

Physics-Grounded Neuromorphic Computing : From Spiking Neurons to Learning Algorithms

Marie Drouhin

▶ **To cite this version:**

Marie Drouhin. Physics-Grounded Neuromorphic Computing : From Spiking Neurons to Learning Algorithms. Micro and nanotechnologies/Microelectronics. Université Paris-Saclay, 2023. English. ⟨NNT : 2023UPAST168⟩. ⟨tel-04477502⟩

HAL Id: tel-04477502

https://theses.hal.science/tel-04477502v1

Submitted on 26 Feb 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

université PARIS-SACLAY

Physics-Grounded Neuromorphic Computing: From Spiking Neurons to Learning Algorithms

Calcul Neuromorphique basé sur la Physique :
Des Neurones à Impulsions aux Algorithmes d'Apprentissage

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) et à **l'Unité mixte de physique** (Université Paris-Saclay, CNRS, Thales), sous la direction de **Damien QUERLIOZ**, directeur de recherche CNRS, et sous le co-encadrement de **Julie GROLLIER**, directrice de recherche CNRS.

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

Marie DROUHIN

Composition du jury

Membres du jury avec voix délibérative

Laurent CARIO Directeur de recherche CNRS, IMN, Nantes Université	Président & Rapporteur
Daniel BRUNNER Chargé de recherche CNRS (HDR), Institut FEMTO-ST	Rapporteur & Examinateur
Marc BOCQUET Professeur des universités, IM2NP, Aix-Marseille Université	Examinateur
Adrien VINCENT Maître de conférences, IMS Bordeaux, Université de Bordeaux	Examinateur

THESE DE DOCTORAT
NNT : 2023UPAST168

université PARIS-SACLAY

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

Titre : Calcul Neuromorphique basé sur la Physique: Des Neurones à Impulsions aux Algorithmes d'Apprentissage

Mots clés : Propagation de l'équilibre, Nanoélectronique, IA efficace en énergie, Memristor

Résumé :

À l'ère numérique actuelle, caractérisée par une augmentation exponentielle de la puissance de calcul et de la capacité de mémoire, nous sommes confrontés à un défi pressant : la consommation d'énergie croissante de la technologie de l'information. La demande croissante de services intensifs en données, notamment l'intelligence artificielle (IA) et le cloud computing, souligne la nécessité de calculs respectueux de l'environnement et propices à l'innovation. Cette thèse explore le potentiel des memristors pour le calcul neuromorphique afin de réaliser une IA basse énergie.

Nous avons d'abord étudié des neurones composés de memristors volatils NbOx, offrant une alternative attrayante aux dispositifs CMOS classiques par leur scalabilité et leur dynamique. Ces dispositifs ont été caractérisés et présentent de nombreux comportements avec impulsions et bursting, tels que l'intégration et tir avec fuite ou le « phasic bursting ». Nous avons modélisé ces comportements à l'aide de dynamique non linéaire. En particulier, l'origine du « phasic bursting » a pu être élucidée : elle émerge d'une bifurcation de Hopf entre les régimes de cycle limite et de point fixe. Ce modèle peut s'avérer utile lors de la conception de puces neuromorphiques à impulsions. Du côté des algorithmes, nous avons adapté la Propagation à l'Équilibre (EqProp) aux systèmes physiques. EqProp, ancré dans la physique plutôt que dans le calcul, offre une perspective intéressante: exploiter la physique inhérente des systèmes réels pour l'apprentissage sur puce. Ce travail a porté sur l'adaptation des gradients continus aux memristors, où la programmation se fait sous forme d'impulsions. Pour cela, nous avons exploré diverses approches de discrétisation des gradients. La première méthode, la discrétisation ternaire, a démontré des taux de précision comparables à ceux de EqProp. Dans ce contexte, nous avons examiné le rôle des hyperparamètres et leur influence sur les performances du réseau. Nous avons ensuite introduit des mise-à-jours probabilistes, ce qui a amélioré les performances et a permis d'obtenir une distribution de mises-à-jours similaire au cas non discrétisé. Une autre approche avec davantage d'états quantifiés a été étudiée. Bien que cette dernière approche surpasse l'approche ternaire non probabiliste en termes de performances, elle a aussi des désavantages - une distribution plus large des impulsions et une consommation d'énergie plus élevée que l'approche ternaire probabiliste. Nous avons ensuite testé la résilience de la version discrétisée de EqProp en remplaçant les synapses idéales par des mesures de memristors HfOx. À cette fin, nous avons utilisé une plateforme où les memristors étaient accessibles individuellement. Nous avons d'abord effectué des simulations de contrôle : un perceptron à une seule couche et un réseau à deux couches avec première couche gelée. Ces expériences ont donné un aperçu du potentiel des synapses memristives, avec des précisions atteignant 78,1 % et 70,8 %, respectivement. Ensuite, nous avons étudié un réseau à une couche cachée. Deux définitions de poids ont été utilisées - les différences linéaire et logarithmique des conductances. La définition linéaire s'est avérée être la meilleure, avec une précision de 91 % comparé à 89,5 % obtenu par la version logarithmique. Pour améliorer ces résultats, nous avons limité la valeur de la conductance au dessus d'un seuil afin d'atténuer l'effet du régime bruité et non linéaire des memristors. Ainsi, les définitions linéaires et logarithmiques des poids ont respectivement atteint des précisions de 91,75 % et 92,14 %. Ces résultats constituent une étape importante pour la mise en œuvre pratique de l'apprentissage sur puce. En résumé, cette thèse explore le potentiel des memristors pour le calcul neuromorphique afin de réaliser une IA basse énergie. Ce domaine promet des avancées innovantes à l'intersection de la physique et de l'IA, offrant au calcul un futur plus durable et puissant.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Physique et ingénierie : Electrons, Photons, Sciences du vivant (EOBE)

Title: Physics-Grounded Neuromorphic Computing: From Spiking Neurons to Learning Algorithms

Keywords: Equilibrium propagation, Memristor, energy-efficient AI, Nanoelectronics

Abstract:

In our digital era, marked by an exponential growth in computational power and memory capacity, we are confronted with a pressing challenge: the escalating energy consumption of information technology. The increasing demand for data-intensive services, notably artificial intelligence and cloud computing, underscores the urgent necessity for energy-efficient computing solutions that are environmentally sustainable and foster innovation. This thesis explores the potential of memristors for neuromorphic computing to achieve energy-efficient AI. Because Spiking Neural Networks could offer the promise of low-energy learning, we first focused on hardware neurons composed of volatile NbOx filamentary memristors. These components emerge as appealing alternatives to conventional CMOS devices because of their scalability and spiking behaviors. These devices were characterized and reproduced numerous neuronal spiking and bursting behaviors, such as Leaky-Integrate-and-Fire characteristics and phasic bursting. These behaviors were modeled with non-linear dynamics equations, which accurately reproduced the experiments. In particular, the origin of the phasic bursting phenomenon could be investigated and was shown to emerge from a Hopf bifurcation between the limit cycle and the fixed point regimes. This model could be beneficial when designing spiking neuromorphic chips. We then focused on the algorithmic side and tackled the challenge of adapting the Equilibrium Propagation (EqProp) algorithm to physical systems. EqProp, rooted in physics rather than calculus, offers an attractive prospect—harnessing the inherent physics of hardware systems for on-chip learning. This work revolved around addressing the challenges posed by continuous-valued gradients in a memristor-based environment, where the mode of programming is a series of pulses. We explored various approaches to gradient discretization. The first method, called ternary discretization, demonstrated accuracy rates nearly matching those of conventional EqProp. In this context, we explored the role of hyperparameters and their influence on network performance. We then introduced probabilistic updates, which enhanced performance and gave a pulse distribution that closely mirrored the non-discretized scenario. Introducing more quantized states in gradient discretization outperformed the non-probabilistic ternary approach in terms of performance, but came with a trade-off—a broader pulse spread and increased energy consumption compared to the probabilistic ternary approach. Next, we tested the resilience of the discretized version of EqProp by replacing the ideal software synapses with HfOx memristor data. For this, we used a hardware platform with individually accessible memristors. We first performed control simulations: a single-layer perceptron and a two-layer network with a frozen first layer. These initial experiments provided a glimpse into the potential of memristor-based synapses, with accuracy rates reaching 78.1% and 70.8%, respectively. Next, we trained a one-hidden-layer network. Two distinct weight definitions were analyzed—the linear and logarithmic differences of conductances. The linear definition emerged as the best method, with 91% accuracy compared to the 89.5% achieved by the logarithmic one. To improve the results, we limited the conductance value if it fell below a threshold to mitigate the high-noise and non-linear regime of memristors. With this approach, the linear and logarithmic weight definitions achieved respectively 91.75% and 92.14% accuracy. These results constitute a milestone on the journey towards practical on-chip. In summary, this thesis explores the potential of memristors for neuromorphic computing to realize energy-efficient AI. This field promises breakthroughs at the intersection of physics and AI, offering a more sustainable and powerful future for computing.



Thèse effectuée au sein du **Centre de Nanosciences et de Nanotechnologies**
de l'Université Paris-Saclay
Centre de Nanosciences et Nanothechnolgies
91120 PALAISEAU cedex
FRANCE

À ma famille et mes amis.
To my family and friends.

Acknowledgements

Je tiens à exprimer ma profonde gratitude envers toutes les personnes qui ont joué un rôle essentiel dans la réalisation de cette thèse.

Tout d'abord, un grand merci à mes superviseurs de thèse, Damien et Julie, pour leur soutien continu tout au long de ma thèse, malgré le contexte du COVID en début de thèse. Votre encadrement, vos conseils avisés et votre expertise ont grandement contribué à mon épanouissement scientifique. Je souhaite remercier sincèrement les membres de mon jury de thèse : Laurent Cario, Daniel Brunner, Marc Bocquet, Adrien Vincent et Elisa Vianello, pour le temps qu'ils m'ont consacré et les échanges très constructifs que nous avons eus lors de ma soutenance.

De mon équipe du C2N, je souhaite tout d'abord remercier Clément qui a commencé sa thèse le même jour que moi et soutenu à quelques jours d'intervalle. Je suis ravie d'avoir partagé cette thèse avec toi, et j'en profite pour te féliciter d'être papa ! Kamel et Atreya, avec qui j'ai passé la quasi intégralité de la thèse, votre gentillesse et votre patience a été un atout majeur pour ce groupe. Je souhaiterais remercier en particulier Tanvi avec qui j'ai partagé un bureau pendant deux ans, et dont les discussions de tout et de rien me manquent. Tu es une personne formidable et je vous souhaite le meilleur, à toi, Patrick et Olliver. Je souhaite aussi remercier les doctorants que l'on voit peu car ils sont de l'autre côté de la France : Bastien et Djohan, les nouveaux thésards du groupe : Guillaume B. et Akib, mais aussi tous les anciens : Axel, Xing, Tifenn, Guillaume H. et les stagiaires Adrien R., Théo, Thomas, Thibaut, Adrien P..

Un grand merci à mes collègues de l'équipe neuromorphique du laboratoire Albert Fert, notamment aux permanents du groupe, Alice, Danjela, Dedalo. Je souhaite remercier particulièrement Jérémie pour ses discussions toujours très intéressantes sur EqProp, Nathan, et tous ceux qui sont arrivés après eux. Je souhaite remercier l'ensemble des membres du laboratoire Albert Fert pour leur accueil toujours très chaleureux. En particulier je tiens à remercier Yanis, Diane et Aurélien pour les soirées jeux toujours très sympathiques. Merci aussi aux doctorants de ma promotion, notamment Aya pour sa gentillesse, Kévin et Diana. Je souhaite aussi remercier Laurette et Pauline pour leur bonne humeur. Vous avez tous et toutes grandement participé à la très chaleureuse ambiance du labo. Merci aussi à la promotion suivante dont notamment Sarah, Hugo... Je ne peux pas citer tout le monde car vous êtes très nombreux, mais je pense à vous.

Gaétan, évidemment je ne t'ai pas oublié. On a parcouru tellement de route ensemble : de l'ESPCI au master ICFP et maintenant aussi la thèse... C'est un vrai plaisir d'avoir partagé tout cela avec toi, tu es un ami sur lequel on peut toujours compter. Ne t'inquiète pas, ta propre soutenance arrive à grand pas !

Un énorme merci à la famille du Lam Son vo Dao (art martial vietnamien pour ceux qui ne connaissent pas) : Alain, Amélie, Catherine, Liêm, Naïr, Rivo, Sylvain et à tous les autres. Vous avez été d'un grand soutien et vous êtes des personnes formidables.

À mes nouveaux collègues et amis : Yanis (j'étais obligée de te mettre là aussi, Spin-Ion ne serait pas pareil sans toi), Maïkane (la meilleure), mais aussi les managers Louis, Elmer, et les chefs Dafiné et Corina. Merci beaucoup à vous tous, c'est un grand plaisir d'avoir achevé ma thèse à vos côtés, et de commencé une nouvelle aventure ensemble.

Enfin, un immense merci à mes amis du lycée et d'avant : Jordane, Juliette, Alizée, Noémie, et merci à celles qui ont pu venir à ma soutenance !

Ma reconnaissance va également à ma famille, en particulier à mes deux sœurs, mes deux frères et mes parents, pour leur amour, leur soutien inconditionnel et leurs encouragements constants.

La même école, le même master, une thèse à moitié au même endroit et maintenant embauchés dans la même start-up... Merci à toi, Matthieu : ta présence constante et ton soutien inébranlable ont fait de cette thèse une aventure partagée. Merci de m'avoir accompagnée tout au long de ce parcours. Et surtout, merci d'être là.

Contents

Introduction **1**

1 State of the Art **7**
1.1 Deep Learning and Artificial Intelligence 8
1.1.1 First Neural Networks 8
1.1.2 Training Neural Networks 11
1.1.3 The deep learning revolution 16
1.1.4 The challenge of AI energy consumption 18
1.2 Taking Inspiration from the brain to realize efficient hardware 21
1.2.1 The elements of the brain 22
1.2.2 Bio-plausible learning 29
1.3 Hardware adapted for AI 34
1.3.1 Emerging devices 35
1.3.2 Integrating emerging memory devices in hardware 43

2 Characterization and Modeling of Spiking and Bursting in Experimental NbOx Neuron **49**
2.1 Fabrication and method 51
2.1.1 Fabrication 51
2.1.2 Electrical measurements 52
2.2 Results 54
2.2.1 Quasistatic properties 54
2.2.2 Spiking behavior: Origin and shape 56
2.2.3 Computational properties 60
2.2.4 Experimental demonstration of phasic bursting 64
2.2.5 Understanding phasic bursting with non-linear dynamics simulations . . 66
2.2.6 Discussion and limitations of this model 68
2.3 Conclusion 70

3 Adapting Equilibrium Propagation to Physical Systems **71**
3.1 Context 72

3.2 Equilibrium Propagation algorithm . . . 72
3.3 Need for gradient discretization . . . 78
3.3.1 Ideal synapse definition and physical constraints . . . 78
3.3.2 Methods . . . 78
3.4 Continuous-valued EqProp study . . . 79
3.5 Discretization strategies . . . 82
3.5.1 Ternary gradient . . . 82
3.5.2 Ternary gradient with probabilistic updates . . . 87
3.6 Increasing the number of quantized values of the gradient . . . 90
3.6.1 Presentation of the discretization step . . . 90
3.6.2 Results . . . 90
3.7 Balancing pulse allocation for reliable physics-based computing . . . 92
3.8 Conclusion . . . 93

4 Implementation of Equilibrium Propagation With Memristor Synapses 95
4.1 Context . . . 96
4.1.1 Non-linearity and asymmetry . . . 96
4.1.2 Intra-device and inter-device variability . . . 97
4.2 Hardware platform . . . 99
4.2.1 Presentation of the platform . . . 99
4.2.2 Memristors details . . . 101
4.2.3 Experimental setup . . . 102
4.2.4 Measurements . . . 103
4.3 Setting the problem . . . 104
4.3.1 What will be done in hardware, what will be done in software . . . 104
4.3.2 Definition of the weights . . . 104
4.3.3 Discretization and learning procedure . . . 105
4.3.4 Methods . . . 106
4.3.5 Challenges . . . 106
4.4 Controls . . . 107
4.4.1 Perceptron . . . 107
4.4.2 One-hidden layer network with first layer frozen . . . 108
4.5 Results for a one-hidden-layer network . . . 108
4.5.1 Accuracy obtained . . . 109
4.5.2 Hyperparameter tuning . . . 109
4.5.3 Comparison between different definitions of the weights . . . 112
4.5.4 Improving the accuracy . . . 113
4.6 Conclusion . . . 114

List of publications 127

Bibliography **150**

Résumé étendu en français **151**

List of Figures

1 State of the Art **7**

1.1 Perceptron with a McCulloch and Pitts neuron. 9

1.2 Hopfield network with graded neurons, adapted from Ref. [1]. 11

1.3 Above: Forward propagation. Below: Back Propagation from Ref. [2] 15

1.4 a: Simplified architecture of a CPU. b: Simplified architecture from a GPU. Both adapted from the NVIDIA documentation [3]. 19

1.5 a: Von Neumann bottleneck. b: Energy per operation, adapted from [4]. 21

1.6 a: Simplified architecture of a neuron (credits to Jack Shuai Li). b: Shape of an action potential with the resting phase, followed by the depolarization, repolarization, and hyperpolarization phases (reproduced from Ref. [5]). 23

1.7 Synapse drawing reproduced from Ref. [6]. 23

1.8 Twelve distinct firing patterns observed in individual neurons within the mammalian cortex [7] 25

1.9 Stochastic firing in rodent trigeminal neurons [8]. Cells were progressively depolarized to potentials near and above the spike threshold (-45 mV). TIntermittent discharges occurred near the threshold. Stochastic bursting occurred when neurons were biased to suprathreshold potentials (-39 mV holding current, top trace). 26

1.10 a: A comparison of the qualitative device requirements for three potential applications. The red line indicates experimental NVM data from previous studies. b–h: Conceptual representations of device requirements for computing: analog states (b), on/off ratio (c), linearity (d), symmetry (e), endurance (f), retention (g), and yield (h). The dashed and solid curves in b–e show the conductance adjustment of an analog NVM device. The conductance modifications of an NVM device during the training process typically occur within a partial scope rather than across the full range of the conductance window (f). After NVM devices are adjusted to various conductance levels, the conductance of the devices can vary over time, potentially leading to overlap between two levels (g). NVM devices that fail to reach the target conductance level are considered unsuccessful (h).Reproduced from Ref. [9] 37

1.11 Adapted from [10] 38

1.12 Left: Typical I-V characteristic of a resistive switching non-volatile memristor. Right: Typical I-V characteristic of a voltage-controlled threshold switching (TS) volatile memristor. Adapted from [11] 39

1.13 Different memory devices. a: Filamentary resistive switching RAM structure. b: Corresponding current–voltage characteristic of a bipolar RRAM switching device. c: Phase change memory structure. d: Corresponding resistance-voltage characteristic. e: Magnetic tunnel junction (MTJ) structure. f: Corresponding resistance–voltage characteristic of an STT-MRAM. g: Ferroelectric random access memory (FeRAM) structure. h: Corresponding polarization–voltage hysteretic characteristic (h). The orientation of electrical dipoles causes permanent polarization of the ferroelectric layer. From [12] 41

1.14 a: 1R architecture with memristor devices. b: Corresponding neural network, with in orange the input (corresponding to the voltages V), and in blue the output, corresponding to the current I. 44

1.15 a: 1T1R architecture with memristor devices. b: 1S1R architecture with memristor devices. Adapted from Ref [13]. 45

2 Characterization and Modeling of Spiking and Bursting in Experimental NbOx Neuron 49

2.1 Top view of devices taken with an optical white light microscope. 52

2.2 Positive current-controlled electroforming with input current going from 0 to 0.5 mA. 53

2.3 Schematics of the voltage pulse to current pulse converter used in the experiments. Here, $R_1 = R_3 = 1\ k\Omega$, $R_2 = R_4 = 100\ \Omega$, $R_S = R_{S'} = 400\ \Omega$. The operational amplifier has the following reference: BB OPA 356A 846LV. 54

2.4 Measured (dashed lines) and simulated (dotted line) I-V characteristics. The V sweep and I sweep correspond respectively to the voltage-controlled and current-controlled I-V characteristics. The hold point H is indicated in green and the threshold switching point TS in red. The inset shows a sketch of the structure of the device. 54

2.5 a: Voltage-controlled I-V characteristic repeated 100 times. b: Current-controlled I-V characteristic repeated 10 times. 55

2.6 a: Circuit for voltage-controlled spiking neuron. b: Device current I_d when a constant voltage of 1.52 V is applied. 57

2.7 Circuit diagram of the integrated NbO_x spiking neuron where C_{ext} and L_{ext} are respectively a parasitic capacitance and inductance. 58

2.8 a. Measurement of a single spike of a NbO_x neuron, with the four stages of an action potential indicated. b. Simulated spiking dynamics of the NbO_x neuron temperature T_d(colored curve) and current I_d (dots) for a constant input current of 180 μA. c. Simulation of the output voltage shape with respect to the value of the circuit inductance for a constant input current of 180 μA. 59

2.9 a: NbO_x neuron output as a function of input current amplitude. A 99 μs current ramp from 0 to 0.46 mA and 1 μs fall time is applied to the device. b: Simulation of NbO_x neuron output as a function of input current amplitude. A 100 μs current ramp from 0 to 680 μA and 100 ns fall time is applied to the device. This simulation is realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1. 60

2.10 a: Tonic spiking. The neuron receives a constant input current of 0.2 mA. b: Simulation of tonic spiking. The neuron receives a constant input current of 335 μA. This simulation is realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1. 61

2.11 Stochastic spiking obtained with a current of 0.109 mA. 61

2.12 a: Spike latency. A pulse with a duration of 1 μs, a rise time and fall time of both 100 ns and an amplitude of 0.131 mA is applied to the neuron. b: Simulation of spike latency. A pulse of duration of 1 μs and value 193 μA with a rise time and fall time of both 100 ns is applied to the neuron. This simulation is realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1. . . 62

2.13 a: Spatial integration. Comparison between two figures where a pulse of duration of 1 μs with a rise time and fall time of both 100 ns are applied to the neuron. The input current value is 0.13 mA on the left and 0.17 mA on the right. b: Simulation of spatial integration. Comparison between two figures where a pulse of duration of 1 μs with a rise time and fall time of both 100 ns are applied to the neuron. On the left, the value of the current is 150 μA. On the right, the input current value is 200 μA. These simulations are realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1. 63

2.14 a: Temporal integration. Three pulses of duration of 1 μs with a rise time and fall time of both 100 ns and of amplitude 0.110 mA are applied to the neuron. The frequency is 0.35 MHz on the left and 0.7 MHz on the right. b: Simulation of temporal integration. Three pulses of duration of 1 μs with a rise time and fall time of both 100 ns and of value 100 μA are applied to the neuron. On the left, the time period is 2.86 μs (frequency of about 0.35 MHz). On the right, the time period is 1.43 μs (frequency of about 0.7 MHz). These simulations are realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1. . . 64

2.15 Example of phasic bursting of the output voltage as a function of time. A current input of amplitude 0.47 mA is applied. The right panel zooms on the end of the phasic bursting. 65
2.16 Variation of the average frequency as a function of the input current. Right: Zoom on the phasic bursting regime, in order to get a statistical understanding of the phenomenon. In blue, the median frequency computed from the different average frequencies (grey dots) is plotted. 66
2.17 a, b, c: simulation of the trajectory (in blue) and the nullclines (in orange for $\dot{T} = 0$ and in green for $\dot{V} = 0$) for different input currents I_s of value 0.9, 0.96702 and 1.1 mA for each figure. The y-axis corresponds to the temperature T_d in the active volume of the device while the x-axis represents the voltage of the device V_d. The black arrows indicate the direction of the gradient at each point. 67
2.18 Simulations of the device current oscillations as a function of time for a current input I_s of 0.96702 mA 68
69figure.caption.39

3 Adapting Equilibrium Propagation to Physical Systems 71

3.1 a: Free phase. b: Nudge phase. 74
3.2 Accuracies obtained in the conventional EqProp case with parameters: $\eta_1 = 0.15$, $\eta_2 = 0.001$. 79
3.3 Histogram of the accumulated weight updates over 10 runs. a: Accumulated weight updates during learning for the first layer: ΔW_1^{tot}. b: Accumulated gradients during learning for the second layer: ΔW_2^{tot} with parameters $\eta_1 = 0.15$ and $\eta 2 = 0.001$. 80
3.4 Histogram of the positive and negative weight updates. a: Accumulated positive weight updates for the first layer: $\Delta W_1^{tot,BL}$. b: Accumulated negative weight updates for the first layer (absolute value): $\Delta W_1^{tot,BLb}$. c: Accumulated positive weight updates for the second layer: $\Delta W_2^{tot,BL}$. d: Accumulated negative weight updates for the second layer (absolute value): $\Delta W_2^{tot,BLb}$. With parameters: $eta1 = 0.15$ $eta2 = 0.001$. 81
3.5 Ternary discretization a: Schematic representing the pulses as a function of the continuous-valued gradient, with threshold θ_{th}. b: Effective weight update as a function of the number of pulses. 82
3.6 Performance of the ternary gradient method with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.00005, \nu = 0.00001$. 83
3.7 Cumulated update pulses distribution at the end of learning for the ternary gradient method. a: Cumulated pulses during learning for the first layer. b: Cumulated pulses during learning for the second layer. With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.005$, $\theta = 0.00005$, $t_{max} = 2$. 83

3.8 Cumulated positive or negative pulses distribution at the end of learning for the ternary gradient method. a: Cumulated positive pulses for the first layer. b: Cumulated negative pulses for the first layer. c: Cumulated positive pulses for the second layer. d: Cumulated negative pulses for the second layer. With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.005$, $\theta = 0.00005$, $t_{max} = 2$. 84
3.9 Performance of the ternary gradient method when the parameter θ varies with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \nu = 0.0001$. 85
3.10 Performance of the ternary gradient method when the parameter η_2 varies with parameters: $\eta_1 = 0.15, \theta = 0.00005, \nu = 0.0001$. 86
3.11 Performance of the ternary gradient method when the speed v varies with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.00005$. 86
3.12 Probability of obtaining a pulse as a function of the continuous-valued gradient, with threshold θ_{th}. 87
3.13 Accuracies obtained for the ternary gradient method with probabilistic updates with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.0002, \nu = 0.0004$. 88
3.14 Cumulated update pulses distribution at the end of learning for the ternary gradient method. a: Cumulated pulses during learning for the first layer. b: Cumulated pulses during learning for the second layer. With parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.0002, \nu = 0.0004$. 88
3.15 Cumulated positive or negative pulses distribution at the end of learning for the ternary gradient method. a: Cumulated positive pulses for the first layer. b: Cumulated negative pulses for the first layer. c: Cumulated positive pulses for the second layer. d: Cumulated negative pulses for the second layer.With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.005$, $\theta = 0.0002$, $\nu = 0.0004$. 89
3.16 Quantized discretization. a: Schematic representing the pulses as a function of the continuous-valued gradient, with threshold θ_{th}. b: Effective weight update as a function of the number of pulses. 90
3.17 Accuracies obtained for the gradient method with a maximum of 9 pulses applied per synapse per update. The parameters are: $\eta_1 = 0.15, \eta_2 = 0.01, \theta = 0.0015, \nu = 0.003$. 91
3.18 Cumulated update pulses distribution at the end of learning for the ternary gradient method. a: Cumulated pulses during learning for the first layer. b: Cumulated pulses during learning for the second layer. With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.01$, $\theta = 0.0015$, $t_{max} = 9$. 91
3.19 Cumulated positive or negative pulses distribution at the end of learning for the quantized gradient method. a: Cumulated positive pulses for the first layer. b: Cumulated negative pulses for the first layer. c: Cumulated positive pulses for the second layer. d: Cumulated negative pulses for the second layer. With parameters: $\eta_1 = 0.15, \eta_2 = 0.01, \theta = 0.0015, \nu = 0.003$. 92

4 Implementation of Equilibrium Propagation With Memristor Synapses 95

4.1 Asymmetry and non-linearity. a: Multilevel I–V characteristics of 1T1R RRAM TiN/HfO2/Ti/TiN device measured for increasing V_G, reproduced from [14]. b: Ten cycles programmed with 500 identical pulses of alternating depression and potentiation operations for a TiN/HfO2/Ti/TiN device with conditions ΔV and Δt are +0.9 V and 0.7 V. c: Evolution of the average conductance (straight line) and associated standard deviation (in grey) against the number of pulses for the data set in b. d: Distributions of the α parameter extracted from fit, where α is a multiplicative parameter that determines the magnitude of modification induced on the synaptic strength by a plasticity event. b,c,d reproduced from [15]. 97

4.2 Intra-device variability. a: Pink noise in a device reported from [16]. b: Random Telegraphic Noise (RTN) in a device, reproduced from [16]. c: Cycle-to-cycle variability on a single device for different set currents, reproduced from [17]. 98

4.3 Intra-device variability. a: Variability of the conductance in the LRS measured on 16 384 devices under six different SET programming currents fitted with a normal distribution (blue line), reproduced from [18]. b: Variability in the HRS for 100 devices fitted with a log-normal distribution, reproduced from [19]. 99

4.4 Fabricated Multimode Hybrid Memristor-CMOS Prototyping Platform. a simplified schematic of a 1T1R cell connected to analog multiplexers, illustrating the concept of switching the access mode. b schematic of the hybrid Memristor-CMOS die, consisting of two-mode circuitry: analog mode (orange color) supplied by nominal voltage VDD5, and digital mode (blue color) supplied by VDD, VDDC, and VDDR. 100

4.5 Fabricated Multimode Hybrid Memristor-CMOS Prototyping Platform. a layout view, b Schematic of the analog mode circuitry, with shift registers selecting inputs via Multiplexers, which consist of analog MUXs connected to SL, BL, and WL terminals. Each MUX is controlled by a shift register, to choose one of the two analog inputs. c Optical microscopy photograph. 101

4.6 a Stack of the HfOx memristor used. b Scanning electron microscopy image of a memristor in the back end of line of the hybrid memristor/CMOS process, reproduced from [20]. 102

4.7 Experimental setup. 102

4.8 Set and Reset configurations 103

4.9 a: Example of synaptic plasticity in a memristor. b: Example of a non-linear regime in a memristor c: Conductances of 314 devices. 104

4.10 Performance for a perceptron on the MNIST task, obtained for a network of size 784-10 and parameters $\eta = 0.1$, $\alpha = 2000$, $\theta = 0.002$. 107

4.11 Performance obtained with a random frozen first layer with parameters$\eta_1 = 0.0$, $\eta_2 = 0.001$, $\theta = 0.5e-5$, $\alpha_1 = 4000$, $\alpha_1 = 3000$. 108

4.12 Best performance obtained with the weights defined as the linear difference of conductances and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000, \theta = 0.0002$. . 109

4.13 Impact of the variation of the threshold θ on the performance. The weights are defined as the linear difference of conductances and the parameters used are $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000$. 110

4.14 Impact of the variation of the scale factor of the first layer α_1 on the performance. The weights are defined as the linear difference of conductances and the parameters used are $\eta_1 = 0.2, \eta_2 = 0, \alpha_2 = 5000, \theta = 0.0002$. 110

4.15 Impact of the variation of the scale factor of the second layer α_2 on the performance. The weights are defined as the linear difference of conductances and the parameters used are $\eta_1 = 0.2, \eta_2 = 0.0 \alpha_1 = 5000 \theta = 0.0002$. 111

4.16 Comparison of the two different definitions of the weights. a: Best accuracy obtained with a linear definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000, \theta = 0.0002$. b: Best accuracy obtained with a logarithmic definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 1.2, \alpha_2 = 1.0, \theta = 0.0003$. . 112

4.17 a: Plot of the 314 different conductances evolution. b: Plot of the logarithm of the 314 different conductances evolution. Both scales have been adjusted to align with high conductance states. 113

4.18 Comparison of the two different definitions of the weights when a conductance (or resistance) threshold is applied. a: Accuracies obtained with a linear definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000, \theta = 0.0002$. b: Accuracies obtained with a logarithmic definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 1.2, \alpha_2 = 1.0, \theta = 0.0003$. 114

4.19 a: Schematic of the circuit used in these experiments.b: Example of result for $V_{s,peak} = 3.0$ and an input frequency of 1 MHz. 121

4.20 Variation of the output frequency f_{out} as a function of the input frequency f_{in} for different peak-to-peak voltages.: $V_{s,peak} = 3.0V$ b: $V_{s,peak} = 3.25V$ c: $V_{s,peak} = 3.5V$ d: $V_{s,peak} = 3.75V$ 122

4.21 a: Simulations of the output frequency f_{out} as a function of the input one f_{in}. b: Simulations of the frequency ratio $\frac{f_{out}}{f_{in}}$ as a function of the input frequency f_{in}. The parameters used are: peak-to-peak voltage $V_{s,peak} = 2.06$ V, load resistance $R_L = 4\ k\Omega$, frequency averaged over a time of $T_f = 0.0001\ s$ and with a number of points of $N = 1000001$. All the parameters of the model are the one presented in Table. 2.1. 123

4.22 Simulations when the label is zero. a: Comparison between inference before a learning step and after a learning step. b: Current in the output device during the free and nudging phases. c: Frequency of the output device during the free and nudging phases. d: Values of $\dot{\rho}$ and $\dot{\rho}_{avg}$ during the free and nudging phases. The parameters used are $x_1 = 0.00034091$, $x_2 = 0.00041204$, $W_1 = 0.38711533$, $W_2 = -0.04242794$, $b = 0$. 124
4.23 Simulations when the label is one. a: Comparison between inference before a learning step and after a learning step. b: Current in the output device during the free and nudging phases. c: Frequency of the output device during the free and nudging phases. d: Values of $\dot{\rho}$ and $\dot{\rho}_{avg}$ during the free and nudging phases. The parameters used are $x_1 = 0.00034091$, $x_2 = 0.00041204$, $W_1 = 0.38711533$, $W_2 = -0.04242794$, $b = 0$, $\eta = 10$, $\eta_b = 0.4e-9$. 125
4.24 a : Caractéristiques I-V controlées en tension (vert) et en courant (orange). La structure des neurones memristifs est présentée en insert. b: Trains d'impulsions lors de l'application d'un courant constant. c: Illustration du phasic busrting lors de l'application d'un courant constant : les impulsions s'arrêtent même sans variation du courant d'entére. 153
4.25 Précisions obtenues pour différentes méthodes de discrétisation du gradient. a : Ternarisation du gradient. b : Ternarisation avec mises à jour probabilistes. c : Quantification du gradient (dix-neuf états différents). 154
4.26 Performances obtenues pour un réseau de neurones à une couche cachée. a : Performances obtenues avec une définition linéaire en conductance des poids. b : Performances obtenues avec une définition logarithmique en conductance des poids. c : Performances obtenues avec une définition linéaire en conductance des poids avec seuil sur la conductance, pour trois différentes valeurs du seuil. d: Performances obtenues avec une définition logarithmique en conductance des poids avec seuil sur la conductance, pour trois différentes valeurs du seuil. 155

List of Tables

2.1 Table of the parameters used in the simulations. These parameters where obtained by fitting the I-V characteristics and estimated by fitting the shape of the spikes. 59

4.1 Parameters used during the measurements. 103

Introduction

As we navigate through the 21st century, the urgency of addressing climate change is increasingly evident. Rising global temperatures, melting ice caps, and extreme weather events serve as stark reminders of the environmental crisis we face. The Intergovernmental Panel on Climate Change (IPCC) warns that we have a limited window of opportunity to prevent catastrophic global warming. This pressing issue calls for immediate and coordinated action across all sectors of society, including the field of information technology (IT). The IT sector, while instrumental in driving global progress, is also a significant contributor to global energy consumption. Data centers, the backbone of our increasingly digital world, are estimated to account for about 1% of global electricity use [21]. Furthermore, the energy consumption of IT is projected to increase with the growing demand for data-intensive services such as cloud computing, artificial intelligence, and high-performance computing [22]. This escalating energy demand underscores the need for developing energy-efficient computing solutions. The pursuit of such solutions not only aligns with the global climate goals but also presents an opportunity for innovation and advancement in computing technologies.

The trajectory of human advancement is strikingly illustrated by the fact that while it took us approximately 4000 years to progress from the creation of the wheel to the first successful airplane flight, a mere 66 years elapsed between that inaugural flight and Neil Armstrong's historic moon landing. This accelerated pace of development was significantly enabled by the advent of computing technology, which provided us with the tools to automate intricate tasks and execute large-scale calculations with unprecedented speed. The computer, now an indispensable instrument in our daily lives, is the culmination of centuries of technological evolution spanning various scientific fields such as physics, mathematics, electronics, and computer science. Each new generation of computing technology has broadened our capabilities, unlocking possibilities that were previously beyond our reach. This cycle of necessity and invention has fueled growth at an exponential rate, leading to an ever-increasing demand for computational power and memory. The development of computers did not occur in isolation. It required simultaneous advancements in multiple fields of science, including the crucial area of memory technology. In this context, memory refers to the data that a calculation requires to be executed. At its core, a computer is made up of two key components: the memory unit, which stores data, and the arithmetic-logic unit, which carries out operations on that data. The complexity of a task is closely tied to the memory it requires for computation.

The journey towards modern computing began with mechanical calculators such as the abacus and the Pascal calculator. However, these devices, while capable of efficiently performing arithmetic operations, were not programmable and thus unsuitable for automation. The first design of a general-purpose computer, albeit mechanical, was proposed by Charles Babbage in 1837 [23]. Named the Analytical Engine, it 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. The first digital, electronic, programmable computer, the Electronic Numerical Integrator and Computer (ENIAC), was completed in 1945 [24]. The construction of this computer was enabled by developments in electronics in the earlier half of the twentieth century, particularly the invention of the thermionic vacuum tube [25]. A technological successor to ENIAC was EDVAC (Electronic Discrete Variable Automatic Computer), which was completed in 1949 [26]. The celebrated engineer John von Neumann was involved with this project as a consultant, and he proposed the architecture-level organization of a computer [27], which came to be known as the von Neumann architecture. The following decades saw the development of transistors and integrated circuits, replacing vacuum tubes entirely. These rapidly decreased the computer's cost and size and culminated in the invention of the first personal desktop computer, IBM-PC, in 1966 [28, 29]. However, as we continue to push the boundaries of what computers can do, we are also confronted with new challenges. One of the most pressing of these is the escalating energy demand of modern computing systems. The history of computing has been marked by an exponential increase in memory and computational needs, also highlighted by Moore's law which states that the number of transistors on a microchip doubles every two years [30, 31]. However, as transistor sizes approach the atomic scale, quantum effects and other physical phenomena become significant challenges. Current silicon-based technologies are approaching these physical limits, making it increasingly difficult to continue shrinking transistors at the pace predicted by Moore's Law. In contrast, the computational demands of Artificial Intelligence (AI), especially deep learning, have been growing at a pace that outstrips the predictions of Moore's Law, doubling every 5 to 6 months [32], far outpacing the transistor density increase predicted by Moore's Law. If current trends continue, we may soon reach a point where traditional computing hardware is unable to efficiently support the training and deployment of advanced AI models. This could stifle innovation and slow the pace of AI advancements.

In response to this challenge, the field of artificial intelligence (AI) is undergoing a significant shift. Traditionally, AI models were trained and deployed on powerful servers in data centers, a practice that is increasingly giving way to a new trend known as Edge AI. This shift towards Edge AI is not only a response to the evolving dynamics of our digital world but also a crucial step towards addressing the pressing issue of climate change. The proliferation of Internet of Things (IoT) devices, such as smart home appliances, wearable devices, and connected vehicles, has led to an explosion in the amount of data being generated at the edge of the network. This surge in data has highlighted the inefficiencies of processing it in the cloud due to latency and bandwidth constraints of network connections. Edge AI addresses this challenge by moving the AI closer to where the data is generated and used, thereby reducing latency and bandwidth usage but also reducing the energy consumed in data transmission. At the same time, privacy and security have become paramount concerns in our increasingly

interconnected world. Edge AI offers a solution to these concerns by processing data locally on the device, rather than sending it to the cloud [33]. This approach ensures that sensitive information remains private and secure, addressing a key concern in today's digital landscape. Moreover, advances in hardware technology have opened up new possibilities for Edge AI. The development of specialized AI chips and efficient model compression techniques have made it possible to run complex AI models on devices with limited computational resources. This has enabled powerful AI capabilities to be embedded in small, low-power devices, extending the reach of AI to new areas and applications [34]. The rise of Edge AI has significant implications for a wide range of applications. From autonomous vehicles and drones that require real-time decision-making, to healthcare devices that need to process sensitive patient data, to smart home devices that aim to provide personalized experiences while respecting user privacy, Edge AI is poised to revolutionize these fields. Importantly, by enabling energy-efficient AI, Edge AI also plays a crucial role in our collective efforts to combat climate change. However, the shift toward Edge AI also presents new challenges. These include the need for adapted hardware, efficient algorithms that can run on resource-constrained devices and the design of new learning paradigms that can adapt to the unique characteristics of edge devices and networks.

This thesis will focus on bridging the gap between hardware and algorithms to realize learning with real devices. In particular, Chapter 1 will introduce key concepts that will be used in this work, starting with a broad explanation of deep learning and recent breakthroughs, before diving into the typical architecture used on computers used for such tasks. Then, I present a source of inspiration to perform adapted hardware: the brain. After a general introduction to neurons and synapses, I present different neuronal behaviors and models used to reproduce them. Then I show different algorithms that are bio-plausible and may be a clue as to how the brain actually learns. In the last section, I present emerging hardware technologies, both synapse and neuron-like, and a few different architectures used in this context.

Chapter 2 presents a spiking neuron based on niobium oxide. This work first explores the quasistatic I-V characteristics of such a device, before diving into the computational properties. This type of neuron is shown to reproduce different types of neural behaviors such as tonic spiking, leaky-integrate and fire, all-or-nothing firing, stochastic firing, and phasic bursting. This last behavior is observed statistically, and a simple model based on non-linear dynamics is able to reproduce all the behaviors shown above. This paves the way to spiking neural networks.

Chapter 3 presents and explains the Equilibrium Propagation algorithm. It then explains why adapting this algorithm to perform on-chip learning is necessary, and proposes ways to discretize the gradient in order to both have a good accuracy but also to not have too many pulses applied during learning. Indeed, this is key to good energy efficiency.

Chapter 4 presents the realization of the algorithm presented in Chapter 3 with memristors. Experimental data is used in simulations to explore the resilience of the EqProp algorithm with imperfect synapses. Control simulations on perceptron and two-layer networks with first

frozen layer are performed. A one-hidden layer network is then optimized, and two different definitions of the synaptic weights are explored.

Chapter 1

State of the Art

Computers are now able to realize challenging tasks, in particular Artificial Intelligence (AI) tasks. However, the classical computer architecture consumption is large compared to the brain, which has similar functionalities. This is where neuromorphic computing is born, which aims at producing new hardware able to compute the same tasks but at very low energy.

1.1 Deep Learning and Artificial Intelligence

Understanding and creating an intelligent system that can produce cognitive tasks is a problem that has been tackled in many ways, either by starting from the task and trying to find a formal way to express the reasoning of the brain, or by starting from understanding how the brain works to get to the tasks [35]. In this section, we will focus on the emergence of the second approach.

If initially neural networks research was motivated by understanding and reproducing the functionalities of the brain, it is now a tool that is very loosely inspired by the architecture of the brain [36]. It is a set of different algorithms, which perform a non-linear transformation of an input to match to an output. This input can be an image, an audio file, a text in order to be used for image classification, text prediction, and so on.

1.1.1 First Neural Networks

1.1.1.1 First artificial neuron and network

In 1943, McCulloch and Pitts presented one of the first attempts at modeling a biological neural network [37]. In this work, the neurons are functional logic devices with a binary response, in order to capture their "all-or-nothing" behavior. First, a neuron's output is a function of its input. Secondly, if a neuron's input is higher than a threshold, the neuron will output a one (corresponding to a maximum frequency firing), and else it will output a zero (not firing). Moreover, in this model, the neuron receives as input a weighted sum of the other connected neurons. The weights, by which another neuron's output is multiplied, correspond to the synapses' strength. These synapses can be either excitatory (corresponding to a positive weight) or inhibitory (corresponding to a negative weight). This model was shown to perform logical operations such as AND, OR, or NOT. This model of a neuron is shown in orange in Fig. 1.1. However, in this model, no algorithm is used to tune the synapses' strengths, which are then fully static. This work introduced fundamental concepts which paved the way for artificial intelligence research but also introduced computational neuroscience.

