Embracing Imperfections : Hardware-compatible Neural Networks for Neuromorphic Computing

Atreya Majumdar

université PARIS-SACLAY

Embracing Imperfections: Hardware-compatible Neural Networks for Neuromorphic Computing

Accepter les imperfections : réseaux de neurones matériels pour le calcul neuromorphique

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°575 : Electrical, Optical, Bio-Physics and Engineering (EOBE)
Spécialité de doctorat : Electronique, Photonique et Micro-Nanotechnologies
Graduate School : Sciences de l'ingénierie et des systèmes
Référent : Faculté des Sciences d'Orsay

Thèse préparée au **Centre de Nanosciences et de Nanotechnologies**, (Université Paris-Saclay, CNRS), sous la direction de **Damien QUERLIOZ**, Chargé de recherche au C2N

Thèse soutenue à Paris-Saclay, le 22 mai 2023, par

Atreya MAJUMDAR

Composition du jury

Membres du jury avec voix délibérative

Dafiné RAVELOSONA Directeur de recherche, Université Paris-Saclay	Président
Louis HUTIN Ingénieur chercheur (HDR) , CEA-LETI Grenoble, France	Rapporteur & Examinateur
Damien DELERUYELLE Professeur à l'INSA, Lyon, France	Rapporteur & Examinateur
Alice MIZRAHI Ingénieure chercheure, Thales Research and Technology, France	Examinatrice
Luis LÓPEZ DÍAZ Professeur à la Universidad de Salamanca Espagne	Examinateur
Jean-Michel PORTAL Professeur à la Institut Matériaux Microélectronique Nanosciences de Provence (IM2NP)	Examinateur

THESE DE DOCTORAT
NNT :2023UPAST077

université PARIS-SACLAY

ÉCOLE DOCTORALE
Physique et ingénierie : Electrons, Photons, Sciences du vivant (EOBE)

Titre : Accepter les imperfections : réseaux de neurones matériels pour le calcul neuromorphique

Mots clés : réseaux de neurones, memristors, apprentissage, neuromorphique, imperfections, Réseaux neuronaux bayésiens

Résumé : Les récents développements en apprentissage profond ont repoussé les limites des possibilités avec de grands modèles de langage présentant des capacités exceptionnelles, des modèles pour la vision par ordinateur et le traitement du langage naturel dépassant les performances humaines. Cependant, ce progrès se fait au détriment d'une consommation d'énergie immense lors de la formation de ces modèles à grande échelle. De ce point de vue, l'avancée n'est pas durable, surtout compte tenu des préoccupations de changement climatique qui planent sur notre époque. L'énormité de la consommation d'énergie peut être attribuée à l'architecture des ordinateurs conventionnels, qui n'est pas optimisée pour la consommation d'énergie pour les applications d'apprentissage profond. D'autre part, le cerveau humain excelle dans cet aspect en effectuant des tâches complexes de reconnaissance de motifs avec un budget énergétique qui est des ordres de magnitude inférieurs à celui de son homologue informatique. La différence découle de la manière fondamentalement différente dont les calculs sont effectués dans le cerveau ; pour cette thèse, nous nous concentrons spécifiquement sur l'aspect de la co-localisation du calcul et de la mémoire, qui est présent dans le cerveau humain via les neurones et les synapses. En revanche, dans l'architecture de von Neumann d'un ordinateur moderne, la mémoire et les unités arithmétiques et logiques sont physiquement séparées, et une grande quantité d'énergie est dépensée dans le transfert d'informations entre ces unités. L'informatique en mémoire avec les technologies de mémoire émergentes est une piste prometteuse à cet égard, où la co-localisation de la mémoire et du traitement peut être réalisée, en particulier pour le type de calculs effectués dans les réseaux de neurones. Néanmoins, cette solution présente des défis en termes de performance car ces nouvelles classes de mémoires ont des imperfections différentes. Pour les mises en œuvre conventionnelles de réseaux de neurones avec des mémoires analogiques, ces imperfections peuvent considérablement affecter leurs performances. Le thème central de cette thèse est d'embrasser de telles imperfections pour les réseaux de neurones compatibles avec le matériel. Dans le chapitre 2, nous examinons spécifiquement l'impact de ces non-idéalités dans le contexte de la formation des réseaux de neurones. Nous proposons un modèle de dispositif basé sur la physique pour la mémoire à base d'HfOx qui correspond aux résultats expérimentaux et peut être incorporé dans des cadres d'apprentissage en profondeur. Des simulations de réseaux de neurones binaires avec ce modèle de dispositif montrent que l'apprentissage est possible même sous le bruit et les variabilités intrinsèques à une telle mémoire. Dans le chapitre 3, nous explorons l'impact des imperfections et des contraintes découlant à la fois du niveau de dispositif et de circuit sur la performance d'inférence des réseaux de neurones. Nous démontrons la robustesse des circuits de calcul en mémoire à base d'HfOx qui implémentent des réseaux de neurones binaires face à des contraintes telles que la taille limitée du tableau, l'alimentation électrique irrégulière et la variabilité des dispositifs. Avec le chapitre 4, nous exploitons la stochasticité des nanodispositifs spintroniques, qui est généralement considérée comme une imperfection pour des applications plus conventionnelles. Ce chapitre propose les réseaux de neurones binaires bayésiens qui peuvent être réalisés avec de tels dispositifs. Nous soulignons l'utilité de ces réseaux : l'immunité à la surajustement et la quantification de l'incertitude dans certains scénarios pour une tâche illustrative à deux lunes et un ensemble de données médicales. Les résultats présentés dans cette thèse montrent qu'avec des innovations dans les algorithmes, les circuits et les dispositifs de mémoire, les imperfections peuvent être véritablement embrassées et qu'un avenir conscient de l'énergie et axé sur l'IA peut être envisagé.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Physique et ingénierie : Electrons, Photons, Sciences du vivant (EOBE)

Title : Embracing Imperfections : Hardware-compatible Neural Networks for Neuromorphic Computing

Keywords : Neural Networks, Memristors, Learning, Neuromorphic, Imperfections, Bayesian Neural Networks

Abstract : Recent developments in deep learning have pushed the limits of possibilities with large language models exhibiting outstanding capabilities, models for computer vision, and natural language processing exceeding human-level performance. However, this progress comes at the expense of immense power consumption while training such large-scale models. From this perspective, the advance is not sustainable, especially considering the climate change concerns looming over the present day. The enormity of the energy consumption can be attributed to the architecture of conventional computers, which is not optimized for energy consumption for deep learning applications. On the other hand, the human brain excels at this aspect by performing complex pattern recognition tasks with a power budget that is orders of magnitude less than its computing counterpart. The difference arises from the fundamentally different way computation is done in the brain ; for this thesis, we specifically focus on the co-location aspect of computing and memory, which is present in the human brain via the neurons and the synapses. In contrast, in the von Neumann architecture of a modern computer, the memory and arithmetic-logic units are physically separated, and a large amount of energy is expended in the shuttling of information between these units. In-memory computing with emerging memory technologies is a promising lead in this regard, where the co-location of memory and processing can be achieved, especially for the type of computations performed in neural networks. Nevertheless, this solution presents challenges in terms of performance as these novel classes of memories have different imperfections. For conventional implementations of neural networks with analog memories, these imperfections can considerably affect their performance. The central theme of this thesis is to embrace such imperfections for hardware-compatible neural networks. In chapter 2, we specifically look at the impact of these non-idealities in the context of training neural networks. We propose a physics-based device model for HfOx-based memory that matches experimental results and can be incorporated within deep learning frameworks. Simulations of Binarized Neural Networks with this device model show that learning is possible even under noise and variabilities intrinsic to such memory. In chapter 3, we explore the impact of imperfections and constraints arising from both the device and circuit levels on the inference performance of neural networks. We demonstrate the robustness of HfOx-based in-memory computing circuits that implement binarized neural networks to constraints such as limited array size, irregular power supply, and device variability. With chapter 4, we harness the stochasticity of spintronics nanodevices, which is typically considered an imperfection for more conventional applications. This chapter proposes the Bayesian Binary Neural Networks that can be realized with such devices. We highlight the usefulness of such networks : immunity to overfitting and the quantification of uncertainty under some scenarios for an illustrative two moons task and a medical dataset. The results presented in this thesis show that with innovations in algorithms, circuits, and memory devices, imperfections can be truly embraced, and an energy-conscious, AI-driven future can be envisioned.

To the countless researchers, teachers, and students of science and technology whose names have been forgotten by history and will be forgotten.

Acknowledgements

The first time I visited C2N was on the Good Friday of 2019. I saw the shining building of our lab for the first time amidst large empty areas that were under construction. Over the next three and a half years, I saw how this empty part of the plateau turned into a bustling campus full of institutes, offices, restaurants, and student residences. Over these three and a half years, I also worked on my doctoral research. And research, as it is today, is fundamentally collaborative, where a significant contribution comes from collaborators and colleagues. In that regard, the first person whom I would like to thank is my supervisor Dr. Damien Querlioz. I am truly grateful for your considerate, encouraging, and effective supervision. Discussing research with you has been a fantastic experience, and I would greatly miss our weekly meetings. One of my fondest memories of these years is our discussion in Crete, overlooking the beautiful Mediterranean Sea. Thank you, especially for your support during the lockdown period; it was much less stressful than it could have been due to your constant support.

Also, many thanks to Prof. Damien Deleruyelle, Dr. Louis Hutin, Dr. Dafiné Ravelosona, Prof. Luis López Díaz, Dr. Alice Mizrahi, and Prof. Jean-Michel Portal for accepting to take part in my PhD committee and carefully reviewing this manuscript. Moreover, I found the questions and discussions during my thesis defense exciting and thought-provoking.

The projects done in this thesis were highly collaborative in nature. And as such, it involved the participation of many researchers of diverse backgrounds, and I learned a lot from them. The first project involved collaborating with Dr. Marc Bocquet, and I am grateful to you for your support and prompt response to all my questions. Prof. Jean-Michel Portal has played an instrumental role in the first two of my projects, and my understanding of circuits increased significantly from the discussions with you. For my second project, I particularly collaborated with two fantastic PhD researchers, Dr. Mona Ezzadeen and Fadi Jebali. Mona, I greatly enjoyed the long discussion and brainstorming sessions with you, and I will always cherish the nice publications we produced together. I would like to particularly thank Fadi for his help in making me understand the circuits part of the second project. The third project was a unique experience as I collaborated with brilliant theoretical and experimental physicists. I learned much about Spintronics from Dr. Liza Herrera Diez and Prof. Luis López Díaz. Liza, thank you for being such a joyful and energetic presence in the lab, and I enjoyed our discussions about art and literature. In this project, I also collaborated with the brilliant researcher Djohan Bonnet. One of my favorite scientific moments in the lab was figuring out with you, the correct metrics for the estimation of uncertainty. I thoroughly enjoyed interacting and working with you, and I hope our paths cross again.

The lab is like a second home for researchers, and I was truly blessed to have a beautiful lab. But it's the people of the lab who truly made it feel like home. I feel grateful to have had amazing colleagues throughout my time. Bogdan, it has been fantastic knowing you, and I thank you for your assistance with the initial things at the lab, for inspiring my passion for Deep Learning

and for being a wonderful friend. The pizza dinner at your place was a nice bonding moment for everyone in the group. Thank you, Maxence, for being such a fun and supportive person! You lit up the lab's atmosphere and were a compassionate and supportive friend. I owe many of my accomplishments to Tifenn; thank you for teaching me the ropes of PyTorch and Binarized Neural Networks, which later became the foundations of my PhD thesis. Thank you, Axel, for all your patience and practical help with the first two projects. I loved interacting with such an excellent yet down-to-earth person. Before the pandemic hit us, talking to and working with all four of you was very fun. I missed your presence at the lab quite a lot.

It is not easy acknowledging the contribution of two of my best friends at the lab, Kamel and Xing, just because of the sheer amount of things I need to mention! I have many marvellous memories with both of you- from finishing challenging hikes in the Alpes to swimming in the warm Mediterranean Sea to discussing intense research. Surviving the multiple lockdowns would have been significantly more difficult without your presence. Kamel, thank you for teaching me to play football; you really rekindled my old love. Thank you also for sharing the very *different* kind of humor we possess. It was incredible how sometimes we could read each other's minds. And Xing, you are among the most courageous, smart, and considerate people I have ever met. Your cool, composed, hard-working personality is a massive inspiration for me. I sure hope that we can keep in touch.

Next, I would like to acknowledge the *next generation* of researchers in the group. Clément, thank you for always being there for my circuit-related questions and teaching me much about French culture, especially food. Thank you, Marie, for being such a great friend with whom I could freely discuss research and other fun topics. Your spiritful attitude contributes to the good mood of the group. One of the best memories during my PhD was when you presented a part of my first project! Thank you, Thibaut, for the nice moments at the lab; you were amazing as an officemate. Maryam, it was really fun knowing you and I enjoyed all our interactions. I would like to both thank and wish the very best to Thomas, both Adriens, Theo, and Akib for their contribution to the group and for their upcoming PhD journeys. I hope all of you enjoy your doctoral research days as much as I did in Integnano. I want to thank Bastien for the few days you spent at our lab and became a close colleague. Above all, I believe in all the upcoming projects in the group involving medical datasets, Eq-Prop in hardware, forward-forward algorithm, and the Ising machine. I am looking forward to the published results of those projects. I want to acknowledge Naim for being a cherished friend and allowing me to discuss science and engineering in my mother tongue. The four months you spent at our lab was a very special time for me, and I hope we keep in touch.

I would also like to thank my other friends in Integnano, firstly, Rohit. I shared the most amount of time at the office with you. I appreciate your kind nature, interesting sense of humor, and sharing common love for Biryani at Cafe Sunderban. Tanvi, well, it's also hard to acknowledge you! I got a friend and elder sister in you, which I never imagined I would find in this foreign land. I greatly cherished our discussions about food (although it annoyed Kamel)

and life in Germany. Thank you, Gyan, for being the Dutch friend I was missing from my life after my masters. It was great fun in all our interactions, and I would like to especially thank you for organizing and actively taking part in various group activities. Thank you, Song, for our occasional chats; it was nice to know you. Subhajit, thank you for being a good friend; I appreciate our frank discussions about differences in industry and academia.

I also had the incredible opportunity to befriend others at C2N outside our research group. I met Arup da at the 2019 end-of-year lunch, and since then, you have been such a remarkable presence during the first half of my PhD. I greatly appreciate the hospitality and care that you and Dipanita di showered on me. Thank you, Ritwik da, for help on many fronts; life at the lab would have been very different without you. I will miss the breaks when I can discuss many things at your office. Finally, I thank my closest friend in France, Sukanya. You meant a lot to me, and I will never forget the many experiences we shared. From going for long walks during the depressing lockdowns to cooking sumptuous meals from all over India to sharing happiness, and sadness; you are one of the most precious friends to me whom I got to know in France. I hope we meet in the future very soon.

I also want to thank all the people working at the C2N who contribute to making the lab run smoothly. Many thanks to Christophe Chassat and Alain Péan from IT, Lydia Andalon, Melissa Legendre, Léa Lemmaitre, Sylvie Lamour, and Bernadette Laborde from the administration, Sophie Bouchoule, Emilia Davodeau, and Jean-Christophe Ginefri from the doctoral school. Thanks also to the employees cleaning the lab every morning.

I feel very fortunate to find a family in the Bengali people residing in and around Paris. Centering around our great festival of Durga puja, we formed a close-knit family of people of different ages and backgrounds. Pabitra da and Debasmita di have been like guardians to me with their warm and welcoming presence. I will miss visiting your house in the lovely neighborhoods of Bures-sur-Yvette. I am grateful to Avigyan da and Atreyee di for all the beautiful memories we had together (not to forget the arrival of Arnaa, too!). You seldom find such amazing, kind-hearted people with whom you can connect on many aspects. Thank you for being my favorite people! Rohan da and Soumi di, thank you for being such incredible hosts for me. I loved our discussions at your place about anything and everything about life, people, and ChatGPT! I can't imagine a dull moment when I was in your company, and I would continue to cherish all those moments. Upasika, Anambar da, and Anwesh, how fun of a ride it has been to talk and laugh with you! It was tough saying goodbye to all of you. Sipra mashi, thank you for being such a lively person; I really look up to you in terms of your outlook on life. Performing theatre and other cultural activities with all of you will be some of the most memorable moments of my life. Saheli di, Dibya da, Subrata da, Rajesh da, Sonima di, thank you all for making life on the plateau much more enjoyable in the first half of my PhD. Thank you, Kanka da, for your support; I hope we meet again soon.

I want to end the acknowledgments by first thanking my family. Maa, Baba, bon, bhai, kaka, kamma, and thakun, this would not have been possible without your incredible support and

love. Your sacrifice, advice, and encouragement have been instrumental in completing my doctoral studies. And last but not least, I would like to thank Tista for being an absolutely amazing partner in my PhD journey. Your presence has been like a shelter of warmth and comfort for me, and it goes without saying that this wouldn't have been possible without you. Being a fellow PhD researcher, you truly understood what this journey feels like and stood by me throughout. Thank you for being there, thank you everything.

Contents

Introduction

> “The beginning is the most important part of the work.”
>
> — Plato

PRESENTLY, we are at a critical juncture where on the one hand, we are experiencing the aggravated effects of climate change. One of the key reasons for this is the emission caused during the generation of electricity from fossil fuels [1]. On the other hand, we are making rapid progress in artificial intelligence (AI), with the large language models already showing preliminary indications resembling artificial general intelligence [2]. The recent progress takes us a step closer to the promises of AI that involve discovering drugs for acute diseases, self-driving cars, and other path-breaking innovations.

However, there is a cost to this: to develop, train and use the state-of-the-art deep learning models, existing computers expend a lot of energy. This type of computation, typically done in data centers with many Graphics Processing Units and other dedicated accelerators, is not usually optimized for their power consumption. Training a single model consumes more power than the amount consumed by 100 households in the United States in a year [3]. This number would only grow continuously with the ever-increasing model size and computational complexities. The resultant carbon footprint would be humongous, and this development is not sustainable from an environmental point of view.

If we consider the computation process at an architectural level, the bottleneck in terms of energy consumption is related to the shuttling of data between the memory and logic units. During the training process of a neural network, three main operations are performed: shuttling data to and from memory and performing multiplication and addition operations in the processing unit. Among these operations, the transmission of information is energetically the most expensive. This computer architecture, called the von Neumann architecture, fundamentally differs from another system adept at pattern recognition tasks: the human brain. It can perform vision, natural language processing, logical deduction, and planning with an energy budget that is orders of magnitude less than what is typically consumed by a modern deep learning model. The brain computes differently: the connectivity is massive with substantial redundancy, the learning rules are local, the information is propagated in electro-chemically induced voltage spikes, and the logic and memory elements are co-located in the form of neurons and synapses. The field of neuromorphic computing aims to emulate or mimic the brain in terms of these aspects to perform more efficient computation. This thesis is about neuromorphic computing with architectural inspiration and attempts to mimic biology from the in-memory or near-memory computing perspective.

In particular, I investigate resistance-based emerging memory technologies for neural networks since they provide a more energy-efficient, CMOS-compatible, non-volatile substrate to perform computation near the memory than their more conventional counterparts (SRAM or DRAM). Low-power, non-volatile memories are well-suited for *edge* applications where power efficiency is prioritized. Additionally, their non-volatile nature is particularly advantageous for equipment that is not constantly used since no power is needed to store a state. Despite such advantages, emerging memories such as oxide-based resistive memories, phase change memories, and magnetic random access memories suffer from imperfections that can dramatically

affect the performance of neural networks. In this thesis, I investigate and present hardware-compatible neural networks which are tolerant to such imperfections to a reasonable extent and can even embrace them for performing computation. I attempt to answer the following questions in the second, third, and fourth chapters.

- **Chapter 2** How can we learn with imperfect oxide-based filamentary resistive memory?
- **Chapter 3** What is the impact of errors and constraints arising at the circuit level on the inference of neural networks?
- **Chapter 4** Can we harness the stochastic nature of stochastic devices to perform probabilistic computing? And what additional advantages could it have?

More specifically, in Chapter 1, I lay the foundation of the thesis by elaborating on the background of my research. I start by describing the advancements made in computers and computer memory that subsequently facilitated the development of AI, especially deep learning. I chronicle the rise and fall of AI and the post-2000s resurgence. A detailed description of neural networks in supervised learning follows this. The ever-increasing sizes of models and the associated required computational prowess and their environmental consequences are then discussed. The concept of neuromorphic computing is introduced as a possible solution to this. After that, emerging memory technologies are discussed as the ideal candidates for in-memory computing systems capable of implementing neural networks in hardware. In this context, the different existing ideas about the hardware realization of neural networks are discussed in detail. The chapter concludes with a thorough discussion about imperfections in these kinds of memories, especially those that offer hindrances to learning, such as device-to-device variability, cycle-to-cyle variability, asymmetry, and nonlinearity.

The second chapter is a study done in collaboration with Dr. Marc Bocquet from the Aix-Marseille University that was published in the journal IEEE Transactions on Electron Devices and was also presented at the CVPR 21 (conference on computer vision and pattern recognition) and the Neal 2022 (Göttingen, Germany) conferences, both as posters [4]. This chapter is about implementing learning in the weak RESET regime of HfO_x-based filamentary resistive RAM using binarized neural networks. I start this chapter by presenting the background of this work, focussing on the importance of on-chip learning and emphasizing the main issues that make it challenging. Next, we introduce the memory technology and detail its co-integration with CMOS and the significance of the weak RESET regime that enhances the endurance of such memory devices, a crucial parameter for on-chip learning. Next, the main focus of this chapter is presented, where I developed a model that considers the different types of variabilities to explain the variation in resistance. Furthermore, I fitted the model to the experimental data and compared the simulations. After this, I describe binarized neural networks as an algorithm suitable for learning with such noisy memory. Then I outline the training process details and illustrate how I incorporated our device model within the PyTorch deep learning framework to simulate learning with these devices. The simulations are done to learn the MNIST

and CIFAR-10 datasets, and the test accuracies exemplify the robustness to different types of imperfections. I conclude the chapter by highlighting how this approach can be generalized to the simulation of other memory technologies.

Chapter 3 is about the inference in binarized neural networks with constraints from circuit-level implementation. This chapter is based on two scientific articles, one is under revision, and the other is under preparation. The first and second studies presented here were done in collaboration with Dr. Mona Ezzadeen and Fadi Jebali from Aix-Marseille University. In the first section of this chapter, I introduce general ideas related to the circuit-based implementation of binarized neural networks and illustrate the significant sources of errors originating from electronic circuits and memories. Next, I present the first study, detailing the circuit used to implement inference in binarized neural networks, the sources of errors, and their analyses. These errors are incorporated into neural network simulations to test how robust the prediction accuracies are to such errors. For the second study, I motivate the scenarios under which this type of circuit could be used: edge applications where the power originates from an energy harvesting system with irregular performance. Then, I present the circuits designed and characterized at the Aix-Marseille University, fabricated in CEA-Leti Grenoble, and present the associated errors and constraints related to the design. Here, I proposed and demonstrated an approach that circumvents array-size-related constraints at the cost of a slight degradation in accuracy. Finally, I use the experimentally characterized error for neural network simulations to show that the binarized neural networks exhibit robust computation even under a low power supply. This chapter highlights the suitability of binarized neural networks for inference, even with different levels of imperfection and constraints, which is especially promising for edge applications.

Chapter 4 differs from the earlier two chapters in that it accepts the imperfection and instead utilizes it for computation. In this context, I discuss Bayesian Binary Neural Networks, the probabilistic analog of binary neural networks where only the weights are binarized. This is a study in progress for which we are starting to prepare a manuscript, and it was presented as a poster at the MagnEFi 2022 conference in Crete, Greece. I begin the chapter by reviewing the theory behind probability-based computing, focussing on ideas related to Bayesian deep learning methods. Next, I discuss some recent studies where the concept of probability-based computing has been realized with emerging memory devices. These ideas give us a glimpse at the potential of this computing paradigm. After that, I discuss the Bayesian Binary Neural Networks theory, emphasizing its differences from its deterministic analog. Here, I introduce the idea of quantifying uncertainty, which is one of the main advantages of using Bayesian Neural Networks and is essential for safety-critical applications. I use this type of neural network for a toy example, the two moons dataset, and demonstrate some scenarios under which our neural network provides robustness or more insight than the deterministic network. After this, I utilized this algorithm to learn actual medical tasks, the MIT-BIH dataset for arrhythmia detection, and showcased its unique advantages. Finally, we end this chapter by discussing some

spintronics-based possible systems that could be used for realizing this algorithm and present results related to performing inference with them.

Chapter 1

Hardware implementation of deep learning

> "If I have seen further, it is by standing on the shoulders of giants."
>
> Sir Isaac NEWTON

THIS CHAPTER serves as a preface to the new ideas and results presented in this thesis. We start by looking back at the past; the historical developments that led to the technology today. After that, the present is discussed: the state-of-the-art and its shortcomings, and from there, the necessity of the research presented in this thesis is motivated.

After discussing the history of computing and memory, it introduces deep learning as a consequence of modern memory-compute capabilities and the abundance of data. Then, it highlights the problems and challenges that deep learning will encounter in the near future and, with this background, introduces brain-inspired or neuromorphic computing. It reviews the development of neuromorphic computing in the light of implementing deep neural networks in hardware with brain-inspired principles that aim to address the energy constraints deep learning faces. Finally, it elaborated on the challenges encompassing the hardware implementation related to the various imperfections present in emerging technologies.

1.1 A brief history of memory and computing

The scientific and technological progress of our race is evident from the fact that it took us about 4000 years from the invention of the wheel to the first successful airplane launch, but only just 66 years between the first airplane and the landing on the moon by Neil Armstrong. Many of these advancements were facilitated by the rapid progress made in terms of computing technology; the invention of the modern computer allowed us to automate complicated tasks and do large-scale calculations fast.

The modern computer we use so ubiquitously is the result of technological progress spanning centuries. It required simultaneous developments in multiple fields of science, including physics, mathematics, electronics, and computer science. Each successive generation of computing technology expanded the boundaries of our capabilities, thereby creating newer opportunities that were previously unimaginable. The cyclic nature of the necessity-invention cycle propelled growth at an exponential rate as well as the need for computational power and memory.

In computing terms, memory refers to the information a certain calculation needs to execute. Fundamentally, a computer is composed of two primary components: the memory unit, which stores information, and the arithmetic-logic unit, which performs operations on that information. The memory required for computation strongly correlates with the task's complexity, as depicted in fig. 1.1. If we focus on some distinct events in the history of human civilization: the invention of the abacus, the conceptualization of the Analytical engine, the first general-purpose computer by Charles Babbage in 1837, the first landing of a human on the moon in 1969, the *solution* of the protein folding problem by the deep learning program AlphaFold 2, and the release of the large language model-based ChatGPT in 2022, we see that the memory associated with each of these developments, as well as the computational needs increase exponentially. Another thing to notice here is that the time difference between such

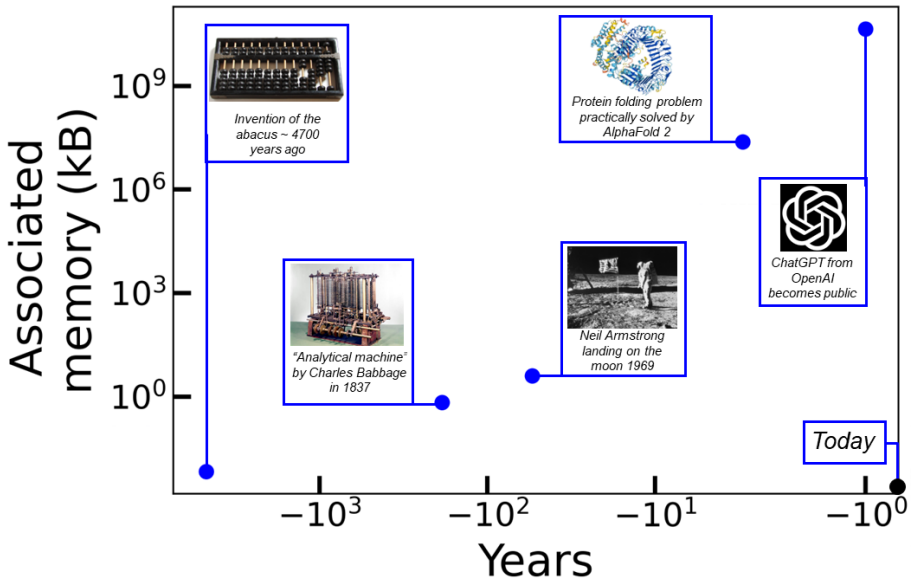


Figure 1.1: Timescales and the associated memory for some of humanity's most important inventions or events. The Sumerian abacus, considered one of the foremost calculating machines, appears in archaeological evidence as early as 2700 BCE. Charles Babbage invented the Analytical engine, considered the first general-purpose computer, in 1837. In 1969, Neil Armstrong became the first astronaut to land on the moon, embarking upon the Apollo 11 spacecraft, which had the Apollo Guidance Computer as the main computing unit. The 50-year-old protein folding problem, one of the biggest questions in structural biology, was solved by the AlphaFold 2 deep learning algorithm in the year 2022 at the CASP 14 event. In 2022 OpenAI released ChatGPT, an online bot based on the large language model GPT3 that can answer questions from a human prompt. In this figure, the associated memory for the deep learning models has been calculated based on the total number of parameters present, which essentially gives a lower limit on the required memory because training such models requires more memory than just the parameters.

major advances is shortening, signifying the speed of these developments.

With this importance of computer memory in mind, let us look at the historical development of computers and the role memory has played in them.

1.1.1 Development of computing

A typical smartphone today can store up to 4 billion bytes of data and can function like a desktop computer. To understand the root of the invention of such technological components, we turn the pages of history to study the progress made in the early days when the computer had mechanical parts, unlike today.

1.1.1.1 Mechanical calculators

The abacus is the earliest instrument that can be called a calculator or, very broadly, a computer. As shown in fig. 1.2(a), the abacus consists of several movable columns of beads, and a single arrangement of the beads denotes a single number. This device allows for calculations such as addition, subtraction, multiplication, division, and even taking the square or cube root of a number. Such computations are performed by manually moving the beads [5]. Another early prototype of the modern calculator was invented by the French inventor Blaise Pascal in 1643 (fig. 1.2(b)) [6]. The Pascal calculator relied on a set of gears and springs to implement

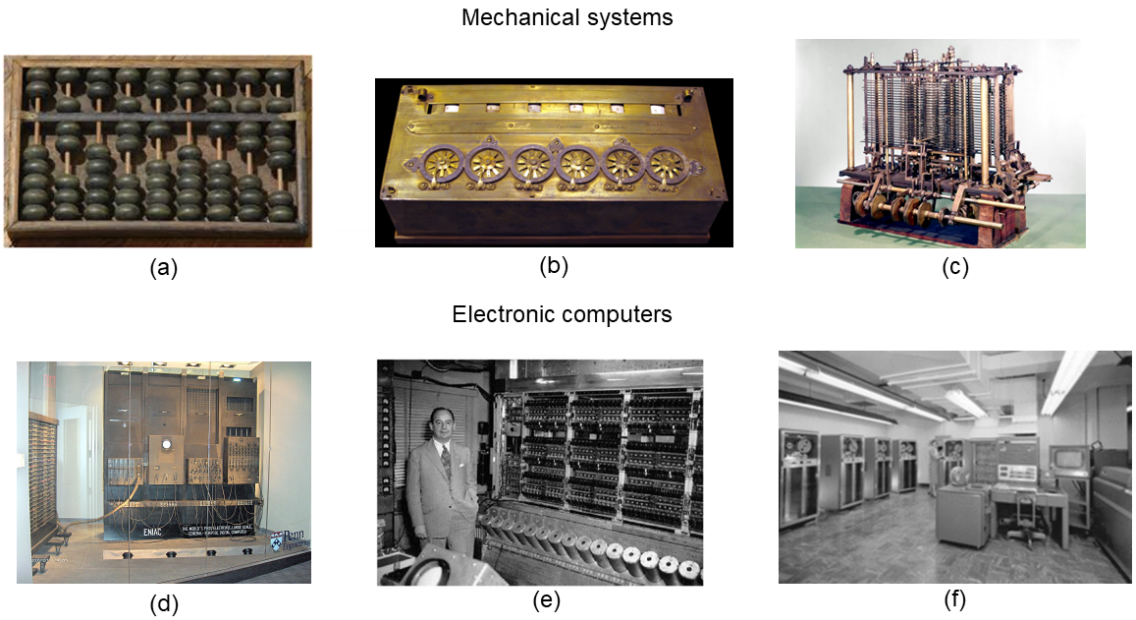


Figure 1.2: Computers through the ages. (a) An abacus, the first ever calculating device used as early as 2400 BCE by the Babylonians. (b) The Pascal calculator. (c) A model of Charles Babbage's Analytical engine. (d) The ENIAC computer. (e) John von Neumann posing with the EDVAC computer. (f) The IBM 704 mainframe computer occupies a whole room.

simple arithmetic operations on numbers represented by the wheels' position. While these devices are capable of efficiently performing arithmetic operations, they are not programmable, thus rendering them unsuitable for automation purposes. The Jacquard loom machine is one of the earliest examples of an instrument where a textile loom could be programmed to function in a certain way. The machine is essentially a control mechanism for the loom in which a chain of *punched cards* are used as the input for a pattern, and the loom patterns automatically [7].

However, all the aforementioned examples are quite far from modern computers. The first design of a general-purpose computer, albeit mechanical, was proposed by Charles Babbage in 1837, named the Analytical engine (fig 1.2(c)). This was the first computer to have integrated

memory in the form of counter wheels, a dedicated arithmetic logic unit, a control flow that enabled loops and conditional branching, an input system with punched cards, and even a printer for producing the output [8]. After that, there were more instances of mechanical or electro-mechanical computers, but all of them had constraints intrinsic to mechanical systems, like speed issues and mechanical wear-and-tear. Furthermore, these were analog computers and consequently were less robust to noise and inefficient.

1.1.1.2 Electronic computers

It took more than a century after this for the first digital, electronic, programmable computer to emerge. In 1945, the Electronic Numerical Integrator and Computer, or ENIAC, was completed at the United State army's Ballistic Research Laboratory [9]. The construction of this computer, shown in fig. 1.2 (d), was enabled by the developments in electronics in the earlier half of the twentieth century. In particular, the invention of the thermionic vacuum tube paved the way for performing logic efficiently [10]. A technological successor to ENIAC was EDVAC (Electronic Discrete Variable Automatic Computer), which was completed in 1949 at the Moore School of Electrical Engineering in Pennsylvania [11]. The celebrated engineer John von Neumann was involved with this project as a consultant (fig. 1.2(e)), and he proposed in his monograph *First Draft of a Report on the EDVAC* the architecture-level organization of a computer [12]. This came to be known as the *von Neumann architecture*, and it represented a computer architecture with several components: a memory unit, an independent arithmetic logic unit, a control unit, and mechanisms for input and output. Later in this chapter, we shall discuss this architecture more in the context of the energy efficiency of a modern computer. The type of internal memory used for these computers were mercury-filled tube-based delay lines, a type of acoustic memory [13].

The next generation of computers was heralded by the invention of the magnetic core memory when Jay Forrester utilized the hysteresis property of magnetic cores. The main advantage of this was that it was truly random access, unlike its predecessors which had a serialized relay of information. The earliest computer to use this was Whirlwind I in 1951 [14], the first computer ever to produce real-time output and function in parallel mode. The year 1954 saw the advent of the first mass-produced computer IBM 704 (fig. 1.2(f)), which was a digital mainframe computer with hardware capable of performing floating-point arithmetic. The following decades saw the development of transistors and integrated circuits, replacing vacuum tubes totally. These rapidly decreased the computer's cost and size and culminated in the invention of the first personal desktop computer, IBM-PC, in 1966 [15, 16].

The history of computers shows us that the advancement of computers didn't happen in a vacuum, and the technological progress of memory devices was closely intertwined with it. With this in mind, let us delve into the history of the development of computer memory itself.

1.1.2 History of computer memory and storage

Computer memory is any system or device storing information for immediate computations. Storage memory, on the other hand, refers to the device that can preserve information over a much longer time period and that is not necessary for computations very frequently. For example, in a smartphone, the memory is where the operating system and application software are stored, whereas the storage consists of photos, videos, and documents. Thus, a computer needs a hierarchy of different types of memories to function.

Punched cards can be considered the earliest forms of memory that a computer could use. In these cards, the information was encoded in the form of holes in cards of stiff paper [17]. A typical characteristic of these early forms of memory was the physical state of the device used to encode the information. A fundamental problem with these is that the physical dimension required to represent a single bit of data is quite big, and thus, the access speed is limited by the arrangement of the data. The design of Babbage's Analytical engine used the rotational state of counter wheels (fig. 1.3(a)) for internal memory and punched cards to input information.

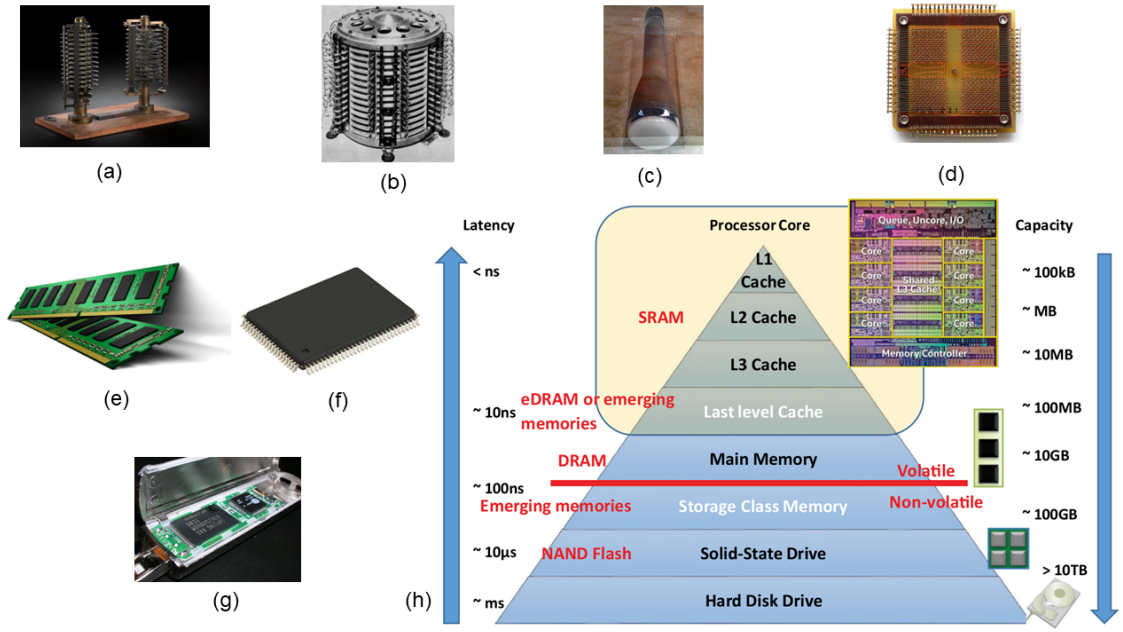


Figure 1.3: Memories of yesterdays and today. (a) Counter wheels played the role of main memory in Charles Babbage's proposed Analytical engine. (b) The magnetic drum memory, along with the read-write heads attached to the cylinder. (c) The William-Kilburn tube, which is a modified Cathode ray tube. (d) The magnetic core memory with the toroids is arranged in four grids and connected by wires for reading and writing. The commercially available forms of (e) Dynamic Random Access Memory (DRAM) and (f) Static Random Access Memory (SRAM). (g) NAND Flash memory inside a USB stick. (h) The hierarchy of memory and storage in modern computers (adapted from [18])

As shown in fig. 1.3(b), magnetic drum memory was one of the earliest devices to use mag-

netism to store data. These were large metallic cylinders with ferromagnetic coating on them, and the magnetic polarity of the film represented the data. There were multiple parallel read-write heads to read and write information from and on the drums as the drum rotated; thus, the access speed was limited by the rotation speed of the drum. Nevertheless, this form of memory was used until as late as the 1960s [19]. An earlier version of this utilized capacitors to achieve the same, but a major drawback was that those needed to be refreshed periodically. A similar memory device is the magnetic tape data storage, which also had a long latency owing to its physical shape but has superior data storage density and endurance (the ability to endure many cycles of switching of memory bits), both of which are important attributes by themselves and are the reasons for its survival even to this day for data archiving.

In the early years of 1940, the success of delay lines in early RADAR systems inspired the development of mercury-based delay lines as a memory device. In this technology, a tube filled with mercury, a quartz transducer, and a receiver comprised a memory device. The transducer was able to produce acoustic waves in the tube, which propagated and were received at the other end. The presence or absence of a wave denoted a bit in such memories, and a single tube typically stored about 1000 bits of data with an access time of just over 200 μs [20]. The mercury-tube delay lines were used in the UNIVAC I (Universal Automatic Computer I), which was the successor to the ENIAC, and the first digital general-purpose computer manufactured for business applications [21].

Apart from latency, data storage density, and read and write times, the fashion in which data is accessed is another important metric. Conventionally, there are two different types: sequential access memory (SAM) and random access memory (RAM). In sequential access memories, the data is stored in a sequence, so there's always a substantial time needed to find specific data. On the contrary, RAM enables the access of data in any order. The delay line memories were of the first kind owing to their intrinsic sequential nature and thus were not suitable for situations that necessitated random access. Around the same time, the William-Kilburn tube was also invented as a random access memory. This tube (fig. 1.3(c)) is a customized Cathode ray tube, where the electron beam could create a charge well in the face of this tube, and a read plate was used to sense the absence or presence of that well representing the memory. This was random access since the electron beam could position itself anywhere on the screen quickly, and thus any bit of data could be accessed at a time. This system also required periodic refresh due to the leakage of charge, and the data read was also destructive [22].

The next major development came in the form of magnetic core memories. A single bit of this memory comprised a core, a magnetic toroid with an electric wire connected to it. The current flow magnetized the core, the direction of the magnetization depending upon the direction of the current and encoding a single bit. This state would be preserved even when no current flows through the core, owing to the hysteresis property of ferromagnets. Such cores were arranged in a grid (fig. 1.3 (d)) with wires connected to facilitate the reading, writing, and selection of a single core. The success of this memory can be attributed to its random access

nature, robustness, and the lower access time of 5 μs compared to the delay-line memories [14, 19].

1.1.3 Memory in a modern computer

Further advancements in computer memory and storage happened hand-in-hand with the progress of transistor technology. The main advantages of transistors are that they are smaller, faster, less power-hungry, and cheaper than the existing solutions of that time. In the year 1966, at the IBM Thomas J. Watson Research Center, the field effect transistor memory or DRAM (dynamic Random Access Memory) was invented. It was built on a semiconductor process with a single silicon transistor and a capacitor in a cell, and the two memory states are represented by the charged or uncharged state of the capacitor. The capacitor circuit element suffers from the leakage of the charge, and hence, this also needs periodic refresh like the Williams tube (fig. 1.3 (e)) [23], from which it derives the 'dynamic' in its name.

Although DRAM is cheap and has a low area overhead, the data access is slower (10-100 ns), which is not desirable in certain cases, like in cache memory or in internal registers. SRAM, or Static Random Access Memory, invented in the early 1970s, is the choice of memory in such cases. A single SRAM cell consists of 6 transistors and hence can have a significantly lower memory density and be more expensive compared to a DRAM which has a single transistor per cell. However, the on-chip location of SRAM, coupled with the fact that it has a small array size with less number of wires and consequently less delay, the memory can be accessed much quicker (a few ns) (fig. 1.3 (f) [18].

Despite the fact that DRAM and SRAM are crucial elements of computer memory, they are volatile; that is, the cell loses its memory when the power supply is turned off and thus cannot be used for storage. Dr. Fujio Masouka, working at Toshiba in the year 1987, invented the Flash memory, a non-volatile, electronically programmable memory [19]. A single cell of such a memory consists of a floating gate transistor, where an additional insulated gate lies between the control gate and the MOSFET channel. The presence or absence of charge in this gate represents the memory states. Since it is electrically insulated, the memory is stored even in the absence of a voltage supply. The NAND flash cell is typically used for storage due to its cheaper cost and high memory density. It is the principal building block for solid-state drives (SSDs), smartphone storage, and USB sticks, as shown in fig. 1.3 (g). Although it is more expensive than the conventional hard-disk drive (HDD), which is magnetic storage with mechanically moving parts, the SSD is faster, smaller, less noisy, more durable, and more energy-efficient.

A computer is a complex device that requires different types of memory storage, each with specific requirements for various applications. The range of storage options spans from the cache memory, which requires speed over size, to the hard disk drive storage, which prioritizes a large size over access speed. Fig. 1.3 (h) details the complete memory hierarchy present in the modern computer that has been discussed in this section. The timescales span from less than a nanosecond to about a couple of milliseconds, and the memory capacity extends from 100s

of kB to more than 10s of TB. Within the extent of these huge time and memory scales, there are gaps that different emerging memory technologies are trying to bridge. We shall discuss such memory later in this chapter. It must be emphasized that the rapid development of transistor technology is enabling the integration of more and more transistors on a single chip, and that, in turn, is giving us access to smaller and more powerful computers. The IBM 704 mainframe computer, with a memory of about 18.4 kB, occupied a whole room, whereas a typical smartphone today has a RAM of 8 GB. Moore's law summarizes this in the form of the statement that the number of transistors on a microchip doubles every two years [24, 25]. Amongst the many benefits that were derived from this, the progress in the field of AI is quite significant.

1.2 The rise of deep learning

The exponential development of technology has been driven by the fact that development in a single very important domain affects several others, which continues the same way to produce an avalanche. The huge advancements in computing power, memory capacity, and the availability of an astronomical amount of data have led to the resurgence of deep learning, a special class of artificial intelligence (AI) algorithms.

We start this section with a short history of AI. Then we move on to specifically focus on the resurgence of this field with the support of memory capabilities. Then finally, we discuss the basic ideas behind the deep learning algorithm.

1.2.1 The rise and fall of AI

The modern form of AI that we observe today arose from innovations in different science and technology domains. Algorithmically, the earliest progress could be traced back to the first half of the 19th century when the least square regression and gradient descent methods were formulated, which have been the backbone of the deep learning algorithms, even today [26, 27]. But, it wasn't until the summer of 1956 that the field of AI was formalized at the Dartmouth workshop in the United States of America [28]. Amongst the organizers were John McCarthy, who was the person to coin the term AI to this domain, Nathaniel Rochester, the chief architect of the IBM 701 computer, and Claude Shannon, who is considered the father of information theory. In the conference, topics like creativity, abstraction, computer architecture, computational theory, neural networks, and natural language processing were discussed, which remain relevant even today [29].

Developments in these aspects were driven mostly by mathematicians and computer scientists, but there was another line of progress that contributed equally. These were attempts by a group of biologists who took inspiration from the nervous system of animals to build *bottom-up* model for intelligence. Warren McCulloch and Walter Pitts proposed a simple model of neurons based on its topology where a single neuron accumulates the inputs that it receives

weighted by the synaptic connection strengths and applies a non-linearity to it to produce the output [30]. This model, shown in fig. 1.4, is widely used in neural networks. Biophysicists Alan Hodgkin and Andrew Huxley extended this idea by introducing the concept of dynamics, which, as it turns out, is a crucial aspect of biological neurons [31]. The Canadian neuropsychologist Donald Hebb took this idea of dynamics to suggest a simple learning algorithm based on the temporal correlation of connected neurons, which came to be known as Hebbian learning [32].

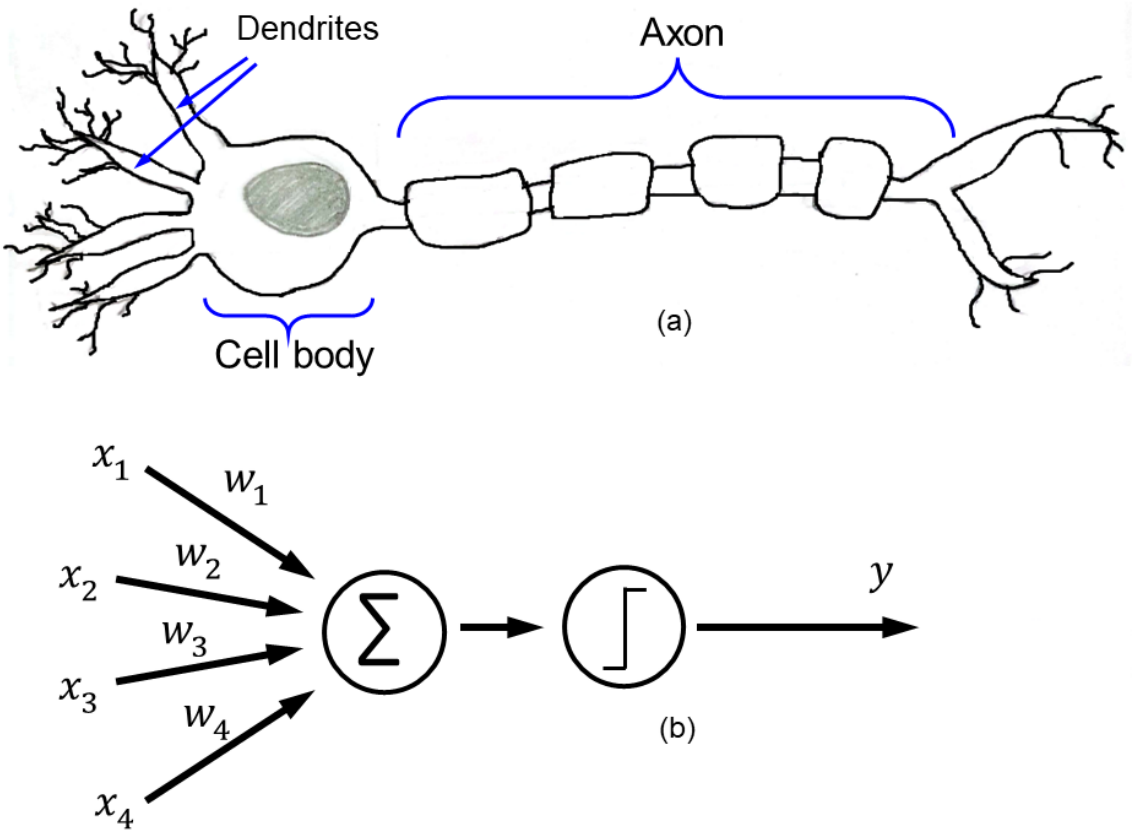


Figure 1.4: The analogy between a biological neuron (a) and an artificial neuron (b) as proposed by McCulloch and Pitts. Through their respective synapses, the dendrites bring in electrical voltage pulses to the cell body, where the signal undergoes processing and eventually propagates through the axon onto the next set of neurons. In an artificial neuron, the inputs x_1, x_2, x_3, x_4 are summed up after being weighted up by their respective weights w_1, w_2, w_3, w_4, and then passed through a non-linear activation function to output y.

The question of having the correct learning rule bugs researchers even today and the bioplausibility of a certain learning rule is often debated. In [30], McCulloch and Pitts discussed ideas that led to the first implementation of a perceptron by Frank Rosenblatt, which is the most rudimentary form of a neural network [33], and the structure is as shown in fig. 1.4(b). It can do a classification or regression task where the N inputs $x_1, x_2, \ldots, x_N$ are fed into the

perceptron, where the output y is calculated by taking their linear combination with weights $w_1, w_2, \ldots, w_N$, and after passing through a non-linear function, which was the Heavyside step function in the original perceptron formulation. The nonlinearity is an important ingredient as a linear setup could learn functions that were linear, which isn't the case most of the time in reality. Learning a good mapping from the input to the output corresponds to learning an appropriate set of $w_1, w_2, \ldots, w_N$, which are referred to as the synaptic parameters because of their functional resemblance to biological synapses. The learning rule for the perceptrons in the simple delta rule is based on the gradient descent algorithm [34]. In this algorithm, a loss metric is defined, which quantifies how far the perceptron is from giving a correct output. Then the parameters are updated in a way that minimizes this metric. This is in contrast to the bio-plausible Hebbian learning and solely depends on the iterative arithmetic updates of the parameters of the system. Although the very first implementation of the perceptron was in software on the IBM 704 computer, eventually, a machine was designed called the Mark I perceptron in 1958. It was connected to 400 photocells in a 20×20 grid, which was then connected to neurons, and the synaptic weights were implemented by potentiometers whose resistance value encoded the weights. The update of the weights was carried out by electrical motors [35]. This physics-based implementation is surprisingly not dissimilar to the hardware implementation of neural networks today, broadly speaking.

This methodology of reducing pattern recognition tasks to simple networks where the output is connected to the input by mathematical operations was termed as *Connectionism*. Initially, the promise behind such an algorithm was huge, and scientists and media alike publicized that general artificial intelligence was just within reach; however, before the 70s, this came to an end. A book called Perceptrons was published in the year 1969, highlighting their severe limitations and criticizing the hyperbolic predictions associated with it [36]. This book What followed was almost two decades worth of disinterest, lack of funding, and research in AI, which is referred to as a period of AI winter [37].

In the 1980s, several developments aroused the interest in AI research anew. In contrast with Connectionism, another AI philosophy relied on symbolic reasoning, such as using decision trees with if-else branchings based on some logic to reproduce intelligence. This gave birth to *expert systems*, a class of computer systems that could emulate the decision-making process in very specific domains [38]. The cause of connectionism was also revived by the physicist John Hopfield in 1982, who proposed Hopfield networks which were a novel, efficient, and bio-realistic implementation of neural networks [39]. From the theoretical side, George Cybenko proved the universal approximation theorem, which stated that an artificial neural network (ANN) with a single hidden layer is able to approximate any continuous function for inputs within a specific range of values [40]. A hidden layer in a neural network refers to an additional set of neurons between the inputs and outputs, which aims to learn intermediate values, simplifying the classification task in the successive layer. The deep neural network is a network that has one or more such hidden layers of neurons. David Rumelhart and Geoffrey Hinton

introduced the method of backpropagation as a method to learn the parameter values of such multi-layered networks [41].

Nevertheless, the general interest in ANNs was also short-lived, as, by the beginning of the 90s, many of the new developments failed to offer practical solutions. The expert systems couldn't generalize to other tasks, and it was very hard to manipulate or improve systems based on symbolic computations. However, other machine learning methods like the support vector machine (SVM) or the K-nearest neighbors were extensively used during this time. After this, it would take about two decades for the resurgence of ANNs to take place.

1.2.2 Renaissance of AI: the deep learning revolution

The following decades witnessed the sheer dominance of computing power in the context of AI applied to domains where human expertise was considered superior. Riding the wave of Moore's law, computers became powerful in the count of memory as well as processing speed, and a direct result of this could be seen in the defeat of the famous Gary Kasparov to the IBM computer called Deep Blue in a game of chess in the 1997 [42]. Another noteworthy accomplishment of AI during this period included the proof of Robin's conjecture in theoretical computer science by the AI automatic theorem prover known as EQuational Prover in the very same year [43]. In 2005 and 2007, significant demonstrations were made by autonomous vehicles in the DARPA Grand Challenge and DARPA Urban Challenge, respectively [44, 45].

However, these achievements relied on machine learning techniques that were not based on deep learning. The first major breakthrough that ushered in the era of deep learning was when Krizhevsky et al. won the large-scale ImageNet competition in 2012 by a big margin using a Graphics Processing Unit (GPU)-accelerated Convolutional Neural Network (CNN) [46]. What followed was a surge in interest and funding in deep learning algorithms. In 2019, Yann LeCun, Geoffrey Hinton, and Yoshua Bengio were awarded the Turing Award, the highest award in computer sciences. This resurgence could be attributed to three major aspects

- **Big data:** Since the early 2000s, the ubiquity of the internet, the spread of usage of multimedia and social media, along with the advent of new technologies like the Internet of Things, there has been an abundance of data available [47]. This wealth of data, known as Big data, supplied deep learning with information that it could leverage effectively because, as opposed to other machine learning methods, the performance of deep learning scales with the amount of data fed.

- **Development of algorithms:** Despite the fact that most of the building blocks of the modern deep learning landscape had been around for some time, access to faster computers with bigger memory has facilitated new implementations or the invention of altogether new algorithms. Deep CNNs, Recurrent Neural Networks (RNN), Long short-term memory (LSTM), Transformers, Generative Adversarial Networks (GAN), and Graph Neural Networks (GNN) are some examples of such algorithms which are directed toward

different applications [48–53].

Other than this, different aspects in training were invented which were crucial in the good performance of such networks like the implementation of Batch Normalizations, the Adam optimizer, the cosine annealing learning rate scheduler scheme, etc. [54–56].

- **Development of memory and compute:** Many of the algorithms listed above date back to the late 1990s, when they failed to perform owing to the mismatch between the required compute/memory and the existing standards. By the early 2010s, computers had grown powerful enough to showcase the effectiveness of these algorithms. A huge part of the computation in neural networks is matrix multiplication. Around the same time, a specialized device emerged that was able to do this specifically very fast in a massively parallel fashion. This device, the Graphics Processing Unit, remains one of the key hardware necessary to perform state-of-the-art deep learning [57]. Presently, more specialized hardware such as the Tensor Processing Unit (TPU) and other accelerators are being developed [58].

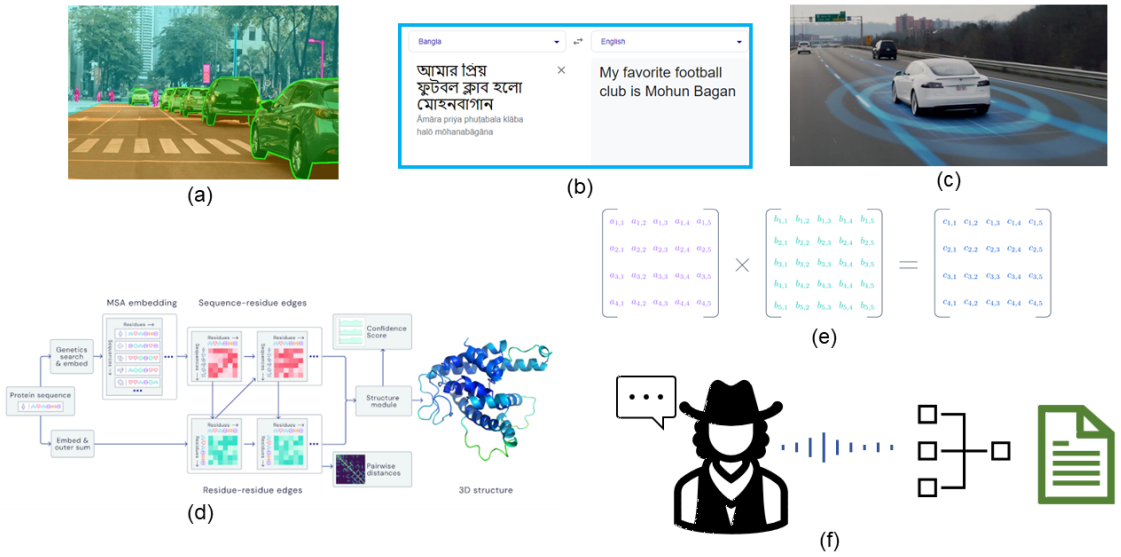


Figure 1.5: A collage of applications of DL today. (a) Image segmentation is a computer vision task where the network learns to separate different components in an image semantically. (b) Google Translate transcribes one language into another using natural language processing. (c) Tesla's autopilot system amounts to international level 2 in terms of vehicle automation. (d) AlphaFold 2 algorithm predicts the 3D crystal structure from the input amino acid sequence. (e) The AlphaTensor algorithm found an efficient way to multiply two matrices that have dimensions larger than 2×2. (f) Speech recognition task: the sound signal is converted to text.

