1):2204, 2017.

[142] Danijela Marković, Nathan Leroux, Mathieu Riou, Flavio Abreu Araujo, Jacob Torrejon, Damien Querlioz, Akio Fukushima, Shinji Yuasa, Juan Trastoy, Paolo Bortolotti, et al. Reservoir computing with the frequency, phase, and amplitude of spin-torque nano-oscillators. *Applied Physics Letters*, 114(1):012409, 2019.

[143] Damir Vodenicarevic, Nicolas Locatelli, Julie Grollier, and Damien Querlioz. Synchronization detection in networks of coupled oscillators for pattern recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2015–2022. IEEE, 2016.

[144] Damir Vodenicarevic. *Rhythms and oscillations: a vision for nanoelectronics*. PhD thesis, Université Paris-Saclay, 2017.

[145] Miguel Romera, Philippe Talatchian, Sumito Tsunegi, Flavio Abreu Araujo, Vincent Cros, Paolo Bortolotti, Juan Trastoy, Kay Yakushiji, Akio Fukushima, Hitoshi Kubota, et al. Vowel recognition with four coupled spin-torque nano-oscillators. *Nature*, 563(7730):230–234, 2018.

[146] P Talatchian, M Romera, Sumito Tsunegi, F Abreu Araujo, Vincent Cros, Paolo Bortolotti, J Trastoy, K Yakushiji, Akio Fukushima, H Kubota, et al. Microwave neural processing and broadcasting with spintronic nano-oscillators. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 27–4. IEEE, 2018.

[147] Yann LeCun : "L'intelligence artificielle a moins de sens commun qu'un rat". Library Catalog: www.sciencesetavenir.fr.

[148] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[149] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[150] Daphne Koller. Probabilistic graphical models, coursera.

[151] Pierre Bessiere, Emmanuel Mazer, Juan Manuel Ahuactzin, and Kamel Mekhnacha. *Bayesian programming*. CRC press, 2013.

[152] Fei Xu and Vashti Garcia. Intuitive statistics by 8-month-old infants. *Proceedings of the National Academy of Sciences*, 105(13):5012–5015, 2008.

[153] Alison Gopnik. Scientific thinking in young children: Theoretical advances, empirical research, and policy implications. *Science*, 337(6102):1623–1627, 2012.

[154] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

[155] Steven J Nowlan. Maximum likelihood competitive learning. In *Advances in neural information processing systems*, pages 574–582, 1990.

[156] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[157] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

[158] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[159] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[160] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

[161] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[162] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[163] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[164] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[165] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[166] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[167] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International conference on learning representations*, 2018.

[168] Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems*, pages 15512–15522, 2019.

[169] David H Hubel. *Eye, brain, and vision*. Scientific American Library/Scientific American Books, 1995.

[170] Vernon B Mountcastle. The columnar organization of the neocortex. *Brain: a journal of neurology*, 120(4):701–722, 1997.

[171] Patrick Pirim. Generic bio-inspired chip model-based on spatio-temporal histogram computation: Application to car driving by gaze-like control. In *Conference on Biomimetic and Biohybrid Systems*, pages 228–239. Springer, 2013.

[172] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[173] Yann LeCun. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[174] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[175] Randy Yates. Fixed-point arithmetic: An introduction. *Digital Signal Labs*, 81(83):198, 2009.

[176] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[177] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[178] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[179] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[180] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. *Coursera, video lectures*, 264(1), 2012.

[181] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[182] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[183] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[184] Axel Laborieux, Marc Bocquet, Tifenn Hirtzlin, J-O Klein, L Herrera Diez, Etienne Nowak, Elisa Vianello, J-M Portal, and Damien Querlioz. Low power in-memory implementation of ternary neural networks with resistive ram-based synapse. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 136–140. IEEE, 2020.

[185] Vishnu Raj, Nancy Nayak, and Sheetal Kalyani. Understanding learning dynamics of binary neural networks via information bottleneck. *arXiv preprint arXiv:2006.07522*, 2020.

[186] Viacheslav Osaulenko. Binary autoencoder with random binary weights. *arXiv preprint arXiv:2004.14717*, 2020.

[187] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[188] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

[189] Mohamed A Shahin, Mark B Jaksa, and Holger R Maier. Artificial neural network applications in geotechnical engineering. *Australian geomechanics*, 36(1):49–62, 2001.

[190] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[191] Shakir Mohamed. Bayesian Reasoning and Deep Learning.

[192] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[193] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[194] Long Chen, Xiyuan Tang, Arindam Sanyal, Yeonam Yoon, Jie Cong, and Nan Sun. A 0.7-v 0.6uw-100-ks/s low-power sar adc with statistical estimation-based noise reduction. *IEEE Journal of Solid-State Circuits*, 52(5):1388–1398, 2017.

[195] Syed Asmat Ali Shah, AN Ragheb, and HyungWon Kim. Low-power voltage converter using energy recycling capacitor array. *Journal of information and communication convergence engineering*, 15(1):62–71, 2017.