In 1949 Hebb suggested that when two neurons are repeatedly activated simultaneously, the connection between them strengthens [38]. This concept is often summarized by the phrase "cells that fire together, wire together." In particular, this introduces the concept that the strength of the synapses is variable. Hebb's theory laid the foundation for understanding how synaptic

connections in the brain are modified through experience and learning.

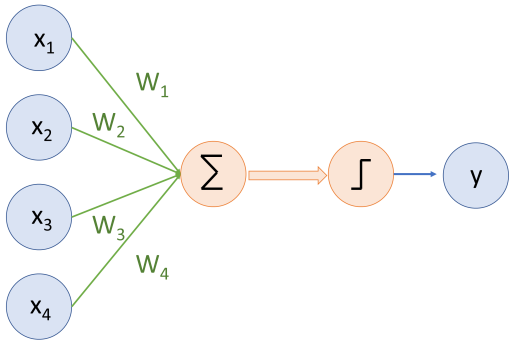


Figure 1.1: Perceptron with a McCulloch and Pitts neuron.

1.1.1.2 Perceptron

An important step in the history of deep learning is the perceptron, a model published in 1957 by Rosenblatt [39]. Taking inspiration from the visual cortex, he designed a machine for pattern classification. The perceptron is intended to mimic the behavior of a single neuron in the brain, which receives multiple inputs, processes them, and produces an output based on certain activation rules, as presented in Fig. 1.1. Based on McCulloch and Pitts' work, all neurons are considered all-or-nothing neurons, and synapse strengths, referred also as weights, are variable. Rosenblatt proposes to train the weights to get good accuracy. The learning rule is not explicitly described in the original paper but suggests an iterative adjustment of the weights based on input-output associations to improve the perceptron's performance.

In follow-up work [40], the perceptron model is more formally described. Given an input $X = x_1, ..., x_n$, and given a set of weights $W_1, ..., W_n$, the output is

$$o = H(\Sigma W_i x_i + b), \tag{1.1}$$

where H is the Heaviside step function and b is a bias term.

The perceptron learning rule (PRL) gives:

$$\Delta W_i = \alpha(t - o) \cdot x_i, \tag{1.2}$$

where t is the target associated with the input X. In this work, Rosenblatt proves the convergence of a learning algorithm in a simple one-layer perceptron, using an iterative tuning of the weights to reach the desired computation. With the perceptron convergence theorem, this model is shown to only solve linearly separable tasks [41].

The limitations of the single-layer perceptron appear clearly in 1969, thanks to the work of Minsky and Papert [42]. In particular, the XOR problem cannot be solved. In 1986, Rumelhart, Hinton and Williams introduced the concept of multi-layer perceptrons (also known as feed-forward neural networks) as a way to overcome the limitations of single-layer perceptrons [43]. They demonstrate that by introducing additional layers and nonlinear activation functions, it becomes possible to learn and represent more complex patterns.

1.1.1.3 Hopfield Networks

In 1982, Hopfield introduced a new kind of network, based on the idea that a physical system could store information and could be addressed to retrieve it even with a corrupt or incomplete query [44]. His idea is based on the intrinsic minimization of energy in a physical system.

More precisely, his neurons are all-or-nothing neurons inspired by McCulloch and Pitts' work [37] which take as input a weighted sum of other neurons and synapses. However, the originality lies in the fact that contrary to the perceptron that is organized in layers (feed-forward), the Hopfield network's connections can be completely random, as shown in Fig. 1.2a. Moreover, taking inspiration from Ising spin systems, the neurons' states are not updated in a synchronous manner, but dynamically updated by being randomly drawn and updated one by one.

Considering the patterns P^k that have to be stored, the neurons σ_i, and W_{ij} (with $W_{ii} = 0$) the strengths of the tunable synapses follow the equation:

$$W_{ij} = \sum_k (2P_i^k - 1)(2P_j^k - 1), \tag{1.3}$$

reminiscent of the Hebbian learning rule [38]. Hopfield proposes an energy-based framework that would govern the system under the condition that the matrix W is symmetric ($W_{ij} = W_{ji}$), which takes deep inspiration from Ising spin systems' Hamiltonians:

$$E = -\frac{1}{2} \sum_j \sum_{i \neq j} W_{ij} \sigma_i \sigma_j \tag{1.4}$$

where W is akin to the exchange coupling, and σ is comparable to spins. If the pattern P presented is altered, the energy will relax to the closest local minimum, which will correspond to the uncorrupted information. A typical example of an energy landscape is presented in Fig. 1.2b and d. The number of patterns that can be stored obviously depends on the number of neurons, and Hopfield found in his original paper that this critical number is equal to 0.15 N where N is the number of neurons. If the stored pattern number is not too large and the pattern is uncorrelated, it is possible to add a new pattern to the collection of memories by using the Hebbian learning rule. The memory feature of this network emerges from the very high num-

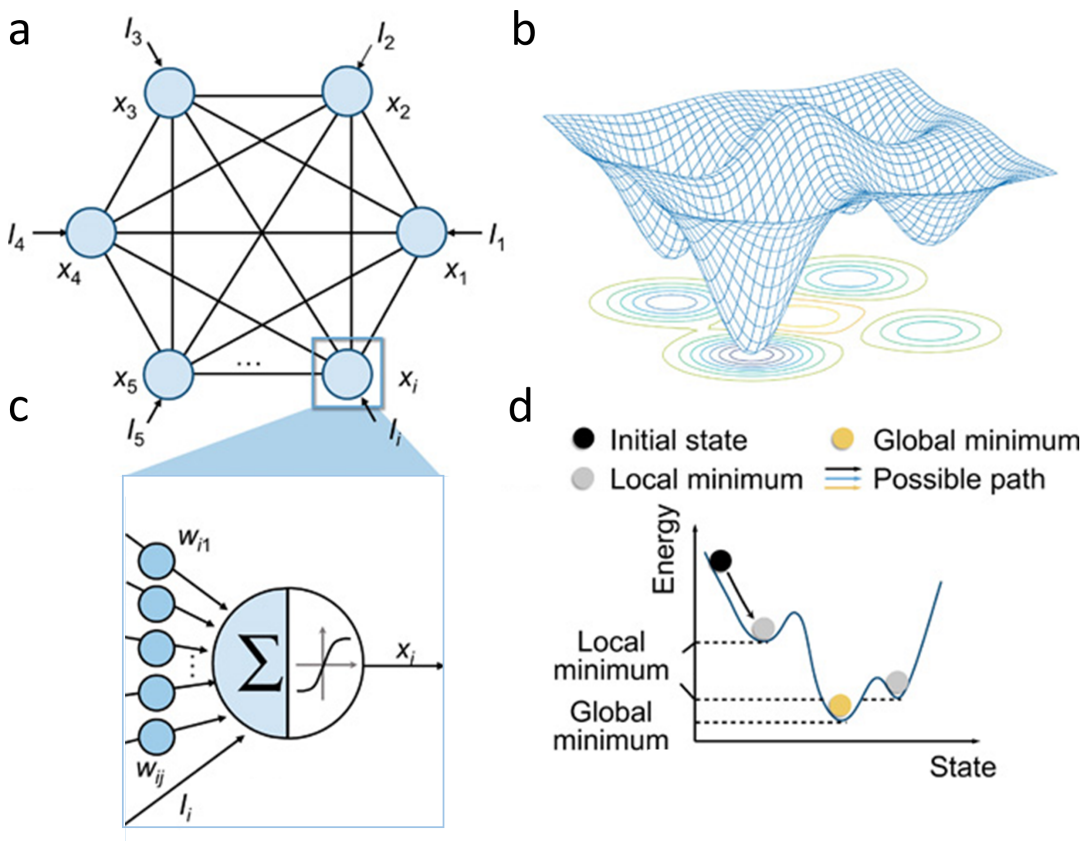


Figure 1.2: Hopfield network with graded neurons, adapted from Ref. [1].

ber of connections between the different neurons, but it is not very dependent on the precise characteristics of these neurons. For example, Hebb later introduced graded neurons in this type of network, whose output is a sigmoid of its input [45].

By introducing an energy function whose minimum corresponds to the best solution of an optimization problem, a Hopfield network is capable of solving this type of problem [46].

1.1.2 Training Neural Networks

Because it was shown in 1969, that the single-layer perceptron could only learn linearly separable tasks [42], and because of a lack of methods to train deeper networks, research considerably slowed in the 70s, until the emergence of the backpropagation training algorithm [47]. Even if earlier instances of this training method seem to exist [48–50], it is the work of Rumelhart, Hinton, and Williams in 1985 that introduced the backpropagation formalism to solve multi-layer fully connected networks and popularized this concept [43, 51]. Hinted in the work of Ref. [42], this paper shows that adding intermediary "hidden" neurons is crucial to learning non-linearly separable tasks, for example, when solving the XOR problem.

Later work, published in 1989, shows that a one-hidden layer network can approximate any continuous function [52]. However, this theoretical guarantee does not give any hint about how hard it is to train such a network, or how many neurons there must be. In practice, "deeper" networks (i.e. with more hidden layers) need to be used with fewer neurons on each layer in order to train with ease more complex functions.

1.1.2.1 Supervised, unsupervised

Numerous different criteria can be used to categorize learning algorithms. One commonly used criterion is the type of learning employed. In the field of machine learning, learning algorithms are traditionally classified into two main categories: supervised learning and unsupervised learning [53]. Other categories are also emerging and will be briefly presented in the following text.

Supervised learning is a technique where every input x in the training dataset has a corresponding label t. The goal of the learning procedure is for the neural network to approximate the mapping function f defined as f(x) = t. This framework can be used either in regression tasks where the target is continuous or in classification tasks where the target is categorical. Usually, a huge number of data needs to be available for training in order to obtain the best performance possible.

Unfortunately, labeled data can be scarce, expensive to obtain, or simply unavailable. For this reason, being able to train a network without knowing a target can be very useful. Unsupervised learning algorithms operate on unlabeled data, where no explicit target labels are available. The objective of unsupervised learning is to discover patterns, structures, or relationships in the data without any predefined notion of what the output should be [54]. Unsupervised learning algorithms can uncover hidden structures, group similar instances together, or reduce the dimensionality of the data. Common techniques in unsupervised learning include clustering, where instances are grouped based on similarity, and dimensionality reduction, which aims to represent the data in a lower-dimensional space [55–57].

Semi-supervised learning mixes a bit of both worlds [58, 59]. It uses a large number of unlabelled data which are easy to collect but hard to classify and improves the overall accuracy by also using labeled data to build better classifiers [60].

Self-supervised learning can be considered a type of unsupervised learning as no explicit label is used. However, contrary to unsupervised learning that aims at finding patterns in the data, self-supervised learning creates pseudo-labels by solving a pretext task. Instead of relying on explicit human annotations, the model creates artificial labels from the input data itself and learns to predict or reconstruct the original data [61, 62].

Reinforcement learning is another type of algorithm that, similarly to unsupervised learning, does not rely on labeled data. However, contrary to unsupervised learning, its goal is not to find structure and similarities in the data, but the learning agent learns from its environment by maximizing a reward signal [63, 64]. It takes deep inspiration from the way a human learns

by interacting with its environment, as an action has an impact, and the agent learns from the consequence.

In this thesis, we will focus exclusively on supervised learning, and we will always consider this particular case in the following text.

1.1.2.2 Structure of a typical fully connected network and forward pass

The overall architecture is inspired by the brain, but neurons do not spike: they operate using continuous-valued activations to perform a non-linear transformation of their input. A typical fully connected network is composed of layers of neurons connected by synapses. The input signals encoding the training data are fed into the input neurons, and then they are processed layer by layer through the hidden layers until reaching the output layer. The activations of the neurons in each layer are computed based on the weighted sum of the activations from the previous layer, followed by the application of an activation function. Several non-linear functions can be chosen such as Rectified Linear Unit (ReLu, corresponding to the function $x \rightarrow max(0,x)$), sigmoids, tanh, and so on. The choice of the activation function depends on the specific requirements and characteristics of the task being addressed.

Synapses are connections between neurons. Positive-valued weights correspond to excitatory synapses, whereas negative-valued ones encode inhibitory synapses. The strengths of the synapses vary depending on the task at hand. Biases are additional parameters associated with each neuron. They introduce a shift or offset in the computation of the neuron's activation. Biases allow the network to have preferences for certain values and influence how easily a neuron gets activated or responds to different input patterns. Mathematically, biases are separate parameters added to the weighted sum of inputs before applying the activation function. They contribute to the network's flexibility in modeling complex relationships and capturing patterns in the data.

We will describe the network in a more formal way. Let us name pre-activation neurons as $a_k^l = \sum_j h_j^{l-1} W_{jk}^l$ where $h_j^{l-1} = f(a_j^{l-1})$ is the post-activation neuron, with activation function f. The weights are W_{ij} the biases b_i. Let us also call the inputs vector $x_i = a_i^0$ and the output as y_i. Let's name L the total number of layers, while l designates any layer between 0 and L.

We get the forward propagation or inference equation illustrated in Fig. 1.3a:

$$y_k = f(a_k^L) = f(\sum_j h_j^{L-1} W_{jk}^L + b_k^L) \tag{1.5}$$

1.1.2.3 Backpropagation algorithm

During training, the strengths of the synapses, i.e., the weights associated with the connections, are adjusted iteratively using the backpropagation algorithm. This algorithm calculates the gradients of the network's error with respect to the weights and updates the weights accordingly. The backpropagation algorithm enables the network to learn and adjust its parameters

to minimize the discrepancy between the predicted outputs and the desired outputs.

More precisely, this loss function is computed during the *forward pass*, or inference. The value computed gives an error, which is then backpropagated through the network. This constitutes the *backward pass*.

Let us consider a set of training examples X, with corresponding labels t. The parameters of the network, which includes both weights and biases, are represented as θ in a d-dimensional real space. A loss function allows the definition of how well the network performs by comparing output and label for specific examples. It is defined as $\mathscr{L}(y,t,\theta)$ where y is the output of the neural network. To evaluate the performance of the network on a specific set, an objective function $J(\theta)$ is defined by summing the loss function over all the examples present in the set. Learning, therefore, means minimizing this objective function. The way to train the network is by using gradient descent to minimize J.

Gradient descent is a method used to minimize any objective function by adjusting the parameters θ in the direction opposite to the gradient of that function (with respect to these parameters) symbolized as $\Delta_\theta J(\theta)$ [65]. The magnitude of the adjustments is controlled by a factor known as the learning rate, denoted as η. This factor influences the size of the steps taken towards a (local) minimum. Essentially, this process is akin to descending a hill by moving in the direction of the steepest slope until reaching a valley, or minimum point [66, 67]:

$$\theta \leftarrow \theta - \eta \Delta_\theta J(\theta). \tag{1.6}$$

The question is now how to compute $\Delta_\theta J(\theta)$. Backpropagation is an algorithm used to compute the gradient of the loss function with respect to the weights of the network. The name "backpropagation" comes from the fact that the computation starts at the output layer of the network and works its way backward, layer by layer, to the input layer. This is done by applying the chain rule of calculus to compute the derivatives.

Taking the same notations as the previous section, we get:

$$\frac{\partial \mathscr{L}}{\partial W_{ij}^{l+1}} = h_i^l \delta_j^{l+1} = h_i^l \left(e_j^{l+1} f'(a_j^{l+1}) \right) = h_i^l \left[\left(\sum_k \delta_k^{l+2} W_{jk}^l \right) f'(a_j^{l+1}) \right] \tag{1.7}$$

where $e_k = \sum_l \delta_l W_{lk}$, as shown in Fig. 1.3b.

Typical loss functions used include Mean Squared Error (MSE), which is commonly used in regression problems. MSE calculates the average squared difference between the predicted and actual values. Cross-Entropy Loss is mostly used in classification problems, especially in tasks where the output can be one of many classes. This function measures the dissimilarity between the predicted probability distribution and the actual distribution.

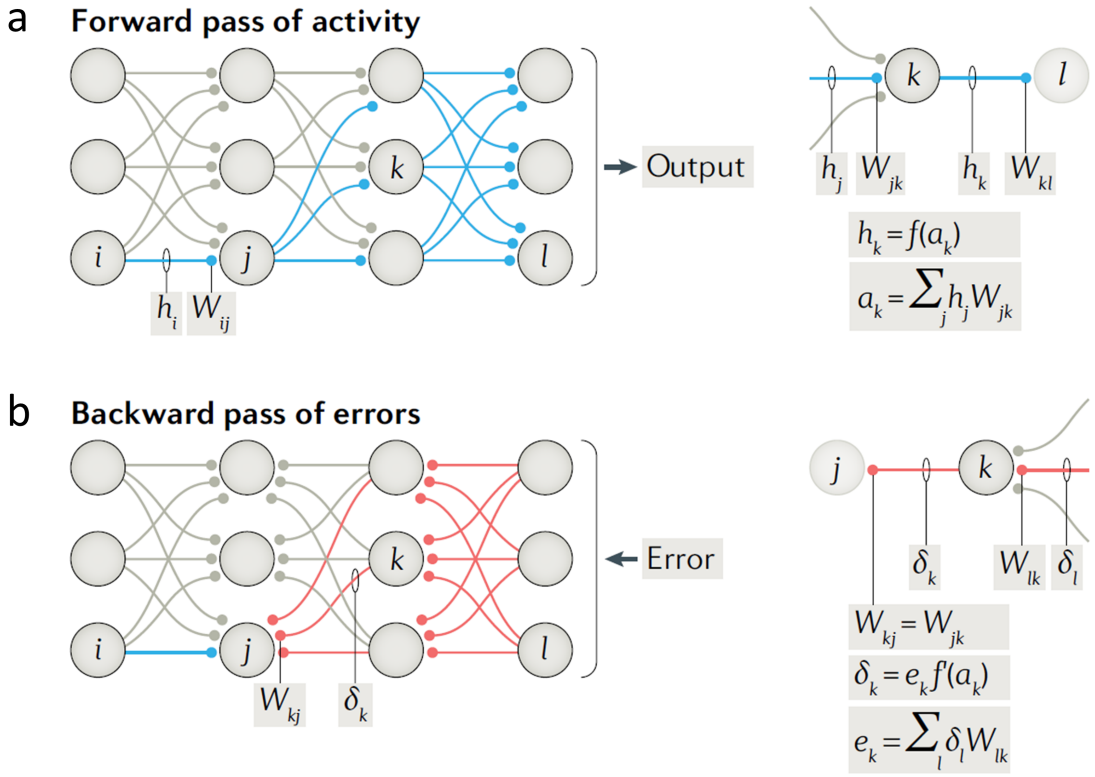


Figure 1.3: Above: Forward propagation. Below: Back Propagation from Ref. [2]

1.1.2.4 Training a network

The data used in the training process is divided into three distinct sets: the training set, the validation set, and the test set. The training set is the primary source for adjusting the weights and biases of the network through the forward and backward passes. The validation set serves a different purpose; it is used during the training phase to evaluate the model's performance and to fine-tune the hyperparameters. Hyperparameters are variables that are not learned from the training process itself but are set prior to training. These are often tuned using the validation set to find the values that produce the best model performance [68, 69]. Examples of hyperparameters (described below) include the number of epochs, batch size, and learning rate, among others. Finally, the test set is used post-training to provide an unbiased evaluation of the final model's performance.

The training process is organized into epochs, each representing a complete pass through the entire training dataset. During each epoch, the model's weights and biases are iteratively adjusted to minimize the loss function. Data is often divided into smaller subsets known as minibatches for each epoch. This approach, known as mini-batch gradient descent, strikes a balance between the computational efficiency of batch gradient descent (which uses the entire dataset) and the stochastic nature of stochastic gradient descent (which uses a single data point). The learning rate determines the size of the steps taken towards the loss (local) mini-

mum. Initialization is also a determining hyperparametric element to ensure that a minimum is reached [70].

Training neural networks involves navigating complex, high-dimensional loss surfaces to find optimal parameter values. While local minima can trap traditional optimization methods, leading to suboptimal solutions, previous work indicates that as the number of parameters in a neural network increases, the loss surface becomes less likely to have non-convex characteristics and local minima [71]. This suggests that larger networks, despite their complexity, may be easier to optimize due to the structure of their loss surfaces, which guide the optimization process towards many equivalent global minima.

However, the increased computational requirements and risk of overfitting necessitate careful model design and regularization techniques to ensure good performance. Overfitting is a common issue in neural network training [72]. It occurs when the model learns the training data too well, to the point where it captures not only the underlying patterns but also the noise or outliers in the data. As a result, the model performs well on the training data but poorly on unseen data, demonstrating a lack of generalization. To mitigate overfitting, we can employ techniques such as early stopping and regularization. Early stopping is a form of cross-validation approach where the training is halted as soon as the performance on the validation set stops improving, indicating the model might be starting to overfit the training data [73],[74]. This helps to ensure that the model generalizes well and does not simply memorize the training data. Regularization techniques are another key tool in preventing overfitting. These techniques add a penalty term to the loss function to constrain the complexity of the model. So-called L_1 and L_2 regularizations are common methods that penalize the absolute and square values of the weights, respectively. Another popular regularization technique is dropout [75], where random neurons are "dropped out" or deactivated during training, forcing the network to learn redundant representations and improving generalization.

1.1.3 The deep learning revolution

Other types of neural networks emerged in the 80s. The Neocognitron, introduced by Fukushima in 1982, provided a model for hierarchical, multilayered artificial neural networks [76]. However, it wasn't until 1989 that LeCun et al. proposed a way to train such networks using back-propagation, paving the way for the development of modern Convolutionnal Neural Networks (CNNs)[77]. In essence, a CNN processes an image through a series of hierarchical layers. The initial layers, typically composed of convolutional and ReLU layers, are designed to recognize simple, low-level features such as edges and textures. As the image progresses through the network, subsequent layers combine these low-level features to recognize more complex, high-level features, such as shapes or specific parts of objects.

Despite these early advancements, the development of CNNs was relatively slow for several decades. This was due in part to the lack of computational resources and data necessary for training these networks, as well as a general waning of interest in the field. However, with

the advent of more powerful computing systems and the availability of large-scale datasets, CNNs have experienced a resurgence and are now a cornerstone of modern machine learning and artificial intelligence research.

The onset of the 21st century marked a pivotal shift in AI, transitioning from conventional machine learning techniques to neural networks. This shift was catalyzed in 2006 when Geoffrey Hinton and his colleagues proposed a novel training methodology for neural networks, termed "deep learning". The launch of the ImageNet database in 2009, a large-scale repository of annotated images, significantly propelled advancements in computer vision. The efficacy of deep learning was underscored in 2012 when Hinton's team, employing their deep Convolutional Neural Network (CNN) known as AlexNet, clinched victory in the ImageNet competition [78]. The mid-2010s witnessed further groundbreaking developments, including the introduction of Generative Adversarial Networks (GANs) [79, 80] and the development of the Residual Network (ResNet) [81], a variant of CNNs designed for effective training of deep networks. Around the same time, Google's DeepMind demonstrated the potential of deep reinforcement learning with an algorithm that could play Atari games at superhuman levels [82]. This was followed by another significant achievement by DeepMind in 2016, the development of AlphaGo, a program capable of mastering the complex board game Go [83]. The latter half of the decade saw significant advancements in natural language processing. In 2017, the Transformer model was introduced [84], followed by Google's BERT in 2018 [85], both setting new performance benchmarks. In 2020, two major milestones were reached. OpenAI unveiled GPT-3, a highly advanced language model [86], and DeepMind introduced AlphaFold, a system that accurately predicts protein structures [87]. In 2023, a significant leap forward in the field of AI was marked by the introduction of GPT-4, which, according to Bubeck et al. [88], began to show the first sparks of general artificial intelligence. As we continue into the 21st century, AI research is evolving rapidly, focusing on areas such as self-supervised learning, large-scale multimodal models, and the ethical and societal implications of AI.

This rise has been accompanied by an exponential increase in the size and complexity of neural networks. Models like GPT-3 [86], with its 175 billion parameters, and Google PaLM [89], with 540 billion parameters, are emblematic of this trend. This growth in model size has been driven by the need to capture increasingly complex patterns in data and to improve performance on challenging tasks. However, training these large models requires vast amounts of data and computational resources [90], which has significant implications for energy consumption and environmental impact.

Different software or algorithmic practices can be used to reduce the number of parameters to train, and therefore reduce the energy consumption of the model. Some of these strategies are listed below.

Feature Selection: This involves selecting the most relevant features (input variables) to use in model training. Techniques for feature selection include filter methods (based on the correlation of each feature with the output), wrapper methods (which try different combinations of features), and embedded methods (which perform feature selection as part of the model training process) [91, 92].

Dimensionality Reduction: Techniques like Principal Component Analysis (PCA) [93] or t-Distributed Stochastic Neighbor Embedding (t-SNE) [94] can be used to reduce the number of dimensions in the data, which effectively reduces the number of parameters in the model.

Model Selection: Choosing a simpler model with fewer parameters can also help to reduce the number of parameters [95]. For example, a linear regression model has fewer parameters than a polynomial regression model.

Use of Pre-trained Models: In deep learning, it's common to use pre-trained models (like ResNet, or BERT) that have been trained on large datasets and then fine-tune them on a specific task. This allows the model to leverage the pre-trained parameters and only learn a small number of task-specific parameters [96].

Pruning: In the context of neural networks, pruning involves removing the connections (and thus parameters) that contribute the least to the model's predictions [97, 98]. This can significantly reduce the number of parameters in the model without a substantial loss in performance.

The goal of reducing parameters is to create a model that can generalize well to unseen data, and does not consume as much energy. It's a balance between making the model simple enough to not overfit the training data, while keeping it complex enough to capture the underlying patterns in the data.

1.1.4 The challenge of AI energy consumption

Artificial neural networks are now mostly trained on standard CMOS hardware. The architecture of classical computers typically follows the von Neumann paradigm which is characterized by a physical separation of the memory and processing units. Two main processing units are typically used in modern computers. A brief presentation of these two will be developed below.

1.1.4.1 CPU

The Central Processing Unit (CPU) is the computational core of a computer, responsible for executing instructions in the form of threads, which are sequences of operations. A simplified architecture of this unit is presented in Fig. 1.4a. When executing instructions, the CPU fetches instructions and data from memory through this bus, one instruction at a time. The fetched instructions are then decoded, executed, and the results are stored back in memory. However, CPUs face a few key challenges. One significant hurdle is managing latency, especially during memory access operations. This latency emerges from the time delay between requesting and

receiving data, slowing down overall system performance. Another challenge is heat management, given that a CPU's high operational speed generates substantial heat. Power consumption, particularly in mobile devices or servers, poses yet another challenge, as efficient energy use is crucial for prolonging battery life and reducing operational costs.

Addressing these challenges involves various strategies. To combat latency, CPUs employ hierarchical memory organization and sophisticated control flow methods. This involves using different types of memory, like volatile DRAM and SRAM for faster access and non-volatile memory for persistent storage. Cache memory, closest to the CPU, provides the fastest access times, helping mitigate latency issues.

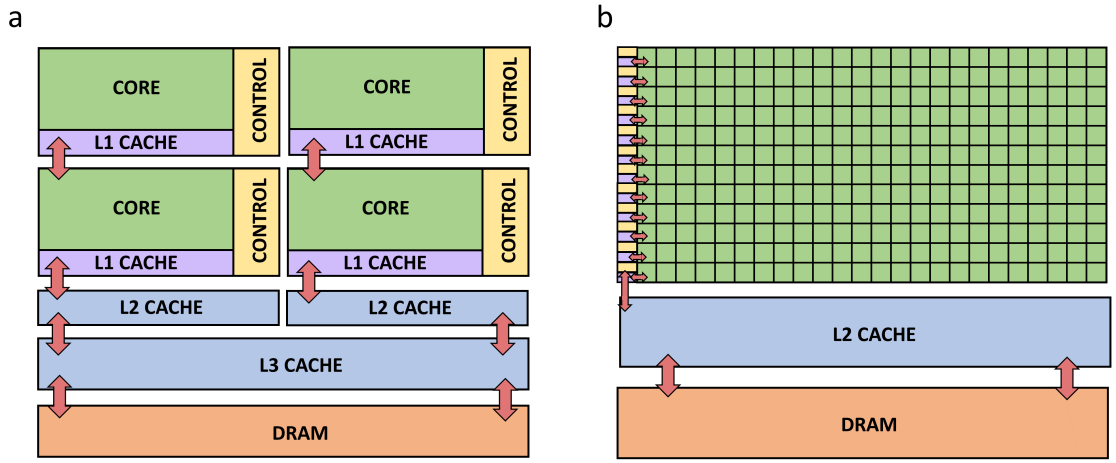


Figure 1.4: a: Simplified architecture of a CPU. b: Simplified architecture from a GPU. Both adapted from the NVIDIA documentation [3].

1.1.4.2 GPU

The Graphics Processing Unit (GPU), presented in Fig. 1.4b, is an essential component in modern computing systems, originally designed to accelerate the creation of images in frame buffers intended for output to a display. The objective was to improve the speed and efficiency of rendering two- and three-dimensional graphics, taking advantage of the inherently parallel nature of the graphics rendering process where each pixel can be computed independently. Around the mid-2000s, the potential of GPUs for general-purpose computing started to be explored. This transformation was catalyzed by the recognition that the same characteristics that make GPUs suitable for graphics rendering, namely high throughput and parallelism, can also be harnessed for tasks outside the realm of image rendering. One such field that has significantly benefited from this transition is machine learning, particularly the training of deep neu-

ral networks, a process that involves performing a vast number of mathematical operations that can be executed concurrently.

Compared to CPUs, GPUs are equipped with a greater number of cores, albeit simpler ones. While CPUs are designed for general-purpose computing and can handle a wide variety of tasks efficiently, GPUs are specifically built for performing a large number of similar computations simultaneously. This makes GPUs significantly faster than CPUs for tasks like matrix operations that are integral to neural network training. However, despite the significant advantages, there are still limitations to GPU usage. One key limitation is memory bandwidth. Even though GPUs have high-speed memory, the sheer amount of data processed in tasks like deep learning can result in a bottleneck. Another issue is latency. While individual operations are faster, the time taken to begin the operation can be longer, an aspect known as GPU kernel launch latency. Moreover, data transfer times between the GPU and the main system memory can also introduce delays.

1.1.4.3 Requirement for Computing Deep learning

Types of operations performed during learning The computational demands of deep learning are multifaceted, encompassing a variety of operations that require specialized hardware capabilities. At the heart of deep learning algorithms are matrix multiplications and vector additions, operations that are efficiently executed on Graphics Processing Units (GPUs) due to their inherent parallelism [99]. Central Processing Units (CPUs), on the other hand, are typically responsible for data loading and preprocessing, task scheduling, and handling input/output operations. The interplay between these different types of hardware is crucial for the efficient execution of deep learning tasks.

Memory access is another critical aspect of deep learning computation. The model's parameters, such as weights and biases, along with input data and intermediate computations, are stored in memory. The processor fetches these elements for computation, making efficient memory access patterns and caching mechanisms vital for reducing memory latency and maximizing data throughput. However, the rate at which data can be transferred between the processing unit and the memory unit, often referred to as the "memory wall," represents a fundamental limitation of modern computers [100]. In fact, for many computing tasks, the majority of energy consumption and time are attributed to data movement rather than to the computation itself [101].

Von Neumann bottleneck Most AI models rely on large datasets for training, which often surpass the memory capacity of even the most advanced GPUs. Furthermore, the size of the AI models can be so large that they exceed the available GPU memory, necessitating flash storage and clever scheduling of training batches to mitigate latency issues. However, these solutions do not address the high energy cost associated with constant data movement between memory and processors, a challenge often referred to as the von Neumann bottleneck, see Fig. 1.5a.

Each iteration of the training process involves reading the model parameters from memory to compute the gradient of the loss function, and then writing the updated parameters back to memory. This constant data transfer accounts for a significant portion of the total energy consumption of deep learning training, as shown in Fig. 1.5b.

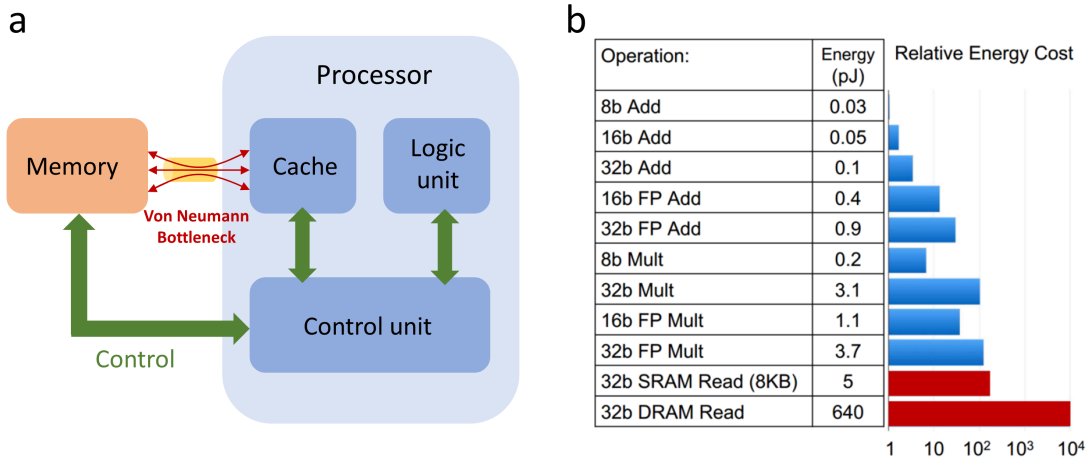


Figure 1.5: a: Von Neumann bottleneck. b: Energy per operation, adapted from [4].

In conclusion, the rise of AI and deep learning has brought forth significant challenges for traditional computing architectures. The high power consumption, scalability issues, and the inherent inefficiencies of the von Neumann architecture have become increasingly apparent. However, a promising alternative lies in Neuromorphic Computing, a concept introduced by Carver Mead [102]. Neuromorphic computing, inspired by the structure and function of the human brain, offers a potential solution to these challenges. These systems are designed to mimic the brain's efficient, adaptive, and speedy processing capabilities, making them well-suited for implementing complex neural network architectures and learning algorithms. Recent developments in neuromorphic hardware, such as IBM's TrueNorth [103] and Intel's Loihi 1 [104] and 2 [105], as well as the Spinnaker project [106], have demonstrated the potential of this approach, which we detail now.

1.2 Taking Inspiration from the brain to realize efficient hardware

The human brain, with its intricate structure and exceptional capabilities, provides a compelling blueprint for the development of artificial intelligence, computing, and efficient hardware. One of the brain standout features include the brain's proficiency in pattern recognition. It seamlessly identifies or classify patterns and establishes connections between diverse pieces of information. This capability underpins many of our daily tasks, from facial recogni-

tion to language comprehension and abstract problem-solving. Another key characteristic of the brain is its capacity to learn and adapt from experience, a process known as neuroplasticity. This involves the formation and reinforcement of new neural connections in response to new information, demonstrating the brain's dynamic and adaptable nature. In contrast to traditional computers that process tasks sequentially, the brain also exhibits a capacity for parallel processing, allowing for the simultaneous handling of multiple tasks [107]. Despite its complex structure and ability to handle tasks, the brain exemplifies energy efficiency, with a usage of approximately 20 W of power (accounting for about 20% of the body's energy, and 2% of the body's weight) [108]. Therefore, studying the brain to get an inkling about what makes it so efficient is a natural first step to building neuromorphic hardware.

1.2.1 The elements of the brain

The brain is composed of about 50 to 100 billion neurons and between 100 to 1,000 trillion synapses [109, 110]. Another type of cell is also present in the brain, the glial cells, in about the same number as neurons. These last cells will not be described in this thesis, but are thought to contribute to numerous processes such as synaptic strength influence [111]. Neurons and synapses will be described more in-depth in this section.

1.2.1.1 Neurons and synapses

Neurons Neurons, the key cellular units of the brain's nervous system, are central to its operation [112]. Each neuron has a body, known as the soma, which houses the cell nucleus, as shown in Fig. 1.6a. This nucleus oversees standard cellular activities, such as protein synthesis. Typically, a neuron has an input section and an output section. The input is usually managed by the dendrites, which are branching tree-like extensions sprouting from the soma receiving incoming signals. Conversely, the output is handled by the axon, a long projection that carries electrical signals away from the neuron. These signals, known as action potentials, exhibit a spiking behavior. The soma plays a crucial role in integrating the information received by the dendrites. If the soma experiences a high enough level of depolarization, it triggers action potentials, as shown in Fig. 1.6b.

A simplified version of how the spike happens is presented here. At rest, the neuron maintains a negative membrane potential (around -70 mV) [113], which is defined as the difference between the electric potential within a cell and its surroundings. Sodium channels and most potassium channels are closed, but some potassium channels are open, allowing a slow leak of potassium ions out of the cell. This helps maintain the resting potential. When the neuron receives a signal that brings the membrane potential to a certain threshold, voltage-gated sodium channels open. Sodium ions, which are in a higher concentration outside the cell, rush into the cell due to the electrochemical gradient. This influx of positive charges rapidly depolarizes the membrane, causing the membrane potential to become positive. This is the *depolarization*

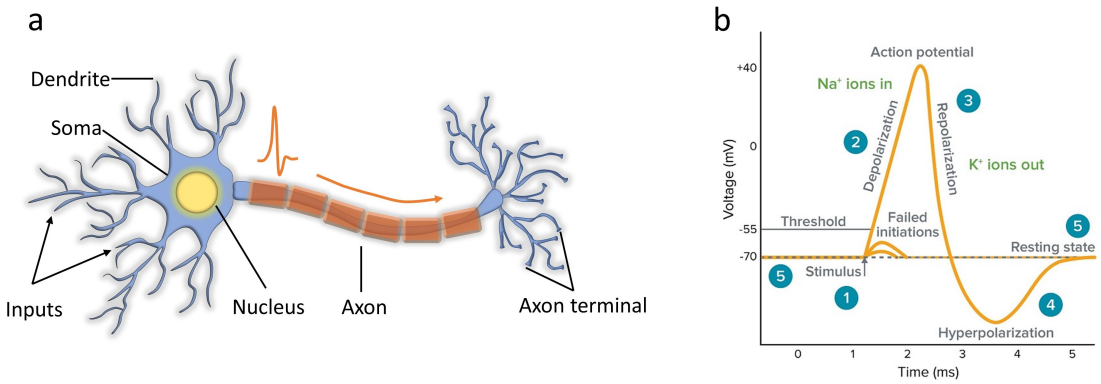


Figure 1.6: a: Simplified architecture of a neuron (credits to Jack Shuai Li). b: Shape of an action potential with the resting phase, followed by the depolarization, repolarization, and hyperpolarization phases (reproduced from Ref. [5]).

phase. After a brief delay, the voltage-gated sodium channels close, and voltage-gated potassium channels open. Potassium ions, which are in higher concentration inside the cell, rush out of the cell, again due to the electrochemical gradient. This efflux of positive charges repolarizes the membrane, causing the membrane potential to return to a negative value. This is the *repolarization* phase. The voltage-gated potassium channels close slowly, causing a brief period of *hyperpolarization* where the membrane potential is more negative than the resting potential. The membrane potential then returns to the resting state, aided by the activity of the sodium-potassium pump, which restores the original ion concentrations. After an action potential, the neuron enters a *refractory period* during which it is less likely to fire another action potential. This period allows the neuron to reset before it can generate another action potential. These action potentials are always of the same size and shape, regardless of the strength of the stimulus that triggered it. This is called the *all-or-nothing* behavior.

It is important to note that neurons can display a wide range of spiking behaviors, a topic we will delve into in the next section.

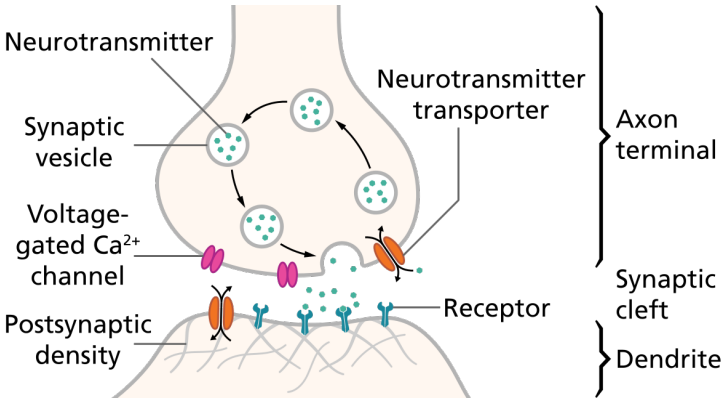


Figure 1.7: Synapse drawing reproduced from Ref. [6].

Synapses In the brain, information flows from one neuron to another through connections known as synapses, presented in Fig. 1.7 [114]. The journey of a signal begins with an action potential reaching the axon terminal of the presynaptic neuron. This event triggers the opening of voltage-gated calcium channels in the terminal's membrane, leading to an influx of calcium ions. These ions prompt synaptic vesicles, which are small sacs filled with neurotransmitters, to fuse with the axon terminal's membrane. This fusion releases neurotransmitters into the synaptic cleft, the small gap of only a few micrometers between the presynaptic and postsynaptic neurons [115]. The neurotransmitters then embark on a short journey across the synaptic cleft, eventually binding to specific receptors on the membrane of the postsynaptic neuron. This binding can cause ion channels on the postsynaptic membrane to open or close, altering the neuron's membrane potential. Depending on the type of neurotransmitter and receptor, this can either excite the postsynaptic neuron, making it more likely to fire an action potential, or inhibit it, making it less likely to fire. Finally, the neurotransmitters in the synaptic cleft are cleared away. They may be taken back up into the presynaptic neuron, a process known as reuptake, broken down by enzymes, or simply diffuse away. This ends the signal, resetting the synapse for the next wave of communication [116].

1.2.1.2 Neuronal behaviors observation and modeling

Neuronal behaviors The behavior of a neuron can be influenced by a multitude of factors such as its type, location within the brain, the nature of the stimulus it receives, and the presence of any modulatory signals. In this context, we will explore various neuronal behaviors, drawing on the work of Izhikevich [7].

Consider *tonic firing*, a behavior characterized by a steady, uninterrupted firing pattern devoid of distinct bursting behavior. Here, neurons generate action potentials at a consistent rate, typically in response to sustained depolarization (refer to Fig. 1.8 (A)). Contrastingly, *phasic spiking* involves a neuron firing a single spike at the onset of a stimulus, failing to fire again despite continued stimulation (Fig. 1.8 (B)). *Tonic bursting*, akin to tonic spiking, involves the neuron firing bursts of spikes at a regular frequency instead of individual spikes (Fig. 1.8 (C)). *Phasic bursting*, on the other hand, sees a neuron firing a burst of spikes at the stimulus onset, but not firing again despite continued stimulation (Fig. 1.8 (D)). In *mixed mode* behavior, a neuron alternates between firing individual spikes and bursts of spikes (Fig. 1.8 (E)). In *spike frequency adaptation*, the neuron initially fires spikes at a high frequency, which decreases over time despite a constant stimulus (Fig. 1.8 (F)). *Class 1 excitability* is characterized by a neuron firing spikes at a frequency that smoothly increases as the stimulus intensity rises (Fig. 1.8 (G)). In contrast, *class 2 excitability* sees the neuron firing spikes at a frequency that abruptly jumps to a high value as the stimulus intensity crosses a certain threshold (Fig. 1.8 (H)). In *spike latency* behavior, the neuron remains silent for a period after the stimulus onset, fires a single spike or burst, and then returns to silence (Fig. 1.8 (I)). *Subthreshold oscillations* refer to oscillations in a neuron's membrane potential that do not reach the threshold for triggering an

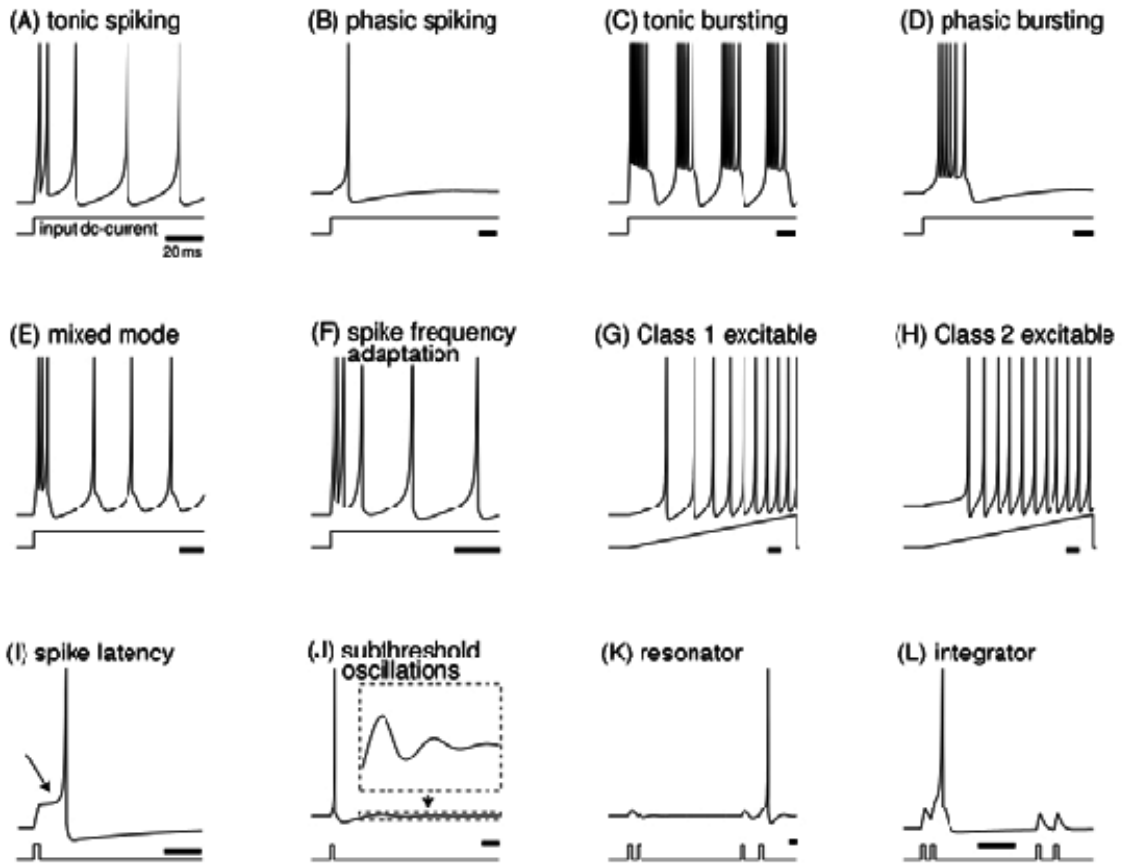


Figure 1.8: Twelve distinct firing patterns observed in individual neurons within the mammalian cortex [7]

action potential (Fig. 1.8 (J)). A *resonator* neuron responds more strongly to oscillatory inputs at certain frequencies (Fig. 1.8 (K)), while an *integrator* neuron responds equally to inputs at all frequencies, integrating the inputs and firing if these inputs are above a threshold (Fig. 1.8 (L)).