Today, deep learning can be found in a plethora of applications, as shown in fig. 1.5. At

the time of writing this thesis, large language models have caused a significant buzz in the AI community, particularly with the recent release of the GPT-4 model, which has shown some indications of potential artificial general intelligence. [2]. Computer vision or image recognition has been at the forefront of this revolution. At this point, it is considered a practically solved problem with the recent neural networks surpassing human-level performance for numerous tasks [59]. Fig. 1.5 (a) shows an example of computer vision: image segmentation, where the system can correctly identify and separate objects semantically in an image from the camera attached to a car. Natural language processing, specifically machine translation, has experienced huge success with websites like Google Translate (fig. 1.5 (b)) and DeepL providing almost human-level performance. Autonomous vehicles (fig. 1.5 (c)) are also seen as another domain where DL holds a lot of promise, with companies like Tesla and Ford investing massively in this research. In recent years, DL applications have pervaded other science domains as well. The protein folding problem is a 50-year-old challenge in structural biology, where the task is to predict the correct 3D geometry of a protein given only its amino acid sequence. In 2020, the AlphaFold 2 algorithm (fig. 1.5 (d)) by DeepMind achieved a great level of accuracy at the CASP 14 competition in predicting the correct crystal structure of proteins [60]. Another algorithm from DeepMind, called AlphaTensor, has discovered novel mathematical algorithms that can efficiently implement matrix multiplication in more than two dimensions (fig. 1.5 (e)) [61]. Speech recognition is an application that exemplifies the end-to-end nature of deep neural networks (fig. 1.5 (f)). In the past, many conventional machine learning techniques relied on the hand-crafting of features to perform pattern recognition tasks. In contrast, neural networks can be used as end-to-end systems where the features are automatically learned, and the mapping is done directly from one type of data to another. For example, in image recognition, it learns the mapping between the image pixel values and the labels, or in speech recognition, the mapping between the sound signal and the text output. Next, we would venture into the details of such deep neural networks in the context of supervised learning, which is able to learn a special class of pattern recognition tasks.

1.2.3 Supervised learning

Learning algorithms can be classified in various ways in terms of the task they do, how the data is presented, and such. A fundamental classification divides them into three categories, with a glimpse of a fourth one appearing on the horizon.

This classification is based on the interaction between the system, the data available, and the kind of task it is trying to achieve. In supervised learning, the data consists of examples with labels; in other words, a single data point consists of an input and output pair, and the system has to learn a representation that maps the input (called features) to the output. Unsupervised learning is the learning paradigm where the data lacks such labels, and the system has to learn patterns underlying the data. The third categorization, Reinforcement learning, has a setup where the system can interact with its environment through actions. Here, learning takes place

through the system, getting rewarded for actions that contribute positively to the achievement of its goals or by getting punished for the opposite. In this thesis, we are going to look at only supervised learning tasks, specifically the ones related to image classification.

Self-supervised learning is emerging as a new type of learning that uses unlabeled data to train an algorithm to learn useful representations of the data. This approach is becoming increasingly popular in the field of deep learning because it can help to reduce the amount of labeled data required for training, which can be time-consuming and expensive to obtain. In self-supervised learning, the algorithm is trained to predict a missing piece of information in the data, which is typically achieved by masking part of the input and then asking the model to predict what the missing piece should be. This approach can be used to train models for a wide range of tasks, including image classification, object detection, and natural language processing [62, 63].

1.2.3.1 The deep neural network

In an image classification task, we have *training data* that contains the pixel values of all the images and their corresponding labels. Let the pixels for the n^{th} image be denoted by x_i^ns, and their corresponding labels (or targets) by t^n. The task boils down to finding an appropriate network that learns how to find the correct label for a given image.

The whole operation of a neural network happens in two phases: the feed-forward propagation phase, also called inference, and the backward propagation phase, where the actual training takes place.

A neural network has two major components, as described earlier: the neural activations or neurons and the synaptic weights or simply weights, which are the parameters of the network. In a feed-forward, fully connected architecture, all the neurons of a layer are connected to all the neurons of the next layer by the synaptic weights w as depicted in fig. 1.6. The weights and architecture are collectively referred to as the *model*. The different layers are supposed to learn features at various levels of abstraction, and the potential to utilize more layers gives a neural network an edge over the perceptron, which is nothing but a single-layered neural network.

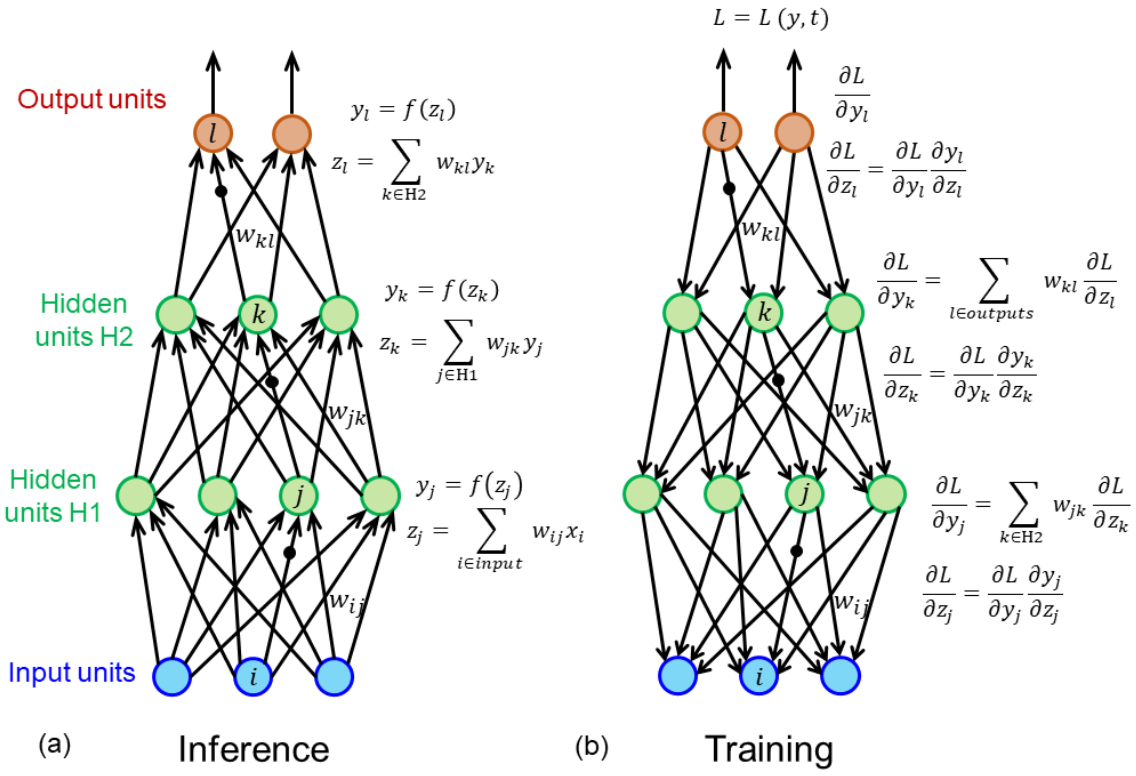


Figure 1.6: Anatomy of a fully-connected neural network. (a) The inference or the feedforward part of the network shows how the neuronal values are calculated, starting from the input in the very first layer to the final output prediction. In the neural network shown, there are two hidden layers. The preactivation zs are calculated by the linear combination of the inputs of that layer, weighted by the synaptic weights, and on that, the activation function f is applied to get the neuronal activation values ys. (b) Training or backward propagation phase is shown for the same network. Here, the reversed direction of the arrows represents the backward direction of propagation of the error signal. The chain rule is used to calculate the gradients for the preactivations and activations, which are then used to calculate the weight updates.

Forward propagation or inference

In this step, shown in fig. 1.6 (a) first, the pre-activation from one layer to another is calculated by taking the linear combination of the inputs of that layer x_i weighted by the synaptic parameters w_{ij}. This pre-activation value z_j is then passed through a non-linear function f called the *activation function* to yield the output y_j.

$$z_j = \sum_i w_{ij} x_i \tag{1.1}$$

$$y_j = f(z_j) \tag{1.2}$$

Here, the x_0 term is always set to 1, and the w_{0j} term is called the bias. The choice of the non-linear function is also a design choice; typically, the ReLU activation function is used for it. Other choices could be the hyperbolic tangent or the sigmoid function. In the output layer, a softmax function is typically used. Thus,

$$f(x) = \text{ReLU}(x) = \max(0, x) \tag{1.3}$$

The application of the batch normalization process makes neural networks learn faster and more stable. In batch normalization, the neuronal pre-activations are normalized over a batch of values [54]. In this context, a batch, or more specifically, a minibatch, is a small part of the whole training data. Because of a dataset's huge size, we only feed a part of the data, called the batch, to our network in a single step. The batch normalization bounds the value of the pre-activations to a smaller range and, in turn, makes the loss function landscape smoother, which facilitates learning [64].

Batch normalization

Let a single batch of pre-activations (z_i) have m data points, and let the mean and standard deviations of this batch be represented by $\mu_B(\frac{1}{m}\sum_{i=1}^{m} z_j)$ and $\sigma_B\,(\frac{1}{m-1}\sum_{i=1}^{m}(z_j-\mu_B)^2)$. Then the batch normalized values are given by

$$z_{BNj} = \frac{z_j - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{1.4}$$

Here, ϵ is a small positive term to maintain numerical stability.

The training process occurs in an iterative manner, where the whole dataset is shown to the network in small batches, and when it has seen the entirety once, we say it has gone through a single *epoch*. One of the most important metrics for a classification task is the accuracy of prediction. This is usually represented as a percentage of the number of correct predictions made. Since a neural network has a huge number of parameters that it can learn, it is very prone to *overfit* to that data, which means that instead of learning the essential features of the data, it almost 'memorizes' the details of the whole training dataset. To diagnose this, we preserve a part of the data called the *testing dataset* or *test dataset* and do not present it during the training phase when we only present to it the *training dataset*. If a network has learned the features well, it will perform similarly on the seen and the unseen datasets. All the accuracies reported in this thesis are the test dataset accuracy unless otherwise mentioned. It is also best practice to have a *validation dataset* which is used for hyperparameter tuning of the neural network.

Initialization of our model, that is, setting the initial values of the weights, is also an important aspect. Setting them too high can lead to exploding gradients, and too low can lead to vanishing gradients, both of which are detrimental to learning. The initial values of the weights are usually set randomly using a normal distribution with zero mean and a variance proportional to the number of neurons they are connected to [65, 66].

With a random initialization of the weights, the network produces random predictions in the form of outputs in the first iteration. To fix this, it is necessary to define a metric that quantifies how bad this prediction is, and this metric is called the loss or objective function L. This measures how far a prediction is from the true target label, and it is named so because the objective of the learning is to minimize the value of this function by tuning the values of the weights. The decrease in the loss function, or conversely, the increase in the test accuracy, means that our network is learning.

In the training phase, this is precisely what we do: we use the gradient descent algorithm to change the values of the parameters in a way that attempts to find the global minimum of our loss function [67]. Like in a univariate minimization problem, we take the derivative of the function with respect to the variable; here also, we take the gradient of the loss function

with respect to the synaptic weights. Then we descend or change our weights to decrease the value of the function. Our network is composed of a different set of values linked with each other by functional mapping, and this gradient has to be propagated *backward* from the loss function to all the weights of the network. The backpropagation algorithm achieves this by the repeated application of the chain rule from differential calculus [68]. It is to be noted here that this is only possible because the functions and linear transforms that we use are continuous, so a well-defined derivative exists for all points.

The choice for the loss function also impacts our training, and it is typically task-dependent. For regression tasks, where the output is not a single label but a continuous real value, the Mean Square Error (MSE) loss is used, which takes the square of the difference between the actual output and the expected output, averaged over the whole batch. In our classification task, the cross entropy loss is used, which, for a single image input for C output classes, has the following form

$$L(y,t) = -\sum_{i=1}^{C} t_i \log(y_i) \tag{1.5}$$

This specific form of the loss function is related to the way in which the outputs are represented in a classification task. Unlike a regression task, where the system has to predict a real number, a classification task involves predicting the output class, which in themselves are not numerical quantities, but just semantic categories. The *one-hot* encoding is typically used for a classification task where the output is represented by a C-dimensional vector whose all entries are zero except for the index of the class to which the output belongs, which is set to unity. Equation 1.5 is designed to punish the network by producing a high value when the prediction for an index is 0 when the expected output is 1 or vice-versa.

The derivative of this function with respect to the parameters of the network is what comprises the training signal, known as the gradient.

Backward propagation or training

Let us consider the neural network shown in fig. 1.6. In the backward propagation phase, we start calculating the gradients from the final layer. The update for the weight w_{kl} is proportional to $\frac{\partial L}{\partial w_{kl}}$, and the sign is negative since we intend to minimize the loss function. The proportionality constant α is called the learning rate since this determines the rate or speed at which the network learns. Thus, the equation for the update is

$$w_{kl} := w_{kl} - \alpha \frac{\partial L}{\partial w_{kl}}. \tag{1.6}$$

The derivative term in this equation cannot be directly calculated, as L is calculated based on the target t_l and output y_l. This output y_l, in turn, depends on z_l, the linear combination of the weights w_{kl}, and inputs of this layer y_k. The chain rule is used to calculate this as

$$\frac{\partial L}{\partial w_{kl}} = \frac{\partial L}{\partial z_l}\frac{\partial z_l}{\partial w_{kl}} = \frac{\partial L}{\partial z_l} y_k = \frac{\partial L}{\partial y_l}\frac{\partial y_l}{\partial z_l} y_k = -\frac{t_l}{y_l} f'(z_l) y_k, \tag{1.7}$$

using equ. 1.1, equ. 1.2 and equ. 1.5.

Next, we move on to the previous layer to determine the updates for the weights w_{jk}. Unlike the final layer, the gradients from the previous layers also contribute downstream, and hence we need to take a sum over all the contributions. Thus,

$$\frac{\partial L}{\partial w_{jk}} = \sum_k \frac{\partial L}{\partial z_k}\frac{\partial z_k}{\partial w_{jk}} = \sum_k \frac{\partial L}{\partial z_k} y_j = \sum_k \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial z_k} y_j = \sum_k \frac{\partial L}{\partial y_k} f'(z_k) y_j = \sum_k (\sum_l w_{kl}\frac{\partial L}{\partial z_l}) f'(z_k) y_j \tag{1.8}$$

This way, the updates to the weights are calculated for each layer of the network until the very first layer.

The backpropagation algorithm is an effective way of learning the parameters of our network as it scales well to the size of a network and where other optimization techniques fail. Although with this method, convergence to the global minimum is not guaranteed, with some modifications, this algorithm can train a network considerably well. The *stochastic gradient descent* is the simplest way of implementing this. Here, a small subset of the training data, selected randomly, enables the descent of our network along the gradient computed on it. The Adam optimizer improves on this by considering the gradients' history, which makes learning faster and better by effectively reducing noise [69].

The equations described above showcased the backpropagation in a fully connected network setting, but it works similarly with any differential operation. In the case of convolutional neural networks, the multiply and sum operation is replaced by convolutions where the role of the weights is served by *kernels* or *filters*. Throughout this thesis, we used the PyTorch deep

learning framework for our neural network simulations [70]. The advantage of using deep learning frameworks is that they come equipped with specialized modules that can perform automatic differentiations when the structure of the network is specified. So, we need to define the forward propagation part, technically called the computational graph, and the package automatically can do the backward propagation step.

However, this training algorithm is not without some demerits. Firstly, it is highly non-bio-plausible; the connectivity in the nervous system doesn't suggest the feasibility of such an algorithm. In the brain, the update signals are local in nature, and the neural circuitry has no simple provision for this signal to propagate backward [71]. Additionally, in backprop, the differentiability of the neuronal values, the transport of exact error signals, and the usage of the same weights for the forward and backward parts seem to go against biological feasibility. There have been alternative algorithms proposed to replace backprop, like Equilibrium propagation, Feedback alignment, and the more recent Forward-forward. Still, they are yet to convincingly exhibit the performance or efficiency of backpropagation [72–74].

Secondly, it can be noted from equ. 1.7 and equ. 1.8 that the update equations all involve the weight values, the derivative of the activation function evaluated at the preactivation, the activations, and the gradients from a subsequent layer. In software, these are just values stored in memory, but from a hardware point of view, we need to store these values, access them, and perform computations on them. The architectural overhead for achieving this is extremely high, to the point of being impractical. This difficulty scales with the size of the neural network, as larger and more complex neural networks would hinder the hardware implementation even further.

1.3 Neuromorphic computing

1.3.1 Memory requirements of deep learning

The success of deep learning can be attributed largely to the availability of an astronomical amount of data. This, in turn, enables the usage of complex models with a large number of parameters that almost always leads to better performance in terms of accuracy and other metrics. However, this comes at a cost: the memory requirements to store the model and the energy requirements to perform the operations, especially during the training phase, scale poorly with the model size.

Since the performance of a neural network scales directly with its size and complexity, we see an ever-increasing trend in terms of the number of parameters, and the total number of floating-point operations done, the rate of which is measured by Flop/s (Floating-point operations per second). In fig. 1.7 (a), we see this very clearly in the plot for the number of parameters count for the deep learning models in the latter half of the 2010s. The blue and red points show the number of parameters for models designed for computer vision and natural language pro-

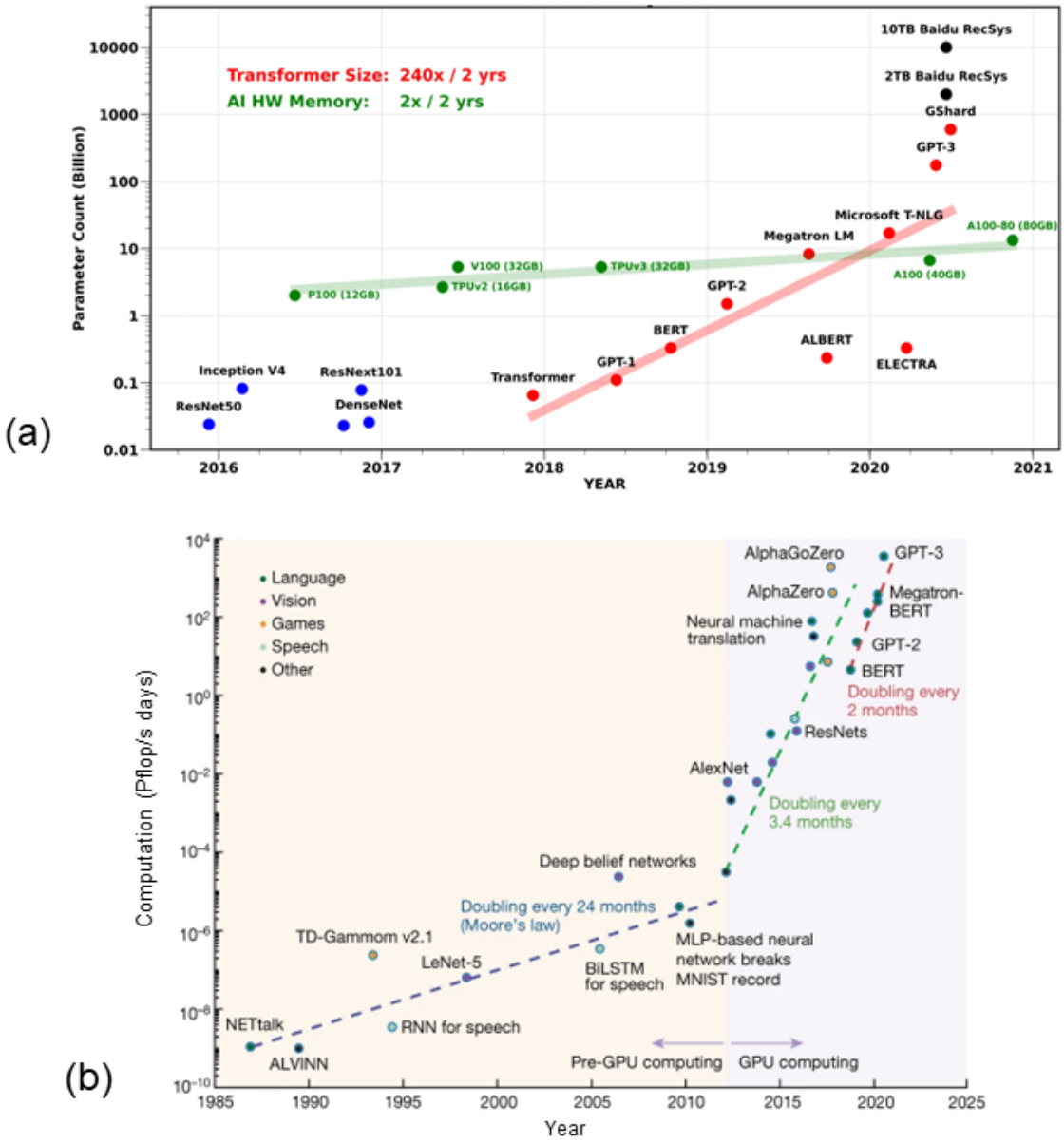


Figure 1.7: Trend in memory and computational performance for deep learning models. (a) The evolution of the total number of parameters that a model needs over time. The plot shows the count for the state-of-the-art computer vision (blue points), natural language procession (red points), recommender systems (black points), as well as the maximum memory capacity for AI-hardware (green points) (Adapted from [75]). (b) The same trend for the computational performance, measured in Flop/s days for vision, language, speech, and game models. The computation in the y-axis measures the computational performance as well as the total quantity of compute needed (in terms of days). Two different slopes can be seen based on the usage of GPUs, where the previous compute doubled every 24 months, and now it is doubling every two months (Adapted from [76]).

cessing. The NLP models are all based on the transformer architecture, and the number of parameters in such models is increasing by a staggering rate of 240 times in 2 years. In contrast, we see from the green points that the memory capacity of hardware dedicated to AI, such as the Nvidia Tesla V100 GPU or the Tensor Processing Units (TPUs), is increasing at a much lower rate of 2 times in 2 years that is in accordance with Moore's law [75].

As we examine more recent advancements like GPT-4, Google PaLM, and larger recommender system models, it becomes clear that this trend is diverging even more, indicating that we are quickly approaching a saturation point with conventional memory. The scenario for computing for deep learning resonates similarly, as we can see from fig. 1.7 (b). In the 'pre-GPU era,' the required number of operations was doubling every two years, in sync with Moore's law, but the usage of GPUs has led to the capability of using bigger and more complex models, and the Flop/s days are doubling every two months for very recent models [76]. The PFlop/s days measure not only the computational performance but also how long the computation runs, therefore measuring the quantity of computation as well. Thus, the gap between the existing hardware capabilities and the memory and compute requirements is growing bigger by the day.

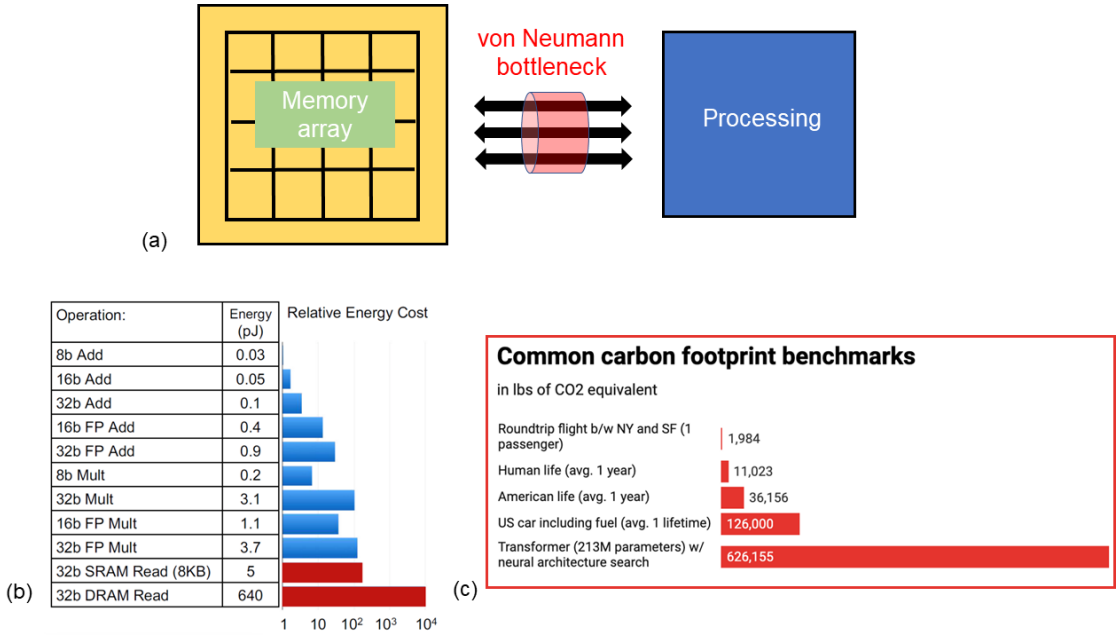


Figure 1.8: Energy consumption and its consequences. (a) In conventional computing architectures, the memory storage unit and arithmetic-logic processing unit are physically separated, and information needs to be constantly shuttled through them via a bus. This imposes a limitation in terms of computing efficiency and is called the von Neumann bottleneck. (b) The relative energy costs of single operations (add, multiply) for different precisions and data access from SRAM and DRAM [77]. (c) The relative carbon footprints of neural networks and other major sources of CO_2 emission. Training a transformer model with 213 million parameters has a carbon footprint that is about six times that of an average lifetime of a car in the United States.

This avalanche of memory and compute needs is not only an issue for the stagnation in development; there is a massive cost to this as well, with consequences for our environment. A major part of the energy consumed for a neural network to function is in its training phase. The process of finding the optimal set of parameters can take a long time, even with multiple GPUs, and the process of hyperparameter tuning can make this even longer. From fig. 1.8, we note the carbon footprint of a modern deep learning model (in this case, a Transformer with neural architecture search), and comparing it with that of other major sources of CO_2 emission, we observe the orders of magnitude difference [78].

To understand where the majority of energy consumption is, we take a closer look at the relative energy costs for different operations relevant to neural network calculations. The operations are dominated by multiplications and additions, but, from fig. 1.8 (b), we find that the energies for these are substantially lower than what is needed for accessing or reading the data from the cache (SRAM) or from memory (DRAM) [77].

This difference in energy is attributed to the manner in which information is accessed on a modern computer. In [12], John von Neumann outlined the blueprint on which modern computers are based; the memory and processing units are separated in space, and a bus shuttles data to-and-from between them. This transport of data causes a bottleneck in terms of efficiency in compute and is termed *von Neumann bottleneck* (fig. 1.8 (a)). In a single iteration of training of neural networks, the parameters and activations are read, and using those values; the updates are calculated and applied to the parameters and then stored. Hence, a lot of back-and-forth information is happening during this process, leading to massive energy expenditure.

1.3.2 Inspiration from the brain

On the other hand, the original inspiration for neural networks, the human brain excels at energy efficiency. It consumes a power of 20 W to perform all the tasks a human being can do, which is similar to that of a light bulb [79]. Traditionally the modern computer has been developed for specializing in tasks like fast, very accurate calculations, whereas the brain excels at computer vision, speech, language processing, and other pattern recognition tasks. These two domains of tasks are, in essence, very different from each other, and from this point of view, it comes as no surprise that they two function fundamentally differently.

The differences between the human brain and the modern computer can be summarized as follows.

- **Architecture:** The neurons in the brain serve as the processing units that transform incoming signals, while the synapses between those neurons are the analogs for the memory elements. Since they are co-located, the need to shuttle information between the processing units and memory elements is eliminated. On the other hand, in the von Neumann architecture of computers, the memory component is physically isolated from the

arithmetic/logic unit.

- **Information signal:** In the nervous system, the information is propagated in the form of voltage pulses called the *action potential*. The successive opening and closing of ionic channels modulate the cell membrane voltage that is responsible for the transport of the signal through a neuron. Due to the ionic origin of this process, typical timescales are of the order of milliseconds. In the case of computers, the transport of data occurs through the flow of electrical current, and hence, all operations are much faster in the order of μs or ns. Also, the action potential is highly sparse in time compared to how electrical signals are transmitted in computers.

- **Structural organization:** Human brain has about 10^{11} units of neurons, and each of them is connected to about 10,000 other neurons, on average, making the count for the number of synapses to be of the order of 10^{15}. The largest neural networks of today have a parameter count of less than 10^{12}, but the key difference lies in their organization. The networks of neurons self-organize into patterns that are dynamic in nature. Formation of new memories and forgetting is related to the strengthening and weakening of synaptic connections [80].

- **Precision:** Because of the substrate and nature of the processing, the brain is inherently noisy, and the information is propagated at low precision, which is very different from the highly accurate precision of computers.

In 1981, Richard Feynman, Carver Mead, and John Hopfield taught a course at Caltech called the "Physics of Computation" [81]. While John Hopfield went on to develop the Hopfield networks, and Feynman contributed to the theoretical development of quantum computers, Carver Mead, one of the pioneers of VLSI (Very Large Scale Integrated) circuits, initiated the field of neuromorphic engineering [39, 82, 83].

The field of Neuromorphic computing or engineering aims to mimic the nervous system (hence the term 'neuro'morphic) for performing efficient computations. At the outset, the goal was twofold: to develop a biological system in silicon that would enable us to study the brain's biology and to advance the development of specialized, energy-efficient computers. Misha Mahowald, a student of Carver Mead, developed the Silicon retina, which was an electronic micro-power chip that could produce a binocular map of distance to objects in a visual field [84, 85]. Scientific and engineering endeavors have proceeded along this line in the form of emulating neurological components through silicon-based electronic circuitry. Many different types of neuromorphic silicon circuits have been developed that have the ability to integrate and fire like neurons and thus could be used for spiking neural networks [86]. Spiking neural networks are a class of neural networks where the input is encoded in the form of spikes (either their frequency or interval between spikes), and the neurons act typically as leaky integrate-and-fire elements [87].

Another direction of research has emerged that takes a more top-down approach to neuromorphic computing that follows biology at a more architectural level and integrates the logic and memory components. This non-von Neumann architecture falls under a specific type of computing called *in-memory computing* or *near-memory computing* because of the proximity of the memory and computing elements. This approach facilitates the hardware implementation of neural networks that can substantially reduce the energy inefficiency problems that are inherent in conventional software-based implementations.

1.4 In-memory computing with emerging memory technologies

1.4.1 Filling the gap in memory hierarchy

We are at a point in time when Moore's law is saturating; as transistors are scaled to lower dimensions, the problem of dissipation of heat and of growing leakage and variability hinder the performance of traditional CMOS [88]. These are significant obstacles in the development of hardware specialized for deep learning, in addition to the energy-related constraints mentioned earlier in this chapter.

The term in-memory computing is an umbrella term for numerous types of computations, including applications such as reservoir computing, combinatorial optimization, etc. [89]. In the context of this thesis, we are specifically looking into brain-inspired hardware implementation of non-spiking deep learning neural networks.

Broadly, there are two classes of memories that are associated with in-memory computing; charge-based and resistance-based. The charge-based memories are already discussed in section 1.1 and are the SRAM, DRAM, and Flash memories (shown in fig. 1.3). The physical process underlying a state of memory is related to the presence or absence of charges.

Let us look at some metrics relevant to computing in these charge-based memories:

Type of memory	Access speed (ns)	Area (F^2)	Volatility
SRAM (cache)	<5 ns	140	Volatile
DRAM	50 ns	20	Volatile
Flash	20 μs	4	Non-volatile
Hard drive	5 ms	<1	Non-volatile

In the above table, F denotes the minimum feature size of that type of memory, and F^2 represents the associated area of a single unit.

We note that there is a gap of at least 2 orders of magnitude between the DRAM and Flash memories, and this is where the resistance-based memories come into the picture. This emerging class of memories can bring the best of both worlds in terms of fast speed, low area over-

head, and non-volatility, all of which are essential requirements for the hardware implementation of deep learning.

1.4.2 Emerging memory technologies

Resistive memories, also called memristors (memory-resistors), are devices whose resistance state depends on the history of electrical current and are non-volatile in nature. The possibility of the existence of a fourth circuit element, the memristors (besides the resistor, capacitor, and inductor), was theoretically proposed by Leon Chua from symmetry-based arguments in 1971 [91]. According to his vision, the memristance connects the magnetic flux and the electrical charge and would have dimensions the same as the resistance. However, when actual physical systems started being identified as memristors, the involvement of the magnetic flux was not apparent. In [92], the authors demonstrated that this memristance arises in nanoscopic systems where the current-voltage characteristics show hysteresis.

The coming decades saw a rise in the research of such materials, and today there are four main types of such resistive memory technologies: Filamentary/interfacial/resistive switching materials (ReRAM or RRAM), Phase change materials (PCM), Spin-transfer torque magnetic random access memory (MRAM), and Ferroelectric field effect transistors (FeFET) based on Ferroelectric random access memory (FeRAM). Let us discuss these in more detail.

1.4.2.1 Resistive switching materials

These devices are typically metal-insulator-metal (MIM) heterostructures where the metal layers are called top and bottom electrodes (fig. 1.9 (a)), and the voltage pulse is applied between them. The insulator is, in many cases, a non-stoichiometric oxide (HfO_x, TiO_x, TaO_x). The non-stoichiometry is related to the absence of oxygen atoms, called oxygen vacancies, and these vacancies interact with the electric field in the dielectric to yield the memristive behavior. In a virgin state, the stack has a high resistance, and to produce such behavior, initially, it needs to be *formed*. This is done by applying a large electrical voltage that causes a soft electrical breakdown to form a filament made up of oxygen vacancies.

Upon application of a positive voltage to the top electrode, the defects undergo field-induced migration and diffusion to the lower electrode and start forming a conductive filament. And when this filament connects the top and bottom electrodes, a SET transition occurs, and the device goes to a low resistance state (LRS). If the polarity of the voltage is reversed, the opposite process starts happening; that is, the filament starts getting destroyed, and when this connection is broken, a RESET transition happens to switch the device from the LRS to a high resistance state (HRS). This is schematically shown in fig. 1.9 (a) and (b), where the first figure shows the formation of the conductive filament, and the second shows the current-voltage (I-V) characteristics. The I-V graph shows a pinched hysteresis loop, which is a signature of the filamentary switching phenomenon. Interfacial resistive switching has also been demon-

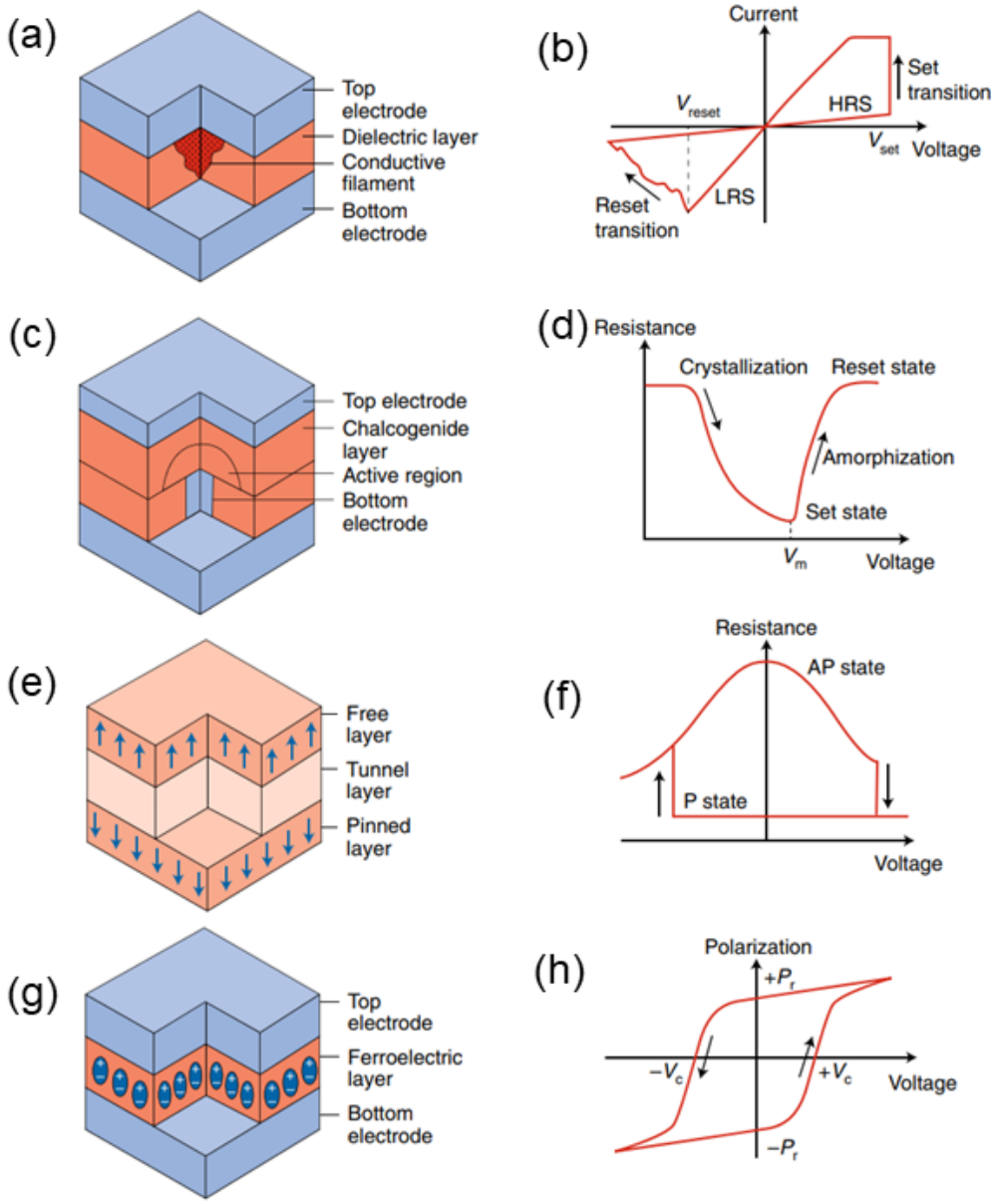


Figure 1.9: Zoo of emerging memory devices. (a, b) Illustration and current-voltage characteristic of a bipolar RRAM. The transition from HRS to LRS occurs with positive voltage due to the formation of a filament connecting the top and bottom electrodes. Resetting the device to HRS occurs with a negative voltage, indicating disconnection of the voltage-induced filament. (c, d) Structure and resistance change of a PCM device, where a voltage pulse causes a decrease in resistance through increased crystallization and an increase in resistance above the melting-point voltage (Vm) through increased amorphization. (e, f) MTJ and resistance-voltage characteristic of an STT-MRAM, showing low and high resistance for parallel (P) and antiparallel (AP) states, respectively, which can be achieved with positive and negative voltages. (g, h) FeRAM structure and polarization-voltage hysteresis showing permanent polarization of the ferroelectric layer caused by electrical dipole orientation. A voltage above the coercive voltage (V_C) results in positive remnant polarization (P_r), and the opposite for a negative voltage. (Adapted from [90]).

strated, where the vacancies get either attracted or repelled at the interface uniformly, leading to the different resistance states [93].

An advantage of such a class of materials is that the technology for the fabrication is quite mature, so it is easier and cheaper to produce. This technology is so developed at this point that commercial microcontroller units are available based on them [94]. Some drawbacks are that because the underlying physics relies on nanoionic mechanisms, the process can be noisy and unreliable. The issue of noise is even worse for analog memories where intermediate resistance states are utilized [95]. In the next chapter, we discuss and present our investigations on this matter. Other disadvantages include typically low endurance, the necessity of the initial forming step, and high inter-device variability.

1.4.2.2 Phase change materials

This type of material typically comprises a Chalcogenide (a compound with at least one Chalcogen element and a metal), the most widely used being $Ge_2Sb_2Te_5$ [96]. Applying a voltage causes local Joule heating in these materials, changing the physical state from amorphous to crystalline. Starting from a pristine state, which is amorphous, if a low voltage pulse (called the SET pulse) is applied, there is Joule heating-induced crystallization. On the application of high amplitude shorter voltage pulses (called the RESET pulse), the temperature in the material exceeds the melting point to cause local melting and consequently amorphization (fig. 1.9 (d)) [97]. In these materials, the crystalline state has much lower resistance owing to the large concentration of carriers, and the amorphous state has high resistance because of the Fermi-level pinning at mid-gap [98].

The heterostructure of the PCM cell is shown in fig. 1.9 (c), where the bottom electrode is connected to a tungsten plug with narrower dimensions to confine the current and heat, leading to a hemispherical shape of the molten or amorphous region. Since the mechanism for this class of materials depends largely on the contact area and current, it scales well in the sense that smaller devices consume less power [99]. PCM is more reliable than ReRAM; it is usually much less noisy and has a higher endurance as the LRS and HRS are linked to the crystalline and amorphous states of the materials, which are thermodynamically stable. It is also technologically mature, and ST Microelectronics has developed embedded memory cells with PCM integrated with 28 nm FDSOI (fully-depleted silicon on insulator) transistors that can be used for automotive microcontroller applications [100]. One of the biggest problems with this technology is that it suffers from resistance drift, which is the passive increase in resistance and the threshold voltage most prominently seen in the HRS at room temperature. It is caused by the thermally-activated structural relaxation process in which there are atom-level rearrangements in the amorphous phase [101]. Also, the requirement of high voltage for the RESET leads to high power consumption.

Although resistive switching mechanisms are very different for ReRAM and PCM, both excel at certain categories. The OFF/ON ratio, which is the ratio of the resistances in the HRS and

LRS, is usually high, allowing for few programming errors and multi-level switching. Both of them are considerably faster in switching time (~10 ns) and have better endurance than Flash memories [90].

1.4.2.3 Magnetic random access memory

The building block of an MRAM cell is a magnetic tunnel junction (MTJ), which is a spintronics-based heterostructure with a thin insulator layer sandwiched between two ferromagnetic metallic layers; the typical choices for the insulator are MgO and for the ferromagnet being CoFeB [102]. As shown in fig. 1.9 (e), the magnetization of one of the layers is structurally fixed, called the pinned layer or reference layer, while the other layer's magnetization is free to change upon programming, and is called the free layer. Thus, the two layers can have parallel (P) or antiparallel (AP) magnetizations that define the LRS and HRS, respectively, due to the tunneling magnetoresistance effect [103].

In the first generation of MRAMs, the orientation of the magnetization was switched by the application of a magnetic field. Still, it had many problems, including the need to apply a high current to generate the magnetic field. The spin transfer torque (STT) effect is used in the second generation of MRAMs (also called STT-MRAM), in which a spin-polarized current applies a torque on the magnetization of the free layer to change its orientation [104]. Fig. 1.9 (f) shows that if the layers are in the AP state, a positive voltage can flip it to the P state by the conservation of magnetic momentum, and the reverse polarity of voltage and current can achieve the opposite. This alternative to field-induced switching uses a much lower power and is more scalable.

MRAMs can exhibit a low switching time (~ns) or a high endurance (>10^{14}), both of which cannot be obtained simultanously for STT-MRAM and earlier technologies. Moreover, since the principal mechanism is based on tunneling, the resistance is exponentially dependent on the thickness of the barrier. Due to this dependence, it is crucial to achieving precision in the fabrication process, and even small variations can significantly impact the performance of the device. As a result, the fabrication of such devices is more difficult than other types of memories [105]. Thus, small differences in thickness can lead to substantial differences in performance. Some of the main drawbacks of this technology are the small ON/OFF ratio, the existence of only two resistance states, and the tradeoff between fast switching and reliability. Another fundamental problem is that the barrier height between the two states depends on the dimensions of the MTJ. So if we scale it to a limit where the barrier height is comparable to the thermal energy at room temperature (about 26 meV), the device would be stochastic. Nevertheless, this type of memory is also commercially available now in the form of both as standalone or as embedded memory [106]. The third generation of MRAM is based on the spin-orbit torque (SOT) mechanism for switching, in which the write current passes a heavy metal layer placed underneath the MTJ, and a torque induced by the spin-orbit coupling switches the state [107, 108]. Due to the fundamentally different mechanism for switching, SOT-MRAM is superior to STT-

MRAM in terms of write speed and can simultaneously achieve low switching speed and high endurance at the cost of increased area [109].

1.4.2.4 Ferroelectric random access memory

The FeRAM is structurally similar to a DRAM where the dielectric of the capacitor is replaced by a ferroelectric material such as PZT or doped HfO_2 [110]. The heterostructure is shown in fig. 1.9 (g), and we see in fig. 1.9 (h) that a voltage sweep can change the polarization of the material in a non-volatile manner. The polarization in a ferroelectric material is caused by the alignment of its atomic dipoles, which can be oriented in one of two directions. When an electric field is applied to the material, it causes a change in the alignment of the dipoles, which changes the polarization state. This change in polarization can be detected and used to store information. Therefore, the FeRAM is in itself not a resistance-based non-volatile memory. To read the two states as resistance values, a ferroelectric field effect transistor (FeFET) has to be used where the ferroelectric is sandwiched between the source-drain conduction region of the device and the gate electrode. The resistive switching effect is achieved by a change in the dielectric polarization which changes the channel resistance. Also, the perovskite-based FeRAMs have a very high endurance ($>10^{14}$) owing to the very minimal internal structural change during the switching process. Also, it requires much less power for operation compared to conventional memory technologies like DRAM and Flash. However, it is a relatively new type of technology and needs more research to overcome the following issues. There is significant inter-device variability originating from the polycrystalline ferroelectric films, which becomes worse with scaling, and the retention is reduced at higher temperatures [111].

1.5 Hardware-based neural networks

Due to the scalability, energy efficiency, speed, and endurance of these resistive technologies, they are ideal candidates to be used as the neural network synaptic parameters in the context of hardware implementation. Another common advantage of these four types of resistive memories is that they are compatible with the existing CMOS technology; they can be very easily integrated with the back end of line (BEOL) of transistors. In the fabrication process of an integrated circuit (IC), the transistors are first patterned on the semiconductor substrate. This part is known as the front end of line (FEOL). After that, the electronic components need to be connected with the wires of the wafer, and this happens in the next layer called BEOL, where several layers of metal (usually copper or aluminum) and the in-between insulators are deposited to form the stage contacts, interconnecting wires, and vias. The metal layers are named M1, M2, ... starting from the bottom, and the resistive memory device is usually fabricated between two of such layers. This integration of the memory heterostructure within the metal layers is shown in fig. 1.10 for the four types of memories discussed in the last section.

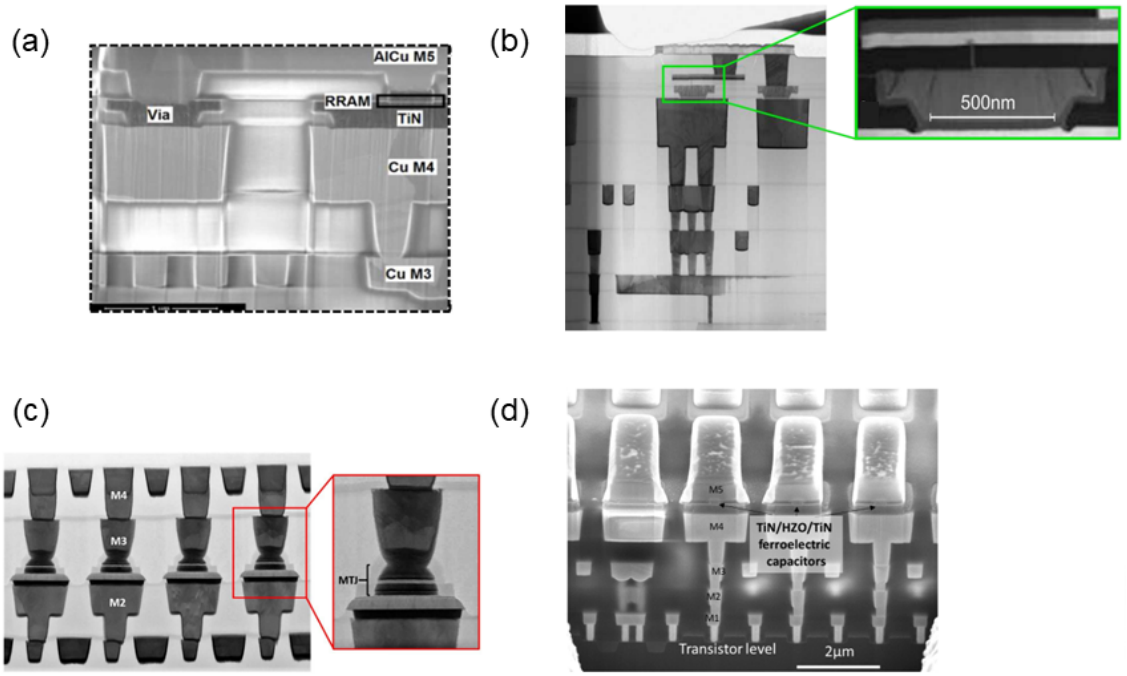


Figure 1.10: Back end of line (BEOL) integration of resistive switching memories for RRAM (a), PCM (b), MRAM (c), and FeRAM (d) (Adapted from [4, 112–114]).

1.5.1 Memory architectures

The memory devices need to be organized in the form of an array to store, read, and program memory states. One possibility is to integrate them in the form of a crossbar architecture, where the simplest arrangement (called the 1R configuration) consists of a single resistor in a memory cell arranged in a matrix. Two orthogonal lines overlap in this structure, and a memory device is present at each intersection. Thus, the two lines are connected to the two ends of each resistor and are called the bit line (BL) and the word line (WL), and because of the absence of any access transistor, cell size can be theoretically as small as possible, that is, 4 F^2.

However, a major issue with this architecture is the presence of *sneak paths*: to program a particular device, a voltage needs to be applied across its corresponding BL and WL. To do so, the WL for that device is set to the programming voltage (for the SET process) while the corresponding BL is grounded, and the voltage of half the programming voltage is applied to all the other lines. As a consequence, there are some cells, called half-selected cells, in the same row and column as the selected device, which receive half the programming voltage, and a current flows through these sneak paths, leading to wastage of energy [115].

Another type of architecture that can be used to mitigate this is the 1T1R (1 transistor 1 resistor) grid architecture, where each memory cell has co-located transistor and memory. As shown in fig. 1.11 (d), the non-volatile memory is integrated with a transistor (typically in the BEOL as shown in fig. 1.10) where the BL connects to one end of the resistor, the WL to the gate of the transistor, and the source line (SL) to the source side of the transistor. These transistors

are known as *access transistors* as turning them on or off leads to the access or selection of a particular memory cell. Fig. 1.11 (a), (b), and (c) show how the SET, RESET, and read are done for a bipolar switching resistive device. To access the selected cell (shown in the pink box) in the figure, a voltage is applied at the WL to turn on the access transistor in the selected cell. For the SET process, simultaneously, a voltage is applied to the BL of that cell (for RESET, the voltage is applied at the SL), and all the other lines are kept grounded. This way, during programming, the cells which are required to be written are only accessed. Similar to the programming, the read is done by applying a low voltage at the BL and measuring the output using a sense amplifier.

Even this architecture is not without its demerits, as switching memory devices requires a certain magnitude of current, and the dimensions of the connected transistor limit that. Thus, larger transistors are typically needed, which take up more area and are costlier to fabricate [18].

A more compact architecture has been proposed that includes a selector element connected in series with the memory called the 1S1R configuration (fig 1.11 (e)). The selector unit, in this case, is a type of diode that, by default, doesn't allow current to flow through it below a certain voltage difference. Thus, the previously half-selected cells would still have no sneak paths through them. A key challenge is that a conventional diode would only work for unipolar switching, typically only found in phase change memories [116], and newer technologies involving bipolar diodes need to be developed. Companies like Micron and Intel have made significant progress on this front, and they even have commercially available products employing this configuration [117, 118].

1.5.2 Neural network dedicated hardware

The training and inference of modern neural networks are typically done in data centers that employ conventional computers with multiple GPUs. Because the computations usually are not optimized for the specific type of calculations and information access, the energy consumptions can reach exceedingly high values [78]. There has been development in terms of hardware dedicated to deep learning applications. For example, Google has developed its very own ASIC (Application Specific Integrated Circuit) called Tensor Processing Unit (TPU). Tensors are a generalization of matrices in higher dimensions, and in deep learning, the data has more than two dimensions (e.g. in convolutional networks, the input has a dimension for height, width, channels, and batch). TPUs offer a more natural way of computing with such high-dimensional units and, although based on DRAMs, can provide a considerable amount of energy efficiency [58].

It is important to note that these factors are primarily taken into account in cloud computing, where performance quality takes precedence over other attributes such as power consumption and speed. Hence, these solutions may not be suitable for low-power embedded applications, where Microcontroller units (MCUs) are not powerful enough to train neural networks. Microcontroller units (MCUs) are purpose-specific computers with small memory and

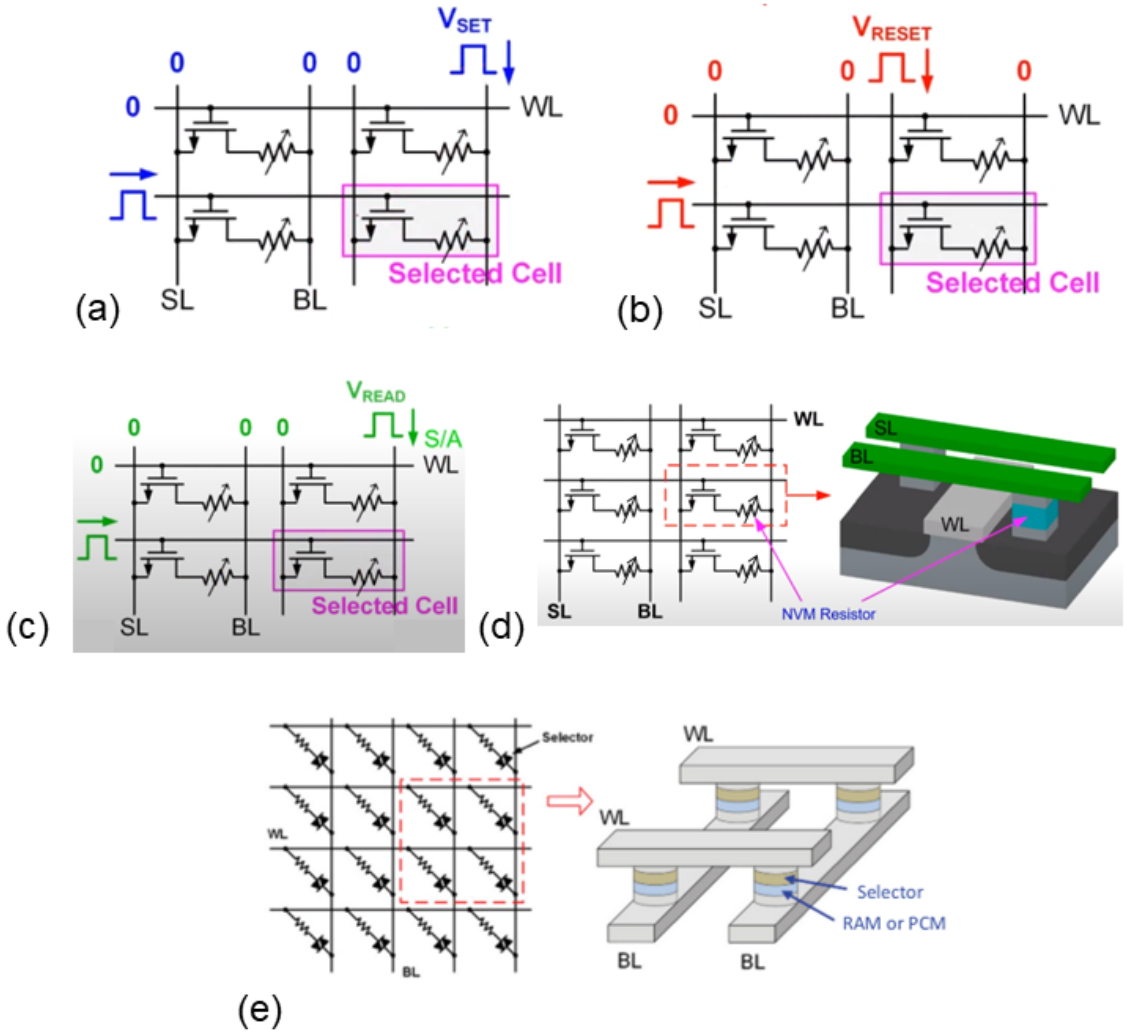


Figure 1.11: Memory architectures. (a) the SET, (b) the RESET, and the (c) Read operations in a 1T1R configuration. A particular cell is selected by applying voltages at the WL, the BL for SET (or SL for RESET), and the other lines grounded. The access transistor allows only the selected cell to be written to prohibit sneak paths. The read is done similarly to the SET with a much lower voltage and the output being read by a sense amplifier circuit. (d) The device-transistors integration is shown explicitly where the WL and SL are connected to the gate and the source of the transistor, respectively, and the BL to one end of the resistor. (e) The 1S1R configuration showing the series connection of the memristors with the selector element. (Adapted from [18])

are not powerful enough to train neural networks. However, they can support inference of some networks. In [119], an MCU was used to run a compressed network where SRAM and Flash memories were used to store the data and parameters respectively, and it could reach

more than 70% top-1 accuracy on the ImageNet dataset with a power budget of the order of a couple of Watts.

However, even such energy efficiency is not sufficient for extreme edge computing applications. In extreme edge computing, data processing, and analysis are performed on the edge devices with an ultra-low power budget. Such devices are located near the data source or the end-user, and this allows for faster processing and reduced latency, which is particularly important for applications that require real-time processing, like IoT devices. In this context, neural networks implemented using emerging memory devices in crossbar architectures show great promise for AI-based extreme edge computing applications, owing to their low-energy, low-area, fast, and in-memory computing-compatible nature.

Typically in this type of implementation, the memory devices are arranged in the form of an array together with access transistors. Such matrices of analog non-volatile memories are very suitable for implementing neural networks since, by virtue of Ohm's law and Kirchoff's law, they can naturally implement the multiply and accumulate operation of a single layer. This idea is illustrated in fig. 1.12: the input is applied to the WL of the crossbar in the form of voltages. At the intersection of the i^{th} WL and j^{th} BL, there is a resistive memory with a certain conductance G_{ij}, and by Ohm's law, the current through it simply given by $I = V_i G_{ij}$. At the BL, Kirchoff's current law accumulates the current from all the WLs as $I_j = \sum_i V_i G_{ij}$. In fig. 1.12 (a), the role played by the voltage V_i, conductance G_{ij}, and the output current I_j is analogous to the neural network layer shown in fig. 1.12 (b) in terms of the input x_i, synaptic weight W_{ij} and the preactivation z_j. The non-linear function needs to be applied using some other devices or circuitry. Another caveat of this approach is that a single conductance can only encode a positive value, whereas the weights of a neural network can have any sign. To circumvent this, a differential conductance pair of G^+_{ij} and G^-_{ij} is used to represent a single weight W_{ij}, and the final measured current is the difference between the current from the positive and negative arms (as shown in fig. 1.12 (c)), allowing for an effective negative contribution from the conductance pair unit.

There have been several demonstrations of this idea with different forms of novel non-volatile memories. ReRAMs with oxide-based resistive elements have been used to train perceptrons [120]. Orders of magnitude reduction in power consumption compared to conventional computers were reported on a face classification task using ReRAM [121]. Another study was able to implement a HfO_x-based neural network capable of in-situ training to learn a compact version of the MNIST dataset to get an accuracy close to the software baseline [122]. In another work, a hybrid training algorithm was utilized to implement a convolutional neural network for the MNIST task which reported an energy efficiency of more than two orders of magnitude while achieving software-comparable accuracies [123].