[196] Michael P Wellman and Max Henrion. Explaining'explaining away'. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):287–292, 1993.

[197] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3,22, pages 41–46, 2001.

[198] Joseph S Friedman, Laurie E Calvet, Pierre Bessière, Jacques Droulez, and Damien Querlioz. Bayesian inference with muller c-elements. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(6):895–904, 2016.

[199] Leo A Goodman. Exploratory latent structure analysis using both identifiable and unidentifiable models. *Biometrika*, 61(2):215–231, 1974.

[200] Jay K Hackett and Mubarak Shah. Multi-sensor fusion: a perspective. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 1324–1330. IEEE, 1990.

[201] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[202] Nhut-Minh Ho and Weng-Fai Wong. Exploiting half precision arithmetic in nvidia gpus. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2017.

[203] Michal Gallus and Alberto Nannarelli. Handwritten digit classification using 8-bit floating point based convolutional neural networks, 2018.

[204] Swagath Venkataramani, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Approximate computing and the quest for computing efficiency. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[205] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. Approximate computing: A survey. *IEEE Design & Test*, 33(1):8–22, 2015.

[206] Benjamin Barrois, Olivier Sentieys, and Daniel Menard. The hidden cost of functional approximation against careful data sizing—a case study. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 181–186. IEEE, 2017.

[207] Jean-Michel Muller, Nicolas Brisebarre, Florent De Dinechin, Claude-Pierre Jeannerod, Vincent Lefevre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, Serge Torres, et al. *Handbook of floating-point arithmetic*, volume 1. Springer, 2018.

[208] Marvin Faix, Raphael Laurent, Pierre Bessière, Emmanuel Mazer, and Jacques Droulez. Design of stochastic machines dedicated to approximate bayesian inferences. *IEEE Transactions on Emerging Topics in Computing*, 7(1):60–66, 2016.

[209] Marvin Faix. *Conception de machines probabilistes dédiées aux inférences bayésiennes*. PhD thesis, UGA, 2016.

[210] Raphael Frisch, Raphaël Laurent, Marvin Faix, Laurent Girin, Laurent Fesquet, Augustin Lux, Jacques Droulez, Pierre Bessière, and Emmanuel Mazer. A bayesian stochastic machine for sound source localization. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2017.

[211] Raphael Frisch. *Stochastic machines dedicated to Bayesian inference for source localization and separation*. PhD thesis, Université Grenoble Alpes, 2019.

[212] Stephen L Wong, Paul Veldman, and J Eugene. Level shifter, March 14 2000. US Patent 6,037,720.

[213] Damir Vodenicarevic, Nicolas Locatelli, Alice Mizrahi, Joseph S Friedman, Adrien F Vincent, Miguel Romera, Akio Fukushima, Kay Yakushiji, Hitoshi Kubota, Shinji Yuasa, et al. Low-energy truly random number generation with superparamagnetic tunnel junctions for unconventional computing. *Phys. Rev. Appl.*, 8(5):054045, 2017.

[214] An Chen. Utilizing the variability of resistive random access memory to implement reconfigurable physical unclonable functions. *IEEE Electron Device Letters*, 36(2):138–140, 2014.

[215] Simone Balatti, Stefano Ambrogio, Zhongqiang Wang, and Daniele Ielmini. True random number generation by variability of resistive switching in oxide-based devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):214–221, 2015.

[216] Prabhat Kumar Gupta and Ramdas Kumaresan. Binary multiplication with pn sequences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(4):603–606, 1988.

[217] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):1–19, 2013.

[218] Laurie E Calvet, Joseph S Friedman, Damien Querlioz, Pierre Bessière, and Jacques Droulez. Sleep stage classification with stochastic bayesian inference. In *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 117–122. IEEE, 2016.

[219] Xiaotao Jia, Jianlei Yang, Pengcheng Dai, Runze Liu, Yiran Chen, and Weisheng Zhao. Spinbis: Spintronics-based bayesian inference system with stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(4):789–802, 2019.

[220] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. Metal–oxide rram. *Proceedings of the IEEE*, 100(6):1951–1970, 2012.

[221] Rainer Waser, Regina Dittmann, Georgi Staikov, and Kristof Szot. Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges. *Advanced materials*, 21(25-26):2632–2663, 2009.

[222] Boubacar Traoré, Philippe Blaise, Elisa Vianello, Luca Perniola, Barbara De Salvo, and Yoshio Nishi. Hfo 2-based rram: Electrode effects, ti/hfo 2 interface, charge injection, and oxygen (o) defects diffusion through experiment and ab initio calculations. *IEEE Transactions on Electron Devices*, 63(1):360–368, 2015.

[223] S Brivio, J Frascaroli, and S Spiga. Role of metal-oxide interfaces in the multiple resistance switching regimes of pt/hfo2/tin devices. *Applied Physics Letters*, 107(2):023504, 2015.

[224] Federico Nardi, Simone Balatti, Stefano Larentis, David C Gilmer, and Daniele Ielmini. Complementary switching in oxide-based bipolar resistive-switching random memory. *IEEE transactions on electron devices*, 60(1):70–77, 2012.