Lastly, *stochastic spiking* is a type of neuronal firing where the timing of action potentials is not strictly deterministic but has a random component, as shown in Fig. 1.9. This randomness can be attributed to various factors, including inherent noise in the biochemical processes involved in generating action potentials, the random arrival times of inputs to the neuron, and the complex, nonlinear dynamics of the neuron's membrane potential.

Neuronal models The development of accurate neuronal models holds significant importance in the fields of neuroscience and neuromorphic computing. These models serve as a fundamental tool for deciphering the intricate workings of neurons, elucidating how they process, transmit, and interact with information. This understanding can subsequently shed light on the complex orchestration of higher-level brain functions. Moreover, neuronal models act as a compass for scientific research. They assist in interpreting experimental data, providing a theoretical framework to make sense of observed results. In terms of practical applications,

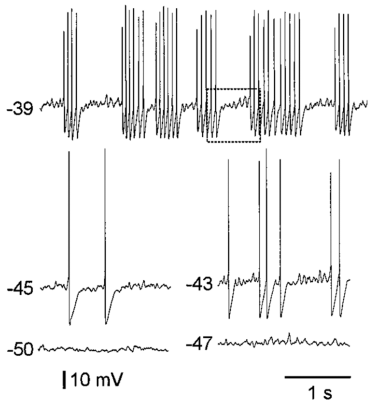


Figure 1.9: Stochastic firing in rodent trigeminal neurons [8]. Cells were progressively depolarized to potentials near and above the spike threshold (-45 mV). TIntermittent discharges occurred near the threshold. Stochastic bursting occurred when neurons were biased to suprathreshold potentials (-39 mV holding current, top trace).

neuronal models can predict neuronal responses under varying conditions or stimuli. This predictive power has far-reaching implications, from advancing our basic understanding of the brain to paving the way for novel treatments for neurological disorders.

Lastly, the importance of neuronal models extends to the development of neuromorphic systems. Robust and simple neuronal models are instrumental in refining the design and enhancing the performance of these systems.

Leaky Integrate-and-Fire The Leaky Integrate-and-Fire (LIF) model was first introduced by Lapicque in 1907, making it one of the earliest models of neuronal behavior [117]. Despite its simplicity, it captures the essential behavior of a neuron: the integration of inputs and the generation of action potentials, or spikes, when the membrane potential reaches a certain threshold. In the LIF model, a neuron is represented as an electrical circuit with a resistor R and a capacitor C in parallel. The resistor represents the leakiness of the neuron's membrane, which allows ions to flow in and out, while the capacitor represents the membrane's ability to store electrical charge. The model is described by a single differential equation, which represents the balance between the input I to the neuron, the leakage of charge through the membrane, and the change in the neuron's membrane potential u over time.

$$\frac{du}{dt} = \frac{1}{RC}\left(-\left[u(t) - u_{rest}\right] + RI(t)\right) \tag{1.8}$$

When the membrane potential u reaches a certain threshold u_{th}, the neuron is said to fire an action potential, and the membrane potential is then reset to a resting value u_{rest}.

In accordance with the Izhikevich and Hodgkin neuron classification system [7, 118], a Leaky Integrate-and-Fire (LIF) neuron typically falls within the category known as Class 1 neurons. Class 1 neurons are distinguished by their continuous membrane potential dynamics,

which do not involve spiking until a specific threshold is attained. Once this threshold is reached, these neurons generate a spike and subsequently reset their membrane potential.

The LIF model is computationally efficient and easy to analyze, making it a popular choice for large-scale simulations of neural networks. However, it is a highly simplified model that does not capture many aspects of real neuronal behavior, such as the detailed dynamics of action potentials or the effects of different types of ion channels. Despite these limitations, the LIF model provides a valuable tool for understanding the basic principles of neuronal computation.

Hodgkin-Huxley model The Hodgkin-Huxley model, developed in the 1950s, is a mathematical framework that describes how neurons generate and propagate electrical signals, or action potentials [119]. This model was groundbreaking in its representation of the neuron's membrane as an electrical circuit, with key components that mirror the biological structures of the neuron. It reproduces all the Izhikevich behaviors mentioned in this section.The model includes a capacitance, which symbolizes the neuron membrane's ability to store electrical charge. It also features resistors, which represent the ion channels that allow sodium and potassium ions to flow in and out of the neuron, which are responsible for the electrical signal. Additionally, batteries in the model represent the driving force for these ions to move across the membrane, known as the electrochemical gradient. The Hodgkin-Huxley model uses a set of four differential equations to describe the changes in the conductances of the sodium and potassium channels over time, and how these changes influence the membrane potential. These equations rely on several parameters, such as the maximum conductances of the ion channels and the membrane capacitance, which can be determined through experiments.

While the Hodgkin-Huxley model has been instrumental in our understanding of neuronal activity, it is also complex and computationally demanding. As a result, simpler models are often used for neural networks, such as the FitzHugh-Nagumo model which will be presented below.

FitzHugh-Nagumo model The FitzHugh-Nagumo model is a simplified mathematical model of neuronal activity that was developed as a more computationally tractable alternative to the Hodgkin-Huxley model. Introduced independently by FitzHugh in 1961 [120] and by Nagumo and his colleagues in 1962 [121], this model reduces the complex dynamics of a neuron to a two-dimensional system, making it easier to analyze and simulate. The FitzHugh-Nagumo model captures the essential features of action potential generation and propagation in neurons. It consists of two variables: one representing the membrane potential of the neuron, and the other representing a recovery variable, which accounts for the activation and inactivation of the ion channels that help generate the action potential. The model is described by a pair of nonlinear differential equations [122]. The first equation represents the fast dynamics of the membrane potential, including the rapid upstroke and downstroke of the action potential. The

second equation represents the slower dynamics of the recovery variable. While the FitzHugh-Nagumo model is much simpler than the Hodgkin-Huxley model, it still captures the key feature of neuronal excitability: the generation of action potentials in response to inputs above a certain threshold. This makes it a useful tool for studying the behavior of individual neurons and the dynamics of neural networks. The FitzHugh-Nagumo model can reproduce behaviors associated with Class 1 and Class 2 neurons in the Izhikevich classification system, depending on the parameters used. Class 1-like behaviors have been described in the LIF model section. Class 2 neurons are characterized by the repetitive firing of action potentials in response to a sustained input, as well as bursting behaviors or irregular spiking.

Learning Mechanisms in the Brain The process by which the brain learns is a complex and multifaceted phenomenon, and it remains an active area of research. Although the output amplitude of individual neurons is entirely digital due to their all-or-nothing behavior, most scientists believe that the brain encodes information through the precise timing and pattern of spikes across large groups of neurons. However, the exact learning algorithm employed by the brain is not yet fully understood. Recent studies in neuroscience have suggested that deep networks trained using back-propagation can more accurately represent the inferior temporal cortex found in actual neural tissues than models trained using other methods [123–125]. This suggests that the brain might employ some form of gradient-based learning. However, the mechanism by which the brain would execute this optimization remains unclear. This challenge is known as the 'credit assignment' problem, which involves determining how changes to hidden neurons should be made to drive the output neurons in the desired direction. This problem is particularly complex because each hidden neuron influences the output in a highly intricate manner [125].

Both backpropagation and BackPropagation Through Time (BPTT, standard algorithm for training RNNs) [126] offer a solution to this problem, but they do so in a way that is not considered biologically plausible for two main reasons. Firstly, an artificial neural network trained with backpropagation performs two types of computation: a forward pass that propagates neural activation, and a backward pass that propagates error vectors. This is problematic because the quantities propagated during the backward pass, denoted as δ, can be signed and potentially extreme-valued, either very small or very large. Moreover, the computing graph of the backward pass does not align with the typical model of neural computation, as the non-linear activation function is replaced by a linear element-wise product of point-wise derivatives of forward activations. This necessitates the storage of information about the forward pass. Secondly, the backpropagation of error gradients requires the use of the transposed version of the forward synaptic weights. This issue, known as the 'weight transport' problem, implies that an efficient hardware implementation of back-propagation would need to use the same physical devices to perform two different computations [127, 128]. Moreover, BackProp is non-local as each weight update takes into account the collective influence of the entire network's struc-

ture and connections. It necessitates the storing of information during both the forward and the backward passes rather than being solely determined by local information near the specific weight being updated, which is a hindrance to on-chip learning. In the following section, we will explore biologically plausible learning algorithms that could potentially be used by the brain.

1.2.2 Bio-plausible learning

The pursuit of understanding the brain's learning mechanisms unifies the fields of computational neuroscience and deep learning, despite their differing methodologies and approaches. Deep learning, heavily influenced by statistical learning theory, primarily employs rate-based neurons that perform continuous non-linear mappings of their inputs, often interpreted as firing rates. This approach has led to the development of powerful learning algorithms such as backpropagation, which has been instrumental in the success of deep learning. However, we have seen that backpropagation's reliance on symmetric weight matrices for forward and backward passes (known as the weight transport problem), its non-local nature, and the synchronous nature of its updates render it biologically implausible. On the other hand, computational neuroscience aims to model the brain's functions more faithfully. It often employs spiking neural networks, where neurons communicate through discrete spikes, more closely mimicking biological neurons. This field's focus is on understanding and replicating the brain's dynamics, which are event-driven and fundamentally different from the static architectures commonly used in deep learning [2]. 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 [129]. In this section, we introduce learning algorithms that draw inspiration from both the biological plausibility of computational neuroscience and the practical effectiveness of deep learning. We adopt a broad interpretation of bio-plausibility. An algorithm is considered more bio-plausible than BackPropagation if it meets one or more of the following criteria: it is local (weight updates depend only on information available to the neuron); it requires similar operations or circuitry for both inference and learning (making them suitable for implementation on neuromorphic hardware); or it allows for different forward and backward weights.

1.2.2.1 Rate-based algorithms

In this section, we first focus on rate-based biologically plausible algorithms.

Local learning rules but symmetric weights Introduced in 1985 Contrastive Hebbian Learning (CHL) is a learning algorithm that computes weight updates based on neural activations [130], and was initially used to train Boltzmann machines, which are probabilistic energy-based models and stochastic in nature. The weight update is computed as:

$$\Delta W_{ij} = \eta\left(\sigma_i^+\sigma_j^+ - \sigma_i^-\sigma_j^-\right), \tag{1.9}$$

where σ_i^+ and σ_i^- denote respectively the activations of neuron i in the positive and negative phases. During the negative phase, inputs are clamped but other neurons evolve freely according to their dynamics until an equilibrium point is reached. During the negative phase, output units are also clamped, but to the corresponding targets. The system evolves until a new equilibrium point is reached. The weight update can then be split into a negative part whose goal is to increase the energy of the pattern one wants to unlearn, and into a positive part whose goal is to decrease the energy of the correct pattern. CHL was adapted to deterministic networks [131–133] and is considered a more biologically plausible alternative to back-propagation as the update is computed only with one type of neural computation.

From then on, several algorithms related to CHL have been introduced, such as Recirculation [134] and General Recirculation [135]. In 2003, Xie et al. showed that backpropagation and CHL were equivalent in the case of feedback connections expect from a small scalar prefactor [136].

In 2015, Bengio and Fischer showed that when the output is slightly nudged in the second phase, the early change in neural activation corresponds to the propagation of error derivatives [137].

This idea is taken even further by Scellier and Bengio in 2017 with their learning algorithm Equilibrium Propagation (EqProp or EP) [138], EqProp is based on the principle of energy minimization. It operates in two phases: a free phase and a weakly-clamped phase. In the free phase, the network is left to reach a stable state or equilibrium where the total energy of the system is minimized. This is achieved by allowing the neurons in the network to update their states iteratively until the change in the states becomes negligibly small. The energy function is defined in terms of the states of the neurons and the weights of the connections between them. In the weakly-clamped phase, a small external force is applied to the output neurons to nudge the network towards the desired output. The network then adjusts its states to reach a new equilibrium. The change in the states of the neurons from the free phase to the weakly-clamped phase is used to update the weights of the network. The key advantage of EqProp is that it only requires local information for weight updates, making it more biologically plausible than backpropagation. A more detailed explanation of Equilibrium Propagation will be given in Chapter 3, as this algorithm will be at the heart of this PhD thesis.

Forward weights do not have to match backward weights for learning rules to be local Target Propagation (TP) is a learning algorithm proposed by Bengio in 2014 as a biologically plausible alternative to backpropagation for training deep neural networks, as the the weight updates are based on local neural activation, and no symmetry between the forward and backward connections is needed [139]. The key idea of target propagation is to define a "target" state for each layer in the network, and then update the weights in each layer to make the actual state of that

layer closer to its target state. This is done in a top-down manner, starting from the output layer and moving towards the input layer. However, to define the target state for each layer, we need to invert the operation performed by the layer. This can be challenging, especially when the number of neurons (i.e., the dimension) changes from one layer to the next. To overcome this challenge, target propagation introduces trainable feedback weights, which are used to learn approximate inverse functions for each layer. These inverse functions are then used to compute the target states. In the original version of target propagation, the target for the second-to-last layer is computed by backpropagating the global loss (i.e., the difference between the actual output of the network and the desired output). The targets for the remaining layers are computed by applying the approximate inverses to the already computed targets. Once all the targets are computed, the forward and backward weights are updated by minimizing layer-wise, local losses. The forward loss is the difference between the actual output of a layer and its target state, while the backward loss is related to the accuracy of the approximate inverse function.

However, the original target propagation algorithm's reliance on approximating inverse operations to compute the target states for each layer can lead to poor targets and, consequently, poor learning performance [139]. To mitigate this, Lee et al. introduced a variant of target propagation called "difference target propagation" [140]. The key idea is to add a linear correction term to the targets to compensate for the errors introduced by the imperfect inversion. With this modification, difference target propagation has been shown to successfully train multilayer perceptrons (MLPs) on the MNIST dataset, a popular benchmark for machine learning algorithms. Its performance closely matches that of backpropagation. However, the problem of imperfect inversions becomes especially noticeable in the final layer for classification tasks that involve a small number of classes, such as the 10 classes found in datasets like MNIST [141] and CIFAR-10 [142].

Feedback Alignment (FA) is a learning algorithm proposed by Lillicrap et al. in 2016 as an alternative to backpropagation for training artificial neural networks [143]. The key innovation of FA is that it eliminates the need for weight symmetry between the forward and backward passes and also uses a local update rule. FA uses a fixed, random feedback weights to propagate the error signal from the output layer to the hidden layers during the backward pass. Despite this randomness, the network is still able to learn useful representations and perform well on various tasks. The advantage of FA is that it only requires local information for weight updates, making it more biologically plausible than backpropagation. Another version of this algorithm, called Direct Feedback Alignment (DFA), was proposed by Nøkland in 2016 [144]. DFA, like FA, is designed to be a more biologically plausible alternative to backpropagation. DFA directly propagates the error signal from the output layer to all preceding layers, bypassing the need for sequential computation of the error signal. This direct propagation is achieved by using a fixed, random matrix for each layer, which is used to project the error signal from the output layer to that layer. Despite the randomness, the network is still able to learn and perform well

on various tasks. The advantage of DFA is that it simplifies the learning process by eliminating the need for sequential error propagation, making it more biologically plausible and potentially more efficient than backpropagation. However, as with FA, the performance of DFA can be less robust than backpropagation for deeper networks or more complex tasks, and understanding its strengths and limitations is an ongoing area of research.

Bartunov et al. demonstrated that feedback alignment and difference target propagation closely match back propagation on MNIST and CIFAR-10 on MLPs and 'locally connected' architectures, which are biologically more plausible convolutional layers without sharing the kernel weights across space [145]. However, these algorithms do not match back-propagation on ImageNet, indicating that their performance can be less robust than backpropagation for deeper networks or more complex tasks.

Eliminating the need for weight symmetry by having two forward passes In an invited talk presented at NeurIPS 2022 [146], Hinton introduced the "forward-forward algorithm" (FF), a new learning algorithm for artificial neural networks that draws inspiration from our understanding of neural activations in the brain. This algorithm aims to replace the forward and backward passes of backpropagation with two forward passes. These two passes are similar but work on different data and have opposite objectives. The "positive pass" operates on real data and adjusts the network's weights to increase a function called the "goodness" of each layer. The "negative pass" operates on negative data and adjusts the weights to reduce goodness. This process works well for a neural network with a single hidden layer. For a multi-layer deep learning model, the output of each hidden layer is normalized before being passed on to the next one. The FF algorithm proves that knowing precisely the non-linearities present in the forward computation is not necessary, which is more bio-plausible in that sense than back-propagation which requires an exact knowledge of the operation and their derivatives.

One of the main differences between the algorithms aforementionned and the functionning of the brain is first and foremost the use of continuous-valued neurons instead of spiking neurons. The following section will present spike-base learning algorithms.

1.2.2.2 Spike-based algorithms

In recent years, there has been a surge of interest in developing algorithms for training spiking neural networks, where the latency between spikes is not fixed but varies in a way that is computationally significant, much like the functioning of the brain. These algorithms are typically adaptations of gradient-based techniques that have been successful in training rate-based deep networks. They can be broadly divided into two categories [147]. The first category is spike-timing based representation, where the exact spiking times of neurons, which are real-valued, are optimized using gradient descent. The second category is activity-based representation, where the network's time step is discrete, akin to RNNs. This makes the spiking times

non-differentiable, and surrogate gradients are used for optimization [148, 149].

STDP The most well-known spike-based learning algorithm is Spike-Timing-Dependent Plasticity (STDP), believed to be a key mechanism for synaptic learning and adaptation. Introduced in the late 1990s and early 2000s through experimental neuroscience [150, 151], STDP is a form of Hebbian learning that takes into account the precise timing of spikes. In STDP, the change in synaptic weight depends on the relative timing of the pre-synaptic and post-synaptic spikes. If the pre-synaptic neuron fires just before the post-synaptic neuron (causal order), the synaptic weight is increased. Conversely, if the pre-synaptic neuron fires just after the post-synaptic neuron (anti-causal order), the synaptic weight is decreased. This rule is sometimes summarized by the phrase "fire together, wire together; fire out of sync, lose your link". STDP provides a biologically plausible learning rule for spiking neural networks, which are models of neural networks that aim to more closely mimic the behavior of biological neurons. However, translating STDP into an effective learning algorithm for artificial neural networks is a challenging task and an active area of research, as STDP has been primarily studied in the context of single-layer networks or small-scale multi-layer networks. It is not clear how to effectively use STDP to train deep networks, which have been shown to be highly effective for many deep learning tasks.

Alternatives to STDP Eligibility Propagation (e-prop) is a learning algorithm introduced by Bellec et al. in 2020 as a biologically plausible method for training recurrent neural networks (RNNs) [152]. The e-prop algorithm is inspired by the concept of eligibility traces in reinforcement learning and the spike-timing-dependent plasticity (STDP) observed in biological neurons. In e-prop, each synapse in the network maintains an eligibility trace, which is a record of the recent pre- and post-synaptic neuronal activity. The eligibility trace captures the information necessary for weight updates and is updated locally at each time step based on the current neuronal activities. The learning signal, which is equivalent to the error term in backpropagation, is a global signal that is broadcast to all neurons in the network. The weight updates are then computed as the product of the learning signal and the eligibility trace. This allows the network to perform credit assignments and learn temporal dependencies in the input data. The key advantage of e-prop is that it only requires local information for updating the eligibility traces and can be implemented efficiently in spiking neural networks. This makes it a promising candidate for implementation in neuromorphic hardware.

The EqSpike algorithm is a spike-based version of EqProp, introduced in 2021 by Martin et al. [153]. The neurons are leaky-integrate and fire neurons. The first layer receives a constant input current (the values of the pixels of the MNIST image to classify),all other neurons receive input which is the weighted sum of the spiking output of other neurons, and some constant current, either a bias or the nudging term (for the nudge phase). The learning rule is the following $dW_{ij} = \dot{\rho}_i \rho_j + \rho_i \dot{\rho}_i$, where ρ is the firing rate of the neuron. Here, the firing rate is either 0

if the neuron does not spike or 1 if it does. The derivative $\dot{\rho}$ has to be smoothed out with a low pass filter. The key advantage of EqSpike is that it performs all these computations locally in space, meaning that each neuron only needs information about its own state and the states of the neurons it's directly connected to. But it is also local in time, as no value needs to be stored during learning.

In 2021, Payeur introduced a variant of EqProp known as BurstProp [154]. This novel approach is grounded in the concept of burst multiplexing, where single spikes and high-frequency bursts carry different types of information, effectively creating two distinct communication channels within each neuron. This allows for the simultaneous transmission of feedforward data and feedback errors. BurstProp is designed to be biologically plausible, incorporating features such as dendritic compartments, short-term plasticity, inhibitory microcircuits, and burst-dependent plasticity. The authors demonstrated the effectiveness of a simplified version of their model by achieving competitive results on large-scale machine learning benchmarks, including ImageNet.

Conclusion of the section The brain, with its remarkable learning efficiency and low energy consumption, serves as a profound inspiration for the development of learning algorithms and energy-efficient hardware. It is this biological marvel that neuromorphic computing seeks to emulate, necessitating the optimization of both learning algorithms and hardware devices and circuitry. As we transition from understanding the brain's mechanisms and bio-plausible algorithms, we now turn our attention to the hardware aspect. The goal is to harness the principles that make the brain so efficient and translate them into practical, energy-efficient hardware designs for learning. This endeavor forms the next focus of our exploration, as we delve into the realm of neuromorphic computing.

1.3 Hardware adapted for AI

Definition and purpose Neuromorphic computing refers to the design and development of computational systems inspired by the structure, function, and efficiency of the biological brain. These systems aim to mimic the brain's ability to process and learn from information in a highly parallel and energy-efficient manner, offering promising solutions for a range of complex, real-world applications. Even if inference-only neuromorphic hardware is very common, a key goal of neuromorphic computing is the ability to perform both inference and training directly on the chip. Performing these operations on-chip, rather than transferring data back and forth between the chip and an external computer, can significantly increase speed, reduce energy consumption, and enhance privacy and security. The ability to perform training on-chip can be particularly important for edge AI applications, where some devices need to be able to learn and adapt in real time, often in resource-constrained environments. On-chip training allows these devices to learn from new data as it becomes available, without needing to send the

data to a central server for processing.

Currently, most neuromorphic hardware is based on CMOS (Complementary Metal-Oxide-Semiconductor) technology, which has been the industry standard for several decades. The "More than Moore" paradigm recognizes that continuing the miniaturization of transistors (as predicted by Moore's Law [155]) is not sufficient to meet the growing demands for computational power and energy efficiency. It suggests that we need to explore new types of devices, materials, and architectures. However, as we push the boundaries of what is possible with neuromorphic computing, there is a growing need to explore beyond CMOS and investigate emerging devices and technologies [12]. Indeed, CMOS technology is ill-suited for neuromorphic computing due to its digital nature, limited parallelism, absence of non-volatile memory, lack of fault tolerance, and constraints on custom, compact designs required for neuromorphic systems.

1.3.1 Emerging devices

1.3.1.1 What makes a good hardware synapse or a good hardware neuron candidate?

In the field of neuromorphic computing, the design of hardware neurons and synapses is guided by several key principles. Firstly, low power consumption is paramount, as one of the primary advantages of neuromorphic computing is its potential for energy efficiency. This means that an ideal hardware candidate should consume minimal power, not only during operation but also in standby mode. Secondly, scalability is crucial. The hardware should be designed in such a way that it can be manufactured at small scales, while also allowing for easy integration into larger systems. This scalability is essential for supporting large networks of neurons and synapses. Thirdly, reliability and durability are important considerations. The hardware should be robust, capable of withstanding a wide range of operating conditions and maintaining its performance over time. Compatibility with CMOS technology is another key factor, as this allows for leveraging existing manufacturing infrastructure and integrating with other electronic components. Lastly, the hardware should be capable of operating at sufficient speeds to support real-time processing requirements.

In the pursuit of designing **artificial synapses** for neuromorphic computing, whether in digital or analog frameworks, several key requirements emerge. These requirements are inspired by the functional characteristics of biological synapses and the practical constraints of hardware implementation, forming a bridge between the worlds of neuroscience and engineering. Biological synapses are characterized by their capacity at adapting their strength depending on the pre and post-synaptic neurons, a feature that allows for complex and nuanced network behaviors. This is mirrored in the design of artificial synapses, which should be capable of evolving between multi-level states under external stimuli, see Fig. 1.10b and c.

This characteristic is a reflection of synaptic plasticity, a key feature of biological learning and memory. Hardware analog synapses, which can be used for realizing any kind of ANNs, can have many different states [156], and the more the better. Hardware binary synapses only have two states, and can only be used for realizing Binarized Neural Networks (BNN) [157]. Non-volatility, the ability of a synapse to retain its state even when power is not supplied, is another crucial feature [158, 159], as illustrated in Fig. 1.10g. This mirrors the long-term memory storage capability of biological systems and is particularly important for certain applications where power supply may be intermittent or constrained. The control of the synaptic device with an external parameter is essential for the precise modulation of synaptic strength during learning. This control should ideally exhibit a linear relationship with the synaptic weight as presented in Fig. 1.10d, allowing for precise changes during learning, much like the precise modulation of synaptic strength that occurs during biological learning [160].

Hardware neurons are expected to reproduce the most neuronal behaviors to allow the network to have the complexity necessary to mimic the brain.

For both hardware neurons and synapses, having simple mathematical models that appropriately describe the behaviors of the physical components is crucial. Indeed, being able to predict the behaviors when integrating into an array or when subject to different conditions of voltage, current, but also of temperature is an important step into the development of efficient neuromorphic hardware.

1.3.1.2 Memristor synapses and neurons

A memristor, a term derived from "memory resistor", is a type of passive circuit element that maintains a relationship between the time integrals of current and voltage across a two-terminal element. Chua first postulated it in 1971 based on symmetry arguments in the relationships between fundamental circuit variables (here electric charge and magnetic flux) [161]. The key characteristic of a memristor is that its resistance can be adjusted and that it "remembers" this resistance even when power is turned off. However, it was not until 2008 that a team from HP Labs led by Williams reported the development of a switching memristor based on a thin film of titanium dioxide [162]. Critics opposed that this was not the fundamental circuit element described by Chua, but a device with a particular non-linear current-voltage characteristic. However, a broader definition of memristive devices is usually used, which encompasses all two-terminal devices showing a pinched hysteresis loop in the I-V plane, and can be either non-volatile or volatile [163, 164]. This definition allows for numerous different families of devices to be classified as memristive devices [165].

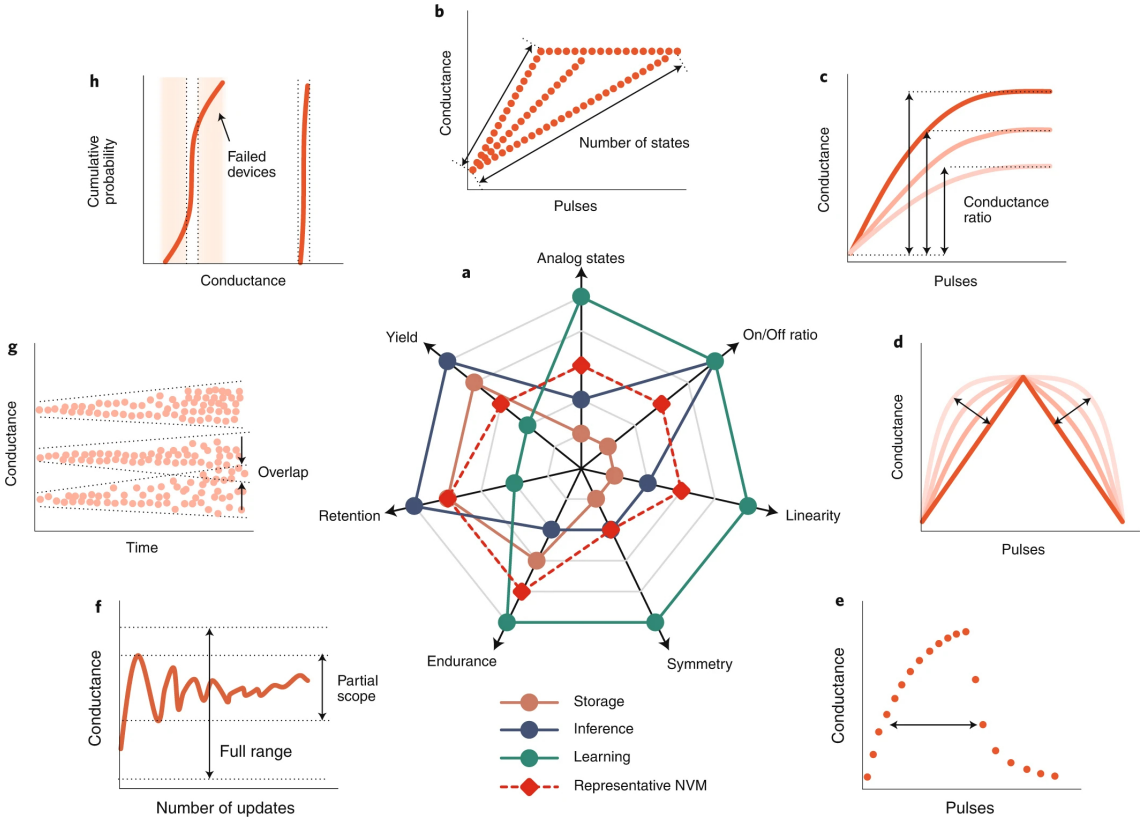


Figure 1.10: a: A comparison of the qualitative device requirements for three potential applications. The red line indicates experimental NVM data from previous studies. b–h: Conceptual representations of device requirements for computing: analog states (b), on/off ratio (c), linearity (d), symmetry (e), endurance (f), retention (g), and yield (h). The dashed and solid curves in b–e show the conductance adjustment of an analog NVM device. The conductance modifications of an NVM device during the training process typically occur within a partial scope rather than across the full range of the conductance window (f). After NVM devices are adjusted to various conductance levels, the conductance of the devices can vary over time, potentially leading to overlap between two levels (g). NVM devices that fail to reach the target conductance level are considered unsuccessful (h).Reproduced from Ref. [9]

Resistive memristors Before the proclaimed discovery of the resistive memristor, resistive materials had already been widely used in Resistive Random Access Memories (ReRAM). Today ReRAM and "nonvolatile memristors" are used as synonyms in most of the literature. Their storage function is based on a physical mechanism called resistive switching, which is responsible for the transition from a High Resistive State (HRS, or 'OFF' state) to a Low Resistive State (LRS or 'ON' state) under the application of a specific voltage value [165]. The transition from HRS to LRS is called the SET process when, whereas the transition from LRS to HRS is named RESET. Usually, a compliance current I_{cc} is applied during the SET process to the device to avoid excessive current which would cause an irreversible hard breakdown [166].

Two different switching modes exist: the bipolar and the unipolar mode. Unipolar switching characterizes a switching that is independent of the voltage or current polarity. In contrast, for bipolar switching, two different polarities are required for the SET and the RESET processes. The two different cases are presented in Fig. 1.11.

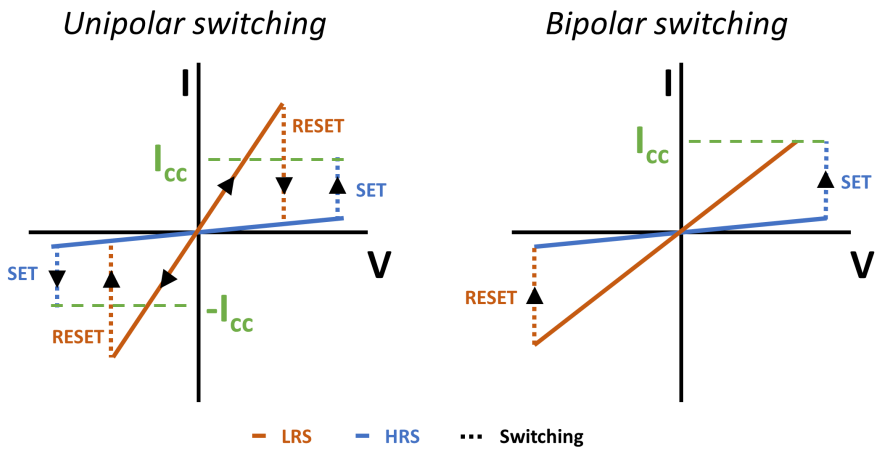


Figure 1.11: Adapted from [10]

Most memristors require an electro-forming process, where a high current or a high voltage is first applied to the pristine HRS material in order to initiate the first LRS state (a compliance current is most often used during this process as well). This can be an issue for integration. That is why a part of the research focuses on designing forming-free devices.

Two different types of memristors exist: the original non-volatile memristors which are good candidates for synapses and volatile memristors which are good candidates for neurons. Their typical I-V characteristics are shown in Fig. 1.12.

Four major types of resistive devices for which the switching mechanism is due to redox reactions, nanionic transport process or insulator-metal transitions [167, 168].

Electrochemical metallization ECM bipolar switching mechanism is based on the electrochemical dissolution and deposition of an active electrode metal. Typically, an ECM cell,

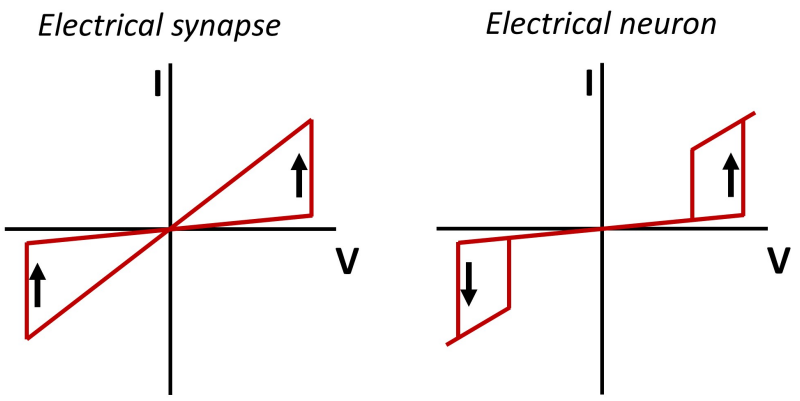


Figure 1.12: Left: Typical I-V characteristic of a resistive switching non-volatile memristor. Right: Typical I-V characteristic of a voltage-controlled threshold switching (TS) volatile memristor. Adapted from [11]

also called conductive bridging random access memory (CBRAM), is made of an active electrode of metal M such as Ag, Cu, or Ni, an electrochemically inert electrode such as Pt, with a thin film of solid electrolyte in between, which is an M^{z+} ion conductor [169]. In the initial HRS, no electrodeposit of the metal M is present on the inert electrode. When sufficiently high positive voltage V is applied to the active electrode, the metal M of the active electrode undergoes an anodic dissolution, giving rise to metal cations in the solid electrolyte thin film. Under a high electric field, these cations drift across the film, reaching the inert electrode. There, they undergo a reduction and electro-crystallization on the surface, therefore, a metallic filament connects both electrodes, leading to a LRS. Under sufficient a negative voltage, the electrochemical dissolution of the metallic filament leads to a HRS.

ECM-based spiking neurons also exist [170, 171].

Thermochemical In the context of resistive memories, a unipolar thermochemical mechanism, or thermochemical memory effect (TCM), is a process that leverages heat to control the state of the memory [172]. This mechanism is also often referred to as a fuse-antifuse mechanism [173]. The structure required to achieve thermochemical switching is symmetric, involving two inert electrode contrary to VCM. However, the same materials and systems can be used, simply by rendering the structure symmetric, with two inert electrodes. The forming and SET mechanisms are identical in both cases, with the creation of a filament of oxygen vacancies. However, the RESET process differs. Here, the current in the 'on' state increases to a point where it generates enough heat to locally destroy the filament. This effectively 'resets' the memory, returning it to its initial HRS state. This switching is therefore unipolar, and a typical I-V characteristic can be found in Fig. 1.11 However, this process often involves high reset currents and has a limited endurance, typically ranging from a few tens to a few hundred cycles. As a result, the focus in recent years has shifted towards valence change-based concepts, which

tend to offer better performance and longevity [174], and which will be presented below.

Mott memristors Mott memristors leverage this Mott transition to achieve their memristive behavior. By applying a voltage across the device, it is possible to induce a Mott transition and change the device's resistance state. This change in resistance can then be read out as a change in the device's conductance, allowing it to be used as a spiking neuron when a capacitance is put in parallel. One of the most commonly studied materials for Mott memristors is vanadium dioxide (VO_2) [175, 176]. VO2 undergoes a Mott transition near room temperature, which makes it a convenient material for these devices. By applying a voltage across a thin film of VO_2, it's possible to induce a Mott transition and change the film's resistance state [177]. Another type of volatile memristor used as a neuron is based on $Nb0_x$, and is sometimes classified as Mott memristor [178]. However the physical mechanism is debated but most believe it is not due to an insulator-metal transition, but caused by an increase in the oxide electrical conductivity due to local Joule heating [179].

Valence change memristors Valence Change Mechanism (VCM) memristors, a type of resistive memory, typically employ a Metal-Insulator-Metal (MIM) structure where the insulator, often a dielectric, serves as the switching layer. These devices can utilize a variety of oxide materials such as titanium oxide (TiO2), hafnium oxide (HfO2), or aluminum oxide (Al2O3) [180]. Most VCM memristors are filamentary, as they form an oxygen vacancy conductive filament in the dielectric layer to switch between high and low resistance states, as shown in Fig. 1.13a with the corresponding I-V characteristic in Fig. 1.13b. An electroforming process is usually needed to initiate the oxygen ion migration. The switching is bipolar(a typical I-V characteristic can be found in Fig. 1.11) and attributed to redox reactions and nanoionic transport [181]. The structure for VCM needs to have a built-in asymmetry with respect to the oxygen concentration in the film, typically with an inert electrode on one side and an active electrode on the other [174]. The electron transport in a MIM structure takes place via a series of mechanisms. Some of these conduction mechanisms rely on the electrical property at the electrode-dielectric interface, such as the energy barrier height of the interface and conduction carriers in dielectric films. Others depend on the properties of the dielectrics itself, such as Poole-Frenkel (P-F) emission; Ohmic conduction; ionic conduction; hopping conduction; and trap-assisted tunneling (TAT)[166].

One of the challenges of these filamentary ReRAM devices is that the change in resistance is often non-linear, which can make it difficult to precisely control the resistance state of the device. This non-linearity can be a significant issue for applications like neuromorphic computing, where the ability to finely tune the resistance state of a memristor is crucial for mimicking the behavior of biological synapses. One of the main axes of research is then to choose adequate materials to obtain the most linear synapse possible. For example, the study by Chandrasekaran et al. [182] suggests that introducing aluminum (Al) into HfO2 can improve the lin-

earity of the resistance change in these devices. The addition of Al can modify the distribution and movement of oxygen vacancies within the HfO2, leading to a more linear and controllable change in resistance when a voltage is applied.

These devices are the most commonly used for neuromorphic synapses in the literature. Different models exist for reproducing the resistive switching of these devices [183], [184]. The ability to simulate an algorithm or an electronic circuit using these models serves as an invaluable tool for predicting system performance. By accurately modeling the resistive switching, researchers and engineers can anticipate how these devices will behave in different scenarios, enabling them to optimize their designs and algorithms before physical implementation.

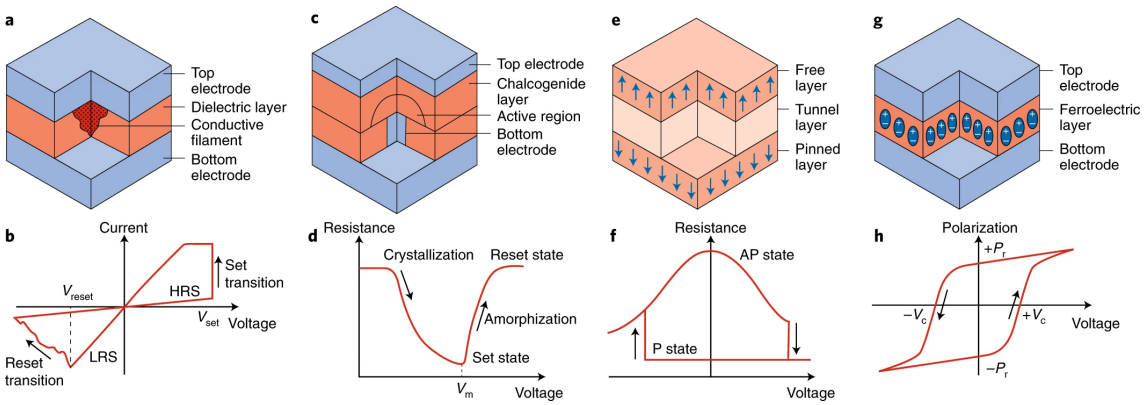


Figure 1.13: Different memory devices. a: Filamentary resistive switching RAM structure. b: Corresponding current–voltage characteristic of a bipolar RRAM switching device. c: Phase change memory structure. d: Corresponding resistance-voltage characteristic. e: Magnetic tunnel junction (MTJ) structure. f: Corresponding resistance–voltage characteristic of an STT-MRAM. g: Ferroelectric random access memory (FeRAM) structure. h: Corresponding polarization–voltage hysteretic characteristic (h). The orientation of electrical dipoles causes permanent polarization of the ferroelectric layer. From [12]

On the most used material to relize VCM memristor synapses is HfO_x. This material can however also be sed to implement volatile spiking neurons [185, 186].

1.3.1.3 Other emerging devices

Spintronic devices To create spintronic synapses, magnetic tunneling junctions (MTJs), which are the basic cells of MRAMs, are utilized as fundamental building blocks. An MTJ consists of two metallic ferromagnetic layers separated by a tunnel barrier, as depicted in Fig. 1.13e. The resistance of the MTJ depends on the relative orientation of the magnetization directions of the two ferromagnetic layers. The pinned layer (PL) , has a fixed spin polarization direction. In contrast, the free layer (FL) can alter its magnetization direction. This dynamic magnetization change in the FL can be achieved through current injection. When the magnetization directions of the two layers are in parallel alignment (P state), there is a higher probability of electron

tunneling through the barrier, resulting in a lower resistance state (LRS), as presented in Fig. 1.13f. Conversely, when the magnetization directions are in antiparallel alignment (AP state), there is a lower probability of electron tunneling, leading to a higher resistance state (HRS) [187]. In addition to the spin-transfer torque (STT)-MTJs, spin-orbit torque (SOT) cells have emerged as an alternative for achieving magnetization switching in spintronic synapses. Bipolar SOT-induced magnetization switching is realized under an in-plane magnetic field collinear with an applied current within multilayers of ferromagnetic (FM) and nonmagnetic (NM) metals. The spin Hall effect (SHE) and Rashba effects, originating from spin-orbit coupling within the NM layer and at the FM-NM interfaces, respectively, play a crucial role in the switching mechanism. All of these synapses are binary in nature, which can be an issue for encoding analog weights in ANNs. This has led to an increased interest in other types of spintronic synapses [188], such as spintronic memristors [189], operating based on the displacement of a magnetic domain wall [190] in a spin-valve, resulting in lower or higher resistance states depending on the domain wall position. This memristive functionality through domain wall motion in magnetic tunnel junctions has been experimentally demonstrated by Chanthbouala et al.[191] and Lequeux et al. [192].