Phase change memories have been also explored for such realizations, although the number of scientific articles has been less in number. A mixed-precision in-memory approach was used to solve a set of 5,000 linear equations accurately using PCRAM [124]. Another work uti-

lized a mixed hardware-software implementation to achieve test accuracies for MNIST, MNIST-backrand, CIFAR-10, and CIFAR-100 datasets equivalent to software and showed that such realization could be done faster and with much less energy than GPUs [125]. Recently, a new approach has emerged for implementing in-memory neural networks. This approach utilizes a crossbar array of MRAM, with a resistance summation scheme employed for the accumulation process. This scheme addresses the issue of high energy consumption caused by low resistance of MRAMs during the MAC operation [126].

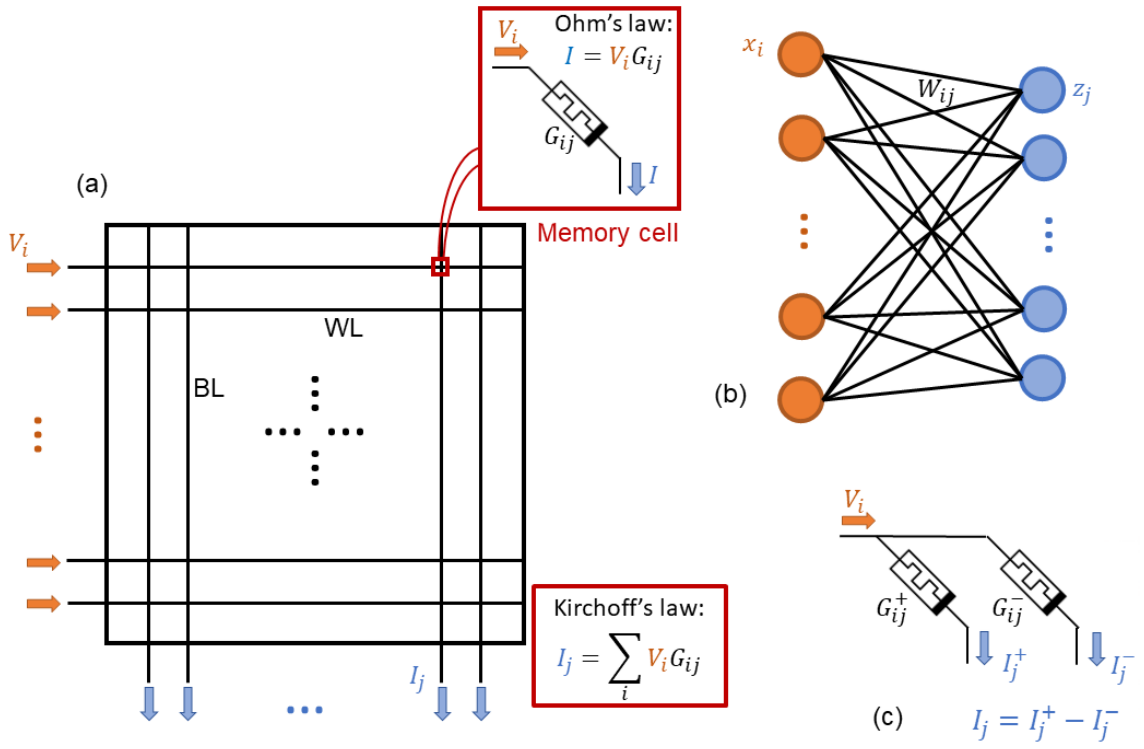


Figure 1.12: Mapping a neural network to a matrix of non-volatile analog memories. (a, b) If the inputs of the layer are applied as voltages at the WL, and the outputs are measured as currents in the BL, then the conductances of the memory cells play the role of the synaptic weights by virtue of Ohm's law and Kirchoff's current law. (c) The differential conductance pair is used to allow for both positive and negative values of the weight.

While the technologies employed in the aforementioned studies show great promise for enabling low-power and fast neural network implementation, it is worth noting that they were predominantly symmetric, substantially linear, and had limited variability. The robustness of these studies to different types of variabilities has not been illustrated convincingly. Since the matrix multiplication is done directly with analog weights, the noise in the memory can significantly impact the networks' performance. This is most prominent in filamentary switching memory devices as the variation of the resistance depends on nanoscopic physics, which is very sensitive to atomic fluctuations [127, 128]. With phase change memories, the resis-

tance drift phenomenon can potentially erase a programmed neural network over time [101]. Also, unlike in software, the conductances/resistances need to be updated by the application of voltage, and for these classes of materials, the I-V characteristics can be highly non-linear and asymmetric, making it more difficult to map networks reliably. The device-to-device variability of the memristors is another crucial impediment in this regard. The HRS tends to show a wide distribution of states, making it complicated to program devices reliably. Also, since the computations are based on the flow of current, the IR drops caused by the current flowing through the connecting wires can change the calculation of activation values.

It is imperative to mention at this point the difference between inference and training hardware. In inference hardware, the training is done offline, which means in software. The trained weight values are then transferred to hardware by programming the devices to the intended resistance values. This is suitable for applications where no further online training is needed after the deployment of the model. However, in scenarios where training has to occur on-chip, specialized training hardware needs to be used. The major difference in training hardware is that the updates need to be calculated and applied to the weights, apart from doing the feed-forward calculations for inference. The distinction between the two different types of hardware is done here to emphasize that they suffer from different aspects of imperfections, which shall be discussed in the next section.

1.6 Challenges in learning: imperfections in resistive memories

In this section, we will provide a detailed review of the imperfections that are inherent in emerging resistance-based memories. Moreover, we will analyze their significance in the context of the hardware implementation of neural networks. Firstly, we discuss non-linearity and asymmetry-related issues, which specifically pose difficulty to training hardware since updates to resistances are involved. The inference process is usually immune to this sort of imperfection. Then, we discuss intra and inter-device variabilities which plague both kinds of hardware.

1.6.1 Non-linearity and asymmetry

The resistive switching phenomenon is related to different types of physical changes in materials; the presence or absence of oxygen vacancy filaments in the case of ReRAM materials, the ordered or disordered state in a PCM, the relative magnetization of MTJs in MRAM, and the dielectric polarization in FeRAMs. All of these have vastly different physical origins, as are the mechanisms that facilitate conduction, or the lack thereof.

For the purpose of this thesis, we mainly focus on filamentary switching-based materials. In this class of materials, the switching occurs through the formation and dissolution of the oxygen vacancy-based conducting filament. However, the exact details of the underlying

mechanisms are not entirely elucidated. Particularly in the HRS, when the filament does not connect the top and bottom electrodes, the conduction happens via quantum mechanical tunneling [129]. This type of transport mechanism is highly non-linear in terms of voltage, which is apparent if we observe the current-voltage(I-V) characteristics of the oxide-based resistive memories. In fig. 1.13 (a), (b), and (c), the I-V plots for HfO_x, TiO_x, and TaO_x are shown respectively, and the non-linearity is quite evident in both the LRS and HRS.

The SET and RESET processes in such memristors occur through the filaments' formation and dissolution, which are separate processes. The SET transition is usually controlled by a compliance current which defines the resistance of the LRS and, internally, the thickness of the conducting filament. On the other hand, the RESET process happens by the destruction of this filament by the dissolution of oxygen vacancies, which has different kinetics than the SET. Thus, we see an asymmetry in the SET and RESET processes; they might happen at different positive and negative voltages (like in fig. 1.13 (c)). Also, the stability of the two states can be very different. The HRS is usually noisier than the LRS, and it is notoriously difficult to understand the physical origin of this noise. Even for PCM, the resistance drift effect is also more pronounced in the HRS (RESET state), as shown in fig. 1.13 (e).

These kinds of imperfections can have a significant impact on programming memory devices as synaptic parameters in a neural network. In software, the weight values are directly updated by ΔW, but a voltage needs to be applied to change the device resistance/conductances in hardware. And since the response of the resistance to the applied voltage is non-linear, the updates cannot be applied precisely in a simple manner. The asymmetry of the two states also makes learning or programming very difficult since the variation of resistance is not the same for positive and negative voltage pulses. This can be seen in fig. 1.13 (d), where the conductance is alternately increased (potentiated) and decreased (inhibited) by the application of positive and negative voltage pulses. The potentiation shows a much abrupt increase, then saturation, while the inhibition happens progressively.

1.6.2 Intra-device and inter-device variability

Variability is another type of imperfection that is commonly found in oxide-based memristors. It is chiefly of two types: intra-device and inter-device. The former is linked to the variations in a single device, whereas the latter concerns the variability among different devices.

The intra-device variability again can be of two types; it can be the noise in the resistance state when we are reading the current over a period of time, or it can be a cycle-to-cycle type of variability where if the device is subjected to the same voltage pattern (be it a voltage sweep or a SET), the output responses differ over the different cycles. Fig. 1.14 (a) and (b) show the read current when a voltage of 10 mV is applied as a function of time. Depending upon the number of defects and the impact of fluctuation of charge near the filament, different types of noises can be observed in such devices. Here, fig. 1.14 (a) shows pink noise, which has a power spectrum that varies inversely with the frequency, and fig. 1.14 (b) shows Random Telegraphic

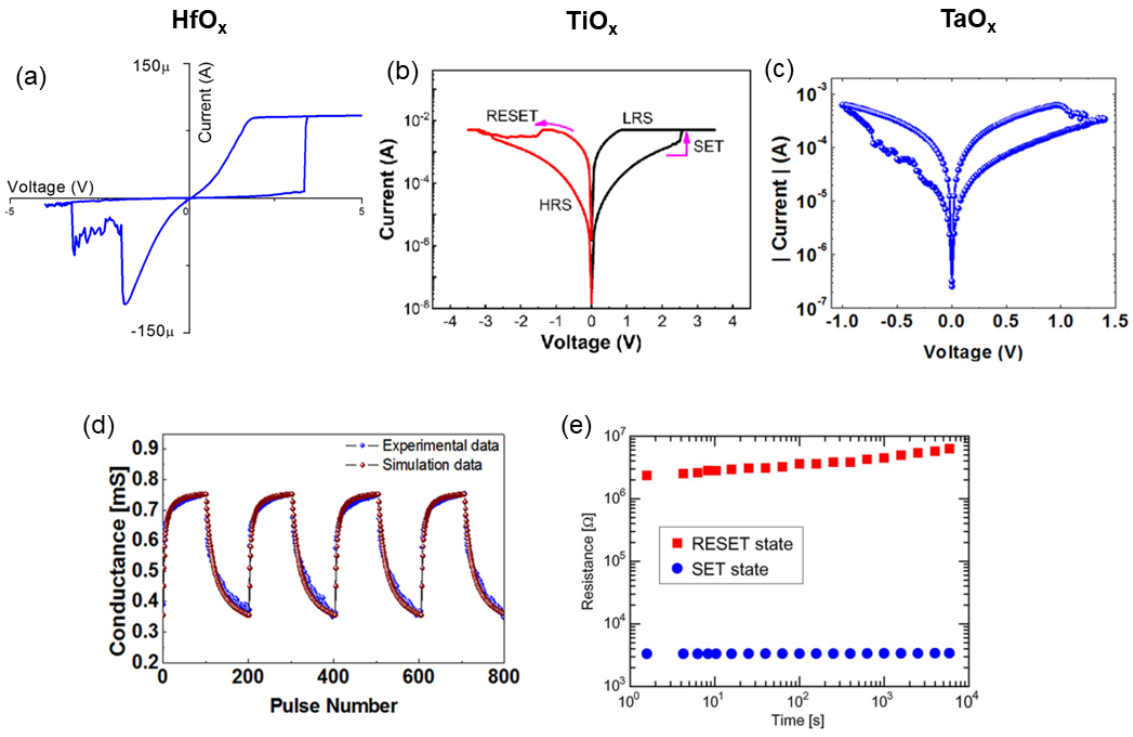


Figure 1.13: Non-linear and asymmetric behavior in resistive memories. (a)HfO_x-based, (b)TiO_x-based, and (c)TaO_x-based resistive memories' current-voltage characteristics showing non-linearity in both LRS and HRS and asymmetry in SET-RESET transitions [130–132]. (d) Potentiation and inhibition characterization with positive and negative voltage pulses exhibiting asymmetry of the two different transitions [132]. (e) The PCM resistance drift effect is more pronounced in the HRS than in the LRS [133].

Noise (RTN) [134], which are sudden jumps between two discreet resistance states.

Fig. 1.14 (c) shows that the voltage sweeps over a device do not always yield the same IV trace, and even the switching voltages can be very different in different cycles [135]. The compliance current (also called SET current or I_{SET}) is the fixed current to which a device is subjected in the SET operation, and this is the parameter that defines the LRS. In fig. 1.14 (d), we see that when a single device is SET to LRS with different SET currents, the device is not programmed to the same conductance value each time. Rather, the LRS has a Gaussian distribution whose mean and standard deviations are coupled and depend on I_{SET} [136].

The inter-device variability or the device-to-device variability refers to variation in-between different devices. A point to be emphasized here is that this variation is not because of fabrication-related variabilities and is due to the intrinsic nanoscopic nature of the switching mechanism. Fig. 1.14 (e) and (f) show the distribution of the conductance and resistance states for the LRS and HRS, respectively, over different devices. The earlier figure illustrates the Gaussian nature and SET current dependence of the LRS, which is similar to the cycle-to-cycle variation. On the

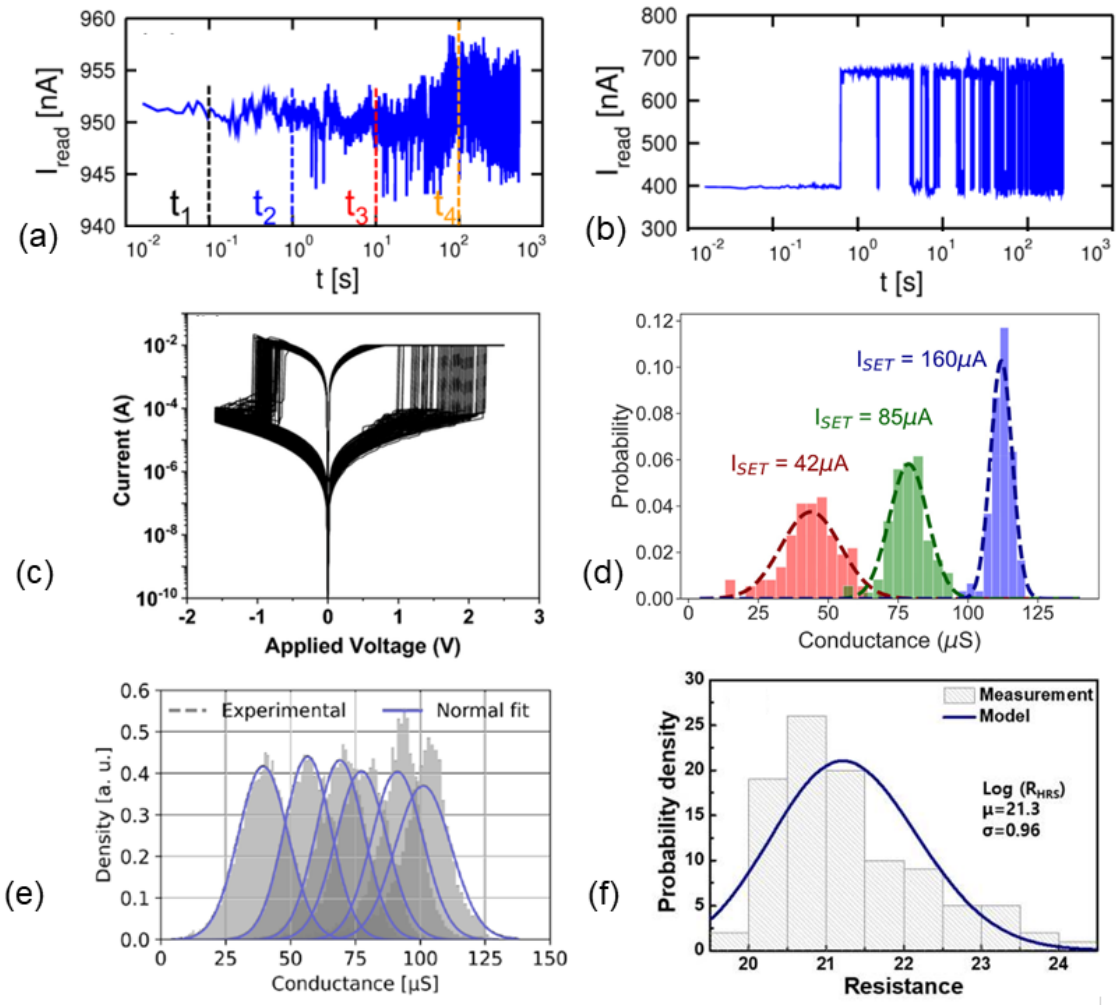


Figure 1.14: Intra-device and inter-device variability, (a) and (b) demonstrates the different types of noise (pink and RTN) that can be present in the same device when the current of the device is read at a constant voltage of 10 mV over a period of time (c) Cycle-to-cycle variability in the I-V characteristics of oxide memristors; 100 voltage sweeps exhibit different responses in the current. (d) Distribution of the LRS for different SET programming conditions, in particular, the I_{SET}. (e) and (f) the inter-device variability in the LRS and HRS, showing a Gaussian and log-normal distribution, respectively. The mean conductance of the LRS can be modulated by the I_{SET} current (Adapted from [134–138]).

other hand, the later figure shows a log-normal distribution for the HRS [137, 138].

The intra-device and inter-device variabilities are significant challenges for realizing hardware-based neural networks. This is because all traditional deep learning algorithms rely on uniform, highly precise operations, and the presence of noise or device-to-device variability is contradictory to that.

On top of all these, there are circuit-related constraints as well. For example, in an IC array, there are resource constraints limiting the total number of inputs or the noisy behavior of cer-

tain elements under low-power conditions. The conventional deep learning algorithms do not work perfectly with such constraints, and there is a need to rethink our algorithms, redesign our circuits, and reoptimize our devices so that they work in tandem with our coveted goal of the hardware implementation of deep learning.

In this thesis, chapter 2 focuses on the implementation of learning in the weak RESET regime of HfO_x-based filamentary resistive RAM using binarized neural networks. The memory technology is introduced, and its co-integration with CMOS is discussed, along with the significance of the weak RESET regime that enhances the endurance of such memory devices, a crucial parameter for on-chip learning. Then a physics-based model for the resistance evolution is developed, taking into account different types of variabilities, and is fitted to the experiments. This device model is then incorporated within the PyTorch framework to simulate learning with these devices. The simulations are done to learn the MNIST and CIFAR-10 datasets, and the test accuracies exemplify the robustness of this approach to different types of imperfections. To summarize, this chapter highlights the potential to generalize this approach for simulating other memory technologies and emphasizes the importance of studying the impact of imperfections on its performance.

Chapter 3 discusses the inference in binarized neural networks and its constraints from circuit-level implementation. First, we introduce the general ideas related to the circuit-based implementation of binarized neural networks and highlight the significant sources of imperfections originating from electronic circuits and memories. Then we present two different studies based on the circuit-level realization of binarized neural networks. For the first study, we detail the circuit used to implement inference in binarized neural networks, the sources, and analyses of the errors. Those errors are then incorporated into neural network inference simulations, and the resilience of the prediction accuracy to such errors is investigated using the MNIST and CIFAR-10 datasets. The next part presents the second study, which follows a similar structure to the first. In this work, an approach is proposed and demonstrated to circumvent array-size-related constraints at the cost of a slight degradation in accuracy. A significant source of errors in this circuit is due to the unreliable solar cell that supplies power to it. Finally, the experimentally characterized error is used in neural network simulations to demonstrate that binarized neural networks exhibit robust computation even under an irregular power supply. This chapter highlights the suitability of binarized neural networks for inference, even with different levels of imperfection and constraints, which are especially promising for edge applications.

Chapter 4 introduces a novel approach to computing that leverages the imperfection of emerging memories. Specifically, the focus is on Bayesian Binary Neural Networks, which are the probabilistic analog of binary neural networks. The chapter begins by reviewing the theory behind probability-based computing and some recent work on probability-based computing with emerging memory devices. The theory of Bayesian Binary Neural Networks is then introduced, with a focus on its differences from its deterministic analog. The concept of quantifying uncertainty is also introduced, as it is one of the main advantages of using Bayesian Neural

Networks. A toy example, the two moons dataset, is used to demonstrate the advantages of this type of neural network. This is followed by an actual medical task, the MIT-BIH dataset for arrhythmia detection, where the benefits of the Bayesian Binary Neural Network over the conventional network are showcased. Finally, spintronics-based physical systems that could be used for realizing this deep learning algorithm are discussed, and results related to performing inference are presented.

Chapter 2

Learning with imperfect Resistive RAM

> With four parameters, I can fit an elephant, with five, I can make him wiggle his trunk.
>
> John von NEUMANN

THE IMPLEMENTATION of current deep learning training algorithms is power-hungry, owing to data transfer between memory and logic units. Oxide-based resistive memories (ReRAMs) are outstanding candidates for implementing in-memory computing, which is less power-intensive. Their weak RESET regime is particularly attractive for learning, as it allows tuning the resistance of the devices with remarkable endurance. However, the resistive change behavior in this regime suffers from many fluctuations and is particularly challenging to model, especially in a way compatible with tools used for simulating deep learning. In this work, we present a model of the weak RESET process in hafnium oxide ReRAM and integrate this model within the PyTorch deep learning framework. Validated on experiments on a hybrid CMOS/ReRAM technology, our model reproduces both the noisy progressive behavior and the device-to-device (D2D) variability. We use this tool to train Binarized Neural Networks for the MNIST handwritten digit recognition and CIFAR-10 object classification tasks. We simulate our model with and without various aspects of device imperfections to understand their impact on the training process. The framework can be used in the same manner for other types of memories to identify the device imperfections that cause the most degradation, which can, in turn, be used to optimize the devices to reduce the impact of these imperfections. This chapter is adapted from a publication in the journal IEEE Transactions on Electron Devices by the author of this thesis [4]. The experimental characterizations were done at Aix-Marseille University by Pr. Marc Bocquet.

2.1 Background

The advance of machine learning algorithms holds remarkable prospects in terms of benefits to society [139]. However, as extensively discussed in the previous chapter, this progress comes at the cost of a considerable energy budget [140]. The bulk of this energy consumption is attributed to the shuttling of information between the memory and logic units of the computing system [141], a bottleneck that the use of in-memory computing can circumvent. For such designs, oxide-based ReRAMs, or memristors, are a major breakthrough. Their fast, low-power, non-volatile switching and full compatibility with the CMOS process lend quite well towards the realization of energy-efficient, adaptable synaptic weights [142, 143]. Unfortunately, owing to their dependence on the nanometer-scale physics of atoms and ions, oxide-based ReRAMs are usually very difficult to model accurately, which is a challenge for designing in-memory neural networks.

Oxide-based ReRAM devices switch through the formation and dissolution of conductive filaments of oxygen vacancies (fig. 2.1(a)). They function based on a combination of transport, thermal, and electrochemical effects; a multiplicity of mechanisms of atomic movement can coexist within the same device, giving rise to different regimes, depending on the state of the device and bias conditions [144]. Additionally, the devices exhibit fluctuations that resist simple modeling [127, 128]. In recent years, considerable progress has been made in model-

ing these devices in the regimes relevant for embedded and standalone memory applications [144–148]. On the other hand, a programming regime known as weak RESET (programmed with a low voltage)[149, 150] remains vastly unexplored, as this regime, presenting exacerbated fluctuations, has no usage for conventional memory applications. Remarkably, recent works suggest that this regime might be extremely useful for artificial intelligence (AI) and neuromorphic applications, allowing such systems to do learning using little power and area [150, 151]. Although studies about low voltage switching[152], device models[153], and noise[154, 155] have been carried out in the past, a comprehensive study integrating all these aspects has not been done. To investigate this lead convincingly and design systems, an accurate model of the weak RESET process is needed. Additionally, the model needs to be compatible with the very specific frameworks used for designing neural networks (PyTorch, TensorFlow, etc.), optimized for operating on graphics processing units (GPUs) and to perform automatic differentiation and were not designed to include device effects such as noise and variability [70, 156].

In this work, we propose an efficient analytical behavioral model for the weak RESET regime of HfO_x-based ReRAM, including device fluctuations, and implement it within a deep-learning framework to model synaptic parameters. We provide and validate this model with extensive measurements, using multiple statistical quantities, on a hybrid HfO_x ReRAM/CMOS integrated circuit.

This device model is specifically optimized for integration within deep learning frameworks. This feature allows us to investigate the behavior of such devices in the context of neural network training. We implement this model within PyTorch, a deep-learning framework, by adapting the optimizer. We present simulation results of binarized neural network (BNN), a quantized form of more traditional neural networks for which the weak RESET regime of ReRAMs is particularly attractive, using fully connected and convolutional architecture for MNIST and CIFAR-10 tasks, respectively. Finally, using these simulations, we study the impact of device imperfections on network performance.

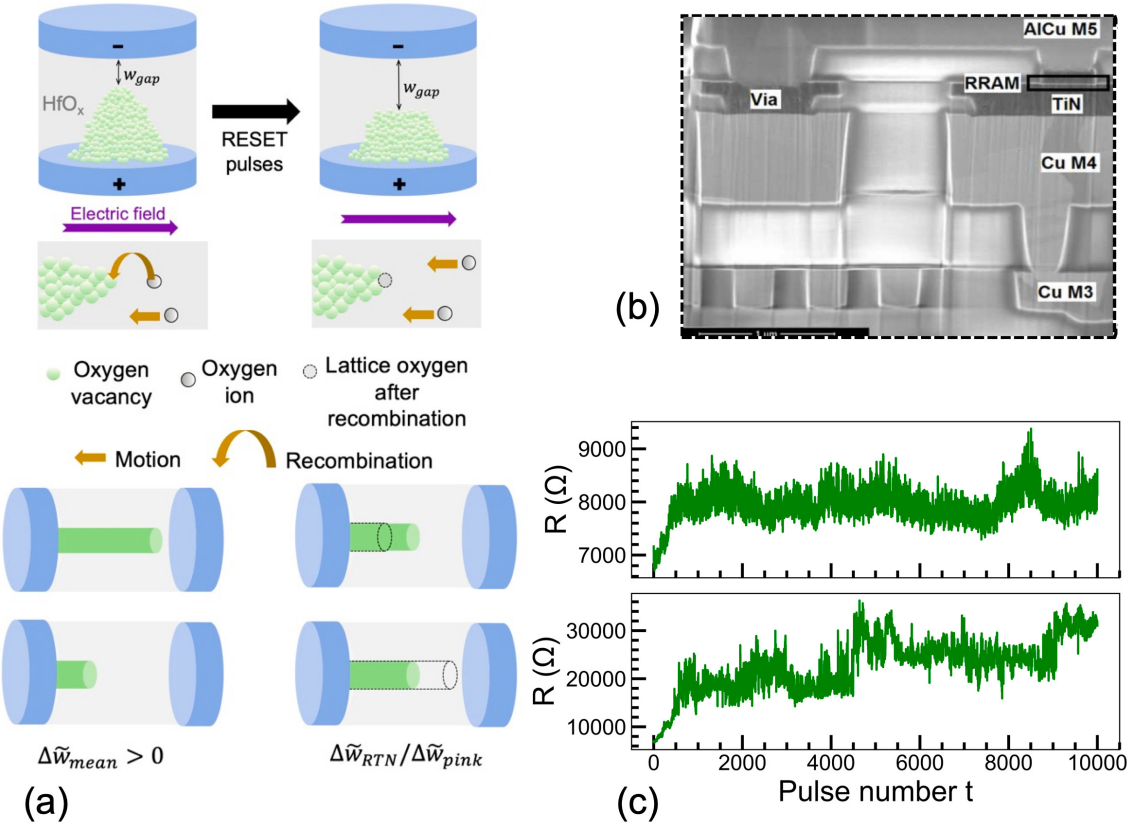


Figure 2.1: (a) Illustration of the progressive dissolution of the conducting filament by recombining oxygen ions and vacancies under the influence of consecutive RESET pulses. Also, the increment ($\Delta\widetilde{w}_{mean}$) and fluctuation terms ($\Delta\widetilde{w}_{RTN}/\Delta\widetilde{w}_{pink}$) of our device model (described in Section 2.3) are shown schematically. (b) SEM image of an HfOx-based ReRAM device integrated into the BEOL of our technology. (c) Progressive evolution of the resistance of two measured devices with consecutive weak RESET pulses of amplitude 1 V and writing time of 0.1 μs.

2.2 Hafnium Oxide ReRAM Technology

2.2.1 The technology

For this work, we rely on measurements of a hafnium oxide (HfO_x)-based OxRAM technology. The memory stack has a TiN/HfO_x(10 nm)/Ti(10 nm)/TiN composition where the TiN layers serve as the electrodes [157]. Our nanodevices are integrated within the back-end-of-line of a 130 nm commercial CMOS process, between metal levels four and five, as shown in fig. 2.1(b). Such integration of logic and memory facilitates the implementation of energy-efficient in-memory computing. Each memory device is associated with an NMOSFET, allowing precise control of the programming conditions, such as the compliance current, which enables the formation of the conducting filaments [145]. After an initial electroforming step, the device can switch between low-resistance (LRS) and high-resistance states (HRS) depending on the polarity of the applied voltage pulses. The switching between LRS and HRS is attributed to the gradual formation and dissolution of the conductive oxygen-vacancy filaments within the oxide.

2.2.2 Weak RESET regime

The weak RESET regime is stimulated by applying low voltage negative pulses, and pulse times are shorter than traditional RESET. It makes the switching smoother, which enables the finer tuning of resistance at the cost of a reduced HRS/LRS ratio. Measurements in fig. 2.1(c) show that repeated 1 V weak RESET pulses lead to a progressive increase in the cell resistance, albeit in a noisy manner. In this Figure, the resistance is read at a very low voltage (0.1 V) after each weak RESET pulse so that a very low read current flows through the device, and, therefore, there is no read disturb effect. We choose the weak RESET regime of operation to achieve high endurance in our devices. This is essential for learning tasks, as individual devices are required to be programmed reliably for many cycles. Fig. 2.2 shows the outstanding endurance of two complementary devices, each with resistances R_{BL} and R_{BLb}, that are programmed in the weak RESET, for more than 10^{10} cycles. This is orders of magnitude more than when devices are used with traditional higher-voltage RESET [157], and orders of magnitude more of what is needed for practical learning tasks (e.g., 10^4 cycles for the CIFAR-10 object recognition task).

However, the resistance increase due to the weak RESET seen in fig. 2.1(c) is particularly noisy and in a way that appears non-trivial. Cells in the weak RESET regime are, therefore, reminiscent of biological synapses, which also modulate their conductivity (weight) during the learning process in a way that is often believed to be noisy [158]. Recently, it has been shown that ReRAM cells in weak RESET could indeed be used to do learning for a type of neural network called binarized neural networks (BNNs), which are more resilient to noise and less energy-consuming than analog neural networks [150, 151].

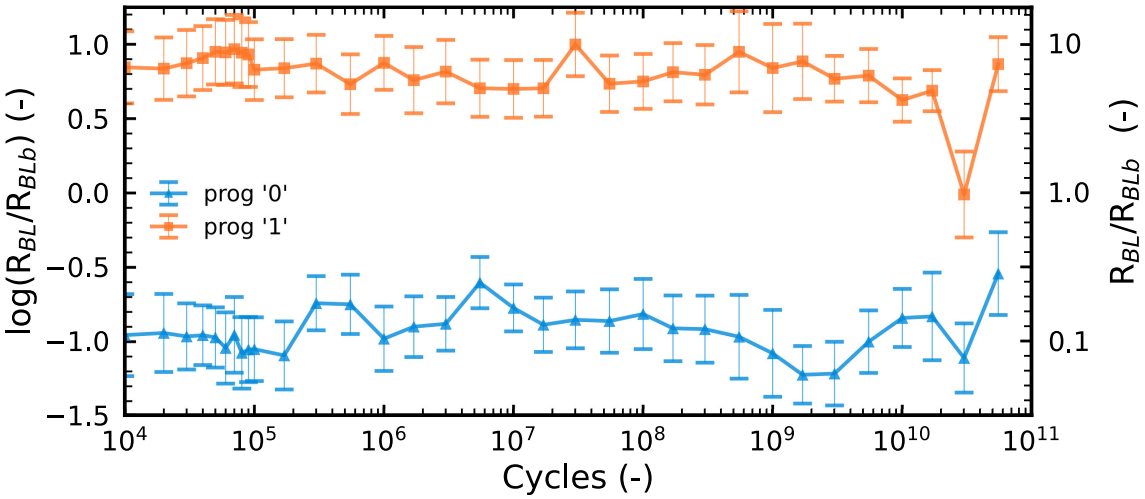


Figure 2.2: Endurance measurement on two complementary devices programmed with weak RESET pulses of width 1 μs and SET compliance current of 200 μA: median value of log resistance ratio (R_{BL}/R_{BLb}), extracted over 10k rounds for measurement of a pair of devices over 5×10^{10} cycles.

2.3 Device Characterization and Modeling

2.3.1 Tunneling gap-based model

In this section, we introduce our device model for resistance in the weak RESET regime. To model the weak RESET behavior, we take the established approach of using the tunneling gap between the partially dissolved oxygen-vacancy filament and the electrode (fig. 2.1(a)), w_{gap} as the state parameter [145, 146]. For practical purpose, we use the dimensionless quantity $\widetilde{w}$, defined as

$$\widetilde{w} = w_{gap}/w_0, \tag{2.1}$$

where w_0 is a length scale associated with the standard size of the filament. Owing to its quantum mechanical origin, the resistance of the device associated with the tunneling gap $\widetilde{w}$ is

$$R(\widetilde{w}) = R_0 \exp(\widetilde{w}), \tag{2.2}$$

where R_0 is the resistance of the device in LRS, i.e, when the tunneling gap is zero. The model does not include filament diameter, which appears to have a second-order effect during the weak RESET process. The variations in the tunneling gap $\widetilde{w}$ give rise to its progressive RESET behavior. It also leads to noise which is a consequence of invasive biasing and is not related to the read noise [159]. Fig. 2.3(a) shows an example of $\widetilde{w}$ extracted from measurements, showing both its increasing trend and its noise when successive weak RESET pulses are applied.

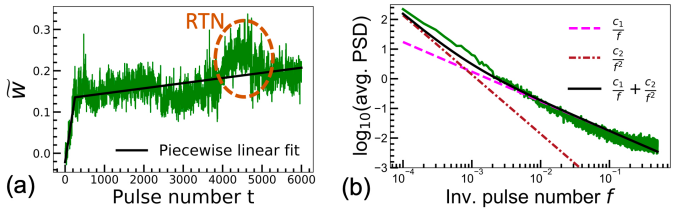


Figure 2.3: (a) Piecewise linear fit of the mean model to the $\widetilde{w}$ of a device (different than fig. 2.1(c)). The parameters m_1, t^*, c_1 and m_2 are extracted from this fit. (b) Power spectral density of $\widetilde{w}$ averaged over 64 devices showing the presence of a $(1/f^2)$ trend for low frequencies and a pink-noise like $(1/f)$ response for higher frequencies.

2.3.2 Mean model

We observe in our devices (fig. 2.3) the existence of two regimes for our quantity of interest $\widetilde{w}$; the initial more progressive increase and the subsequent noisier and less monotonic parts. Hence, in our model, the stable, background contribution, $\widetilde{w}_{mean}$ is described by a piecewise linear model as a function of the pulse number t, parameterized by the device-dependent parameters m_1, c_1, t^* and m_2 as

Mean model equation

$$\widetilde{w}_{mean} = \begin{cases} m_1 t + c_1 & t < t^* \\ m_2 t + (m_1 - m_2) t^* + c_1 & t \geq t^*. \end{cases} \tag{2.3}$$

The first regime ($t < t^*$), where the increase of resistance is steeper and less noisy, is physically related to conditions where the heating due to the Joule effect is more pronounced, compared to the later one ($t \geq t^*$), where the resistance of the device is higher. Under this condition, the resistance increase is much less monotonic and prone to more noise.

2.3.3 Noise components

We first compute the power spectral density (PSD) of $\widetilde{w}$ extracted from measurements to characterize the fluctuations in the value of the resistance. As shown in fig. 2.3(b), the PSD averaged over 64 devices exhibits both a $1/f^2$ and a $1/f$ contribution. The $1/f^2$ part is consistent with the Random Telegraph Noise (RTN) that we find in our devices (fig. 2.3(a)) [160]. On the other hand, the $1/f$ dependence indicates the existence of pink noise. Both types of noise are related to the switching process and are independent of the passive noise we would get during read-out only. In our model, we capture these two types of noise by the quantities $\widetilde{w}_{pink}$ and $\widetilde{w}_{RTN}$.

2.3.3.1 RTN

RTN can be found in the second regime of the mean model (fig. 2.3(a)) and is attributed to the perturbations related to the creation and destruction of oxygen vacancies in non-stoichiometric hafnium oxide [159]. The RTN component ($\widetilde{w}_{RTN}$) is modeled as a two-state Markov process

Simulation of RTN

$$\widetilde{w}_{RTN} = aX, \tag{2.4}$$

where X is a random variable taking a value of zero or one depending upon the resistive contribution from the fluctuations of the vacancies, and a is the amplitude of the resistance jumps [161]. The probabilities of switching from zero to one and vice-versa are given by P_{high} and P_{low}, which are asymmetric. Hence, the transition matrix T, of the Markov process is defined as $T_{12} = P_{high}$ and $T_{21} = P_{low}$.

2.3.3.2 Pink noise

On the other hand, the pink noise might be related to the dynamically changing defect states in the oxide [162]. It is modeled using an approach where the values can be sequentially generated, which is more suitable for the GPU-based implementation expected for deep learning frameworks [163]. In this method, the pink noise is generated in the following manner.

2.3.3.3 Sequential generation of pink noise

Simulation of pink noise

Firstly, $pole$ number of white Gaussian random numbers, ω_r $(r = 0, 1, ..., pole)$ are generated. These values are then passed through a low-pass FIR filter with coefficients (b_r), which are the impulse response values so that the generated noise is pink in its PSD. Thus, mathematically the pink noise component is described by

$$\widetilde{w}_{pink} = \alpha \sum_{r=0}^{pole} b_r \omega_r, \tag{2.5}$$

where α is a scaling factor.

Since we are simulating the update of the state variable $\widetilde{w}$ for each device, the number of pulses applied to them can vary. Hence, we need this method to generate new pink noise values sequentially. We always have $pole$ number of pink noise values stored for all devices. Then, if n pulses are applied to a device, the first n numbers from the store are removed, and the same

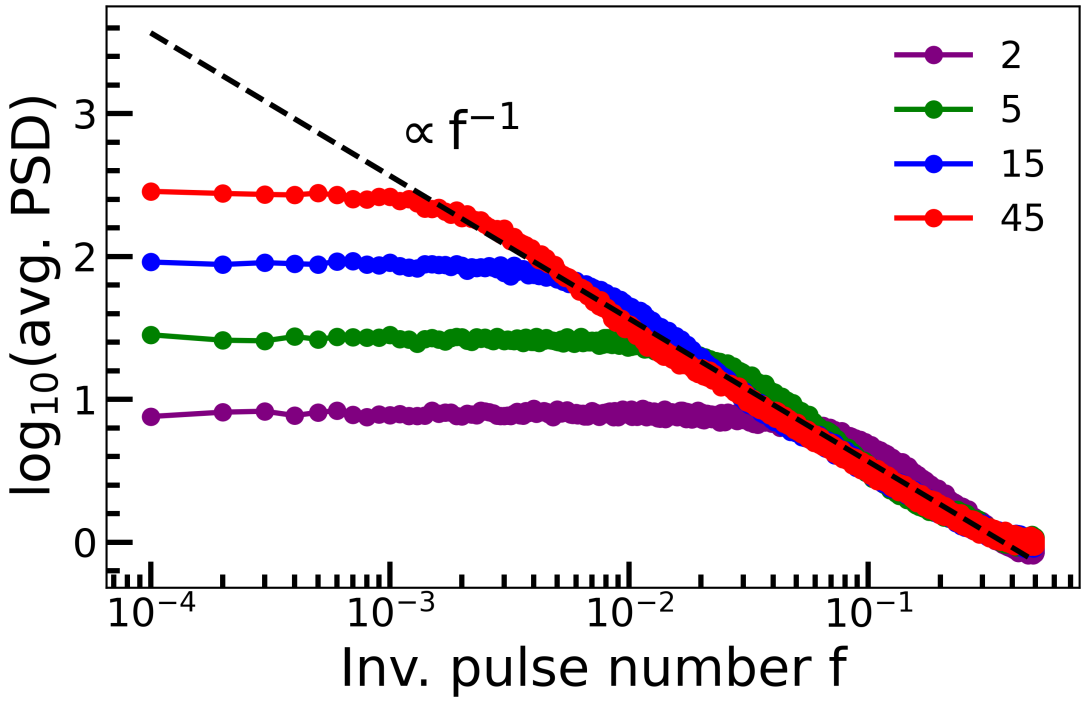


Figure 2.4: Power spectral density of the simulated pink noise values generated sequentially for different $pole$ values. The density is averaged over 1000 instances and plotted as a function of the inverse pulse number f. The black line represents the ideal behavior where the density varies with the inverse of f, and the simulations match this more with higher values of $pole$.

number of values are appended. This method ensures that we are not generating new numbers unnecessarily, which is computationally expensive. On the other hand, because of this, we can apply at most $pole$ number of pulses. Essentially the parameter $pole$ relates to the temporal correlation of the pink noise sequence, which tends to infinity in an ideal case. Fig. 2.4 shows the power spectral density for the sequential pink noise generation algorithm for different $pole$ values.

The power spectra of the simulated pink noise sequences follow the f^{-1} law for the higher values of f, which corresponds to low values of pulse numbers, which is expected since the $pole$ values here only range from 2 to 45. The deviation for low values of f is not detrimental because, as we see in fig. 2.3(b), the low f limit is dominated by the $1/f^2$ noise. Also, the long-term variation is dominated by the piecewise linear increase, which is captured by the mean-model aspect. We choose a $pole$ value of 15 because it preserves the f^{-1} nature up to a value of 10^{-2} with a reasonable computational time.

The variation of our state variable $\widetilde{w}$ is then obtained by the superposition of these three components as

$$\widetilde{w} = \widetilde{w}_{mean} + \widetilde{w}_{RTN} + \widetilde{w}_{pink}. \tag{2.6}$$

The physical impact of the variation of these three terms is shown schematically in fig. 2.1(a). In addition, ReRAM devices are subject to important device-to-device (D2D) variability due to the various possible topologies of the conductive filaments and dynamic perturbations, which can considerably impact neuromorphic applications and should be modeled carefully.

2.3.4 Fitting the parameters

To account for the device-to-device variability, we fit our mean model to the experiments on 64 devices integrated into a memory array, as shown in fig. 2.5. In this figure, we see the D2D variability can be very high for $\widetilde{w}$; wherein the range of values, pulse number at which the change of regime occurs t^*, and the slopes (c_1 and c_2) can all vary significantly. To incorporate this into our model, we plot the statistical distribution of these parameters.

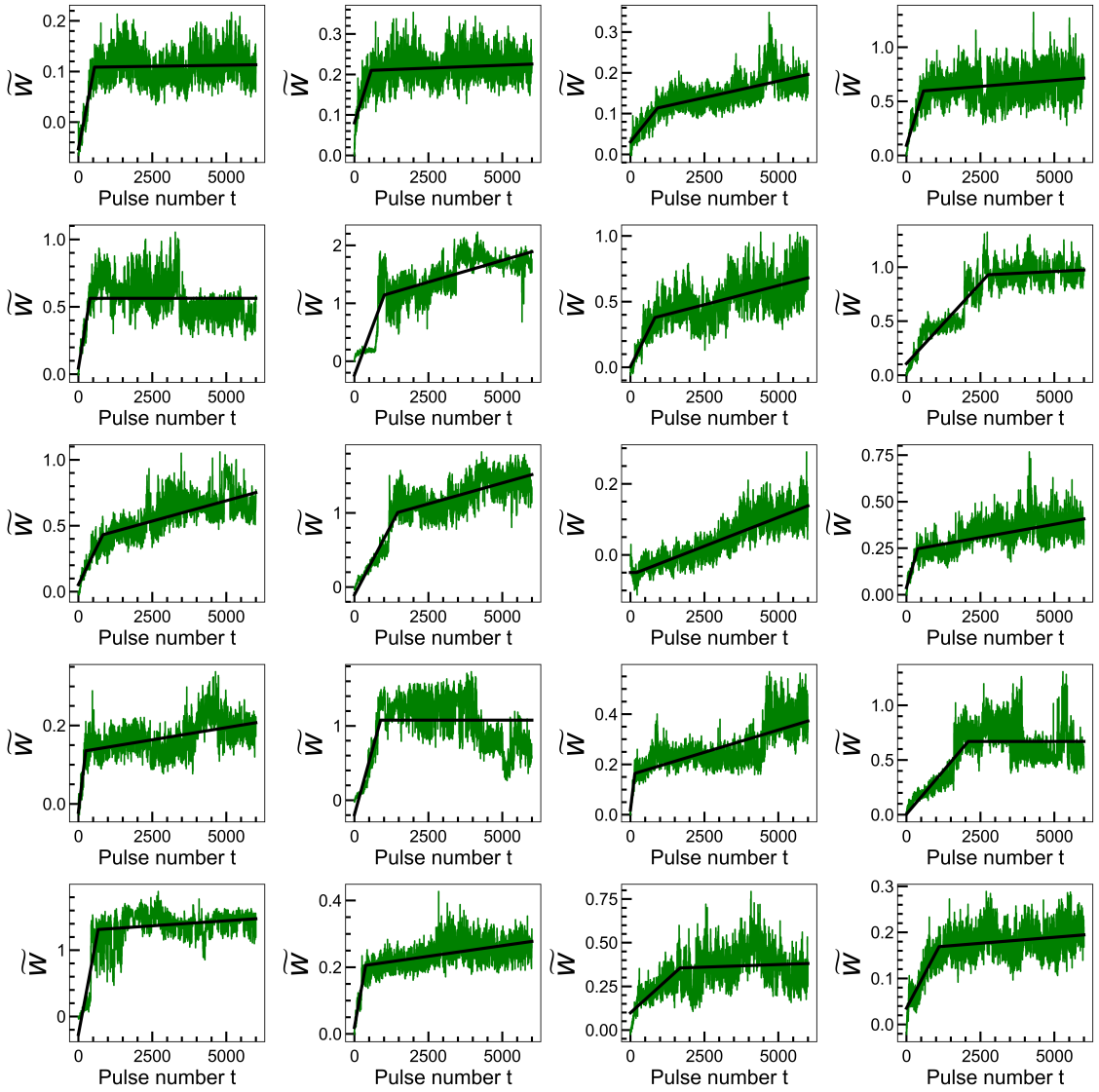


Figure 2.5: Piecewise linear fit of the mean model to a set of devices. The mean model with a piece-wise form is meant to capture the two regimes, the initial progressive increase and the consequent noisier and less steep increase.

Fig. 2.6(a)-(d) shows the distribution of the parameters of our mean model. The distributions of the m_1, t^*, c_1, and m_2 parameters can be well fitted using an exponential, a lognormal, a Gaussian, and an exponential distribution, respectively. The variation in the absolute value of the resistance is done by sampling the initial resistance R_0 (Eq. 2.2) from a Gaussian distribution whose parameters are extracted from the experimental initial LRS distribution of the devices (fig. 2.6(e)). This makes sure that even if we are only dealing with the variations in the state parameter $\tilde{w}$, the absolute resistances also bear the same variability as the devices. Table 2.1 lists the extracted parameters used for our simulations. The parameters m_1 and m_2 describe the monotonic progressive increase of the filament gap and follow an exponential law,

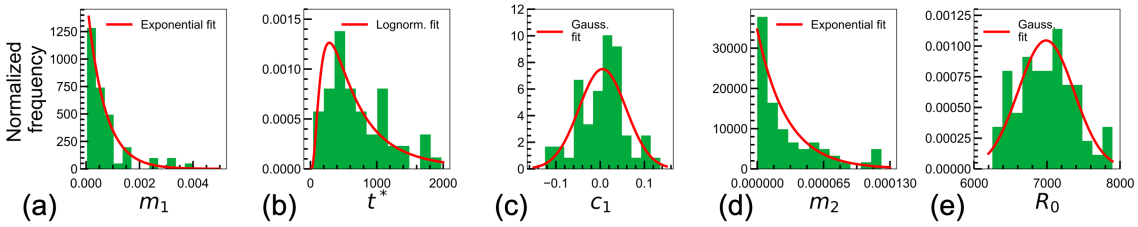


Figure 2.6: The statistical distribution of the extracted parameters over the 64 devices. The respective slopes for the two regimes m_1 and m_2 both follow exponential distributions (a) and (d). The threshold pulse number t^* follows a log-normal distribution (b), whereas both the initial intercept c_1 (c) and the initial resistance (e) follow Gaussian distributions.

highlighting that some devices are relatively insensitive to weak RESET. The Gaussian distribution of R_0 and c_1 is connected to the Gaussian distribution of the LRS.

The parameters used to generate the noise (cycle-to-cycle variation) are fine-tuned so that the experiments and simulations in fig.2.7(b) and (c) match. The values are summarized in Table 2.2. The obtained values naturally replicate the noise levels observed in both regimes of the mean model.

2.3.5 Comparison of experiments and simulation

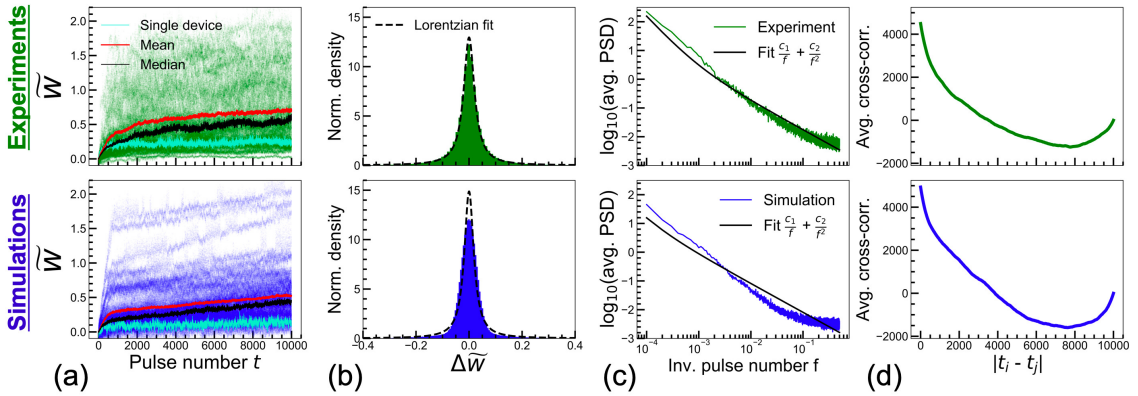


Figure 2.7: Comparison of experiments and simulations over 64 devices. (a) Scatter plots of $\widetilde{w}$ as a function of the number of applied weak RESET pulses. Also, the evolution of $\widetilde{w}$ for a single device is shown for both. (b) Histograms of changes in $\widetilde{w}$ after each pulse. (c) Average power spectral density of $\widetilde{w}$ following Lorentzian distributions. (d) Average cross-correlation between the devices.

Fig. 2.7 shows that the resulting model, integrating D2D, reproduces all measured aspects of the experiments with outstanding accuracy. Fig. 2.7(a) shows the individual trajectories in the weak RESET process of 64 measured and 64 simulated devices. Fig. 2.7(b) shows that the distribution of the changes in $\widetilde{w}$ after each weak RESET pulse follows the same Lorentzian distribution in both the experiments and simulations. It is centered at zero, which implies that

Table 2.1: **Device-to-device variation**: Parameters for RTN, mean model components, and the initial resistances that characterize the variability between devices and extracted from fig. 2.6. The probability density functions had the following forms:
$f_{uniform}(x; a_1, a_2) = \frac{1}{a_2 - a_1}$, $a_1 \leq x \leq a_2$.
$f_{exponential}(x; x_0, \lambda) = \frac{1}{\lambda} \exp\left(-\frac{x - x_0}{\lambda}\right)$, $x \geq x_0$.
$f_{Gaussian}(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$.
$f_{lognormal}(x; s, \sigma) = \frac{1}{sx\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{log(x/\sigma)}{s}\right)^2\right]$, $x \geq 0$.

Component	Model param.	Distr.	Distr. param.
RTN amplitude	a	Uniform	a_1=0, a_2=0.5
Mean model	m_1	Exponential	x_0=3.74e-5, λ=6.56e-4
	c_1	Gaussian	μ=5.29e-3, σ=5.32e-2
	t^*	Log-normal	s=0.80, σ=542.5
	m_2	Exponential	x_0=1.64e-34, λ=2.89e-5
Resistance	R_0	Gaussian	μ=6988 Ω, σ=381.7 Ω

the fluctuations dominate over the monotonic changes arising from the mean model, which would have caused a bimodal distribution of positive values. The narrow peaks and wide tails of the Lorentzian distribution represent the more frequent pink noise and the less frequent RTN-induced fluctuations. Fig. 2.7(c) shows the mean spectral power spectrum of fig. 2.7(a), and fig. 2.7(d) shows the mean cross-correlation of $\widetilde{w}$ between the 64 devices, where t_i and t_j are the pulse numbers to the i^{th} and j^{th} devices, defined as:

$$Cross\,Corr.(|t_i - t_j|) = \sum_{all\ t_i, t_j} \widetilde{w}(t_i)\,\widetilde{w}(t_j). \tag{2.7}$$

The average cross-correlation between the 64 devices measures the D2D variability captured by our model, which also agrees with the experiments. The mean auto-correlation at zero shift is about 10,000, which is twice the average cross-correlation at zero shift, indicating that the inter-device variability is larger than the intra-device one. Overall, the model, therefore, seems ideal to mimic ReRAM cells.

2.3.6 Algorithm for device model

Our final goal is to integrate this model within the PyTorch framework. To that end, we need an algorithmic setting for the aforementioned model. The following pseudo-code represents how

Table 2.2: **Cycle-to-cycle variation**: Parameters for RTN and pink noise that account for the noise on the resistance of devices.

Component	Model param.	Value
RTN	P_{high}	0.0008
	P_{low}	0.002
Pink noise	α	0.025
	$pole$	15

the resistances change in our devices based on the number of applied pulses (RESET function).

Algorithm 1 The RESET function. The inputs to the function are the number of pulses to apply n, the number of pulses already applied to the device t, the RTN state X, the transition matrix T, the pink noise state represented by ω_0, ω_1, ..., ω_{pole}. The parameters of this function are of two types; the device-specific ones m_1, c_1, t^*, m_2, R_0, a, and the general parameters P_{high}, P_{low}, α, $pole$

Inputs: n, t, X, T, ω_0, ω_1, ..., ω_{pole}, m_1, c_1, t^*, m_2, R_0, a, P_{high}, P_{low}, α, $pole$.
Outputs: Resistance R, updated values of t, X, and ω_0, ω_1, ..., ω_{pole}.

Mean model: $t \leftarrow t + n$
if $t < t^*$ **then**
    $\widetilde{w}_{mean} \leftarrow m_1 t + c_1$
else if $t \geq t^*$ **then**
    $\widetilde{w}_{mean} \leftarrow m_2 t + (m_1 - m_2) t$
end if
RTN: $T \leftarrow T^{n+1}$
if X=0 **then**
    $X \leftarrow 1$ with probability T_{12}
else if X=1 **then**
    $X \leftarrow 0$ with probability T_{21}
end if
$\widetilde{w}_{RTN} \leftarrow aX$
Pink noise: Generate n new ω_r^{new} values (see Appendix) for $r = 0, 1, 2, ..., n-1$ and then append the new values such that there are $pole$ number of pink noise states.
$\widetilde{w}_{pink} \leftarrow \sum_{r=0}^{n-1} b_r \omega_r^{new} + \sum_{r=n}^{pole} b_r \omega_r$
$\widetilde{w} = \widetilde{w}_{mean} + \widetilde{w}_{RTN} + \widetilde{w}_{pink}$
$R = R_0 exp(\widetilde{w})$
return R, t, X, and ω_0, ω_1, ..., ω_{pole}

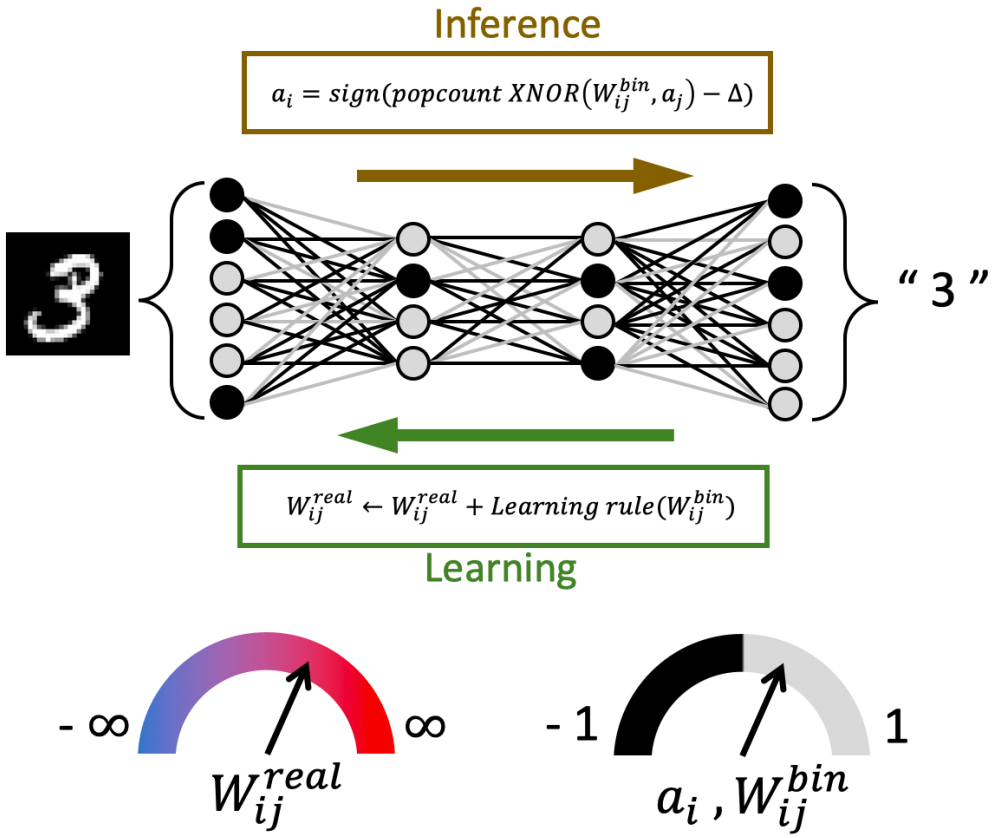


Figure 2.8: Principle of operation of a Binarized Neural Network (BNN) showing the forward pass (inference) and backward pass (learning or training). The inference depends only on the binarized weights W_{ij}^{bin}, whereas the training involves updating the real weights W_{ij}^{real}.