[225] Tifenn Hirtzlin, Marc Bocquet, Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Frontiers in Neuroscience*, 13, 2019.

[226] Tifenn Hirtzlin, Bogdan Penkovsky, Marc Bocquet, Jacques-Olivier Klein, Jean-Michel Portal, and Damien Querlioz. Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access*, 7:76394–76403, 2019.

[227] Ardavan Pedram, Stephen Richardson, Mark Horowitz, Sameh Galal, and Shahar Kvatinsky. Dark memory and accelerator-rich system optimization in the dark silicon era. *IEEE Design & Test*, 34(2):39–50, 2017.

[228] Shimeng Yu. Neuro-inspired computing with emerging nonvolatile memorys. *Proc. IEEE*, 106(2):260–285, 2018.

[229] Editorial. Big data needs a hardware revolution. *Nature*, 554(7691):145, February 2018.

[230] Damien Querlioz, Olivier Bichler, Adrien Francis Vincent, and Christian Gamrat. Bioinspired programming of memory devices for implementing an inference engine. *Proc. IEEE*, 103(8):1398–1416, 2015.

[231] Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proc. IEEE*, 103(8):1379–1397, 2015.

[232] Mirko Prezioso, Farnood Merrikh-Bayat, BD Hoskins, Gina C Adam, Konstantin K Likharev, and Dmitri B Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61, 2015.

[233] Alexander Serb, Johannes Bill, Ali Khiat, Radu Berdan, Robert Legenstein, and Themis Prodromakis. Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses. *Nature communications*, 7:12611, 2016.

[234] Sylvain Saïghi, Christian G Mayr, Teresa Serrano-Gotarredona, Heidemarie Schmidt, Gwendal Lecerf, Jean Tomas, Julie Grollier, Sören Boyn, Adrien F Vincent, Damien Querlioz, et al. Plasticity in memristive devices for spiking neural networks. *Frontiers in neuroscience*, 9:51, 2015.

[235] Erika Covi, Stefano Brivio, Alexander Serb, Themis Prodromakis, Marco Fanciulli, and Sabina Spiga. Analog memristive synapse in spiking networks implementing unsupervised learning. *Frontiers in neuroscience*, 10:482, 2016.

[236] Zhongqiang Wang, Stefano Ambrogio, Simone Balatti, and Daniele Ielmini. A 2-transistor/1-resistor artificial synapse capable of communication and stochastic learning in neuromorphic systems. *Frontiers in neuroscience*, 8:438, 2015.

[237] A Aldo Faisal, Luc PJ Selen, and Daniel M Wolpert. Noise in the nervous system. *Nature reviews neuroscience*, 9(4):292, 2008.

[238] Konstantin Klemm and Stefan Bornholdt. Topology of biological networks and reliability of information processing. *Proceedings of the National Academy of Sciences*, 102(51):18414–18419, 2005.

[239] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature communications*, 9(1):2385, 2018.

[240] Zhongrui Wang, Saumil Joshi, Sergey Savel'ev, Wenhao Song, Rivu Midya, Yunning Li, Mingyi Rao, Peng Yan, Shiva Asapu, Ye Zhuo, et al. Fully memristive neural networks for pattern classification with unsupervised learning. *Nature Electronics*, 1(2):137, 2018.

[241] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[242] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 345–353, 2017.

[243] Shimeng Yu, Zhiwei Li, Pai-Yu Chen, Huaqiang Wu, Bin Gao, Deli Wang, Wei Wu, and He Qian. Binary neural network with 16 mb rram macro chip for classification and online training. In *IEDM Tech. Dig.*, pages 16–2. IEEE, 2016.

[244] Alessandro Grossi, E Nowak, Cristian Zambelli, C Pellissier, S Bernasconi, G Cibrario, K El Hajjam, R Crochemore, JF Nodin, Piero Olivo, et al. Fundamental variability limits of filament-based rram. In *IEDM Tech. Dig.*, pages 4–7. IEEE, 2016.

[245] Daniele Ielmini and H-S Philip Wong. In-memory computing with resistive switching devices. *Nature Electronics*, 1(6):333, 2018.

[246] Denys Riwan Bunsothy Ly, Alessandro Grossi, Claire Fenouillet-Beranger, Etienne Nowak, Damien Querlioz, and Elisa Vianello. Role of synaptic variability in resistive memory-based spiking neural networks with unsupervised learning. *J. Phys. D: Applied Physics*, 2018.

[247] Stefano Gregori, Alessandro Cabrini, Osama Khouri, and Guido Torelli. On-chip error correcting techniques for new-generation flash memories. *Proc. IEEE*, 91(4):602–616, 2003.

[248] W. H. Chen, K. X. Li, W. Y. Lin, K. H. Hsu, P. Y. Li, C. H. Yang, C. X. Xue, E. Y. Yang, Y. K. Chen, Y. S. Chang, T. H. Hsu, Y. C. King, C. J. Lin, R. S. Liu, C. C. Hsieh, K. T. Tang, and M. F. Chang. A 65nm 1mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *Proc. ISSCC*, pages 494–496, February 2018.