Spintronic neurons in neuromorphic computing can be realized through different techniques. Spin-torque nano-oscillators are specific magnetic tunnel junctions driven into spontaneous microwave oscillations by injecting current [193–195]. They exhibit memory, stable behavior, and nonlinearity, and can synchronize with other oscillators. Superparamagnetic tunnel junctions, with low-energy barriers, emulate stochastic behavior and serve as low-energy artificial neurons [196]. Magnetic solitons like domain walls and skyrmions can be manipulated to emulate leaky integrate-and-fire neurons [188].

FeRAM Ferroelectric materials, discovered nearly a century ago, have recently re-emerged as a promising candidate for neuromorphic computing applications. Initially used in ferroelectric random-access memories (FeRAMs), these materials exhibit spontaneous electric polarization that can be reversed by an external electric field, leading to changes in conductivity as shown in Figs. 1.13g and h. However, early FeRAMs faced scalability issues due to the large thickness of the ferroelectric layer. Recent advancements in material fabrication technology have enabled the production of nanometer-thin ferroelectric layers, reigniting interest in ferroelectric devices [187]. New ferroelectric materials such as strained SrTiO3 and AlScN have been discovered, and the observation of ferroelectric properties in doped HfO2 has further expanded the range of potential applications [197]. For instance, ferroelectric tunnel junctions (FTJs) and ferroelectric field-effect transistors (FeFETs) have been explored for their potential in neuromorphic computing. FTJs, characterized by non-volatility, analog switching capability, high endurance, energy efficiency, and scalability, are particularly desirable for neuromorphic applications [198]. FeFETs, based on ferroelectric HfO2, are also gaining attention due to their low-voltage and fast switching, good data retention, and compatibility with CMOS fabrication.

The non-volatile memory operation of FeFETs relies on the two stable polarization configurations in HfO2, corresponding to high and low conduction states, making them suitable for use as artificial synapses [198].

Furthermore, FeFETs can also serve as artificial neurons, emulating real neurons at various abstraction levels, from complex biophysical models to basic integrate-and-fire circuits [198].

PCM Phase Change Memory (PCM) is a type of non-volatile random-access memory that exploits the unique behavior of chalcogenide glass, a material that can undergo a unipolar switching mechanism between two states through Joule heating: amorphous and crystalline, presented in Figs. 1.13c and d. [199]. The most common phase change material is an alloy of germanium, antimony, and tellurium (Ge2Sb2Te5 or GST). In its amorphous (disordered) state, GST has high electrical resistance, while in its crystalline (ordered) state, it has low resistance. By applying a voltage, the material can be heated and rapidly cooled to transition between these states. The key advantages of PCM include its fast read and write times, high endurance, and excellent scalability. It also retains data even when power is turned off, making it a type of non-volatile memory. PCM is considered a promising technology for future memory and storage applications, and it has also been explored for use in neuromorphic computing due to its ability to support multiple resistance levels, which can be used to emulate the synaptic weights in a neural network. However, PCM also has some challenges. Its susceptibility to rapid crystallization can lead to an abrupt drop in resistance [200]. The high programming current required to change the phase of the material can lead to high power consumption. The resistance drift over time can also affect the reliability of the stored data.

An intriguing development in the field of phase change memory (PCM) based neural devices was made by Tuma et al.[201]. They constructed a device by placing a nanometer-scale layer of $Ge_2Sb_2Te_5$ (a common material used in PCM) between two electrodes [200].

1.3.2 Integrating emerging memory devices in hardware

In neural networks, one of the fundamental computations is the multiply and accumulate operation (MAC), or more generally, matrix-vector multiplication. This operation lies at the heart of propagating information through the layers of a neural network. The MAC operation involves multiplying input data by associated weights and summing the results. When implementing these computations in hardware, particularly for large-scale neural networks, efficiency, and speed become crucial factors.

In an effort to address these needs, certain architectures turn to the laws of physics, specifically Ohm's Law and Kirchhoff's Current Law. By expressing weights as conductances in a well-arranged architecture, the product of each weight and its associated input can be calculated simultaneously by measuring the current through each path. In addition, Kirchhoff's Current Law states that the sum of currents entering a node must equal the sum leaving it, as shown in Fig. 1.15a. This law can be exploited to accumulate the results from multiple paths, thus

completing the summation part of the MAC operation. By taking advantage of these physical laws, the MAC operation, which constitutes the core computation of a neural network, can be performed in an analog and massively parallel manner.

Resistances and conductances are positive values, whereas to perform neural network training (see Fig. 1.15b), we need both positive and negative weights. A common way is to use two different devices to implement the positive and negative parts of the weight and subtract both currents to get the final output, as explained in Fig. 1.15c.

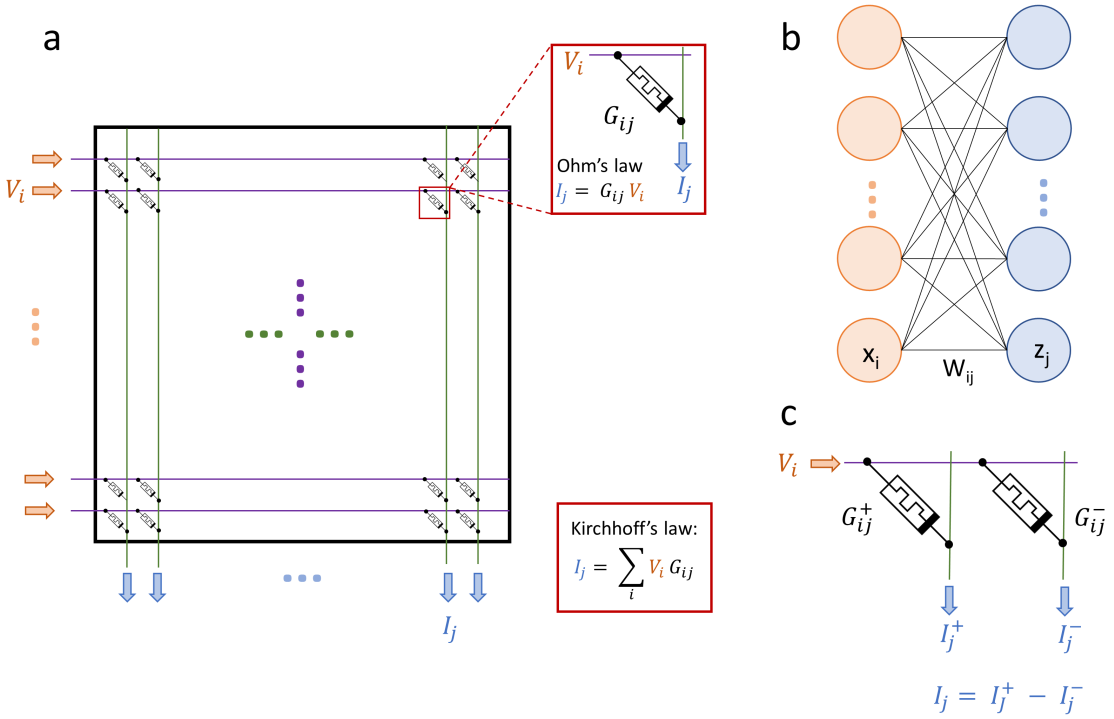


Figure 1.14: a: 1R architecture with memristor devices. b: Corresponding neural network, with in orange the input (corresponding to the voltages V), and in blue the output, corresponding to the current I.

1R architecture The 1R memristor crossbar array is a classic architecture used in neuromorphic computing and ReRAM systems. Each memory cell in the array consists of a single memristor located at the intersection of wordlines and bitlines. In this architecture, the wordlines act as the inputs, while the bitlines serve as the outputs. When performing read or write operations on this 1R crossbar array, a voltage is applied across a selected wordline and bitline, targeting the cell at their intersection, the so-called full-selected cell. However, due to the interconnected nature of the array, other cells sharing the same wordline or bitline— the half-selected cells— also experience a partial voltage when programming one cell. This can result in an undesired leakage of current, or 'sneak current', which can cause errors in the operation of the memory ar-

ray. Mitigating this challenge is key to optimizing the design and performance of 1R memristor crossbar arrays. A common strategy for reducing sneak current during programming operations involves biasing non-selected wordlines and bitlines at V/2, or half the full voltage. This restricts the voltage drop across the half-selected cells, which in turn, limits the likelihood of unwanted current leakage [202].

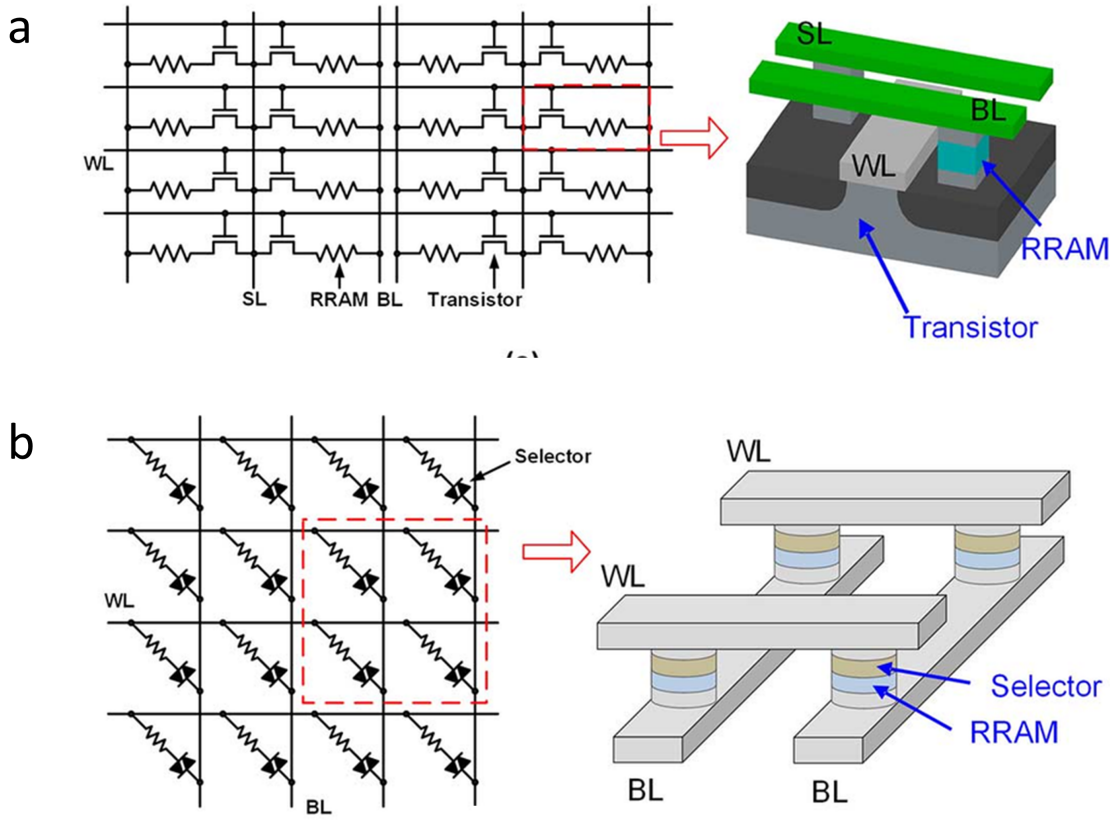


Figure 1.15: a: 1T1R architecture with memristor devices. b: 1S1R architecture with memristor devices. Adapted from Ref [13].

1T1R architecture One easier approach to implementing large-scale neural networks in hardware involves the use of 1T1R cells, a combination of a memristor (1R) and a MOS transistor (1T). Each transistor acts as a controllable switch. With the transistor in series with each memristor, the current flow through unselected cells is effectively suppressed. This ensures accurate reading and programming of memristors, an advantage that helps address some of the challenges associated with pure memristor architectures. The transistor's gate allows for additional control, facilitating the linear and symmetric updating of synaptic weights during the training of neural networks. Furthermore, the ability to control each gate voltage individually enables semi-parallel programming of the array, thereby improving the efficiency of the training process. During the execution of the neural network's computations, all transistors in the array

are turned ON to minimize the influence of channel resistance. Meanwhile, during weight updating or training, the transistors are partially turned ON, which provides precise control over weight adjustments [203].

However, this architecture presents its own challenges. The integration of transistors increases circuit complexity and area, affecting the array's packing density and potentially raising manufacturing costs. Additionally, there is an energy overhead associated with the transistors due to their gate capacitance, which results in increased power consumption, especially during the training phase. Careful selection of transistor types—balancing between depletion-mode transistors favored for inference and the increased power consumption during training—thus becomes a critical aspect of optimizing these systems.

1S1R architecture In the quest for a more compact and effective solution to the sneak path problem in memristor crossbar arrays, the 1S1R (One Selector-One Resistor) architecture presents a promising approach. This architecture employs a selector device, functioning much like a diode, connected in series with the memory unit. This selector is designed not to permit the flow of current until a specified voltage difference is achieved, thus efficiently preventing sneak paths in half-selected cells. Selector devices need to have a bidirectional non-linearity. Two types are commonly worked on: exponential I-V such as Ni/TiO2/Ni, or threshold switching such as Mott memristors (NbO2) [204]. Noting that in this architecture, it is possible to create 1S1R devices, stacking selector and memristor together, considered as the most preferable scheme for high-density 3D integration of RRAM [205]. However, optimizing both the 1R and 1S properties of such devices is very challenging [203].

With their inherent capacity to emulate both synaptic and neuronal behaviors, memristors present a promising foundation for constructing efficient, densely packed neural network architectures, well-suited for neuromorphic computing applications. However, translating this potential into fully realized, practical memristive systems presents obstacles. One of the major challenges is the imperfect behavior of memristors, as they exhibit inter-device and intra-device variabilities inconsistencies and are subject to drift and aging. These factors can deteriorate the precision of memristor conductance control, a crucial aspect of synaptic weight updates in learning algorithms. Moreover, because of their filamentary nature, these devices are intrinsically noisy, which can introduce further unpredictability in their operation. Non-linearity and asymmetry are additional factors complicating the implementation of memristors in neural networks as both of these characteristics prevent precise control of the memristor state.

In this chapter, we explored the complexity of how neurons behave and the various models used to understand them. Implementing such devices on hardware is a promising lead to implement Spiking Neural Networks. Chapter 2 will focus on NbOx volatile memristors, which

exhibit spiking and bursting behaviors. They hold the promise of scalable and low-energy neuron devices for neuromorphic applications. We also saw that employing bio-plausible or locally inspired learning algorithms is a promising lead to realize energy-efficient hardware capable of learning. Chapter 3 will explain how we adapt the Equilibrium Propagation algorithm to work in real-world physical systems. In Chapter 4, we shift focus to test the resilience of the Equilibrium Propagation algorithm when it is implemented with HfOx memristor devices as synapses. This step is vital in translating our research from theory to practical applications, as these filamentary devices exhibit intra and inter-device variability.

Chapter 2

Characterization and Modeling of Spiking and Bursting in Experimental NbOx Neuron

As the interest in Artificial Intelligence (AI) grows, spiking neural networks offer an energy-efficient, hardware-compatible, and event-driven alternative to conventional artificial neural networks [206], particularly adapted for processing sensory and dynamical data. Hardware spiking neurons can be realized solely using complementary metal oxide semiconductor (CMOS) technology, but this type of implementation suffers from a lack of scalability [207]. This limitation explains the growing interest in the realization of new devices that feature neuronal behavior and that can be scaled easily [208, 209] (see section 1.3). However, researchers face the choice between single, scalable nanodevices that exhibit a limited range of neuronal responses and more complex neurons that offer more diverse behavior but limited scalability. Having more diverse behavior provides the potential of reproducing the brain's computational power to its full extent. Biological neurons may indeed exhibit different types of spiking responses, as well as bursting responses, where a neuron produces multiple spikes in response to an input pulse. A neuron implementing a highly simplified response will fail to provide the complexity required to emulate neurobiology. For example, the bursting response is believed to be of importance for ensuring reliable communication and synchronization between neurons [7, 154]. Therefore, considerable effort has been devoted to realizing new scalable devices with diverse neuronal characteristics [7, 11, 175, 210–213].

A leading idea to engineer this new type of devices is to exploit the intrinsic physics of nanoscale materials to implement neurons [214–218]. A large number of devices have been studied for their neuronal applications [219–221] (see section 1.3.1): phase change neuron [201], valence change neuron [185, 186], electrochemical metallization neuron [222], diffusive neuron [223], Mott insulator neuron [224], and spintronic neuron [188]. Within these examples, metal/insulator/metal structures based on transition metal oxides such as VO_x and NbO_x are particularly promising candidates, as they exhibit reliable threshold switching and current-controlled negative differential resistance (NDR) characteristics. NbO_x memristor neurons feature high endurance [225] and have been shown to be capable of leaky integrate-and-fire, all-or-nothing spiking and chaotic oscillations [226]. This type of device has also been used to implement dynamic, logic, and multiplicative gain modulation [227]. However, the behavior of a single device is nowhere near as complex as a real biological neuron. To obtain more sophisticated behavior, complex devices featuring multiple electrophysical processes have to be created [210], which can be challenging to model and control precisely. Alternatively, several neuronal devices can be used together in appropriately engineered circuits [175].

In this work, we fabricate and characterize memristor neurons based on a simple $Pt/Nb_2O_5/Ti/Pt$ stack with current inputs and output voltage shapes that are close to the shape of a biological action potential, thanks to the effect of an inductance. These devices are straightforward to model with physics equations, and simultaneously, feature multiple computational properties such as tonic spiking, stochastic spiking, spike latency, leaky-and-fire integration (LIF),

all-or-nothing firing, and phasic bursting. These neuron-like dynamics can be modelled and understood through physical equations and standard non-linear dynamics.

This chapter is adapted from Ref. [228].

2.1 Fabrication and method

2.1.1 Fabrication

NbO_x memristors, comprising 5 $\mu m \times 5 \mu m$ cross-point structures, were fabricated by successive film deposition and patterning, which will be detailed below.

A 4-nm Ti adhesion layer and a 25-nm thick Pt layer were first deposited on a SiO_2/Si substrate by electron-beam evaporation. Electron beam evaporation, often abbreviated as E-beam evaporation, is a type of physical vapor deposition (PVD) process in which a target anode material is bombarded with an electron beam given off by a charged tungsten filament under high vacuum. The electron beam causes material from the target to transform into the gaseous phase. These atoms then precipitate into solid form, coating everything in the vacuum chamber (within line of sight) with a thin layer of the anode material. The process starts by creating a high vacuum inside a deposition chamber. This is necessary to minimize the presence of air or other gaseous molecules that can interfere with the evaporation process. An electron beam is then generated using a filament, often composed of tungsten. When a high voltage is applied, electrons are emitted from the filament, creating the electron beam. This beam is directed and focused using magnetic fields onto the material that is to be evaporated. The high-energy electron beam heats the target material. The energy transferred from the beam to the target is so intense that it causes the target material to heat up and eventually evaporate. The evaporated atoms or molecules travel in a straight line from the source to the substrate in the vacuum chamber. As they reach the substrate, they cool and condense, forming a thin layer or film on the substrate's surface. The film's characteristics can be controlled by adjusting the process parameters such as beam current, deposition rate, and substrate temperature.

The wafer was first dried for 5 minutes at 100°, then spin-coated with SPR700 photoresist, then soft baked for 1 minute at 95°C, before being patterned using optical lithography with a digital mask. A hard bale step of 1 minute at 115°C follows before development in acetone and isopropanol. Ion-beam etching was then used to define the bottom electrodes. The milling step was done at a temperature of 4 Celsius degrees and a 30-degree angle to lower the odds of redeposition. This step was done with secondary-ion mass spectrometry (SIMS) to ensure that no platinum or titanium was left. Ion Beam Etching (IBE) is a form of dry etching, a critical technique utilized for precise, controlled material removal from a substrate. In IBE, a high-

energy ion beam is directed at the substrate, sputtering or ejecting surface atoms by collisional energy transfer. Unlike some other etching techniques, IBE does not depend on chemical reactions to remove material; rather, it employs a physical process, which grants the advantage of material selectivity. By adjusting the ion beam parameters such as energy, angle of incidence, and ion species, a high degree of etch control and anisotropy can be achieved.

If the previous steps (fabrication of the platinum bottom electrodes) had been done for more than a month, the application of an oxygen plasma was needed to ensure that the spin-coating stuck to the samples. The same optical lithography steps were repeated, and the new digital mask was manually aligned with the previous layer to ensure the superposition of the result.

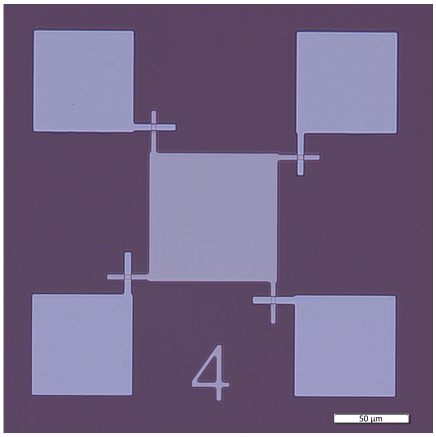


Figure 2.1: Top view of devices taken with an optical white light microscope.

A 30 nm Nb_2O_5 layer was then deposited onto the bottom electrodes using radio-frequency (RF) sputtering from a Nb_2O_5 target at room temperature in an Ar ambient. During the process, an RF power source is used to generate a plasma in a chamber filled with a gas such as argon. The gas atoms are ionized, and the resulting ions are accelerated towards a target material, typically a metal or ceramic, which is placed in the same chamber. The ions collide with the target material, causing atoms to be ejected or "sputtered" from the surface.

The metal-oxide-metal device was completed by adding a top electrode (10 nm Ti - 25 nm Pt) deposited by electron beam evaporation. The top view of resulting devices is presented in , figure 2.1.

2.1.2 Electrical measurements

2.1.2.1 DC measurements

For electrical measurements, the bottom electrode was connected to ground and the source applied to the top electrode. I-V characteristics were measured with a Keysight B1500A Semiconductor Device Analyzer after current-controlled electroforming with a positive polarity. Before the electroforming process, the resistance of the device was about 4 MΩ at 0.3 V. Electroforming was achieved by the application of a current ramp from 0 to 0.5 mA to the device, see figure 2.2. After this step, the device resistance was reduced to 93 kΩ at 0.3 V.

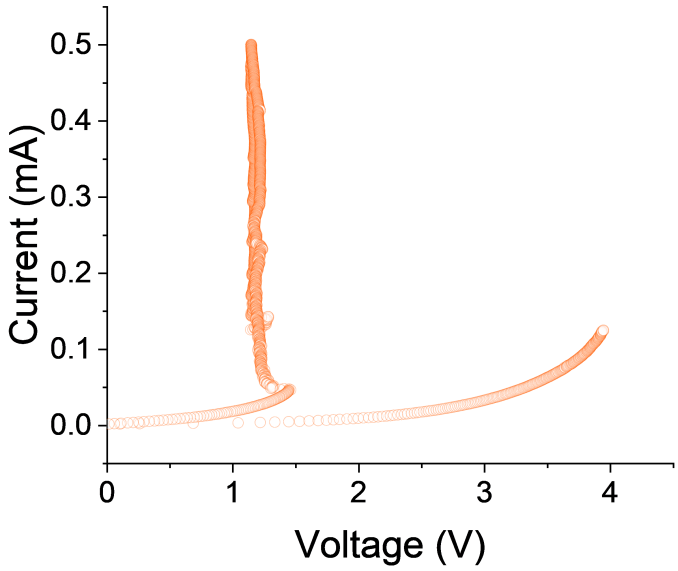


Figure 2.2: Positive current-controlled electroforming with input current going from 0 to 0.5 mA.

Several I-V sweeps were needed to stabilize the characteristics of the device. If this step were not done, the I-V curve would either shift for each run, or the NDR could even be absent. Sometimes, the devices would become a memory, and a big voltage would be needed to reach the threshold switching regime again.

2.1.2.2 Spike measurements

Current-controlled pulse measurements were performed using an Agilent 81160A pulse generator and the voltage-pulse to current-pulse converter presented in figure 2.3. The spiking behavior was monitored on a 2 GHz-bandwidth Keysight MSOS204A oscilloscope. All measurements were performed with a DC probe station.

For the voltage-controlled measurement, the setup was identical but the voltage-pulse to current-pulse converter was not needed.

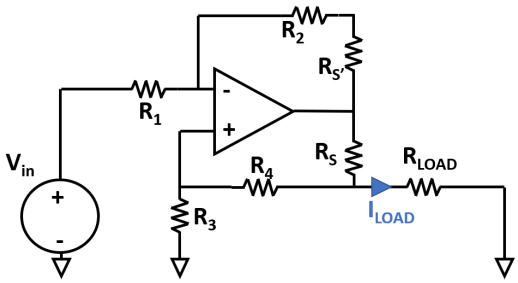


Figure 2.3: Schematics of the voltage pulse to current pulse converter used in the experiments. Here, $R_1 = R_3 = 1\ k\Omega$, $R_2 = R_4 = 100\ \Omega$, $R_S = R_{S'} = 400\ \Omega$. The operational amplifier has the following reference: BB OPA 356A 846LV.

2.2 Results

2.2.1 Quasistatic properties

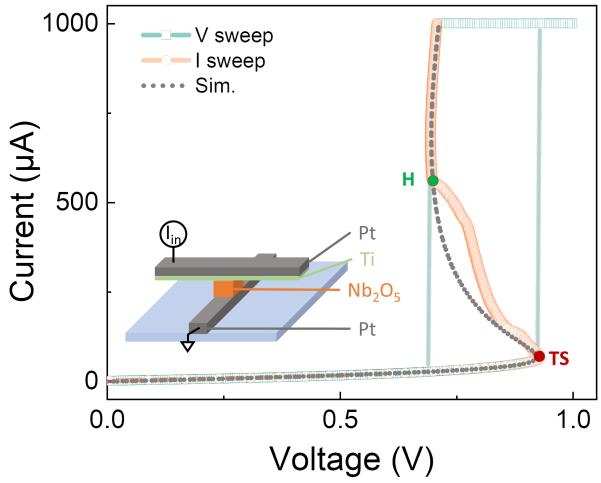


Figure 2.4: Measured (dashed lines) and simulated (dotted line) I-V characteristics. The V sweep and I sweep correspond respectively to the voltage-controlled and current-controlled I-V characteristics. The hold point H is indicated in green and the threshold switching point TS in red. The inset shows a sketch of the structure of the device.

The quasistatic I-V characteristics of our device are shown in figure 2.4, highlighting the current-controlled S-shaped Negative Differential Resistance (NDR) response, characteristic of a voltage-controlled Threshold Switching (TS). Two characteristic values are included on the graph. The first one is the Threshold Switching point (called TS in figure 2.4), where the slope of the current-controlled I-V characteristic goes from positive to negative. This point also coincides with the abrupt transition from a high-resistance state to a low-resistance state under

voltage controlled transitions. The second is the hold point H, where the differential resistance becomes positive again. It is worth noting that unlike typical memristors (as in 'memory' and 'resistor'), this device loses information about its previously set state when the power is turned off. Therefore, it cannot be used as a memory.

The slight cycle-to-cycle variations of the I-V characteristics of the device are shown in figure 2.5. A hundred repetitions have been realized for the voltage-controlled characteristics, and ten measurements have been done for the current-controlled ones. The characteristics can shift from cycle to cycle, but this does not impact the overall behavior of the neuron.

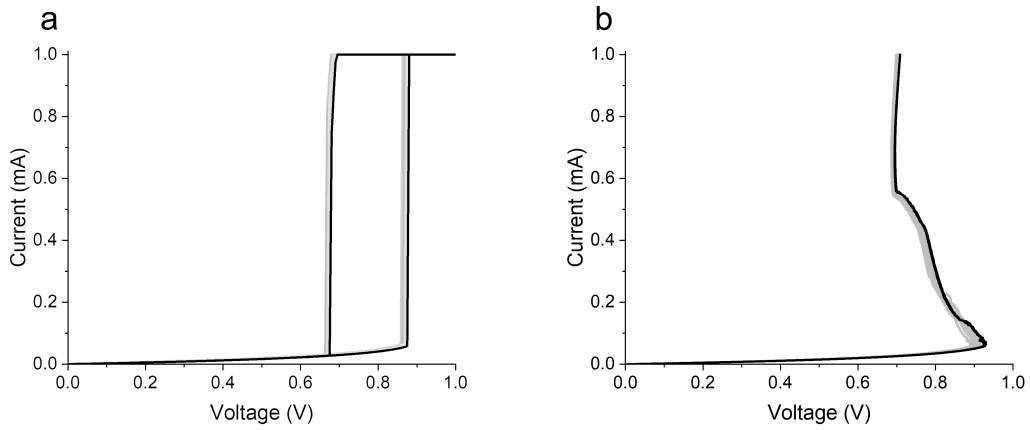


Figure 2.5: a: Voltage-controlled I-V characteristic repeated 100 times. b: Current-controlled I-V characteristic repeated 10 times.

The physical basis of this behavior has been under debate but is generally understood to arise from an increase in the oxide electrical conductivity due to local Joule heating. Indeed, Gibson [229] has shown that the NDR response can arise from any mechanism that gives rise to a superlinear increase in conductivity with temperature. In the case of NbO_x, some authors initially attributed it to a characteristic insulator-to-metal transition (IMT) in NbO_2 [230], but it is now generally accepted that it arises from a trap-assisted transport mechanism, such as Poole-Frenkel conduction [231, 232].

In the case of the Poole-Frenkel effect, a filament of oxygen vacancies connects both electrodes after electroforming. The oxygen vacancies act as potential traps for electrons. If an electric field is applied to the device, the energy profile of the conduction band in the oxide

around the traps becomes asymmetric. Trapped electrons are then able to be thermally injected into the conduction band, leading to the traditional Poole-Frenkel equation for the device resistance R_d as a function of the temperature T_d and the voltage V_d across the device:

$$R_d = R_0 exp\left(\frac{E_a - q\sqrt{\frac{qV_d}{\pi\epsilon_0\epsilon_r d}}}{k_B T_d}\right), \tag{2.1}$$

where E_a is the activation energy associated with the carrier trap level, ϵ_0 the vacuum permittivity, ϵ_r the relative permittivity of NbO_x, q the elementary charge, and d the thickness of the oxide film. V_d is the device voltage and T_d is the temperature of the active device volume [231]. The occurrence of electrical current through the filament results in a positive feedback, where Joule heating raises the local temperature T_d, reducing the device resistance further [233, 234]. This phenomenon can be modeled from a lumped element model of the device, where the Newton's Cooling Law is used to describe the evolution of the temperature,

$$\frac{dT_d}{dt} = \frac{V_d^2}{R_d C_{th}} - \frac{T_d - T_{amb}}{C_{th} R_{th}} \tag{2.2}$$

where T_{amb} is the room temperature, and C_{th} and R_{th} are respectively the thermal capacitor and resistor. We simulated the I-V curve of our device using these equations (see methods). The simulation results presented with a dotted line in figure 2.4 show that the model reproduces the experimental data.

2.2.2 Spiking behavior: Origin and shape

2.2.2.1 Voltage-controlled spiking

This chapter focuses on the phenomenon of current-induced spiking, which can be a complex topic to understand. To aid in comprehension, we first provide an explanation of the *voltage-controlled* behavior. To facilitate this understanding, we conduct an experiment where a voltage source is connected to a load resistor of 4000 Ohms, and then connected to the device under test (DUT), as shown in the circuit presented in figure 2.6a. The DUT can be seen as a capacitor in parallel to the Poole-Frenkel (PF) resistance of equation (2.1).

Initially, the PF resistance is in a high resistance state, and the capacitor is discharged. As the voltage is applied, the capacitor begins to charge, leading to an increase in voltage across the DUT. If the applied voltage is sufficient, the voltage across the device will reach the threshold point (denoted as TS in figure 2.4), causing the PF resistance to drop and shortening the capacitance. This leads to a discharge of the capacitor and a subsequent decrease in voltage across the device. If the voltage across the DUT is below the hold voltage (denoted as H in Figure 2.4) when the capacitor is fully discharged, the PF resistance abruptly increases once again, and the capacitor starts to charge anew. This hysteresis then explains the spiking behav-

ior in such devices, which can be observed in figure 2.6b. This explanation is more of an image than an accurate description of the current and voltage's evolution in the device, as figure 2.4 presents the quasistatic values of I and V and not the dynamic, out-of-equilibrium values. This dynamical behavior of this device is studied with more detail in section 2.2.5.

To conclude, the voltage-controlled spiking behavior originates from the current and voltage hysteresis, with each description of the cycle representing one spike. This voltage-controlled hysteresis (TS) is equivalent to the current-controlled NDR.

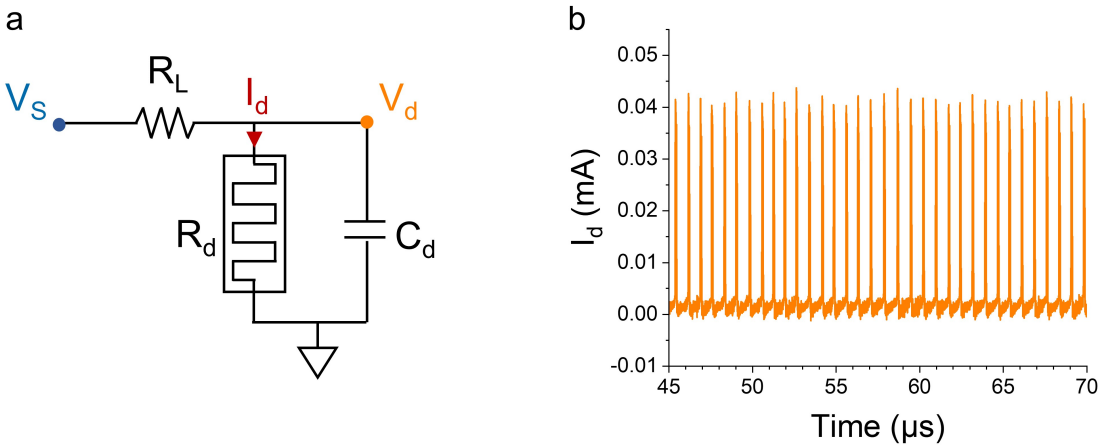


Figure 2.6: a: Circuit for voltage-controlled spiking neuron. b: Device current I_d when a constant voltage of 1.52 V is applied.

2.2.2.2 Current-controlled spiking

Figure 2.8b presents the simple experimental setup used to measure the current-controlled spiking behavior of neurons. In this circuit, R_d is the device resistance described by equation 2.1 and C_d is the intrinsic device capacitance arising from its metal/insulator/metal structure. C_{ext} and L_{ext} respectively account for parasitic capacitance and inductance of the measurement set-up. R_{out} is an external resistor of 25 Ohms across which the output voltage is measured.

The input of the circuit is a current, and the output is a voltage, in line with the biological configuration. Figure 2.8c shows an experimentally measured spike, observed by applying a constant 150 μA current input to the circuit. The shape of the output spike strongly resembles that of a biological neuron, with an initial depolarization followed by hyperpolarization: starting from a resting phase, the output voltage increases rapidly during the activation phase, and then decreases to become negative before rising again to the resting phase.

To understand this behavior, figure 2.8b shows simulations of the current I_d flowing through the device (dots) and the simulated temperature T_d of the active device volume (colored curve)

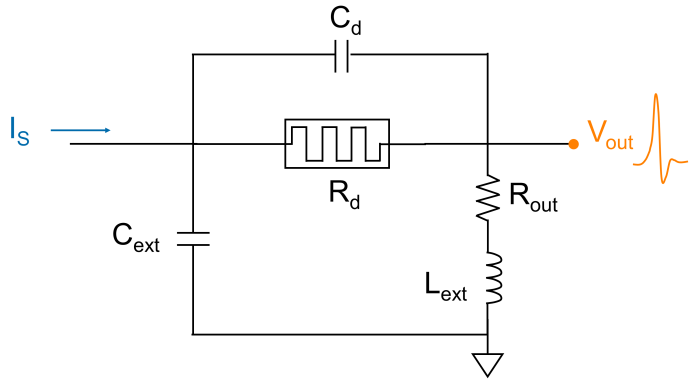


Figure 2.7: Circuit diagram of the integrated NbO_x spiking neuron where C_{ext} and L_{ext} are respectively a parasitic capacitance and inductance.

during a spike, using the LTSpice model of our experiment and a current input of 180 μA. The simulations presented in figure 2.8 were computed with LTSpice using the electrical circuit presented in figure 2.7a based on the Newton law of cooling and the Poole-Frenkel effect (see equations (2.1) and (2.2) below), with a 5 ns time step. The values of all parameters used in these simulations are listed in table 2.1. The temperature evolution was implemented in LTSpice following guidelines described in the supporting information of [235].

The current I_d and the temperature T_d are clearly correlated, with both curves exhibiting a rapid increase and a slower decrease, which can be explained as follows. The device is initially in an insulating state. When a constant current is applied, the capacitance C_{ext} charges and the voltage across the device increases until it approaches the threshold voltage, at which point the device resistance drops, producing the increase in current and temperature evident in figure 2.8b. This discharges the capacitor, reducing the device voltage to the point where the memristor reverts to its subthreshold resistance. The transition to a high resistance state causes a reduction in current and temperature, ending the spike response. Note that without the external capacitance C_{ext} the neuron would not spike. In a hardware implementation involving NbO_x neurons, capacitors would have to be added.

The restoration part of the neuron-like voltage spike is seen in the output voltage but not in the current and temperature curves; this is due to the presence of a parasitic inductance (see figure 2.7). The device intrinsic capacitance C_d is small, and the current in that branch is also small. Therefore, the current going through the inductance L_{ext} and the output resistor R_{out} (figure 2.8b), is close to that going through the neuron R_d. Because the voltage across the inductance opposes the variations of the current, it is first positive and then negative. The output voltage is the sum of two terms, $V_{out} = R_{out} i_{out} + L_{ext} \frac{di_{out}}{dt}$: if the inductance is large enough, the output voltage is first positive (during the activation part) and then decreases until it becomes negative (during the cooling and restoration parts). This mechanism explains the

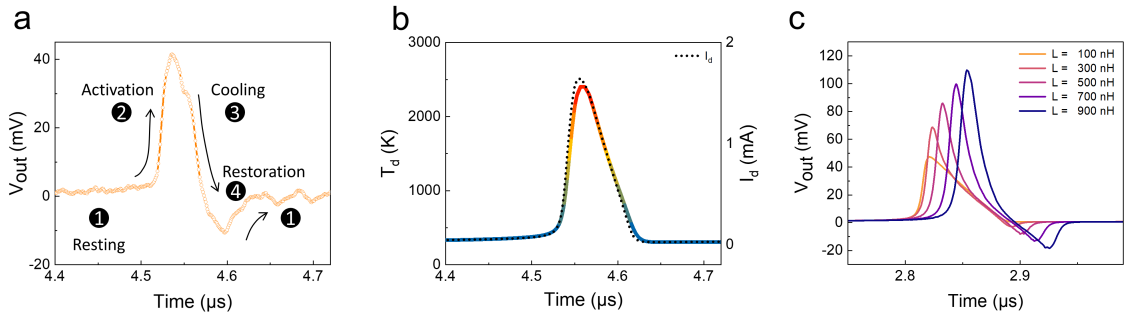


Figure 2.8: a. Measurement of a single spike of a NbO_x neuron, with the four stages of an action potential indicated. b. Simulated spiking dynamics of the NbO_x neuron temperature T_d(colored curve) and current I_d (dots) for a constant input current of 180 μA. c. Simulation of the output voltage shape with respect to the value of the circuit inductance for a constant input current of 180 μA.

results shown in figure 2.8c, where the evolution of the shape of the pulse with respect to the circuit inductance is simulated. When the inductance is smaller than 100 nH, it has a negligible impact on the output voltage (shown in figure 2.8a); for higher inductance values, a restoration phase is observed.

Variable	Value
C_{ext}	200 pF
L_{ext}	700 nH
R_{out}	25 Ω
C_d	0.33 pF
R_0	190 Ω
E_a	0.215 eV
ϵ_r	45
d	31 nm
C_{th}	2e-15 $J \cdot K^{-1}$
R_{th}	2040816 $K \cdot W^{-1}$

Table 2.1: Table of the parameters used in the simulations. These parameters where obtained by fitting the I-V characteristics and estimated by fitting the shape of the spikes.

The energy consumption was estimated by integrating the power over a period of time and dividing the resulting energy by the number of corresponding spikes. We derive a value of about 80 pJ/spike, which is comparable to the value found in other papers for NbO_x devices [236] [237]. In the future, reducing the external capacitance (here parasitic) could drastically reduce the energy consumption. Simulations using a model detailed below indeed show that with a parasitic of 10 pF the energy is close to 8 pJ/spike whereas for the 200 pF capacitance, the results are close to 80 pJ/spike.

Three different batches of samples have been realized, all showing devices with the same type of behaviors as the one reported in this work, with a significant device-to-device variation. This variability both impacts the shape of the I-V characteristics, but also the amplitude of the spikes.

2.2.3 Computational properties

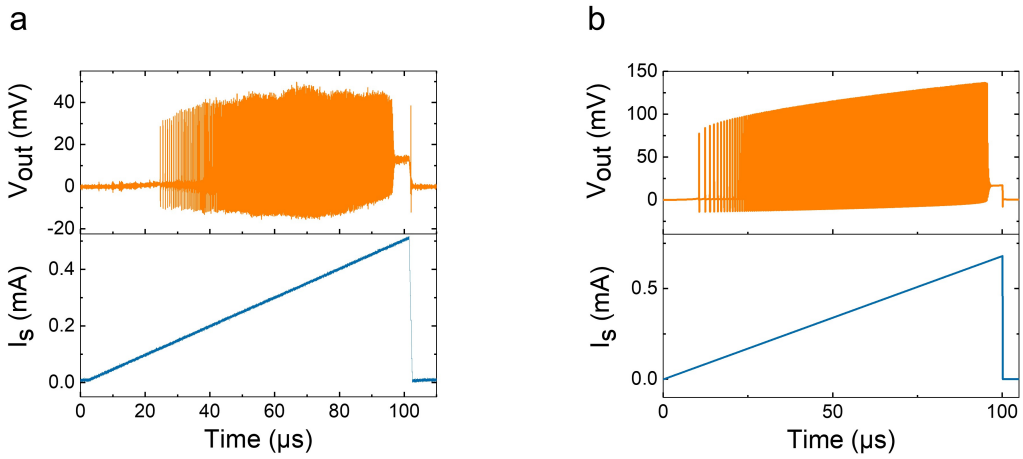


Figure 2.9: a: NbO_x neuron output as a function of input current amplitude. A 99 μs current ramp from 0 to 0.46 mA and 1 μs fall time is applied to the device. b: Simulation of NbO_x neuron output as a function of input current amplitude. A 100 μs current ramp from 0 to 680 μA and 100 ns fall time is applied to the device. This simulation is realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1.

Having analyzed the NbO_x neuron spike shape (see section 2.2.2.2), we now explore its computational properties. Figure 2.9a shows the neuron behavior when a current ramp is applied at its input. For low currents the neuron does not spike, as the NDR behavior needed for spike generation does not appear until the current reaches the TS point in figure 2.4. Above this threshold current, the neuron spikes with increasing frequency until the current exceeds the hold value (H) of figure 2.4, above which the NDR disappears as well as the related spiking behavior. This characteristic is reproduced in simulations in figure 2.9b. The behavior described above is also found in the simulated figure. However, a major difference between the experimental results (figure 2.9a) and the simulations (figure 2.9b) is the evolution of the amplitude. Over all the devices measured, the amplitude tends to increase with frequency, but this increase is not monotonous. In contrast, the simulations show a very regular increase, which is almost linear.