2.4 Implementation within a Deep Learning Framework

Artificial neural networks (ANN) are networks of neurons, connected by synapses, laid hierarchically: the neuronal activations of a layer are computed from the neurons of the previous layer. The value of neuron activation is computed by taking the sum over the previous activations weighted by their corresponding synaptic values and then applying a non-linear function. Learning a task aims to find an optimum set of values for the synaptic connections, called weights. To that end, analog memory cells have been used as the weights owing to their ability to adapt conductances [125].

2.4.1 Binarized neural networks

However, to train ANNs, precise values of these weights need to be stored and updated since the weights and activations can take any real value. This is a problem for ReRAM-based implementation in the weak RESET regime, as inter-device and intra-device variabilities are ubiquitous

in such nano-devices, as seen in section 2.3. An alternative approach is to use BNNs, where both the neuronal activations and synaptic weights take binary values (+1 and -1) [164, 165]. Despite this simplicity of representation, BNNs can approach state-of-the-art accuracy on vision tasks [164]. During inference, that is, calculating the network output given the input, their arithmetic is extremely simple. A simple XNOR operation replaces the product of the activation and the weight. Also, the accumulation of the products can be simply done by counting the number of ones, called the population count. Both of these can be implemented using relatively simple, low power-consuming circuitry [157]. The advantage of binarization is both from the reduced read-out complexity and the fact that low-precision synapses and weights can be used for inference.

During the training phase i.e., when the network learns the optimum values of the weights, a hidden, real-valued weight is also associated with the synapses [164, 165]. As shown in fig. 2.8, the binarized weight W_{ij}^{bin} connecting the neuron a_j of the previous layer to the neuron a_i of the next layer relates to the hidden real weight, W_{ij}^{real}, as

$$W_{ij}^{bin} = sign(W_{ij}^{real}), \tag{2.8}$$

and the binarized activation is given by

$$a_i = sign(POPCOUNT(XNOR(W_{ij}^{bin}, a_j) - \Delta)), \tag{2.9}$$

where Δ is a threshold that serves the role of shifting in batch normalization of the activation values. During the inference phase, only the binarized weights need to be calculated. On the other hand, for learning, the hidden real weights need to be updated by a learning rule but not explicitly read. We utilize this by avoiding using energy-intensive circuits that are required to read the analog resistance state that plays the role of the real weights. Following the approach of [150], we employ a differential 2T2R structure within a crossbar array (fig. 2.9(a)), in which the two resistances R_{BL} and R_{BLb} account for a single real synaptic weight as

$$W_{ij}^{real} = \log_{10}(R_{BL}/R_{BLb}). \tag{2.10}$$

As shown in [166] and [167], the 2T2R scheme based on the ratio of two resistances provides a lower error rate compared to 1T1R which is crucial for the device to operate in the weak RESET regime. The 2T2R structure also allows performing training relying solely on RESET pulses.

2.4.2 Training in ReRAM-based BNNs

In the training phase, to update the real weight, the ReRAM devices are programmed using weak RESET pulses on either of the two devices. If the BNN learning rule suggests increasing the real weight by δW_{ij}^{real}, we apply weak RESET pulses to the BL device, therefore increasing W_{ij}^{real}. Conversely, if δW_{ij}^{real} is negative, we apply weak RESET pulses to the BLb device, there-

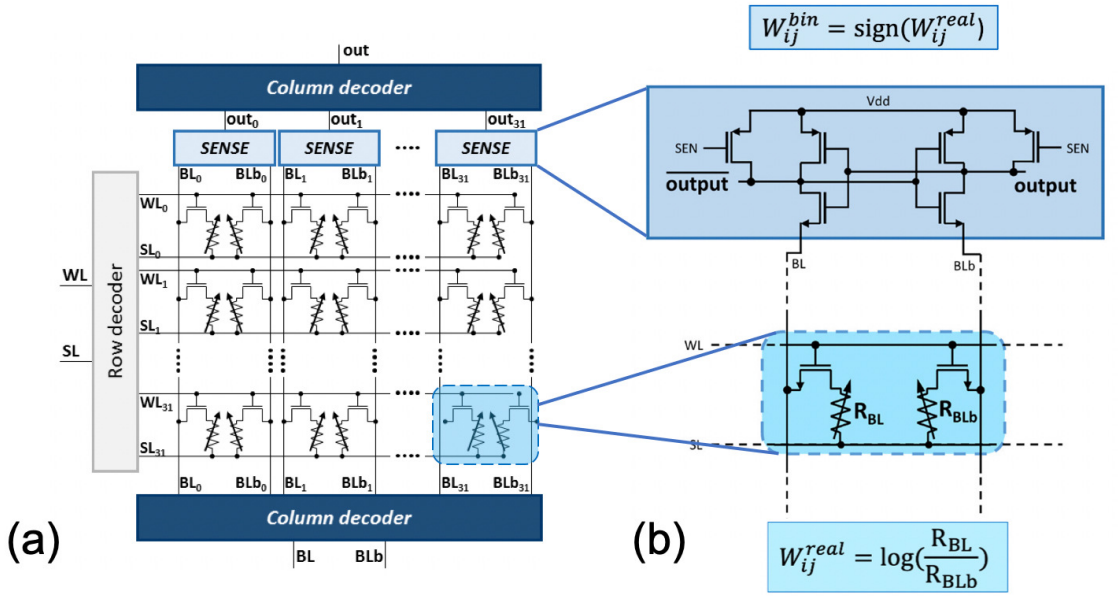


Figure 2.9: (a) Schematic of the 2T2R memory array used for implementing BNNs. (b) Circuit of the sense amplifier used to extract the binary weight from a 2T2R synapse, along with equations showing how the resistances R_{BL} and R_{BLb} connect with the real (W_{ij}^{real}) and binary (W_{ij}^{bin}) weights.

fore reducing W_{ij}^{real}. In both cases, the number of pulses is chosen proportionally to δW_{ij}^{real}. Due to the differential 2T2R nature of the synapses, this training technique requires only RESET pulses. For the tasks we have performed, we have seen that the progressivity of the RESET process is sufficient; however, for more complex tasks, this might not be enough. In that case, we can apply a reprogramming strategy, proposed in [150], to bring back the system where proper RESET is applicable.

For the inference, the sign of this real hidden weight has to be read, and this can be achieved by an energy-efficient and fast circuit called pre-charge sense amplifier [157, 168]. It compares R_{BL} and R_{BLb} to give an output of +1 when the former is larger and -1 for the opposite. Fig. 2.9(b) shows how the real and binarized weights are computed in the circuit.

2.4.3 Framework implementation

The frameworks normally used for designing neural networks, such as PyTorch and TensorFlow, model synapses as floating-point real weights. When a neural network is trained, sophisticated optimization algorithms, called optimizers, such as adaptive moment estimation, optimize these weights values by making noiseless, highly precise, and deterministic updates [55]. To test our vision, i.e., to design a physical model where synapses are implemented by ReRAM, and the weights are updated using weak RESETs, we adapted the PyTorch deep learning framework in three important ways. First, in deep learning frameworks, the synaptic parameters

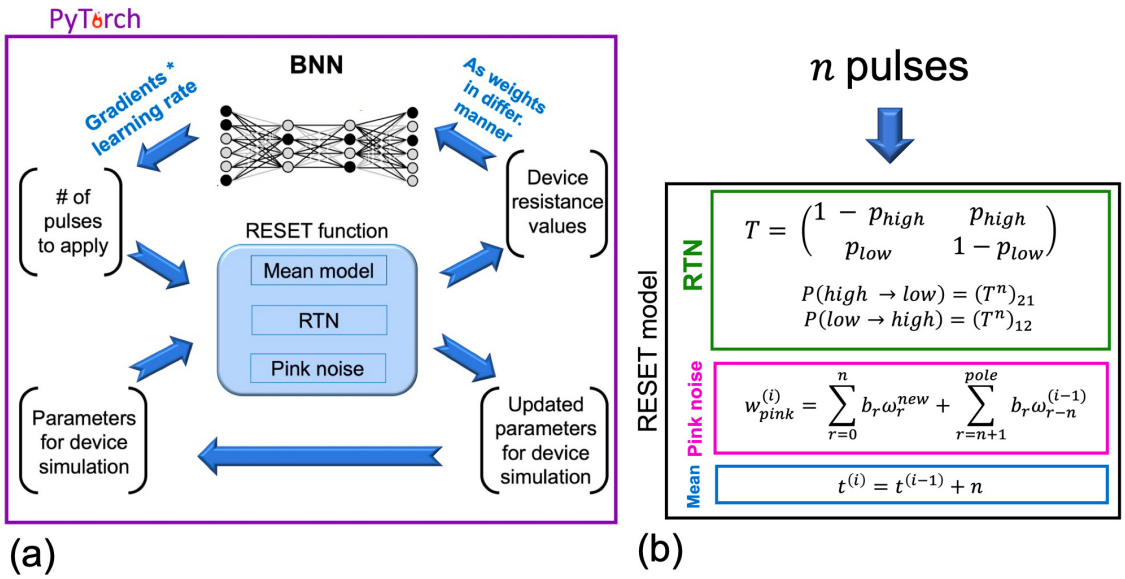


Figure 2.10: (a) Integration of device simulation and neural network learning within the PyTorch framework. The device resistances act as the synaptic weights in a differential manner. They are updated according to the device model from the updates provided by the network in the backward pass. (b) The equations for the mean, RTN, and pink noise components inside the RESET function that models the programming of ReRAMs within the PyTorch adaptive moment estimation optimizer.

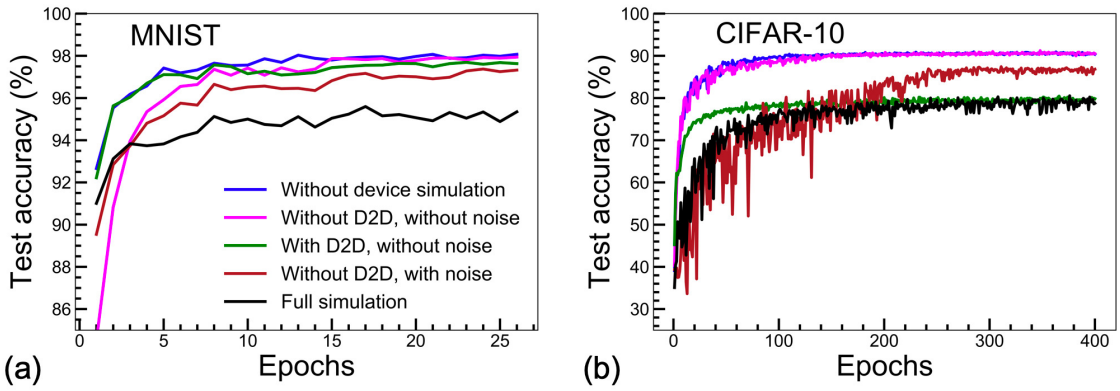


Figure 2.11: Impact of noise and device-to-device variability on the performances of the binarized neural networks for the (a) MNIST and (b) CIFAR-10 tasks. The plots show the test accuracy during training for five different cases - without device simulation (blue), without both D2D variation and noise (pink), with D2D variation but without noise (green), without D2D but with the full noise and mean model simulation (brown), and the full simulation incorporating both the D2D variability, mean model and noise (black).

are stored as tensors with dimensions appropriate to the corresponding architecture. In our approach, these parameters are now modeled by an added dimension that accounts for the device state variables. These are the different parameters that are needed to store the number of

pulses that have been previously applied to a device and to generate pink and telegraph noise (pulses already applied t, RTN state variable X and ω_rs).

Secondly, the parameters of the neural network are typically initialized according to certain pre-defined initialization schemes [65, 169]. In our case, as the synaptic parameters are linked with the device resistances, we initialize the devices by sampling through the distributions mentioned in Table 2.1.

Finally, the in-built optimizers provide updates that are real-valued floating-point numbers. But, in ReRAM-based networks, we can only modify the resistances by the application of a discrete number of voltage pulses. Thus, the updates given by PyTorch's adaptive moment estimation methods are discretized by multiplication by a suitable learning rate and rounding down to integer values. These pulses then produce the synaptic updates following the model of section 2.3.

The scheme of integrating our device model into the PyTorch framework is schematically shown in fig. 2.10(a). Synaptic weights are initialized as device resistance values in a differential manner incorporating the D2D variability explored in section 2.3. The network does the forward pass on the input and calculates the updates for the weights, which are then converted to integer-valued pulses numbers n that are to be applied to the devices. Using the number of pulses and the device-based parameters, the new device resistance states are calculated as shown in fig. 2.10(b).

The RTN, pink noise, and mean model components are calculated separately. The RTN state variable is calculated from exponentiating the transition matrix T to the n^{th} power. Pink noise values are generated by drawing n new Gaussian white random numbers and combining them with the existing $(pole - n)$ values. And, for the mean model component, the pulse number n is simply added to the number of pulses already applied t. Now, with the new device resistances and the new synaptic weights, the network continues onto the next forward pass.

2.4.4 Algorithm for learning with device model

Algorithm 2 The learning algorithm with the device model. This is how we incorporate the device model into our learning procedure. $\mathbf{W^{real}}$ and $\mathbf{W^{bin}}$ are the vectors of real and binarized weights, respectively. The BL and BLb resistances of the differential cells are represented by $\mathbf{R_{BL}}$ and $\mathbf{R_{BLb}}$. The update to the real weight is denoted by $\mathbf{U}$, and the corresponding number of pulses to be applied, $\mathbf{n}$. ($\mathbf{x}$, $\mathbf{y}$) represents the input-output pairs corresponding to a batch of training data, and η is the learning rate. 'cache' denotes the intermediate values that are required to be stored for the backpropagation.

Initialization: R_0, m_1, t^*, c_1, m_2 for all the weights of our networks are initialized by sampling values from the distributions presented in fig. 2.6 and Table. 2.1. The initial mean model state ($\mathbf{t}$) and RTN state for all devices ($\mathbf{X}$) is set to 0. The pink-noise states ($\boldsymbol{\omega_r}$s) are generated as mentioned in APPENDIX 2

Inputs: ($\mathbf{x}$, $\mathbf{y}$), η, $\mathbf{R_{BL}}$, $\mathbf{R_{BLb}}$, $\mathbf{t}$, $\mathbf{X}$, , $\boldsymbol{\omega_0}$,..., $\boldsymbol{\omega_{pole}}$, $\boldsymbol{m_1}$, $\boldsymbol{c_1}$, $\boldsymbol{t^*}$, $\boldsymbol{m_2}$, $\boldsymbol{R_0}$, $\boldsymbol{a}$, P_{high}, P_{low}, α, $pole$.

Outputs: $\mathbf{R_{BL}}$, $\mathbf{R_{BLb}}$,$\mathbf{t}$, $\mathbf{X}$, $\boldsymbol{\omega_0}$,..., $\boldsymbol{\omega_{pole}}$

$\boldsymbol{W}^{real} \leftarrow \log_{10}\left(\boldsymbol{R}_{BL}/\boldsymbol{R}_{BLb}\right)$
$\boldsymbol{W}^{bin} \leftarrow \text{Sign}\left(\boldsymbol{W}^{real}\right)$
$\hat{\mathbf{y}}, \text{cache} \leftarrow \text{Forward}\left(\mathbf{x}, \boldsymbol{W}^{bin}\right)$
$C \leftarrow \text{Cost}(\hat{\mathbf{y}}, \mathbf{y})$
$\partial_{\boldsymbol{W}} C \leftarrow \text{Backward}(C, \hat{\mathbf{y}}, \boldsymbol{W}^{bin}, \text{cache})$
$\boldsymbol{U} \leftarrow \text{Optimizer}(\partial_{\boldsymbol{W}} C)$
$\boldsymbol{n} \leftarrow \text{HardTanh}(\min = -pole, \max = pole, \text{Int}(\eta \boldsymbol{U}))$
for n in $\boldsymbol{n}$ **do**
    if $n \geq 0$ **then**
        $R_{BL}, t, X, \omega_0, \ldots \leftarrow \text{RESET}(n, t, X, T, \omega_0, \ldots, m_1, c_1, t^*, m_2, R_0, a, P_{high}, P_{low}, \alpha, pole)$
    else
        $R_{BLb}, t, X, \omega_0, \ldots \leftarrow \text{RESET}(|n|, t, X, T, \omega_0, \ldots, m_1, c_1, t^*, m_2, R_0, a, P_{high}, P_{low}, \alpha, pole)$
    end if
end for
return $\mathbf{R_{BL}}$, $\mathbf{R_{BLb}}$, $\mathbf{t}$, $\mathbf{X}$, $\boldsymbol{\omega_0}$,..., $\boldsymbol{\omega_{pole}}$

2.5 Neural Network Simulation Results

2.5.1 The tasks and the architecture

We now test our device model, integrated into PyTorch, on two pattern recognition tasks. First, we train a fully connected (FC) BNN with one hidden layer of 3,000 units for solving the MNIST handwritten digit recognition benchmark. We then train a convolutional BNN to solve the CIFAR-10 object-recognition task. The architecture uses 3x3 kernels for convolutions (Conv), and 2x2 for MaxPool (MP) and reads: [Conv384, Conv384, MP, Conv768, Conv768, MP, Conv1536, Conv512, MP, FC(1024-1024-10)]. Figs. 2.11(a) and (b) show PyTorch simulations of the training process of binarized neural network for the MNIST and CIFAR-10 tasks, respectively. Test accuracies of 98% and 90% on the MNIST and CIFAR-10 tasks, respectively, were achieved without device simulations (ideal floating-point synapses). Including the full device simulation in the BNN training simulation makes it four times slower, the bottleneck being the sequential generation of pink noise.

2.5.2 Impact of imperfections

Incorporating the ReRAM model allows testing of how various aspects of the ReRAM imperfections affect the training performance. We first performed simulations, including the device model, but where the noise and the D2D variability were artificially deactivated (see Figs. 2.11(a)-(b)). We observe that the network can reach the baseline accuracy for both tasks. Thus, our BNN scheme is robust to the non-linearity of the devices, which is a major advantage with regard to non-binarized techniques [150]. Also, this result highlights that the conversion of floating-point updates to a discrete number of pulses had little effect on the final accuracy.

Figs. 2.11(a) and (b) also show that upon the introduction of noise(both RTN and pink) only, a point accuracy degradation of 1% and 2.5% for the MNIST and CIFAR-10 tasks is obtained. Adding D2D variability, the respective degradation of point accuracies are 3% and 10%. Also, to identify the impact of the noise independently, we performed simulations with only the noise components artificially deactivated. For MNIST, we find a point degradation of 0.3%, whereas, for the CIFAR-10, it is 10%. For both tasks, the inclusion of the D2D variability, therefore, caused degradation of test accuracy, although it is more prominent in the CIFAR-10 task (Figs.2.11(a)-(b)).

These results highlight that neural networks have the potential to fully benefit from the advantageous properties of weak RESET (progressivity, high endurance) without suffering from its high level of fluctuations. Also, via this kind of modeling, we explored the effects of D2D variability, noise, and non-linearity in greater detail than is possible with only experimental studies.

2.6 Conclusion

In this work, we presented a model of the weak RESET behavior of HfO_x ReRAM and its fluctuations and its integration within a deep learning framework for simulations of hardware neural networks on GPUs. The results suggest the outstanding potential of the weak RESET regime in such conditions. This work also explores the various aspects of ReRAM device imperfections on neural network performance.

Using the proposed framework, future work will investigate the design of more advanced neural networks on difficult tasks and how neural network design can be optimized for robustness to the fluctuations of ReRAM technology. Our modified PyTorch optimizer could also be adapted to all kinds of emerging devices considered for neuromorphic applications.

Chapter 3

Implementation of BNN inference immune to circuit-based constraints

> One of the basic rules of the universe is that nothing is perfect. Perfection simply doesn't exist.....Without imperfection, neither you nor I would exist.
>
> Stephen HAWKING

In the previous chapter, we explored the possibility of learning neural network models using noisy memristive devices as synaptic weights. Although we successfully demonstrated that the backpropagation algorithm could cope with non-ideal behavior, our focus was primarily on memory while assuming that all other computations occurred perfectly.

However, in this chapter, we take a systems-level approach to investigate the implementation of binarized neural networks (BNNs), which involves considering not only memory but all other circuits involved in the computation. Our goal is to assess the impact of various imperfections on inference. We focus on implementing neural networks that are already pre-trained with a set of weights programmed into memory, using our circuits for inference.

We present two different studies that employ complementary differential resistive devices as synaptic weights for a BNN. We begin by highlighting the different mathematical operations required for inference and emphasizing aspects that cause errors in the inference of the BNNs. The two projects presented in this chapter are collaborative works. The fabrication of the circuits was done at Université Grenoble Alpes (CEA-LETI, Grenoble), and the design and electrical characterizations were done by Pr. Jean-Michel Portal, Fadi Jebali, and Dr. Mona Ezzadeen. The author of this thesis performed the neural networks simulations with the experimentally measured and modeled errors and also did the inference result-related analyses. The work in this chapter culminated in the realization of two journal articles, one of which is under preparation (titled "Powering AI at the Edge: A Robust, Memristor-based Binarized Neural Network

with Near-Memory Computing and Miniaturized Solar Cell"), and the other is under review (titled "Implementation of Binarized Neural Networks Immune to Device Variation and IR Drop Employing Resistive RAM Bridges and Capacitive Neurons"). This chapter is adapted from both of these articles.

3.1 Circuits and Binarized Neural Networks

In non-binarized artificial neural networks (ANNs), the inference step involves computing the output neuronal activations y_i for a given layer, based on the synaptic weights W_{ij}, inputs a_j, and batch normalization threshold Δ_i and scaling parameter σ_i. The inference equation is given by:

$$y_i = f_{act}\left(\frac{\sum_j W_{ij} \cdot a_j - \Delta_i}{\sigma_i}\right). \tag{3.1}$$

Here, f_{act} is the activation function, and Δ_i and σ_i represent the threshold and scaling parameters, respectively, which are related to the batch normalization operation. These parameters are pre-determined using the running mean and running standard deviation calculated from the training set.

Traditional implementations of ANNs with memristors rely on Ohm's and Kirchoff's laws for multiplication and accumulation (MAC), respectively. However, this approach is highly susceptible to resistance variability and IR drop in the connecting wires, which can lead to performance degradation. This is because the values being multiplied or added can theoretically take any real value, and even small imperfections in the values can result in significant deviations during calculation.

In contrast, the activations and the weights of a BNN are all binarized to two values, +1 and -1. This quantization greatly reduces the computational complexity and has two major implications in the context of the hardware implementation of neural networks. Firstly, since we are bound to only two values for the weight, we can map our device resistance to these values in a manner that makes it more robust to noise. As a simple example, we can set a threshold resistance value, and if the device resistance is higher than that, we refer to that weight as +1; otherwise, it is -1. This quantization ensures that our weight values are less affected by small fluctuations in resistance as long as they remain above or below the threshold value. Secondly, the MAC operations can be replaced by simple logic gates as we are dealing with only two values. This is beneficial from the point of view of energy since in the implementation of ANNs, although Ohm's law and Kirchoff's law take care of the MAC operation, specialized readout circuitries are needed for them to function effectively. These specialized circuits, which often involve analog-to-digital conversions, are power-hungry and have a large area overhead, and thus are not desirable for the hardware-based implementation of neural networks.

Let us look at the various mathematical operations in BNN and how simple electronic cir-

cuits can implement them. For the multiplication between the binarized weights W_{ij}^{bin} and activation a_j, since each of them can take only two values, there can be a total of four combinations.

Weight	Input	Output
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

If we replace -1 with 0, this would give us the exact truth table as an XNOR gate, allowing us to use the XNOR gate to implement the multiplication. In the same vein, the accumulation process involves counting the total number of 1's amongst all the outputs, which are either 1 or 0. This is the population count or the popcount (PC) operation, and we shall see in this chapter how this can be done using energy and area-efficient circuitry.

After the MAC, we need to consecutively apply the batch normalization and the activation function. Since the outputs are also binary, the activation function is the sign function. Another point to be noted here is that since we apply the scale-invariant sign function, we only need to use the threshold value since the scaling has no impact on the output. Effectively, this means that for batch normalization, the PC value needs to be compared with the threshold value T_i. Then the sign function would output +1 if the MAC value is higher than the threshold and -1 otherwise. This can be achieved in a circuit by using a comparator whose one end gets the PC, and the other end gets the corresponding value of T_i. Putting all of this together, we find the final equation for the output y_i to be

$$y_i = sign(POPCOUNT(XNOR(W_{ij}^{bin}, a_j) - T_i)). \tag{3.2}$$

However, even such implementation is not totally immune to imperfections; the occurrence of errors pays the cost of energy efficiency. As we shall see, the true strength of BNNs lies in the fact that they are quite robust to these issues.

3.1.1 Imperfections in inference circuit

The imperfections leading up to an error can be categorized into two types; the first type relates to the variabilities present in the resistance of the memory devices. The second source is the variability or the non-ideal behavior of transistors. To ensure robust behavior from our memory devices, we program them in a 2-transistor, 2-resistor (2T2R) arrangement instead of a 1T1R arrangement. In the projects we pursued, the devices are programmed in a complementary fashion in LRS-HRS or HRS-LRS pairs. Noise in such devices can invert this pairing and result in a different weight than what was intended.

In a 2T2R cell, there are two access transistors, each connected to a Resistive random access memory (ReRAM) device in a complementary fashion. The relative states of the two devices

represent either the +1 or the -1 state of the synaptic weight. The two resistances are called R_{BL} and R_{BLb}, as they connect to the bit line (BL) and bit line bar (BLb) of the circuit, and if the resistance pairs are programmed to be in HRS-LRS state, the synaptic weight has a value of +1, and similarly, it encodes the value -1 for the LRS-HRS pair.

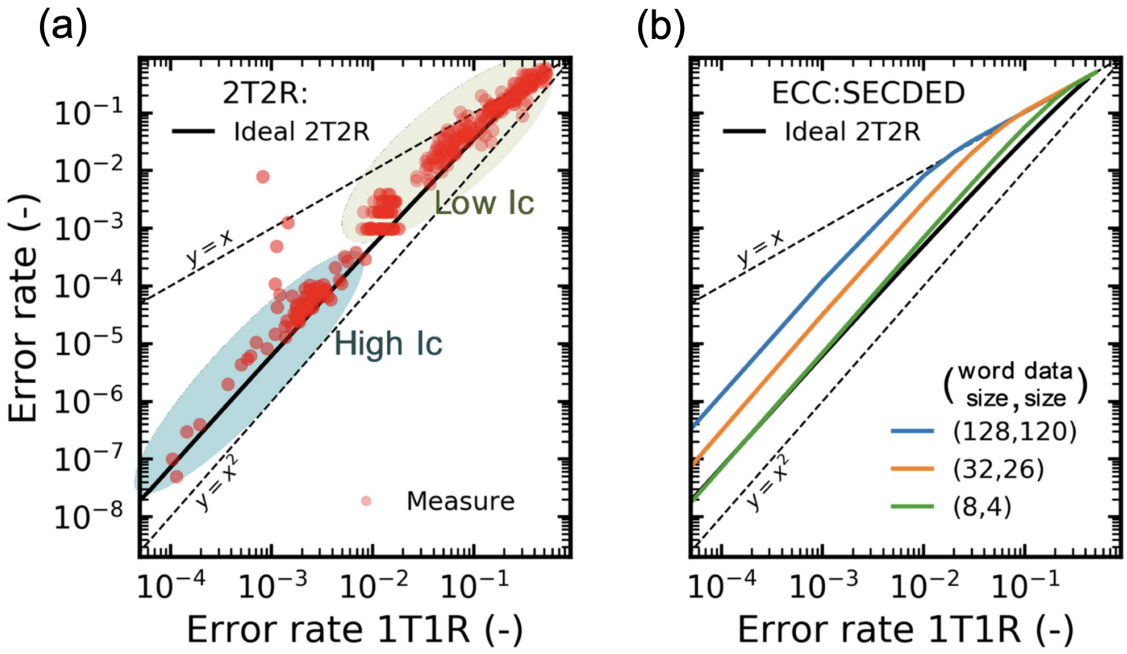


Figure 3.1: Comparison of 2T2R robustness to 1T1R and ECCs. (a) Experimentally measured bit error rate of a 2T2R array as a function of the bit error rate obtained with individual (1T1R) ReRAM devices under the same programming conditions. (b) The Bit error rate obtained with Single Error Correction Double Error Detection (SECDED) ECC as a function of the error rate of the individual devices for different word and data sizes (Adapted from [157]).

The superiority of the 2T2R approach, in terms of robustness to errors, is shown in fig. 3.1 where the bit error rate in a test chip with HfO_x-based ReRAMs is presented along with the error rates of a 1T1R cell under the same conditions of programming [157]. In fig. 3.1 (a), the red experimental data points lie lower than the $y = x$ line signifying that we have fewer errors in the 2T2R cells. Fig. 3.1 (b) also shows the comparison with an ECC where a Single Error Correction Double Error Detection (SECDED) ECC code is used, and it approaches the ideal 2T2R behavior for a word size of 8 and data size of 4. However, this type of ECC requires decoding circuits that comprise hundreds to thousands of logic gates, increasing the area and energy costs significantly.

The reason why the 2T2R is more robust to errors can be understood as follows. In the 1T1R architecture, the resistance is only compared with respect to a single reference resistance, whereas in 2T2R, we have two resistances programmed to opposite resistance states, and only their relative value dictates the state. An error occurs when the fluctuation of the resistance

drives it to be in the other regime. It is much more likely for a single resistor's fluctuation to cross the reference value than for two resistors programmed at two extremes to cross each other. This is what makes the 2T2R more robust than the 1T1R memory cells.

The other source of erroneous behavior stems from the non-ideal behavior of transistors. Transistor mismatch is a type of variability that is one of the most significant contributors to errors in such systems. The mismatch is a type of transistor variability that arises due to the natural variations in the electrical properties of multiple transistors that are intended to be identical. Mismatch can occur in various electrical properties of the transistors, such as their threshold voltage, channel length, or mobility. These variations cause differences in the way that the transistors operate and can result in performance differences in circuits that use these transistors.

In this chapter, we present two different circuit-implementation of BNNs and investigate the impact of errors on inference accuracy. The remaining sections of this chapter will be organized as follows: first, we describe the details of the circuit being used to implement BNNs, specifically highlighting the cause and sources of errors. Then, we present the experimental error measurements and the associated error model that gives us a mathematical handle that is used for our neural network inference simulation. We finish each section showing the degradation in performance with these errors and take a deeper dive into the types of errors and how they relate to the circuit attributes.

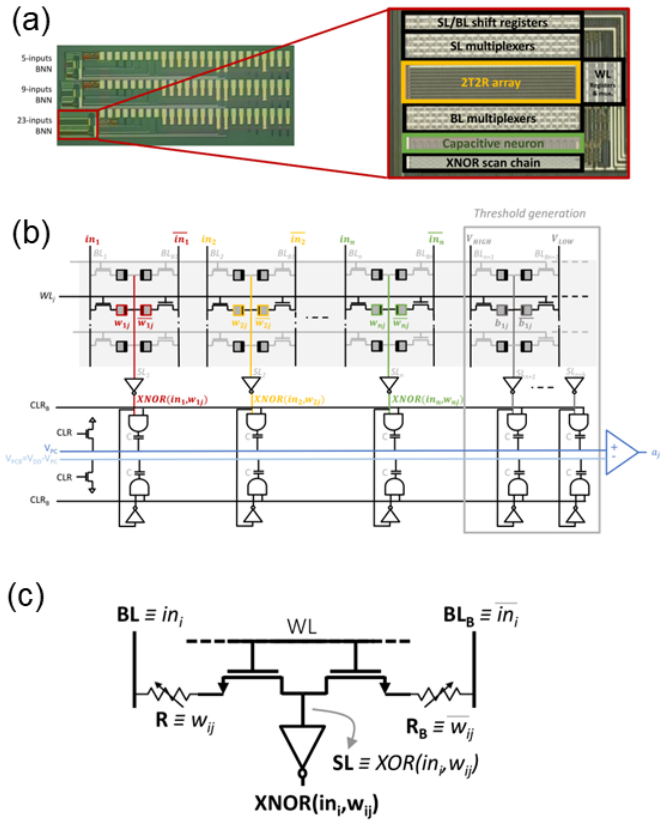


Figure 3.2: The BNN circuit implementation. (a) Optical microscopy photograph showing the BNN test chip, which includes implementations of neurons with 5, 9, and 23 inputs. The inset shows a detailed view of the 23-input neuron version. (b) The global architecture of the BNN circuit used in the test chip features 2T2R ReRAM cells that employ complementary coding to ensure robust XNOR operation. The popcount and threshold use fully differential coding with capacitive bridges to enhance the comparison margin. (c) Schematic of the proposed bit cell. The weights stored in the ReRAM 2T2R cell, and the activation input, applied on the BL/BLB, are both coded in a complementary fashion. An inverter gate generates the final XNOR value at the bottom of the SL.

3.2 Implementation of BNN with ReRAM bridges and capacitive neurons

3.2.1 Circuit

In this work, the overall implementation of BNN is based on a fully-differential capacitive neuron with 2T2R synapses wired in a resistive bridge configuration. A test chip was fabricated in a 130 nm CMOS technology with co-integrated ReRAM memory cells in the BEOL between metal layers four and five, with the aim of computing equ. 3.2 in an efficient manner. In fig. 3.2 (a),

a micro-photograph of this circuit is shown with the neuron having 23 inputs. The circuit has two main components: a ReRAM array that stores the weights and a capacitive neuron circuit at the bottom of this array. Also, there are shift registers that control multiplexers, which give direct access to the memory cells for the purpose of electrical characterizations. To extract the error rates, a scan chain retrieves the XNOR values in parallel and outputs them serially. The capacitive divider bridge is designed with capacitors of capacitance 105 femtofarad.

Equ. 3.2 has four main components: the XNOR between the weight and the input, the population count (popcount), the threshold, and finally, the sign function. We now discuss their circuit implementations individually.

3.2.1.1 In-memory XNOR operation

The weights of a neuron are contained in a single row of a ReRAM array, as illustrated in fig. 3.2 (b). Our work utilizes a 2T2R (two transistors - two resistors) ReRAM cell architecture (fig. 3.2 (c)), where the cells are connected in series to form a resistive bridge. The synaptic weights, W_{ij}, are encoded in a complementary manner in the two ReRAM cells of the 2T2R structure. Depending on the synaptic weight value, either the left (R) or the right (RB) ReRAM cell is programmed to a High Resistance state, while the complementary ReRAM is programmed to a Low Resistance state. This causes the source line SL to be pulled towards either the left or the right bit line, depending on the synaptic weight value. The input neuron values are presented in a complementary fashion on the two bit lines, meaning that depending on the input neuron value, either the left or the right bit line is at the lowest voltage. As a result, the source line naturally follows an exclusive OR (XOR) between the weight and the neuron input, which allows the memory array to perform XOR operations directly within memory. Finally, the source line voltages are used as inputs to the inverter gates located at the bottom of each SL to output the XNOR values.

The in-memory XNOR operation is highly reliable and is only expected to fail in rare circumstances, specifically when the device programmed into a low-resistance state has a higher resistance than the device programmed into a high-resistance state. However, this situation has a very low probability of occurring since both devices would have to be improperly programmed, as mentioned above. Additionally, the nonlinearity of the inverter amplifies the signal, leading to clean binary outputs, which enhances the robustness of our approach to variability.

Compared to other implementations, a unique advantage of this approach is that the two devices are connected in series, ensuring that the current paths in the memory array always include a high-resistance device [157, 166]. As a result, the in-memory XNOR operation relies on a low current, regardless of the input and weight values, making our approach naturally immune to IR-drop effects.

3.2.1.2 The near-memory Popcount, threshold, and sign operations

The popcount and the sign operation after thresholding are performed near-memory by utilizing a switched-capacitor addition circuit and a comparator, as illustrated in fig. 3.2 (b), following an approach that is inspired by the SRAM-based works of [170, 171]. The switched-capacitor circuit is highly energy-efficient, as it doesn't require direct current to be applied, unlike in in-memory MAC designs utilizing Kirchoff's current law, and instead only consumes energy when the capacitors are switched.

The popcount circuit, which is based on a fully differential approach with two capacitive bridges connected to complementary inputs, ultimately leads the voltages of two capacitive bridges to be at

$$V_{PC} = \frac{m}{n} V_{DD} \tag{3.3}$$

and

$$V_{PCB} = V_{DD} - \frac{m}{n} V_{DD}, \tag{3.4}$$

where m is the popcount value, which is the number of XNOR outputs that are equal to one, n is the total number of XNOR outputs connected to each capacitive bridge, and V_{DD} is the source voltage. An activation a_j is set to one by the comparator only when more than half of the XNOR values are equal to one (i.e., $m > \frac{n}{2}$). Therefore, without any further modification, the circuit implements a neuron (equ. 3.2) with a threshold T_j of $\frac{n}{2}$.

Statistically, for neural network inference, it has been shown that a threshold-setting capability of ±5% around the mean value of $\frac{n}{2}$ is necessary and usually enough to achieve high accuracy. To provide this capability, a total of $b = 2 \times \lfloor 0.05n \rfloor$ capacitors were added to each bridge in a complementary manner. These additional capacitors are connected to the source line of additional columns in the ReRAM array, where the thresholds are programmed. A point to note in equ. 3.3 is that the voltage V_{PC} is inversely related to the total number of XNOR outputs, so for high values of n, this voltage can be too low for the comparator.

3.2.2 Measurement of error and error model

To validate this circuit, experimental measurements were done on the 23-input circuit for different values of the read voltages and the compliance current, which dictates how low the LRS shall be. The XNOR error percentages were calculated, and it was seen that there were no errors for read voltages higher than 0.3 V and compliance currents larger than 110 μA. For the lower compliance current and read voltage values, the resistance states were measured, and it was seen that in this regime, the LRS and HRS states are widely overlapping, causing errors. A remarkable thing to notice here is that even for low values of the preactivation Δ (which is the difference between the Popcount and the threshold), the circuit had no errors. This is because the voltage difference between the two capacitive bridges always remained adequately large, and there was no output neuron activation error.

However, in circuits with larger input numbers and low Δ values, errors may arise due to low voltage differences at the comparator inputs. To address this, extensive Monte Carlo simulations were performed on circuits with BNN sizes up to 513 input neurons and clock periods ranging from 4 to 20 ns, covering the full range of possible popcount and threshold combinations (preactivation Δ values). These simulations considered both global and local sources of variability, including mismatch, at three standard deviations, with 1,000 runs performed for each case. To account for ReRAM variability, the source line measured distributions were directly injected, which correspond to error-free XNOR operations at a compliance current of 200 μA and a read voltage of 0.6 V at the XNOR inverter's inputs.

Fig. 3.3 (a), (b), and (c) illustrate the extracted error distributions for the 33, 257, and 513-input neurons and their corresponding Gaussian fits. The output remains error-free for neurons up to 33 inputs with a clock period of 6 ns or higher, which aligns with our measured results. The smallest Δ value needed for 33 inputs corresponds to a voltage difference of 34 milliV. However, for larger neuron sizes, the smallest voltage difference decreases, reaching only 2 mV for 513 inputs, leading to higher error rates for small Δ values. Nonetheless, the Gaussian error distributions stay narrow for clock periods equal to or greater than 6 ns. The fig. 3.3 (d) displays the standard deviation values obtained for various neuron sizes and clock periods. It is observed that the standard deviation decreases as the clock period increases since more time is available for the clear and capacitive divider voltage settling. Based on these results, we set our minimum clock period to a value of 6 ns.

With these results, we developed a mathematical model to generate the error distribution for an arbitrary number of neurons. This is necessary since the number of neurons in our neural network is typically much higher than 513.

Consider a neuron a_j with N inputs (including the bias terms), where n_1 of each input are expected to lead to a one XNOR value. We focus on the case where $n_1 \leq \lfloor N/2 \rfloor$ for simplicity, such that a_j is expected to be one (or derivation can be easily adapted to the other case). We denote p as the probability of a single 2T2R-based XNOR operator giving an erroneous output. We extracted p for various programming conditions and read voltages from the experimental measurements. We obtain P({f_0=i}) the probability of having i XNOR outputs turning from a correct zero state to an erroneous one state, and P({f_1=j}) the probability of having j XNOR outputs turning from a correct one state to an erroneous zero state, using binomial laws

$$P(\{f_0 = i\}) = \binom{N - n_1}{i} \times p^i \times (1-p)^{N-n_1-i} \tag{3.5}$$

$$P(\{f_1 = j\}) = \binom{n_1}{j} \times p^j \times (1-p)^{n_1-j}. \tag{3.6}$$

We also introduce P({CN(x)=1}), the probability of the capacitive neuron (CN) giving an output of one when x XNOR outputs equal to one, obtained for various neuron sizes and clock

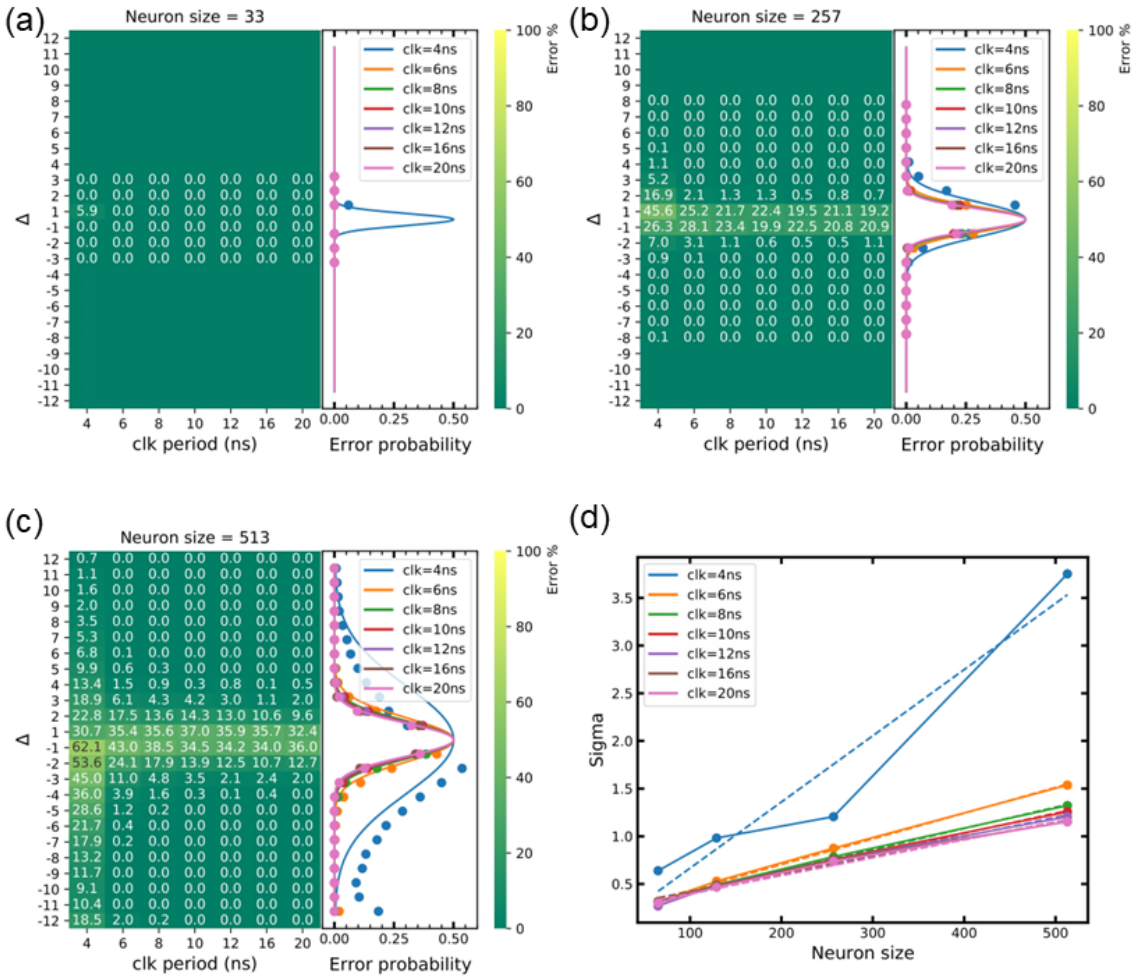


Figure 3.3: Monte Carlo simulations of neuron operation. (a) - (c) Simulated neuron error rate, as a function of the preactivation (Δ) and of the clock period, for neuron sizes of (a) 33 inputs, (b) 257 inputs, and (c) 513 inputs (not counting the bias terms). Error rates are plotted with a Gaussian fit. These simulations include transistor variability using the foundry design kit and the resistance distributions with a read voltage of 0.6 V and compliance current of 200 μA. (dd) The standard deviation of the Gaussian fit of the simulated neuron error rate for different neuron sizes and clock periods

periods from the Gaussian distributions of fig. 3.3 (d). Then, we can compute the probability of the neuron output a_j being equal to one instead of zero by

$$P(\{a_j = 1 | n_1 \leq \lfloor N/2 \rfloor\}) = \sum_{i=\lceil N/2 \rceil - n_1}^{N-n_1} P(\{f_0 = i\}) \sum_{j=0}^{\min(n_1, n_1+i-\lceil N/2 \rceil)} P(\{f_1 = j\}) \times P(\{CN(n_1+i-j) = 1\})$$
$$+ \sum_{i=0}^{\lfloor N/2 \rfloor} P(\{f_0 = i\}) \sum_{j=\max(0, n_1+i-\lfloor N/2 \rfloor)}^{n_1} P(\{f_1 = j\}) \times P(\{CN(n_1+i-j) = 1\}) \tag{3.7}$$

Using this model, we can compute the error probabilities for any value of N, for all the

different values of n_1, which play the role of Δ.

3.2.3 Neural network inference

To evaluate the performance of our BNN circuit at the neural network scale, we incorporated the error model introduced in the previous section (and described by eq. 3.7) into the PyTorch [172] deep learning simulation framework. Inferences are performed for multiple programming conditions, read voltages, and clock periods on the MNIST handwritten digit recognition and the CIFAR-10 image recognition datasets.

During the neural network inference, we perform the MAC operation normally and then take the probability value computed from the error model corresponding to the preactivation Δ values. This is the probability of having an error in that particular output. For example, for a Δ of value -2, we get an error probability of 0.1 for a particular clock period from our error model; then, the sign of that output would be flipped with a probability of 0.1.

Fig. 3.4 (a) shows the obtained test recognition rate, along with error-free baselines. For the MNIST task, negligible accuracy degradation is reported for all compliance current values. Even for the most critical configuration (a 6 ns clock period, a read voltage of 0.2 V, and a compliance current of 40 microamperes), the accuracy degradation is only 0.2 point percent for a baseline accuracy of 98.3%.

CIFAR-10 image recognition is a much more challenging task. Fig. 3.4 (b) shows the accuracy loss, compared to a software precision baseline of 90.6%, for various conditions. The accuracy loss (in percentage points) is low, although it is higher than in the MNIST case. Even for a read voltage of 0.2 V and a standard compliance current (110 microamperes), we observe only 0.9 point percent precision loss for a clock period of 20 ns (1.4 point percent for a clock period of 8 ns and 2.3 point percent for a clock of 6 ns). Overall, the compliance current has a remarkably low impact on the accuracy: only a truly low value of 40 microamperes substantially degrades the accuracy.

Errors are not considered for the first and last layers since the respective inputs and outputs of those layers are not binarized. For the MNIST task, we used a fully connected network with three hidden layers of 1,025 neurons each. For the more challenging CIFAR-10 task, we used an architecture based on the binarized Visual Geometry Group (VGG) structure, consisting of six convolutional layers followed by three fully connected layers [173].

In conclusion, this study implemented a BNN circuit based on a 2T2R ReRAM array with a capacitive output neuron. Experimental measurements and computer simulations show the robustness of this approach to imperfections related to both ReRAMs and transistors. Neural network simulation for the MNIST and CIFAR-10 tasks shows that the degradation in accuracy is low even for low compliance current and short clock periods. These neural network simulations reveal that due to the intrinsic tolerance of binarized neural networks to errors, it can be favorable to choose low read voltages and programming currents, as they respectively promote energy efficiency and device endurance, with low impact on network-level accuracy.

For a clock period of 6 nanoseconds, our 513 inputs BNN circuit provides an appealing peak energy efficiency of 96 TOPS/W and 449.3 TOPS/W for, respectively, a 130-nanometer and a 22-nanometer implementation.

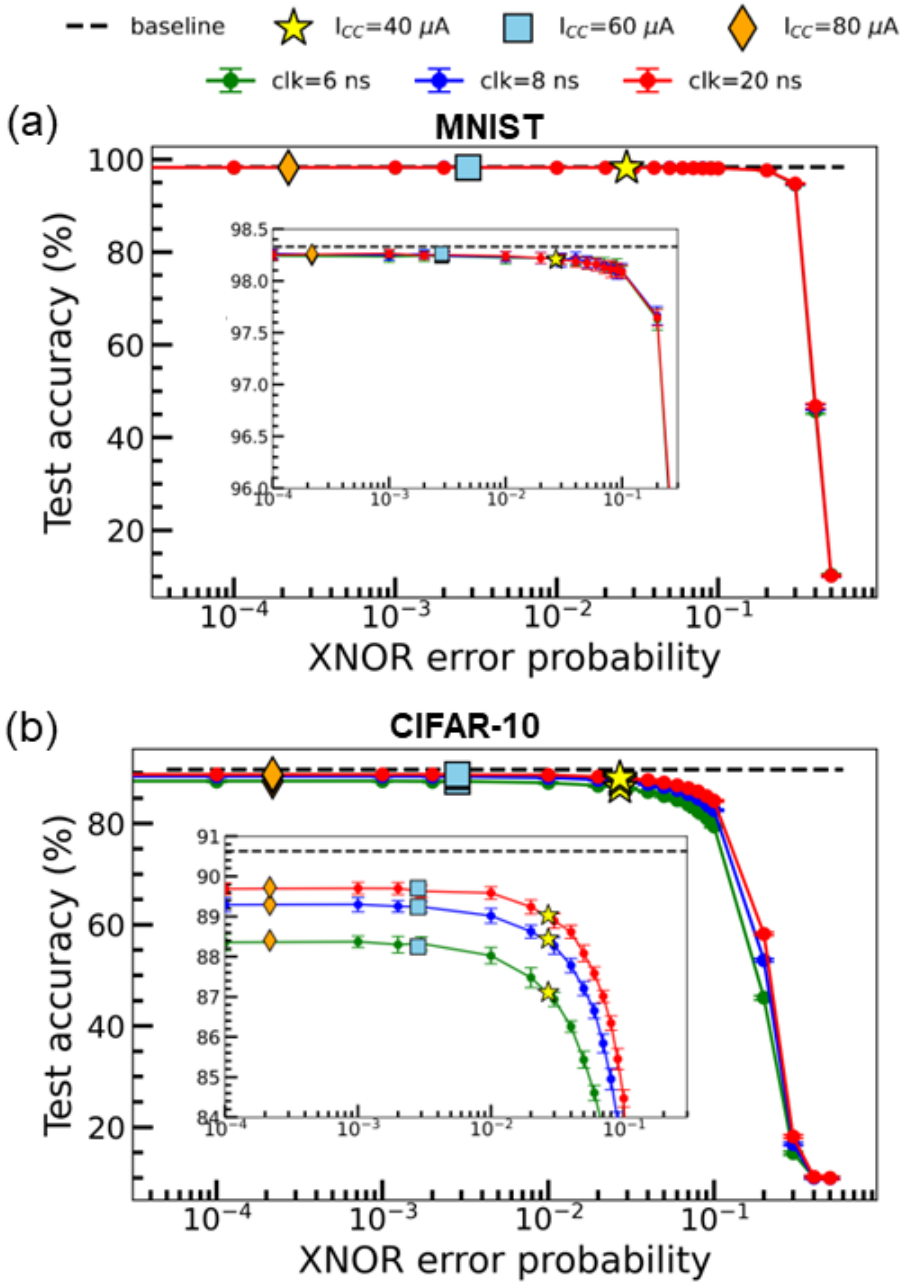


Figure 3.4: Neural network simulation results with errors. Inference accuracy for the (a) MNIST and (b) CIFAR-10 datasets as a function of XNOR error probability for different clock periods. Markers indicate the inference accuracies for I_{CC}=40 microamperes, 60 microamperes, and 80 microamperes with V_{read}=0.3 V.

3.3 A self-powered memristor-based BNN

3.3.1 Extreme-edge AI

Artificial intelligence (AI) has become increasingly prevalent in embedded applications, including patient monitoring, building, and industrial safety [174]. To optimize security and minimize energy consumption resulting from communication, it is preferable to conduct the majority of data processing at the edge of such systems [175]. However, integrating AI into extreme-edge environments presents a challenge due to its high power consumption, which often necessitates its deployment to the "cloud" or "fog" [176, 177]. The use of memristor-based systems offers a promising solution to this problem, as they can significantly reduce AI energy consumption [139, 178]. This makes it feasible to create self-powered edge AI systems that can derive their energy from the environment rather than requiring batteries.

As highlighted in the previous section, the most energy-efficient memristor-based AI circuits rely on analog-based in-memory computing to perform the fundamental operation of neural networks,(MAC) [120, 125, 179]. However, this concept is difficult to implement in practice due to the high variability of memristors, imperfections of analog CMOS circuits, and voltage (IR) drop effects. To address these challenges, memristor-based AI systems require highly complex peripheral circuits that are optimized for specific supply voltages [123, 126, 180–184]. Unfortunately, this requirement for stable voltage directly contradicts the characteristics of miniature energy harvesters such as tiny solar cells or thermoelectric generators, which produce fluctuating voltage and energy. As a result, the realization of self-powered memristor-based AI systems presents a significant obstacle.

In this study, we present a new approach to memristor-based binarized neural networks that can effectively handle power supply issues. We employed a hybrid memristor/CMOS process to design, fabricate, and test a circuit that includes four 8,192-memristor arrays, utilizes a 2T2R method to store synaptic weights, employs multiplication within a robust differential sense amplifier, and applies a simple digital circuit for accumulation. To demonstrate the circuit's resilience, we connected it to an III-V semiconductor-based solar cell optimized for energy harvesting under low illumination conditions. Our findings reveal that the circuit performs as well as it does with a lab-bench power supply under illuminations higher than 1 sun. Even when exposed to illuminations as low as 0.08 suns, the circuit remains functional with a moderate decrease in neural network accuracy. Our circuit is capable of adapting to the power supply by automatically switching between exact and approximate computing, indicating its versatility in handling power supply variations.

3.3.2 Circuit

Since this is another implementation of BNN, the computations necessary for inference are exactly the same as equ. 3.2, and this circuit also needs to be able to perform the same calcula-

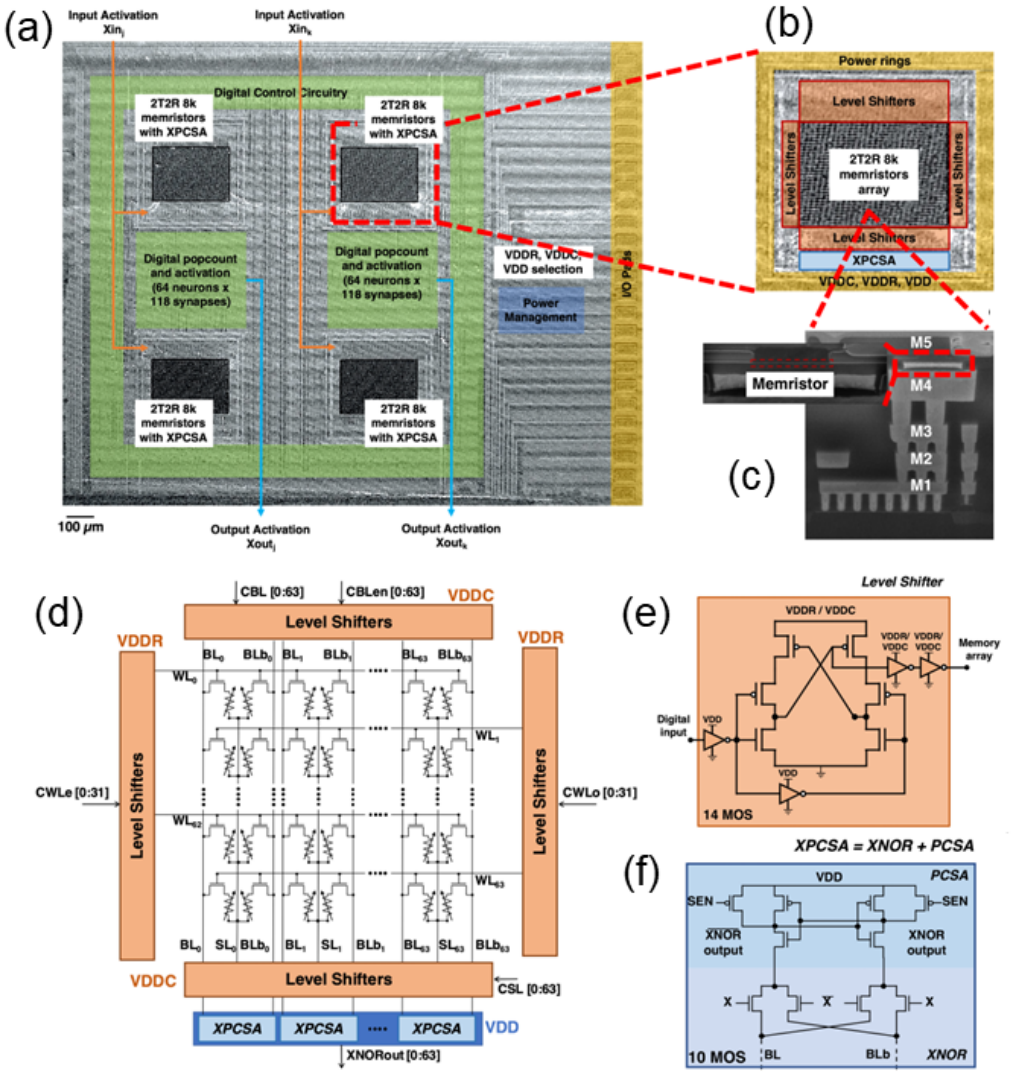


Figure 3.5: Overview of the memristor-based BNN circuit (a) Optical microscopy image of the fabricated die, showing four memory modules and their associated digital circuitry and power management unit. (b) Zoomed image of one of the memory modules. (c) Scanning electron microscopy of a cut of a hybrid CMOS/memristor circuit, showing a memristor between metal levels four and five. (d) Schematic of a memory module showing the co-located ReRAM and their access transistors, the level shifters for the columns and rows, and the XPCSAs connected to the ReRAM array. (e) Schematic of the level shifter, used for shifting digital voltage input to medium voltages needed during programming operations or nominal voltage during reading operations of the memristors. (f) Schematic of the differential pre-charge sense amplifier PCSA used to read the binary memristor states, with embedded XNOR function, to compose an XPCSA.

tions in or near memory. The memory array is similar to the implementation presented in the last section, with HfO_x-based ReRAM integrated with a low-power 130 nm CMOS process node shown in fig. 3.5 (a), (b) and (c).