[249] W. H. Chen, W. J. Lin, L. Y. Lai, S. Li, C. H. Hsu, H. T. Lin, H. Y. Lee, J. W. Su, Y. Xie, S. S. Sheu, and M. F. Chang. A 16mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme. In *IEDM Tech. Dig.*, pages 28.2.1–28.2.4, December 2017.

[250] Farnood Merrikh-Bayat, Xinjie Guo, Michael Klachko, Mirko Prezioso, Konstantin K Likharev, and Dmitri B Strukov. High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays. *IEEE transactions on neural networks and learning systems*, 29(10):4782–4790, 2017.

[251] Qing Dong, Yejoong Kim, Inhee Lee, Myungjoon Choi, Ziyun Li, Jingcheng Wang, Kaiyuan Yang, Yen-Po Chen, Junjie Dong, Minchang Cho, et al. 11.2 a 1mb embedded nor flash memory with 39$\mu$w program power for mm-scale high-temperature sensor nodes. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 198–199. IEEE, 2017.

[252] An Chen. A review of emerging non-volatile memory (nvm) technologies and applications. *Solid-State Electronics*, 125:25–38, 2016.

[253] Alessandro Grossi, Elisa Vianello, Cristian Zambelli, Pablo Royer, Jean-Philippe Noel, Bastien Giraud, Luca Perniola, Piero Olivo, and Etienne Nowak. Experimental investigation of 4-kb rram arrays programming conditions suitable for tcam. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12):2599–2607, 2018.

[254] Wei-Ting Hsieh, Yue-Der Chih, Jonathan Chang, Chrong-Jung Lin, and Ya-Chin King. Differential Contact RRAM Pair for Advanced CMOS Logic NVM applications. *Proc. SSDM*, page 171, 2017.

[255] Yi-Hong Shih, Meng-Yin Hsu, Chrong Jung Lin, and Ya-Chin King. Twin-bit Via RRAM in 16nm FinFET Logic Technologies. *Proc. SSDM*, page 137, 2017.

[256] Weisheng Zhao, Mathieu Moreau, Erya Deng, Yue Zhang, Jean-Michel Portal, Jacques-Olivier Klein, Marc Bocquet, Hassen Aziza, Damien Deleruyelle, Christophe Muller, et al. Synchronous non-volatile logic gate design based on resistive switching memories. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(2):443–454, 2014.

[257] Daniel Bankman, Lita Yang, Bert Moons, Marian Verhelst, and Boris Murmann. An always-on 3.8muj/86 % cifar-10 mixed-signal binary cnn processor with all memory on chip in 28-nm cmos. *IEEE Journal of Solid-State Circuits*, 54(1):158–172, 2018.

[258] Xiaoyu Sun, Xiaochen Peng, Pai-Yu Chen, Rui Liu, Jae-sun Seo, and Shimeng Yu. Fully parallel rram synaptic array for implementing binary neural network with (+ 1,- 1) weights and (+ 1, 0) neurons. In *Proc. ASP-DAC*, pages 574–579. IEEE Press, 2018.

[259] Xiaoyu Sun, Shihui Yin, Xiaochen Peng, Rui Liu, Jae-sun Seo, and Shimeng Yu. Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks. *algorithms*, 2:3, 2018.

[260] Tianqi Tang, Lixue Xia, Boxun Li, Yu Wang, and Huazhong Yang. Binary convolutional neural network on rram. In *Proc. ASP-DAC*, pages 782–787. IEEE, 2017.

[261] Masanori Natsui, Tomoki Chiba, and Takahiro Hanyu. Design of mtj-based nonvolatile logic gates for quantized neural networks. *Microelectronics journal*, 82:13–21, 2018.

[262] Edouard Giacomin, Tzofnat Greenberg-Toledo, Shahar Kvatinsky, and Pierre-Emmanuel Gaillardon. A robust digital rram-based convolutional block for low-power image processing and learning applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(2):643–654, 2019.

[263] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[264] Tifenn Hirtzlin, Marc Bocquet, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Outstanding bit error tolerance of resistive ram-based binarized neural networks. *arXiv preprint arXiv:1904.03652*, 2019.

[265] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, page 23. IEEE Press, 2016.

[266] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proc. Int. Symp. FPGA*, pages 15–24. ACM, 2017.

[267] Brian R Gaines. Stochastic computing systems. In *Advances in information systems science*, pages 37–172. Springer, 1969.

[268] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[269] Ki Chul Chun, Hui Zhao, Jonathan D Harms, Tae-Hyoung Kim, Jian-Ping Wang, and Chris H Kim. A scaling roadmap and performance evaluation of in-plane and perpendicular mtj based stt-mrams for high-density cache memory. *IEEE Journal of Solid-State Circuits*, 48(2):598–610, 2012.

[270] Bogdan Penkovsky, Marc Bocquet, Tifenn Hirtzlin, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. In-memory resistive ram implementation of binarized neural networks for medical applications. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 690–695. IEEE, 2020.