When the input is constant and lies between the threshold current and the hold current, the

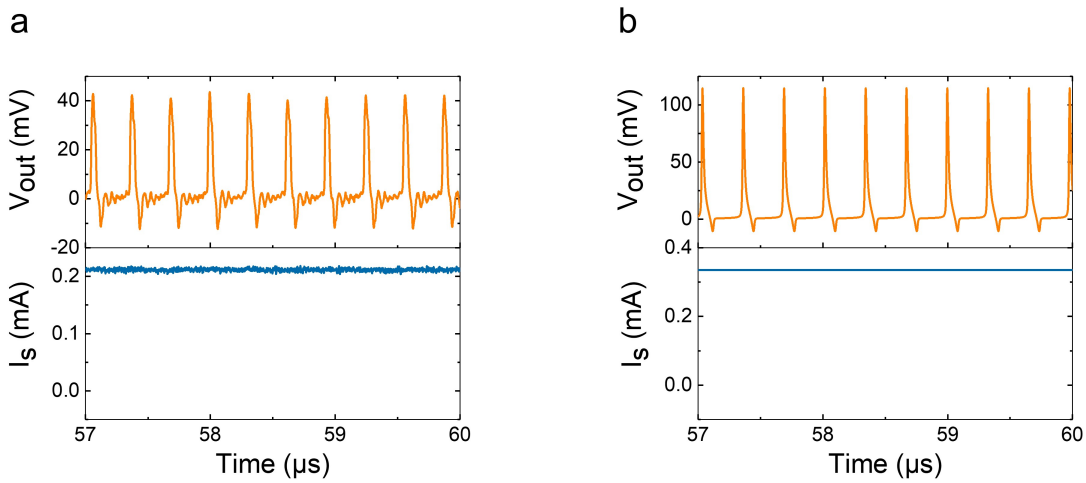


Figure 2.10: a: Tonic spiking. The neuron receives a constant input current of 0.2 mA. b: Simulation of tonic spiking. The neuron receives a constant input current of 335 μA. This simulation is realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1.

neuron spikes with a constant frequency, a behavior called tonic spiking for biological neurons, as shown in figure 2.10a, and reproduced in simulations in figure 2.10b. The behavior is similar, with a constant frequency. However, the amplitude of the spikes is more than two times bigger. The spikes amplitude have be found to differ a lot from device to device.

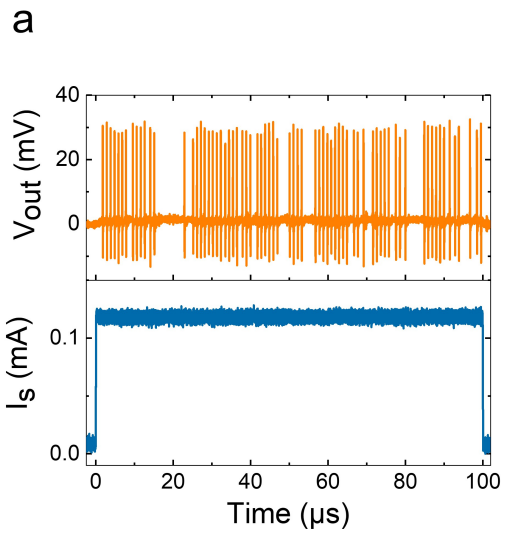


Figure 2.11: Stochastic spiking obtained with a current of 0.109 mA.

Close to the threshold current, the behavior is stochastic, as shown in figure 2.11a, as can be expected from a thermally-driven process, but with a non-random occurrence of spiking

events, that can be described by quiet periods followed by bursts of spikes with constant frequency. Due to input current noise, the neuron output indeed fluctuates between its below-threshold behavior (no spikes) and its above-threshold behaviors (spikes with a constant frequency). This stochastic bursting behavior is reminiscent of biological neuron bursting and could be exploited for computations and learning in hardware circuits [154]. No simulations have been provided for this figure. Indeed, when the input current is close to the hold current, a stochastic-like behavior can be observed in simulation, but is caused by a numerical instability.

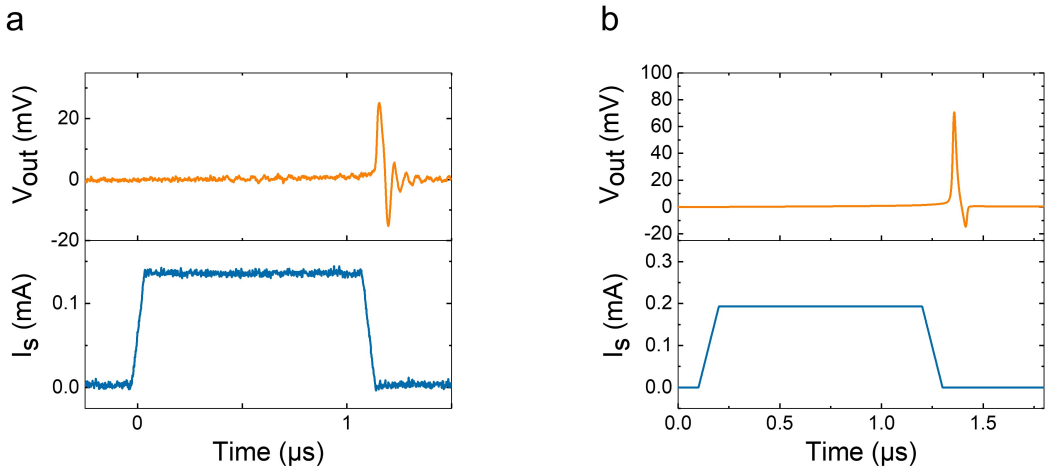


Figure 2.12: a: Spike latency. A pulse with a duration of 1 μs, a rise time and fall time of both 100 ns and an amplitude of 0.131 mA is applied to the neuron. b: Simulation of spike latency. A pulse of duration of 1 μs and value 193 μA with a rise time and fall time of both 100 ns is applied to the neuron. This simulation is realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1.

The neuron also exhibits spike latency, as evidenced in figure 2.12a for a 1 μs-duration pulse applied to the device. During the whole duration of the input, the output voltage does not show any significant response. However, once the pulse is back to zero, the neuron spikes. This effect can be explained naturally within the context of the above model. Indeed, when the current pulse is applied long enough for the temperature to activate the Poole-Frenkel effect, the positive feedback mechanism starts and the temperature keeps increasing even as the source stops, giving rise to spike latency. This behavior is simulated in figure 2.12b, and agrees quite well with the experiment. However, the simulation result is lacking the oscillations seen after the spike.

Moreover, the neuron may exhibit all-or-nothing behavior. In figure 2.13, two pulses with the same duration of 1 μs are applied to the neuron with different current input values: 0.13 mA for the left figure and 0.17 mA for the right one. The first pulse is not sufficient to make the neuron spike, but a slight variation of the output voltage can be observed. The second pulse is high enough to make the neuron spike, as the value of the current has been increased. In the context

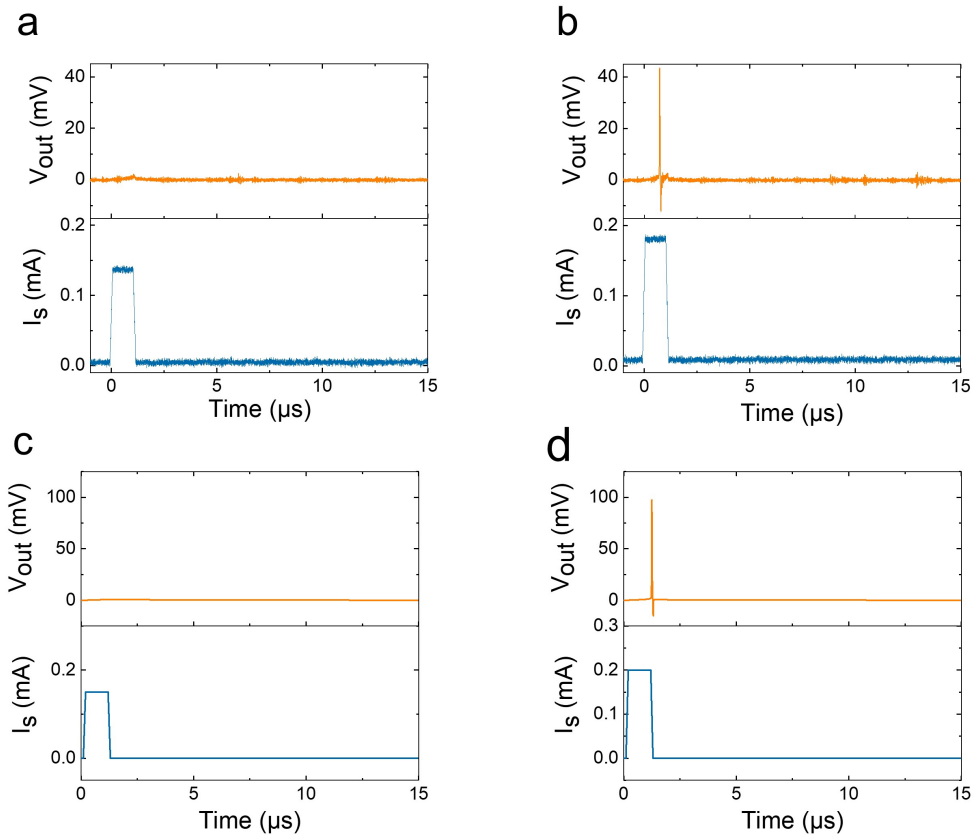


Figure 2.13: a: Spatial integration. Comparison between two figures where a pulse of duration of 1 μs with a rise time and fall time of both 100 ns are applied to the neuron. The input current value is 0.13 mA on the left and 0.17 mA on the right. b: Simulation of spatial integration. Comparison between two figures where a pulse of duration of 1 μs with a rise time and fall time of both 100 ns are applied to the neuron. On the left, the value of the current is 150 μA. On the right, the input current value is 200 μA. These simulations are realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1.

of a spiking neural network, this all-or-nothing behavior allows triggering a neuron only when a sufficient number of spikes (with below-threshold amplitude) arrives simultaneously at its input, thus filtering meaningful signal only, a behavior akin to spatial summation. Indeed, in a biological neuron, spatial summation corresponds to the possible trigger of an action potential when multiple inputs arrive simultaneously. Therefore, the spatial information can be encoded in the current amplitude. This all-or-nothing behavior is reproduced with simulations in figure 2.13b.

Finally, figure 2.14 displays a different situation where three pulses of identical duration (1μs) and peak current (0.11 mA) are applied. On the left, the input frequency of 0.35 MHz is not high enough for the neuron to spike, contrary to the right panel in which the frequency is increased to 0.7 MHz, allowing it to spike. This behavior indicates a frequency-dependent

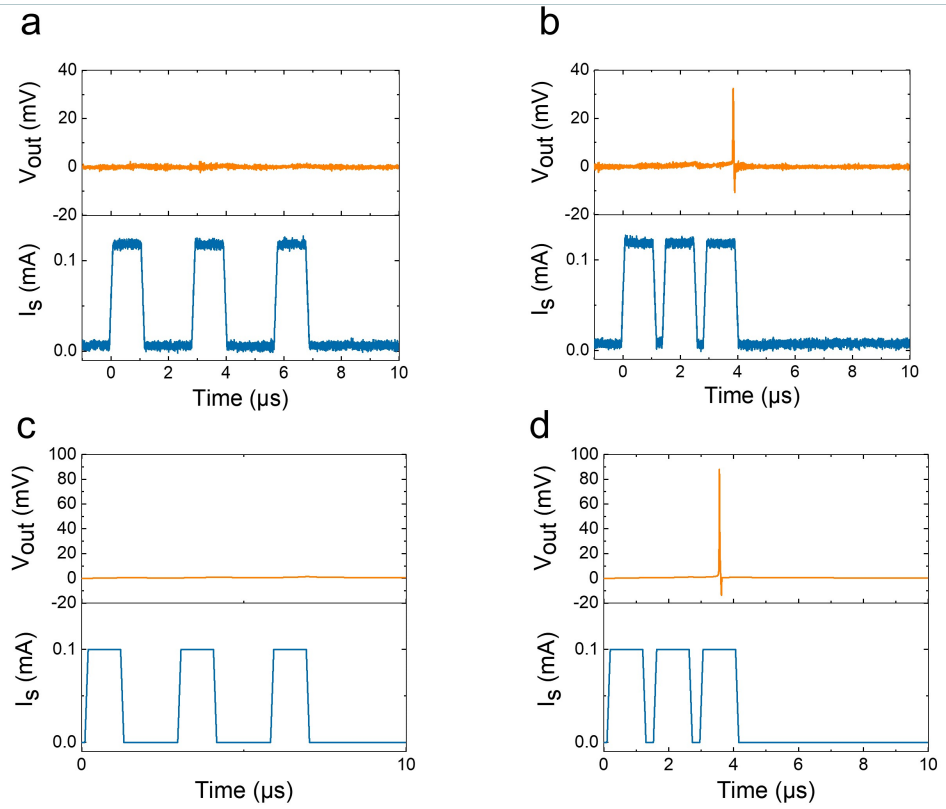


Figure 2.14: a: Temporal integration. Three pulses of duration of 1 μs with a rise time and fall time of both 100 ns and of amplitude 0.110 mA are applied to the neuron. The frequency is 0.35 MHz on the left and 0.7 MHz on the right. b: Simulation of temporal integration. Three pulses of duration of 1 μs with a rise time and fall time of both 100 ns and of value 100 μA are applied to the neuron. On the left, the time period is 2.86 μs (frequency of about 0.35 MHz). On the right, the time period is 1.43 μs (frequency of about 0.7 MHz). These simulations are realized in LTSpice, using the circuit shown in figure 2.7 and the parameters of table 2.1.

temporal summation by the neuron, reproduced with simulations in figure 2.14b. This typical leaky-integrate-and-fire behavior is particularly adapted for spiking neural networks where frequency encodes the information.

2.2.4 Experimental demonstration of phasic bursting

While most of the spiking features presented in figures 2.9, 2.10, 2.11, 2.12 2.13 and 2.14 have been reported for various types of solid-state neurons [185, 210, 222, 224], figure 2.15 shows that our simple NbO_x neuron exhibits a behavior observed in biological neurons and scarcely investigated in memristive systems, named phasic bursting [210]. In this case, for a constant input current just above the hold point (see figure 2.4a), the neuron starts to spike before stopping abruptly, as shown in figure 2.15. This situation differs from figure 2.9a, where a current ramp

was applied. In figure 2.9a, the neuron stopped spiking near the end of the input ramp, because the input current ended well above the Hold current (H point in figure 2.4). In figure 2.15, the input is now constant and the neuron still spikes before stopping abruptly. The amplitude of the spikes appears constant, before sharply decreasing until completely disappearing. Once the neuron stops spiking, it does not start spiking again if the input does not change. Our measurements indicate that, if pulses of the right current values are applied successively, the neuron will start spiking each time before eventually stopping. However, the duration of phasic bursting is not always the same even if the input is identical.

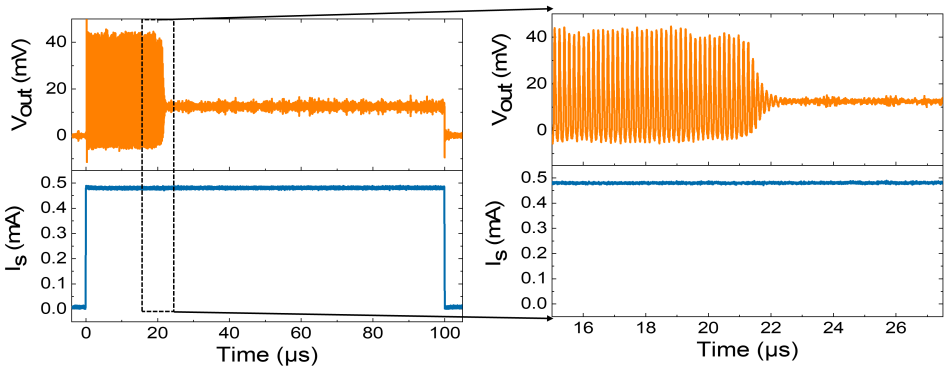


Figure 2.15: Example of phasic bursting of the output voltage as a function of time. A current input of amplitude 0.47 mA is applied. The right panel zooms on the end of the phasic bursting.

In order to quantify the effect, a statistical study of phasic bursting as a function of input current is presented in figure 2.16. A current pulse is applied to the neuron, its output is recorded on the oscilloscope, and the average frequency during the pulse duration is then computed for each point. When the phenomenon of phasic bursting occurs, spikes stop during a fraction of the total duration of the pulse, which decreases the average frequency. Despite the apparent stochastic behavior, a clear trend in the mean frequency evolution as a function of input emerges. For low currents, there is at first almost no phasic bursting, and the median frequency is almost equal to the maximum frequencies observed. Then as the input current increases, the proportion of phasic events increases and the median frequency decreases until no phasic bursting occurs.

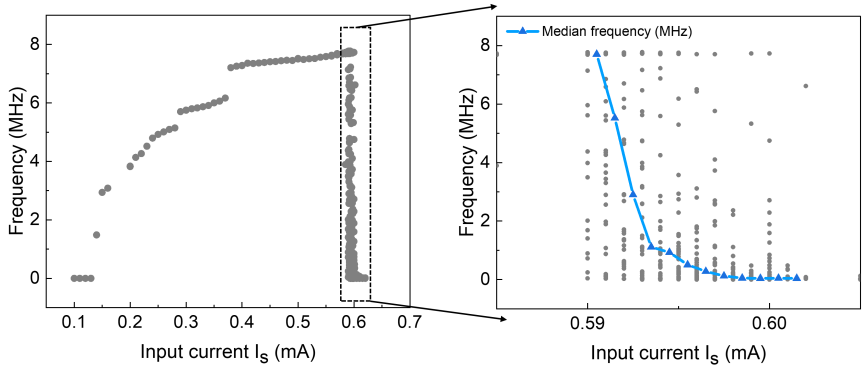


Figure 2.16: Variation of the average frequency as a function of the input current. Right: Zoom on the phasic bursting regime, in order to get a statistical understanding of the phenomenon. In blue, the median frequency computed from the different average frequencies (grey dots) is plotted.

2.2.5 Understanding phasic bursting with non-linear dynamics simulations

We now present a theoretical analysis to determine the origin of the experimentally-observed phasic bursting. We model our system with the circuit of figure 2.7, neglecting the parasitic inductance and the intrinsic capacitance, that do not impact the qualitative neuron dynamics, in order to gain in simplicity and generality. The system is then simplified to two coupled first-order differential equations that link the voltage V_d across the device and the temperature T_d inside the active volume of the device. The first equation reads

$$\frac{dV_d}{dt} = \frac{I_s}{C_{ext}} - \frac{V_d}{R_d C_{ext}}, \tag{2.3}$$

where I_s is the input current and R_d is the Poole-Frenkel resistance defined in equation 2.1. The second equation is the Newton Law of Cooling (equation 2.2).

Equations 2.2 and 2.3 can be solved numerically. The simulations shown in this section were executed in Python with a Runge-Kutta solver of order 5 and a timestep of 50 ps using equations (2.1) and (2.2). These simulations result in the different trajectories plotted in blue in figures 2.17a,b,c for the input current values I_s of 0.9, 0.96702 and 1.A mA respectively. The system nullclines are also shown in dotted lines. These curves correspond to the zero values of

the right-hand side of equations 2.3 and 2.2. Their intersection in the two-dimensional phase space (T_d,V_d) corresponds to points for which the derivatives of T_d and V_d are zero, and therefore gives the fixed point of the system for each input current.

Consistent with equation 2.2, the temperature nullcline does not depend on the input current I_s and is therefore identical in figures 2.17a,b,c (orange curve). On the other hand, increasing the input current vertically shifts the voltage nullcline to the top of the phase space. The current-dependent fixed points can therefore be obtained by following the temperature nullcline. For each of these points the Poole-Frenkel resistance can be computed, and by plotting the input current I_s as the function of the voltage V_d (thanks to the equilibrium relation $I_s = \frac{V_d}{R_d}$) the simulated quasistatic curve of figure 2.4 is obtained.

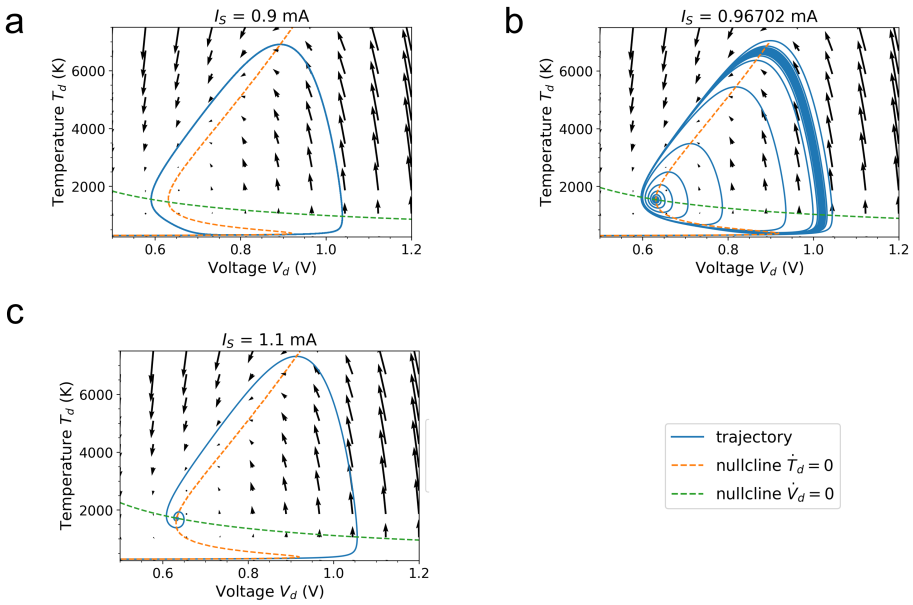


Figure 2.17: a, b, c: simulation of the trajectory (in blue) and the nullclines (in orange for $\dot{T} = 0$ and in green for $\dot{V} = 0$) for different input currents I_s of value 0.9, 0.96702 and 1.1 mA for each figure. The y-axis corresponds to the temperature T_d in the active volume of the device while the x-axis represents the voltage of the device V_d. The black arrows indicate the direction of the gradient at each point.

The analysis of figure 2.17 shows that phasic bursting is a particular situation that occurs around the hold point. The occurrence of this behavior is simply controlled by the constant source current applied to the device. Below the hold point, the fixed point is not stable, and the trajectory therefore reaches a limit cycle: this is what happens in figure 2.17a. At the hold point, the system undergoes a supercritical Hopf bifurcation, where the limit cycle becomes a

stable equilibrium point (as seen in figure 2.17c). Just above the transition (figure 2.17b), the system reaches a stable equilibrium point, but the convergence of the trajectory is quite slow (see figure 2.17b).

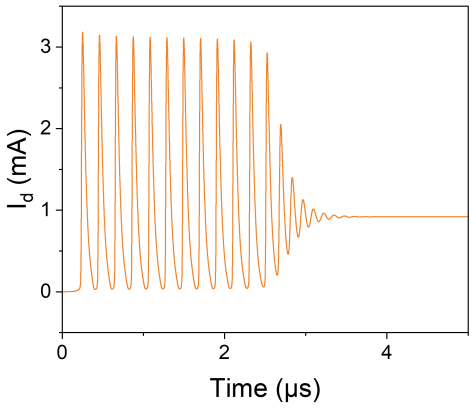


Figure 2.18: Simulations of the device current oscillations as a function of time for a current input I_s of 0.96702 mA .

This dynamic naturally gives rise to the phasic bursting phenomenon of figure 2.18, where an apparently stable train of spike unexpectedly fades out then stops.

2.2.6 Discussion and limitations of this model

In this chapter, we have demonstrated that the spiking memristor neurons we fabricated exhibit a range of both static and dynamic properties. Our work also shows that a simple model based on a one-dimensional Poole-Frenkel resistance is sufficient to replicate these properties, and provides deeper insights into the different behaviors observed. It is worth noting that the purpose of this model is not to provide a perfect understanding of the physics and mechanisms involved, but rather to have a set of simple equations that accurately describe the system. This is particularly important for neuromorphic computing, where simulating a physics-based neural network is a crucial first step towards on-chip learning, even though it can be computationally demanding. As a result, some discrepancies exist between the physics and the simulations, which we will address in this section.

2.2.6.1 Phasic bursting range

Interestingly, in the experiments as shown in figure 2.16, the current input range where the phasic bursting happens (ΔI = 0.04 mA) is about ten times larger than in the simulations (ΔI = 0.003 mA). The noise inherent to physical devices and to the input current (close to 0.018 mA in our experiments) explains the experimentally observed stochasticity of phasic bursting and expands the phasic bursting range. Indeed, even if the bias conditions of the device are set outside of the narrow range where phasic bursting is predicted in the absence of noise, fluctuations will enable the system to reach it and initiate the bifurcation, a phenomenon akin to stochastic resonance observed in biological neurons [238]. Other factors can also impact the details of the phasic bursting behavior. Simulations indicate that its corresponding current range ΔI could be increased for possible applications by lowering the value of the external capacitance, see figure 2.19.

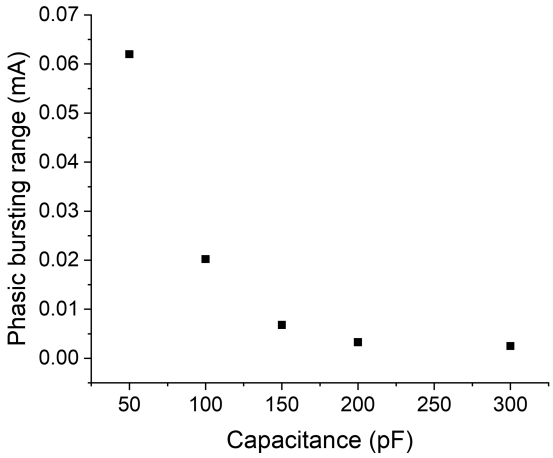


Figure 2.19: Evolution of the phasic bursting range with respect to the external capacitance. The parameters can be found in table 1.

An experimental study would be needed to confirm this trend. In the model, the thermal resistance is considered constant for simplicity, but this is not true in a real device, as shown in reference [239]. However, both measurements and adequate models are still lacking for temperatures higher than 450 K to include this dependency in the phasic bursting simulations.

2.2.6.2 Temperature in the filament

The Poole-Frenkel resistance model provides insights into the switching behavior of NbOx memristors, where the oxygen vacancy filament serves as the switching volume. However, the model's predictions of the temperature rise during the spike events, as seen in Figure 2.8 and Figure 2.17a,b,c, may seem implausible at first glance. The observed temperatures during

spikes are approximately 2500°C and 6500°C, respectively, which raises concerns about the validity of the model.

Several factors must be considered when interpreting the model's temperature predictions. Firstly, the switching volume has been assumed to be the entire NbOx layer, which is not necessarily accurate. It could be a much smaller portion of the layer. The filament is actually composed of just a few atoms, and its position and size may not be uniform throughout the device. Therefore, the concept of a well-defined temperature may not be applicable in this context. Moreover, the Poole-Frenkel model does not account for heat dissipation in the filament. The Joule heating effect caused by the current passing through the filament results in a localized temperature increase. However, since the model uses a simplified expression of heat dissipation in the filament, which does not take into account all the detailed thermal transfers inside the nanodevice, its temperature predictions are higher than the actual temperature.

In summary, while the Poole-Frenkel model provides a useful framework for modeling and understanding the switching behavior of NbOx memristors, its temperature predictions must be interpreted with caution.

2.3 Conclusion

Volatile NbO_x memristors are excellent neuron candidates as they are scalable, present reliable threshold switching, and are compatible with memristive synapses such as HfO_2 Metal-Insulator-Metal structures. We have shown that the $Pt/Nb_20_5/Ti/Pt$ stack presents well-suited I-V characteristics: a current-controlled S-shaped Negative Differential Resistance, which can be modeled by assuming Poole-Frenkel conduction. This type of device is able to spike and the resulting shape is very close to the one of a biological neuron with initial depolarization followed by hyperpolarization due to an inductance. We demonstrated that this device presents multiple computational properties such as Leaky-Integrate-and-Fire (LIF) characteristics, all-or-nothing-firing, and phasic bursting. We also investigated the origin of phasic bursting through the analysis of the physical equations of the devices. This phenomenon comes from the bifurcation between an unstable fixed point (limit cycle) and a stable fixed point (equilibrium) driven by Poole-Frenkel dynamics. These results pave the way to easily-scalable neurons that can be easily modelled and simulated but still show a complex behavior in order to mimic biological computations.

Chapter 3

Adapting Equilibrium Propagation to Physical Systems

As discussed in Chapter 1, on-chip-learning requires hardware and software co-design. This also means adapting algorithms for hardware, which are ideally local in space. This chapter discusses ways to adapt Equilibrium Propagation, a spatially local algorithm, to make it highly implementable in hardware.

3.1 Context

As we saw in Chapter 1, the BackPropagation algorithm, with its highly non-local updates, is difficult to implement on-chip. An alternative is then to take inspiration from biology and think about biologically plausible algorithms, as presented in Chapter 1, section 2.2. However, the physical nature of a system can also be exploited. This idea is not new, as in certain instances, the fundamental principles of physical systems have inspired learning algorithms. Notable examples of this include Hopfield networks [44] and Boltzmann machines, both of which are inspired by Ising spin systems. Numerous other methods have been suggested, leveraging the effects of nonlinear dynamics, such as bifurcations and chaos, for computation [240], [241], [242].

The existence of these works leads to the idea of using energy-based models for neuromorphic computing. These models use the minimization of an energy function to learn a task. This is particularly interesting because a physical system will minimize its energy function by nature. Using this intrinsic learning property with a local learning rule seems to be a promising path for neuromorphic computing. However, traditional energy-based models do not implement the supervised learning of a global objective function, and these models suffer from low accuracy, especially compared to BackPropagation.

In fact, in a network based on an energy model, neurons typically gravitate towards the lowest energy state that is nearest to the system's initial state, which corresponds to the input. The energy function of the network is parameterized, and by appropriately tweaking these parameters, we can establish minima that correspond to the various patterns we aim to store. Hopfield networks are often referred to as self-associative because they map an input (or initial state) to an equilibrium state that matches the size of the input. Limitations of Hopfield networks include notably the lack of emergent hierarchy in these networks due to the unsupervised learning and self-associative nature.

3.2 Equilibrium Propagation algorithm

Equilibrium Propagation (EqProp) is a learning algorithm introduced by Scellier and Bengio [138] based on gradient descent. However, contrary to BackPropagation, both the inference phase (free phase) and the learning phase (nudge phase) use the same operations. This is a considerable advantage for hardware implementation.

EqProp is an energy-based model that can be used for any connected networks (layered or

not), as long as the weights are symmetric ($W_{ij} = W_{ji}$), meaning that the weight W_{ij} will both evolve upon the variations of the rate-based neurons i and j. The most common architecture is to arrange the neurons in layers, with input on one side and output on the other, as presented for a MLP in chapter 1. EqProp takes inspiration from Contrastive Hebbian learning presented in Chapter 1 section 2, but use a weakly clamped second phase instead.

EqProp is a convergent recurrent neural network In a Recurrent Neural Network (RNN), some neurons take their own previous outputs as part of their input. This creates a kind of loop in the network, which allows it to maintain a form of 'memory' about previous inputs. This is what gives RNNs their temporal dynamic behavior, as the state of a neuron at time 't' is dependent not just on the current input at time 't', but also on its own state at the previous time step 't-1'. This makes RNNs particularly useful for tasks involving sequential data, where the order and context of inputs are important, such as language modeling, speech recognition, and time series prediction.

EqProp belongs to a subclass of RNNs, called convergent RNNs with a static input. This means that this type of RNNs is fed by a static input x and reach an equilibrium state. It is worth noting that EqProp is equivalent to Back Propagation Through Time (BPTT) [243].

Learning procedure The system considered follows a certain energy function E, which will be naturally optimized. This energy depends on the states of neurons called s, on the symmetric synaptic weights called W, and on the biases called b. Here, we will refer to the weights and biases as the parameters θ of the network. Therefore, the energy function can be written as: $E(x, s, \theta)$.

The first phase, the inference phase, is called the free phase. Given some input x, its corresponding target t, and some parameters θ, the states of the neurons will evolve to minimize the energy function according to the formula:

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s} \tag{3.1}$$

The states eventually reach an equilibrium state, referred to as $s^\star$, which is stored. The output neurons, called y, will give a first result that has no reason to be close to the target t. A cost function C is therefore defined to determine the error between the target and the real output of the network. The higher the cost function, the further away the output from the target is.

The learning phase, called the nudging phase, uses a new energy, the total energy function $F = E + \beta C$ where β is a non-zero real-valued parameter. The neurons, therefore, evolve according to the new equation:

$$\frac{ds}{dt} = -\frac{\partial F}{\partial s} = -\frac{\partial E}{\partial s} - \beta\frac{\partial C}{\partial s} \tag{3.2}$$

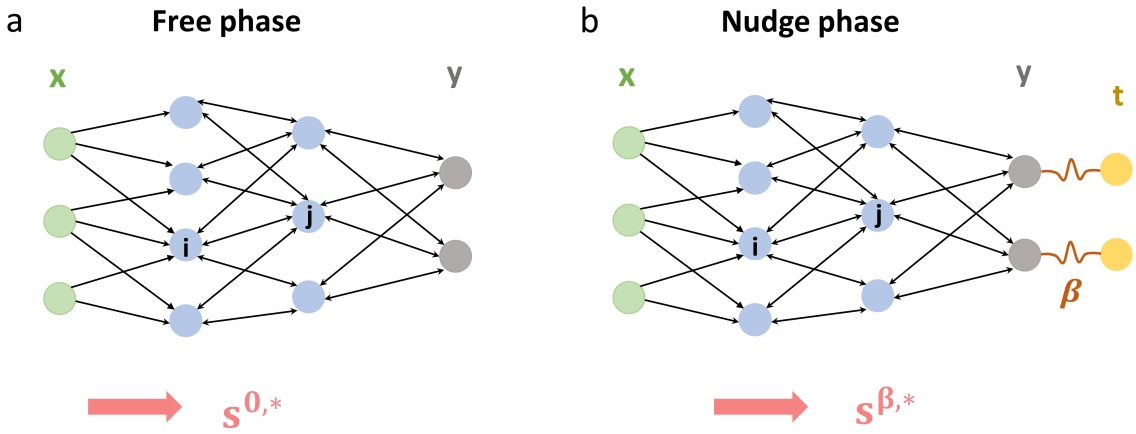


Figure 3.1: a: Free phase. b: Nudge phase.

This is similar to applying a force proportional to β to the output units y to nudge them towards the target t (see Fig. 3.1), while waiting for a new equilibrium to be reached. This equilibrium state is called $s^{\star,\beta}$. On the notation point of view, the equilibrium reached at the end of the free phase can be noted $s^{\star,0}$, as it corresponds to a dynamic evolution where $\beta = 0$.

One of the strengths of Scellier and Bengio's original work is the following theoretical guarantee:

$$\frac{\partial J}{\partial \theta} = \lim_{\beta \to 0} \frac{1}{\beta} \left(\frac{\partial F}{\partial \theta} \left(\theta, x, \beta, s^{\star,\beta} \right) - \frac{\partial F}{\partial \theta} \left(\theta, x, \beta, s^{\star,0} \right) \right), \tag{3.3}$$

where J is the objective function the network aims to minimize. Its expression is the same as the cost function C, but when C is defined for a single example, J is defined over the whole training set. In practice, the parameter update is:

$$\Delta\theta = -\eta \frac{1}{\beta} \left(\frac{\partial F}{\partial \theta} \left(\theta, x, \beta, s^{\star,\beta} \right) - \frac{\partial F}{\partial \theta} \left(\theta, x, \beta, s^{\star,0} \right) \right) \tag{3.4}$$

Note Scellier and Bengio reported that using a β with a random sign yielded better results that keeping the sign of β constant [138].

Energy choice The initial energy function proposed by Scellier and Bengio is [138]:

$$E(u) = \frac{1}{2} \sum_i u_i^2 - \frac{1}{2} \sum_{i \neq j} W_{ij} \rho(u_i) \rho(u_j) - \sum_i b_i \rho(u_i). \tag{3.5}$$

where ρ is the activation function of the neurons, akin to a firing rate. The original EqProp work defined ρ as a hard sigmoid function, but other functions have been used such as a sigmoid or a tanh [244]. This energy function has first been studied by [137], [245] , [246] and[247]. It differs from the original Hopfield energy, as the quadratic term $\frac{1}{2} u_i^2$ was not present.

In this particular framework, the neurons evolve during the free phase according to the

equation:

$$\frac{ds_i}{dt} = -\frac{\partial E}{\partial s_i} = \rho'(s_i)\left(\sum_{i\neq j} W_{ij}\rho(u_j) + b_i\right) - s_i. \tag{3.6}$$

The derivative of the energy function $\frac{\partial F}{\partial \theta}$ can be computed and gives:

$$\frac{\partial F}{\partial W_{ij}} = \frac{\partial E}{\partial W_{ij}} + \beta\frac{\partial C}{\partial W_{ij}} = -\frac{1}{2}\rho(u_i)\rho(u_j) + \beta\frac{\partial C}{\partial W_{ij}} \tag{3.7}$$

$$\frac{\partial F}{\partial b_i} = \frac{\partial E}{\partial b_i} + \beta\frac{\partial C}{\partial b_i} = -\rho(u_i) + \beta\frac{\partial C}{\partial b_i} \tag{3.8}$$

If the cost function partial derivative with regards to the weights and biases is zero $\frac{\partial C}{\partial W_{ij}} = \frac{\partial C}{\partial b_i} = 0$, then the parameter updates derived from equations (3.4), (3.7) and (3.8) yield:

$$\Delta W_{ij} = \frac{\eta}{\beta}\left(\rho(u_i^{\star,\beta})\rho(u_j^{\star,\beta}) - \rho(u_i^{\star,0})\rho(u_j^{\star,0})\right), \tag{3.9}$$

$$\Delta b_i = \frac{\eta}{\beta}\left(\rho(u_i^{\star,\beta}) - \rho(u_i^{\star,0})\right). \tag{3.10}$$

Note that these equations still hold true for any cost function, as long as this cost function does not depend on any of the parameters (weights and biases).

The first work to apply EqProp to a real system is the work of Kendall et al. [248], which proposes a learning framework where synapses are memristive devices and neurons' non-linearities are created by diodes. This work shows that this kind of system naturally evolves to minimize a quantity which is called the pseudo-power of the system, defined in the case of a linear resistance network (neurons connected by linear resistors) as:

$$P(V_1, ..., V_N) = \frac{1}{2}\sum_{i<j} g_{ij}(V_i - V_j)^2 \tag{3.11}$$

where the pseudo-voltage P is defined for any configuration of voltages V_i, even for the ones not compatible with Kirchhoff's laws.

This definition of pseudo power leads to the following weight updates:

$$\Delta W_{ij} = \frac{\eta}{\beta}\left(\left(\Delta V_{ij}^{\beta}\right)^2 - \left(\Delta V_{ij}^{0}\right)^2\right) \tag{3.12}$$

It is technically possible to use the true energy of a physical system for learning with EqProp. For electrical circuits, this may not be the optimal choice, and for our work we choose to use the energy function defined in Eq. 3.5.

Cost function choice As we have seen, the update equations presented above hold true for any cost function C as long as C does not depend on any trainable parameters. In the original

EqProp paper, the mean square loss (MSE) was used with [138]:

$$C = \frac{1}{2}||y - t||^2 \tag{3.13}$$

However, other cost functions can be used, such as the cross-entropy loss used in the work of Laborieux et al. [244].

Variants of Equilibrium Propagation

- The initial EqProp work proposed a time-dynamic evolution of neurons. It is worth mentioning that a discrete-time version of EqpProp also exists, introduced by Ernoult et al. [243]. This framework proposes to find the fixed point by iteration until convergence, instead of resolving the time-dynamic differential equation.

- EqProp is local in space, as a weight update only depends on the two neurons which are connected to that particular synapse. However, it is not local in time as the equilibrium state of the free phase has to be stored in order to enact the weight update at the end of the nudging phase. This is not ideal for hardware, and that is the issue solved by Continual EqProp [249]. In this algorithm, the weight updates happen continuously during the nudging phase, removing the need to store the equilibrium point. Basically the idea comes from the fact that $\Delta W_{i,j} = \rho(s_i^\beta)\rho(s_j^\beta) - \rho(s_i^0)\rho(s_j^0) = \frac{1}{T}\int_0^T \left(\rho(s_i)\dot{\rho}(s_j) + \dot{\rho}(s_i)\rho(s_j)\right)$. The update can then happen continuously during the nudging phase.

- The weights are symmetric, which can be an obstacle to hardware implementation. The work of Scellier et al. solves the weight transport issue with a vector-field approach [250].

- Initially, the EqProp algorithm didn't scale well, and difficult tasks like CIFAR 10 were hard to solve because of the length of computation times and degraded performance. Laborieux et al. proposed to use EqProp in CNNs, and instored three phases instead of one, by having a free phase at $\beta = 0$, then a first nudge phase at $+\beta$, and then a second nudge phase at $-\beta$ [244]. This improved the approximation of the derivative of the loss compared to the parameter, overall helping the performance. Laborieux and Zenke then proposed a complex version of EqProp, called Holomorphic EqProp, where βC is a slowing oscillating nudging force [251]. It can be noticed that this second work meets the first one in the case N =2. In particular, this means that there is no need for three phases in [244], two phases with a positive and a negative β are enough.

- Laydevant et al. also introduced a Binary version of EqProp with either binary neurons and full precision weights or with binary weights and binary neurons [252].

- Martin et al. also introduced a spiking version of EqProp, based on Continual EqProp's ideas [153]. The weight updates are $\rho(s_i)\dot{\rho}(s_j) + \dot{\rho}(s_i)\rho(s_j)$ as we have seen. However, the definition of the firing rate and its derivative are ambiguous for a spiking neuron.

In this learning rule, Martin et al. have decided to binarize ρ, meaning that if neuron i spikes, $\rho(s_i) = 1$, and if it doesn't spike, $\rho(s_i) = 0$. This greatly simplifies the learning rule. However, the computation of $\dot{\rho}$ is not obvious. In this case, the authors choose to compute this value by first using an integrator to compute the value $V_{LI} \sim \frac{\rho}{\gamma}$ where γ is the leak factor. Then a delay τ is introduced to compute $\dot{\rho} = \gamma \frac{\partial V_{LI}}{\partial t} \cong \frac{V_{LI}(t) V_{LI}(t-\tau)}{\tau}$. This value $\dot{\rho}$ has to be smoothed with a low pass filter to use in the learning rule.

- In EqProp, only neurons evolve according to some dynamics. Agnostic EqProp, introduced in 2022 by Scellier et. al. [253] also incorporate the synapses in the energy function, and a control knob enables the trainable parameters to remain fixed in the free phase, while enabling them to evolve freely in the nudge phase.

Hardware realizations of Equilibrium Propagation A few hardware realizations of training a dynamical system in-situ exist; firstly in the work of Dillavou et al . [254]. They realized a small resistive neural network based on digital variable resistances of 128 values, deeply inspired by Kendall's work [248]. To palliate the need to store the states of the free and nudged phases, the authors proposed to realize two identical neural networks, one for each phase. At the end of the simultaneous phases, both networks are updated. More precisely, they use a variant of EqProp, called Coupled Learning [255]. This training algorithm is an in-between between CHL and EqProp. During the nudging phase, the output units are completely clamped and don't move, unlike in EqProp. However, contrarily to CHL, the clamped state is not equal to the target, but simply closer to the target than the free phase result was. The update of the resistors is simplified, as only binary updates are applied to the resistors. Training on the iris dataset is successfully realized. However, this task is the largest one that can be realized on their hardware system. Scalability could be an issue with this approach due to the lack of non-linearity in the system.

Activity-difference energy minimization (MADEM) is a learning framework on a memristor crossbar array [256]. This implementation is realized on a chip that integrates complementary metal–oxide–semiconductor (CMOS) digital control circuitry with two 64 × 64 analogue memristor crossbar arrays made of tantalum oxide. In this system, the free phase is denoted as A_{free}, which is a column vector representing the activities of all neurons in the network, including the input, hidden, and output neurons. During the nudge phase of the operation, a bias is introduced at the end of the network while the activities of the input neurons are held constant. This setup utilizes a binary activation function in a discrete-time context, with a linear nudge term. The energy of the system is designed to only accommodate an Ising-like term. It shows Braille's word pattern recognition. Both the free phase and the nudge phase equilibrium points are obtained by iteration, no time dynamics is involved.

A recent study of Laydevant et al. implements the Binary EqProp algorithm on a physical

system [257]. More precisely, the authors use the binary version of the learning algorithm to train an Ising machine, in this case D-Wave. The application of biases for the nudging phase was not enough, as the system was stuck in its ground state following the initial free phase. To solve this particular issue, either simulated annealing or quantum annealing was used to obtain the nudge phase state.