The design choices in this study are aimed at achieving the most reliable operation in the presence of an unreliable power supply, following the differential strategy proposed by Hirtzlin in [157]. Our approach involves utilizing two memristors per synaptic weight, programmed in a complementary manner, with one memristor in a low resistance state and the other in a high resistance state (as depicted in fig.3.5 (d)). One obstacle is that the forming process necessitates voltages up to 4.5 V, while our CMOS process's typical voltage is just 1.2 V. To address this issue, we integrated level shifter circuits into the peripheral circuitry of the memory arrays (as shown in fig. 3.2 (e)), which are capable of withstanding high voltages. These circuits leverage thick-oxide transistors to increase the voltage of the on-chip signals that instruct the programming of memristors. Additionally, we have incorporated a dedicated logic-in-memory precharge sense amplifier (PCSA) to execute the multiplication, which simultaneously reads the state of the two memristors representing the weight and performs an XNOR with its input (as illustrated in fig.3.5 (f) where X represents the input) [168]. Moreover, power supply voltage fluctuations affect both branches of the sense amplifier symmetrically, further enhancing the robustness of our design. Therefore, unlike other analog in-memory computing implementations that require finely controlled supply voltage, our approach eliminates the need for compensation and calibration circuits.

In this study, the resistances are connected in parallel, and the memory state of a cell is connected to their relative resistances. This differential approach provides increased circuit resilience by minimizing the impact of memristor variability. Specifically, even if the memristors deviate significantly from their programmed values, the sense amplifier will produce the correct output as long as the relative resistances of the two devices preserve their relative order in terms of magnitude.

As shown in fig. 3.5 (f), the reading of the weight and the subsequent XNOR operation is done in a single step with a circuit block that combines an XNOR gate and a PCSA circuit and is called the XPCSA. The XPCSA operates by pre-charging BL and BLb to a voltage VDD and then letting it discharge through their corresponding memristors and the XNOR layer. Due to the difference in resistances, the time constants of the discharge are different, and it allows the latch in the PCSA to acquire the XNOR value.

In our memory array having 64 rows, the first six rows contain the threshold value, where the topmost row encodes the most significant bit of this value. The rest 58 rows are used for the synaptic weights. First, the threshold value is loaded, and then the popcount operation is done by going through the rows and subtracting one from the threshold value whenever the XNOR value equals one. This is done using a decounter, and the final sign operation is performed by comparing the value of the decounter to zero, which yields the final binarized output. In this setup, with maximum voltage, the only source of errors is the variability of the ReRAMs or failures in their programming.

3.3.3 Divide-and-conquer mapping strategy

A resource-based constraint of our circuit is that it has only the provision to have 58 input rows (since the other 6 are dedicated to the thresholds). On the other hand, typically, the number of inputs in a neural network is much higher, and the value itself varies across different layers. In this section, we describe the Divide-and-conquer strategy that we use to map neural networks to our chip, taking into account this particular resource constraint of the array. Our solution involves partitioning the inputs and performing the popcount operation on each partition individually, producing a partial output. These partial outputs are then combined to obtain the final output of that layer. Fig. 3.6 shows the general idea of this concept, where (a) shows the ideal scenario where the number of rows in the array n_{rows} is equal to the number of inputs of that layer n_{inputs}. This is not realistic, and typically the number of rows is less in number and has a fixed value, whereas, for a real neural network, the input sizes of different layers are not the same. In fig. 3.6 (b), we show our proposed solution to this, where we divide our input into partitions, each with n_{rows} number of units. We compute each of their outputs individually and, finally, combine them to compute the final output. The specific implementation techniques differ between fully connected and convolutional architectures, and we provide a detailed account of these strategies below.

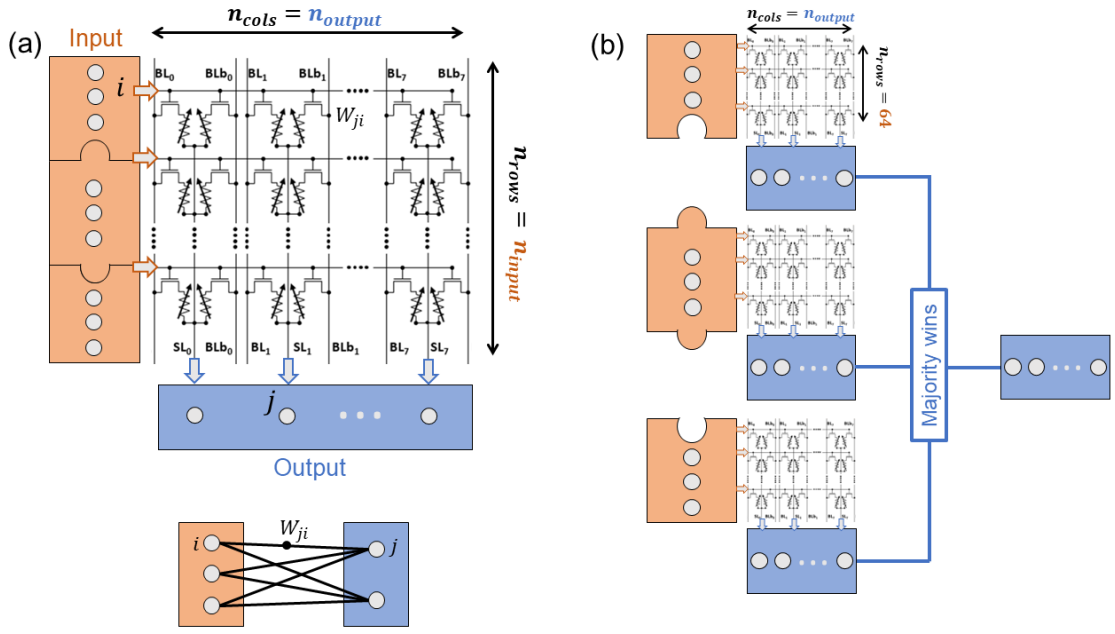


Figure 3.6: Illustration of the Divide-and-conquer mapping strategy. (a) The mapping of our network to the circuit when the number of rows n_{rows} and the number of inputs of the neural network n_{inputs} are the same. (b) Our Divide-and-conquer mapping strategy for a more realistic circuit with a fixed number of rows (64 is shown in this case).

Divide-and-conquer in fully connected architecture

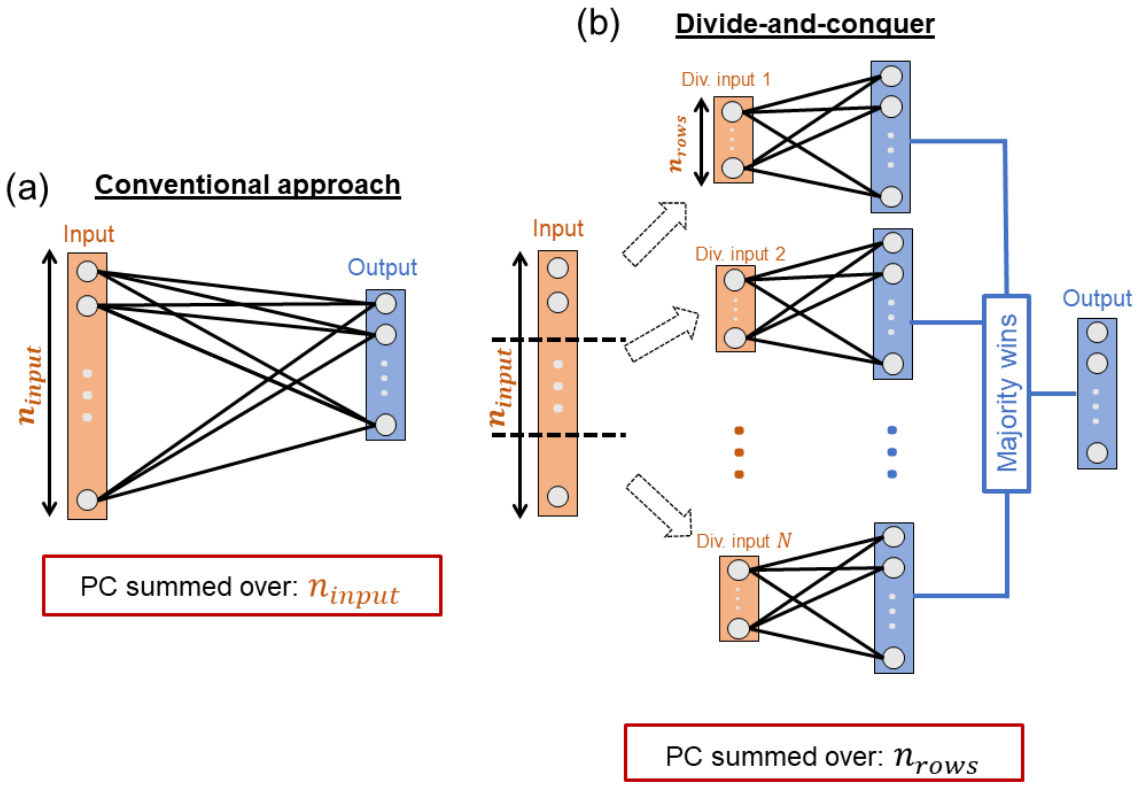


Figure 3.7: Schematic for the Divide-and-conquer strategy for a fully connected architecture. (a) A conventional fully connected layer of a neural network where the popcount operation is performed over n_{input} values of neurons. (b) Our Divide-and-conquer approach for the same neural network layer. The input layer is divided into N partitions with n_{rows} neurons in each, independently connected to output layers. These intermediate output layers are combined in a 'majority wins' function to produce the final output. Here, the popcount is over n_{rows} values.

The neural network operation of calculating the output from n_{input} number of inputs, without any circuit consideration, has the following form

$$X_{out,j} = \text{sign}\left(\sum_{i=1}^{n_{input}} XNOR\left(W_{ji}, X_{in,i}\right) - T_j\right). \tag{3.8}$$

In our case, the number of values over which we can perform the popcount operation is fixed by the number of rows present in our array (n_{rows}). Therefore, as shown in fig. 3.7, we divide the n_{input} into N partitions with n_{rows} units in each (hence, $N = \frac{n_{input}}{n_{rows}}$). If W^r_{ji}, $X^r_{in,i}$, and T^r_j represents the weights, inputs, and thresholds for the r^{th} partition, the output is then calculated as

$$X_{out,j} = \text{sign}\left(\sum_{r=1}^{N} \text{sign}\left(\sum_{i=1}^{n_{rows}} XNOR\left(W_{ji}^{r}, X_{in,i}^{r}\right) - T_{j}^{r}\right)\right). \tag{3.9}$$

In fig. 3.7, the summation of the intermediate outputs and the following sign function is represented by the 'majority wins' function: if the +1s are in the majority among the outputs, it yields +1 and -1 otherwise. Since we are combining the intermediate outputs in this manner, N must be an odd number; otherwise, some outputs could have an equal number of +1s and -1s. We achieve this by choosing n_{inputs} such that N is odd.

The Divide-and-conquer strategy is used for the first hidden layer with 1,102 neurons as input in the fully connected architecture. For our array, among the 64 rows, a total of six rows are reserved to represent the threshold value, leaving a total of 58 rows available for the input values, which leads to N being equal to 19.

The Divide-and-conquer mapping strategy keeps the total number of weights in the network the same as a conventional implementation. Still, it involves a loss of information as it performs the summation in parts, and we suffer a slight degradation in accuracy (98.0% to 97.2% on MNIST).

Divide-and-conquer in convolutional architecture

For convolutional neural networks, the popcount operation is performed differently. In our approach, we use filters with dimensions of $3\times3\times N_C$ to convolve over the input feature map, where N_C represents the number of channels. To address the resource constraints of our array, we partition the input feature maps along the channel dimension and apply the appropriate number of corresponding filters, as shown in fig. 3.8. Unlike in the case of fully connected neural networks, the number of channels in each partitioned feature map, denoted by N_{div}, is not equal to n_{rows}. Instead, owing to the 3×3 filter shape, N_{div} is given by the expression:

$$N_{div} = \left\lfloor \frac{n_{rows}}{9} \right\rfloor. \tag{3.10}$$

The total number of partitions is given by $N = \frac{N_C}{N_{div}}$, and the number of popcount operations performed in each block is $3 \times 3 \times N_{div}$. The intermediate outputs obtained from each division are combined in the same way as in the fully connected network to generate the output feature map. For the same reasons as the fully connected architecture, we choose the total number of filters N_C such that the total number of partitions N is odd.

For the convolutional (feature-extracting) part of our network, we used a value of $N_{div} = 6$ because of our n_{rows} size of 58 and eq. 3.10, and for the fully connected (classifier) part we used mapping as described in the previous section. Table 3.1 lists the number of partitions used for each of the layers:

In this architecture as well, the total number of filters, or in other words, learnable weights, are preserved as compared to an undivided architecture, although with some degradation of

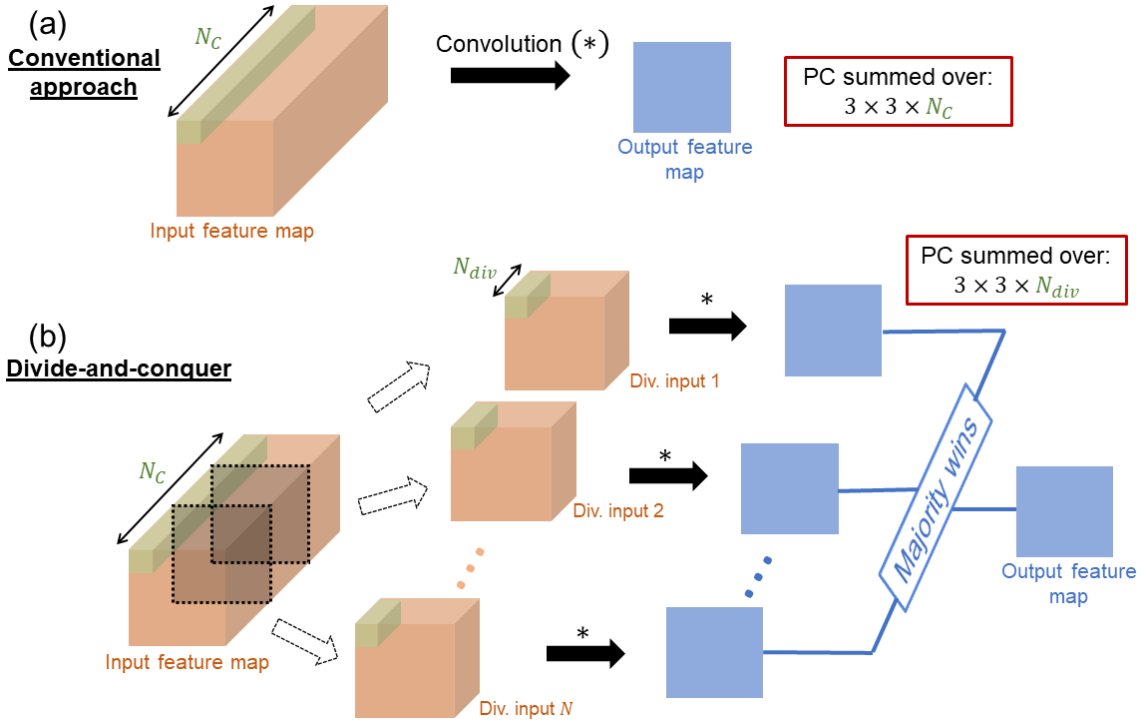


Figure 3.8: Schematic for the Divide-and-conquer strategy for a convolutional architecture. (a) In a conventional convolutional architecture, the input feature map with a channel-wise dimension of N_C is convolved with a filter of dimensions $3\times3\times N_C$ to produce the output feature map. Here, the popcount is done over $9N_C$ values. (b) In our Divide-and-conquer strategy, both the input feature maps and the filters are divided along the channel dimension into N partitions, each with N_{div} channels. Subsequently, the convolutions are performed on each of them, and the outputs are combined using a majority function to yield the final output. The popcount is performed over $9N_{div}$ values in this case.

accuracy (90.0% to 86.6%).

3.3.4 Error in the circuit inference

To verify the compatibility of our circuit with energy harvesters, we connected it to a miniature (5 mm ×5 mm) AlGaAs/GaInP heterostructure solar cell. This type of solar cell, fabricated following the procedure of [185], with a 1.73 eV bandgap, performs better than conventional silicon-based cells under low-illumination conditions, making it particularly suitable for extreme edge applications. Energy harvesters are usually connected to electronic circuits through sophisticated voltage conversion and regulation circuits. Here, to demonstrate the robustness of our system, we connect it directly to the solar cell. This demonstration is possible as the maximum voltage provided by our solar cell (1.25 V under 1 sun illumination) matches the nominal supply voltage of our CMOS technology (1.2 V), unlike silicon solar cells, whose maximum voltage is only 0.7 V and would require voltage conversion for use under low illumination.

Type of layer	N_C	N
Conv	198	33
Conv	354	59
Conv	738	123
FC	406×3×3	63
FC	1102	19

Table 3.1: The number of channels/neurons used in the different layers of our simulated convolutional network and the corresponding number of partitioned blocks in our Divide-and-conquer mapping strategy.

We measure the circuit's performance under the illumination of a lamp producing four different intensities: 8 suns, 0.8 suns, 0.36 suns, and 0.08 suns. Under the illumination of 1.5 suns, the circuit performs almost equivalently when powered by a 1.2 V lab bench supply. Fig. 3.9 shows the error probabilities of the circuit output neurons for the different illumination levels as a function of the preactivation Δ, which is the difference between the popcount and the threshold. No errors were found for $|\Delta| > 5$, and since the cause of these errors is the errors in the weights (ReRAM issue), the probability of an error is higher when the popcount and threshold values are similar. This is so because there is an error in the output when the expected output is +1, whereas the measured output is -1, or vice-versa, and this change of sign is more likely when the Δ value is close to zero, which is the crossover point between the two values. The measurements are shifted with respect to the $\Delta = 0$ because, in our circuit, this case is considered to have an output of -1, and in this case, the true point of symmetry of Δ is between 0 and -1. Another issue we observed is that the measurements are asymmetric, so we didn't fit a Gaussian as we did in the previous section. We used an exponential fit for the two branches separately and used the average value of the fit parameters to get the shown fits in fig. 3.9. We use this error model to simulate the real errors of our circuit for the neural network simulations.

3.3.5 Neural network inference

We now evaluate the performance of our circuit on neural networks using the Divide-and-conquer mapping scheme we presented earlier. To assess the accuracy of our hardware, we incorporated the error rates measured experimentally as a function of preactivation value and illumination (fig.3.9) into neural network simulations. Table 3.2 lists the obtained accuracy on a fully-connected neural network trained on MNIST and a convolutional neural network trained on CIFAR-10. Remarkably, the MNIST accuracy is hardly affected by the errors in the circuit: even under very low illumination of 0.08 suns, the MNIST accuracy drops by only 0.7 percentage points. Conversely, bit errors in our circuit significantly reduce the accuracy of the more challenging CIFAR-10 task. Under 0.08 suns, the accuracy drops from the software baseline of 86.6% to 73.4%. The difference with the MNIST is that neurons tend to have low preactivation

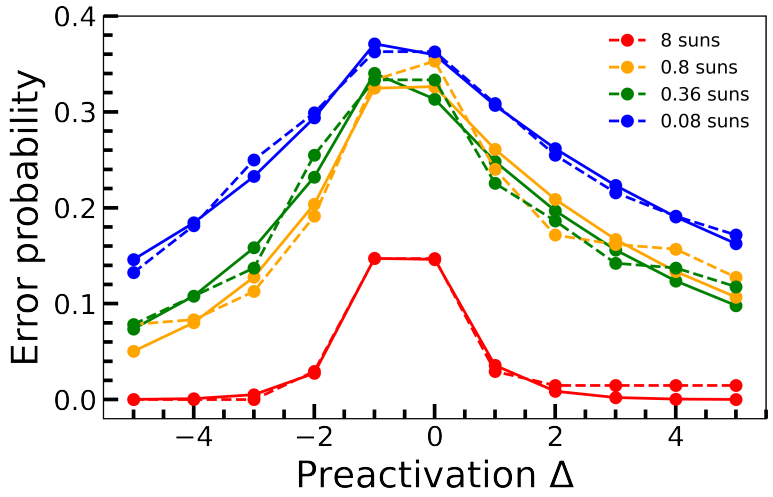


Figure 3.9: BNN inference output neuron error probability as a function of the difference between popcount and threshold, the preactivation Δ for different illumination levels and fits for the error models used in the neural network simulations.

when solving CIFAR-10, as the differences between classes are more subtle.

Solar cell illumination	MNIST Accuracy	CIFAR-10 Accuracy
Baseline	97.2%	86.6%
8 suns	97.1%	83.6%
0.8 suns	96.9%	78.2%
0.36 suns	96.9%	78.3%
0.08 suns	96.5%	73.4%

Table 3.2: Simulated accuracy of solar-cell power in a fully-connected (MNIST task) and a convolutional (CIFAR-10 task) binarized neural network under various illuminations. The software baseline assumes no bit error.

To further understand the impact of low illumination on neural network performance, we plotted the t-distributed stochastic neighbor embedding (t-SNE) representation of the MNIST test dataset in fig. 3.10 [186]. This method represents each image as a point in a two-dimensional space, with similar images corresponding to nearby points and dissimilar images corresponding to distant points. In the left image, we marked in black the correctly classified images by a neural network under 8 suns illumination but not under 0.8 suns. Interestingly, these images tend to be on the edges of the clusters corresponding to the different digit classes or even outliers that do not belong in a cluster. This suggests that the images the network starts misclassifying under 0.8 suns illumination tend to be subtle or atypical cases. The right image shows that this effect is even more pronounced under 0.08 suns illumination, with a few images inside clusters also being misclassified. Fig. 3.11 presents the same analysis for the CIFAR-10 dataset. The same trend that images wrongly classified due to low illumination tend to be edge

or atypical cases is seen, although it is less unequivocal than in the MNIST case.

For the neural network simulations, we used a fully connected and convolutional neural network architecture for the MNIST handwritten digit recognition task and the CIFAR-10 image classification tasks. Except for the input to the first layer, the activations and weights of the network were binarized, following the binarized neural network implementation [187]. The fully connected (FC) network had two hidden layers with 1,102 and 64 neurons. In contrast, the convolutional architecture was based on the VGG-16 network. It consisted of 3x3 kernels for convolutions (Conv), batch normalizations (BN), and nxn for MaxPool (MPn) and reads: [Conv 198, BN, Conv 198, MP 2, BN, Conv 354, BN, Conv 354, MP2, BN, Conv 738, BN, Conv 406, MP3, FC(1102-1102-10)]. The number of hidden layer units and convolutional filters was chosen in accordance with the dedicated mapping technique described previously, such that the total number of blocks is always odd when a block size of 58 is used.

3.3.5.1 Visualizing erroneous examples

To understand the nature of the errors, we visualize the test examples where our neural network makes an error in the prediction. In the t-SNE plots of fig. 3.10 and fig. 3.11, we saw that these errors were mainly near the edges of the clusters for the cases where the prediction was incorrect for 0.8 suns but correct for 8 suns which meant that these cases have some ambiguity to them. Whereas the examples where the 0.08 suns made incorrect predictions included both the images at the edges and more centrally located points, which should mean that those examples are much clearer. To test this, we plot some examples where the network made errors for the 0.8 suns and 0.08 suns illuminations and compare them with some random examples from the datasets.

The MNIST dataset is considered one of the most standard computer vision datasets as it is relatively easy to get a good prediction accuracy. This is because most of the images are very clear, and the data is pre-processed, as seen from the examples in fig. 3.12. The majority, if not all, of these examples, are very clear to us, and there is little ambiguity in which class a single image would belong to. We compare this to fig. 3.16, which shows examples of mistakes with 0.8 suns illumination, to observe that this set of examples has a few ambiguous cases, and generally, the form of the handwritten digits is less standard than what we usually write. Above each example, we represent the true label and the label predicted by the network as 't' and 'p,' respectively. From this, we learn that all the errors the network makes are not random; it is sometimes making mistakes where an image looks similar to another image. Some examples of such cases are the sixth image in the first row (3 predicted as 7), the sixth image in the second row (7 predicted as 2), the fourth image in the third row (3 predicted as 8), the second image in the fifth row (6 predicted as 1), etc. Although there are some examples where it makes an error, the digit is quite clear to the human eye. Now, let's look at the cases of mistakes with 0.08 suns illumination in fig. 3.14. We see three different types of images: intelligible ones (like the first image of the second row and the fourth image of the sixth row), ambiguous ones (examples

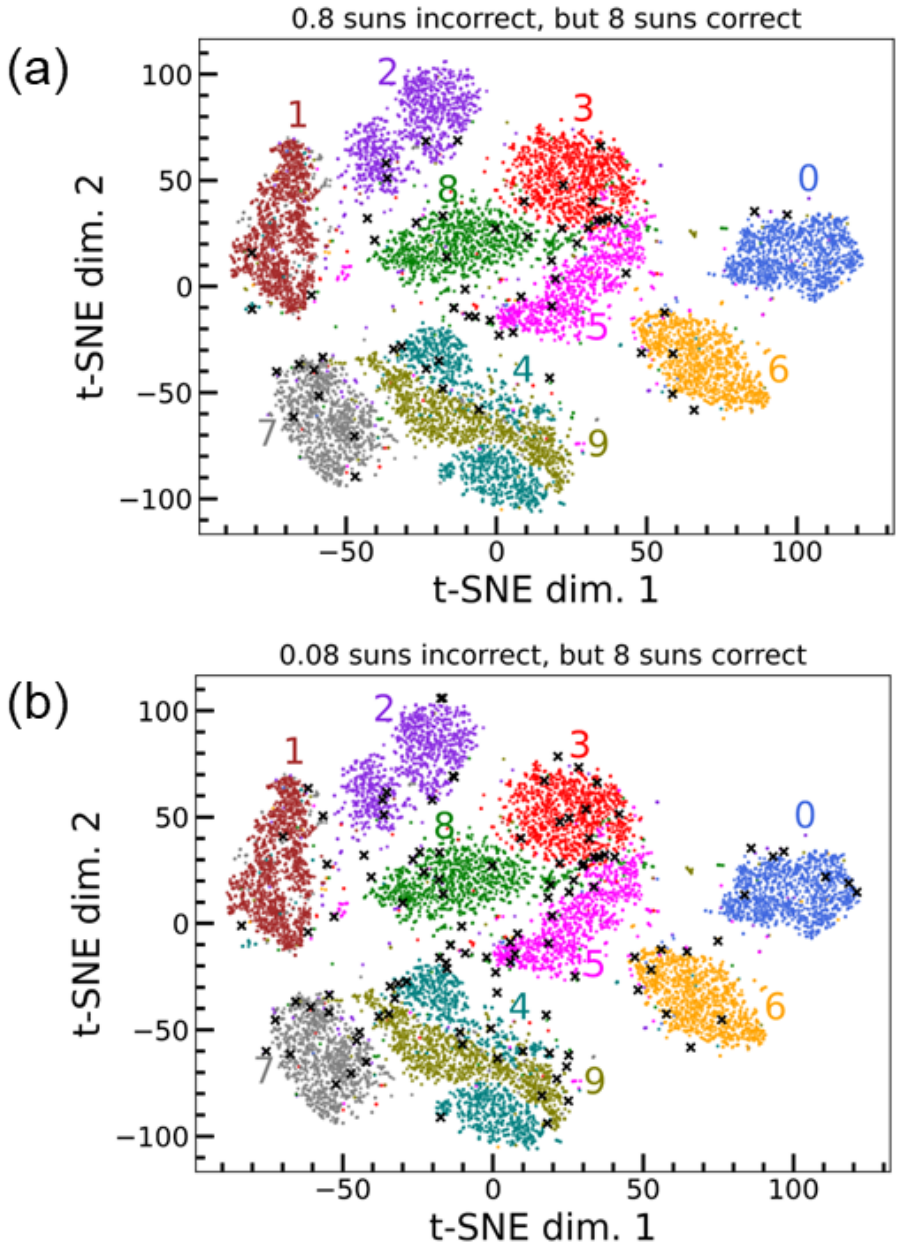


Figure 3.10: t-distributed stochastic neighbor embedding (t-SNE) representation of the MNIST test dataset. The black data points are incorrectly classified under 0.8 suns (top) and 0.08 suns (bottom) illumination, but they are correctly classified under 8 suns, using a binarized fully-connected neural network.

similar to or identical to fig. 3.16), and clear examples. This means that it is making errors in almost all the cases where it made an error in the 0.8 suns illumination and also in instances where the image is clearer.

Compared to the MNIST dataset, the CIFAR-10 dataset is more challenging as the images are now colored and have much more variability than the MNIST. The reason behind this is

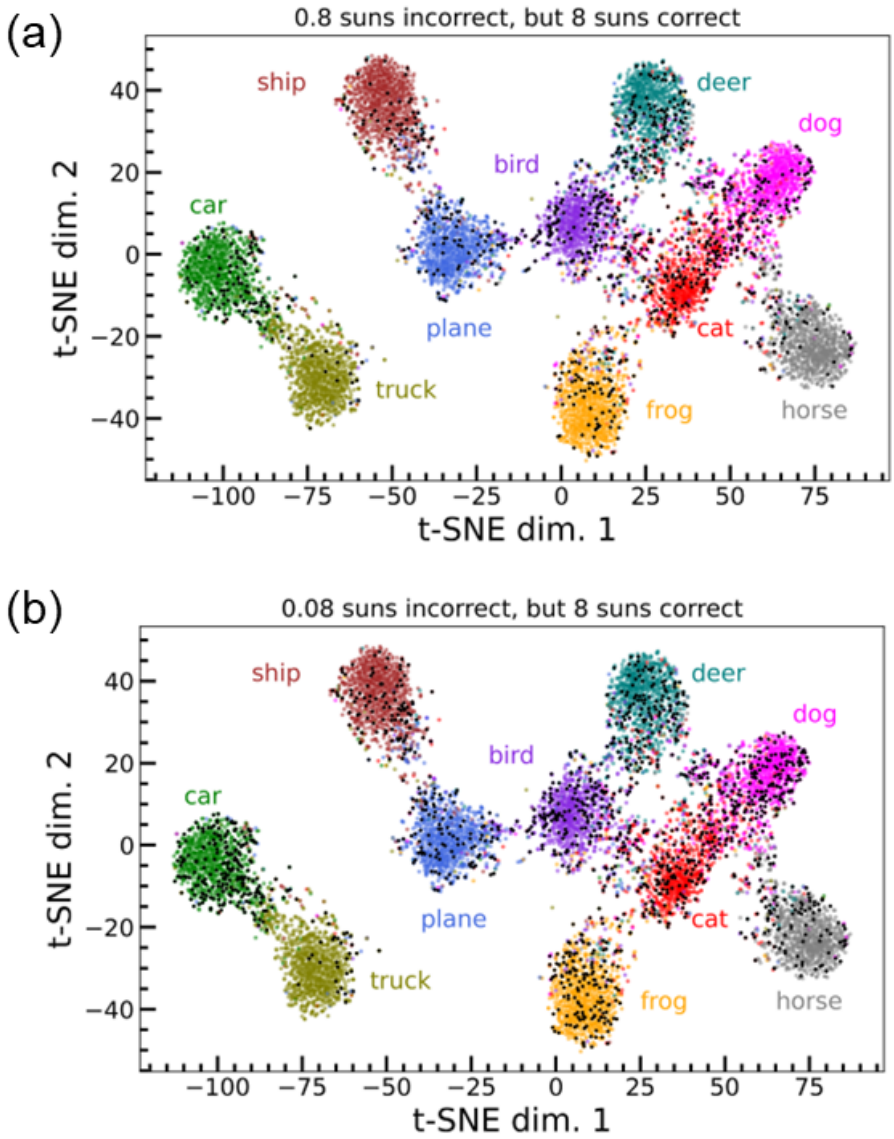


Figure 3.11: t-distributed stochastic neighbor embedding (t-SNE) representation of the CIFAR-10 test dataset. The black data points are incorrectly classified under 0.8 suns (top) and 0.08 suns (bottom) illumination, but they are correctly classified under 8 suns, using a binarized fully-connected neural network.

that for a single digit, say '0', there are only a handful of different ways we can write this; hence all the examples in the dataset with the same label look very similar. On the other hand, the output classes of the CIFAR-10 dataset are real-world objects like dog, cat, horse, deer, frog, bird, car, truck, ship, and plane. As we can see from fig. 3.15, which are random images from the test dataset of CIFAR-10, even for a particular class, the image can significantly differ based on the type of the object, the background, the angle from which it the photograph is captured, the picture quality, etc. This makes the dataset more complex and, thus, harder for our network to predict correctly. This is reflected in the fact that we have lower baseline accuracies

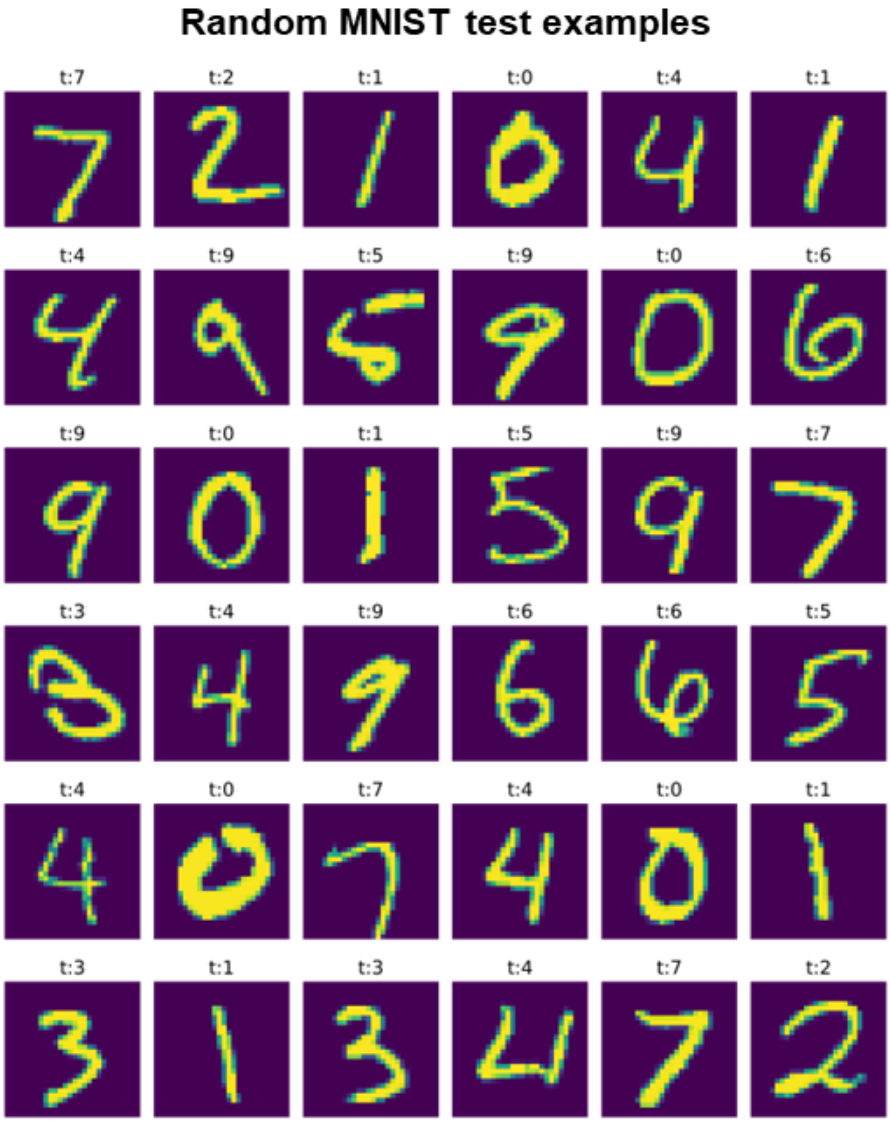


Figure 3.12: Visualization of random MNIST examples from the test dataset where each image is labeled by the true label 't.'

and higher degradation when we incorporate the errors. Fig. 3.16 and fig. 3.17 show some examples from the test set where the network makes mistakes for illuminations of 0.8 suns and 0.08 suns respectively, but the predictions are correct for 8 suns. For the cases with errors under 0.8 suns illumination, we observed common mistakes where the class labels for the true and predicted labels are swapped commonly. This happens for the plane-ship, cat-dog, and car-truck (not shown here) pairs and can be understood from the point of view that these often have very similar backgrounds like the ship and plane both often has blue backgrounds, and the cat and dogs have household in their backgrounds commonly. In fig. 3.17, for the 0.08 suns illumination, we see some of the same mistakes as in 0.8 suns, and also some mistakes where the images are not clear, even to us, like the fourth image in the first row, first, and third images

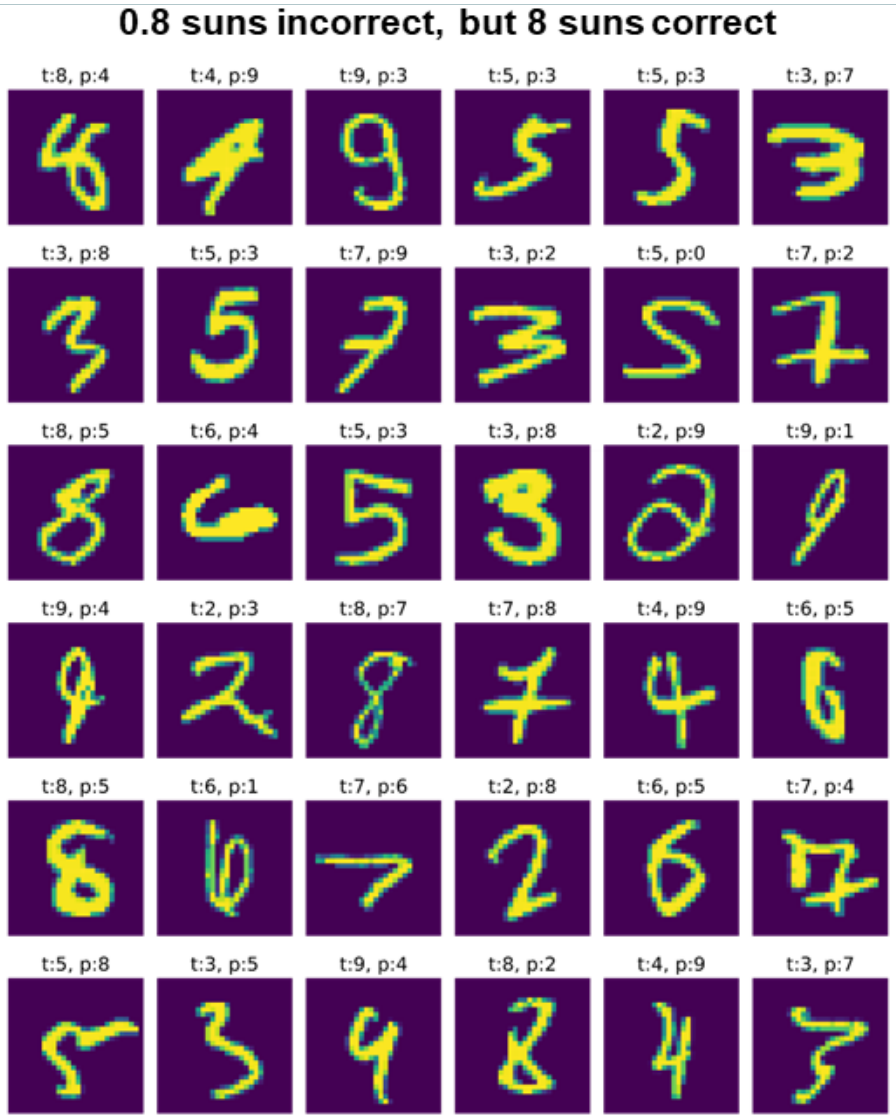


Figure 3.13: Visualization of MNIST examples from the test dataset where the network made an error for illumination of 0.8 suns but made a correct prediction for 8 suns. Each image is labeled by the true label 't' and the predicted label 'p.'

in the fourth row, the first image in the fifth row, etc. The conclusion for the CIFAR-10 mistakes is not as clear cut as the MNIST because the task's difficulty makes the prediction highly sensitive to errors, and in our case, both the 0.8 suns and 0.08 suns illumination levels cause too many errors for the visualization to be truly illustrative.

However, it can be concluded that with less illumination, the network can still perform the predictions, but it makes mistakes for more complicated or ambiguous cases. Thus our BNN inference circuit can function even in low-light settings, but it would make more mistakes in less straightforward cases. The reason for this is that for the more ambiguous cases, the magnitude of the preactivation Δ values are lower, where the probability of having an error is significantly

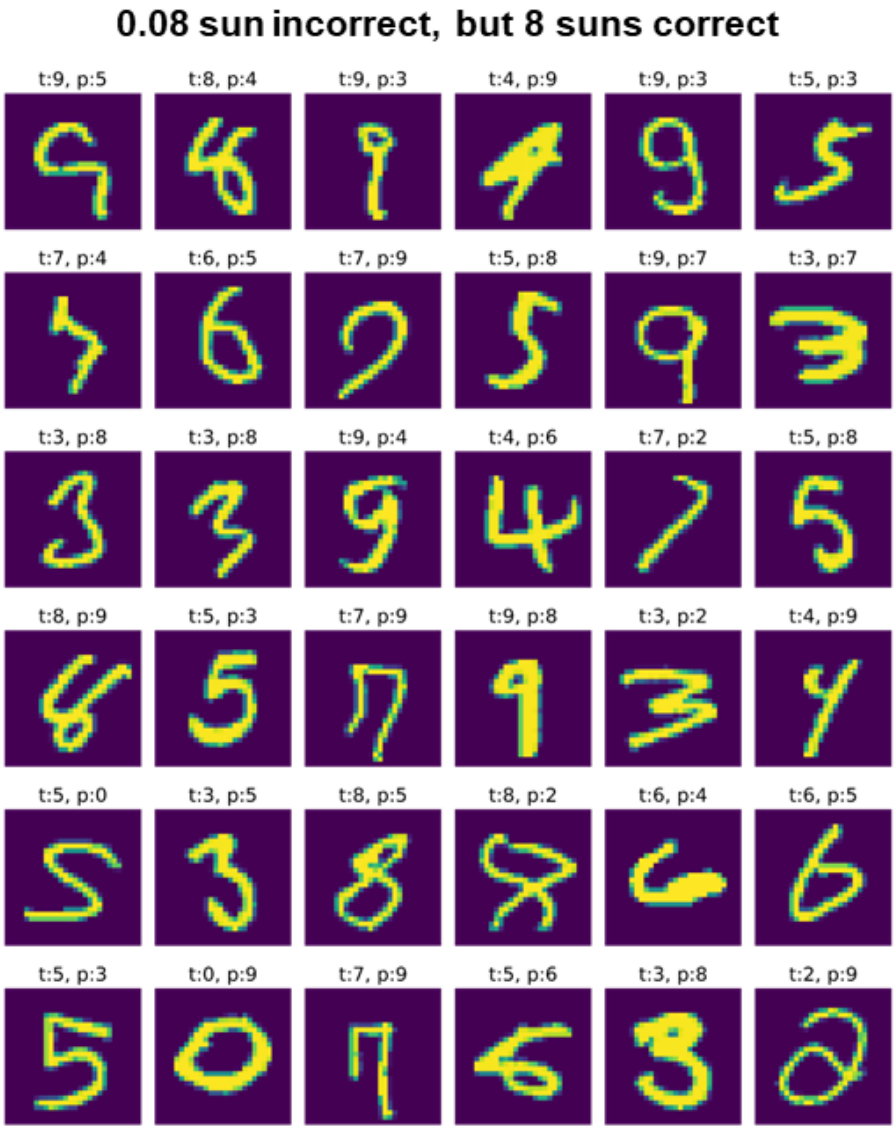


Figure 3.14: Visualization of MNIST examples from the test dataset where the network made an error for illumination of 0.08 suns but made a correct prediction for 8 suns. Each image is labeled by the true label 't' and the predicted label 'p'

higher. There is a note of caution to be remembered here: interpreting how a neural network produces an output is a notoriously difficult task, and extensive research is being done in the field of *interpretable deep learning*. Hence, the intuition we gain from studying the test examples might not always represent how a network functions. Especially for the fully connected architecture, it is doubtful that the network learns from the dataset in a way that is similar to how humans perceive information. So, the t-SNE-based study is probably more reliable as it does not depend on human perception.

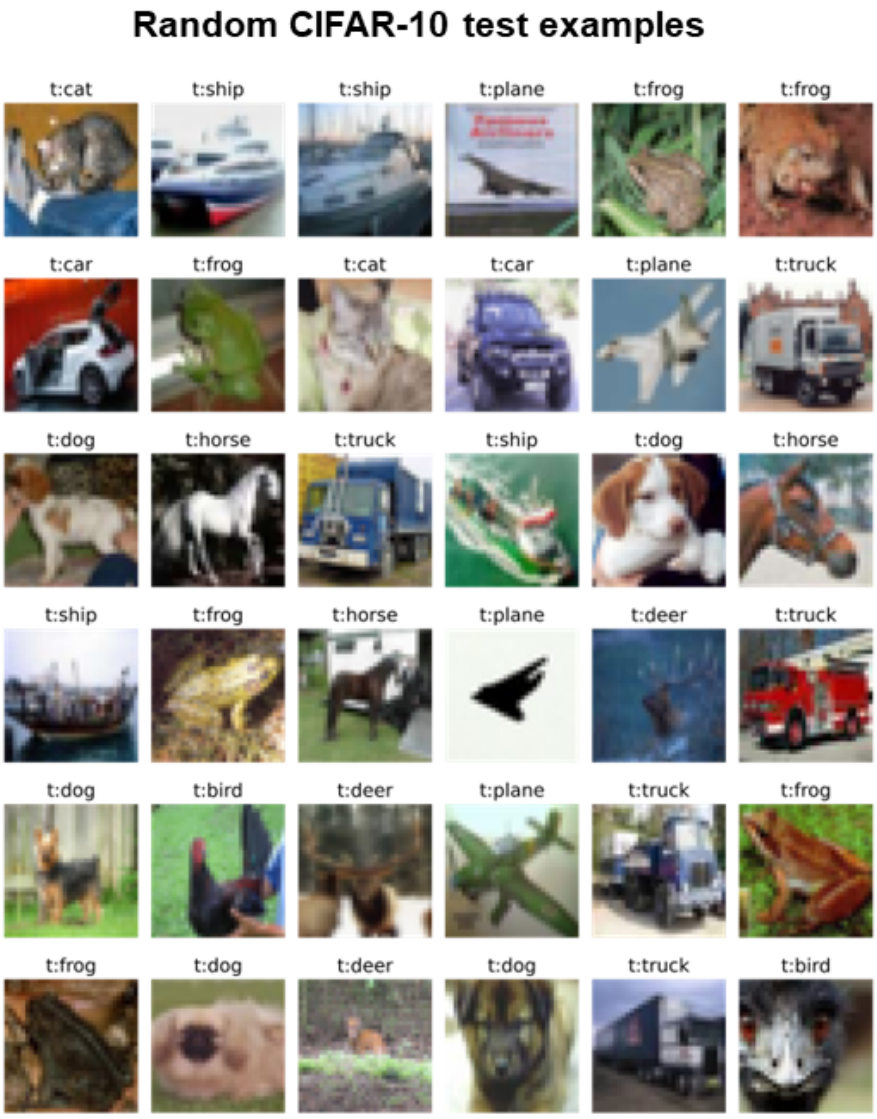


Figure 3.15: Visualization of random CIFAR-10 examples from the test dataset where each image is labeled by the true label 't.'

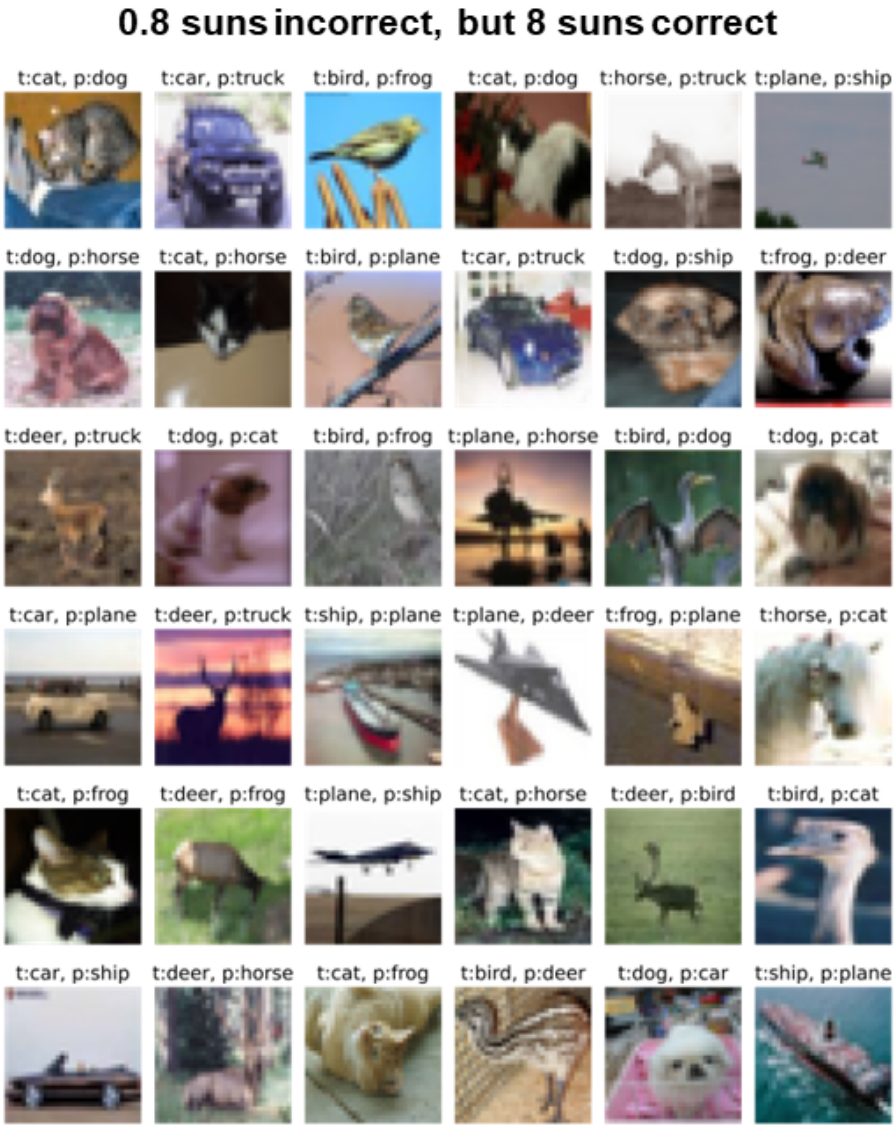


Figure 3.16: Visualization of CIFAR-10 examples from the test dataset where the network made an error for illumination of 0.8 suns but made a correct prediction for 8 suns. Each image is labeled by the true label 't' and the predicted label 'p.'

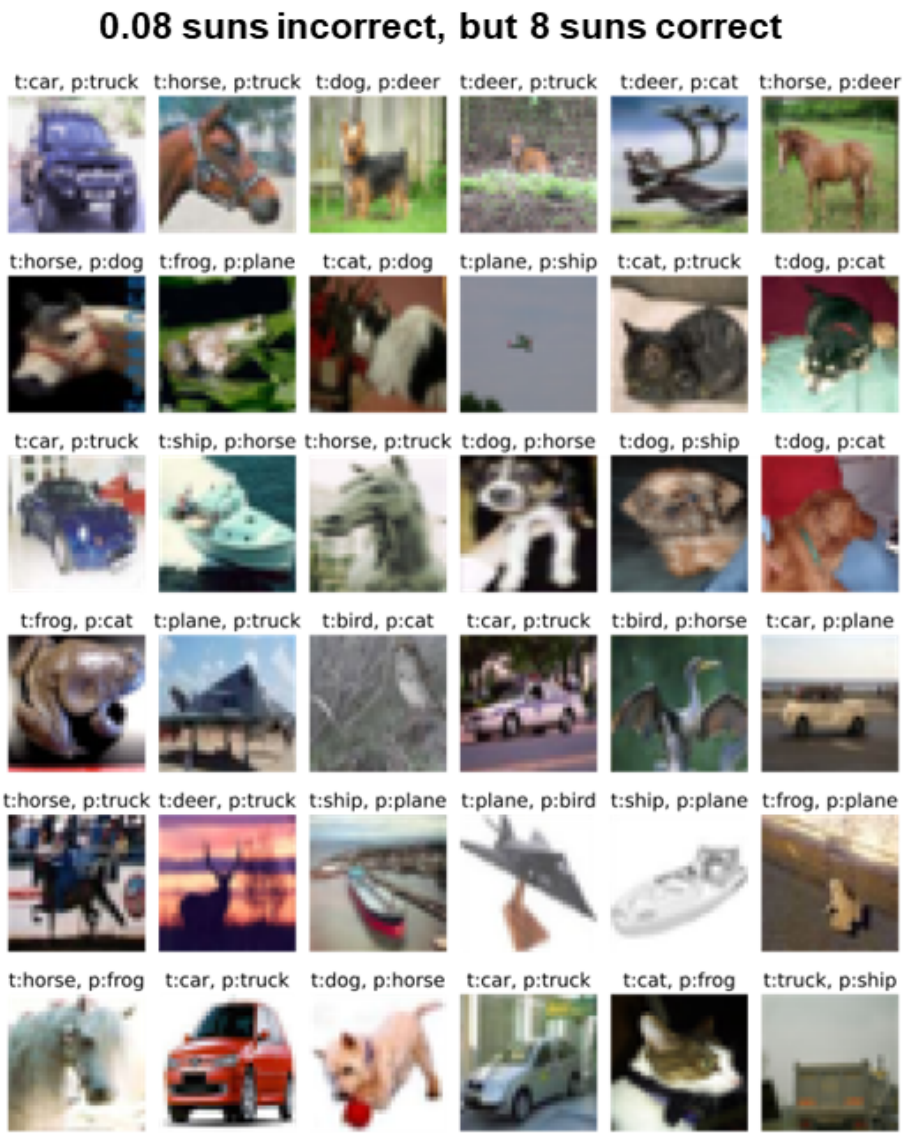


Figure 3.17: Visualization of CIFAR-10 examples from the test dataset where the network made an error for illumination of 0.08 suns but made a correct prediction for 8 suns. Each image is labeled by the true label 't' and the predicted label 'p.'

3.4 Conclusion

In this chapter, we presented two studies involving the circuit-level implementation of binarized neural networks dedicated to low-power inference. For the memory, HFO_x-based 2T2R ReRAMs are used for both works, which are integrated with the BEOL of a 130 nm CMOS process node. The design of the circuits shows remarkable robustness to the device variabilities of ReRAM and IR drop, which are significant obstacles in the traditional ReRAM-based implementation of neural networks.

The first study presented the implementation of inference of BNN, where the neurons were implemented using switched capacitive bridges and comparators. The chip was validated, and the probability of errors was measured experimentally, and these errors were used in the inference of simulated neural networks for the MNIST and CIFAR-10 tasks. We show that the neural network performance is quite robust to device-based errors even under programming conditions where the circuit is more prone to errors.

The second study concerned non-optimal power sources, relevant for extreme-edge applications where power harvesters like solar cells can provide less than maximum power. We show that even under such conditions of low illumination when the supply voltages are less than what is optimum, our circuit shows very few errors and even those errors are primarily for low preactivation magnitudes. In this work, we also consider another resource-based constraint of the hardware implementation of neural networks: the fixed number of inputs that can be presented to an array. To mitigate this, we developed the Divide-and-conquer mapping strategy, where the accumulations are done in parts. We show that this strategy suffers only from a slight degradation in accuracy. Finally, using this mapping and the bit-errors, we simulate neural networks for the MNIST and CIFAR-10 tasks and, using t-SNE plots, show that under low illumination, the network makes mistakes in more ambiguous or challenging cases.

To conclude, in this chapter, we illustrated that the binarized neural networks are robust to different kinds of circuit-level imperfection through experimental demonstration and simulations. This makes it a promising candidate for power-efficient, error-tolerant implementation of neural networks in extreme-edge environments.

Chapter 4

Bayesian binary neural networks for uncertainty quantification in medical tasks

> Intelligence takes chance with limited data in an arena where mistakes are not only possible but also necessary.
>
> Frank HERBERT

In THE last two chapters, we discussed neural network implementation that is robust to imperfections like noise, non-ideal circuit behavior, or variable power supply. This is because the deep learning algorithms that we were using relied on deterministic computations. An alternative approach is to employ probabilistic computing. Instead of aiming for device variability immunity, this computing paradigm employs it to enable computations based on probabilities. This type of computation, in the context of machine learning, is a variant of statistical learning that uses the tool of probability to model mapping from input to output. In other words, instead of dealing with finding suitable outputs, given some input, in probabilistic computing, we are concerned with finding the probability of a certain output while some information is given.

The content of this chapter revolves around probabilistic computing in the context of deep learning. We show how this approach is compatible with a class of materials whose intrinsic stochasticity could be harnessed for the hardware implementation of such models. First, we lay the theoretical foundation of this approach to computation by introducing the key ideas and algorithms behind probabilistic learning. Then, we review some experimentally demonstrated realizations of such neural networks where the variabilities of emerging memories are embraced. After that, we introduce the Bayesian binary neural network (Bayes BiNN), the model we use for the rest of the chapter. This is followed by a section where we describe the different

types of uncertainties that we can quantify using this approach and motivate how this estimation of uncertainties is an important attribute of probabilistic models where it outshines the conventional deterministic implementations. The next two sections describe, respectively, a toy task and more realistic bio-medical tasks and how our model performs for them. We conclude the chapter by proposing some physical systems that are suitable for implementing this special type of neural network in hardware.

4.1 Theoretical background

The notion of probability and its quantification has been discussed in the history of mathematics since the sixteenth century by the likes of Gerolamo Cardano, Pierre de Fermat, Blaise Pascal, and Christian Huygens. The probability of an event is simply a measure of how likely it is for an event to occur. To quantify the probability of an event, the classical or frequentist interpretation says that it is the limit value of its relative frequency over many trials [188]. If we conduct N trials, and the event under consideration, say event A, happens n number of times, then the probability of event A, is given by

$$P(A) = \lim_{N\to\infty}\left(\frac{n}{N}\right). \tag{4.1}$$

4.1.1 Bayesian interpretation of probability

This interpretation of probability assumes that there exists an absolute value of the probability independent of the extent of our belief. However, Thomas Bayes presented a different definition of probability in 1763 that is more evidence-based [189]. According to Bayes' interpretation, the probability is considered a 'degree of belief' in the chance of the occurrence of an event. This approach is more subjective and cannot be directly measured like the frequentist approach. Instead, the probability is updated as more evidence becomes available. This update starts from a *prior* degree of belief in the event before any evidence is considered, which is then updated as more information becomes available. The Bayesian interpretation centers around Bayes' theorem, which utilizes the sum rule and product rule of probabilities for conditional events.

Let A be an event, and let $E_1, E_2, E_3, ..., E_n$ be n mutually exclusive partitions of the whole sample space. Suppose we only have access to the conditional probabilities $P(A|E_1)$, $P(A|E_2)$, $P(A|E_3)$,..., $P(A|E_n)$. Concretely, if the events E_is are events that cause event A, this can be interpreted as information about the causal connections. For instance, E_1 and E_2 might represent the events of windy and calm weather, respectively, and A can be the event that a person wears a coat. By observing what the person wears given the weather conditions, we have about the probabilities $P(A|E_1)$ and $P(A|E_2)$. Now, if we observe that the person is wearing a coat, can we infer about the weather? Bayes' theorem answers this question by inverting the probability

and finding $P(E_1|A)$.

Bayes' theorem states

$$P(E_i|A) = \frac{P(A|E_i)P(E_i)}{\sum_{i=1}^{n} P(A|E_i)P(E_i)}. \tag{4.2}$$

Let us describe each of the terms in this equation separately.

- **Prior** $P(E_i)$: This is the aforementioned prior probability of a causal event happening independent of any other conditioning. In terms of our example, it is just the probability of the weather being windy and is totally uncorrelated with the person wearing a coat.