[271] Marc Bocquet, Tifenn Hirztlin, J-O Klein, Etienne Nowak, Elisa Vianello, J-M Portal, and Damien Querlioz. In-memory and error-immune differential rram implementation of binarized deep neural networks. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 20–6. IEEE, 2018.

[272] Hauke Dose, Jakob S Møller, Helle K Iversen, and Sadasivan Puthusserypady. An end-to-end deep learning approach to mi-eeg signal classification for bcis. *Expert Systems with Applications*, 114:532–542, 2018.

[273] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[274] Hai Phan, Yihui He, Marios Savvides, Zhiqiang Shen, et al. Mobinet: A mobile binary network for image classification. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 3453–3462, 2020.

[275] A Lingamneni and K Palem. What to do about the end of moore's law, probably. In *Proc. Design Automation Conference (DAC)*, 2012.

[276] Thomas Dalgaty, Niccolo Castellani, Damien Querlioz, and Elisa Vianello. In-situ learning harnessing intrinsic resistive memory variability through markov chain monte carlo sampling. *arXiv preprint arXiv:2001.11426*, 2020.

[277] Michael N Shadlen and William T Newsome. Noise, neural codes and cortical organization. *Current opinion in neurobiology*, 4(4):569–579, 1994.

[278] Richard B Stein, E Roderich Gossen, and Kelvin E Jones. Neuronal variability: noise or part of the signal? *Nature Reviews Neuroscience*, 6(5):389–397, 2005.

[279] Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

[280] Rafatul Faria, Kerem Y Camsari, and Supriyo Datta. Implementing bayesian networks with embedded stochastic mram. *AIP Advances*, 8(4):045101, 2018.

[281] Kerem Yunus Camsari, Rafatul Faria, Brian M Sutton, and Supriyo Datta. Stochastic p-bits for invertible logic. *Physical Review X*, 7(3):031014, 2017.

[282] Theodore Papamarkou, Jacob Hinkle, M Todd Young, and David Womble. Challenges in bayesian inference via markov chain monte carlo for neural networks. *arXiv preprint arXiv:1910.06539*, 2019.

[283] Hironori Kumano and Takanori Uka. The spatial profile of macaque mt neurons is consistent with gaussian sampling of logarithmically coordinated visual representation. *Journal of neurophysiology*, 104(1):61–75, 2010.

[284] Alice Mizrahi, Tifenn Hirtzlin, Akio Fukushima, Hitoshi Kubota, Shinji Yuasa, Julie Grollier, and Damien Querlioz. Neural-like computing with populations of superparamagnetic basis functions. *Nature communications*, 9(1):1–11, 2018.

[285] Horace B Barlow. Summation and inhibition in the frog's retina. *The Journal of physiology*, 119(1):69, 1953.

[286] Choongkil Lee, William H Rohrer, and David L Sparks. Population coding of saccadic eye movements by neurons in the superior colliculus. *Nature*, 332(6162):357–360, 1988.

[287] Apostolos P Georgopoulos, John F Kalaska, Roberto Caminiti, and Joe T Massey. On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *Journal of Neuroscience*, 2(11):1527–1537, 1982.

[288] Anitha Pasupathy and Charles E Connor. Population coding of shape in area v4. *Nature neuroscience*, 5(12):1332–1338, 2002.

[289] Chetan Singh Thakur, Tara Julia Hamilton, Runchun Wang, Jonathan Tapson, and André van Schaik. A neuromorphic hardware framework based on population coding. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[290] Chetan Singh Thakur, Runchun Wang, Tara Julia Hamilton, Jonathan Tapson, and André van Schaik. A low power trainable neuromorphic integrated circuit that is tolerant to device mismatch. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(2):211–221, 2016.

[291] Tomas Tuma, Angeliki Pantazi, Manuel Le Gallo, Abu Sebastian, and Evangelos Eleftheriou. Stochastic phase-change neurons. *Nature nanotechnology*, 11(8):693, 2016.

[292] Hyungkwang Lim, Vladimir Kornijcuk, Jun Yeong Seok, Seong Keun Kim, Inho Kim, Cheol Seong Hwang, and Doo Seok Jeong. Reliability of neuronal information conveyed by unreliable neuristor-based leaky integrate-and-fire neurons: a model study. *Scientific reports*, 5:9776, 2015.

[293] Peter Dayan and LF Abbott. Computational and mathematical modeling of neural systems. *Theoretical neuroscience*, 2001.

[294] H Sebastian Seung and Haim Sompolinsky. Simple models for reading neuronal population codes. *Proceedings of the National Academy of Sciences*, 90(22):10749–10753, 1993.

[295] Wei Ji Ma, Jeffrey M Beck, Peter E Latham, and Alexandre Pouget. Bayesian inference with probabilistic population codes. *Nature neuroscience*, 9(11):1432–1438, 2006.

[296] Dmytro Apalkov, Bernard Dieny, and JM Slaughter. Magnetoresistive random access memory. *Proceedings of the IEEE*, 104(10):1796–1830, 2016.