3.3 Need for gradient discretization

3.3.1 Ideal synapse definition and physical constraints

As we saw in the first chapter, an ideal hardware synapse has to follow some requirements. Its strength should move linearly compared to a control parameter that will be tuned during learning. In our particular case, we want to use this algorithm with memristors. The strength of the synapse will therefore be either the resistance or the conductance of a device. Implementing both positive and negative weights can be done in two ways. Either the SET and RESET are symmetric, and the zero is considered to be between the ON and the OFF states; or two different devices can be used to implement either the positive (excitatory) and the negative (inhibitory) parts of the synapses.

Equilibrium Propagation gives a real continuous-valued update that has to be applied to a synapse. Updating the weights during learning can also be done in several ways. A voltage pulse whose amplitude is proportional to the update can be applied to a synapse, or several pulses of the same amplitude, proportional to the gradient, can be applied. In hardware realizations, it is much easier to apply several pulses of identical amplitude rather than to apply one pulse of variable amplitude. That explains why this option is favored in this work.

Energy efficiency is a constraint implying a limitation of the total number of pulses applied over the whole chip. In addition, the more pulses are applied to a single memristor, the less linear the conductance or resistance of this specific device is. So ideally, we also want to limit the number of pulses applied per memristor.

Different discretization strategies will be discussed in this chapter.

3.3.2 Methods

The results in this chapter are obtained using the time-dynamic EqProp framework presented above, in section 3.2. The algorithm is implemented in Python, using the numba just-in-time compilation framework to accelerate the computation time on CPU. The Modified National In-

stitute of Standards and Technology (MNIST) database is used, composed of 28x28 images of digits and their corresponding labels, ranging from 0 to 9. In this case, the pixel values, ranging from 0 to 255 are scaled between 0 and 1. The labels are one-hot encoded. The MNIST database is imported using the sklearn library, before being shuffled with a seed equal to 20. Then, the dataset is split into three different sets: the training set of size 56000, and the validation and test sets each of size 7000. The network used is a fully connected network with one hidden layer and dimensions: 784-512-10. If nothing is specified, the mini-batch size is 50. The free phase is made of 30 iterations, the nudge phase of 10 iterations, with a time step of 0.5. The nudging factor β is fixed at 0.4, and its sign varies randomly as suggested in Scellier's and Bengio's initial work [138]. The activation function is the hard sigmoid, with neuron states clamped between 0 and 1, as in the initial EqProp work. The weights are initialized uniformly using Xavier's formula: $W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ [70]. The biases are initialized at 0. Each result consists in an average over ten runs, and the standard deviation is also shown as a lighter area around the mean value.

3.4 Continuous-valued EqProp study

Before diving into different discretization strategies, we first study how the conventional Equilibrium Propagation algorithm behaves, by presenting an example of learning with this algorithm.

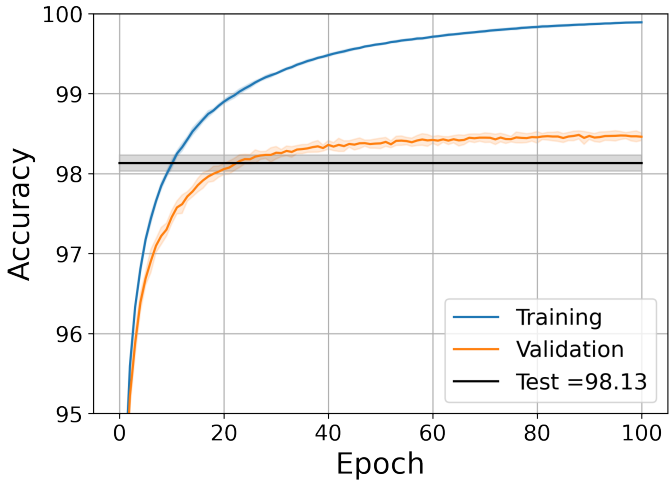


Figure 3.2: Accuracies obtained in the conventional EqProp case with parameters: $\eta_1 = 0.15$, $\eta_2 = 0.001$.

Figure 3.2 shows an example of accuracy obtained, which corresponds to 98.46 % for the validation accuracy and 98.13 % for the test accuracy.

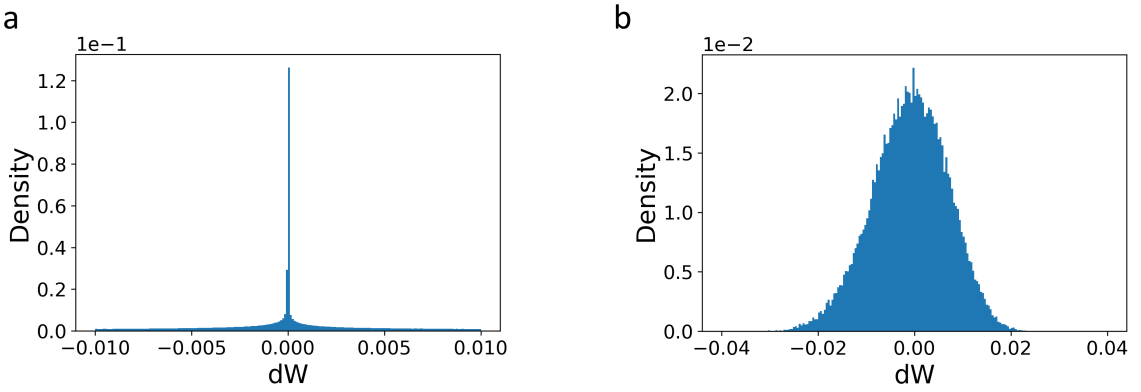


Figure 3.3: Histogram of the accumulated weight updates over 10 runs. a: Accumulated weight updates during learning for the first layer: ΔW_1^{tot}. b: Accumulated gradients during learning for the second layer: ΔW_2^{tot} with parameters $\eta_1 = 0.15$ and $\eta 2 = 0.001$.

To better understand the training process, it is possible to look at the distribution of accumulated gradients over learning. Figure 3.3 presents the update values $\Delta W^{tot} = \frac{\eta}{\beta}\frac{\Delta W}{n_{batch}}$ where n_{batch} is the batch size used (in our case 50, as written in the method section). On the left, the distribution of the accumulated gradient ΔW_1^{tot} during learning is plotted, corresponding to the weights of the first layer W_1. The right part shows the accumulated ΔW_2^{tot} during learning, corresponding to the weights of the second layer W_2.

Both figures show a peak around $dW = 0$. However, the shapes of the distributions are different in both cases, as the ΔW_1^{tot} distribution presents a high peak, whereas the ΔW_2^{tot} distribution presents a much smaller peak around $\Delta W^{tot} = 0$. This means that the weight matrix W_1 is more sparse than the weight matrix W_2. This is due to the dataset used, the MNIST one, where pixels in the border of the picture are equal to zero (black edge). Therefore the weights connected to these specific neurons are never updated. By comparison, the second weight matrix updates are distributed along a Gaussian, with a prominent peak at $dW = 0$.

This observation does not give any information on how many positive updates or negative updates have happened during learning. As a reference for future discretization techniques, it is interesting to accumulate separately all positive gradient values and all negative gradients over training. That way, it becomes possible to determine synapses that have never been updated. It also enables the comparison of positive accumulated gradients with negative ones. For a weight matrix W_i, we first computed the value $dW_{i,BL} = max(0, dW_i)$ (for i = 1 and 2) and accumulated over learning. Similarly, we then accumulated the value $dW_{i,BLb} = min(0, dW_i)$. The results of the positive and negative contributions are presented in Fig. 3.4.

Figure 3.4a shows the distribution of the accumulated positive $dW_{1,BL}$ at the end of learning, whereas Figure 3.4b shows the negative ones $dW_{1,BLb}$. A similar trend is observed in both

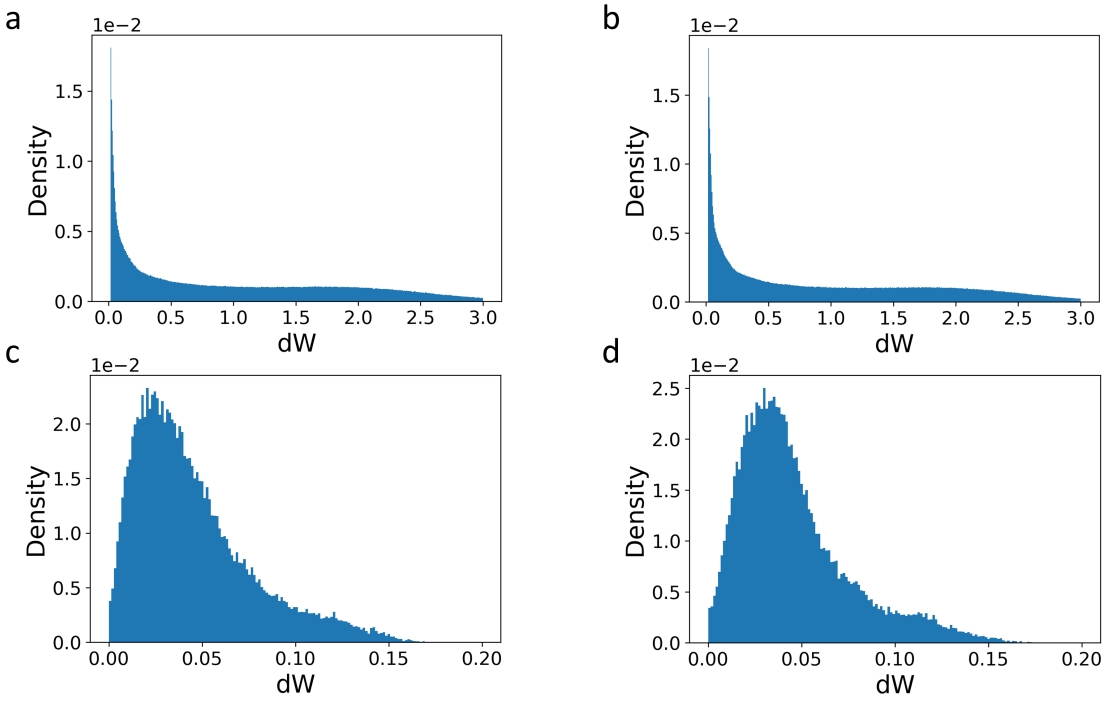


Figure 3.4: Histogram of the positive and negative weight updates. a: Accumulated positive weight updates for the first layer: $\Delta W_1^{tot,BL}$. b: Accumulated negative weight updates for the first layer (absolute value): $\Delta W_1^{tot,BLb}$. c: Accumulated positive weight updates for the second layer: $\Delta W_2^{tot,BL}$. d: Accumulated negative weight updates for the second layer (absolute value): $\Delta W_2^{tot,BLb}$. With parameters: $eta1 = 0.15$ $eta2 = 0.001$.

graphs with a peak at $dW = 0$, which steeply goes down at about $dW = 0.25$. The distribution then reaches a plateau before decreasing to reach approximately 0 at $dW = 3$. Overall both distribution evolutions are monotonous: they decrease when the value of the accumulated gradient increases.

Figures 3.4c and d display respectively the distributions of $dW_{2,BL}$ and $dW_{2,BLb}$. They both show a peak at $dW = 0$. However, the distribution evolution is this time not monotonous: the distributions increase before decreasing again to reach near-zero values at $dW = 0.16$. Similarly to the first layer (Figs 3.4a and b), the negative and positive contributions presented in Figs. 3.4 c and d have canceling effects that reduce the overall range of weights, shown in 3.3b. Note that contrarily to the first layer (Fig. 3.3a), this results in a Gaussian-like distribution.

Ideally, we want the discretization techniques to reproduce this type of distribution.

3.5 Discretization strategies

Discretizing the gradient necessarily introduces an error in the computation, as discretization necessarily implies that some synapses do not receive the exact gradient update that they should have had. The goal in designing a discretization strategy is, therefore, to minimize that error to obtain the best performance possible.

3.5.1 Ternary gradient

3.5.1.1 Definition

The first discretization strategy that we discuss is a simple ternary discretization of the gradient. In this specific case, the continuous-valued gradient is turned into -1, 0 or 1 "pulses", i.e. gradient steps: given a gradient ΔW computed by the EqProp algorithm, we would like to either apply one pulse to increase the weight, one pulse to decrease the weight, or no pulse at all. Therefore, a simple approach consists in defining a threshold θ_{th}. The weight update used is not directly the gradient but:

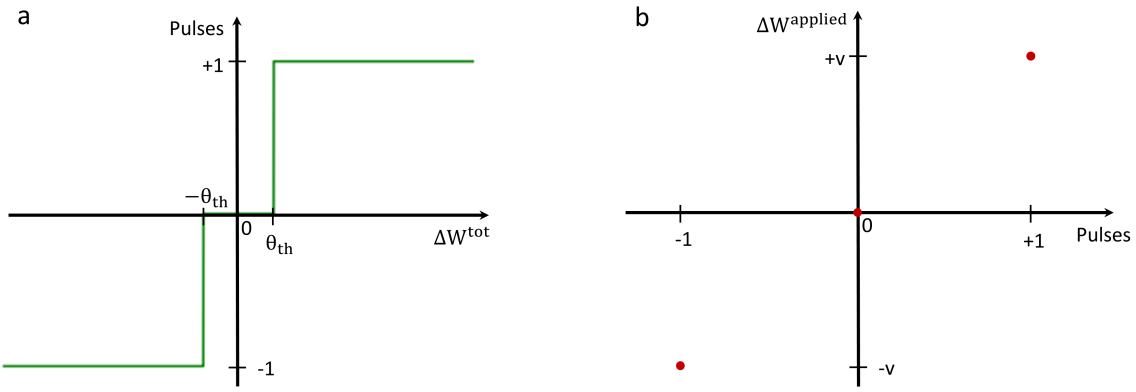


Figure 3.5: Ternary discretization a: Schematic representing the pulses as a function of the continuous-valued gradient, with threshold θ_{th}. b: Effective weight update as a function of the number of pulses.

$$\Delta W^{tot} = \frac{\eta}{\beta} \frac{\Delta W}{n_{batch}}, \tag{3.14}$$

which will be discretized. For all gradient values above the threshold θ_{th}, a positive step is taken, and for all values below the opposite of the threshold $-\theta_{th}$ a negative step is taken. Figure 3.5 evidences how the gradient is discretized.

For a real device, these pulses are directly applied to it. Let us first assume that the weight update after one step is determined by the "device speed" v:

$$W_{n+1} = W_n + \Delta W^{real}, \tag{3.15}$$

where

$$\Delta W_{real} = v \cdot t, \tag{3.16}$$

and where t is the number of pulses obtained with the discretization method. The speed v is a hyperparameter that would only be partially tunable in an experiment, as it would largely be determined by the devices' behaviors.

It is worth mentioning that the learning rate η has an influence on the number of pulses to apply. For instance, if η is large, the applications of pulses are favored as $\Delta W > \theta_{th}$ and will occur more easily. Note that the weight update step remains fixed at a value v and does not depend on the learning rate.

3.5.1.2 Results

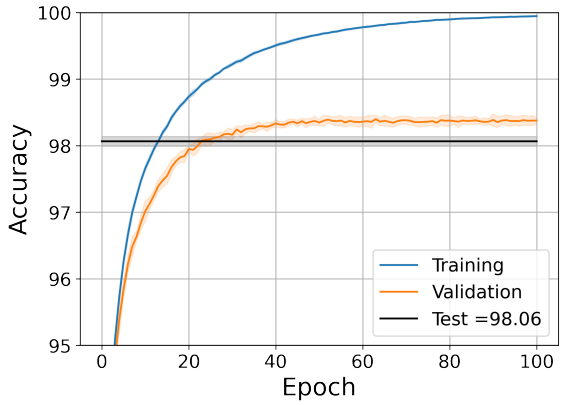


Figure 3.6: Performance of the ternary gradient method with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.00005, v = 0.00001$.

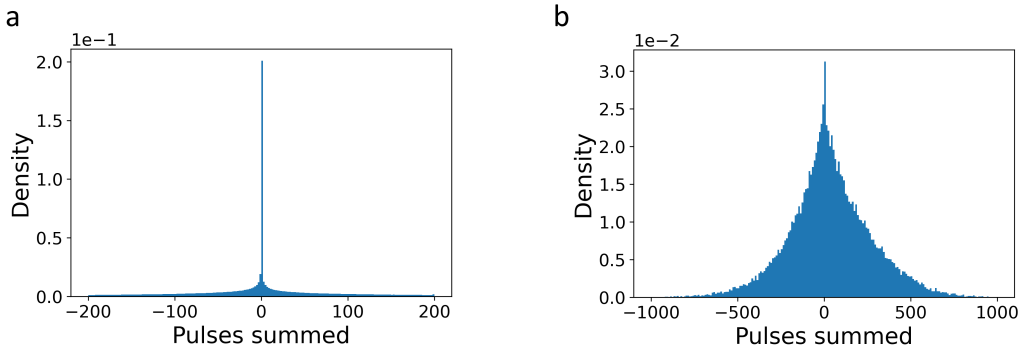


Figure 3.7: Cumulated update pulses distribution at the end of learning for the ternary gradient method. a: Cumulated pulses during learning for the first layer. b: Cumulated pulses during learning for the second layer. With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.005$, $\theta = 0.00005$, $t_{max} = 2$.

Figure 3.6 shows an example of accuracy obtained with this discretization technique. Surprisingly, the accuracy is only slightly degraded compared to the previous case (98.13%, see

Fig. 3.2) as we get 98.06 % accuracy. We follow a similar process to what was presented for the Eqprop algorithm. The accumulated pulses value t_{summed} is computed at the end of learning by summing the total number of -1 pulses t_{BLb} and +1 pulses t_{BL} applied to a given synapse ($t_{summed} = t_{BL} - t_{BLb}$). The results are shown in figure 3.7 a and b.

The accumulated weight updates of the first layer show a peak at zero values. This is reminiscent of the Eqprop algorithm previously presented in figure 3.3, even though the peak is more prominent in the current case. The accumulated weight updates of the second layer do not seem to follow a Gaussian-like distribution, which differs from the classical EqProp case (see Fig 3.3b).

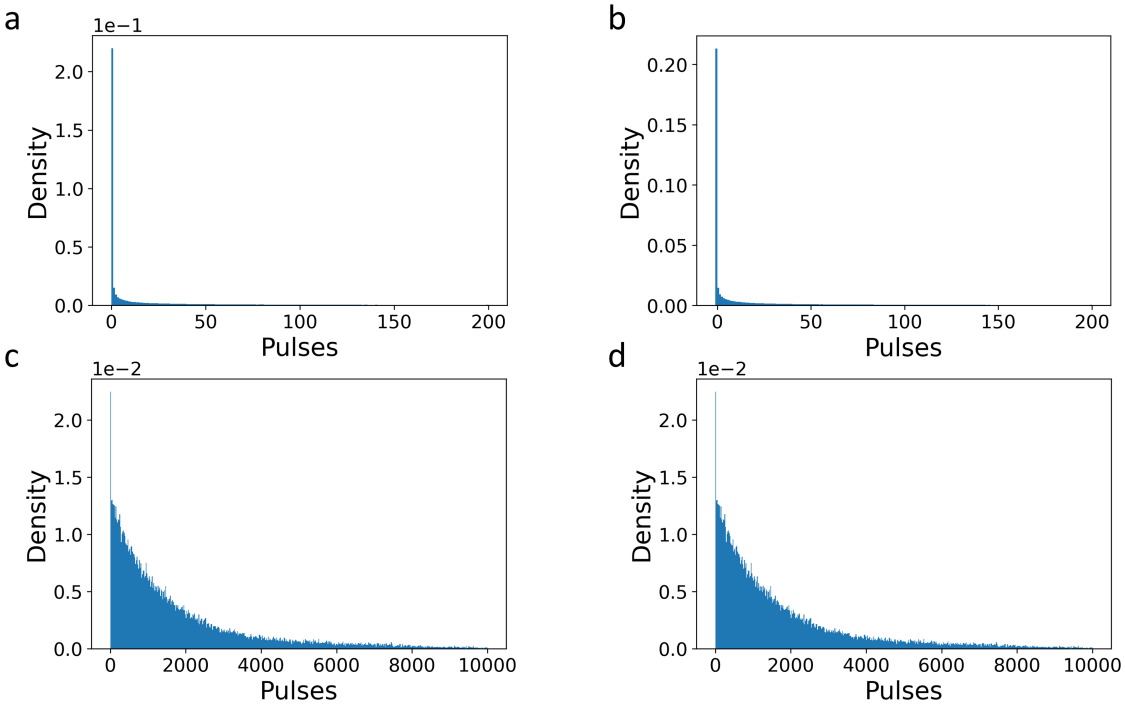


Figure 3.8: Cumulated positive or negative pulses distribution at the end of learning for the ternary gradient method. a: Cumulated positive pulses for the first layer. b: Cumulated negative pulses for the first layer. c: Cumulated positive pulses for the second layer. d: Cumulated negative pulses for the second layer. With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.005$, $\theta = 0.00005$, $t_{max} = 2$.

Figure 3.8 shows how the positive pulses and negative pulses are spread. The first layer (Figs. 3.8a and b) displays a monotonous repartition similar to the conventional EqProp case with a very high peak around $t_{summed} = 0$. The second layer differs again from the previous conventional EqProp case as it demonstrates a monotonous evolution of the updates.

3.5.1.3 Variation of the threshold parameter

The threshold parameter θ is the key parameter that governs the discretization step, which requires tuning as finely as possible to obtain the best possible performances.

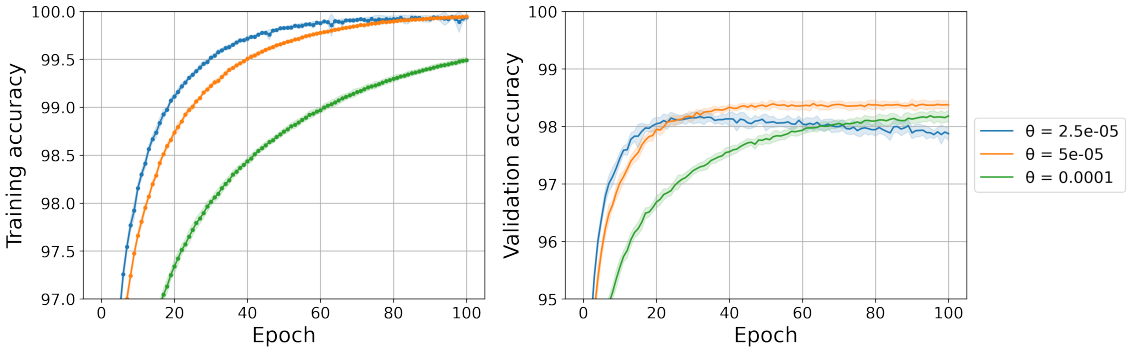


Figure 3.9: Performance of the ternary gradient method when the parameter θ varies with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \nu = 0.0001$.

Figure 3.9 shows the training and validation accuracy for three different values of the threshold θ, all averaged over ten different runs. The curve corresponding to $\theta = 5.10^{-5}$ (in orange) was previously shown in Fig. 3.6 and corresponds to the best performance. Higher θ values do not allow us to reach better accuracies (green), and the curve corresponding to $\theta = 2.5 \cdot 10^{-5}$ (blue) shows a degradation of training accuracy after 50 epochs.

These results can be interpreted in a straightforward fashion. If a high threshold is set, numerous synapses are never updated and the learning procedure relies only on a few synapses. On the contrary, if the threshold θ is too low, a large part of the synapses is updated, which deteriorates learning. To obtain an appropriate behavior corresponding to the proper balance between the two previous cases, it is thus necessary to tune the threshold value.

3.5.1.4 Variation of the learning rate of the second layer independently from the first layer

In this section, the learning rate of the first layer is considered fixed at $\eta_1 = 0.15$, and both the threshold θ and the learning rate of the second layer η_2 are allowed to vary. Modifying those two parameters allows us to explore the full parameter space. Indeed, the threshold and the two learning rates are related. For instance, if $\eta_2 < \eta_1$, the threshold value can be chosen to obtain a +1 pulse update in the first layer but not in the second one for an identical gradient.

Results for four different values of η_2 are presented in Fig. 3.10. The green curve for $\eta_2 = 0.005$ corresponds to the result presented in Fig. 3.6. When η_2 is high, the second layer is often updated, whereas if it is low, the second layer is scarcely updated. Similarly to the threshold parameter, a compromise has to be found here as well, as $\eta_2 = 0.001$ (blue) and $\eta_2 = 0.0075$ (red) result in a worse accuracy than the optimum value (green). Similarly as in the conventional EqProp case, the learning rate of the second layer η_2 is smaller than the one of the first

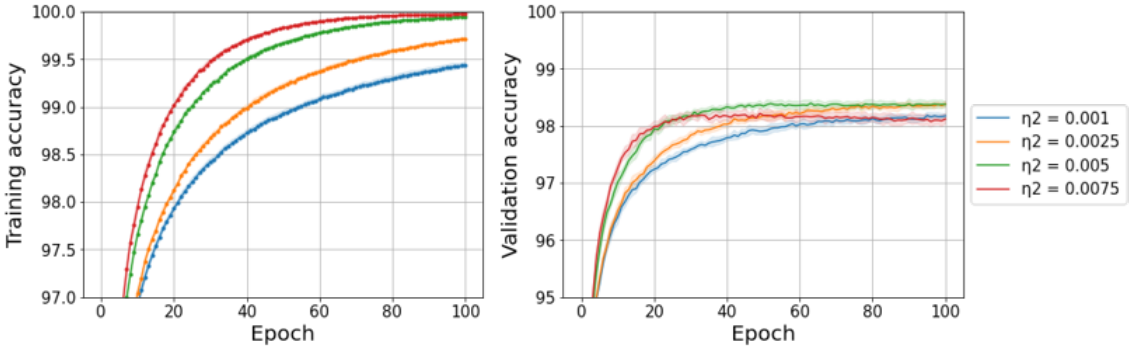


Figure 3.10: Performance of the ternary gradient method when the parameter η_2 varies with parameters: $\eta_1 = 0.15, \theta = 0.00005, \nu = 0.0001$.

layer η_1. This was already the case in the original EqProp paper [138], and corresponds to an amplification of the error gradients for layers further away from the output layer.

3.5.1.5 Variation of the speed v

Since we have decided to link the threshold parameter θ to the speed ν, for completeness of the study we will analyze the impact varying ν has on the overall performance.

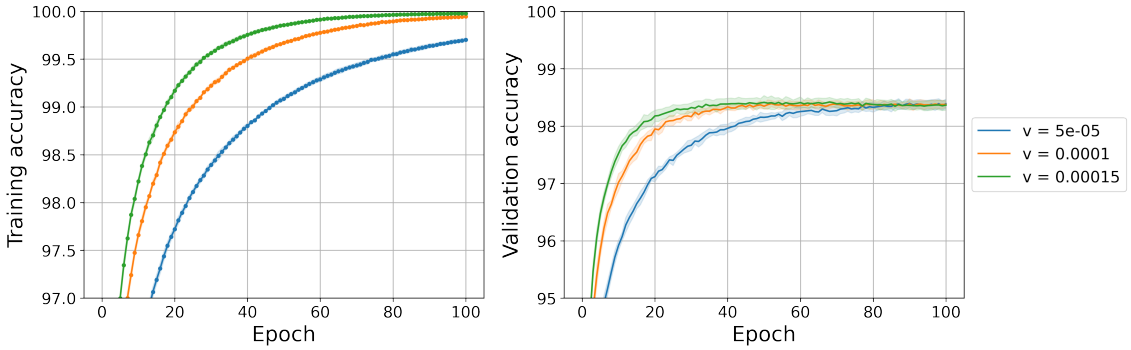


Figure 3.11: Performance of the ternary gradient method when the speed v varies with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.00005$.

The results presented in Fig. 3.11 where obtained with a threshold parameter $\theta = 5.10^{-5}$ (parameter used in Fig. 3.6). The orange curve shows the case where $\nu = 2\theta = 10^{-4}$, which corresponds to the results presented in Fig. 3.6. The green curve shows the case $\nu = 3\theta = 1.5 \cdot 10^{-4}$ whereas the blue curve presents the accuracy for $\nu = \theta = 0.5 \cdot 10^{-4}$. From both the training accuracy and the validation accuracy, we can observe that the higher ν is, the faster the increase in accuracy is. This was expected, as when ν is high, the weight increase after one update is consequently large. However, the validation accuracy at the end of training is similar in all three cases.

3.5.2 Ternary gradient with probabilistic updates

We saw in the previous section that the ternary gradient updates decreased the performance of the network. The distribution of pulses did not match the conventional EqProp case and spread along an important range of pulses. This degradation comes in part from the hard threshold that was put in place. All gradients above the threshold θ_{th} corresponded to an update, and all the others to none. To improve this result, another type of discretization is studied here, which we call the ternary gradient with probabilistic updates.

3.5.2.1 Definition of the discretization

To minimize the threshold effect, an idea is to introduce a probability of update instead of having a hard threshold.

Instead of only comparing the total gradient update ΔW_{tot} with θ_{th}, we will now define a probability p of having an update. This probability p is defined as follows.

$$If \ \ |\Delta W^{tot}| >= \theta_{th}, \ \ then \ \ p = sign\left(\Delta W^{tot}\right). \tag{3.17}$$

$$If \ \ |\Delta W^{tot}| < \theta_{th}, \ \ p = \frac{\Delta W^{tot}}{\theta_{th}}. \tag{3.18}$$

In these notations, the negative probability $p < 0$ corresponds to the probability of having a negative pulse.

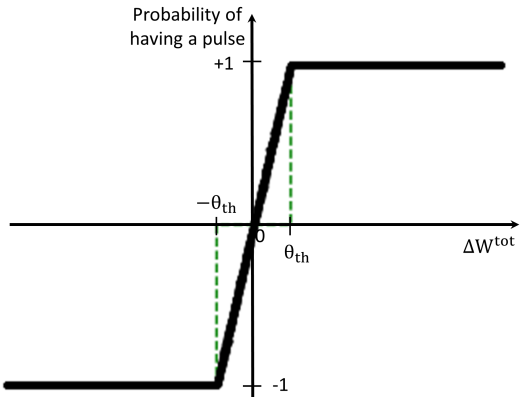


Figure 3.12: Probability of obtaining a pulse as a function of the continuous-valued gradient, with threshold θ_{th}.

The function p is plotted in Fig. 3.12. With the exception of the plateaux at ± 1 above the threshold, it yields a linear evolution between - 1 and + 1 for ΔW between $-\theta$ and $+\theta$ respectively.

3.5.2.2 Results

In this section, the same definition of the parameters is used compared to the previous section.

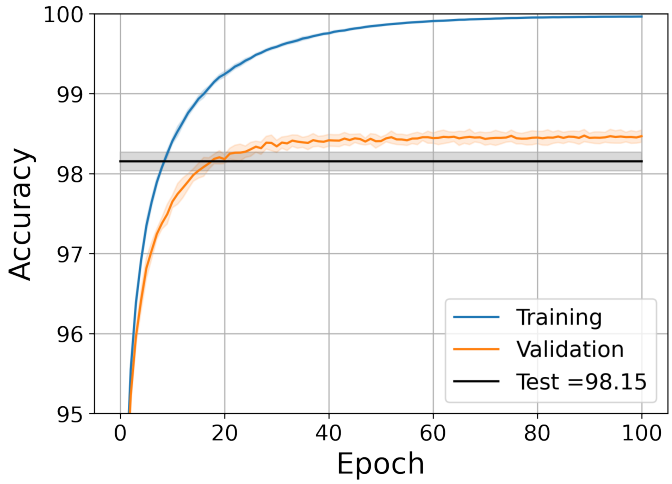


Figure 3.13: Accuracies obtained for the ternary gradient method with probabilistic updates with parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.0002, \nu = 0.0004$.

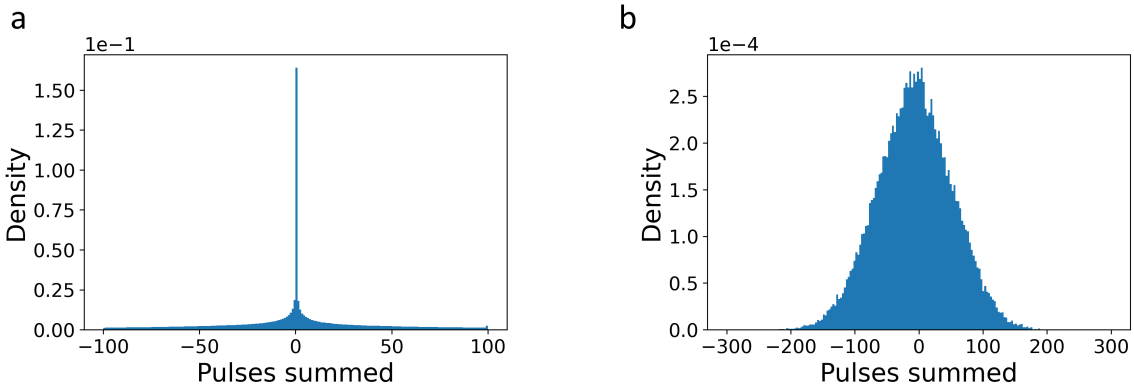


Figure 3.14: Cumulated update pulses distribution at the end of learning for the ternary gradient method. a: Cumulated pulses during learning for the first layer. b: Cumulated pulses during learning for the second layer. With parameters: $\eta_1 = 0.15, \eta_2 = 0.005, \theta = 0.0002, \nu = 0.0004$.

The results obtained with the probabilistic updates presented in figure 3.13 reach better accuracies than the non-probabilistic case: 98.47 for the validation accuracy and 98.15 % for the test accuracy. It is a very good performance for a one-hidden layer network, comparable to the full precision gradient one (see Fig. 3.2).

To continue our comparison between the full precision method and the different discretization techniques, the same graphs as the ones plotted in sections 3.4 and 3.5.1.2 are presented.

In figure 3.15, the distributions of the total number of pulses for the first weight matrix (a) and the second weight matrix (b) are shown. The overall shapes are much closer to the original EqProp case rather than the hard threshold discretization one. In particular, the distribution of the second weight matrix (Fig. 3.15b) is more Gaussian-like, even though it lacks the sharp peak at $t = 0$ present in the classic EqProp case. The pulses are distributed between -200 and 200, which consists in a much lower range than in the non-probabilistic case (-700 to 700).

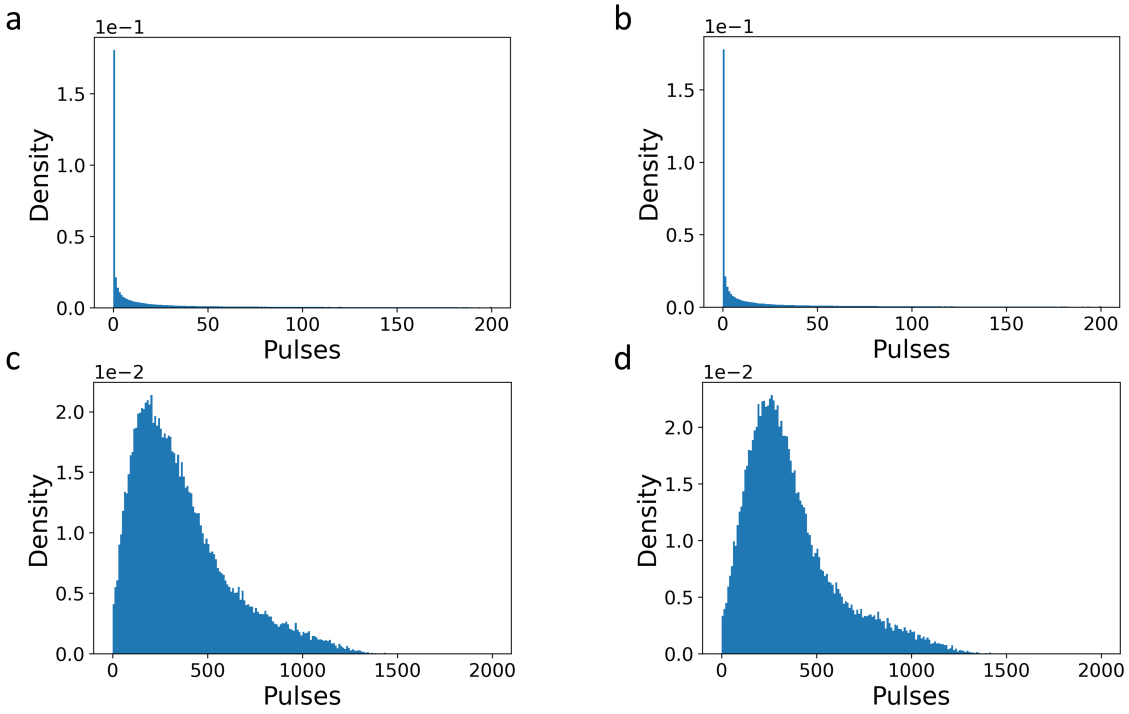


Figure 3.15: Cumulated positive or negative pulses distribution at the end of learning for the ternary gradient method. a: Cumulated positive pulses for the first layer. b: Cumulated negative pulses for the first layer. c: Cumulated positive pulses for the second layer. d: Cumulated negative pulses for the second layer.With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.005$, $\theta = 0.0002$, $\nu = 0.0004$.

Similarly, figure 3.15 shows the repartition of all the positive and negative pulses. This repartition is much closer to the original Equilibrium propagation algorithm. In particular, figures 3.15c and d show a non-monotonous repartition, which was not the case in the non-probabilistic case (see Fig. 3.7b).

In conclusion, the process of gradient discretization into a ternary representation, comprising only three distinct values, might have initially raised concerns about potential accuracy degradation. Nevertheless, it is noteworthy that while the discretization method without probabilities does indeed yield results inferior to the full-precision counterpart (98.06 % and 98.13 % of test accuracies respectively), it still demonstrates a level of performance that deems it suit-

able for practical application within real-world physical systems. The probabilistic approach gives better results (98.15 %), and the cumulated repartition of the updates is much closer to the full-precision one. However, this particular approach could be harder to implement in a physical system. The ideal implementation would be to use an intrinsic probabilistic property of the system instead of having to design outside circuitry to implement random number generation and computation.

3.6 Increasing the number of quantized values of the gradient

3.6.1 Presentation of the discretization step

Energy-wise, it might be more efficient to apply trains of pulses corresponding to higher precision gradients instead of single pulse sequences corresponding to ternary gradients. This can be numerically reproduced by setting multiple thresholds which will trigger a given number of pulses (see Fig. 3.16).

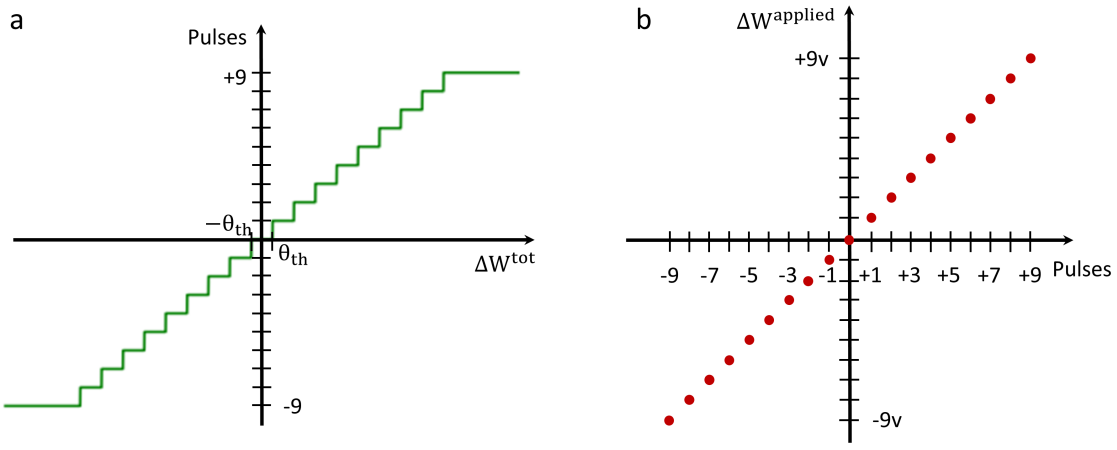


Figure 3.16: Quantized discretization. a: Schematic representing the pulses as a function of the continuous-valued gradient, with threshold θ_{th}. b: Effective weight update as a function of the number of pulses.

Because we have previously seen that the threshold effect could be an issue for learning, we chose to make the width of all steps identical.

3.6.2 Results

This section presents the results obtained using multiple quantized values of the gradients. The performance is shown in Fig. 3.17, highlighting an accuracy reaching 98.127 %, *i.e* an improve-

ment over the non-probabilistic ternary gradient performance (98.06 % of test accuracy).

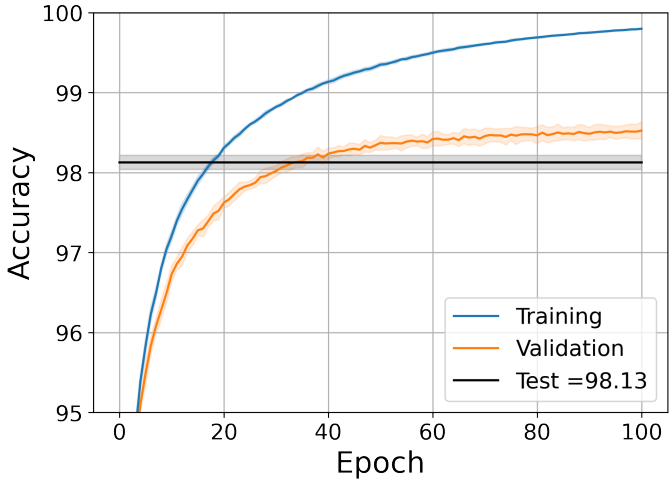


Figure 3.17: Accuracies obtained for the gradient method with a maximum of 9 pulses applied per synapse per update. The parameters are: $\eta_1 = 0.15, \eta_2 = 0.01, \theta = 0.0015, \nu = 0.003$.

The distribution of the pulses shown in Fig. 3.18 shows the same behavior as for the case of ternary discretization without probabilistic updates.

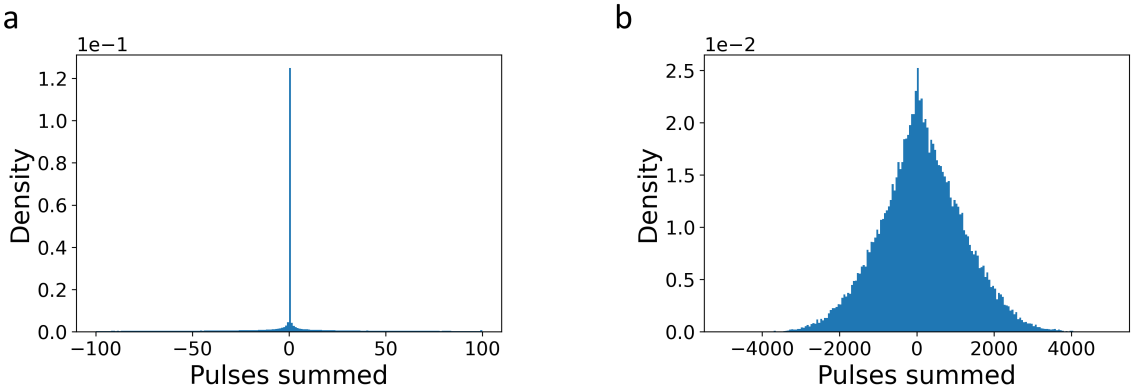


Figure 3.18: Cumulated update pulses distribution at the end of learning for the ternary gradient method. a: Cumulated pulses during learning for the first layer. b: Cumulated pulses during learning for the second layer. With parameters: $\eta_1 = 0.15$, $\eta_2 = 0.01$, $\theta = 0.0015$, $t_{max} = 9$.

However, one difference is that the distribution is more spread: a range of pulses going from -4000 to 4000 is necessary to observe the whole distribution (while -1000 to 1000 where enough before, as seen in Fig. 3.8). This discrepancy arises from the higher number of pulses that can be applied at each update.

The distribution of positive and negative pulses for both layers is presented in figure 3.19.