- **Likelihood** $P(A|E_i)$: The probability of event A conditioned on the event E_i, or in other words, what is the likelihood of occurrence of event A when the event E_i has already occurred. For our example, this is the probability that we get from observing the person on different days with different wind conditions.

- **Evidence** $\sum_{i=1}^{n} P(A|E_i)P(E_i)$: This term is the normalization constant that takes into account all possible ways that event A could have happened.

- **Posterior**$(P(E_i|A))$: It is the *inverted* probability that we calculate. The process of our probability calculation goes as follows: we have a prior idea about the weather conditions of the area that is given by $P(E_i)$. We observe the person for some days to note the proportion of times he is wearing a coat given a certain weather condition to get $P(A|E_i)$. Then we use equ. 4.2 to find the posterior value of $P(E_i|A)$. The next time we observe this person, this posterior probability acts as the prior.

To summarize, we start with a prior idea about the probabilities and update them as new information gets available. This is operationally quite similar to how we train our neural networks, as presented in Chapter 1, where we present new training examples, and to get a better prediction, we update the network parameters. The Bayes' theorem can be recast in the form of a learning task if we consider our event A to be the whole dataset $\mathbf{D}$, the events E_is to be the parameters of the model $\mathbf{W}$, then we can rewrite Bayes' theorem as

$$P(\mathbf{W}|\mathbf{D}) = \frac{P(\mathbf{D}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{D})}. \tag{4.3}$$

Here, $\mathbf{D} = \{X, Y\}\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ is the training dataset with inputs $\mathbf{x}_i$, their corresponding output classes being $y_i \in \{1, 2, ..., C\}$ where C is the total number of classes, and N is the size of this dataset. In learning, the goal is to optimize model parameters $\mathbf{W}$ so that our model, denoted by $y = f^{\mathbf{W}}(\mathbf{x})$, can produce the intended output. In order to achieve this using the Bayesian approach, we define a $P(y|\mathbf{x}, \mathbf{W})$ to be the model likelihood, and for classification, the softmax likelihood for class index c is:

$$P(y=c|\mathbf{x},\mathbf{W}) = \frac{\exp(f_c^{\mathbf{W}}(\mathbf{x}))}{\sum_{c'}\exp(f_{c'}^{\mathbf{W}}(\mathbf{x}))} \tag{4.4}$$

For a test input sample $\mathbf{x}^\star$, the class label probability can be calculated as

$$P(y^\star|\mathbf{x}^\star,\mathbf{D}) = \int P(y^\star|\mathbf{x}^\star,\mathbf{W})P(\mathbf{W}|\mathbf{D})d\mathbf{W}. \tag{4.5}$$

This calculation is the same as what we did for the inference in our supervised learning of neural networks and is also called inference. It is also known as marginalization, as we are integrating over $\mathbf{W}$. In other words, this output probability is a probabilistic average or expectation over different models that are sampled from the distribution of weight parameters. Instead of a deterministic model, as we had until now, if we have a neural network where the weights are random variables that have associated probability distributions, then equ. 4.3 can be used to update those probability distributions. In this case, instead of learning the values of the weights, we learn the values of the parameters of the probability distribution $P(\mathbf{W}|\mathbf{D})$.

In general, it is impossible to analytically calculate the denominator term of equ. 4.3 and hence also $P(\mathbf{W}|\mathbf{D})$. To circumvent this problem, different methods are adapted, the variational inference being one of them, where a variational distribution $q_\theta(\mathbf{W})$ is used to approximate $P(\mathbf{W}|\mathbf{D})$. Then, the Kullback-Leibler divergence (KL divergence), a measure of distance between two probability distributions, is used to quantify how good our approximation is. This divergence is minimized by iteration through our dataset [35, 190, 191]. The KL divergence between two distributions $q_\theta(\mathbf{W})$ and $P(\mathbf{W}|\mathbf{D})$ is calculated by the formula

$$\mathrm{KL}(q_\theta(\mathbf{W})||P(\mathbf{W}|\mathbf{D})) = \int q_\theta(\mathbf{W})\log\frac{q_\theta(\mathbf{W})}{P(\mathbf{W}|\mathbf{D})}d\mathbf{W}. \tag{4.6}$$

4.1.2 Bayesian deep learning

As of now, we have just stated an alternative way of doing statistical learning. We will gradually present the advantage of this approach and how it can be superior to the more conventional approach to deep learning in some contexts. Here, we present some ways in which the Bayesian approach is applied to deep learning models.

4.1.2.1 Monte Carlo (MC) dropout

Dropout is a regularization technique that prevents overfitting in deep neural networks by randomly dropping out a fraction of nodes during each training iteration. This forces the remaining nodes to learn more robust representations that do not rely on the presence of any single node [192]. Monte Carlo dropout extends this technique to perform approximate Bayesian inference in deep neural networks. The dropout mask is randomly sampled at each forward pass during inference, and this is done multiple times (10-100) to obtain a distribution of outputs

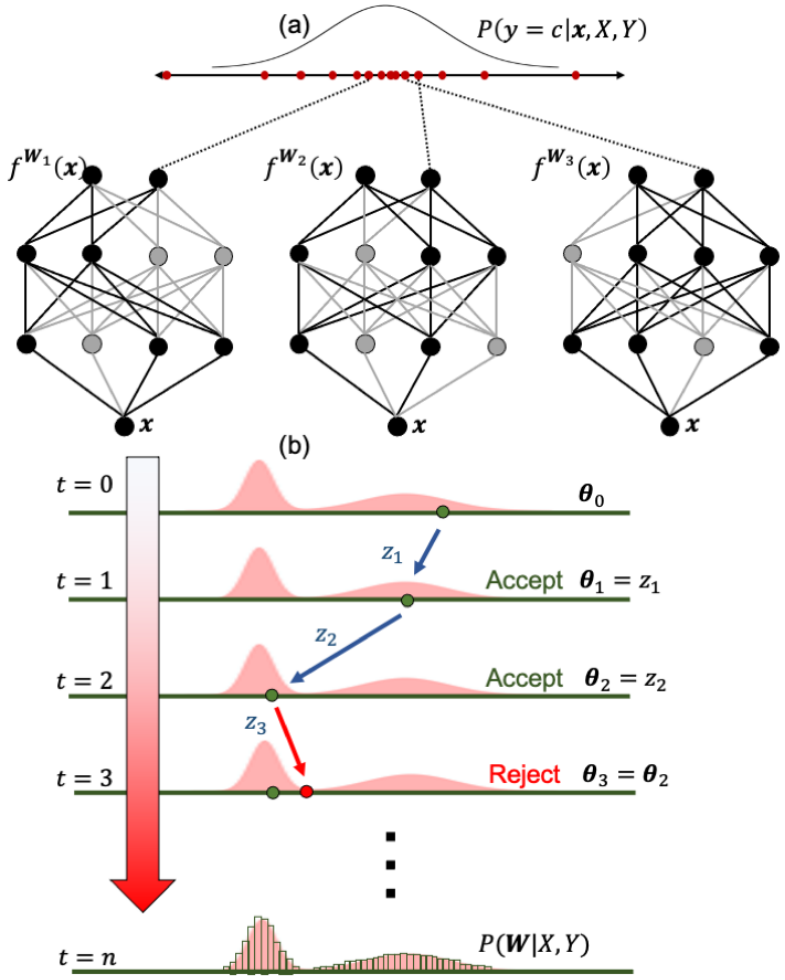


Figure 4.1: MC dropout and MCMC. (a) The Monte Carlo dropout method of doing Bayesian inference with neural networks. In each iteration, some neuron nodes are masked at random with some probability, effectively making the model different in each iteration ($f^{\mathbf{W}_1(\mathbf{x})}$, $f^{\mathbf{W}_2(\mathbf{x})}$, and $f^{\mathbf{W}_3(\mathbf{x})}$, here). Thus, for the same input $\mathbf{x}$, we get different outputs which give us the distribution of the output probability $P(y|\mathbf{x}, \mathbf{D})$. (b) Markov chain Monte Carlo method is illustrated by a simple example. The distribution parameters $\boldsymbol{\theta}$ explore the sample space by a random walk where each step taken is either accepted or rejected based on a ratio of the current and proposed states. The resultant frequency distribution of points, when normalized, gives an approximation of the posterior distribution.

for each input. The mean of these outputs is then used as the final prediction, making it an effective way to estimate model uncertainty [193, 194]. Fig. 4.1 (a) demonstrates the approach, whereby three different models $f^{\mathbf{W}_1(\mathbf{x})}$, $f^{\mathbf{W}_2(\mathbf{x})}$, and $f^{\mathbf{W}_3(\mathbf{x})}$ produce different outputs for the same input due to different active units, and these outputs are gathered to get the final distribution of outputs $P(y|\mathbf{x}, \mathbf{D})$.

4.1.2.2 Markov chain Monte Carlo (MCMC)

The Markov chain Monte Carlo (MCMC) method generates samples from a difficult-to-sample probability distribution by constructing a Markov chain that converges to the desired distribution. In deep neural networks, MCMC can be used to perform Bayesian inference to estimate model uncertainty. MCMC is initialized with a random set of weights, and at each iteration, a new set of weights is proposed based on the current state of the chain. The acceptance probability of the proposed state is computed, and if accepted, it becomes the new state of the chain. Fig. 4.1 (b) shows an example of the MCMC process. By accumulating all the accepted points, we can output a distribution for the posterior. MCMC is a powerful and widely used method for estimating complex probability distributions. [195–199].

4.1.2.3 Variational autoencoder (VAE)

Autoencoder is a type of neural network that efficiently codes unlabelled data [200]. As shown in fig. 4.2 (a), it consists of an encoder that maps the input data X to a lower-dimensional representation called the latent variable Z and a decoder that generates a reconstruction of the original input data X'. Variational Autoencoder (VAE) is a probabilistic variant that learns a probability distribution over the latent code, enabling flexible modeling of input data [201, 202]. To train VAEs using backpropagation, the reparametrization trick is used, expressing the latent code as a function of a noise variable, and the mean and variance of the Gaussian distribution are learned. VAEs have been used for generative modeling, including image and text generation, anomaly detection, and dimensionality reduction [203, 204]. VAEs can naturally learn a more regularized latent variable due to their probabilistic nature.

4.1.2.4 Bayes By Backprop (BBB)

Bayes by Backprop is another method used to train Bayesian neural networks [205]. The method is based on the principle of variational inference, which is a way to approximate the posterior distribution of a Bayesian model by a surrogate distribution. The key idea of the method is to introduce a set of variational parameters, which are optimized to approximate the true posterior distribution of weights. These parameters are learned using a gradient-based optimization method, such as stochastic gradient descent.

The variational parameters are used to define a probability distribution over the weights in the neural network. Let $\boldsymbol{\theta}$ be the variational parameters of the distributions of the weights $\mathbf{W}$, and $q_{\boldsymbol{\theta}}(\mathbf{W})$ be the surrogate variational distribution, then during training, the cost function, called the variational free energy $F(\mathbf{D},\boldsymbol{\theta})$is defined as

$$F(\mathbf{D},\boldsymbol{\theta}) = \mathrm{KL}(q_{\boldsymbol{\theta}}(\mathbf{W})||P(\mathbf{W})) - \mathbb{E}_{q(\boldsymbol{\theta},\mathbf{W})}[\log P(\mathbf{D}|\mathbf{W})]. \tag{4.7}$$

In equ. 4.7, the cost function is composed of two parts: the likelihood cost, which depends

on the data D, and the complexity cost, which depends on the prior P(w). This cost function balances the complexity of the data and the simplicity of the prior. It determines the optimal values of the variational parameters $\boldsymbol{\theta}$ that best fit the data while avoiding overfitting. Fig. 4.2 (b) shows the learning and inference of a Bayesian neural network using this method.

Once the training is complete, the posterior distribution over the weights can be used to make predictions on new data using equ. 4.5. The predictive distribution is obtained by averaging over the posterior distribution of weights, weighted by their probability under the posterior distribution.

BBB has been shown to improve the performance of neural networks in a variety of tasks, including image classification, speech recognition, and reinforcement learning. It also has the advantage of providing uncertainty estimates for the predictions, which can be useful in safety-critical applications such as medical diagnosis or autonomous driving, as we will motivate later in this chapter. Also, since this method is based on the backpropagation algorithm, it is highly compatible with the neural network implementation of the deep learning frameworks. This will be our method of choice for the neural network that we will be studying in this chapter.

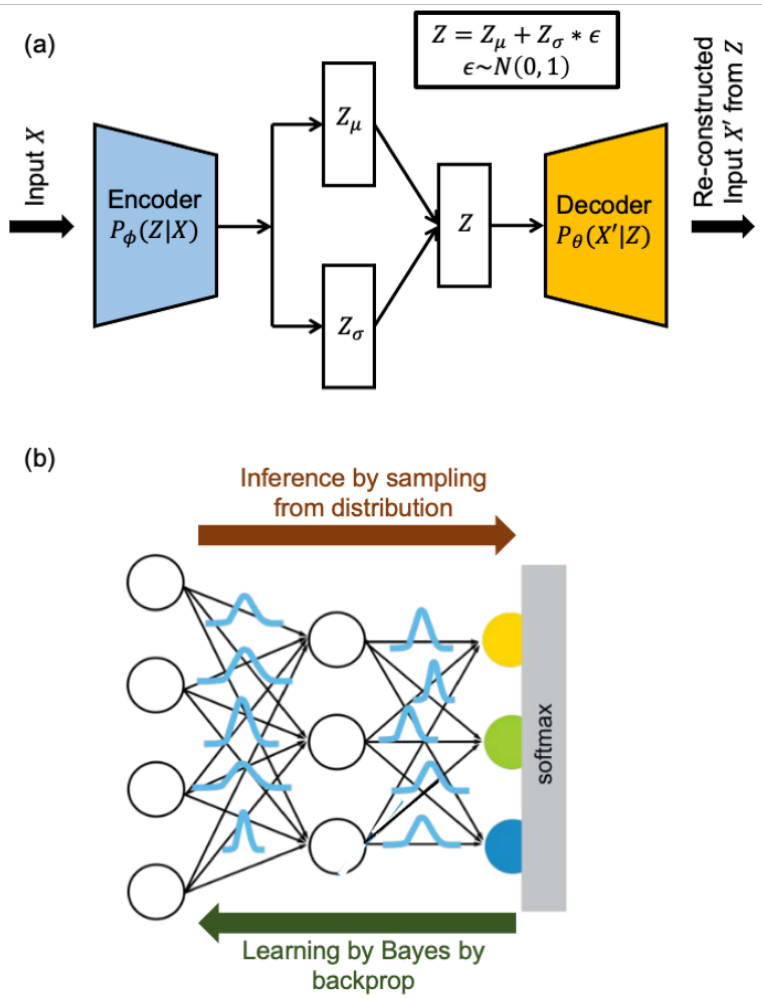


Figure 4.2: VAE and BBB. (a) The VAE architecture shows the reparametrization trick. The encoder $P_\phi(Z|X)$ generates the mean and standard deviation parameters for the latent variable Z, which is then passed through the decoder network $P_\theta(X'|Z)$ to generate the reconstruction of the input. (b) The Bayes through backprop uses a combination of the backpropagation and Bayes' theorem to derive the updates to the variational parameters. The inference of the model is done by sampling through the weight distributions to get the output distribution. Then equ. 4.7 is used to calculate the loss function, and then the updates are backpropagated through our network.

4.2 Memristor-based probabilistic ML

In the following section, we will present three recent studies about the implementation of probabilistic computing on hardware using emerging memory technologies.

4.2.1 Bayesian machine

A Bayesian machine is a non-deep machine learning model that uses Bayesian inference for predictions. In [206], a memristor-based implementation of Naïve Bayes method for Bayesian inference was achieved. Given n observations $O_1, O_2, ..., O_n$ and the output $Y = y$, Bayes' theorem (equ.4.2) was used to calculate $P(Y = y|O_1, O_2, ..., O_n) = P(O_1, O_2, ..., O_n|Y = y)P(Y = y)$. Then, Naïve Bayes technique was used, which assumes that the observations are conditionally independent. Thus, the following simplification is obtained

$$P(Y = y|O_1, O_2, ..., O_n) = P(O_1|Y = y)P(O_2|Y = y)...P(O_n|Y = y)P(Y = y). \tag{4.8}$$

This Bayesian machine stored 8-bit probability values in a ReRAM-based memory array using a complimentary fashion where the LRS-HRS and HRS-LRS pairs respectively denote the 0 and 1 bits, as shown in fig. 4.3 (a). These probability values were converted to a bitstream by digital 'Gupta' circuits and fed into an AND gate to perform the stochastic multiplications. The non-volatile and low-power attributes of memristive devices were utilized for storing the probability values, and the computations were done by other circuits. The work utilized stochastic computing, a type of computation that represents continuous values as a sequence of random bits, allowing for straightforward bit-wise operations to perform complex computations [207].

4.2.2 MCMC on chip

Cycle-to-cycle and device-to-device variability in ReRAMs can be harnessed for probabilistic computing, as demonstrated in [208]. When a SET operation is performed on a ReRAM, a different conductance state is obtained following a Gaussian distribution. An on-chip demonstration of learning the posterior distribution was performed using the MCMC method, utilizing conductances to represent the weights of the network (fig. 4.3 (b)). A deterministic model was stored in each row of the array, with its parameters encoded by the conductance difference between positive and negative sets of devices. The Metropolis-Hastings MCMC algorithm was used to generate a proposed model at each row based on the previous row's model [209]. The ReRAM intrinsic random variability naturally generates each parameter of the proposed model by performing a SET operation on each device in the row, with a programming current that samples a new conductance value from a Gaussian distribution centered on the corresponding device's previous-row conductance value. After training, the learned posterior distribution in the array can be used for inference, which is performed by taking the expectation of the outputs of all the rows over the posterior distribution.

4.2.3 Bayesian neural network on chip

In [112], a Bayesian neural network was implemented where the intrinsic variabilities in filamentary memristors and phase change memories in crossbar arrays were used to store the

probabilistic weights [1]. The conductance of a single memristor is used as a single deterministic model, and the distribution over a set of them comprises the probability distribution (as shown in fig. 4.3 (c)). It was trained using the variational inference-based BBB method discussed in the preceding section. The high conductance state of these memories exhibits variability that can be modeled by a Gaussian distribution with a mean μ and standard deviation σ, both of which depend upon the SET programming current. However, the mean and standard deviations are strongly correlated and cannot be set independently for such devices. This is an obstacle in the variational inference as all the parameters need to be changed independently. In order to expand the range of Gaussian distributions, we adopt a method where each sample of a probabilistic weight is stored as the difference between the conductance values of two neighboring memory cells. This approach proves to be highly effective since the difference between two Gaussian distributions remains a Gaussian distribution.

Still, the region of space covered by the memristors in the μ-σ plane is insufficient to achieve effective training using the variational inference method. For that reason, a term is added to the total loss function called the technology loss. This term takes into account the constraints of the memory technology being used and penalizes the model for having distribution parameter values outside its domain of possible value.

In this work, it has been demonstrated that implementing Bayesian neural networks using memristive devices is possible by leveraging the device variabilities to emulate the probability distributions. This approach has been shown to have advantages over deterministic neural networks, particularly in quantifying uncertainties for safety-critical applications like bio-medical tasks. Later in this chapter, we will explore this aspect in greater detail.

[1] The author of this thesis had a minor contribution to this work and is a co-author of the article

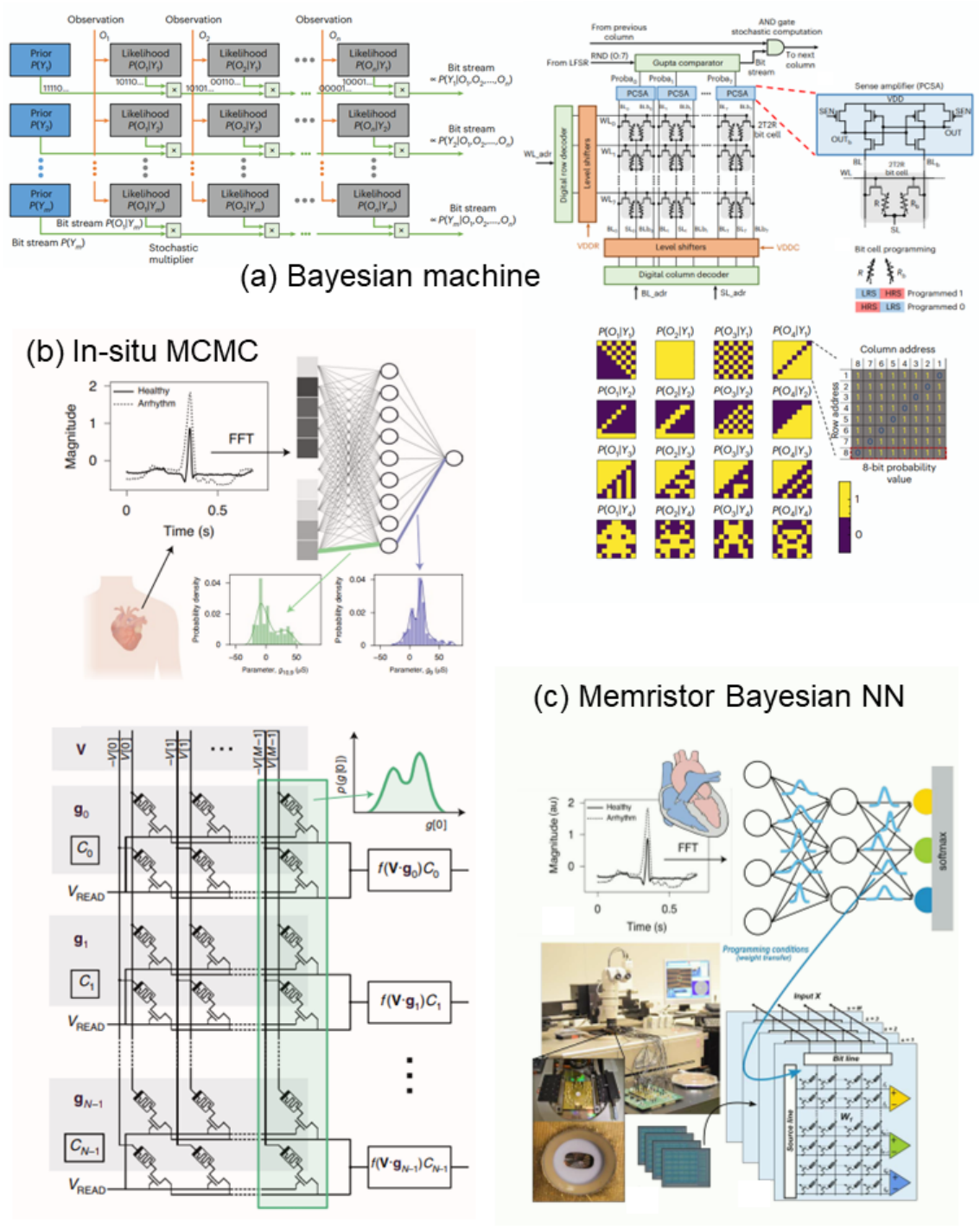


Figure 4.3: Probabilistic computing in hardware. (a) The memristor-based Bayesian machine where the Naïve Bayes method is used for Bayesian inference wherein the probability bits are stored in the array, and their multiplication is carried out by stochastic computing. (b) In-situ MCMC using the variability of memristors to perform the Metropolis-Hastings sampling algorithm. A single column in the array represents the posterior probability of a parameter. (c) A Bayesian neural network is implemented where the Gaussian distribution of device variability plays the role of the probabilistic weights (Adapted from [112, 206, 208]).

4.3 Bayesian binary neural networks

4.3.1 Architecture and inference

A deterministic binary neural network (BiNN) has binarized values for the synaptic weights and real values for the activations (unlike the binarized neural network (BNN), which we studied in section 2.4.1, where both the neurons and synapses are binarized). Its Bayesian counterpart, the Bayesian binary neural network (Bayes BiNN), replaces the deterministic binarized value of the weight by a random variable that follows a Bernoulli distribution [210].

Concretely, in the deterministic version, each weight $\mathbf{W}^{bin}$ is either +1 or -1 and remains fixed once a network is fully trained. On the other hand, for Bayes BiNN, we learn the probability $\boldsymbol{p}$ that the weight would take a value of +1. Mathematically, the j^{th} weight $W_j^{bin} \sim \text{Bern}(p_j)$ and so, the probability of sampling a value of +1 is given as

$$P(W_j^{bin} = +1) = p_j \tag{4.9}$$

Fig. 4.4 (a) and (b) highlight the differences between the deterministic and Bayesian networks. Unlike the fixed weights of the vanilla BNN, the Bayes BiNN learns a joint distribution over the weights, and for inference, we draw samples from this distribution. Thus, we get multiple models, and we consider their cumulative output as the output probability distribution. To compare with BiNN, as described in section 2.4.1, there exists hidden real weights $\mathbf{W}^{real}$, the sign of which leads to the binary weight $\mathbf{W}^{bin}$. The Bernoulli probability $\boldsymbol{p}$ plays an analogous role to this real weight, as it also generates the binarized value, albeit using probabilities. For brevity, we shall omit the 'bin' prefix from here onwards, as we will always be talking about binary weights.

The training algorithm is based on the Bayes-by-backprop algorithm described in section 4.1.2 and is detailed in appendix 4.8. Effectively, the probability associated with each synaptic weight $\mathbf{p}$ is learned, and for testing, a Monte-Carlo average is taken over the outputs of different sampled models. If $\mathbf{W}^{(c)} \sim q(\mathbf{W})$ is the c^{th} sample of the model from the learned posterior $q(\mathbf{W})$, and C is the total number of samples, then the output probability for the k^{th} class is given as

$$P(y = k|\mathbf{x}) = \frac{1}{C}\sum_{i=1}^{C} P(y = k|\mathbf{x}, \mathbf{W}^{(c)}). \tag{4.10}$$

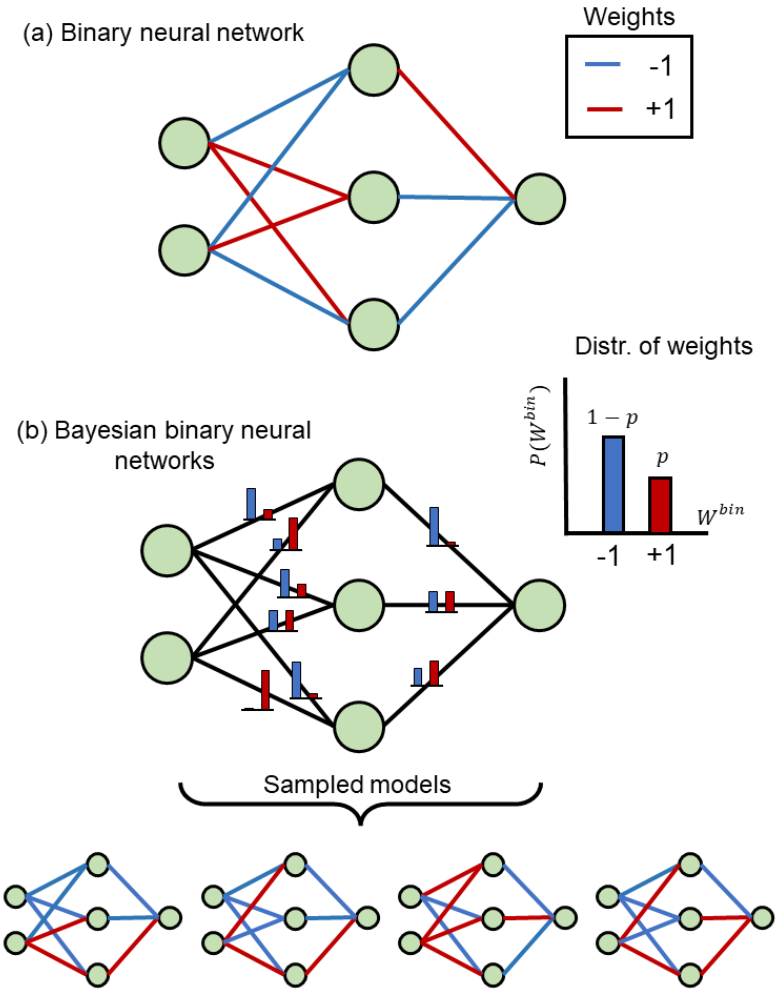


Figure 4.4: Comparison of a binary neural network and its Bayesian analog. (a) The deterministic binary neural network where the weight is fixed and is always either -1 or +1. (b) The Bayes BiNN, where each weight is a random variable following a Bernoulli distribution with an associated probability parameter. In training, these probabilities are learned. During inference, we sample C models to compute the MC average for finding the output probability.

4.4 Uncertainty quantification

4.4.1 Safety-critical applications

In this section, we bring into focus the advantage of using the Bayesian approach: the ability to quantify uncertainty in the prediction of an output. The aspect that conventional neural networks excel at is the prediction accuracy of a task, but there are many applications where merely being accurate is not sufficient. There are safety-critical applications in which a single error in prediction can cause a huge loss in terms of human lives or finance. Bio-medical diagnostic applications are one such domain; if the output predictions are wrong, that can lead to a

completely wrong diagnosis, potentially putting human life in jeopardy. An error in the prediction can arise from a plethora of sources, the two most prominent sources being noise in the data and testing data which is *out-of-distribution* with respect to the data on which the model was trained. The type of uncertainty deriving from the first kind of source is called **aleatoric** uncertainty, and the latter is called **epistemic** uncertainty [194, 211]. In such applications, it is more desirable that the network also provides some information about the uncertainty of the output. In that case, a human doctor intervention can be made to make the final decision. For example, we can consider the Covid-19 pandemic: suppose that the virus has mutated in country Y to a new variant X which is yet to be identified as a variant in the scientific community. A person comes back from vacation from country Y, finds himself with Covid-like symptoms, and takes an RT-PCR test. At this point, the ideal case would be that our model, which has not been trained on variant X gives a high epistemic uncertainty. The doctors and scientists would further look into the data to learn more about variant Y. Another case would have been that the RT-PCR test was not done properly. Instead of providing a wrong output, the model says that the aleatoric uncertainty is high; that is, it cannot predict confidently if it is a positive case or not.

Bayesian neural networks provide a natural way to express these types of uncertainties. As opposed to conventional neural networks, which are only capable of point estimates, these neural networks learn a distribution over the dataset that can capture the intricacies of the data much better. In particular, Bayesian neural networks offer advantages over conventional ones in the following aspects.

- Conventional neural networks are infamous for performing poorly in terms of uncertainty quantification. The softmax outputs, although considered erroneously to be the output probability, fail to capture the subtleties that are necessary for evaluating uncertainties [211–213].

- Conventional neural networks often overfit small datasets, which is common in medical applications, leading to highly certain predictions in all scenarios [214, 215].

- Deterministic neural networks cannot distinguish the two different types of uncertainties since the definition of these uncertainties, as we would see next, requires averages over distributions [216].

Other safety-critical applications include autonomous vehicles, automated flight control, using deep learning for providing loans, delivering justice, or taking critical policy decisions [217–219]. Another key aspect of these applications is the interpretability of models, which also can be derived from probabilistic neural networks [220, 221]. Now, with the importance of quantifying the uncertainty ascertained, let us proceed to describe the exact mathematical formulation that allows us to calculate the two different types of uncertainties.

4.4.2 Quantification of uncertainty

The total uncertainty in prediction, which is referred to as *predictive uncertainty*, comprises two main components: epistemic and aleatoric uncertainty [216, 222]. Epistemic uncertainty pertains to the degree of uncertainty we have in the model parameters. This can be conceptualized as the spread of the posterior weight distribution $P(\mathbf{W}|\mathbf{D})$, whereby a wider posterior distribution indicates higher epistemic uncertainty, while a narrower posterior distribution indicates lower epistemic uncertainty. The origin of this kind of uncertainty stems from the lack of knowledge about the input example. Semantically, the word 'epistemic' has the meaning of being related to knowledge. Concretely, if the input during testing comes from a distribution that is not the same as the distribution in the training set, the epistemic uncertainty would be high.

Conversely, aleatoric uncertainty stems from the input itself. In cases where the input instance and fixed weight parameters are provided, high aleatoric uncertainty suggests that the output estimate is noisy (in the case of regression) or that the class to which it belongs is unknown or ambiguous (in the case of classification). When aleatoric uncertainty is high, it indicates that there is insufficient information to predict the output value for an input with fixed weight settings. This can be attributed to unobserved or latent variables that the model is unable to capture or noise in the input data that is due to the imperfect data acquisition of the sensors.

Breaking down the total predictive uncertainty can be crucial because epistemic and aleatoric uncertainty provides us with different information about the input. A high value of the epistemic uncertainty suggests that the test input is an outlier with respect to the distribution of the training set. This type of uncertainty can be reduced by collecting more training data from the distribution from which the test data originated. In an ideal scenario, if we had access to an infinite amount of data, the output for every possible input would be known, the posterior distributions would be delta functions, and our epistemic uncertainty would collapse to zero. On the other hand, access to more data does not help with the aleatoric uncertainty. In order to reduce it, we need either more accurate measurements or further knowledge about the unobserved variables and add them as features.

Another reason for decomposing the predictive uncertainty is that depending upon the application, it might be beneficial to prioritize a single type of uncertainty. For example, in reinforcement learning, the aim is to explore the state-action space efficiently. Under such cases, it is favorable to collect data in regions of high epistemic uncertainty. Conversely, if the goal is to predict the return from different types of stocks, it is more beneficial to prioritize the aleatoric uncertainty. If the objective is to have a low-risk investment portfolio, we would want to pick stocks that have a high return but low aleatoric uncertainty. Such kinds of distinctions cannot be made from the overall predictive uncertainty only; inputs with a high predictive uncertainty can have significant contributions from the aleatoric part, epistemic part, or even both [223].

From the perspective of information theory, the entropy of a distribution is the average

level of uncertainty present in the random variable's possible outcomes. If we have a random variable following a uniform distribution, then the entropy of that would be quite high since information about a single sample would give almost no information about the subsequent samples; thus, the uncertainty is high. Alternatively, if we have a degenerate distribution that only takes a single value, then the entropy of that would be zero, as there is literally no uncertainty in the outcome.

With this in mind, we consider the entropy of the output softmax distribution as the predictive uncertainty, which for an input $\mathbf{x}^\star$, dataset $\mathbf{D}$, and softmax outputs $\mathbf{y}^\star$ is given by $\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{D}]$ [224]. This can be decomposed as the sum of two terms as

$$\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{D}] = \mathbb{I}[\mathbf{y}^\star,\mathbf{W}|\mathbf{x}^\star,\mathbf{D}] + \mathbb{E}_{\mathbf{W}\sim P(\mathbf{W}|\mathbf{D})}[\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{W}]]. \tag{4.11}$$

In equ. 4.11, $\mathbb{I}$ represents the information gain. The authors of [222] interpreted the two terms in this expression as the epistemic and aleatoric components of the predictive uncertainty $\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{D}]$. The second term, $\mathbb{E}_{\mathbf{W}\sim P(\mathbf{W}|\mathbf{D})}[\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{W}]]$ is the average entropy of the outputs when the weights are fixed, and so, the uncertainty solely derives from the input $\mathbf{x}^\star$, and not the weights. Hence, this term can be interpreted as the aleatoric uncertainty, and it quantifies the uncertainty in the predicted class using only a fixed set of weight values. Finally, we can rearrage equ. 4.11 to have $\mathbb{I}[\mathbf{y}^\star,\mathbf{W}|\mathbf{x}^\star,\mathbf{D}]\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{D}] = \mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{D}] - \mathbb{E}_{\mathbf{W}\sim P(\mathbf{W}|\mathbf{D})}[\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{W}]]$. This difference term is the epistemic uncertainty since it is the remaining uncertainty from the probabilistic nature of the model weights and not from the input. A high value of the epistemic uncertainty signifies that with each sample from the posterior distribution, the model predicts a different class [223].

4.4.2.1 Calculating aleatoric uncertainty

The calculation of the average entropy or the expectation requires us to evaluate integrals over all possible model configurations, which is practically impossible. We resort to performing the Monte Carlo average once more to calculate these values. In MC average for a quantity, we take samples from the model, calculate that quantity for each of them, and then compute their average over the different samples. If we have taken C model samples from the variational distribution. mathematically this can be written as

$$\begin{aligned}
\mathbb{E}_{\mathbf{W}\sim P(\mathbf{W}|\mathbf{D})}[\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star,\mathbf{W}]] &= -\int P(\mathbf{W}|\mathbf{D})\Big[\sum_k P(\mathbf{y}^\star=k|\mathbf{x}^\star,\mathbf{W})\log P(\mathbf{y}^\star=k|\mathbf{x}^\star,\mathbf{W})\Big]d\mathbf{W} \\
&\approx -\int q(\mathbf{W})\Big[\sum_k P(\mathbf{y}^\star=k|\mathbf{x}^\star,\mathbf{W})\log P(\mathbf{y}^\star=k|\mathbf{x}^\star,\mathbf{W})\Big]d\mathbf{W} \\
&\approx -\frac{1}{C}\sum_c\sum_k P(\mathbf{y}^\star=k|\mathbf{x}^\star,\mathbf{W}^{(c)})\log P(\mathbf{y}^\star=k|\mathbf{x}^\star,\mathbf{W}^{(c)}).
\end{aligned} \tag{4.12}$$

4.4.2.2 Calculating epistemic uncertainty

In the same vein, the epistemic uncertainty is calculated as the difference as

$$\begin{aligned}\mathbb{I}[\mathbf{y}^\star, \mathbf{W}|\mathbf{x}^\star, \mathbf{D}]\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star, \mathbf{D}] &= \mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star, \mathbf{D}] - \mathbb{E}_{\mathbf{W}\sim P(\mathbf{W}|\mathbf{D})}[\mathbb{H}[\mathbf{y}^\star|\mathbf{x}^\star, \mathbf{W}]] \\ &\approx -\sum_k \left(\frac{1}{C}\sum_c P(\mathbf{y}^\star = k|\mathbf{x}^\star, \mathbf{W}^{(c)})\right) \log\left(\frac{1}{C}\sum_c P(\mathbf{y}^\star = k|\mathbf{x}^\star, \mathbf{W}^{(c)})\right) \\ &\quad + \frac{1}{C}\sum_c \sum_k P(\mathbf{y}^\star = k|\mathbf{x}^\star, \mathbf{W}^{(c)}) \log P(\mathbf{y}^\star = k|\mathbf{x}^\star, \mathbf{W}^{(c)}).\end{aligned} \tag{4.13}$$

4.5 Two Moons dataset

4.5.1 The dataset and methods

To showcase the advantages of our Bayes BiNN, we first consider a toy dataset called the 'two moons' dataset [225]. This is a simple task where we have two classes of data points that interleave with each other in a two-dimensional plane in the form of two semi-circles. This dataset is shown in fig. 4.5 (a), where the red triangle and blue circle points denote the two classes. This is a classification task where a set of such points are used to train our model, and the goal is to classify any test point in that region.

The neural network model used for this task has a fully connected architecture with two hidden layers, each with 64 neurons, two input neurons providing the x and y coordinates of a point, and the single output is the prediction for the point to be in the blue circles class. The softmax output is represented by the color map, which ranges from red (belonging to the red triangle class) to blue (belonging to the blue circle class). We generate the training dataset from the scikit learn datasets package using the command sklearn.datasets.make_moons, and use it to train our model. In this section, we present our findings by comparing the Bayes BiNN to the deterministic implementation of binary neural networks having the exact same architecture, following the work of [210]. From here onwards, for all the 2-plots in the $x-y$ plane, the left plot represents the deterministic version, while the right represents the output from the BayesBiNN method. During testing, we input all the points in this region of space to yield the color map we see in the background of fig. 4.5 (a). For the deterministic network, the softmax value of the output is plotted in the color map, whereas in the Bayes BiNN, the mean of the outputs from many sampled models is shown.

From fig. 4.5 (a), we observe the following features: both the deterministic and Bayesian networks learn the decision boundary between the two classes perfectly. The first prominent difference between the two is in terms of the abruptness of the decision boundary; the deterministic output softmax values jump quite abruptly from 0 to 1 as we move across the boundary. On the other hand, the boundary is wider and much spread out in the case of the Bayesian

network, and the more gradual change from one class output to another is quite clearly visible. If we follow the decision boundary further in the regions where there are little to no training data points, we shall see the breadth of the boundary increase substantially. The mean softmax value lies between 0.3 and 0.7 in this region, and it means where we have less training data, the model is uncertain about the output. In contrast, the same region for the deterministic network has the same sharp decision boundary, and significant parts of the region yield output of 0 or 1. Also, in this area, the boundary appears linear, for which there is simply no information from the training dataset. This ties in with the idea that non-probabilistic deep neural networks are typically prone to overfitting, whereas their probabilistic counterparts handle this more realistically by producing ambiguous outputs.

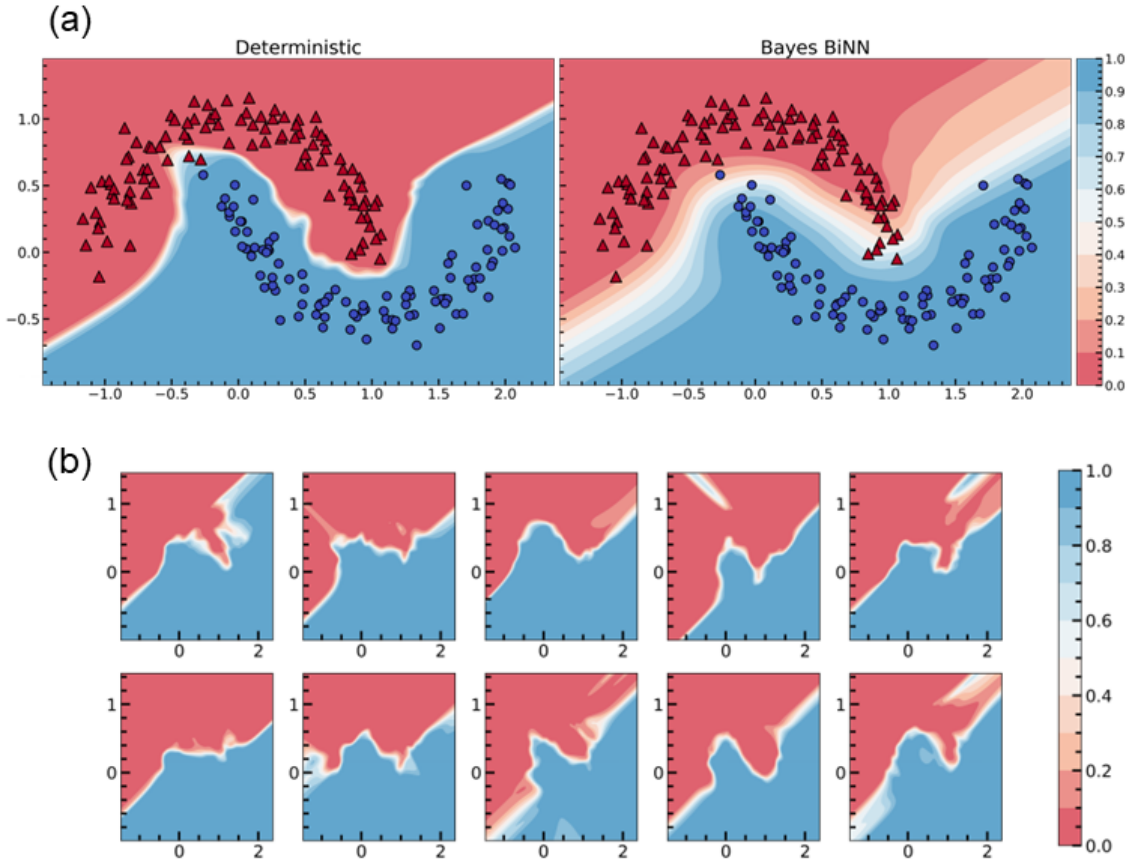


Figure 4.5: The two moons dataset. (a) The output of the deterministic and Bayes BiNN with the same architecture is for the task of classifying two sets of points in a two-dimensional plane. (b) Outputs corresponding to ten sampled Bayes BiNN models from the learned distribution.

The Bayes BiNN output is the result of averaging over the outputs from several sampled models, and this provides a distribution of the output probability. To understand the impact of this on our Bayesian output, we check the outputs of each individual model. After training, we have learned the Bernoulli distribution parameter probability p_j for a synapse to be in the +1

state. We use these probabilities to sample from the Bernoulli distribution and get C individual deterministic models. For the simulation results, we take an average of C=500 samples, ten of which are shown in fig. 4.5 (b). Firstly, if we just look at the decision boundary, it is very similar to the deterministic network; the transition between the classes is abrupt. Secondly, let us focus on two different locations in the ten plots: the top right corner, where there are no training data points, and the central region, where the two moons interleave. For the central region, for most of the plots, we can clearly identify the curved decision boundary. Still, the top right portion is significantly different and appears random in most of the output instances. Since the output values in this area are randomly varying, the average over many samples yields values close to 0.5 and smoothly changes to 0 or to 1 for the two classes.

4.5.2 Uncertainty quantification

Until now, we have qualitatively discussed how the Bayes BiNN model is good at expressing uncertainty in our inference. Here, we implement the uncertainty quantification methodology discussed in the previous section. We use equ. 4.12 and equ. 4.13 to calculate the predictive, aleatoric, and epistemic uncertainties for all the test points in the region. The results are shown in fig. 4.6, where now the color map represents the uncertainty value for the particular type of uncertainty.

The predictive uncertainty in fig. 4.6 in the value range of 0.6-0.75 (shown in yellow) forms a thick strip that spans along the diagonal of the region, curving between the two classes. The total uncertainty decreases as we move away from this central region towards where there are more data points. In the region above the red triangles and below the blue circles, the uncertainty is essentially zero, as there is little doubt as to which class that region would belong to.

We further decompose the total predictive uncertainty into its epistemic and aleatoric components to gain further insight. The epistemic uncertainty peaks in three regions; the bigger two of them lie in the part where there are no data points and a small one exactly between the two classes. The epistemic uncertainty quantifies the uncertainty in the data arising from the lack of knowledge, or in this case, lack of data points in those regions. On the other hand, the aleatoric uncertainty lights up in the parts where the edges of the two moons almost intersect. This is consistent with the concept of aleatoric uncertainty, which is a measure of the ambiguity in the data. These intersecting sections have red and blue points in very close proximity, and the model is uncertain between the two classes.

4.5.3 Impact of dataset size

We study this dataset under two circumstances that emulate the conditions of more realistic bio-medical tasks. This type of dataset is typically smaller than other domains, and here, we highlight how the Bayes BiNN outshines the deterministic network in a small data setting [226, 227].

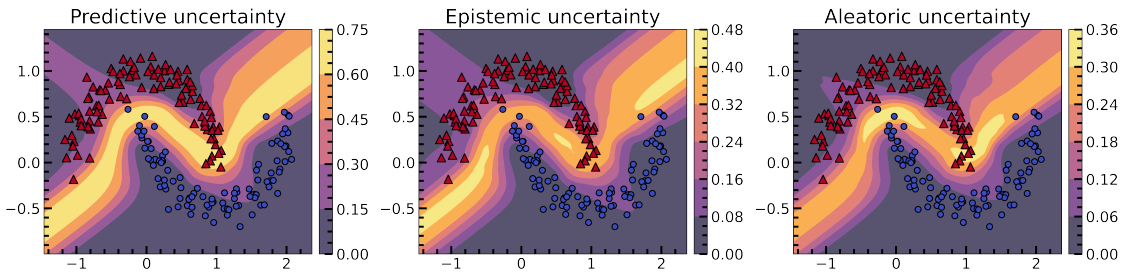


Figure 4.6: Uncertainty quantification in the two moons task. The color map shows the predictive, epistemic, and aleatoric uncertainties in the three plots, respectively. The predictive uncertainty is decomposed into two components with different sources; epistemic for regions with no training data points and aleatoric for ambiguous parts.

We perform the same two moons task but with less number of training points and observe the resulting inference color maps. Fig. 4.7 shows the inference when the number of data points used for training varied between 10 and 60. For the deterministic network, we note that the decision boundary is either highly irregular (10, 20, 50) or there are unphysical strips or islands of regions with a different class (10, 30, 60). This is again due to the propensity of deterministic models to overfit data. In the inference with 30 data points, there is a single red point isolated from the others, and that is enough to cause a strip of red to form inside the blue region. On the other hand, although the Bayes BiNN does not give us perfect inference like in fig. 4.5 (which had 200 data points), we do not see the *artifacts* that we see for the deterministic case. The overall shape of the broad decision boundary is quite realistic, even for the number of data points as small as 20. Another noteworthy feature of the Bayesian case is that the regions with high outputs (greater than 0.9 or less than 0.1), i.e. the deep blue and red portions, cover a small part of the whole area. This reflects that our model has less confidence in its prediction of the output, which is a suitable conclusion given the lack of sufficient data.

Next, we wanted to evaluate the uncertainty in the inference under such small data conditions. To do so, for each number of data points in the training set, we calculated the uncertainties for the whole region during inference. In fig. 4.8, we plot the uncertainties averaged over all points in this region as a function of the total number of training data points. As expected, the predictive uncertainty, which captures the overall uncertainty, decreases as we have more data points. The epistemic uncertainty also decreases, but the aleatoric component remains almost constant. This again exemplifies that epistemic uncertainty is related to the availability of information or knowledge about the total distribution. With more points, the training dataset captures the original distribution in a much better way, thus reducing the epistemic uncertainty. On the other hand, the aleatoric component is related to the ambiguity between the two classes, which is unaffected by the availability of data in training.

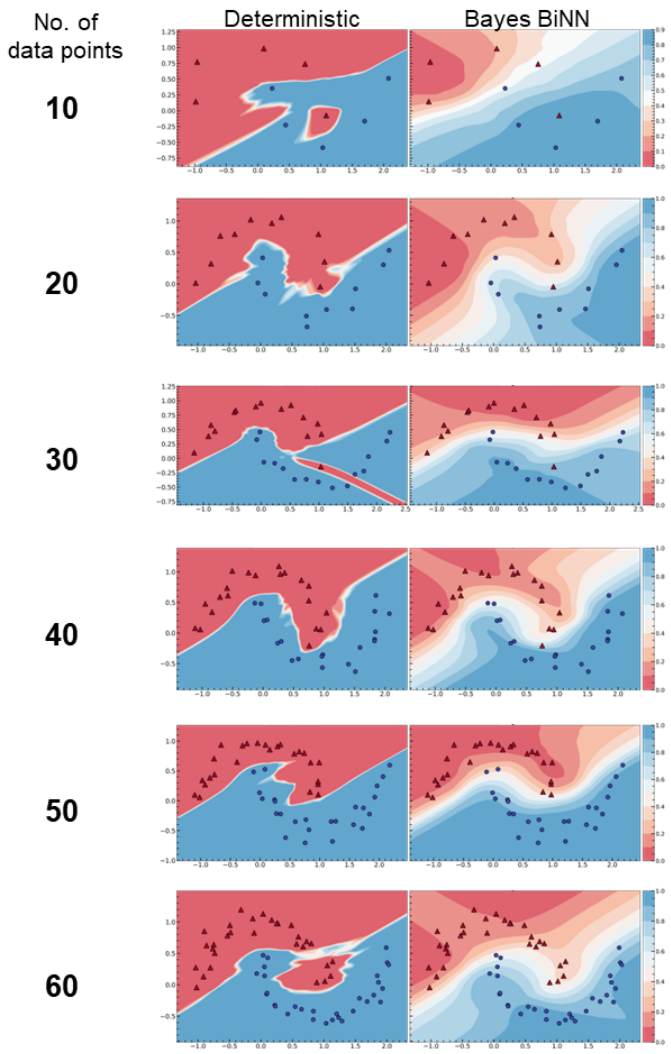


Figure 4.7: Inference color maps for the two moons task for a small number of training data points in both deterministic and the Bayes BiNN models. The color map denotes the output softmax value for the deterministic case, and the average of the output softmax of models sampled from the trained Bayes BiNN.

4.5.4 Impact of label noise

The success of deep learning algorithms is attributed to a large extent to the quality of the large quantity and highly processed data that is available today. However, if the data is not perfect, the performance of conventional deep learning models decreases, especially for small-sized datasets. Here, we specifically focus on the impact of *mistakes* in our two moons dataset to emulate the occurrence of label noise in our training dataset [228, 229]. Label noise in machine learning refers to the situation where the labels or target values assigned to the training data points are incorrect. Label noise can arise due to errors in data collection, human annotation errors, or data corruption during transmission or storage. These errors cause the model to learn

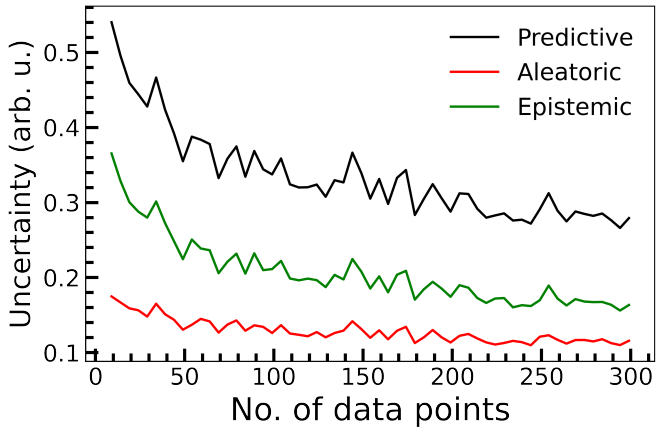


Figure 4.8: Uncertainties for the two moons task as a function of the total number of training data points. The black, green, and red lines show the predictive, epistemic, and aleatoric uncertainties respectively. For a fixed number of data points, the uncertainty plotted is the average uncertainty over the whole domain.

incorrect patterns, resulting in poor generalization and low accuracy when applied to new data. This can have catastrophic consequences in the medical domain, where labeling of data can be particularly difficult, and this necessitates the quantification of uncertainty, alongside having a decent prediction accuracy.

For our experiment, we define the probability of mistakes, $p_{mistake}$: the probability of a training datapoint being mistakenly classified to the other class. For illustrative experiments, we consider three scenarios in terms of the size of the training dataset: small (with 20 data points), medium (100), and large (500). In fig. 4.9, we show the results and the training dataset for $p_{mistake}$ values ranging from 0.05 to 0.3.

For the small data setting (the first column in fig. 4.9), we observe that until for all values of $p_{mistake}$ shown here, the inference color maps show performance degradation. This is because we have very few data points in the first place, and a few mistakes substantially impact the training procedure, especially for high values of $p_{mistake}$. However, the Bayes BiNN output is more continuous and does not usually have the artifacts and irregular boundary shapes like the deterministic output. Also, the regions of high certainty (more than 0.9 or less than 0.1) are limited to small parts, whereas for the deterministic, almost the whole plane is panned by them. Hence, although the Bayes BiNN does not clearly perform better for small data, it still captures the uncertainty in the small dataset size.

In the third column of fig. 4.9, we observe the other extreme, where we have 500 data points that define our training dataset distribution concretely. In this case, even for high values of $p_{mistake}$, the Bayes BiNN inference gives a reasonable output, albeit not perfectly. This is in contrast to the deterministic case where for high $p_{mistake}$ values, the inference color map is highly irregular but erroneously gives high confidence from the high output softmax values. Even for a small $p_{mistake}$ value of 0.05, we observe unrealistic spikes in the color map, which

worsens with higher mistake probabilities. The inference for the medium dataset size (middle column) is somewhat in-between these two in that the irregular nature of the deterministic inference is always more pronounced, and the Bayesian network, although not always showing good results, is relatively uncertain.

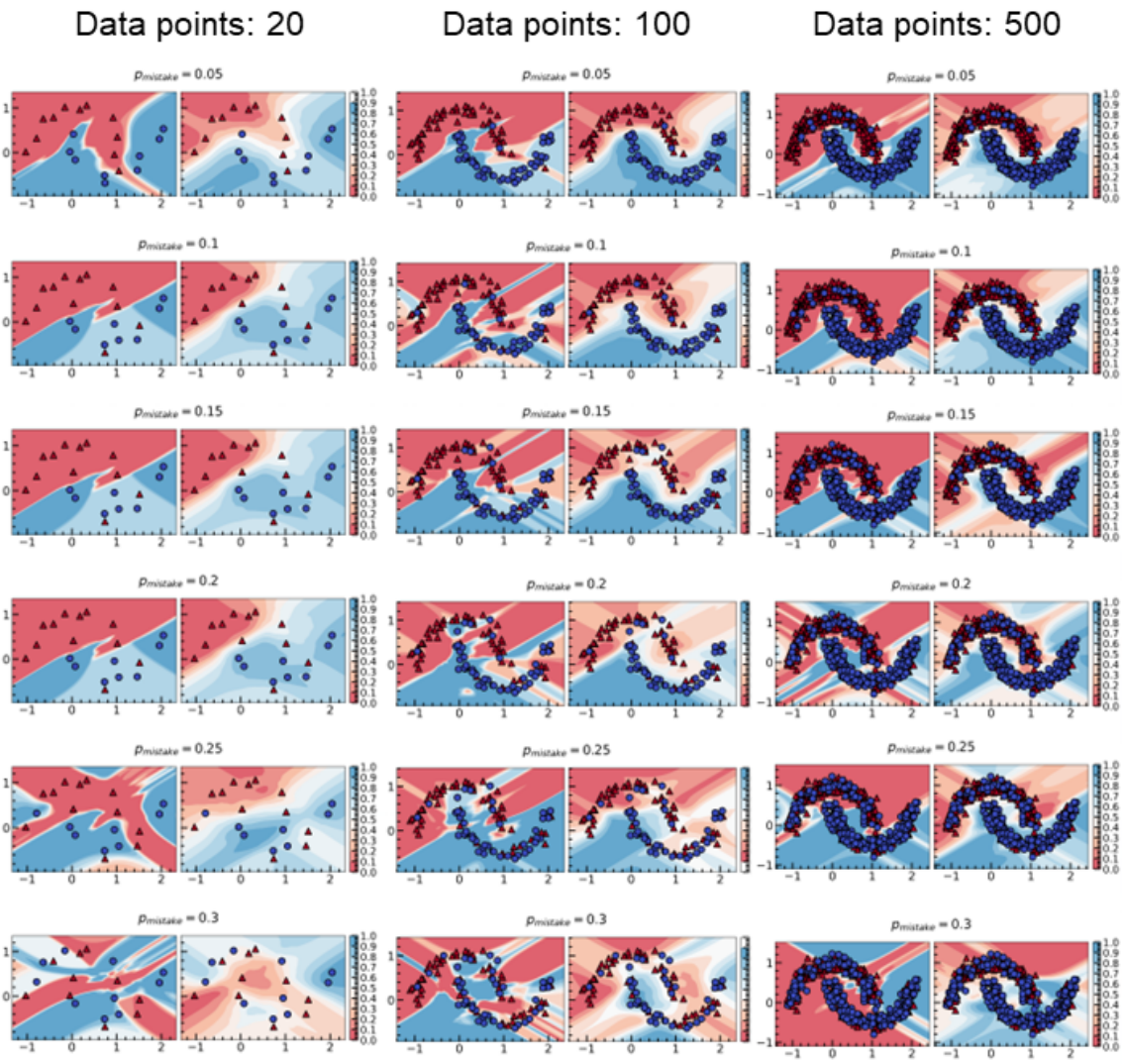


Figure 4.9: Impact of label noise on the two moons task. The inference color maps for the deterministic and Bayes BiNN and the dataset with label mistakes are shown for three train dataset sizes: 20, 100, and 500 data points. The probability of a single train data being mistakenly labeled in the other classes, $p_{mistake}$, is varied from 0.05 to 0.3 for the different dataset sizes.