[297] William Rippard, Ranko Heindl, Matthew Pufall, Stephen Russek, and Anthony Kos. Thermal relaxation rates of magnetic nanoparticles in the presence of magnetic fields and spin-transfer effects. *Physical Review B*, 84(6):064439, 2011.

[298] Alice Mizrahi, Nicolas Locatelli, Romain Lebrun, Vincent Cros, Akio Fukushima, Hitoshi Kubota, Shinji Yuasa, Damien Querlioz, and Julie Grollier. Controlling the phase locking of stochastic magnetic bits for ultra-low power computation. *Scientific reports*, 6(1):1–7, 2016.

[299] Z Li and Shufeng Zhang. Thermally assisted magnetization reversal in the presence of a spin-transfer torque. *Physical Review B*, 69(13):134416, 2004.

[300] Punyashloka Debashis, Rafatul Faria, Kerem Y Camsari, Joerg Appenzeller, Supriyo Datta, and Zhihong Chen. Experimental demonstration of nanomagnet networks as hardware for ising computing. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 34–3. IEEE, 2016.

[301] Brian Sutton, Kerem Yunus Camsari, Behtash Behin-Aein, and Supriyo Datta. Intrinsic optimization using stochastic nanomagnets. *Scientific reports*, 7(1):1–9, 2017.

[302] Nicolas Locatelli, Alice Mizrahi, A Accioly, Rie Matsumoto, Akio Fukushima, Hitoshi Kubota, Shinji Yuasa, Vincent Cros, Luis Gustavo Pereira, Damien Querlioz, et al. Noise-enhanced synchronization of stochastic magnetic oscillators. *Physical Review Applied*, 2(3):034009, 2014.

[303] John C Slonczewski. Currents, torques, and polarization factors in magnetic tunnel junctions. *Physical Review B*, 71(2):024411, 2005.

[304] Alexandre Pouget, Jeffrey M Beck, Wei Ji Ma, and Peter E Latham. Probabilistic brains: knowns and unknowns. *Nature neuroscience*, 16(9):1170–1178, 2013.

[305] Emilio Salinas and Larry F Abbott. Transfer of coded information from sensory to motor networks. *Journal of Neuroscience*, 15(10):6461–6474, 1995.

[306] Alexandre Pouget and Terrence J Sejnowski. Spatial transformations in the parietal cortex using basis functions. *Journal of cognitive neuroscience*, 9(2):222–237, 1997.

[307] Jeffrey M Beck, Wei Ji Ma, Roozbeh Kiani, Tim Hanks, Anne K Churchland, Jamie Roitman, Michael N Shadlen, Peter E Latham, and Alexandre Pouget. Probabilistic population codes for bayesian decision making. *Neuron*, 60(6):1142–1152, 2008.

[308] Rubén Moreno-Bote, David C Knill, and Alexandre Pouget. Bayesian sampling in visual perception. *Proceedings of the National Academy of Sciences*, 108(30):12491–12496, 2011.

[309] Luqiao Liu, OJ Lee, TJ Gudmundsen, DC Ralph, and RA Buhrman. Current-induced switching of perpendicularly magnetized magnetic layers using spin torque from the spin hall effect. *Physical review letters*, 109(9):096602, 2012.

[310] William A Borders, Hisanao Akima, Shunsuke Fukami, Satoshi Moriya, Shouta Kurihara, Yoshihiko Horio, Shigeo Sato, and Hideo Ohno. Analogue spin–orbit torque device for artificial-neural-network-based associative memory operation. *Applied physics express*, 10(1):013007, 2016.

[311] Ioan Mihai Miron, Gilles Gaudin, Stéphane Auffret, Bernard Rodmacq, Alain Schuhl, Stefania Pizzini, Jan Vogel, and Pietro Gambardella. Current-driven spin torque induced by the rashba effect in a ferromagnetic metal layer. *Nature materials*, 9(3):230–234, 2010.

[312] Chetan Singh Thakur, Runchun Wang, Saeed Afshar, Gregory Cohen, Tara Julia Hamilton, Jonathan Tapson, and Andre van Schaik. An online learning algorithm for neuromorphic hardware implementation. *arXiv preprint arXiv:1505.02495*, 2015.

[313] George L Gerstein and Benoit Mandelbrot. Random walk models for the spike activity of a single neuron. *Biophysical journal*, 4(1):41–68, 1964.

[314] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[315] Chetan Singh Thakur, Runchun Wang, Tara Julia Hamilton, Ralph Etienne-Cummings, Jonathan Tapson, and André van Schaik. An analogue neuromorphic co-processor that utilizes device mismatch for learning applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(4):1174–1184, 2017.

[316] Janusz J Nowak, Ray P Robertazzi, Jonathan Z Sun, Guohan Hu, Jeong-Heon Park, JungHyuk Lee, Anthony J Annunziata, Gen P Lauer, Raman Kothandaraman, Eugene J O'Sullivan, et al. Dependence of voltage and size on write error rates in spin-transfer torque magnetic random-access memory. *IEEE Magnetics Letters*, 7:1–4, 2016.