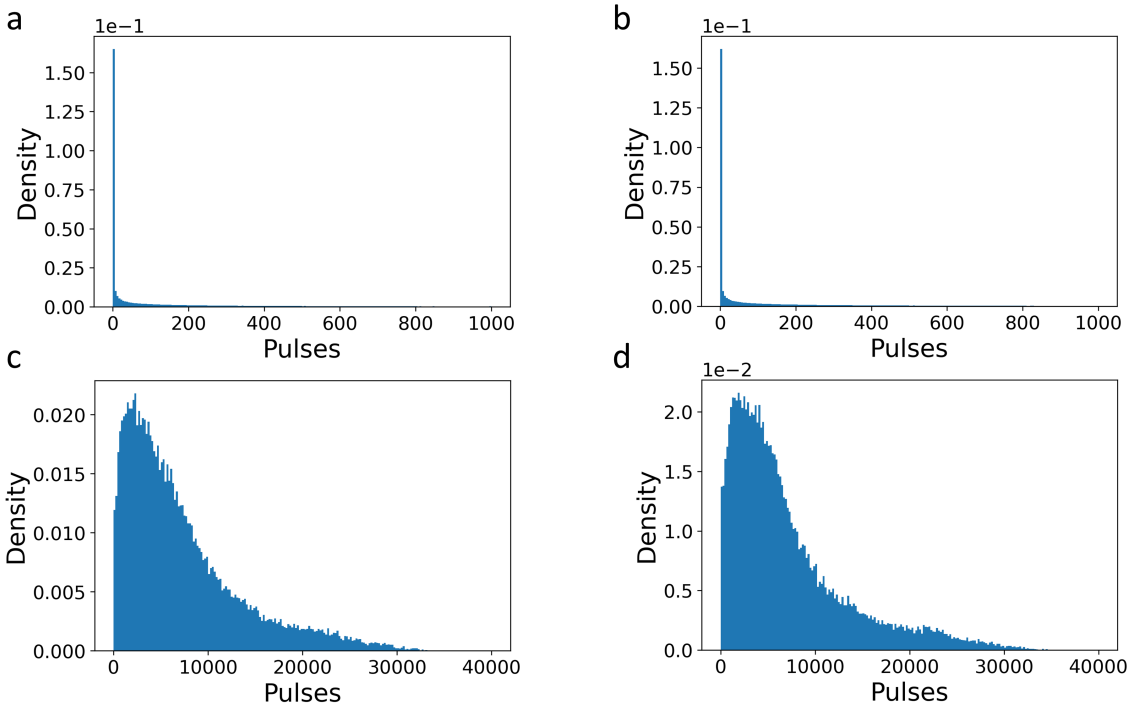


Figure 3.19: Cumulated positive or negative pulses distribution at the end of learning for the quantized gradient method. a: Cumulated positive pulses for the first layer. b: Cumulated negative pulses for the first layer. c: Cumulated positive pulses for the second layer. d: Cumulated negative pulses for the second layer. With parameters: $\eta_1 = 0.15, \eta_2 = 0.01, \theta = 0.0015, \nu = 0.003$.

It can be observed on this figure that the distribution of the cumulated number of positive or negative pulses applied per synapse is closer to the standard EqProp case (Fig. 3.4).

We can notice that the number of pulses has drastically increased when compared to the ternary case. This increase allows us to improve the performance and brings the pulse distribution close to the ideal case. However, it's important to note that this method demands a substantial number of pulses, exceeding 25,000, to achieve improved accuracy compared to the ternary case, which required more than 5,000 pulses. Interestingly, the ternary gradient with probabilistic updates outperforms this method, delivering a test accuracy of 98.15 % with just over 1,500 pulses.

3.7 Balancing pulse allocation for reliable physics-based computing

We have seen the impact of different strategies on the accuracy and distribution of the updates. It is also crucial to monitor the total pulse count and the maximum number of pulses applied

to individual devices. The objective is to minimize pulse usage while ensuring an even distribution across synapses, thereby mitigating the risk associated with excessive pulse application per device. This approach not only guards against potential device failure but also prevents synapses from transitioning into noisy or non-linear regimes.

Let us look at the maximum number of pulses applied to one synapse in layers one and two at the end of learning presented in three different cases: non-probabilistic ternary case, probabilistic ternary case, and non-probabilistic quantized case.

- **Ternary gradient:** The test accuracy is 98.06 %. For ten different runs, the maximum number of pulses at the end of learning for layer one averages 30,388, with 11,945 for layer two. The mean total number of pulses across these runs is approximately 2,921,073,702.6.
- **Ternary gradient with probabilistic updates:** The test accuracy is 98.15 %. Across ten runs, the maximum number of pulses for layer one averages 14,367, with 1,498 for layer two. The mean total number of pulses over these runs is about 1,578,167,171.9.
- **Quantized gradient:** The test accuracy is 98.13 %. In ten different runs, the maximum number of pulses for layer one averages 282,964, with 36,201 for layer two. The mean total number of pulses across these runs is approximately 37,837,052,959.1.

In the probabilistic case, the maximum number of pulses is significantly lower for both layers compared to the non-probabilistic ternary case. Specifically, it's about twice as low for layer one and almost eight times lower for layer two. On the other hand, the quantized case exhibits a higher maximum number of pulses compared to the non-probabilistic ternary case. Layer one has about nine times more pulses, and layer two has approximately three times more pulses. This trend is also reflected in the total number of pulses after learning. Overall, the probabilistic ternary case achieves the best accuracy with the fewest number of pulses.

3.8 Conclusion

In this chapter, we first presented the state of the art regarding Equilibrium Propagation and its hardware implementation. We discussed the need to discretize the gradient updates to realize a physics-based version of EqProp. Different strategies have been studied. Firstly, we chose to examine the performances of two distinct scenarios: one involving a maximum of one pulse per update and the other utilizing up to nine pulses. Our findings indicate a notable performance enhancement when employing the nine-pulse approach compared to a ternary gradient update. However, this improvement comes at the expense of a significantly higher pulse count—more than twelve times the amount of the ternary approach.

In the next chapter, we aim to implement Equilibrium Propagation with memristor synapses. Therefore, we must find a tradeoff between accuracy and energy efficiency. Our goal is to min-

imize both the energy consumption and the number of pulses applied to each device. With these objectives, the ternary approach emerges as the preferred one. We also investigated another strategy, consisting of introducing a probability of updating the synapses instead of having a hard threshold. This approach demonstrated superior performance and yielded a pulse distribution that closely resembled the gradients found in continuous-valued updates. This discretization technique stands out as the most efficient. However, it is worth noting that its hardware implementation is more resource-intensive and harder to realize due to the necessity of random number generation. That is why, in the next chapter, we opt for a ternary discretization method without probabilistic updates. This choice aligns with our objective of energy efficiency and minimizing the pulse count per device, striking a balance between computational efficiency and hardware simplicity.

Chapter 4

Implementation of Equilibrium Propagation With Memristor Synapses

Chapter 3 focused on adapting the Equilibrium Propagation algorithm to physical systems by discretizing the gradient. The effects were studied on ideal perfect synapses. This chapter aims at getting a step closer to on-chip learning by applying this algorithm to real memristor data.

4.1 Context

Memristors have gathered considerable attention due to their unique potential in neuromorphic computing. These nanoscale devices can mimic both data storage and processing functions reminiscent of biological synapses in the human brain. An interesting feature of memristor crossbar arrays is that they naturally implement Multiply and Accumulate operations thanks to Ohm and Kirchhoff's laws, as presented in Chapter 1. This makes memristor crossbar arrays appealing candidates for the hardware implementation of inference tasks.

However, transitioning from inference to learning with memristor arrays is not a simple task. Different challenges are involved. First, the lack of an adapted learning algorithm with local learning rules is an issue for on-chip learning, as presented in the first chapter. Moreover, if iterations of write and program cycles (called write and verify) are possible during inference to alleviate the impact of memristor imperfections, they would not be efficient for learning, as this would be detrimental to the spatial locality of the algorithm. This means that tackling the device properties becomes crucial in this case. In this work, we look into this challenge and we focus on oxygen-vacancy-filament-based metal-oxide-metal structures.

4.1.1 Non-linearity and asymmetry

One of the fundamental challenges of learning with memristors is an incomplete grasp of the precise physical mechanisms governing filamentary memristors. While significant strides have been made in elucidating their behavior, the exact details remain elusive. At a high-resistance state (HRS), the oxygen-vacancy filament does not connect both metal electrodes. Therefore, the conduction happens via quantum tunneling [258], which is highly non-linear. This results in non-linear responses to applied voltages, as shown in Fig. 4.1a.

A source of asymmetry and non-linearity arises from the inherent dissimilarity between the set and reset processes in filamentary memristors. During the set operation, filament formation is primarily controlled by a compliance current, which limits the formation of the filament [259]. In the absence of current compliance, a Joule-induced positive feedback loop can either burn the device or cause a permanent low resistance state (LRS) [260]. In contrast, a negative feedback loop during reset gives rise to a more gradual switching behavior [261]. The reset operation is governed by voltage, and no compliance current is needed. These divergent mechanisms give rise to a pronounced asymmetry between the two processes, as shown in Fig. 4.1a and d. There is no inherent reason for the voltages applied during set and reset to be equiva-

lent, leading to operational disparities that must be managed, as shown in Fig. 4.1a and b. The non-linearity is also quite apparent in Fig. 4.1b and c, which complicates programming during learning. Furthermore, the stability of the HRS presents distinct challenges. The HRS tends to exhibit greater noise compared to the LRS. The origin of this noise in the HRS, largely due to the presence of traps and defects in the active region of the device, is not readily apparent, further complicating efforts to maintain stable synaptic behavior.

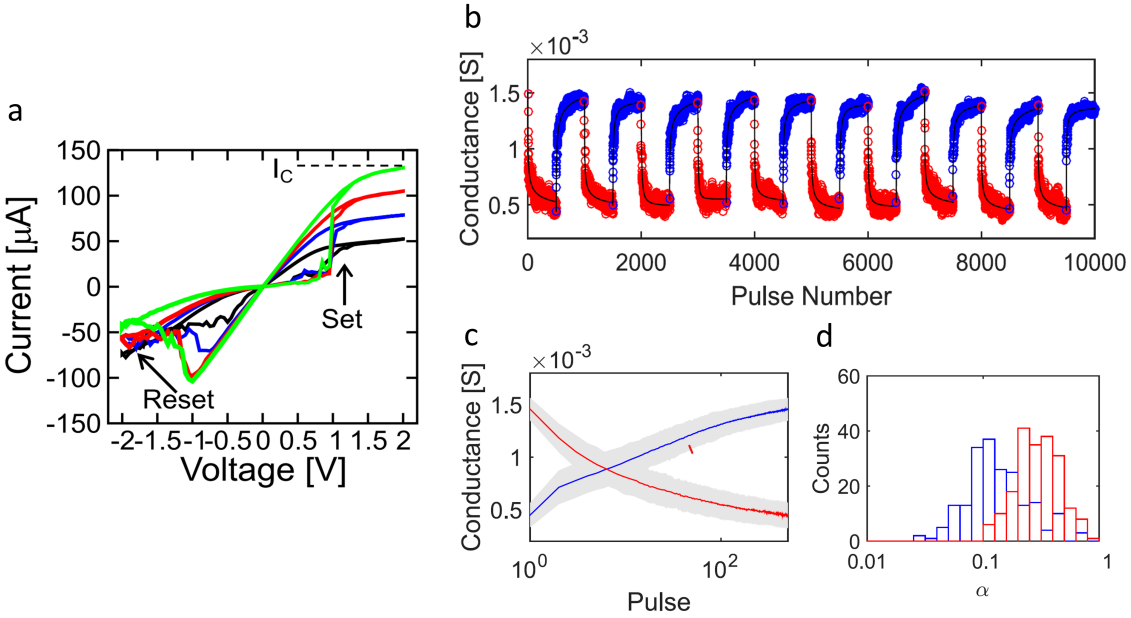


Figure 4.1: Asymmetry and non-linearity. a: Multilevel I–V characteristics of 1T1R RRAM TiN/HfO2/Ti/TiN device measured for increasing V_G, reproduced from [14]. b: Ten cycles programmed with 500 identical pulses of alternating depression and potentiation operations for a TiN/HfO2/Ti/TiN device with conditions ΔV and Δt are +0.9 V and 0.7 V. c: Evolution of the average conductance (straight line) and associated standard deviation (in grey) against the number of pulses for the data set in b. d: Distributions of the α parameter extracted from fit, where α is a multiplicative parameter that determines the magnitude of modification induced on the synaptic strength by a plasticity event. b,c,d reproduced from [15].

Both the asymmetry and the non-linearity are particularly relevant challenges when aiming at implementing memristor synapses that mimic the ideal synaptic devices, which was discussed in Chapter 1. The asymmetry prevents easy control of the increase and decrease of the weights, whereas the non-linearity causes the device's response to vary with its conductance.

4.1.2 Intra-device and inter-device variability

Variability is a common characteristic encountered in oxide-based memristors, constituting another challenge. This variability comes in two main forms: intra-device and inter-device variability, each with its distinct implications.

Intra-device variability pertains to variations within a single memristor device. It manifests in two primary ways: the noise in the resistance state and the cycle-to-cycle variability. The first form of intra-device variability is associated with the resistance state of a memristor when measuring current over time. This variation results from fluctuations in the charge near the filament inside the memristor. Here, figure 4.2a shows for example pink noise, characterized by its power spectrum that varies inversely with the frequency. Another example of noise is called Random Telegraphic Noise (RTN), illustrated in Fig. 4.2b, a random and abrupt fluctuation of electrical states in devices, where charge carriers randomly switch between discrete energy states due to localized defects or traps in the material [16]. The second aspect of intra-device variability involves cycle-to-cycle variability. This means that if a memristor is subjected to the same voltage pattern (whether it is a voltage sweep or a SET operation), the responses may differ across multiple cycles, as shown in Fig. 4.2c.

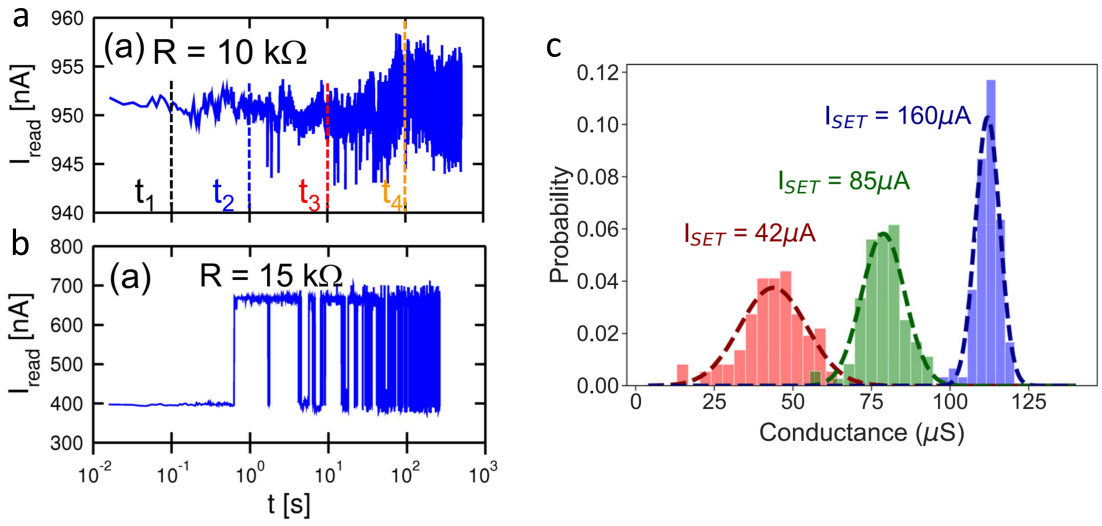


Figure 4.2: Intra-device variability. a: Pink noise in a device reported from [16]. b: Random Telegraphic Noise (RTN) in a device, reproduced from [16]. c: Cycle-to-cycle variability on a single device for different set currents, reproduced from [17].

Inter-device variability concerns variations of conductance observed among different memristor devices, in both the LRS (see Fig. 4.2a) and the HRS (see Fig. 4.2b). This variability includes fabrication-related variations originating during manufacturing and intrinsic variabilities inherent to the devices themselves, arising from their filamentary nature. Inter-device variability can result from differences in material properties, device geometry, or other factors that affect memristor behavior. Managing and mitigating inter-device variability is essential for ensuring consistent and reliable performance in neuromorphic systems.

Asymmetry, non-linearity, intra-device and inter-device variabilities all pose significant chal-

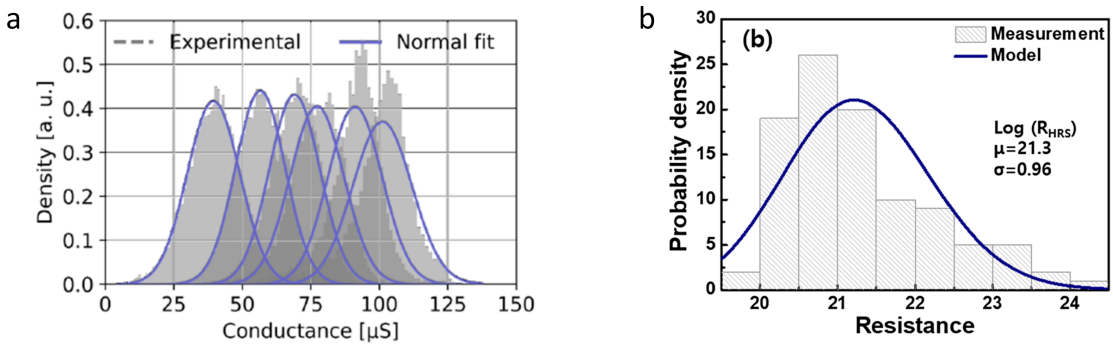


Figure 4.3: Intra-device variability. a: Variability of the conductance in the LRS measured on 16 384 devices under six different SET programming currents fitted with a normal distribution (blue line), reproduced from [18]. b: Variability in the HRS for 100 devices fitted with a log-normal distribution, reproduced from [19].

lenges when implementing hardware-based neural networks that can learn. Traditional deep learning algorithms rely on uniform and highly precise operations. The presence of variability, whether within a single device or across multiple devices, can contradict these requirements. As no model can reproduce all these different factors accurately, being able to test learning algorithms on real data, with real memristors, is a big step towards on-chip learning. This chapter will therefore focus on implementing the discretized version of Equilibrium Propagation presented in Chapter 3 with memristors.

4.2 Hardware platform

4.2.1 Presentation of the platform

This section presents the hardware platform designed by my colleagues Kamel-Eddine Harabi and Clément Turck, which was used in our experiments. This section is adapted from [262].

The primary objective of this platform is to facilitate the exploration of memristor behavior and support hardware-in-the-loop applications. To accomplish this, the platform can switch between two distinct operating modes: a digital mode and an analog mode. The platform features an array of 8k memristors, which were fabricated by LETI, and integrated in the backend of line of CMOS circuitry designed by our team and manufactured by a global foundry.

Fig. 4.4a provides a simplified schematic representation of a 1T1R cell, showing the integration of analog multiplexers for mode-switching. Each cell within the array is controlled by three mode-switching multiplexers: one for the source line, one for the bit line, and one for the word line. Additionally, the integrated circuit includes peripheral circuitry that facilitates memristor utilization in both digital and analog modes. Fig. 4.4b offers a simplified overview

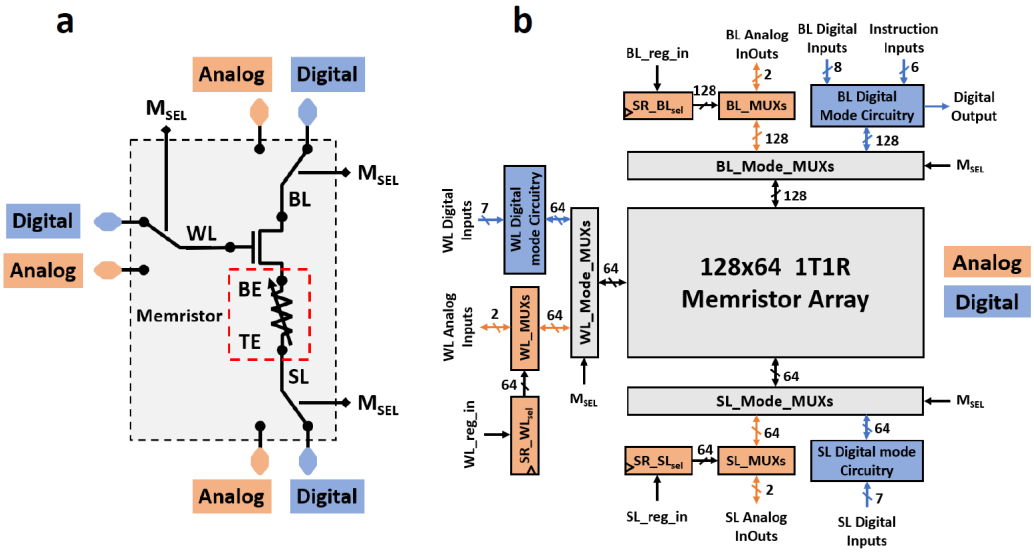


Figure 4.4: Fabricated Multimode Hybrid Memristor-CMOS Prototyping Platform. a simplified schematic of a 1T1R cell connected to analog multiplexers, illustrating the concept of switching the access mode. b schematic of the hybrid Memristor-CMOS die, consisting of two-mode circuitry: analog mode (orange color) supplied by nominal voltage VDD5, and digital mode (blue color) supplied by VDD, VDDC, and VDDR.

of the complete circuit, highlighting crucial components.

The grey-colored blocks in the diagram denote shared circuitry between both modes. Notably, this includes the 128x64 1T1R memristor array, consisting of 8,192 individual memristor devices, and the mode-switching multiplexers, designed using thick oxide transistors to ensure compatibility with high voltages. Conversely, the blue-colored blocks represent digital-mode circuits, designed using low-power, thin oxide transistors, and powered by digital nominal voltage (excluding level shifters). The orange-colored blocks represent analog-mode circuits, also designed using thick oxide transistors.

This thesis predominantly revolves around the utilization of the analog mode, rendering digital mode circuitry beyond the scope of this discussion. For comprehensive insights into the digital mode's functionalities, please refer to Kamel-Eddine Harabi's work [262, 263].

The integrated circuit layout and the optical microscopy image are presented respectively in Figs. 4.5a and c. Upon activation of the analog mode, the digital circuits are deactivated, and the connections within the memristor array are switched to the analog circuitry. In this mode, shift registers take on the role of configuring input multiplexers, enabling direct access to the analog state of the memristors (see Fig. 4.5b). Each word line, bit line, and source line is connected to one of two analog InOut Pads. In most cases, one of these analog InOuts re-

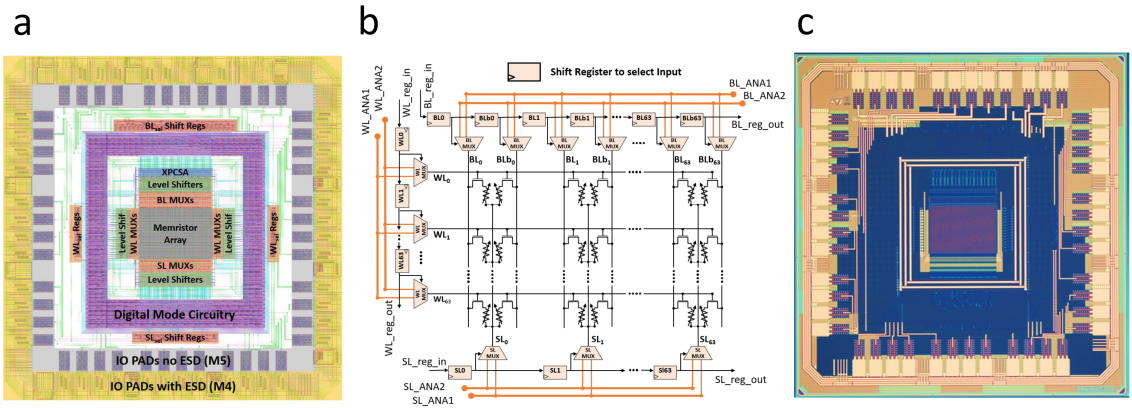


Figure 4.5: Fabricated Multimode Hybrid Memristor-CMOS Prototyping Platform. a layout view, b Schematic of the analog mode circuitry, with shift registers selecting inputs via Multiplexers, which consist of analog MUXs connected to SL, BL, and WL terminals. Each MUX is controlled by a shift register, to choose one of the two analog inputs. c Optical microscopy photograph.

mains grounded. These connections offer the flexibility to link the platform to external equipment, such as the Keysight B1530, a widely used pulse source and measurement unit, often employed for memory device characterization. Fig. 4.5b illustrates a sample configuration that enables the measurement of memristor conductance from the array. While operating in analog mode, the platform leverages the 8,192 memristor devices to function as analog storage units. However, due to the inherent limitation of having only two analog inputs, the array can execute analog Multiply-Accumulate (MAC) operations exclusively for Binarized Neural Networks (BNN). Therefore, the implementation of multi-level analog MAC operations is achieved virtually through computer-in-the-loop experiments, relying on measurements from each device.

This analog mode represents an important hardware platform, enabling the exploration of various aspects of memristor behavior and its potential applications, in particular for neuromorphic computing.

4.2.2 Memristors details

Memristors are fabricated on top of exposed vias and are composed of a TiN/HfOx/Ti/TiN stack. The active HfOx is deposited by atomic layer deposition and is 10 nm thick. The Ti layer is also 10 nm thick, and the memristor structure has a diameter of 300 nm. The structure of the stack is represented in Fig.4.6a. The scanning electron microscopy image of such a stack is shown in Fig. 4.6b.

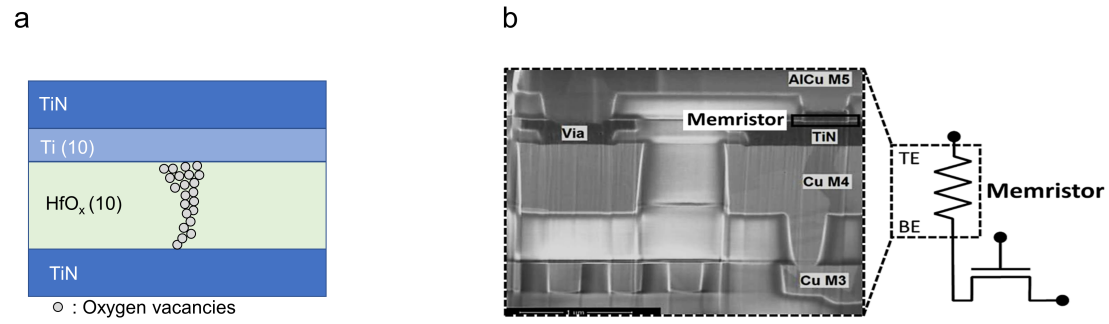


Figure 4.6: a Stack of the HfOx memristor used. b Scanning electron microscopy image of a memristor in the back end of line of the hybrid memristor/CMOS process, reproduced from [20].

4.2.3 Experimental setup

The experimental setup depicted in Fig. 4.7 is as follows: a PCB routes a microcontroller unit (STM32 F747ZGT6 on a test card Nucleo F747ZG) and measurement equipment (two DC Supply Sources and Keysight B1530) with the packaged die.

The first DC source applies VDD5 (5V) to the chip, whereas the other DC source applies VDD, VDDR, VDDC, all equal to 1.2 V. The WGFMUs of the Keysight B1530 is used as a voltage pulse source, and a measurement equipment at the same time.

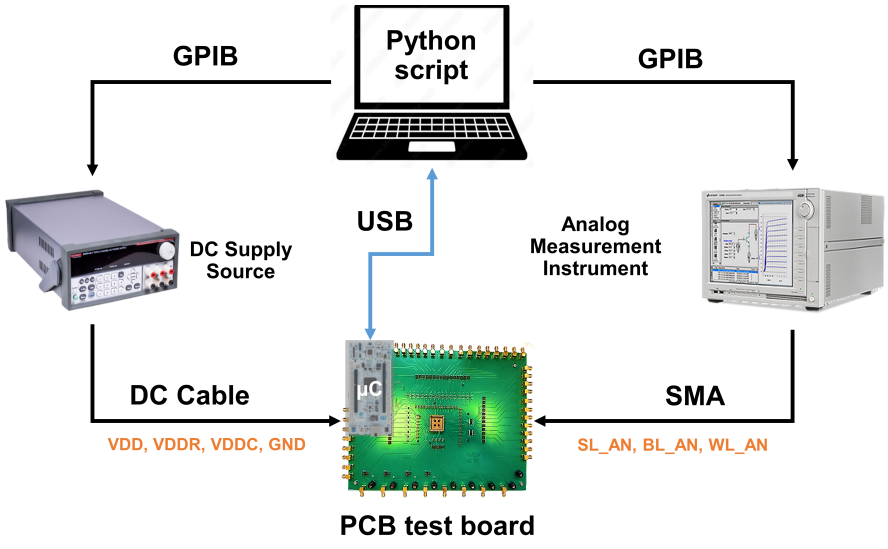


Figure 4.7: Experimental setup.

A code controls the microcontroller to program the set and reset configurations, presented in Fig. 4.8.

The voltage Vin1_col is always connected to the ground. The voltage Vin2_row will be referenced as V_g, because it controls the transistor gate, whereas the other voltage Vin2_col is

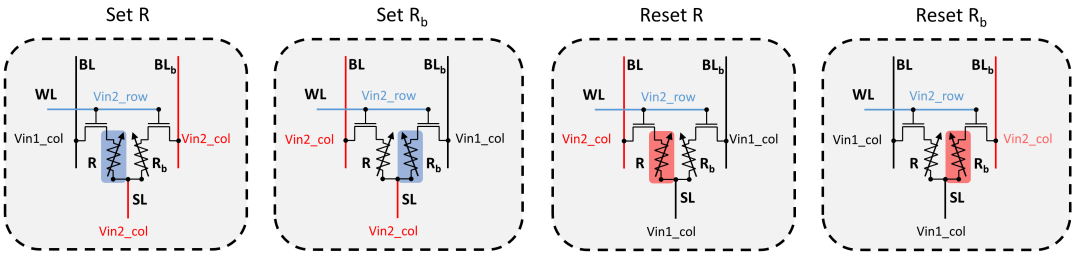


Figure 4.8: Set and Reset configurations

referenced as V_{col}. Depending if a measurement is in set or reset mode and a device R or R_b is selected, the microcontroller adapts the location of voltage applications.

4.2.4 Measurements

An electroforming step is necessary to induce the creation of an oxygen vacancy filament. After forming, the conductances have decreased from the megaOhms range into a few kiloOhms (typically 30 $k\Omega$).

Ten cycles of hard set and hard reset are needed to stabilize the switching of the devices. Parameter values are summarized in Table 4.1

	V_{col} (V)	V_g (V)	pulse length (μs)
Read	0.2	3	100
Forming	2.4	2.8	10
Set	2.0	2.5	50
Hard reset	2.0	4.5	1
Weak reset	0.9	4.5	0.6

Table 4.1: Parameters used during the measurements.

The weak RESET regime is characterized by a gradual decrease in the conductance of memristors when applying pulses. This phenomenon is shown in Fig. 4.9a, where the conductance decreases almost linearly with the number of pulses. In this work, we will only focus on this specific regime. However, not all memristors behave in such an ideal case. Figure 4.9b shows a case were a memristor reaches a noisy non-linear regime, corresponding to low conductances. The conductance evolution of 314 devices during weak reset is plotted in Fig. 4.9c. The variability of the devices is apparent here. Some conductance curves show a slight decrease, whereas others show a steeper slope. Low conductance regimes are very noisy, and would not be fit for computing.

These measurements are used throughout this chapter. They are used in the learning process to determine the evolution of the weights when pulses are applied to synapses.

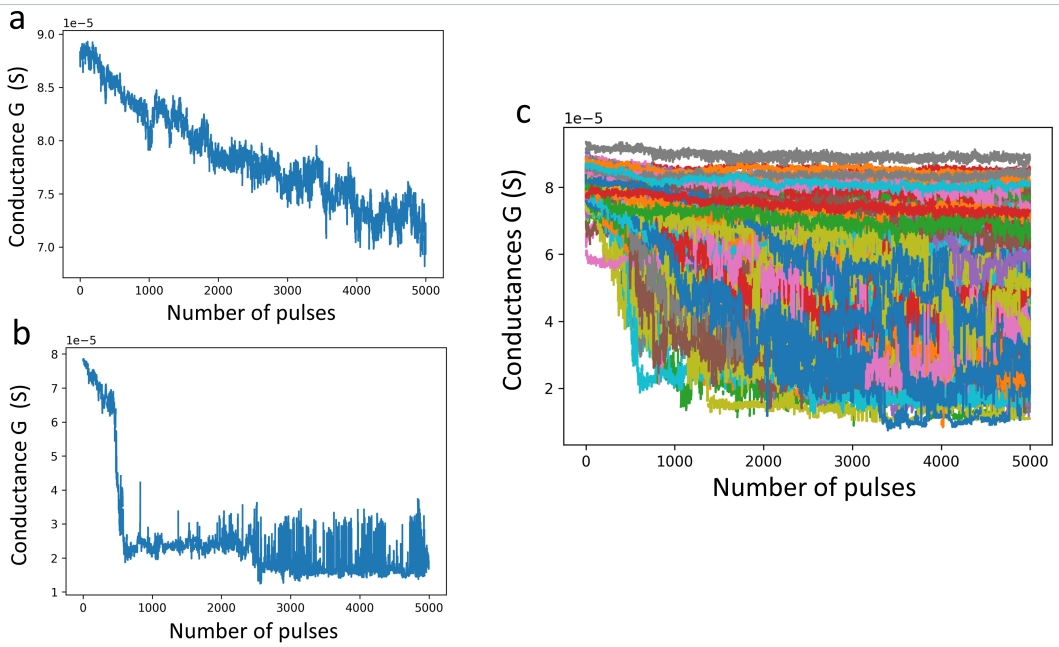


Figure 4.9: a: Example of synaptic plasticity in a memristor. b: Example of a non-linear regime in a memristor c: Conductances of 314 devices.

4.3 Setting the problem

4.3.1 What will be done in hardware, what will be done in software

The hardware platform presented in the previous section cannot be used to perform inference by taking advantage of Ohm's and Kirchhoff's laws as was explained in Chapter 1 section 1.3.2, as each device has to be accessed individually.

Our choice is to measure devices, store the measurements, and then use the experimental data to simulate the half-simulation-half-experiment learning process. In this process, neurons are fully software, and their states are computed similarly to Chapter 3. Synapses, however, are based on real experimental data, and gradient computations are made on a computer.

4.3.2 Definition of the weights

The weights in Equilibrium Propagation are fully software-based and have no dimension. When implementing this algorithm with real devices, there exist multiple choices regarding the synapse definition. As the weak set regime is complicated to obtain in our devices, only the weak reset regime is used. That necessarily means that two different devices have to be used per synapse. These two different memristors are named BL and BLb. Two different options are studied in this manuscript.

Linear difference of conductances:

$$W^l = \alpha^l \left(G_{BLb} - G_{BL}\right), \tag{4.1}$$

where l refers to the layer, W the weight, G the conductance, and α is a strictly positive floating number. In this case, the scale factor α has the same dimension as resistance and can compensate for the small values of the conductances. This definition is the typical one chosen for memristor crossbars.

Logarithmic difference of conductances:

$$W^l = \alpha^l \left(log_{10}\left(G_{BLb}\right) - log_{10}\left(G_{BL}\right)\right), \tag{4.2}$$

where l refers to the layer, W the weight, G the conductance, and α is a strictly positive floating number. Contrary to the previous, the scale factor α has no dimension here but can also scale the amplitude of the weights. This definition is the one chosen in Ref. [264].

The hyperparameter α can be different for both layers. It scales at the same time the values of the weights and the effect of a pulse on the weights' values. In both cases, when a pulse is applied to memristor BL, the weight increases, and when a pulse is applied to memristor BLb, the weight decreases. The choice of solely relying on the RESET process also means that when memristor BL or memristor BLb has reached its high resistance state, the value of the weight can not be tuned freely anymore. This limits the number of pulses that can be applied to memristors during learning.

4.3.3 Discretization and learning procedure

To tune the conductance of memristor devices, voltage pulses of identical amplitude are applied. Specifically, we select a range of -1, 0, or 1 as the number of pulses to use. Here, a "positive" pulse corresponds to the application of a pulse to the device BL of the synapse, whereas a "negative" pulse is applied to the BLb device of the same synapse. This choice necessitates a gradient discretization step to convert the continuous updates into discrete ternary values. This particular step has been discussed extensively in Chapter 3. A ternary approach without probabilistic updates, presented in section 3.6.1.1, is chosen in the following work, as it would be the easiest to implement in hardware. This approach in the case of ideal memristor devices is studied in section 3.6.1.2.

To test our idea, we use the following methodology. Each synapse gets randomly assigned two numbers between 1 and 314 (the number of memristors measured), representing the two

memristors and their corresponding experimental data. The weight initialization is determined by different elements: the measurement data, the random numbers assigned to each synapse, the scale factors α_1 and α_2 (corresponding to each layer), and the choice of weight definition (linear of logarithmic). A pulse tracking variable, called t, is initialized to zero for all devices.

Examples of handwritten digits are presented to the network, and both the free phase and the nudge phase are computed thanks to the weights based on real data. A continuous-valued gradient is obtained with the EqProp algorithm. This gradient is discretized into a number of pulses to apply. The pulse tracking variable t is updated by adding these new pulses. The conductance of each device is then read in the measurement data as G(t), and all weights are updated.

This process is repeated until the end of learning is reached.

4.3.4 Methods

The results in this chapter are obtained using the time-dynamic EqProp framework presented in the previous chapter. The algorithm is implemented in Python, using the numba just-in-time compilation framework to accelerate the computation time on CPU. The Modified National Institute of Standards and Technology (MNIST) database is used, composed of 28x28 images of digits and their corresponding labels, ranging from 0 to 9. In this case, the pixel values, ranging from 0 to 255 are scaled between 0 and 1. The labels are one-hot encoded. The MNIST database is imported using the sklearn Python library, before being shuffled with a seed equal to 20. Then, the dataset is split into two different sets: the training set composed of the first 56,000 images, and the validation set of the following 7,000 images.

If nothing is specified, the mini-batch size is 50. The free phase is made of 30 iterations, the nudge phase of 10 iterations, with a time step of 0.5. The nudging factor β is fixed at 0.4, and its sign varies randomly as in Scellier's and Bengio's initial work [138]. The activation function is the hard sigmoid, with neuron states clamped between 0 and 1, as in the initial EqProp work. Four random matrices D^1_{BL}, D^1_{BLb} of size 512x784 and D^2_{BL}, D^2_{BLb} of size 10x512 all composed of integers ranging from 0 to 313 are created to assign each synapse with two real memristor measurements. Biases are initialized at 0 and are not learned. Each result consists in an average over ten runs, and the standard deviation is also shown as a lighter area around the mean value.

4.3.5 Challenges

Different points have to be taken into account. All the different hyperparameters have to be tuned to reach the best accuracy possible, which is more challenging as more parameters exist in discretized EqProp.

For all 314 devices, we only have 5,000 datapoints. If, during learning, a memristor reaches

a number of pulses of 4,999, which corresponds to the last experimental data point, the device is not updated anymore. We call this phenomenon *saturation*, but it does not correspond to any physical mechanism. It is a simulation artifact that has to be avoided. During learning, this saturation is checked at each epoch. If it becomes non-zero, the computation is truncated as the following epochs are considered wrong.

4.4 Controls

First, different studies are performed to check the accuracy in the case where no error propagation takes place: the single-layer perceptron and the 3-layer network with a frozen first layer. All these simulations are performed with the linear conductance difference definition of the weights.

4.4.1 Perceptron

First, let us study the perceptron case. The network is a fully connected network of dimensions 784-10. The maximum purely software accuracy on MNIST for such a network is of 88 %. The best results for the hardware-based version is presented in Fig. 4.10.

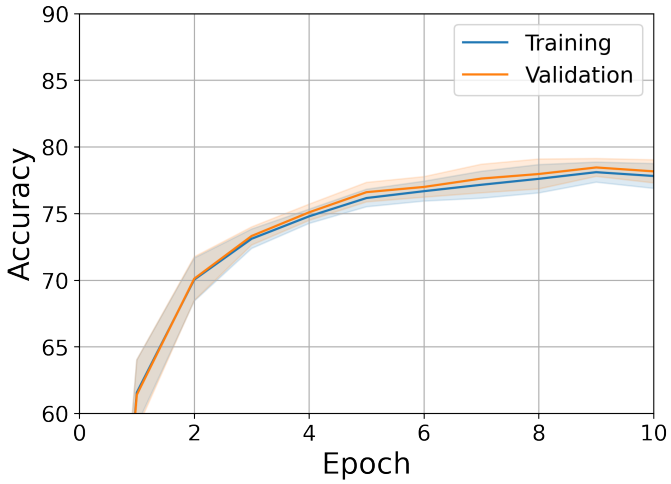


Figure 4.10: Performance for a perceptron on the MNIST task, obtained for a network of size 784-10 and parameters $\eta = 0.1$, $\alpha = 2000$, $\theta = 0.002$.

The best accuracy is then 78.1 %. This is worse than the ideal version, and this accuracy degradation is caused by inter-device variability and also arises from the noisy behavior of the memristors as a function of the number of pulses.

4.4.2 One-hidden layer network with first layer frozen

The network used is now a fully connected two-layer network with layers of dimensions: 784-512-10. All the weights are initialized as presented in the Method section (4.3.4), with experimental data. The learning rate of the first layer is zero, but the scale factor α_1 can be tuned. The result is shown in Fig. 4.11.

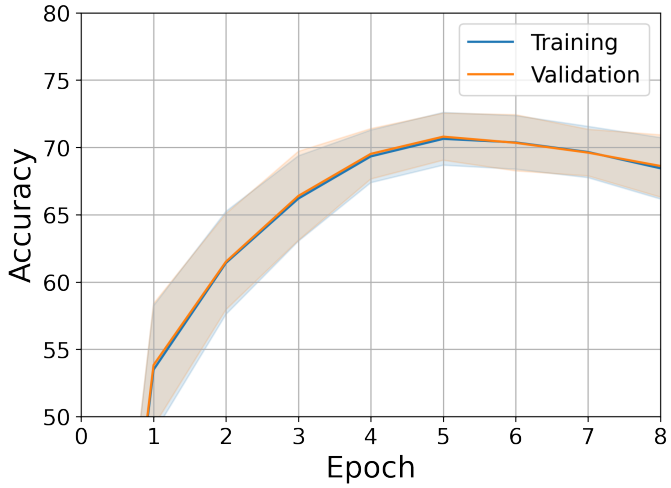


Figure 4.11: Performance obtained with a random frozen first layer with parameters$\eta_1 = 0.0$, $\eta_2 = 0.001$, $\theta = 0.5e-5$, $\alpha_1 = 4000$, $\alpha_1 = 3000$.

The best accuracy obtained is now 70.8 %, which is worse than in the perceptron case. This might be caused in parts because of the lower number of tunable synapses in this case than in the perceptron case. Contrary to the software case, the training accuracy does not always increase, as starting from epoch 5 both accuracies start to decrease. This is caused by some devices reaching the noisy or non-linear regime. After too many epochs, the learning can be considerably degraded because of this reason.

4.5 Results for a one-hidden-layer network

The network used is a fully connected one-hidden-layer network with layers of dimensions: 784-512-10. In this section, we set the learning rate η_2 to zero for the second layer. In this configuration, learning is still driven by error propagation, and the task of hyperparameter tuning is simplified. Specifically, modifying the scale factor of the second layer, denoted as α_2, solely influences the weight initialization of the second layer. Notably, this configuration yields the best results. Unless mentioned, the definition of the weights is the linear difference of conductances.

4.5.1 Accuracy obtained

The resulting accuracy, presented in Fig. 4.12 reaches 91 % accuracy, which is a clear improvement compared to the perceptron, and frozen first layer cases presented in the previous section.

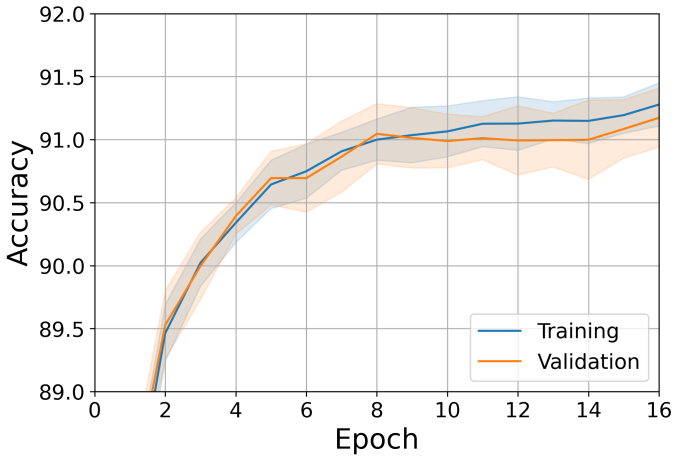


Figure 4.12: Best performance obtained with the weights defined as the linear difference of conductances and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000, \theta = 0.0002$.