The inference color maps give us a qualitative picture of how the $p_{mistake}$ impacts our prediction quality. Still, to understand the uncertainty, we need to quantify it. We perform the same uncertainty calculations as done to produce fig. 4.8, except for this, we vary the $p_{mistake}$ for the small, medium, and large number of data points. Fig. 4.10 shows the uncertainty evaluation for the three different dataset sizes. In all three cases, the predictive uncertainty increases

with $p_{mistake}$; the Bayes BiNN model can directly reflect the noise in the training data labels. For a small dataset with 20 data points, the epistemic uncertainty dominates the aleatoric uncertainty by almost a factor of two. This is expected since we are in a regime with fewer data or knowledge about our true dataset. If we increase the dataset size to 100, we see that the predictive uncertainty values remain pretty similar to the 20 data points case, but the epistemic uncertainty has reduced and is now almost comparable to the aleatoric uncertainty. We see a reversal in the trend in the large dataset regime where we have 500 data points, where for $p_{mistake}$ values higher than 0.05, the aleatoric uncertainty is higher than the epistemic uncertainty, which has now reduced to be at a level that is less than 0.2, which used to be about 0.35 for $p_{mistake}$=0.4. This trend reversal can be understood from the fact that with 500 data points, the epistemic uncertainty is significantly reduced because we have more information about the general dataset distribution. Under this scenario, the noise in the dataset from incorrect labeling becomes more prominent, and the aleatoric uncertainty becomes the major contributor to the predictive uncertainty.

From the experiments shown in this section, it can be concluded that for the two moons dataset, the Bayes BiNN model gives us more robust, realistic predictions than its deterministic counterpart. Another significant advantage of Bayes BiNN is the ability to decompose the uncertainty into two components that provide us with different information and can have a significant impact under more realistic conditions where the dataset size is small, or the train dataset labels are incorrect.

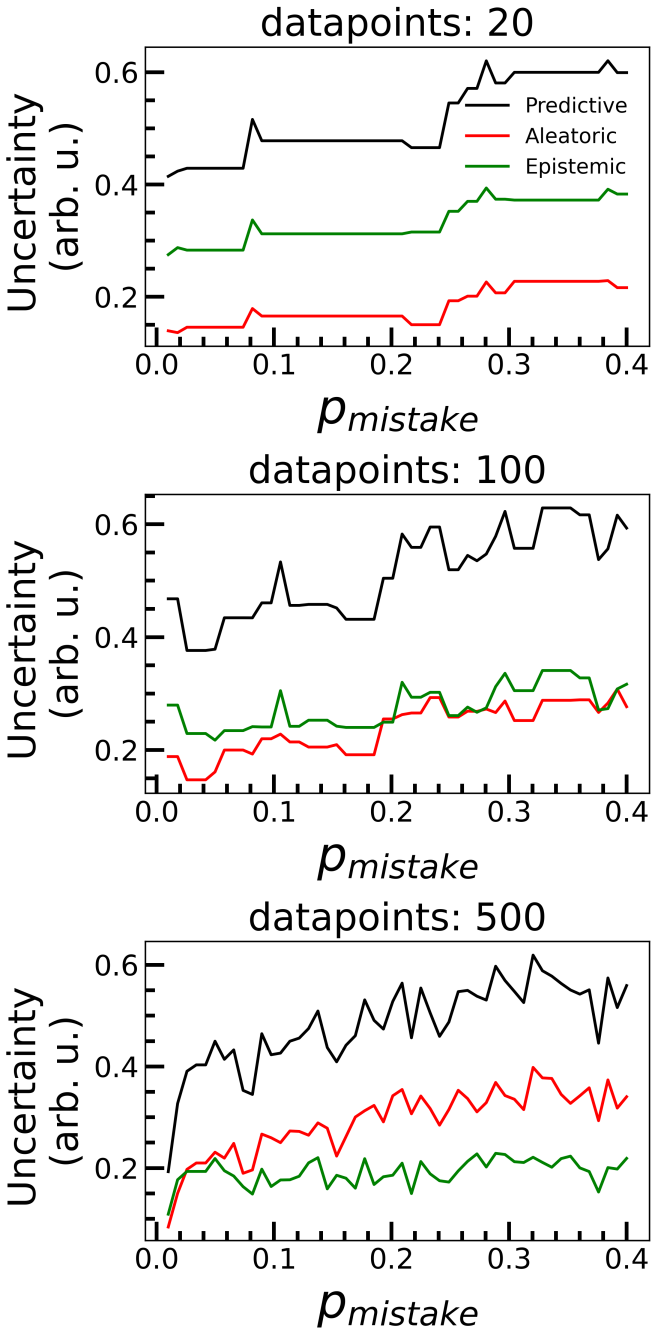


Figure 4.10: Uncertainty estimation in the two moons dataset with label noise. The predictive (black), aleatoric (red), and epistemic (green) uncertainties are evaluated for different values of $p_{mistake}$ and for 20, 100, and 500 training data points.

4.6 Medical task

4.6.1 The dataset and methods

In the previous section, we elaborated on some scenarios where the Bayesian implementation of neural networks is superior to its deterministic analog. However, we only considered an illustrative toy task. To assess the scalability of this approach in more practical scenarios, we conducted a study on a real-world medical dataset. Our focus was on the MIT-BIH Arrhythmia database, where the aim was to classify heartbeat rhythms.

The MIT-BIH Arrhythmia Database is a collection of electrocardiogram (ECG) recordings compiled by researchers at the Massachusetts Institute of Technology and Beth Israel Hospital [230, 231]. It is one of the most widely used datasets for evaluating algorithms that detect and classify cardiac arrhythmias [232]. The dataset contains 48 half-hour ECG recordings, each of which includes two simultaneously recorded leads. These recordings were obtained from 47 patients, most of whom had a history of cardiac arrhythmias. The recordings were digitized at a sampling rate of 360 Hz and annotated beat-by-beat by human experts, who identified the type of beat (e.g., normal sinus rhythm, premature ventricular contraction, etc).

For our simulations, we perform the following preprocessing on the data. First, we do a Fourier transform of the ECG sequence for a single heartbeat and then select 32 features using the scikit-learn package SelectKBest. This is then fed to a fully connected feedforward neural network with two hidden layers, each with 1,024 neurons. In the whole dataset, twenty different types of annotations correspond to the heartbeats representing different classes of arrhythmia, which are different ways the heart can beat. Out of them, we consider five categories that appear quite frequently, including the regular beats.

- **N** (Normal beat): This class corresponds to normal sinus rhythm, which is the regular beating of the heart.
- **L** (Left bundle branch block beat): Heartbeat characterized by a delay in the activation of the heart's left ventricle.
- **R** (Right bundle branch block beat): Heartbeat that is characterized by a delay in the activation of the right ventricle of the heart.
- **A** (Atrial premature beat): Heartbeat that originates in the atrium of the heart rather than the sinoatrial node (the natural pacemaker of the heart).
- **V** (Premature ventricular contraction): Heartbeat that originates in the ventricles of the heart rather than the sinoatrial node.

4.6.2 Impact of dataset size

Similar to the two moons task, we investigate the impact of training dataset size on classification accuracy in the small dataset regime. Datasets in the medical domain are typically smaller

than other domains like computer vision or language processing, and the datasets are exceedingly scarce for rare diseases. We emulate this in the MIT-BIH dataset by considering only a part of the dataset to train our networks. Usually, in medical datasets, the "normal" class or the data corresponding to no disease is the most abundant, whereas the rare conditions are less represented. We balance all the classes in our dataset to prevent this from biasing our results. We keep the testing dataset size to 1,600 and vary our training dataset size from 50 to 3,300. The resulting train and test accuracies for the deterministic and Bayesian networks are shown in fig. 4.11.

Firstly, the training dataset accuracy reaches almost 100% for both networks, so there is no indication of underfitting. Secondly, we observe that even for a small dataset size of 50 in which each class is represented only ten times, the Bayesian network reaches an accuracy of over 99%, and the corresponding accuracy for the deterministic network is less than 91%. It only approaches the Bayes BiNN accuracy for larger dataset sizes. This is, again, due to the susceptibility of deterministic networks to overfit to data which is evident from the difference in train and test accuracies, especially when the dataset is small. A measly dataset size of 50 exposes the network to a tiny part of the training distribution, and it cannot learn the distribution very well using only point estimates. On the other hand, when we shift the problem to learning the distribution, it captures the total distribution more naturally. This is the essence of probabilistic learning; the idea of a probability distribution replacing the point estimates allows for more flexibility, reducing overfitting.

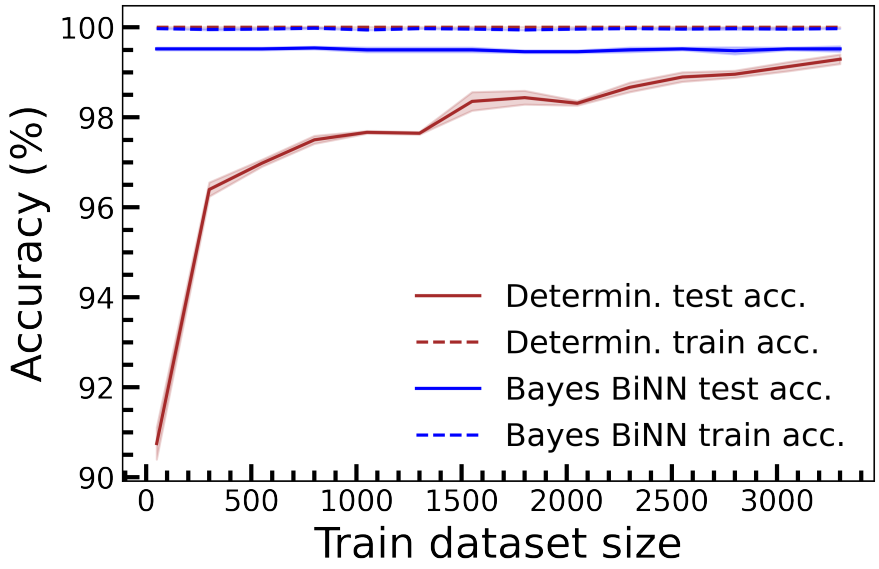


Figure 4.11: Train (dotted line) and test (solid line) accuracies of both deterministic network (brown) and Bayes BiNN (blue) as the training dataset size is varied from 50 to 3,300. The points represent the average accuracy, and the shaded region around it denotes one standard deviation.

4.6.3 Uncertainty under realistic scenarios

The Bayes BiNN architecture prevents overfitting, and the intrinsic probabilistic nature allows uncertainty quantification. This can be leveraged in many real scenarios to infer about the predictions or even about the datasets themselves. We explore one such situation where the test examples have a class not present in the training data. Biological diseases can mutate relatively fast and take a completely different form in their expression in the human body. This presents a challenge to the neural network that is trained on previously available data, which does not contain this newer mutated variant. Conventional neural networks, typically under such cases, would give wrong answers with high confidence. The ability to decompose the uncertainties for Bayesian neural networks provides a natural way to solve this.

To replicate such a scenario, we artificially remove the **L** class from the training dataset, keeping the test dataset the same. In particular, the reason for choosing this class can be understood if we look at fig. 4.12, where we plotted the two-dimensional t-SNE representation for our preprocessed input data for the test dataset, color-coded by the different classes. The green class **L** is isolated from the other clusters, with no outlier points closer to the other classes. This implies that the class in question exhibits distinct characteristics compared to the other classes, making it an ideal candidate for simulating an unseen class to test our neural network.

We train both our networks with the training dataset without this class, and for both, we get a test accuracy of about 78%. The other plots in fig. 4.12 show the same t-SNE but with the color coding representing the uncertainties. For the deterministic case, we compute the 'predictive uncertainty' by supposing that we have taken only one sample (C=1) and by using equ. 4.12. If we focus on the green cluster that represents class **L**, the epistemic and aleatoric uncertainties have higher values for most of the points in this cluster. In contrast, the highest uncertainty values for the deterministic case are more spread out between the **L**, **A**, and **V** classes. We plot the histogram of these values in fig. 4.13 (a) to look at the class-wise distribution of the calculated uncertainties. The deterministic network has high uncertainty for all three classes: **L**, **A**, and **V**, but for the Bayesian uncertainties, the green **L** class stands out with high values of uncertainties, especially for the epistemic uncertainty. For all other classes, the epistemic uncertainty is close to zero. As discussed before, epistemic uncertainty captures the uncertainty in the data about the already learned distribution. Unsurprisingly, this type of uncertainty is significantly higher for a class that the model has not seen during training.

Another way to investigate the uncertainties is to study the relationship between the correctness of a prediction and the corresponding uncertainty. This is shown for the deterministic and Bayesian networks in fig. 4.13 (b), where the green, red, and blue histograms show the correct, incorrect predictions, along with the unseen class, respectively. In the case of the deterministic network, the high values of the predictive uncertainty correspond to both correct and incorrect predictions and the unseen class. So, simply by looking at the uncertainty value, it is difficult to infer the confidence of the prediction. On the other hand, the epistemic uncertainty is quite clearly higher for the unseen class and is generally higher for incorrect prediction

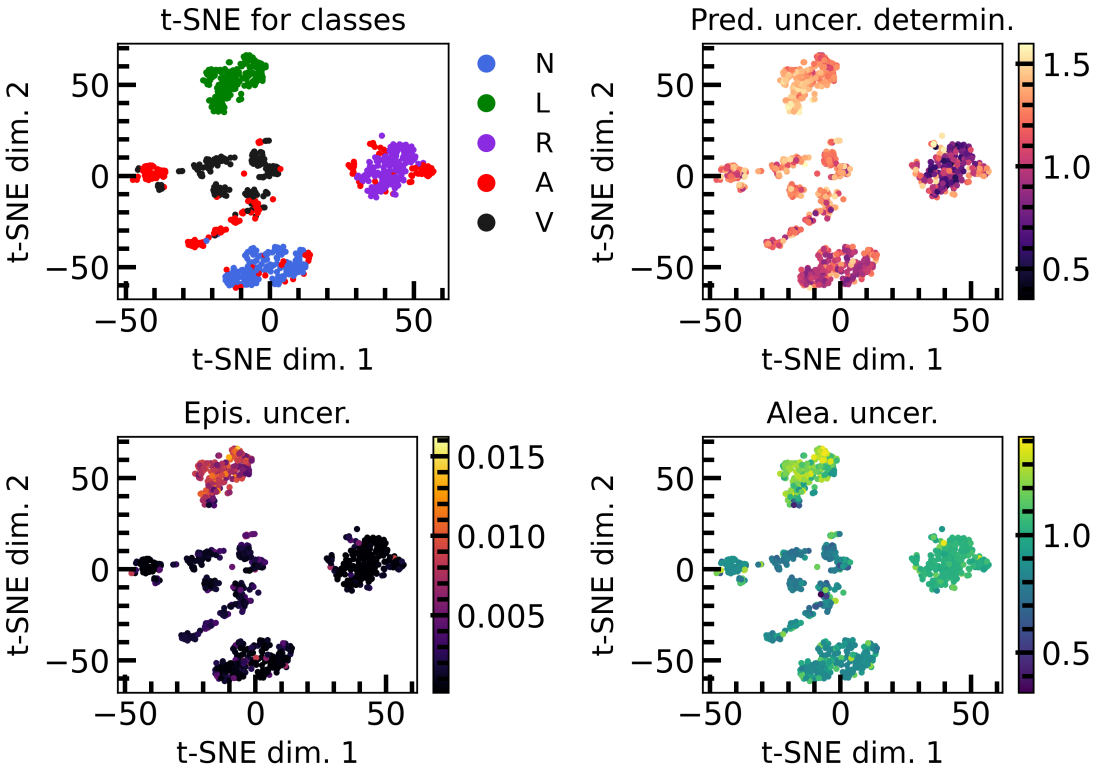


Figure 4.12: The two-dimensional t-distributed stochastic neighbor embedding (t-SNE) plots for the MIT-BIH test dataset. The color coding in the top left plot represents the different classes: **N**, **L**, **R**, **A**, and **V**. The color coding in the other plots shows the uncertainty in prediction for the deterministic 'predictive uncertainty' (top right), Bayesian epistemic (bottom left), and Bayesian aleatoric (bottom right) for the same t-SNE plots.

predictions. In the case of aleatoric, the correct predictions have a lower uncertainty than the unseen and incorrect predictions.

To summarize, uncertainty quantification gives us a handle on the confidence of our prediction in the following way. While testing, if both the aleatoric and epistemic uncertainties are low, it is very likely that the prediction is correct. On the other hand, if the epistemic uncertainty is low and the aleatoric uncertainty is higher, it signifies that probably the test example is not the unseen class, but it is an ambiguous example (might be class **A** or **V**). Finally, if both the epistemic and aleatoric uncertainty is high- we can infer that this is an example from a class not included in the training dataset.

In these simulations, we observed two aspects that differed from the two moons dataset in the previous section, which needs to be discussed here. Firstly, the epistemic uncertainty we get for the MIT-BIH arrhythmia classification task is about one order of magnitude less than the aleatoric uncertainty; in the two moons task, the range of values for the two kinds of uncertainties was similar. Secondly, in the two moons task, the regions of high aleatoric and epistemic

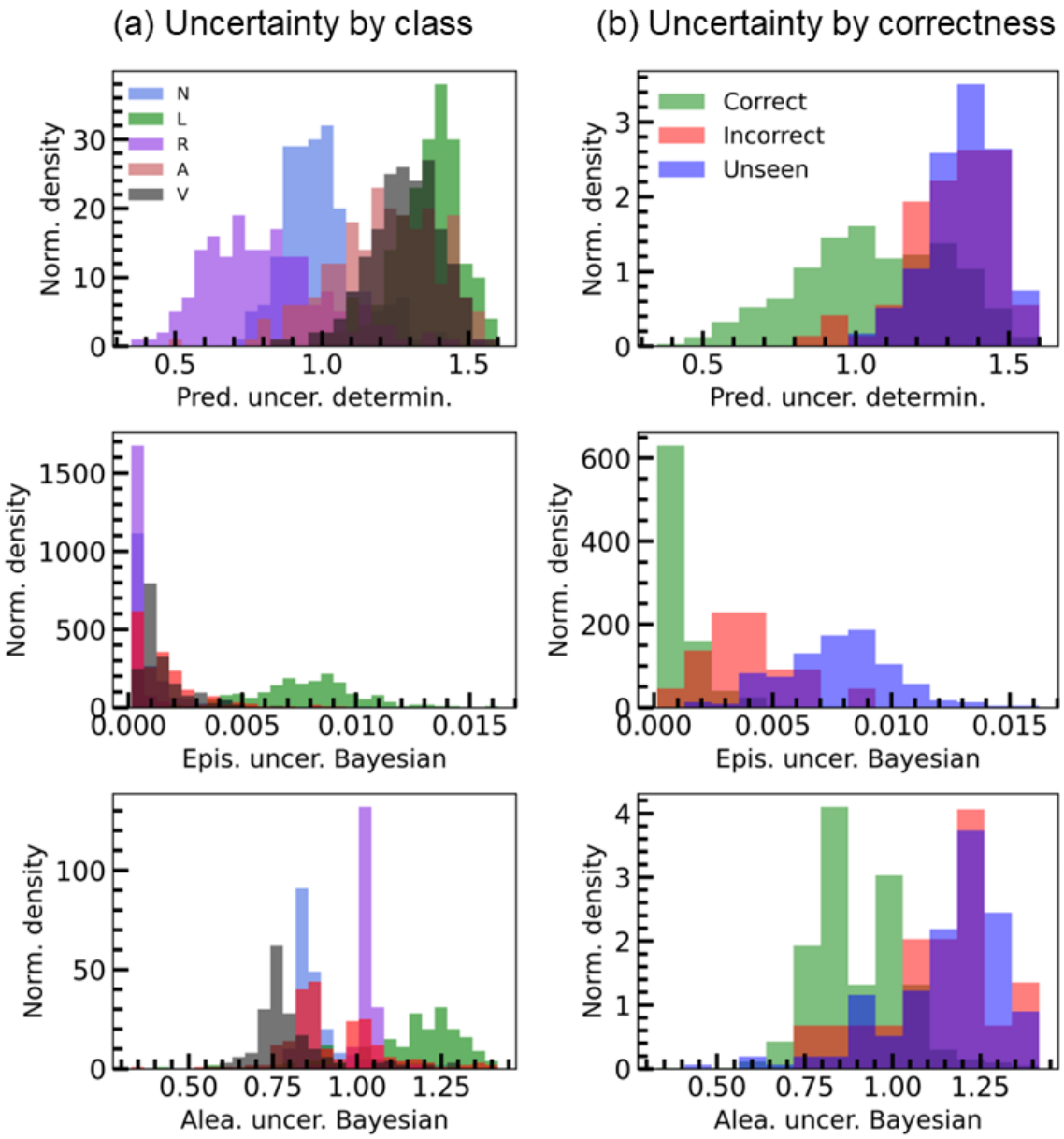


Figure 4.13: Distribution of uncertainties by class and correctness. (a) Histogram of the different uncertainties for both deterministic and Bayes BiNN for the five classes present in our dataset. (b) Histograms of the same uncertainty values but according to the correctness in prediction (red for incorrect and green for correct). The unseen class **L** predictions are almost always incorrect; hence, they are shown separately (blue).

uncertainties were quite different- the epistemic was higher in the area with no training data points, and the aleatoric was higher where the two moons overlapped. In the medical task, we observe that for the unseen data, both the epistemic and aleatoric uncertainties are pretty high.

The possible explanation lies in the datasets' fundamentally different nature or our training algorithm. As for the datasets, the two moons task is a simple, almost caricaturish task,

whereas the MIT-BIH is a traditional benchmark task for neural networks in medical applications. When we calculated the Bayesian uncertainties for the two moons task, we calculated it for the whole region. We take x and y points across the entire domain and compute the prediction and uncertainty. On the other hand, we cannot *choose* the test samples over which we evaluate our network. Since all the data comes from ECG recordings, any two examples are not too different. Hence, learning the distribution for the other classes might not be very different from the particular class it has not seen during training, and this can explain the generally low value of epistemic uncertainty. Also, this can account for the simultaneous high values of both aleatoric and epistemic uncertainties; the distinction between ambiguity and lack of knowledge is unclear when all the examples look similar. The other possible explanation pertains to the algorithm we use for training; the training algorithm has hyperparameters that can significantly impact the training, such as the temperature parameter (explained in Appendix. 4.8) or how we initialize the Bernoulli parameters of the Bayes BiNN. There may be a regime of values for these hyperparameters wherein the epistemic uncertainty is more pronounced. However, more simulations must be performed on multiple datasets to assert either of the two hypotheses discussed here.

4.7 Spintronics-based implementation

So far in this chapter, we have discussed the probabilistic paradigm of machine learning, focussing on the Bayesian binary neural networks, which outperform the conventional deterministic network in small datasets and in terms of quantifying uncertainty in a prediction. This section discusses how this particular kind of neural network aligns well with a specific class of physical materials, namely Spintronic devices.

Spintronics is the spin-based analog of electronics, where instead of only the charge of electrons, the spin degree of freedom is studied in solid state devices [233, 234]. The spin can be used for both memory and processing, allowing for a versatile, low-power alternative to the typical electronic devices [235–237]. For this thesis, we only consider using such devices as memory. MRAMs are based on magnetic tunnel junctions (MTJs) in which a thin insulating layer is sandwiched between two ferromagnetic layers. The relative orientation of the spins encodes the resistance state of the device via the tunneling magnetoresistance effect. If the magnetization states are parallel, the device is more conducting compared to when they are aligned in the opposite direction. The retention time of a programmed memory device is related to the stability of the state. Regarding the energy landscape, the parallel and antiparallel states are the two local minima with a barrier in between. For standard MRAMs, the height of this barrier, E_B, is typically much larger than the thermal energy $k_B T$ at room temperature (T), and the retention time is of the order of 10 years [113]. The height of this barrier is directly related to the physical dimensions of the MTJ, and if we make them small enough, we can make $E_B \sim k_B T$. At this point, the thermal noise drives a rapid and stochastic magnetization switching between the two states; such systems are also referred to as superparamagnetic tunnel junctions [238].

Thus, nanomagnetic devices can be natural candidates for the weights of the Bayesian Binary neural network owing to their stochastic nature and the binary nature of the states. Now we elaborate on the device-related requirements for implementing such networks in hardware and then discuss some possible candidates for their realization.

4.7.1 Candidate systems

From an algorithmic point of view, a device representing a weight can be derived from any physical system having the following three features.

- The quantity representing the weight has to have two clearly defined, stable states.
- It has to be stochastic, and the fluctuation rate needs to be higher than the sampling rate.
- It is crucial that the probability can be programmed to any value between 0 and 1. We investigate later the impact of not fulfilling this criterion completely.

These attributes are represented in fig. 4.14 (a) and (f), wherein a physical system has two states, "-1" and "+1," energetically separated by a barrier of the order of thermal energy. Fig. 4.14

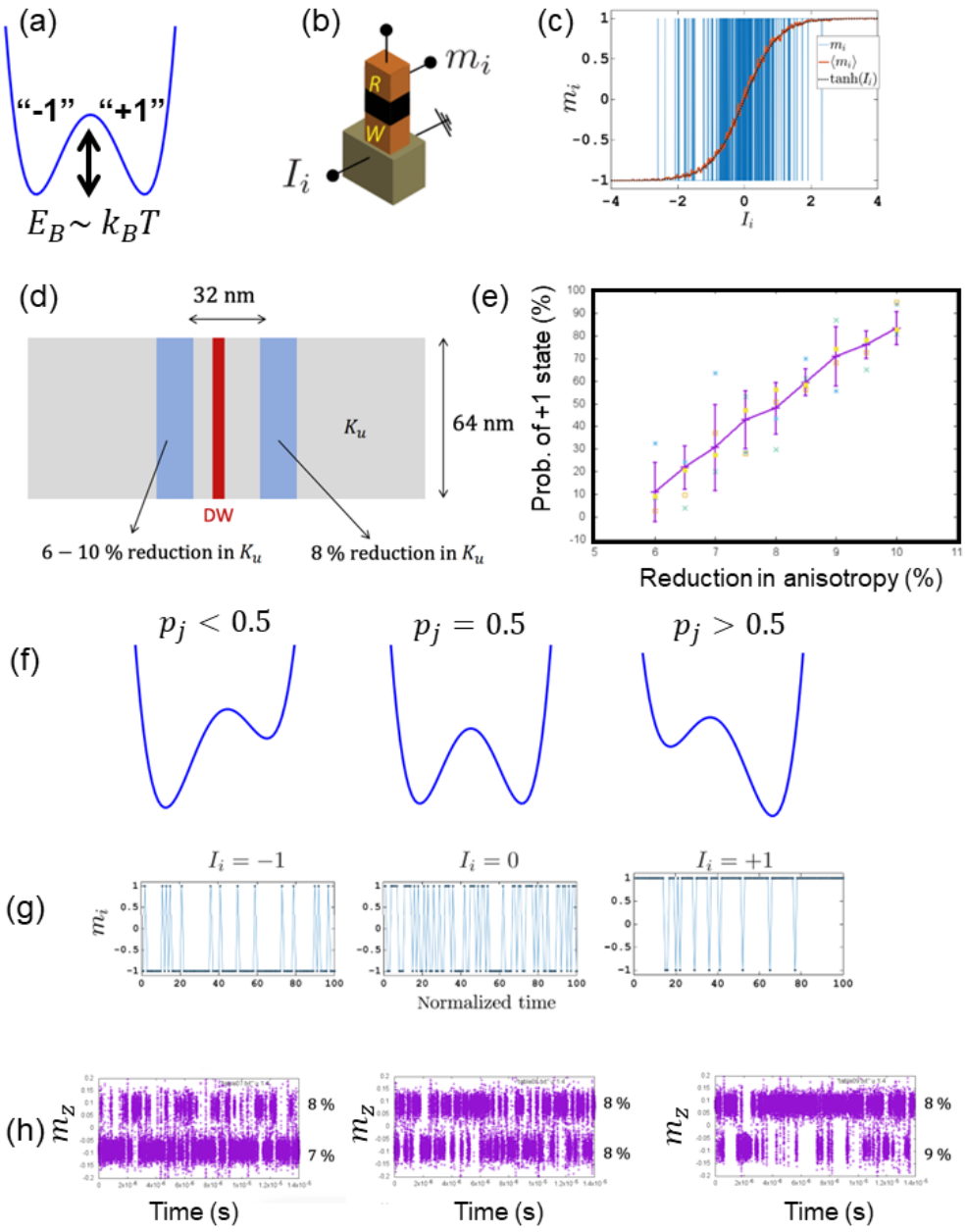


Figure 4.14: Device implementation of Bayes BiNN. (a) Energy landscape of a stochastic element where the two states, represented by -1 and +1, are local minima and the barrier between them $E_B \sim k_B T$. (b) A p-bit device with three terminals for the input I_i and output m_i. (c) The output of the p-bit as the input is ramped from -4 to 4. (d) The proposed DW hopping system in a CoFeB magnetic strip where the DW (red) is nucleated between two regions (blue) of reduced magnetic anisotropy. (e) The variation of the programmed probability in terms of the reduction in anisotropy of the left blue region. (f) The energy landscape for different values of the probability. (g) and (h) Evolution of the output magnetization with time for the p-bit and DW hopping systems, respectively, for different biasing conditions [239].

(f) shows that if we can tune the double-well energy landscape, we can produce different probabilities for the resultant physical variable, which would now follow a Bernoulli distribution with a certain probability. Here, we discuss two systems that can potentially serve the purpose.

4.7.1.1 p-bits

A probabilistic bit, or p-bit, is a special kind of memory bit that encodes the probability of being in a state. It is different from the classical bit, which always has a unique value of 0 or 1, and also from a qubit in a quantum mechanically superposed state of 0 and 1 [239, 240]. An MTJ can realize this p-bit in either a 1T1R configuration using a SOT-MRAM or as a 3-terminal device in the STT-MRAM, where the biasing from the input current or the transistor gate voltage respectively changes the probability value. As shown in the middle panel of fig. 4.14 (g), under no input to the MTJ, the resistance state of the device fluctuates uniformly between the LRS and HRS. This corresponds to the 0.5 probability state, which can be changed by the biasing voltage or current to favor one of the two states, effectively tuning the probability. This is shown in fig. 4.14 (c), where the output of a p-bit is plotted as the input is ramped from -4 to 4. A high negative or positive input biases the device quite strongly to one of the states, and on the other hand, for no input, the two states are equally likely. A p-bit is a three-terminal device, as depicted in fig. 4.14, where the inputs I_i are used to write the probability state, and the other two terminals are used to read the output m_i.

A major challenge with such memories is that the biasing conditions need to be applied as long as the p-bit is being used as a weight, which is unsuitable for the power-conscious applications we are trying to achieve. It will be ideal if there is a way to passively tune the energy landscape that does not require the continuous application of voltage or current. Another significant issue stems from programming the extreme probability values of 0 or 1. The average output is related to the input as $\langle m_i \rangle = \tanh(I_i)$ and the probability to be in the 1 state, p_i, is linked to the output as $p_i = \frac{\langle m_i \rangle + 1}{2}$, resulting in the input depending on the probability as $I_i = \text{arctanh}(2p_i - 1)$. So, to program a state of $p_i = 1$ or $p_i = 0$, we need a positive or negative infinitely large input, which is practically impossible. In real systems, we are limited by a power budget that limits the input to I_{max}, and that subsequently limits how close to the value of 0 or 1 we can program to, which being $\frac{\tanh(I_{max})+1}{2}$ or $\frac{\tanh(-I_{max})+1}{2}$.

4.7.1.2 Magnetic domain walls with VCMA

Another more unconventional approach is to utilize magnetic domain walls in a nanomagnetic strip. Magnetic domains are regions within a material where the atomic magnetic moments align in a particular direction. In a ferromagnetic material, for example, the magnetic moments of the atoms are aligned in the same direction, giving rise to a net magnetic moment. However, the magnetization of a material can be complex and have regions where the magnetization direction changes. These regions, where the magnetic moments point in different directions, are called magnetic domain walls (DWs) [241]. These DWs can be stable and be moved by low

currents, and they can be used as a special class of memory called racetrack memory [242]. Here, we propose a different usage for the DWs: as stochastic elements for encoding the Bayes BiNN weights.

DWs are pretty sensitive to variations in the local magnetic properties, which can be prominent in nanoscopic systems where the width of a magnetic wire is less than 100 nm. In this regime, the DW can get easily pinned to a location depending on the local value of the magnetic anisotropy or the presence of defects [243]. Also, in such small dimensions, the thermal energy-driven fluctuations in the position of the DW are quite significant [244, 245]. The idea here is to utilize such properties of DWs to create a stochastic unit where the stochasticity can be controlled. This idea was discussed and developed by Pr. Luis López Díaz from the University of Salamanca, who also performed the micromagnetic simulation results shown here. As shown in fig. 4.14 (d), a DW can be nucleated in a CoFeB magnetic wire of width 64 nm. Then, two regions of reduced magnetic anisotropy are created beside the DW, which are only 32 nm apart. These two areas serve as pinning sites for our DW and owing to the small dimensions, the DW hops between the two pinning sites. Micromagnetic simulations were performed with such a configuration under room temperature conditions, and the magnetization (m_Z)at the center of the wire was monitored over time, as illustrated in fig. 4.14 (h). Suppose the two regions experience the same reduction in uniaxial magnetic anisotropy (8%). In that case, their pinning strength is equivalent, and the probability of the domain wall being pinned at either site is equal. We see this in the central panel of fig. 4.14 (h), where the probability of being in one of the states is 0.5. Now, if the reduction of anisotropy in the right region is kept the same and only for the left region is varied from 6% to 10%, we can get a range of probability values; two of such magnetization evolutions are shown in the left and right panels of fig. 4.14 (h). The correspondence between the reduction in anisotropy is shown in fig. 4.14 (e), wherein by varying the reduction of the magnetic anisotropy in the left site, the programmed probability can be changed from about 0.1 to about 0.9.

Unlike the p-bit-based system, the method to change the magnetic anisotropy can be non-volatile. This change could be employed by the voltage-controlled magnetic anisotropy effect in which a gate applies an electric voltage to change the magnetic anisotropy of the material [246–248]. The non-volatility of this system allows us to program the device only once, and it would preserve that state with no additional energy cost. However, DWs suffer from other issues related to inhomogeneities in the medium, such as grain distributions or edge roughness, which are pretty pronounced under such nanoscopic dimensions. More investigations need to be done to verify the suitability of such devices for usage as stochastic synapses.

4.7.2 Device-based inference simulations

We have discussed the essential features a device must possess to use as a memory device for our synapses effectively. Any physical system containing such properties can be potentially used as the synapse in our Bayes BiNN. However, any real system would have non-ideal behav-

iors, as discussed earlier, and we need to verify the robustness of our implementation to such imperfections. To do so, we consider a very simple model for programming our devices and look at the impact of different non-ideal behaviors.

In practice, when performing inference with such novel memory devices, we first train the network in computers and get the set of optimum values for all the probabilities, p_{tr}s. Then, the goal is to program these values to the devices exactly. Since we are dealing with a real physical system, we need to apply some bias to it to write that particular probability state p_{pr}. This bias can be voltage, current, or magnetic field, depending on the application, and here, we denote it by B. And once we convert the p_{tr} to B, this biasing quantity is applied to the devices, and a probability p_{pr} is programmed. In an ideal scenario, we would have $p_{pr} = p_{tr}$, but it would not be the case due to the physical constraints. The effect of the bias B on the programmed probability p_{pr} would depend on the details of the physical system being used. We show this process schematically in fig. 4.15 (a).

Here, the assumption about B is that it is only possible to generate values between -1 and 1. Hence, we use a simple affine transform of $B = 2p_{tr} - 1$, and then we use the $p_{pr} = device(B, params)$ to represent the conversion from the biasing quantity to the programmed probability where the $params$ represents the parameters of the model that are related to the specific physics of the material. For demonstration, we have chosen a linear model as the $device$ function between the bias and the probability p_{pr} with an additional noise term as

$$p_{pr} = 0.5(1 + \alpha B) + \eta N(0,1). \tag{4.14}$$

In this model, the slope is controlled by the parameter α and the noise by η, which is the standard deviation of a Gaussian distribution with zero mean ($N(0,\eta)$. The model is shown in fig. 4.15 (b). The two parameters of this simple model represent two different aspects potentially present in a real system. The parameter α denotes how strongly the biasing variable B changes p_{pr} where a low value means that the probability can be only programmed to a small range of values around 0.5. As shown in the figure, the green plot for $\alpha = 0.5$ attains neither 0 nor 1 probability value, whereas if $\alpha \geq 1$, then the full range of values is programmable. Hence, this parameter α signifies the range of values of p_{pr} to which this particular system can be programmed. On the other hand, the parameter η, which is the spread of the Gaussian noise, denotes the accuracy in programming. The probability intended to be programmed p_{tr} is a real number between 0 and 1, but the bias B is a physical quantity that we apply with a finite resolution. Consequently, the probability we can program with this is also restricted and will have a value close to what is intended owing to this granularity. Also, the programming process can be noisy for other reasons, and both phenomena are captured by the noise parameter η.

To study the impact of this *realistic* device model, we look back at our two moons task. In fig. 4.15 (c), we present the inference color maps where the probabilities have been programmed according to the method mentioned earlier. We increase the noise parameter η horizontally rightwards and the slope parameter α vertically downwards. These simulations are

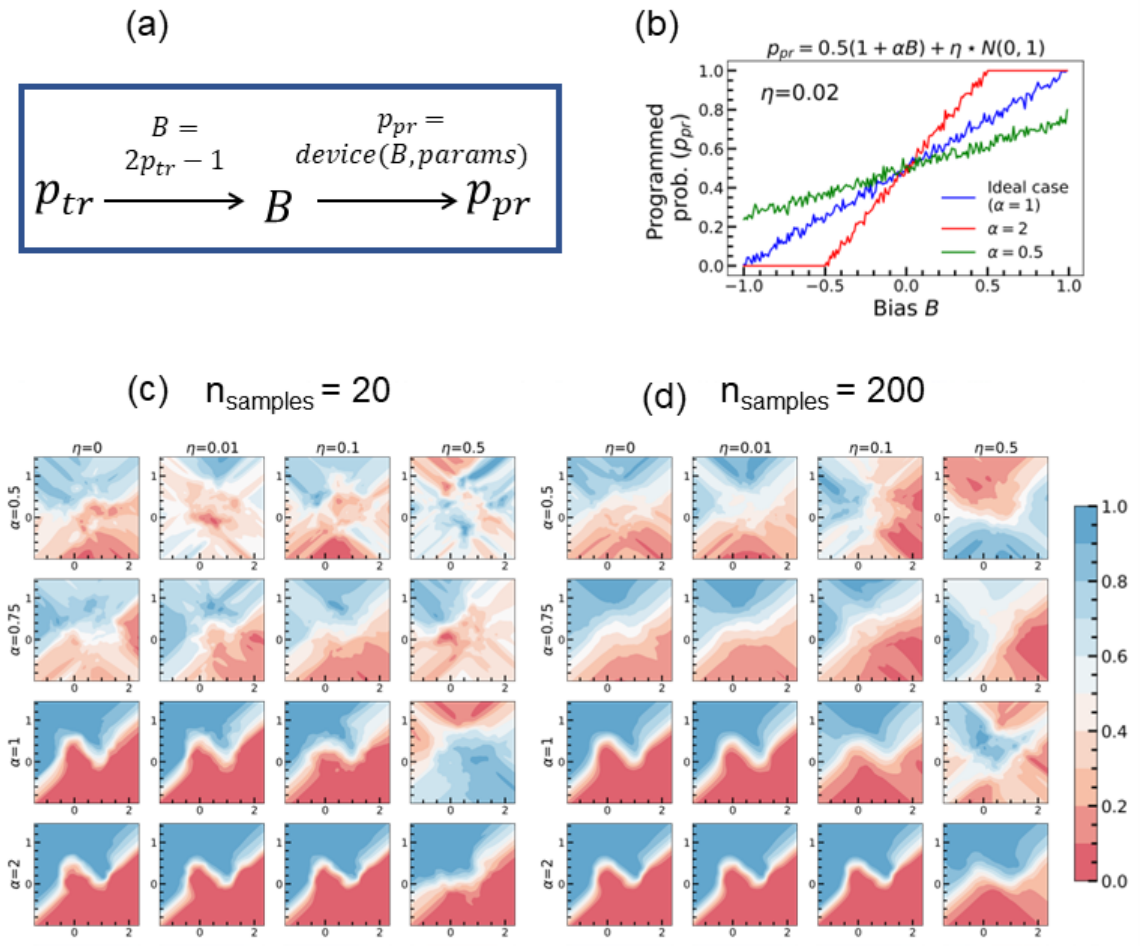


Figure 4.15: Device-based inference. (a) We use the schematic to convert the intended probabilities we get after training p_{tr} to the programmed probability p_{pr} via the generalized biasing variable B. (b) The linear model for an arbitrary device that maps the bias B to the programmed probability p_{pr}. (c) The inference color maps for the two moons task using the linear model with different slope parameter values α and the noise parameter η. The output shown is the average over 20 MC samples ($n_{samples}$). (d) The same plot as (c) but with $n_{samples}$ = 200 .

done taking 20 Monte Carlo samples ($n_{samples}$), and we observe that the inference is very poor for $\alpha = 0.5$. It progressively improves with the increase of slope, and for $\alpha = 2$, even with the highest magnitude of noise $\eta = 0.5$, the decision boundary in the inference somewhat resembles the ideal case. However, an increase in the value of α signifies that the programmed probability sharply transitions from 0 to 1, representing a system that has reduced stochasticity. Under such circumstances, the resolution to which we can tune the bias B would play an important role as that would dictate how precisely we can program the intermediate probability values.

If we focus on the row with $\alpha = 1$, we can conclude that the Bayes BiNN system is robust to the noise and up to a certain level. From this analysis, we can conclude that there is a region

in the $\alpha - \eta$ space where the Bayes BiNN inference is considerably resilient to device imperfections. Suppose we increase the number of MC samples over which we average the softmax outputs to 200. In that case, we see that now for lower values of α like 0.75, the inference color map looks more reasonable, although the probabilities we get are never less than 0.1 or more than 0.9. Thus, taking more samples can be another way to improve the inference quality when dealing with such imperfect devices.

4.8 Conclusion

This chapter highlights a new paradigm of performing deep learning with probabilities in the framework of Bayesian computing. We started the chapter by reviewing Bayesian probability and the different deep-learning algorithms with probabilistic flavors. Next, we discussed a couple of examples where memristive devices are used for computing with probabilities showcasing the suitability of such materials for probabilistic applications. After that, we introduced the Bayesian Binary Neural Networks and the quantification of uncertainties in Bayesian neural networks. Then we applied this to a toy example called the two moons dataset to illustrate some of its advantages. Then, we explore this algorithm in a more scaled-up version for a more realistic medical task, the MIT-BIH arrhythmia classification dataset. In the last part of this chapter, we discussed some potential Spintronics-based physical systems that have the potential to implement the Bayes BiNNs. We conclude this chapter by exploring a simple device model with imperfections used to perform inference in the two moons dataset.

The potential of Bayesian neural networks is quite evident from our results, especially in regimes of small data; the Bayes BiNN outperforms its deterministic analog. The affinity of such conventional neural networks to overfitting is mitigated naturally when we use probabilities instead of a fixed weight. This is highly beneficial in situations where the data is scarce, for example, in medical datasets for extremely rare diseases.

Another important domain in which our network surpasses the deterministic network is quantifying uncertainty in the prediction. In safety-critical applications, uncertainty evaluation can be as crucial as prediction. The probabilistic nature of Bayes BiNN and other Bayesian neural nets allows the quantification of the uncertainty in an organic manner. Especially the decomposition of the aleatoric and epistemic uncertainties can give us insight even into the source of uncertainty, which can be invaluable in such applications.

To conclude, in this chapter, we have explored the potential of Bayesian Binary Neural Networks for the quantification of uncertainty and proposed some real physical systems that can potentially implement such networks.

Conclusions and future work

> "Exploration is in our nature. We began as wanderers, and we are wanderers still."
>
> Carl SAGAN

Summary

As mentioned at the outset of this thesis, our civilization finds itself at a pivotal moment: the long-awaited promises of artificial intelligence are being rapidly realized, with new developments emerging almost monthly. However, we are also entering an era in which the impact of climate change is increasingly evident in our daily lives, and the ever-increasing need for computational power for deep learning-based applications is worsening the situation. Neuromorphic computing offers a power-efficient solution to this by changing the paradigm of computation at an architectural level to mimic the human brain [82]. Utilizing emerging memory technologies for in or near-memory computing seems to be a promising lead. This novel class of memory possesses features like power efficiency, non-volatility, and speed that are attractive for the hardware realization of deep learning algorithms [90].

However, as discussed in chapter 1, the requirements for implementing neural networks in hardware do not always align with emerging memory technologies' characteristics. Specifically, the imperfections, constraints derived from circuits, and dependence on stable power supply make them unsuitable for edge applications where the low-power, low-latency aspects could genuinely shine. In this thesis, we concretely investigated the interplay between imperfections and the performance of neural networks from three different perspectives.

- **Learning with imperfect memory**. The physics of resistance-based novel memory is based on the atomic-level phenomenon and is consequently prone to different types of noises and non-idealities. On the contrary, the training in conventional neural networks is based on precise values that need to be changed accurately. In chapter 2, we explored a quantized form of the neural network, the binarized neural network, and studied its robustness to imperfection.

- **Inference with circuit-based constraints**. Different limitations crop up in an integrated circuit where these resistive memories are integrated with other electronic circuits to perform the computations necessary for inference. In chapter 3, such limitations and their impact on neural network performance were investigated.

- **Probabilistic neural networks with stochasticity of novel memory**. The stochastic nature of spintronics devices at the nanoscale is usually a nuisance for conventional applications. In chapter 4, we propose ideas to harness this stochastic behavior for probabilistic computation using a specialized algorithm called the Bayesian Binary Neural Networks.

In chapter 2, we focussed on implementing learning using binarized neural networks in the weak RESET regime of HfO_x-based filamentary resistive RAM. A physics-based model for resistance variation is developed for this material in the weak RESET regime, which includes different types of variabilities. The model is fitted to and compared against experimental characterizations. This model is then incorporated within the PyTorch deep learning framework,

and training of binarized neural network is simulated for the MNIST handwritten digit recognition and CIFAR-10 object detection tasks. The main conclusion of this chapter is that learning is indeed possible in this regime using binarized neural networks where the quantization provides robustness to the different variabilities present in filamentary ReRAMs. For this work, a methodology is also developed to test the contribution of various imperfections to performance degradation in neural networks. This methodology can shed light on the specific imperfection that affects learning the most. The devices could be optimized for that imperfection at the fabrication level [4].

This chapter focussed solely on the variabilities of the memory devices, assuming all other computations are performed perfectly. However, that is not the case: the memory devices are co-located within integrated circuits. These other electronic circuits are used to read the weights, compute the MAC and perform the thresholding operation for only the inference process. During inference, it is required to program the binarized weights, which are learned from the training (done separately on software). But due to the variability present in the ReRAMs, this does not occur ideally. This, along with circuit-related reasons, introduce errors in the computations of our neural network. The impact of such errors on the inference performance of neural networks is presented in chapter 3. By inference simulations on the MNIST and CIFAR-10 tasks, we show that binarized neural networks are robust to such errors in realistic programming conditions. Further, we looked at array-size-related limitations in the realization of neural networks and proposed a strategy that can circumvent this with a slight degradation in performance. In the context of edge applications, we investigated the impact of errors introduced by a non-optimal power supply derived from a solar cell. We showed the resilience of our network to these errors. This work highlights that even under low power, corresponding to low illumination, the circuit can deliver approximately correct computations that do not significantly diminish performance.

Finally, in chapter 4, we focus on harnessing an aspect of imperfections, the intrinsic stochasticity for a particular type of computation; the Bayesian Neural Network. We propose the implementation of Bayesian binary neural networks using nanometric stochastic spintronics devices. This class of algorithm is based on the idea of learning probability distributions instead of point estimates learned in conventional deterministic networks. The main advantage of Bayesian neural networks is their innate ability to prevent overfitting data, a common problem for deterministic networks. This issue is aggravated for small datasets, and the impact can be detrimental with mistakes in data labeling. Another remarkable feature of such neural networks is the ability to quantify uncertainty in the prediction, which is essential for safety-critical applications where the prediction uncertainties are as significant as the predictions. We explored both aspects in this chapter using a toy dataset called the two moons dataset and a medical task, the MIT-BIH dataset, for arrhythmia classification.

To conclude, the overarching theme of this thesis is to utilize the best of both worlds: combining the energy-efficient, bio-inspired emerging memories with powerful deep learning algo-

rithms to achieve efficient, environmentally friendly computation adept at pattern recognition.

Perspectives

The three projects presented here revolve around the concept of quantized neural networks and their realization using imperfect memory devices. In the second chapter, we investigated learning, where we simulated the device resistance evolution and its impact on the neural network performance. This presents progress toward implementing on-chip learning; however, other issues exist. If we revisit the learning process in a neural network, the input is fed forward through the network to produce an output that is then compared to the expected target output. The loss function measures the 'distance' between the prediction and expected output. To calculate the update of the model weights, the derivative of the loss function with respect to the weights needs to be calculated. Implementing all of this computation on-chip is challenging, mainly because vanilla backpropagation requires the gradients in earlier layers to depend on the gradients and weights of later layers.

This makes the circuit-based realization difficult since all the values must be stored. There are two possible directions for solving this issue: one is to design efficient circuits that can perform backpropagation without significant memory overhead. However, there is another more bio-realistic alternative: the equilibrium propagation algorithm. Unlike backpropagation, here, the update rule is local; the weight change depends only on the activation of its connecting neurons. The device model can be used for simulating equilibrium propagation similarly to the backpropagation algorithm [72, 249]. Also, the more recently proposed forward-forward algorithm holds promise in this regard [74]. The Forward-Forward algorithm is another alternative to backpropagation, where instead of using forward and backward passes, it utilizes two forward passes. One forward pass involves real or positive data, while the other involves negative data that does not look like real data. There is an objective function for each layer, which is supposed to have a high value for the positive data and a low value for the negative data. In this case, too, the learning could be simulated with the device model proposed in the chapter in the same manner. The methodology presented in this chapter could be used to simulate different learning algorithms, which would take us one step closer to on-chip learning.

The two studies presented in chapter 3 established that the inference on the binarized neural networks could be accomplished on-chip. A natural extension of this would be to implement the same for ternarized neural networks [250]. The potential for implementing such networks with ReRAM has already been demonstrated, and just due to the less quantization, it is more powerful than its binarized counterpart [251]. However, the impact of errors on inference accuracy needs to be explored in more detail following the same methodology adopted in this thesis. Another aspect that needs to be mentioned at this point is related to the architecture of neural networks. The design of the arrays is done in a manner that makes implementing fully connected networks natural. For the simulation of convolutional neural networks, we have

assumed that the same circuits are used, and as a result, we have used the same errors. But, in reality, the convolution operation and max pooling are practically different from the simple MAC operations in fully connected networks and require dedicated circuitry. This should be explored in more detail, and the associated errors should be considered.

For the Bayesian project, the development is still mainly at the algorithmic level. We explored some scenarios where quantifying the uncertainty and its decomposition to the aleatoric and epistemic components gives us more insight into the data. The device-based, and then in the long term, the circuit-based implementation and its related issues need to be addressed. To successfully implement probabilistic computing, it is crucial to take Monte Carlo samples from the model distribution and compute their average. This has to be accomplished by circuits in a manner that is energy efficient and fast to impact applications significantly.

Chapter 1 has highlighted that technological advancement does not occur in a vacuum, especially in a multidisciplinary field like neuromorphic computing. Progress should be made simultaneously in terms of physical devices, deep learning algorithms, and our understanding of the neuroscience of the human brain to achieve significant breakthroughs. The existing device technologies should be optimized to be more compatible with the requirements of neural networks. Specifically, the variabilities, asymmetry, and nonlinearity aspects should be reduced as much as possible. At the same time, new types of physics of materials need to be explored for computation. The reservoir computing algorithm is a promising lead in this regard, where the physical dynamics of complex systems are utilized for pattern recognition [252, 253]. Photonics-based neuromorphic computing is another different paradigm where instead of current, light is used to propagate information and perform computation, and optical RAMs are used as memory [254, 255]. In the future, circuits integrating both electronic and photonic elements could potentially be researched, allowing more flexibility in performance.

On the algorithmic front as well the direction of research should not be restricted to producing higher performance, and equal emphasis should be given to algorithms that are biorealistic, hardware-friendly, and thus energy efficient. Algorithms like equilibrium propagation, forward-forward, and direct feedback alignment hold high potential in this regard [73]. Currently, extensive research is being conducted on self-supervised computing, and the hardware implementation of this type of learning must be realized soon [62]. Transformers, the backbone of most large language models today, rely on the attention mechanism, for which the memory complexity depends quadratically on the maximum input sequence length [51, 256]. This makes it highly memory-hungry, and algorithmic solutions are being researched to solve this.

In the more biological domain of neuromorphic computing, there has been substantial progress in implementing spiking neural networks, where the sparsity of representation naturally leads to energy efficiency. However, such algorithms are notoriously hard to train, and the performance is usually not at par with the backpropagation-based conventional neural networks [87, 257]. The development of such algorithms will be crucial in the future, especially as

we learn more about the biology of the brain.

The human brain originally inspired deep learning, and recent technological developments have allowed for large-scale computations, enabling deep learning to reach its full potential. It will not be surprising that progress in all three domains - deep learning, memory technology, and neuroscience would pave the way for the realization of energy-efficient, neuromorphic computers, which would play a significant role in improving our lives in the future.

Synthèse en Français

Introduction et contexte

Actuellement nous nous trouvons à une transition critique où nous subissons les conséquences du changement climatique, dont l'une des causes principales est les émissions causées lors de la génération d'électricité à partir de combustibles fossiles [1]. Par ailleurs, nous progressons rapidement dans le domaine de l'intelligence artificielle (IA), avec de grands modèles de langage montrant dès à présent des ressemblances avec une intelligence artificielle générale [2]. Les progrès récents sont un pas de plus vers ce que promet l'IA, comme par exemple la découverte de médicaments pour des maladies graves, la conception des voitures autonomes ou d'autres innovations révolutionnaires.

Cependant, cela a un coût : pour développer, entraîner et utiliser les modèles d'apprentissage profond de pointe, les ordinateurs existants consomment beaucoup d'énergie. Ce type de calcul, généralement réalisé dans des centres de données avec de nombreuses unités de traitement graphique et autres accélérateurs dédiés, n'est généralement pas optimisé en termes de consommation d'énergie. L'entraînement d'un seul modèle consomme plus d'énergie que celle consommée par 100 foyers aux États-Unis en un an [3]. Ce nombre ne ferait qu'augmenter continuellement avec la taille des modèles et la complexité des calculs. L'empreinte carbone résultante serait énorme, et ce développement n'est pas viable d'un point de vue environnemental.

Si l'on considère le processus de calcul au niveau architectural, le goulot d'étranglement en termes de consommation d'énergie est lié à la transmission des données entre la mémoire et les unités logiques. Pendant le processus d'entraînement d'un réseau de neurones, trois opérations principales sont effectuées : la transmission des données vers et depuis la mémoire, ainsi que l'exécution des opérations de multiplication et d'addition dans l'unité de traitement. Parmi ces opérations, la transmission d'information est la plus coûteuse en termes d'énergie. Cette architecture informatique, appelée architecture de von Neumann, diffère fondamentalement d'un autre système capable de reconnaître des motifs : le cerveau humain. Il peut effectuer des tâches de vision, de traitement du langage naturel, de déduction logique et de planification avec une consommation énergétique des ordres de grandeur inférieurs à ce qui est généralement consommé par un modèle d'apprentissage profond moderne. Le cerveau

calcule différemment : la connectivité est massive avec une redondance importante, les règles d'apprentissage sont locales, l'information est propagée par des décharges électrochimiques induites par des variations de tension, et les éléments logiques et de mémoire sont situés respectivement sous forme de neurones et de synapses. Le domaine du calcul neuromorphique vise à émuler ou imiter le cerveau pour réaliser des calculs plus efficaces. Cette thèse traite du calcul neuromorphique avec une inspiration architecturale et tente d'imiter la biologie du point de vue du calcul en mémoire ou du calcul près de la mémoire.

En particulier, j'étudie les technologies de mémoire émergentes basées sur la résistance pour les réseaux de neurones, car elles offrent un substrat plus économe en énergie, compatible avec la technologie CMOS, et non volatil pour effectuer des calculs près de la mémoire par rapport à leurs homologues plus conventionnels (SRAM ou DRAM). Les mémoires non volatiles à faible consommation d'énergie conviennent bien aux applications embarquées où l'efficacité énergétique est priorisée. De plus, leur nature non volatile est particulièrement avantageuse pour les équipements qui ne sont pas constamment utilisés, car aucune alimentation n'est nécessaire pour stocker un état. Malgré ces avantages, les mémoires émergentes telles que les mémoires résistives à base d'oxyde, les mémoires à changement de phase et les mémoires à accès aléatoire magnétique souffrent d'imperfections qui peuvent affecter considérablement les performances des réseaux de neurones. Dans cette thèse, j'étudie et présente des réseaux de neurones compatibles avec le matériel qui sont relativement tolérants à ces imperfections et peuvent les intégrer pour effectuer des calculs. J'essaie de répondre aux questions suivantes dans les chapitres deux, trois et quatre :

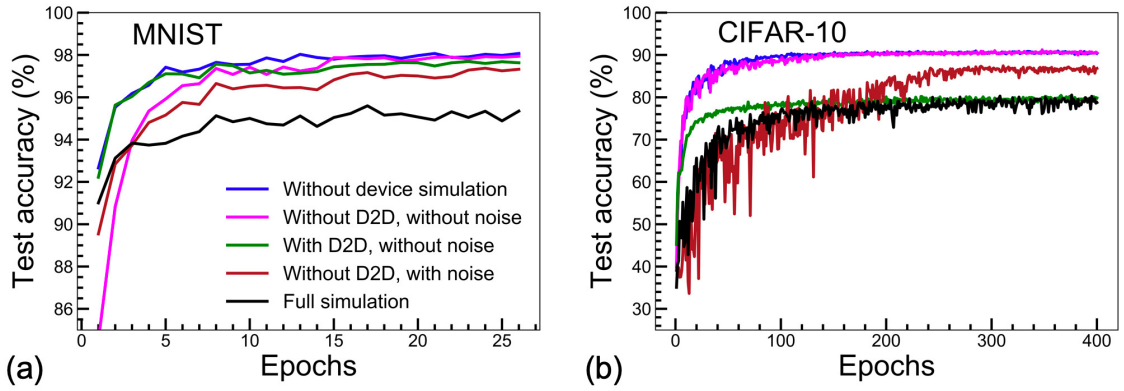
- **Chapitre 2** Comment pouvons-nous apprendre avec une mémoire résistive à base d'oxyde imparfaite et filamentaire ?
- **Chapitre 3** Quel est l'impact des erreurs et des contraintes survenant au niveau du circuit sur l'inférence des réseaux neuronaux ?
- **Chapitre 4** Pouvons-nous exploiter la nature stochastique des dispositifs stochastiques pour effectuer un calcul probabiliste ? Et quels avantages supplémentaires cela pourrait-il avoir ?