[317] H Sato, ECI Enobio, M Yamanouchi, S Ikeda, S Fukami, S Kanai, F Matsukura, and H Ohno. Properties of magnetic tunnel junctions with a mgo/cofeb/ta/cofeb/mgo recording structure down to junction diameter of 11 nm. *Applied Physics Letters*, 105(6):062403, 2014.

[318] Yahya Lakys, Wei Sheng Zhao, Thibaut Devolder, Yue Zhang, Jacques-Olivier Klein, Dafiné Ravelosona, and Claude Chappert. Self-enabled "error-free" switching circuit for spin transfer torque mram and logic. *IEEE Transactions on Magnetics*, 48(9):2403–2406, 2012.

[319] Paolo Livi and Giacomo Indiveri. A current-mode conductance-based silicon neuron for address-event neuromorphic systems. In *2009 IEEE international symposium on circuits and systems*, pages 2898–2901. IEEE, 2009.

[320] Chia-Hung Chen, Yi Zhang, Tao He, Patrick Y Chiang, and Gabor C Temes. A 11$\mu$w 250 hz bw two-step incremental adc with 100 db dr and 91 db sndr for integrated sensor interfaces. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–4. IEEE, 2014.

[321] Bruno B Averbeck, Peter E Latham, and Alexandre Pouget. Neural correlations, population coding and computation. *Nature reviews neuroscience*, 7(5):358–366, 2006.

[322] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[323] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141, 2015.

[324] M. Bocquet, T. Hirztlin, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz. In-memory and error-immune differential rram implementation of binarized deep neural networks. In *IEDM Tech. Dig.*, page 20.6.1. IEEE, 2018.

[325] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[326] Francesco Conti, Pasquale Davide Schiavone, and Luca Benini. Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2940–2951, 2018.

[327] Axel Laborieux, Maxence Ernoult, Tifenn Hirtzlin, and Damien Querlioz. Synaptic metaplasticity in binarized neural networks. *arXiv preprint arXiv:2003.03533*, 2020.

[328] Stefano Fusi, Patrick J. Drew, and L. F. Abbott. Cascade models of synaptically stored memories. *Neuron*, 2005.

[329] Marcus K Benna and Stefano Fusi. Computational principles of synaptic memory consolidation. *Nature neuroscience*, 19(12):1697, 2016.

[330] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2013.

[331] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[332] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[333] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[334] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[335] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[336] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[337] Quantan Wu, Hong Wang, Qing Luo, Writam Banerjee, Jingchen Cao, Xumeng Zhang, Facai Wu, Qi Liu, Ling Li, and Ming Liu. Full imitation of synaptic metaplasticity based on memristor devices. *Nanoscale*, 10(13):5875–5881, 2018.

[338] Xiaojian Zhu, Chao Du, YeonJoo Jeong, and Wei D Lu. Emulation of synaptic metaplasticity in memristors. *Nanoscale*, 9(1):45–51, 2017.

[339] Tae-Ho Lee, Hyun-Gyu Hwang, Jong-Un Woo, Dae-Hyeon Kim, Tae-Wook Kim, and Sahn Nahm. Synaptic plasticity and metaplasticity of biological synapse realized in a knbo3 memristor for application to artificial synapse. *ACS applied materials & interfaces*, 10(30):25673–25682, 2018.

[340] Bo Liu, Zhiwei Liu, In-Shiang Chiu, MengFu Di, YongRen Wu, Jer-Chyi Wang, Tuo-Hung Hou, and Chao-Sung Lai. Programmable synaptic metaplasticity and below femtojoule spiking energy realized in graphene-based neuromorphic memristor. *ACS applied materials & interfaces*, 10(24):20237–20243, 2018.

[341] Zheng-Hua Tan, Rui Yang, Kazuya Terabe, Xue-Bing Yin, Xiao-Dong Zhang, and Xin Guo. Synaptic metaplasticity realized in oxide memristive devices. *Advanced Materials*, 28(2):377–384, 2016.

[342] T Hirtzlin, Marc Bocquet, M Ernoult, J-O Klein, E Nowak, E Vianello, J-M Portal, and D Querlioz. Hybrid analog-digital learning with differential rram synapses. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 22–6. IEEE, 2019.

[343] Fuxi Cai, Justin M Correll, Seung Hwan Lee, Yong Lim, Vishishtha Bothra, Zhengya Zhang, Michael P Flynn, and Wei D Lu. A fully integrated reprogrammable memristor–cmos system for efficient multiply–accumulate operations. *Nature Electronics*, 2(7):290–299, 2019.

[344] G Piccolboni, G Molas, JM Portal, R Coquand, Marc Bocquet, D Garbin, E Vianello, C Carabasse, V Delaye, C Pellissier, et al. Investigation of the potentialities of vertical resistive ram (vrram) for neuromorphic applications. In *2015 IEEE International Electron Devices Meeting (IEDM)*, pages 17–2. IEEE, 2015.