Résultats

Chapitre 2 Dans le chapitre, nous avons concentré nos efforts sur la mise en œuvre de l'apprentissage à l'aide de réseaux de neurones binarisés dans le régime de RESET faible de mémoires résistives à base d'oxyde d'hafnium (HfO_x). Un modèle basé sur la physique est développé pour décrire la résistance de ce type de dispositifs dans le régime de RESET faible, incluant différents types de variabilités. Le modèle est ajusté et comparé à des caractérisations expérimentales. Ce modèle est ensuite incorporé dans le framework d'apprentissage profond PyTorch, et l'apprentissage d'un réseau de neurones binaire est simulé pour les tâches de reconnaissance des chiffres

manuscrits MNIST et de détection d'objets CIFAR-10. La principale conclusion de ce chapitre est que l'apprentissage est en effet possible dans ce régime en utilisant des réseaux de neurones binarisés, où la quantification offre une robustesse aux différentes variabilités présentes dans les mémoires ReRAM à filament. Pour ce travail, une méthodologie est également développée pour tester la contribution des diverses imperfections à la dégradation des performances des réseaux neuronaux. Cette méthodologie peut mettre en lumière l'imperfection spécifique qui affecte le plus l'apprentissage. Les dispositifs peuvent être optimisés pour cette imperfection au niveau de la fabrication.

La figure suivante montre l'impact du bruit et de la variabilité d'un appareil à l'autre sur les performances des réseaux de neurones binarisés pour les tâches (a) MNIST et (b) CIFAR-10. Les graphiques montrent la précision des tests pendant l'entraînement pour cinq cas différents : sans simulation de dispositif (bleu), sans variation dispositif à dispositif (D2D) ni bruit (rose), avec variation D2D mais sans bruit (vert), sans D2D mais avec simulation complète du bruit et du modèle moyen (marron), et simulation complète incorporant à la fois la variabilité D2D, le modèle moyen et le bruit (noir).

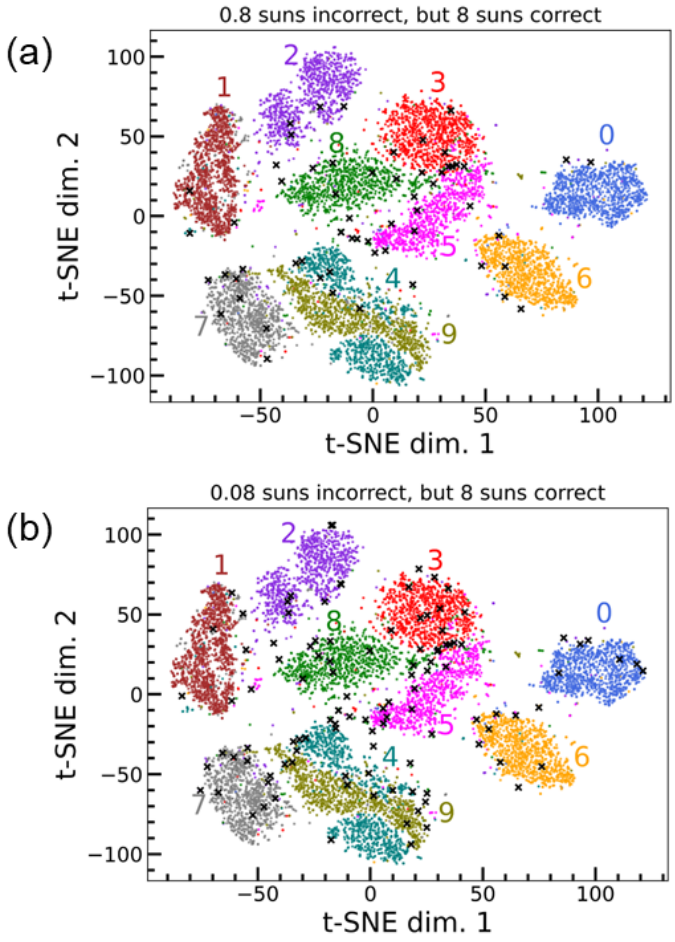


Chapitre 3

Le dernier chapitre s'est concentré uniquement sur les variabilités des dispositifs de mémoire, en supposant que toutes les autres opérations sont effectuées de manière parfaite. Cependant, ce n'est pas le cas : les dispositifs de mémoire sont co-localisés dans des circuits intégrés. Ces autres circuits électroniques sont utilisés pour lire les poids, effectuer le calcul Multiplication et accumulation (MAC) et effectuer l'opération de seuillage uniquement dans le processus d'inférence. Pendant l'inférence, il est nécessaire de programmer les poids binarisés, qui sont appris à partir de l'entraînement (réalisé séparément sur un logiciel). Cependant, en raison de la variabilité présente dans les ReRAMs, cela ne se produit pas de manière idéale. Ceci, ainsi que les raisons liées au circuit, introduit des erreurs dans les calculs de notre réseau neuronal. L'impact de ces erreurs sur les performances d'inférence des réseaux neuronaux est présenté dans ce chapitre. En effectuant des simulations d'inférence sur les tâches MNIST et CIFAR-10, nous montrons que les réseaux neuronaux binarisés sont robustes à de telles erreurs dans des conditions réalistes de programmation.

De plus, nous avons examiné les limitations liées à la taille des matrices dans la réalisation des réseaux neuronaux et avons proposé une stratégie qui peut contourner cela avec une légère dégradation des performances. Dans le contexte des applications embarquées, nous avons étudié l'impact des erreurs introduites par une alimentation non optimale provenant d'une cellule solaire. Nous avons montré la résilience de notre réseau à ces erreurs. Ce travail met en évidence que même à faible puissance, correspondant à une faible illumination, le circuit peut fournir des calculs approximativement corrects sans diminuer significativement les performances.

La figure suivante montre la représentation de l'intégration stochastique des voisins distribuée en t (t-SNE) de l'ensemble de données de test MNIST. Les points de données noirs sont mal classés sous un éclairage de 0,8 soleil (a) et de 0,08 soleil (b), mais ils sont correctement classés sous un éclairage de 8 soleil, à l'aide d'un réseau neuronal binarisé entièrement connecté.

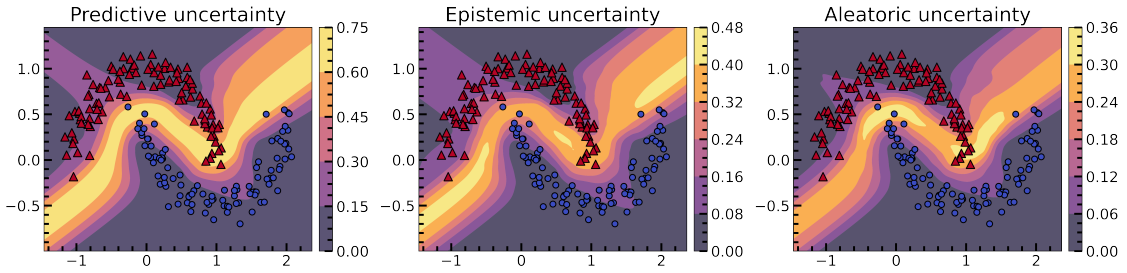


Chapitre 4

Enfin, dans ce chapitre, nous nous concentrons sur l'exploitation d'un aspect des imperfections, la stochasticité intrinsèque, pour un type particulier de calcul : le réseau de neurones

bayésien. Nous proposons la mise en œuvre de réseaux de neurones binaires bayésiens en utilisant des dispositifs de spintronique stochastiques nanométriques. Ce type d'algorithmes est basé sur l'apprentissage de distributions de probabilité plutôt que l'estimation ponctuelle apprise dans les réseaux déterministes conventionnels. L'avantage principal des réseaux de neurones bayésiens est leur capacité innée à éviter le surajustement des données, un problème courant pour les réseaux déterministes. Ce problème est aggravé lors del'utilisation de petits ensembles de données, et l'impact peut être conséquent en cas d'erreurs d'étiquetage des données. Une autre caractéristique remarquable de ces réseaux de neurones est leur capacité à quantifier l'incertitude de la prédiction, ce qui est essentiel pour les applications critiques en matière de sécurité où les incertitudes de prédiction sont aussi importantes que les prédictions elles-mêmes. Nous avons exploré ces deux aspects dans ce chapitre en utilisant un ensemble de données d'exemple appelé le jeu de données des deux lunes et une tâche médicale, le jeu de données MIT-BIH, pour la classification des arythmies.

Cette figure montre la quantification de l'incertitude dans la tâche des deux lunes. La carte en couleur montre les incertitudes prédictives, épistémiques et aléatoires respectivement avec les trois tracés. L'incertitude prédictive est décomposée en deux composantes ayant des sources différentes : épistémique pour les régions sans données d'entraînement et aléatoire pour les parties ambiguës. La quantification de l'incertitude est un avantage clé des réseaux neuronaux bayésiens par rapport aux réseaux déterministes.



Perspectives

Les projets présentés se concentrent sur les réseaux de neurones quantifiés mis en œuvre à l'aide de dispositifs de mémoire imparfaits. Les défis de l'apprentissage sur puce sont discutés, notamment la nécessité de calculer les gradients et les poids pour la rétropropagation, ce qui nécessite une capacité de stockage mémoire importante. Des approches alternatives telles que l'algorithme de propagation d'équilibre et l'algorithme forward-forward sont suggérées. Le potentiel de mise en œuvre de l'inférence sur puce pour les réseaux neuronaux ternarisés est mis en évidence, ainsi que la nécessité d'étudier l'impact des erreurs sur la précision de prédiction. L'importance de prendre en compte l'architecture des réseaux de neurones, d'optimiser les technologies des dispositifs et d'explorer de nouveaux types de matériaux pour le calcul est

soulignée. La recherche ne devrait pas seulement se concentrer sur la performance, mais aussi sur des algorithmes bio-réalistes, adaptés **au matériel** et économes en énergie. La computation auto-supervisée et la mise en œuvre matérielle de ces méthodes d'apprentissage devraient être explorées. Dans l'ensemble, les avancées multidisciplinaires concernant les dispositifs physiques, les algorithmes d'apprentissage profond et la compréhension de la neurosciences du cerveau humain sont cruciales pour obtenir des avancées significatives dans le domaine du calcul neuromorphique.

List of publications

Peer-Reviewed Journal Articles

✠ **Atreya Majumdar**, Marc Bocquet, Tifenn Hirtzlin, Axel Laborieux, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz, "Model of the Weak Reset Process in HfOx Resistive Memory for Deep Learning Frameworks" , *IEEE Transactions on Electron Devices*, vol. 68, No. 10, p. 4925-4932, 2021.

✠ Mona Ezzadeen, **Atreya Majumdar**, Olivier Valorge, Niccolò Castellani, Valentin Gherman, Guillaume Regis, Bastien Giraud, Jean-Philippe Noël, Valentina Meli, Marc Bocquet, François Andrieu, Damien Querlioz, and Jean-Michel Portal, "Implementation of Binarized Neural Networks Immune to Device Variation and IR Drop Employing Resistive RAM Bridges and Capacitive Neurons" , *Under revision at Communications Engineering.*

✠ Fadi Jebali, **Atreya Majumdar**, Clément Turck, Kamel-Eddine Harabi, Mathieu-Coumba Faye, Eloi Muhr, Jean-Pierre Walder, Oleksandr Bilousov, Amadéo Michaud, Elisa Vianello,Tifenn Hirtzlin, François Andrieu, Marc Bocquet, Stéphane Collin, Damien Querlioz, and Jean-Michel Portal, "Powering AI at the Edge: A Robust, Memristor-based Binarized Neural Network with Near-Memory Computing and Miniaturized Solar Cell" , *Under preparation.*

✠ Djohan Bonnet, Tifenn Hirtzlin, **Atreya Majumdar**, Thomas Dalgaty, Marc Bocquet, Eduardo Esmanhotto, Valentina Meli, Niccolò Castellani, Simon Martin, Jean-Francois Nodin, Guillaume Bourgeois, Jean-Michel Portal, Damien Querlioz, and Elisa Vianello, "Bringing uncertainty quantication to the extreme-edge with memristor-based Bayesian neural networks" , *Under revision at Nature Communications.*

Peer-Reviewed Conference Proceedings

✠ Fadi Jebali, **Atreya Majumdar**, Axel Laborieux, Tifenn Hirtzlin, Elisa Vianello, Jean-Pierre Walder, Marc Bocquet, Damien Querlioz, and Jean-Michel Portal, "CAPC: A Configurable Analog Pop-Count Circuit for Near-Memory Binary Neural Networks" , *2021*

IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), p. 158-161, 2021.

✠ MONA EZZADEEN, **ATREYA MAJUMDAR**, MARC BOCQUET, BASTIEN GIRAUD, JEAN-PHILIPPE NOËL, FRANÇOIS ANDRIEU, DAMIEN QUERLIOZ, and JEAN-MICHEL PORTAL, "Low-overhead implementation of binarized neural networks employing robust 2T2R resistive RAM bridges" , *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, p. 83-86, 2021.

✠ MONA EZZADEEN, **ATREYA MAJUMDAR**, SIGRID THOMAS, JEAN-PHILIPPE NOËL, BASTIEN GIRAUD, MARC BOCQUET, FRANÇOIS ANDRIEU, DAMIEN QUERLIOZ and JEAN-MICHEL PORTAL, "Binary ReRAM-based BNN first-layerimplementation" , *To be presented at DATE 2023*.

Conferences Without Proceedings

✠ **ATREYA MAJUMDAR**, MARC BOCQUET, TIFENN HIRTZLIN, AXEL LABORIEUX, JACQUES-OLIVIER KLEIN, ETIENNE NOWAK, ELISA VIANELLO, JEAN-MICHEL PORTAL, and DAMIEN QUERLIOZ, "Resistive RAM-based Implementation of Binarized Neural Networks: Inference and Training" , *CVPR 2021*, 2021. *Poster presentation.*

✠ **ATREYA MAJUMDAR**, MARC BOCQUET, TIFENN HIRTZLIN, AXEL LABORIEUX, JACQUES-OLIVIER KLEIN, ETIENNE NOWAK, ELISA VIANELLO, JEAN-MICHEL PORTAL, and DAMIEN QUERLIOZ, "Inference and training in Resistive RAM-based Implementation of Binarized Neural Networks" , *Neal 2022, Göttingen, Germany*, 2022. *Poster presentation.*

✠ **ATREYA MAJUMDAR**, LUIS LOPEZ-DIAZ, LIZA HERRERA DIEZ, and DAMIEN QUERLIOZ, "Spintronics-based Bayesian Binary Neural Networks for biomedical applications" , *MagnEFi 2022*, 2022. *Poster presentation.*

Training Bayes BiNN

Training of Bayes BiNN amounts to learning the values of the probability $\boldsymbol{p}$ for all the binary weights of the network. This problem is cast in the mold of a variational inference problem and solved by a variant of the BBB. In this case, the variational or surrogate distribution is the Bernoulli distribution, and the prior distribution $P(\mathbf{W})$ is defined as follows. A priori, it is assumed that the weights would be equally likely to be in either of the two states +1 and -1. Hence, the prior $P(\mathbf{W})$ is a symmetric Bernoulli distribution where each state has a probability of $\frac{1}{2}$. The approximate posterior distribution, $q_{\boldsymbol{p}}(\mathbf{W})$ or simply $q(\mathbf{W})$ is given by the mean-field symmetric Bernoulli distribution

$$q(\mathbf{W}) = \prod_{j=1}^{W} p_j^{\frac{1+W_j}{2}} (1-p_j)^{\frac{1-W_j}{2}}. \tag{15}$$

Here, W is the total number of synapses in our network. The product is due to the fact that all the weights are independent of each other and the exponents because the Bernoulli random variable takes values +1 and -1 (instead of +1 and 0 for the general Bernoulli distribution). The goal of the training algorithm is to learn the values of these probability parameters, p_js.

For the sake of calculations, equ. 15 needs to be recast in the form of the more general minimal exponential family distribution [35]. In that form, we have two parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, called the natural and expectation parameters, and defined in terms of the p_js as

$$\lambda_j := \frac{1}{2} \log \frac{p_j}{1-p_j}, \quad \mu_j := 2p_j - 1. \tag{16}$$

The natural parameter for the prior $P(\mathbf{W})$, written as $\boldsymbol{\lambda}_0$, is thus $\mathbf{0}$. Now, we need to define our optimization criteria, that is, the value that we are trying to minimize. In this case, it is the Bayesian formulation of the loss-based approach where we combine two terms, the first of which is the expectation of the loss function over our surrogate posterior $q(\mathbf{W})$, and the latter is the KL divergence between the posterior and the prior [258, 259]. Mathematically, the optimization objective is equal to

$$\mathbb{E}_{q(\mathbf{W})} \left[\sum_{i=1}^{N} l(y_i, f^{\mathbf{W}}(\mathbf{x}_i)) \right] + \mathrm{KL}[q(\mathbf{W}) || P(\mathbf{W})]. \tag{17}$$

The $\mathbf{x}_i$, y_i, $f^{\mathbf{W}}$, N and l denotes the input, the output, the model, the size of the mini-

batch, and the loss function respectively. This is equivalent to the objective function defined in equ. 4.7 when the loss function takes the form of the cross-entropy loss.

A recurring theme of difficulty in training binary neural networks stems from the fact that the learning is a discrete optimization problem for which standard optimizers like Adam do not perform very well. As a solution to this, in the original publication, the straight-through-estimator(STE) was used, which served as a proxy to the true gradient [165, 173, 260]. More recently, another approach was proposed based on the *inertia* of the model parameters called the Binary Optimizer (Bop) [261]. In this Bayesian version as well, we are faced with a similar challenge; to backpropagate through a stochastic node, which can take only discrete value. We saw in the first section how this problem is solved for a variational autoencoder by using the Reparametrization trick, but it only works for a continuous distribution (specifically, Gaussian distribution for VAEs). Instead, the Gumbel-softmax trick is used, where a *concrete distribution* is used to *relax* the discrete variables into a continuous form [262, 263]. If there is a binary variable $W_j \in \{-1,+1\}$, with $P(W_j = +1) = p_j = \frac{\tanh(\lambda_j)+1}{2}$, the corresponding relaxed variable $W_b^{\epsilon_j,\tau} \in (-1,1)$ can be defined as

$$W_b^{\epsilon_j,\tau} := \tanh((\lambda_j + \delta_j)/\tau). \tag{18}$$

In this expression, $\tau > 0$ is the temperature parameter that defines how sharply the transition from -1 to 1 occurs, and in the limit of $\tau \to 0$, it behaves exactly like the sign function used in the STE method of non-Bayesian BiNN. The δ_j values come from the Gumbel distribution as

$$\delta_j := \frac{1}{2}\log\frac{\epsilon_j}{1-\epsilon_j}. \tag{19}$$

Here, the $\epsilon_j \sim U(0,1)$ is sampled from a uniform distribution [264]. With this setup, using the backpropagation algorithm, the learning rule or update equation for the natural parameters $\boldsymbol{\lambda}$ becomes

$$\boldsymbol{\lambda} \leftarrow (1-\alpha)\boldsymbol{\lambda} - \alpha[\mathbf{s}\odot\mathbf{g} - \boldsymbol{\lambda}_0]. \tag{20}$$

In equ. 20, α is the learning rate, the symbol $\odot$ signifies the element-wise product and $\mathbf{s}$, $\mathbf{g}$ are defined as

$$\mathbf{s} := \frac{N(1-(\mathbf{W}_b^{\epsilon_j,\tau}(\boldsymbol{\lambda}))^2)}{\tau(1-\tanh(\boldsymbol{\lambda})^2)}, \tag{21}$$

$$\mathbf{g} := \frac{1}{N}\sum_{i\in N}\nabla_{\mathbf{W}_r} l(y_i, f^{\mathbf{W}}(\mathbf{x}_i))\Big|_{\mathbf{W}_r=\mathbf{W}_b^{\epsilon_j,\tau}}. \tag{22}$$

Equ. 22 is the minibatch gradient of the loss function term with respect to the relaxed variable.

Bibliography

[1] Lea Berrang-Ford, James D Ford, and Jaclyn Paterson. Are we adapting to climate change? *Global environmental change*, 21(1):25–33, 2011.

[2] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[3] Josh Saul and Dina Bass. Artificial intelligence is booming—so is its carbon footprint, 2023.

[4] Atreya Majumdar, Marc Bocquet, Tifenn Hirtzlin, Axel Laborieux, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Model of the weak reset process in hfo x resistive memory for deep learning frameworks. *IEEE Transactions on Electron Devices*, 68(10):4925–4932, 2021.

[5] Takashi Kojima. *Japanese Abacus Use & Theory*. Tuttle Publishing, 2012.

[6] Teun Koetsier. On the prehistory of programmable machines: musical automata, looms, calculators. *Mechanism and Machine theory*, 36(5):589–603, 2001.

[7] Ylva Fernaeus, Martin Jonsson, and Jakob Tholander. Revisiting the jacquard loom: threads of history and current patterns in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1593–1602, 2012.

[8] Allan G Bromley. Charles babbage's analytical engine, 1838. *Annals of the History of Computing*, 4(3):196–217, 1982.

[9] Scott McCartney. Eniac: The triumphs and tragedies of the world's first computer. 1999.

[10] PA Redhead. The birth of electronics: Thermionic emission and vacuum. *Journal of Vacuum Science & Technology A: Vacuum, Surfaces, and Films*, 16(3):1394–1401, 1998.

[11] Michael R Williams. The origins, uses, and fate of the edvac. *IEEE Annals of the History of Computing*, 15(1):22–38, 1993.

[12] John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

[13] David L Arenberg. Ultrasonic solid delay lines. *The Journal of the Acoustical Society of America*, 20(1):1–26, 1948.

[14] William N Papian. The mit magnetic-core memory. In *Papers and discussions presented at the Dec. 8-10, 1953, eastern joint AIEE-IRE computer conference: information processing systems—reliability and requirements*, pages 37–42, 1953.

[15] Charles J Bashe, Lyle R Johnson, John H Palmer, and Emerson W Pugh. *IBM's early computers*. MIT press, 1986.

[16] Nallur S Prasad. *IBM mainframes: Architecture and design*. McGraw-Hill, Inc., 1989.

[17] Lars Heide. *Punched-card systems and the early information explosion, 1880–1945*. JHU Press, 2009.

[18] Shimeng Yu. *Semiconductor Memory Devices and Circuits*. CRC Press, 2022.

[19] Dean Klein. The history of semiconductor memory: From magnetic tape to nand flash memory. *IEEE Solid-State Circuits Magazine*, 8(2):16–22, 2016.

[20] EA Newman, DO Clayden, and MA Wright. The mercury-delay-line storage system of the ace pilot model electronic computer. *Proceedings of the IEE-Part II: Power Engineering*, 100(76):445–452, 1953.

[21] J Presper Eckert Jr, James R Weiner, H Frazer Welsh, and Herbert F Mitchell. The univac system. In *Papers and discussions presented at the Dec. 10-12, 1951, joint AIEE-IRE computer conference: Review of electronic digital computers*, pages 6–16, 1951.

[22] Harry D Huskey. Williams tube memory. In *Encyclopedia of Computer Science*, pages 1851–1853. 2003.

[23] Kiyoo Itoh. The history of dram circuit designs–at the forefront of dram development–. *IEEE Solid-State Circuits Society Newsletter*, 13(1):27–31, 2008.

[24] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[25] Ethan Mollick. Establishing moore's law. *IEEE Annals of the History of Computing*, 28(3):62–75, 2006.

[26] Mansfield Merriman. On the history of the method of least squares. *The Analyst*, 4(2):33–36, 1877.

[27] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251(254):10, 2012.

[28] Stan Franklin. History, motivations, and core themes. *The Cambridge handbook of artificial intelligence*, pages 15–33, 2014.

[29] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[30] Frederic B Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *The Journal of Symbolic Logic*, 9(2):49–50, 1944.

[31] 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, 1952.

[32] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[33] Marvin Minsky and Seymour A Papert. *Perceptrons, Reissue of the 1988 Expanded Edition with a new foreword by Léon Bottou: An Introduction to Computational Geometry*. MIT press, 2017.

[34] W Bernard and ME Hoff. Adaptive switching circuits, technical report, 1960.

[35] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[36] Marvin L Minsky and Seymour A Papert. Perceptrons: expanded edition, 1988.

[37] Michael Haenlein and Andreas Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, 61(4):5–14, 2019.

[38] Haocheng Tan. A brief history and technical review of the expert system research. In *IOP Conference Series: Materials Science and Engineering*, volume 242, page 012111. IOP Publishing, 2017.

[39] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[40] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[41] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988.

[42] Feng-hsiung Hsu. Ibm's deep blue chess grandmaster chips. *IEEE micro*, 19(2):70–81, 1999.

[43] Gina Kolata. Computer math proof shows reasoning power. *Math Horizons*, 4(3):22–25, 1997.

[44] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[45] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[47] J. Manyika, M. Chui, McKinsey Global Institute, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A.H. Byers. *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. McKinsey, 2011.

[48] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[49] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[52] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[53] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, pages 448–456. PMLR, 2015.

[55] D Kingma and J Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

[56] Mahesh Nagubandi, Raman Walia, Abhishek Karanath, and GN Pillai. Electric load forecasting using dual-stage attention network with cosine annealed warm restart schedule. In *2022 International Conference on Emerging Techniques in Computational Intelligence (ICETCI)*, pages 141–146. IEEE, 2022.

[57] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[58] 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 *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[59] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021.

[60] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[61] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[62] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2020.

[63] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6707–6717, 2020.

[64] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. ICCV*, pages 1026–1034, 2015.

[66] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[67] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[68] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

[69] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[70] Adam Paszke, S. Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zach DeVito, Zeming Lin, Alban Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[71] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

[72] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

[73] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[74] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

[75] Amir Gholami. Ai and memory wall, March 2021.

[76] Kwabena Boahen. Dendrocentric learning for synthetic intelligence. *Nature*, 612(7938):43–50, 2022.

[77] Mark Horowitz. 1. 1 computing's energy problem (and what, we can do about it). in, international solid-state circuits. In *Conference Digest of Technical Papers (ISSCC)*, 2014.

[78] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[79] David Attwell and Simon B Laughlin. An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow & Metabolism*, 21(10):1133–1145, 2001.

[80] Jiawei Zhang. Basic neural units of the brain: neurons, synapses and action potential. *arXiv preprint arXiv:1906.01703*, 2019.

[81] Tony Hey. Richard feynman and computation. *Contemporary Physics*, 40(4):257–265, 1999.

[82] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[83] Carver Mead. How we created neuromorphic engineering. *Nature Electronics*, 3(7):434–435, 2020.

[84] Carver Mead and Mohammed Ismail. *Analog VLSI implementation of neural systems*, volume 80. Springer Science & Business Media, 1989.

[85] Misha Mahowald and Misha Mahowald. The silicon retina. *An Analog VLSI System for Stereoscopic Vision*, pages 4–65, 1994.

[86] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73, 2011.

[87] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019.

[88] M Mitchell Waldrop. The chips are down for moore's law. *Nature News*, 530(7589):144, 2016.

[89] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature nanotechnology*, 15(7):529–544, 2020.

[90] Daniele Ielmini and H.-S. Philip Wong. In-memory computing with resistive switching devices. *Nature Electronics*, 1(6):333–343, Jun 2018.

[91] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[92] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[93] Akihito Sawa. Resistive switching in transition metal oxides. *Materials today*, 11(6):28–36, 2008.

[94] Yue Zha, Zhiqiang Wei, and Jing Li. Recent progress in rram technology: From compact models to applications. In *2017 China Semiconductor Technology International Conference (CSTIC)*, pages 1–4. IEEE, 2017.

[95] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301, 2010.

[96] Noboru Yamada, Eiji Ohno, Kenichi Nishiuchi, Nobuo Akahira, and Masatoshi Takao. Rapid-phase transitions of gete-sb2te3 pseudobinary amorphous thin films for an optical disk memory. *Journal of Applied Physics*, 69(5):2849–2856, 1991.

[97] Rakesh Jeyasingh, Jiale Liang, Marissa A Caldwell, Duygu Kuzum, and H-S Philip Wong. Phase change memory: Scaling and applications. In *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pages 1–7. IEEE, 2012.

[98] Daniele Ielmini and Yuegang Zhang. Analytical model for subthreshold conduction and threshold switching in chalcogenide-based memory devices. *Journal of Applied Physics*, 102(5):054517, 2007.

[99] Manan Suri, Olivier Bichler, Damien Querlioz, Boubacar Traoré, Olga Cueto, Luca Perniola, Veronique Sousa, Dominique Vuillaume, Christian Gamrat, and Barbara DeSalvo. Physical aspects of low power synapses based on phase change memory devices. *Journal of Applied Physics*, 112(5):054904, 2012.

[100] F Arnaud, P Zuliani, JP Reynard, A Gandolfo, F Disegni, P Mattavelli, E Gomiero, G Samanni, C Jahan, R Berthelon, et al. Truly innovative 28nm fdsoi technology for automotive micro-controller applications embedding 16mb phase change memory. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 18–4. IEEE, 2018.

[101] Daniele Ielmini, Simone Lavizzari, Deepak Sharma, and Andrea L Lacaita. Physical interpretation, modeling and impact on phase change memory (pcm) reliability of resistance drift due to chalcogenide structural relaxation. In *2007 IEEE International Electron Devices Meeting*, pages 939–942. IEEE, 2007.

[102] S Ikeda, J Hayakawa, Y Ashizawa, YM Lee, K Miura, H Hasegawa, M Tsunoda, F Matsukura, and H Ohno. Tunnel magnetoresistance of 604% at 300 k by suppression of ta diffusion in co fe b/ mg o/ co fe b pseudo-spin-valves annealed at high temperature. *Applied Physics Letters*, 93(8):082508, 2008.

[103] Claude Chappert, Albert Fert, and Frédéric Nguyen Van Dau. The emergence of spin electronics in data storage. *Nature materials*, 6(11):813–823, 2007.

[104] John C Slonczewski. Current-driven excitation of magnetic multilayers. *Journal of Magnetism and Magnetic Materials*, 159(1-2):L1–L7, 1996.

[105] Roberto Carboni, Stefano Ambrogio, W Chen, M Siddik, J Harms, A Lyle, W Kula, G Sandhu, and Daniele Ielmini. Understanding cycling endurance in perpendicular spin-transfer torque (p-stt) magnetic memory. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 21–6. IEEE, 2016.

[106] Yong Kyu Lee, Yoonjong Song, JooChan Kim, SeChung Oh, Byoung-Jae Bae, SangHumn Lee, JungHyuk Lee, UngHwan Pi, Boyoung Seo, Hyunsung Jung, et al. Embedded stt-mram in 28-nm fdsoi logic process for industrial mcu/iot application. In *2018 IEEE Symposium on VLSI Technology*, pages 181–182. IEEE, 2018.

[107] Kevin Garello, Farrukh Yasin, Sébastien Couet, Laurent Souriau, Johan Swerts, Siddharth Rao, Simon Van Beek, Wonsub Kim, Enlong Liu, Shreya Kundu, et al. Sot-mram 300mm integration for low power and ultrafast embedded memories. In *2018 IEEE Symposium on VLSI Circuits*, pages 81–82. IEEE, 2018.

[108] Pietro Gambardella and Ioan Mihai Miron. Current-induced spin–orbit torques. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1948):3175–3197, 2011.

[109] Rajesh Saha, Yogendra Pratap Pundir, and Pankaj Kumar Pal. Comparative analysis of stt and sot based mrams for last level caches. *Journal of Magnetism and Magnetic Materials*, 551:169161, 2022.

[110] N Izyumskaya, Y-I Alivov, S-J Cho, H Morkoç, H Lee, and Y-S Kang. Processing, structure, properties, and applications of pzt thin films. *Critical reviews in solid state and materials sciences*, 32(3-4):111–202, 2007.

[111] Halid Mulaosmanovic, Evelyn T Breyer, Stefan Dünkel, Sven Beyer, Thomas Mikolajick, and Stefan Slesazeck. Ferroelectric field-effect transistors based on hfo2: a review. *Nanotechnology*, 32(50):502002, 2021.

[112] Djohan Bonnet, Tifenn Hirtzlin, Atreya Majumdar, Thomas Dalgaty, Eduardo Esmanhotto, Valentina Meli, Niccolò Castellani, Simon Martin, Jean-Francois Nodin, Guillaume Bourgeois, et al. Bringing uncertainty quantification to the extreme-edge with memristor-based bayesian neural networks. 2023.

[113] Oleg Golonzka, J-G Alzate, U Arslan, M Bohr, P Bai, J Brockman, B Buford, C Connor, N Das, B Doyle, et al. Mram as embedded non-volatile memory solution for 22ffl finfet technology. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 18–1. IEEE, 2018.

[114] T Francois, L Grenouillet, J Coignus, P Blaise, C Carabasse, N Vaxelaire, T Magis, F Aussenac, V Loup, C Pellissier, et al. Demonstration of beol-compatible ferroelectric hf 0.5 zr 0.5 o 2 scaled feram co-integrated with 130nm cmos for embedded nvm applications. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 15–7. IEEE, 2019.

[115] Cong Xu, Dimin Niu, Naveen Muralimanohar, Rajeev Balasubramonian, Tao Zhang, Shimeng Yu, and Yuan Xie. Overcoming the challenges of crossbar resistive memory architectures. In *2015 IEEE 21st international symposium on high performance computer architecture (HPCA)*, pages 476–488. IEEE, 2015.

[116] Shimeng Yu and Pai-Yu Chen. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-State Circuits Magazine*, 8(2):43–56, 2016.

[117] Frank T Hady, Annie Foong, Bryan Veal, and Dan Williams. Platform storage performance with 3d xpoint technology. *Proceedings of the IEEE*, 105(9):1822–1833, 2017.

[118] Jiun-Jia Huang, Yi-Ming Tseng, Chung-Wei Hsu, and Tuo-Hung Hou. Bipolar nonlinear \hbox {Ni/TiO} _ {2}\hbox {/}\hbox {Ni} selector for 1s1r crossbar array applications. *IEEE Electron Device Letters*, 32(10):1427–1429, 2011.

[119] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020.

[120] Mirko Prezioso, Farnood Merrikh-Bayat, Brian D 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):61–64, 2015.

[121] Peng Yao, Huaqiang Wu, Bin Gao, Sukru Burc Eryilmaz, Xueyao Huang, Wenqiang Zhang, Qingtian Zhang, Ning Deng, Luping Shi, H. S. Philip Wong, and He Qian. Face classification using electronic synapses. *Nature Communications*, 8(1):15199, 2017.

[122] 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.

[123] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646, 2020.

[124] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. Mixed-precision in-memory computing. *Nature Electronics*, 1(4):246–253, 2018.

[125] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M. Shelby, Irem Boybat, Carmelo di Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan C. P. Farinha, Benjamin Killeen, Christina Cheng, Yassine Jaoudi, and Geoffrey W. Burr. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60, 2018.

[126] Seungchul Jung, Hyungwoo Lee, Sungmeen Myung, Hyunsoo Kim, Seung Keun Yoon, Soon-Wan Kwon, Yongmin Ju, Minje Kim, Wooseok Yi, Shinhee Han, et al. A crossbar array of magnetoresistive memory devices for in-memory computing. *Nature*, 601(7892):211–216, 2022.

[127] Stefano Ambrogio, Simone Balatti, Antonio Cubeta, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. Statistical fluctuations in hfo x resistive-switching memory: part i-set/reset variability. *IEEE Trans. Electron Devices*, 61(8):2912–2919, 2014.

[128] Stefano Ambrogio, Simone Balatti, Antonio Cubeta, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. Statistical fluctuations in hfo x resistive-switching memory: Part ii—random telegraph noise. *IEEE Trans. Electron Devices*, 61(8):2920–2927, 2014.

[129] G Bersuker, DC Gilmer, D Veksler, P Kirsch, LUCA Vandelli, ANDREA Padovani, Luca Larcher, K McKenna, A Shluger, V Iglesias, et al. Metal oxide resistive memory switching mechanism based on conductive filament properties. *Journal of Applied Physics*, 110(12):124518, 2011.

[130] Vivek Parmar and Manan Suri. Design exploration of hybrid cmos-oxram deep generative architectures. *arXiv preprint arXiv:1801.02003*, 2018.

[131] Erika Covi, Stefano Brivio, Alexantrou Serb, Themistoklis Prodromakis, M Fanciulli, and S Spiga. Hfo2-based memristors for neuromorphic applications. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 393–396. IEEE, 2016.

[132] Shinhyun Choi, Patrick Sheridan, and Wei D Lu. Data clustering using memristor networks. *Scientific reports*, 5(1):1–10, 2015.

[133] Mattia Boniardi, Daniele Ielmini, Simone Lavizzari, Andrea L Lacaita, Andrea Redaelli, and Agostino Pirovano. Statistics of resistance drift due to structural relaxation in phase-change memory arrays. *IEEE Transactions on Electron Devices*, 57(10):2690–2696, 2010.

[134] Stefano Ambrogio, Simone Balatti, Vincent McCaffrey, Daniel C Wang, and Daniele Ielmini. Noise-induced resistance broadening in resistive switching memory—part i: Intrinsic cell behavior. *IEEE Transactions on Electron Devices*, 62(11):3805–3811, 2015.

[135] Xing Wang, Hongxia Liu, Lu Zhao, Yongte Wang, and Shulong Wang. Improved resistive switching characteristics of atomic layer deposited al 2 o 3/la 2 o 3/al 2 o 3 multi-stacked films with al+ implantation. *Journal of Materials Science: Materials in Electronics*, 30:12577–12583, 2019.

[136] Thomas Dalgaty, Eduardo Esmanhotto, Niccolo Castellani, Damien Querlioz, and Elisa Vianello. Ex situ transfer of bayesian neural networks to resistive memory-based inference hardware. *Advanced Intelligent Systems*, 3(8):2000103, 2021.

[137] Eduardo Esmanhotto, Tifenn Hirtzlin, Djohan Bonnet, Niccolo Castellani, Jean-Michel Portal, Damien Querlioz, and Elisa Vianello. Experimental demonstration of multilevel resistive random access memory programming for up to two months stable neural networks inference accuracy. *Advanced Intelligent Systems*, 4(11):2200145, 2022.

[138] Jiyong Woo, Tien Van Nguyen, Jeong Hun Kim, Jong-Pil Im, Solyee Im, Yeriaron Kim, Kyeong-Sik Min, and Seung Eon Moon. Exploiting defective rram array as synapses of htm spatial pooler with boost-factor adjustment scheme for defect-tolerant neuromorphic systems. *Scientific Reports*, 10(1):11703, 2020.

[139] Shimeng Yu. Neuro-inspired computing with emerging nonvolatile memorys. *Proc. IEEE*, 106(2):260–285, 2018.

[140] Danijela Marković, Alice Mizrahi, Damien Querlioz, and Julie Grollier. Physics for neuromorphic computing. *Nature Reviews Physics*, 2(9):499–510, 2020.

[141] Ardavan Pedram, Stephen Richardson, Mark Horowitz, Sameh Galal, and Shahar Kvatinsky. Dark memory and accelerator-rich system optimization in the dark silicon era. *IEEE Des. Test*, 34(2):39–50, 2016.

[142] A. Grossi, E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. El Hajjam, R. Crochemore, J.F. Nodin, P. Olivo, and L. Perniola. Fundamental variability limits of filament-based rram. In *IEDM Tech. Dig.*, pages 4–7. IEEE, 2016.

[143] Stefano Ambrogio, Simone Balatti, Valerio Milo, Roberto Carboni, Zhong-Qiang Wang, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. Neuromorphic learning and recognition with one-transistor-one-resistor synapses and bistable metal oxide rram. *IEEE Trans. Electron Devices*, 63(4):1508–1515, 2016.

[144] Marc Bocquet, Damien Deleruyelle, Hassen Aziza, Christophe Muller, Jean-Michel Portal, Thomas Cabout, and Eric Jalaguier. Robust compact model for bipolar oxide-based resistive switching memories. *IEEE Trans. Electron Devices*, 61(3):674–681, 2014.

[145] Daniele Ielmini. Modeling the universal set/reset characteristics of bipolar rram by field- and temperature-driven filament growth. *IEEE Trans. Electron Devices*, 58(12):4309–4317, 2011.

[146] Zizhen Jiang, Yi Wu, Shimeng Yu, Lin Yang, Kay Song, Zia Karim, and H-S Philip Wong. A compact model for metal–oxide resistive random access memory with experiment verification. *IEEE Trans. Electron Devices*, 63(5):1884–1892, 2016.

[147] Haitong Li, Zizhen Jiang, Peng Huang, Yi Wu, H-Y Chen, Bin Gao, XY Liu, JF Kang, and H-SP Wong. Variation-aware, reliability-emphasized design and optimization of rram using spice model. In *Proc. DATE*, pages 1425–1430. IEEE, 2015.

[148] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 37(12):3067–3080, 2018.

[149] G. Piccolboni, G. Molas, J. M. Portal, R. Coquand, M. Bocquet, D. Garbin, E. Vianello, C. Carabasse, V. Delaye, C. Pellissier, T. Magis, C. Cagli, M. Gely, O. Cueto, D. Deleruyelle, G. Ghibaudo, B. De Salvo, and L. Perniola. Investigation of the potentialities of vertical resistive ram (vrram) for neuromorphic applications. In *IEDM Tech. Dig.*, pages 17–2. IEEE, 2015.

[150] 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 *IEDM Tech. Dig.*, pages 22–6. IEEE, 2019.

[151] Z. Zhou, P. Huang, Y. C. Xiang, W. S. Shen, Y. D. Zhao, Y. L. Feng, B. Gao, H. Q. Wu, H. Qian, L. F. Liu, X. Zhang, X. Y. Liu, and J. F. Kang. A new hardware implementation approach of bnns based on nonlinear 2t2r synaptic cell. In *IEDM Tech. Dig.*, pages 20–7. IEEE, 2018.

[152] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23(7):075201, 2012.

[153] Shahar Kvatinsky, Misbah Ramadan, Eby G Friedman, and Avinoam Kolodny. Vteam: A general model for voltage-controlled memristors. *IEEE Trans. Circuits Syst. II Express Briefs*, 62(8):786–790, 2015.

[154] Shinhyun Choi, Yuchao Yang, and Wei Lu. Random telegraph noise and resistance switching analysis of oxide based resistive memory. *Nanoscale*, 6(1):400–404, 2014.

[155] Spyros Stathopoulos, Alexantrou Serb, Ali Khiat, Maciej Ogorzałek, and Themis Prodromakis. A memristive switching uncertainty model. *IEEE Transactions on Electron Devices*, 66(7):2946–2953, 2019.

[156] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian

Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[157] 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*, 13:1383, 2020.

[158] Dmitri A Rusakov, Leonid P Savtchenko, and Peter E Latham. Noisy synaptic conductance: bug or a feature? *Trends in Neurosciences*, 2020.

[159] Naga Raghavan, Robin Degraeve, Andrea Fantini, Ludovic Goux, Sebastiano Strangio, Bogdan Govoreanu, DJ Wouters, Guido Groeseneken, and Malgorzata Jurczak. Microscopic origin of random telegraph noise fluctuations in aggressively scaled rram and its impact on read disturb variability. In *Proc. IRPS*, pages 5E–3. IEEE, 2013.

[160] Stefano Ambrogio, Simone Balatti, V McCaffrey, D Wang, and Daniele Ielmini. Impact of low-frequency noise on read distributions of resistive switching memory (rram). In *IEDM Tech. Dig.*, pages 14–4. IEEE, 2014.

[161] Kyosuke Ito, Takashi Matsumoto, Shinichi Nishizawa, Hiroki Sunagawa, Kazutoshi Kobayashi, and Hidetoshi Onodera. Modeling of random telegraph noise under circuit operation—simulation and measurement of rtn-induced delay fluctuation. In *Proc. ISQED*, pages 1–6. IEEE, 2011.

[162] Tibor Grasser. *Noise in Nanoscale Semiconductor Devices*. Springer Nature, 2020.

[163] N Jeremy Kasdin. Discrete simulation of colored noise and stochastic processes and 1/f/sup/spl alpha//power law noise generation. *Proc. IEEE*, 83(5):802–827, 1995.

[164] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proc. NIPS*, pages 4114–4122, 2016.

[165] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. ECCV*, pages 525–542. Springer, 2016.

[166] 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.

[167] Miguel Angel Lastras-Montaño, Osvaldo Del Pozo-Zamudio, Lev Glebsky, Meiran Zhao, Huaqiang Wu, and Kwang-Ting Cheng. Ratio-based multi-level resistive memory cells. *Sci. Rep.*, 11(1):1–12, 2021.

[168] Weisheng Zhao, Mathieu Moreau, Erya Deng, Yue Zhang, Jean-Michel Portal, Jacques-Olivier Klein, Marc Bocquet, Hassen Aziza, Damien Deleruyelle, Christophe Muller, Damien Querlioz, Nesrine Ben Romdhane, Dafiné Ravelosona, and Claude Chappert. Synchronous non-volatile logic gate design based on resistive switching memories. *IEEE Trans. Circuits Syst. I*, 61(2):443–454, 2014.

[169] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[170] Daniel Bankman, Lita Yang, Bert Moons, Marian Verhelst, and Boris Murmann. An always-on 3.8μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 222–224. ISSN: 2376-8606.

[171] Hossein Valavi, Peter J. Ramadge, Eric Nestler, and Naveen Verma. A mixed-signal binarized convolutional-neural-network accelerator integrating dense weight storage and multiplication for reduced data movement. In *2018 IEEE Symposium on VLSI Circuits*, pages 141–142. IEEE, 2018.

[172] 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. *Advances in neural information processing systems*, 32, 2019.

[173] 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.

[174] Laizhong Cui, Shu Yang, Fei Chen, Zhong Ming, Nan Lu, and Jing Qin. A survey on application of machine learning for internet of things. *International Journal of Machine Learning and Cybernetics*, 9(8):1399–1417, 2018.

[175] Pete Warden and Daniel Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[176] Amir M Rahmani, Tuan Nguyen Gia, Behailu Negash, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, and Pasi Liljeberg. Exploiting smart e-health gateways at the edge of

healthcare internet-of-things: A fog computing approach. *Future Generation Computer Systems*, 78:641–658, 2018.

[177] Yazdan Ahmad Qadri, Ali Nauman, Yousaf Bin Zikria, Athanasios V Vasilakos, and Sung Won Kim. The future of healthcare internet of things: a survey of emerging technologies. *IEEE Communications Surveys & Tutorials*, 22(2):1121–1167, 2020.

[178] Daniele Ielmini and H-S Philip Wong. In-memory computing with resistive switching devices. *Nature Electronics*, 1(6):333, 2018.

[179] 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.

[180] Cheng-Xin Xue, Yen-Cheng Chiu, Ta-Wei Liu, Tsung-Yuan Huang, Je-Syu Liu, Ting-Wei Chang, Hui-Yao Kao, Jing-Hong Wang, Shih-Ying Wei, Chun-Ying Lee, et al. A cmos-integrated compute-in-memory macro based on resistive random-access memory for ai edge devices. *Nature Electronics*, 4(1):81–90, 2021.

[181] Can Li, Jim Ignowski, Xia Sheng, Rob Wessel, Bill Jaffe, Jacqui Ingemi, Cat Graves, and John Paul Strachan. Cmos-integrated nanoscale memristive crossbars for cnn and optimization acceleration. In *2020 IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2020.

[182] Weier Wan, Rajkumar Kubendran, S Burc Eryilmaz, Wenqiang Zhang, Yan Liao, Dabin Wu, Stephen Deiss, Bin Gao, Priyanka Raina, Siddharth Joshi, et al. 33.1 a 74 tmacs/w cmos-rram neurosynaptic core with dynamically reconfigurable dataflow and in-situ transposable weights for probabilistic graphical models. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 498–500. IEEE, 2020.

[183] Riduan Khaddam-Aljameh, Milos Stanisavljevic, Jordi Fornt Mas, Geethan Karunaratne, Matthias Brändli, Feng Liu, Abhairaj Singh, Silvia M Müller, Urs Egger, Anastasios Petropoulos, et al. Hermes-core—a 1.59-tops/mm 2 pcm on 14-nm cmos in-memory compute core using 300-ps/lsb linearized cco-based adcs. *IEEE Journal of Solid-State Circuits*, 57(4):1027–1038, 2022.

[184] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, et al. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608(7923):504–512, 2022.

[185] Ahmed Ben Slimane, Amadeo Michaud, Olivia Mauguin, Xavier Lafosse, Adrien Bercegol, Laurent Lombez, Jean-Christophe Harmand, and Stéphane Collin. 1.73 ev algaas/ingap

heterojunction solar cell grown by mbe with 18.7% efficiency. *Progress in Photovoltaics: Research and Applications*, 28(5):393–402, 2020.

[186] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[187] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[188] Lorraine J Daston. Probabilistic expectation and rationality in classical probability theory. *Historia Mathematica*, 7(3):234–260, 1980.

[189] Mr. Bayes and Mr. Price. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. *Philosophical Transactions (1683-1775)*, 53:370–418, 1763.

[190] Mark Girolami. *A first course in machine learning*. Chapman and Hall/CRC, 2011.

[191] James M Joyce. Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer, 2011.

[192] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[193] 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. PMLR, 2016.

[194] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

[195] Kai Lai Chung. Markov chains. *Springer-Verlag, New York*, 1967.

[196] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[197] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International conference on machine learning*, pages 1218–1226. PMLR, 2015.

[198] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

[199] Nan Ding, Youhan Fang, Ryan Babbush, Changyou Chen, Robert D Skeel, and Hartmut Neven. Bayesian sampling using stochastic gradient thermostats. *Advances in neural information processing systems*, 27, 2014.

[200] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[201] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[202] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[203] Adrian Alan Pol, Victor Berger, Cecile Germain, Gianluca Cerminara, and Maurizio Pierini. Anomaly detection with conditional variational autoencoders. In *2019 18th IEEE international conference on machine learning and applications (ICMLA)*, pages 1651–1657. IEEE, 2019.

[204] Nikola Simidjievski, Cristian Bodnar, Ifrah Tariq, Paul Scherer, Helena Andres Terre, Zohreh Shams, Mateja Jamnik, and Pietro Liò. Variational autoencoders for cancer data integration: design principles and computational practice. *Frontiers in genetics*, 10:1205, 2019.

[205] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[206] Kamel-Eddine Harabi, Tifenn Hirtzlin, Clément Turck, Elisa Vianello, Raphaël Laurent, Jacques Droulez, Pierre Bessière, Jean-Michel Portal, Marc Bocquet, and Damien Querlioz. A memristor-based bayesian machine. *Nature Electronics*, pages 1–12, 2022.

[207] Brian R Gaines. Stochastic computing systems. *Advances in Information Systems Science: Volume 2*, pages 37–172, 1969.

[208] Thomas Dalgaty, Niccolo Castellani, Clément Turck, Kamel-Eddine Harabi, Damien Querlioz, and Elisa Vianello. In situ learning using intrinsic memristor variability via markov chain monte carlo sampling. *Nature Electronics*, 4(2):151–161, 2021.

[209] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[210] Xiangming Meng, Roman Bachmann, and Mohammad Emtiyaz Khan. Training binary neural networks using the bayesian learning rule. In *International conference on machine learning*, pages 6852–6861. PMLR, 2020.

[211] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[212] HM Dipu Kabir, Abbas Khosravi, Mohammad Anwar Hosen, and Saeid Nahavandi. Neural network-based uncertainty quantification: A survey of methodologies and applications. *IEEE access*, 6:36218–36234, 2018.

[213] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022.

[214] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[215] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[216] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural safety*, 31(2):105–112, 2009.

[217] Mary Cummings. Rethinking the maturity of artificial intelligence in safety-critical settings. *AI Magazine*, 42(1):6–15, 2021.

[218] Jyotika Athavale, Andrea Baldovin, Ralf Graefe, Michael Paulitsch, and Rafael Rosales. Ai and reliability trends in safety-critical autonomous systems on ground and air. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 74–77. IEEE, 2020.

[219] Richard M Re and Alicia Solow-Niederman. Developing artificially intelligent justice. *Stan. Tech. L. Rev.*, 22:242, 2019.

[220] Hiske Overweg, Anna-Lena Popkes, Ari Ercole, Yingzhen Li, José Miguel Hernández-Lobato, Yordan Zaykov, and Cheng Zhang. Interpretable outcome prediction with sparse bayesian neural networks in intensive care. *arXiv preprint arXiv:1905.02599*, 2019.

[221] Qiao Hua, Ye Yaqin, Bo Wan, Bo Chen, Yingqiang Zhong, and Jiao Pan. An interpretable model for ecg data based on bayesian neural networks. *IEEE Access*, 9:57001–57009, 2021.

[222] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Uncertainty decomposition in bayesian neural networks with latent variables. *arXiv preprint arXiv:1706.08495*, 2017.

[223] Lucy R. Chai. Uncertainty estimation in bayesian neural networks and links to interpretability. 2018.

[224] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International conference on machine learning*, pages 1183–1192. PMLR, 2017.

[225] Misha Belkin, Partha Niyogi, and Vikas Sindhwani. On manifold regularization. In *International Workshop on Artificial Intelligence and Statistics*, pages 17–24. PMLR, 2005.

[226] Torgyn Shaikhina and Natalia A Khovanova. Handling limited datasets with neural networks in medical applications: A small-data approach. *Artificial intelligence in medicine*, 75:51–63, 2017.

[227] Der-Chiang Li, Chiao-Wen Liu, and Susan C Hu. A fuzzy-based data transformation for feature extraction to increase classification performance with small medical data sets. *Artificial intelligence in medicine*, 52(1):45–52, 2011.

[228] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1944–1952, 2017.

[229] Davood Karimi, Haoran Dou, Simon K Warfield, and Ali Gholipour. Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis. *Medical image analysis*, 65:101759, 2020.

[230] George B Moody and Roger G Mark. The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[231] Ziti Fariha Mohd Apandi, Ryojun Ikeura, and Soichiro Hayakawa. Arrhythmia detection using mit-bih dataset: A review. In *2018 International Conference on Computational Approach in Smart Systems Design and Applications (ICASSDA)*, pages 1–5. IEEE, 2018.

[232] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. Ecg heartbeat classification: A deep transferable representation. In *2018 IEEE international conference on healthcare informatics (ICHI)*, pages 443–444. IEEE, 2018.

[233] Supriyo Bandyopadhyay and Marc Cahay. *Introduction to spintronics*. CRC press, 2015.

[234] Teruya Shinjo. *Nanomagnetism and spintronics*. Elsevier, 2013.

[235] Sabpreet Bhatti, Rachid Sbiaa, Atsufumi Hirohata, Hideo Ohno, Shunsuke Fukami, and SN Piramanayagam. Spintronics based random access memory: a review. *Materials Today*, 20(9):530–548, 2017.

[236] Atsufumi Hirohata, Keisuke Yamada, Yoshinobu Nakatani, Ioan-Lucian Prejbeanu, Bernard Diény, Philipp Pirro, and Burkard Hillebrands. Review on spintronics: Principles and device applications. *Journal of Magnetism and Magnetic Materials*, 509:166711, 2020.

[237] Zongxia Guo, Jialiang Yin, Yue Bai, Daoqian Zhu, Kewen Shi, Gefei Wang, Kaihua Cao, and Weisheng Zhao. Spintronics for energy-efficient computing: An overview and outlook. *Proceedings of the IEEE*, 109(8):1398–1417, 2021.

[238] 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. *Physical Review Applied*, 8(5):054045, 2017.

[239] Kerem Yunus Camsari, Rafatul Faria, Brian M Sutton, and Supriyo Datta. Stochastic p-bits for invertible logic. *Physical Review X*, 7(3):031014, 2017.

[240] Kerem Y Camsari, Brian M Sutton, and Supriyo Datta. P-bits for probabilistic spin logic. *Applied Physics Reviews*, 6(1):011305, 2019.

[241] Carlos Javier Garcia Cervera. *Magnetic domains and magnetic domain walls*. New York University, 1999.

[242] Stuart SP Parkin, Masamitsu Hayashi, and Luc Thomas. Magnetic domain-wall racetrack memory. *Science*, 320(5873):190–194, 2008.

[243] R Lavrijsen, G Malinowski, JH Franken, JT Kohlhepp, HJM Swagten, B Koopmans, M Czapkiewicz, and T Stobiecki. Reduced domain wall pinning in ultrathin pt/co 100-x b x/pt with perpendicular magnetic anisotropy. *Applied Physics Letters*, 96(2):022501, 2010.

[244] J-P Tetienne, T Hingant, J-V Kim, L Herrera Diez, J-P Adam, K Garcia, J-F Roch, S Rohart, A Thiaville, D Ravelosona, et al. Nanoscale imaging and control of domain-wall hopping with a nitrogen-vacancy center microscope. *Science*, 344(6190):1366–1369, 2014.

[245] E Martinez, L Lopez-Diaz, L Torres, C Tristan, and O Alejos. Thermal effects in domain wall motion: Micromagnetic simulations and analytical model. *Physical Review B*, 75(17):174409, 2007.

[246] Bivas Rana and YoshiChika Otani. Towards magnonic devices based on voltage-controlled magnetic anisotropy. *Communications Physics*, 2(1):90, 2019.

[247] K Duschek, D Pohl, S Fähler, K Nielsch, and K Leistner. Research update: Magnetoionic control of magnetization and anisotropy in layered oxide/metal heterostructures. *APL Materials*, 4(3):032301, 2016.

[248] R. Pachat, D. Ourdani, J.W. van der Jagt, M.-A. Syskaki, A. Di Pietro, Y. Roussigné, S. Ono, M.S. Gabor, M. Chérif, G. Durin, J. Langer, M. Belmeguenai, D. Ravelosona, and L. Herrera Diez. Multiple magnetoionic regimes in ta/$co_{20}fe_{60}b_{20}$/hfo_2. *Phys. Rev. Appl.*, 15:064055, Jun 2021.

[249] 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. *Frontiers in neuroscience*, 15:633674, 2021.

[250] Zhezhi He and Deliang Fan. Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11438–11446, 2019.

[251] Axel Laborieux, Marc Bocquet, Tifenn Hirtzlin, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Implementation of ternary weights with resistive ram using a single sense operation per synapse. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(1):138–147, 2020.

[252] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer science review*, 3(3):127–149, 2009.

[253] Julie Grollier, Damien Querlioz, KY Camsari, Karin Everschor-Sitte, Shunsuke Fukami, and Mark D Stiles. Neuromorphic spintronics. *Nature electronics*, 3(7):360–370, 2020.

[254] Bhavin J Shastri, Alexander N Tait, Thomas Ferreira de Lima, Wolfram HP Pernice, Harish Bhaskaran, C David Wright, and Paul R Prucnal. Photonics for artificial intelligence and neuromorphic computing. *Nature Photonics*, 15(2):102–114, 2021.

[255] Theoni Alexoudi, George Theodore Kanellos, and Nikos Pleros. Optical ram and integrated optical memories: a survey. *Light: Science & Applications*, 9(1):91, 2020.

[256] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[257] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[258] Arnold Zellner. Optimal information processing and bayes's theorem. *The American Statistician*, 42(4):278–280, 1988.

[259] Pier Giovanni Bissiri, Chris C Holmes, and Stephen G Walker. A general framework for updating belief distributions. *Journal of the royal statistical society. series b, statistical methodology*, 78(5):1103, 2016.

[260] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[261] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. *Advances in neural information processing systems*, 32, 2019.

[262] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[263] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[264] Samuel Kotz and Saralees Nadarajah. *Extreme value distributions: theory and applications*. world scientific, 2000.