[345] Z Zhou, P Huang, YC Xiang, WS Shen, YD Zhao, YL Feng, B Gao, HQ Wu, H Qian, LF Liu, et al. A new hardware implementation approach of bnns based on nonlinear 2t2r synaptic cell. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 20–7. IEEE, 2018.

[346] Maxence Ernoult, Julie Grollier, and Damien Querlioz. Using memristors for robust local learning of hardware restricted boltzmann machines. *Scientific reports*, 9(1):1–15, 2019.

[347] Damien Querlioz, Philippe Dollfus, Olivier Bichler, and Christian Gamrat. Learning with memristive devices: How should we model their behavior? In *2011 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 150–156. IEEE, 2011.

[348] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[349] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[350] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[351] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online normalization for training neural networks. In *Advances in Neural Information Processing Systems*, pages 8433–8443, 2019.

[352] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[353] Johannes Christian Thiele, Olivier Bichler, and Antoine Dupret. Spikegrad: An ann-equivalent computation model for implementing backpropagation with spikes. *arXiv preprint arXiv:1906.00851*, 2019.

[354] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. In *Advances in neural information processing systems*, pages 7533–7544, 2019.

[355] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[356] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[357] Mario Miscuglio and Volker J Sorger. Photonic tensor cores for machine learning. *arXiv preprint arXiv:2002.03780*, 2020.

[358] Tiwei Wei. All-in-one design integrates microfluidic cooling into electronic chips, 2020.

[359] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[360] Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic, and Volume I User level Isa. The risc-v instruction set manual. *Volume I: User-Level ISA', version*, 2, 2014.

[361] Davide Rossi, Francesco Conti, Andrea Marongiu, Antonio Pullini, Igor Loi, Michael Gautschi, Giuseppe Tagliavini, Alessandro Capotondi, Philippe Flatresse, and Luca Benini. Pulp: A parallel ultra low power platform for next generation iot applications. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–39. IEEE, 2015.

[362] Sara Hooker. The hardware lottery. *arXiv preprint arXiv:2009.06489*, 2020.

[363] David JC MacKay. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992.

[364] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[365] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[366] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[367] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[368] Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 1(3), 2016.

[369] Bayesian deep learning.

[370] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

[371] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[372] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015.

[373] Tamara Broderick. Tutorial session: Variational bayes and beyond: Bayesian inference for big data, 2018.

[374] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[375] Thomas Dyhre Nielsen and Finn Verner Jensen. *Bayesian networks and decision graphs*. Springer Science & Business Media, 2009.

[376] Angus Galloway, Graham W Taylor, and Medhat Moussa. Attacking binarized neural networks. *arXiv preprint arXiv:1711.00449*, 2017.

[377] Sandy L Zabell. The rule of succession. *Erkenntnis*, 31(2-3):283–321, 1989.

**Titre :** Conception de circuits Neuromorphiques numériques exploitant des Nano-composants émergents

**Mots clés :** intelligence artificielle, électronique neuro-inspirée, systèmes neuromorphiques, synapses artificielles, mémoires émergentes

**Résumé :** (FR)
Depuis les années soixante-dix l'évolution des performances des circuits électroniques repose exclusivement sur l'amélioration des performances des transistors. Ce composant a des propriétés extraordinaires puisque lorsque ses dimensions sont réduites, toutes ses caractéristiques sont améliorées. Mais, dû à certaines limites physiques fondamentales, la diminution des dimensions des transistors n'est plus possible. Néanmoins, de nouveaux nano-composants mémoire innovants qui peuvent être intégré conjointement avec les transistors voient le jour tant au niveau académique qu'industriel, ce qui constitue une opportunité pour repenser complètement l'architecture des circuits électroniques actuels. L'une des voies de recherche possible est l'inspiration du fonctionnement du cerveau biologique. Ce dernier peut accomplir des tâches complexes et variées en consommant très peu d'énergie. Ces travaux de thèse explorent trois paradigmes neuro-inspirés pour l'utilisation de ces composants mémoire. Chacune de ces approches explore différentes problématiques du calcul en mémoire.

**Title :** Digital Implementation of Neuromorphic systems using Emerging Memory devices

**Keywords :** artificial intelligence, neuro-inspired electronics, neuromorphic systems, artificial synapse, emerging memories

**Abstract:** (EN)
While electronics has prospered inexorably for several decades, its leading source of progress will stop in the next coming years, due to the fundamental technological limits of transistors. Nevertheless, microelectronics is currently offering a major breakthrough: in recent years, memory technologies have undergone incredible progress, opening the way for multiple research venues in embedded systems. Additionally, a major feature for future years will be the ability to integrate different technologies on the same chip.
new emerging memory devices that can be embedded in the core of the CMOS, such as Resistive Random Access Memory (RRAM) or Spin Torque Magnetic Tunnel Junction (ST-MRAM) based on naturally intelligent in-memory-computing architecture. Three brain-inspired algorithms are carefully examined: Bayesian reasoning binarized neural networks, and an approach that further exploits the intrinsic behavior of components, population coding of neurons.
Each of these approaches explores different aspects of in-memory computing.

**Université Paris-Saclay**
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France