This presents a significant achievement as memristors are very noisy and show a big inter and intra-device variability. Moreover, the non-linearity presents a hindrance to learning.

4.5.2 Hyperparameter tuning

To reach such an accuracy, hyperparameter tuning was necessary. The different roles and effects of the hyperparameters are presented in this section.

4.5.2.1 Threshold

The discretization parameter, the threshold θ, plays an important role in hyperparameter tuning. Figure 4.13 presents four different validation accuracies for different threshold parameters, corresponding to the range of thresholds that give good results. The results shown in Fig. 4.12, presenting the best performance, correspond to the orange curve $\theta = 0.0002$ in Fig. 4.13.

When the threshold is high (green curve, $\theta = 0.00025$), fewer devices are updated, and the devices updated are often the same ones. As a consequence, the validation accuracy does not reach a high value before the saturation issue comes up. However, because the same devices are always updated, the non-linear noisy regime limits the performance and the accuracy is

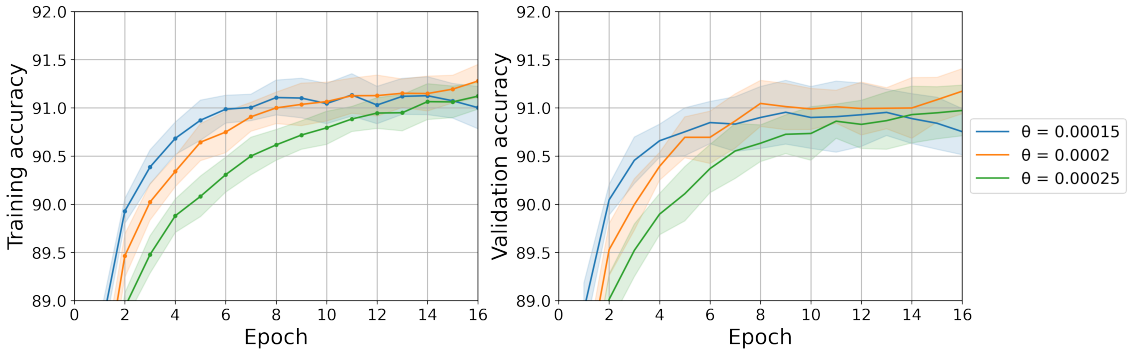


Figure 4.13: Impact of the variation of the threshold θ on the performance. The weights are defined as the linear difference of conductances and the parameters used are $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000$.

not as good as for $\theta = 0.0002$.

When the threshold is low (blue curve, $\theta = 0.00015$), more devices are updated at the same time with a single pulse. As most devices undergo the same update, the information of the free and nudge phases is not optimally learned by the network. As more epochs occur, numerous devices enter the non-linear and noisy regime, and the performance gets degraded. This explains the decrease in accuracy that can be observed starting from epoch 13.

4.5.2.2 Scale factors of the first layer

One of the differences between Chapter 3 and Chapter 4 is the introduction of scale factors in the definition of the weights. These are hyperparameters that need to be tuned. In this section, we study the influence of the scale factor α_1 on the performance of the network.

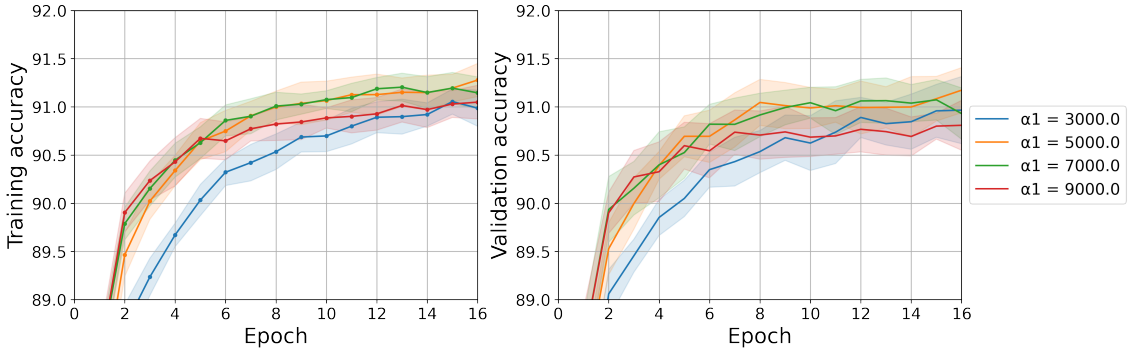


Figure 4.14: Impact of the variation of the scale factor of the first layer α_1 on the performance. The weights are defined as the linear difference of conductances and the parameters used are $\eta_1 = 0.2, \eta_2 = 0, \alpha_2 = 5000, \theta = 0.0002$.

The parameter α_1 scales the amplitude of the weights of the first layer. By doing so, it also changes the effect one pulse has on the weight value (this is referred to as the speed v in Chapter 3). In Chapter 3, the speed v was linked to the threshold parameter θ. Here, the speed between pulse t and pulse $t+1$ would be $v_1(t) = \alpha_1 \cdot (G(t+1) - G(t))$.

The results are shown in Fig. 4.14, presenting the range of α_1 giving good accuracies. The optimized scale factor α_1 corresponding to Fig. 4.12 is $\alpha_1 = 5000$ shown with the orange curve. For a fixed threshold θ, if the scale factor α_1 is high, then the weights have a big amplitude. This means that the continuous-valued gradients $\frac{\partial L}{\partial W_{ij}}$ get smaller, and after discretization, fewer pulses are applied. This leads to a smaller accuracy, as shown by the red curve corresponding to $\alpha_1 = 9000$. If the scale factor α_1 is too low, then the continuous-valued gradients $\frac{\partial L}{\partial W_{ij}}$ get bigger and this leads to a higher number of updates. If updates are applied to too many synapses, the network does not learn as much as in the previous case. This is shown by the blue curve corresponding to $\alpha_1 = 3000$.

4.5.2.3 Scale factor of the second layer

We previously fixed the learning rate of the second layer to zero. This makes tuning the scale factor of the second layer α_2 easier, as this parameter will only impact the initialization of the weights, which remain fixed during learning.

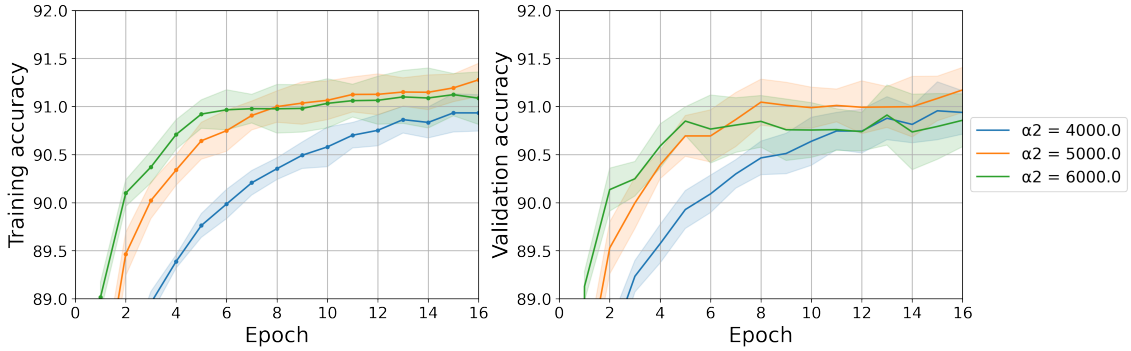


Figure 4.15: Impact of the variation of the scale factor of the second layer α_2 on the performance. The weights are defined as the linear difference of conductances and the parameters used are $\eta_1 = 0.2, \eta_2 = 0.0 \alpha_1 = 5000 \theta = 0.0002$.

The results are presented in Fig. 4.15, showing the range of α_2 that gives good accuracies. The result corresponding to the best accuracy presented in Fig. 4.12 is plotted here as the orange curve with $\alpha_2 = 5000$. When α_2 is low, the weights of layer two are small, and the error signal does not back-propagate as well (blue curve corresponding to $\alpha_2 = 4000$). This leads to a lower performance. When the scale factor α_2 is big, the weights of the first layer are updated more often, pushing them to the non-linear noisy regime. This explains the lower accuracy that

can be observed with the blue curve($\alpha_2 = 6000$).

4.5.3 Comparison between different definitions of the weights

Two different definitions of the weights have been presented in section 4.3.2: the linear difference of conductances and the logarithmic difference of conductances. Until this point of the study, only the linear difference of conductances has been considered. In this section, we compare the results of both definitions.

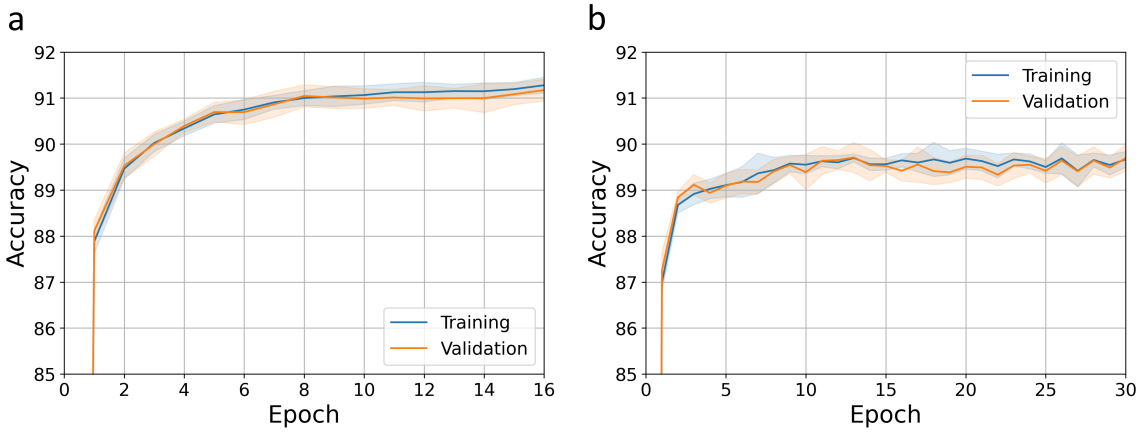


Figure 4.16: Comparison of the two different definitions of the weights. a: Best accuracy obtained with a linear definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000, \theta = 0.0002$. b: Best accuracy obtained with a logarithmic definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 1.2, \alpha_2 = 1.0, \theta = 0.0003$.

Figure 4.16a shows the previous result of Fig. 4.12 based on the linear difference of conductances. The accuracy reaches 91%. The results obtained for the logarithmic difference of the conductances are presented in Fig. 4.16b. The accuracy reaches 89.7 % which is significantly worse than in the linear difference case. This can be explained by considering that the undesired noisy non-linear regime, which we aim to avoid due to its negative impact on learning, primarily occurs at low conductance levels. Consequently, the use of a logarithmic weight definition tends to accentuate the distinction between memristors that remain outside this regime and those that enter it (for pulses less than 5000).

The experimental data of the non-volatile memristors used in our simulations is plotted in two different ways in Fig. 4.17 in order to highlight the point discussed above. Figure 4.17a presents the conductance G as a function of the number of pulses, whereas Figure 4.17b displays the logarithm of the conductance $log_{10}(G)$. When we examine the conductances following the application of 4999 pulses in both graphs, distinct behaviors emerge. When conductances exit the linear regime, their corresponding curves are notably closer to the bottom when

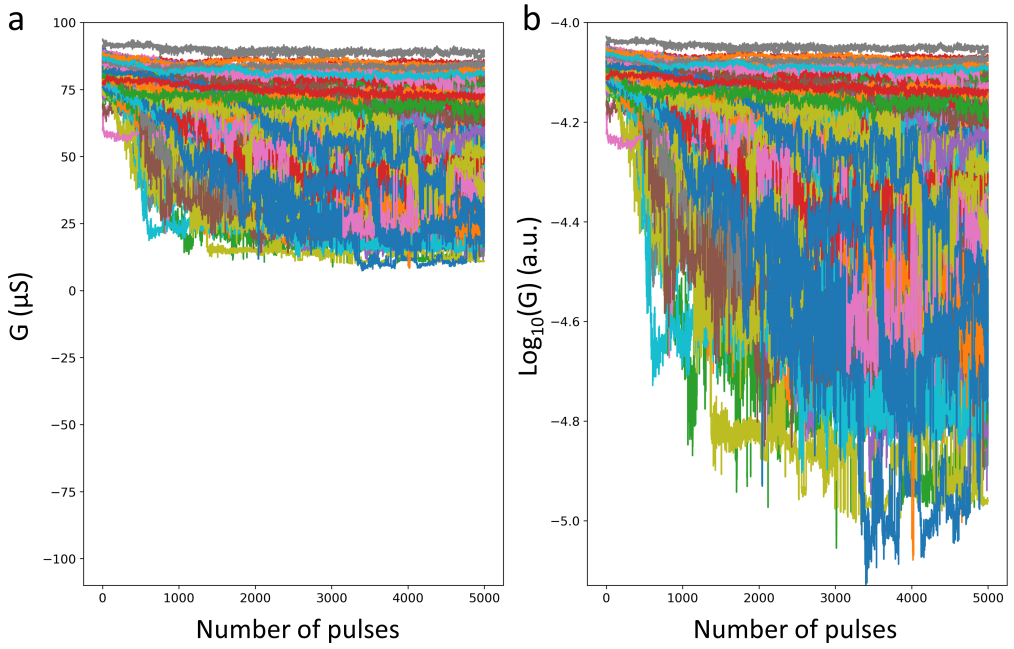


Figure 4.17: a: Plot of the 314 different conductances evolution. b: Plot of the logarithm of the 314 different conductances evolution. Both scales have been adjusted to align with high conductance states.

represented using $log_{10}(G)$ as opposed to the conductance G. These two scenarios show that the logarithmic representation amplifies the contrast in conductance behaviors.

4.5.4 Improving the accuracy

One challenge affecting performance is that devices enter a highly non-linear and noisy regime at low conductances. To enhance the accuracy, we propose the following strategy: if the algorithm has determined that one pulse has to be applied, the conductance of the corresponding memristor is read. If its value is higher than a certain threshold G_{th}, a pulse is indeed applied. If the conductance falls below this threshold G_{th}, no pulse is applied.

Results are shown in Fig. 4.18 for both the linear difference of weights (Fig. 4.18a) and logarithmic difference of weights (Fig. 4.18b) for three different thresholds: $G_{th} = \frac{1}{16000}\ S$ (green), $G_{th} = \frac{1}{18000}\ S$ (red) and $G_{th} = \frac{1}{20000}\ S$ (purple). Both results are obtained with the same conditions as in Fig.4.16 but yield different accuracies.

Let us first consider Fig. 4.18a. The threshold $G_{th} = \frac{1}{20000}$ gives a test accuracy of 91.5 %. The two other conductance thresholds $G_{th} = \frac{1}{16000}\ S$ and $G_{th} = \frac{1}{18000}\ S$ give almost the same accuracy: respectively 91.7% and 91.75%. For all three cases, introducing a threshold on the conductance value yields better results than in the case without a threshold where the accuracy was 91% (see Fig. 4.16a). However, changing the value of the threshold does not have a

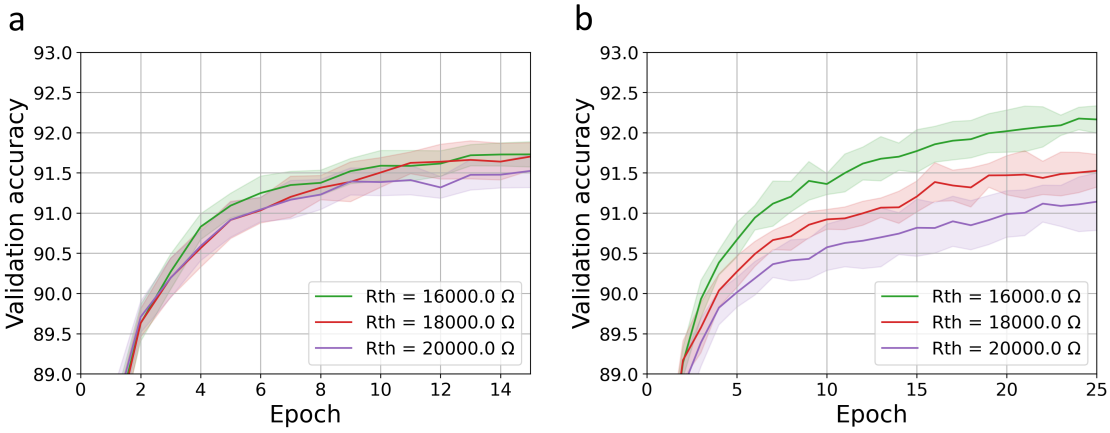


Figure 4.18: Comparison of the two different definitions of the weights when a conductance (or resistance) threshold is applied. a: Accuracies obtained with a linear definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 5000, \alpha_2 = 5000, \theta = 0.0002$. b: Accuracies obtained with a logarithmic definition of the weights and parameters $\eta_1 = 0.2, \eta_2 = 0, \alpha_1 = 1.2, \alpha_2 = 1.0, \theta = 0.0003$.

very significant impact on the resulting accuracy.

Let us now consider Fig. 4.18b, where the logarithmic difference of conductance is used. The accuracy with the same parameters was 89.5 % when no threshold was introduced (see Fig. 4.16b). Applying a threshold has a an important impact on the accuracy as with $G_{th} = \frac{1}{20000}$ the accuracy already reaches 91.14 % accuracy (green curve). Moreover, reducing the threshold improves even more the accuracy, as shown when the threshold is reduced to $G_{th} = \frac{1}{18000}$ (red curve) and the accuracy reaches 91.53 %. Further reducing the threshold keeps increasing the accuracy, as when $G_{th} = \frac{1}{16000}$ the accuracy reaches 92.16%, the highest accuracy obtained in our experiments.

Whereas the linear difference of weights gave better results when no threshold was applied, this is no longer the case when a conductance threshold is used.

4.6 Conclusion

In this chapter, we studied a realization of Equilibrium Propagation (EqProp) using memristor data. EqProp, a learning algorithm that takes advantage of the principles of physics, has emerged as a promising candidate for hardware implementation, as it solves the non-locality issue of BackPropagation. The challenge we addressed in Chapter 3 was the need to discretize gradient updates to suit hardware applications. This step is vital to bridge the gap between theory and practical implementation.

In this chapter, we demonstrated that thanks to an adapted hardware platform enabling individual access to memristors, we could exceed 91% on the MNIST dataset after hyperparameter tuning. The fact that learning was possible with very noisy devices showing a high variability and non-linearity underscores the viability of memristors as essential components in the pursuit of neuromorphic computing hardware. Two studies were performed to verify the benefits of error propagation in these networks: a single-layer perceptron which obtained 78.1 % accuracy and a two-layers network with a frozen first layer which gave 70.8 % accuracy. We have studied two different definitions of the weights called the linear difference of conductances and the logarithmic difference of conductances. We have found that the linear definition yields better results than the logarithmic one. To improve the previous results, we investigated the impact of limiting maximum resistance. We observed that such limitations exerted a more substantial influence on linear conductance differences compared to logarithmic ones. This observation paves the way for future optimizations in the design and utilization of memristive devices.

To further improve the results, different paths could be looked upon. The memristor technology used in this work has not been optimized for analog use. In particular, the linearity of the devices could be improved, which would give better performance. A linear weak set regime could also be optimized, which would mean only one device per synapse would be necessary, improving the scalability. Another approach would be to modify the EqProp algorithm to render it more robust against variability and noise. In sum, this work emphasizes the importance of a comprehensive hardware-software co-design approach for optimal performance.

Conclusions and Future Work

Summary

In our digital age, information technology has been a driver of global progress. Yet, it comes at a significant energy cost. The increasing demand for data-intensive services, such as artificial intelligence and cloud computing, has highlighted the urgent need for energy-efficient computing solutions that align with climate goals while fostering innovation in the field. Computers, the result of centuries of scientific evolution, have propelled us into an era of exponential growth in computational power and memory. However, new challenges are now present, in particular the soaring energy demands associated with the rise of deep learning.

Neuromorphic computing, drawing inspiration from the human brain, holds promise as an avenue for energy-efficient hardware. Learning algorithms and hardware neurons and synapses have to be co-designed to obtain the most efficient systems. Memristors, which exhibit compatibility with CMOS technology, offer efficiency, speed, non-volatility for synapses, and spiking behavior for neurons. The investigation of such emerging technologies and their compatibility with learning algorithms is an important step to realizing on-chip learning. In this work, we pursued diverse approaches:

- **Device Modeling and Characterization:** As Spiking Neural Networks offer the promise of low-energy learning, we explored emerging neuron devices as alternatives to CMOS. Scalable devices that replicate biological behaviors offer the potential to create intricate systems that mimic the brain. Spiking memristor neurons based on NbOx have been studied in Chapter 2.

- **Adapting Learning Algorithms for Hardware Implementation:** Back-Propagation is not an adapted algorithm for learning in hardware because of the non-locality of the updates. Equilibrium Propagation, an algorithm rooted in physics rather than calculus, opens the door to harnessing the inherent physics of hardware systems for on-chip learning. Adapting a pure software algorithm for future hardware implementation is a pivotal step toward on-chip learning, and is studied in Chapter 3.

- **Training Learning Algorithms with Real Hardware Data:** Real devices are characterized by noise, variability, non-linearity, and behavior that theoretical models often fail to capture. Thus, we emphasize the importance of working with experimental data to train neural networks, particularly when targeting on-chip applications. Chapter 4 aims at training a neural network where synapses are based on real devices, to pave the road for on-chip learning.

Chapter 2 focuses on the characterization and modeling of memristor neurons consisting of volatile NbO_x filamentary memristors. These memristors emerge as promising neuron candidates for several reasons, such as their scalability and their compatibility with non-volatile memristors and Complementary Metal-Oxide-Semiconductor (CMOS) technologies.

The $Pt/Nb_20_5/Ti/Pt$ stack demonstrates I-V characteristics corresponding to a current-controlled S-shaped Negative Differential Resistance and to voltage-controlled threshold switching, phenomena that can be effectively modeled by considering the Poole-Frenkel conduction. The dynamic properties of such devices are particularly interesting, as their spiking behavior is reminiscent of biological neurons. The shape of the spikes is characterized by an initial depolarization followed by hyperpolarization due to the presence of an inductance. These devices show behaviors such as Leaky-Integrate-and-Fire (LIF) characteristics, all-or-nothing-firing, and phasic bursting. We investigate the origin of this last behavior with a non-linear dynamics model. It emerges as a complex interplay between an unstable fixed point (limit cycle) and a stable fixed point (equilibrium), originating from the Poole-Frenkel effect. In this Chapter, we have both fabricated and characterized these neurons, and we have developed a simple model based on non-linear dynamics that accurately reproduces the neural behaviors mentioned above. This is a particularly interesting tool when designing spiking neuromorphic computing systems.

Chapter 3 takes a different approach to neuromorphic computing, this time exploring the adaptation of a learning algorithm, Equilibrium Propagation (EqProp), to physical systems. In particular, we focus on the challenge of handling continuous-valued gradients in a memristor-based environment, more adapted to trains of pulses. To tackle this challenge, we study different approaches to gradient discretization. We first choose to explore ternary discretization, where all synapses above or below a threshold are updated. This approach gives accuracies closely resembling those of the conventional EqProp algorithm (99.06 %). During this exploration, we examine the role of hyperparameters and their impact on the network's performance. Another approach we explore involved introducing probabilities into the update process. This modification not only enhances performance but also results in a pulse distribution similar to the ideal non-discretized scenario. To further analyze the impact of discretization, another approach is taken consisting in more quantized states for gradient discretization. While this path shows potential and surpasses the non-probabilistic ternary approach in performance, it comes with a trade-off – a wider pulse spread and ultimately, more energy consumption than the probabilistic ternary approach.

In Chapter 4, we now focus on hardware synapses for learning. To do so, we use a hardware platform consisting of HfOx filamentary memristors. These devices, which can be individually accessed, serve as the foundation for experiment-based learning, whereas neurons remain purely software-based. Firstly, control simulations are performed, beginning with the evaluation of a single-layer perceptron and a two-layer network with frozen weights in the first layer. These initial experiments yielded accuracy rates of 78.1% and 70.8%, respectively. In a second time, we train a one-hidden-layer fully connected network. Two distinct weight definitions are studied: the linear difference of conductances and the logarithmic difference of conductances. We obtain that the linear definition outshines its logarithmic counterpart, with 91% accuracy rate compared to the latter's 89.5% respectively. To improve these results, memristors

are then only written when their conductances falls is above a threshold, to prevent memristors from reaching a low-conductance high-noise non-linear regime. With this method, the linear weight definition reaches 91.75% accuracy, while the logarithmic version soars to an impressive 92.14%, constituting our best result.

Perspectives

While EqProp demonstrates spatial locality, there is a need to consider the temporal aspect, particularly in storing the states of the network during different phases and nudging phases. This is a significant challenge when implementing EqProp in hardware. In Chapter 4, we showed the possibility of implementing EqProp with real memristor synapses, a critical step towards hardware realization. However, it is important to note that in this implementation, neurons were still software-based. A potential avenue for further research is to explore a fully hardware-based implementation that includes both memristor synapses and neurons. Another intriguing prospect is the EqSpike algorithm already mentioned in this thesis (sections 1.2.2.2 and 3.2) [153]. EqSpike offers the advantage of being both temporally and spatially local, making it an appealing variant of EqProp. This algorithm requires spiking neurons for a full hardware implementation. Chapter 2 presented a memristor-neuron capable of numerous spiking behaviors, and it introduced a simple model that could exhaustively reproduce the real physical behaviors. Therefore, it would be valuable to explore the integration of EqSpike with memristor neurons using this model. This would mean first exploring the frequency response of memristors, which could provide insights into their dynamic behavior, further enhancing our understanding of these devices and their potential for neuromorphic computing applications.

Frequency response of a memristor neuron

This section provides work on the same NbOx neurons as described in Chapter 2, but the frequency response of these memristors is here studied. The experiments presented here have been realized by an intern, Thomas Bersani–Veroni.

Method

In this section, memristors are voltage-controlled utilizing a triangular voltage input generated through with an Agilent instrument.

Figure 4.19a shows the circuit used. The load resistance R_{load} is $4k\Omega$. To accurately determine the current passing through the memristor, we use an oscilloscope of impedance $R_{out} = 50\Omega$ to measure the voltage. Furthermore, to ensure precise interpretation of the results, we took measures to eliminate any inductance components from our setup, therefore removing the hyperpolarization-like characteristic of the spike shape.

An example of measurement is presented in Fig. 4.19b. It is worth noting that the slope

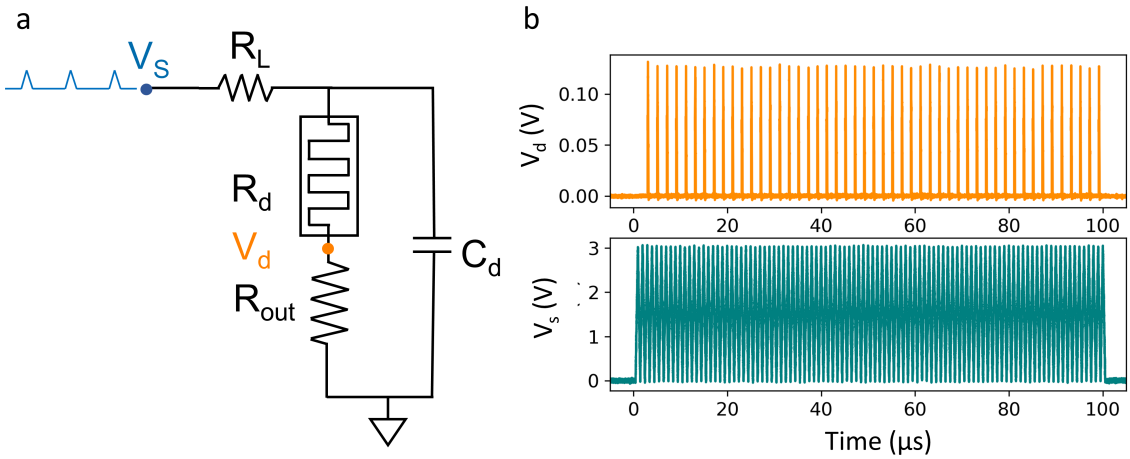


Figure 4.19: a: Schematic of the circuit used in these experiments.b: Example of result for $V_{s,peak} = 3.0$ and an input frequency of 1 MHz.

of the triangular waveforms was deliberately constrained due to the DC setup's limitations in handling extremely rapid voltage variations. The rise and fall times of the triangles have been fixed to $0.5\mu s$, hence limiting the maximum frequency of the input to 1 MHz.

Experimental results

In this section, we present the outcomes of our experiments designed to investigate the relationship between input and output frequencies in our memristor-based system.

We examine how the output frequency f_{out} varies as a function of the input frequency f_{in} for different peak-to-peak voltages. The results are presented in Figure 4.20. This analysis is crucial to understanding the behavior of our memristor-based system when subjected to different input conditions. Initially, for low input frequencies and for all voltage conditions, the device does not spike. As the frequency increases, the device frequency becomes non-zero and increases. Because of the limitation in frequencies (due to the shape of the input), both curves in Figs 4.20a and b seem truncated. On the other hand, for higher voltages as the ones presented in Figs. 4.20c and d, the output frequencies ends by matching the input one. An interesting feature can be observed in Fig. 4.20c, where between 0.45 and 0.6 MHz for z voltage of V = 3.5 V, the slope is of 0.5. This means that for every two input pulses, an output pulse occurs. This phase locking behavior is the one displayed in Fig 4.19b.

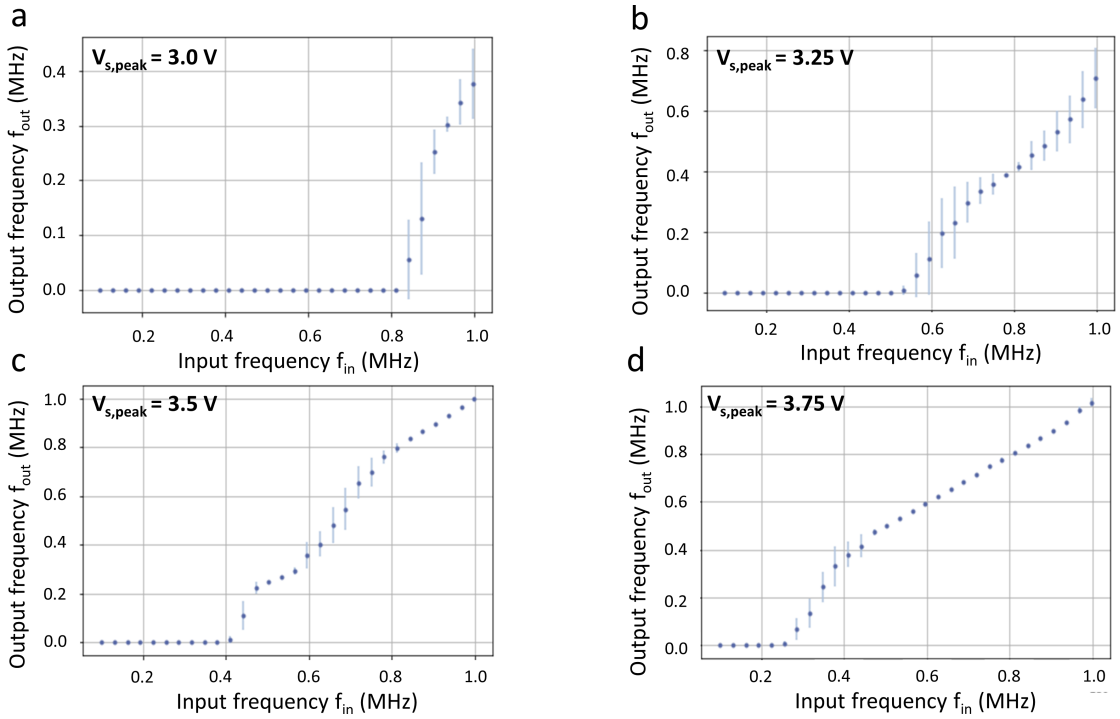


Figure 4.20: Variation of the output frequency f_{out} as a function of the input frequency f_{in} for different peak-to-peak voltages.: $V_{s,peak} = 3.0V$ b: $V_{s,peak} = 3.25V$ c: $V_{s,peak} = 3.5V$ d: $V_{s,peak} = 3.75V$

Simulations

To better understand the phenomenon presented in Fig. 4.20, simulations are performed using the model presented in Chapter 2 and the parameters presented in Table 2.1. A triangular input voltage of amplitude V_{max} is applied with a frequency f_{in} called the input frequency. The output frequency is measured for each different input frequency. The results are plotted in Fig. 4.21. Figure 4.21a shows the output frequency as a function of the input frequency, whereas 4.21b shows the frequency ratio $\frac{f_{out}}{f_{in}}$.

The resulting curve is akin to a devil's staircase which can be found both in oscillator or neuron models [265] [266] [194] , in electronics [267] or condensed matter physics [268]. The plateaus correspond here to fractional numbers $\frac{1}{3}$, $\frac{1}{2}$,$\frac{2}{3}$, $\frac{3}{4}$, $\frac{4}{5}$, $\frac{5}{6}$, $\frac{6}{7}$, $\frac{7}{8}$, $\frac{8}{9}$, and demonstrate a frequency lock-in of the output frequency depending on the input frequency.

The output frequency increases with the input frequency, and different modes or phase locks-in can be observed. These results show that our nanoscale neuron is capable of exhibiting sophisticated spiking neuron behaviors when using spiking inputs. This suggests that it is a viable choice for implementing advanced spiking neural network concepts, such as EqSpike. The next step towards a fully memristive implementation of EqSpike would then be to try to learn using our memristor model.

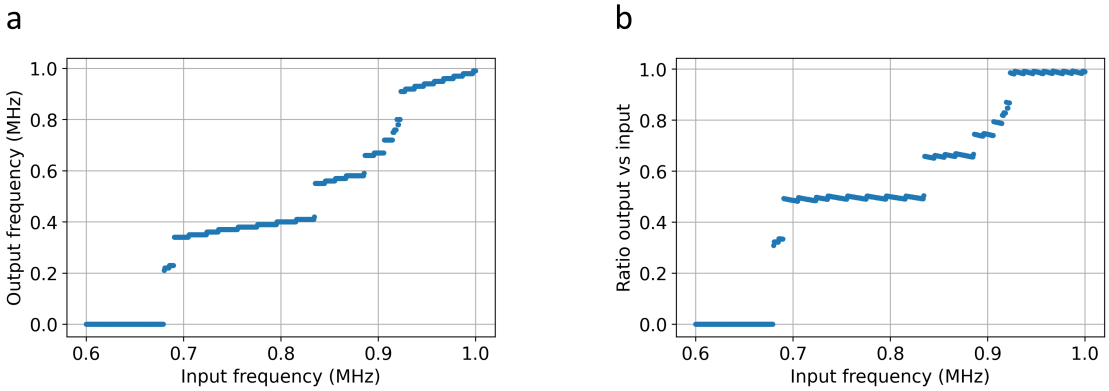


Figure 4.21: a: Simulations of the output frequency f_{out} as a function of the input one f_{in}. b: Simulations of the frequency ratio $\frac{f_{out}}{f_{in}}$ as a function of the input frequency f_{in}. The parameters used are: peak-to-peak voltage $V_{s,peak}$ = 2.06 V, load resistance $R_L = 4\ k\Omega$, frequency averaged over a time of $T_f = 0.0001\ s$ and with a number of points of N = 1000001. All the parameters of the model are the one presented in Table. 2.1.

EqSpike preliminary results

EqSpike, introduced in both Chapters 1 and 3, stands out for its local updates in both time and space, a characteristic that aligns with the spatial-temporal dynamics of biological neural systems [153]. A promising future to bridge the gap between hardware and algorithmic approaches is to achieve a fully memristive-based implementation of EqSpike, by incorporating memristor neurons and synapses. EqSpike has been designed with Leaky Integrate-and-Fire neurons and has yet to be tested with other types of models or devices. As discussed in Chapters 1 and 2, emerging technologies offer promising neuron candidates, and the devices obtained show complex behaviors that can mimic neuronal behaviors. Therefore, to bridge the gap between all chapters of this thesis, studying an implementation of EqSpike with the neuron model developed in Chapter 2 is a promising lead to realize biomimetic neuromorphic computing at low energy costs. The EqSpike algorithm is dimensionless, which implies that all parameters must be adapted to real physical systems for practical implementation, and this is a first challenge we aim to bring some answers to.

Simulations: methods

In our simulations, the state of each neuron is determined by two critical parameters: the voltage, denoted as V, and the temperature, denoted as T. To calculate these states accurately at a given time point t, we employ the Runge-Kutta method of order 4.

Notably, memristor neurons within our simulation framework are computed using a current-controlled framework, in the same context as the one studied in Chapter 2. In this context, these neurons receive an input current and produce an output current as a result. To replicate

the application of a bias, we employ a modeling approach where a current is incorporated into a neuron's input.

All devices parameters are identical, and are the ones presented in Table 2.1. The parameters of EqSpike used here are the following. The free phase and the nudge phase are obtained using $K_{free} = 60000$ and $K_{nudge} = 300000$ points respectively, with a time step of $dt = 0.67$ ns. The nudging factor β is chosen to be $\beta = 5 \cdot 10^{-10}$ V.s, and the learning rate for the weight is $\eta = 10$ and the one for the bias is $\eta_b = 0.4 \cdot 10^{-9}$. The maximum target frequency is fixed to $f_{max} = 10$ MHz. The output frequency ρ is computed over $T = 10000$ time steps. To compute ρ, the current is binarized and fed to a leaky integrateor (V_{LI}), with a leak factor of $\gamma_{LI} = 1000000$. To obtain the derivative $\dot{\rho}$, we delay the signal V_{LI} by $\tau = 5000$ time steps. To obtain the average of this derivative $\dot{\rho}_{avg}$, a low pass filter is implemented by computing the average over $N_{filt} = 10000$ time steps.

First results

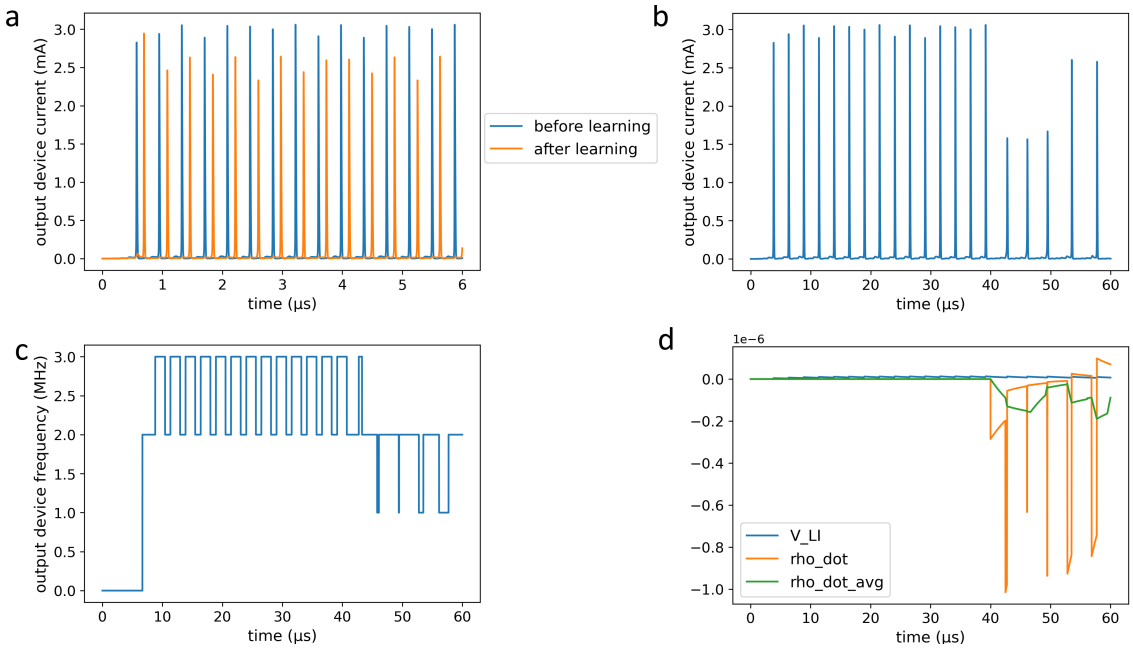


Figure 4.22: Simulations when the label is zero. a: Comparison between inference before a learning step and after a learning step. b: Current in the output device during the free and nudging phases. c: Frequency of the output device during the free and nudging phases. d: Values of $\dot{\rho}$ and $\dot{\rho}_{avg}$ during the free and nudging phases. The parameters used are $x_1 = 0.00034091$, $x_2 = 0.00041204$, $W_1 = 0.38711533$, $W_2 = -0.04242794$, $b = 0$.

To determine all the accurate parameters, a first step is to control that in a two-input neurons, one-output neuron setup, the system can learn when an example is presented. Here, the input is the same:$x_1 = 0.00034091A$, and $x_2 = 0.00041204A$. Figures 4.22 and 4.23 have all the same parameters, inputs and weights, but in Fig. 4.22 the target is set to zero whereas for Fig.

4.23 the target is set to one.

Figure 4.22 corresponds to a target zero, which means that between the inference before learning and the inference after learning,the frequency should have decreased. This is indeed the case, as presented in Fig. 4.22a. Figure 4.22b shows the ouput device current, which has a decrease in frequency in the nudge phase when the nudge factor β is applied, as shown in Fig. 4.22c where the output frequency is plotted. Figure 4.22d shows $\dot{\rho}$ and $\dot{\rho}_{avg}$, which are negative because of the decrease in frequency needed.

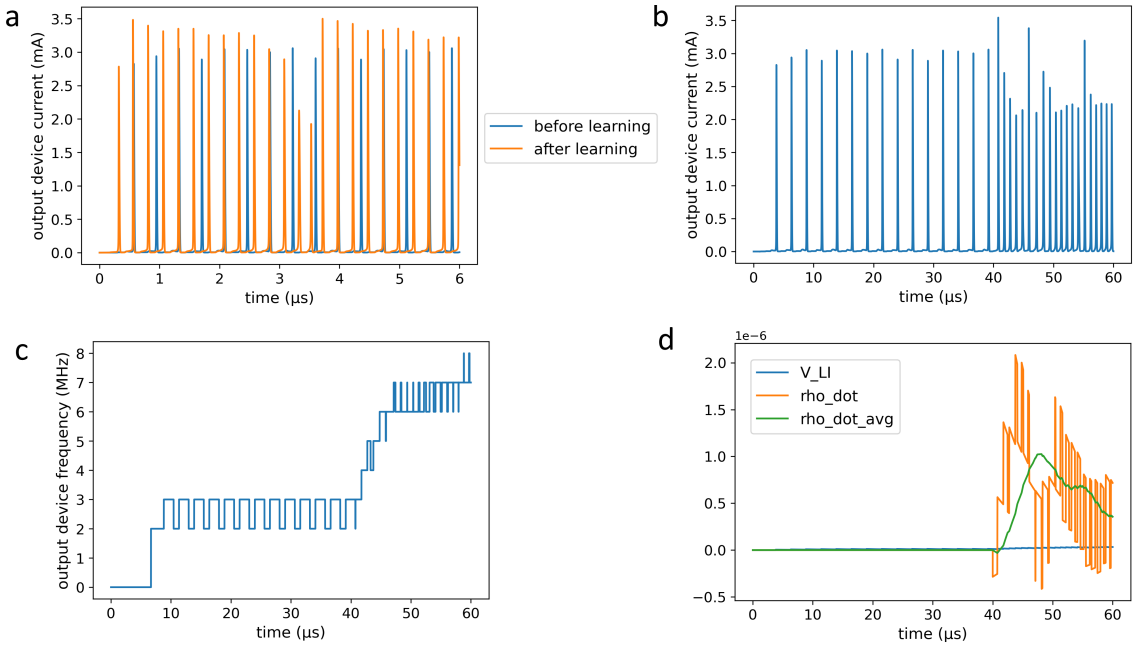


Figure 4.23: Simulations when the label is one. a: Comparison between inference before a learning step and after a learning step. b: Current in the output device during the free and nudging phases. c: Frequency of the output device during the free and nudging phases. d: Values of $\dot{\rho}$ and $\dot{\rho}_{avg}$ during the free and nudging phases. The parameters used are $x_1 = 0.00034091$, $x_2 = 0.00041204$, $W_1 = 0.38711533$, $W_2 = -0.04242794$, $b = 0$, $\eta = 10$, $\eta_b = 0.4e-9$.

Figure 4.23 shows the exact same plots but for a target of 1. This means that the output frequency after learning is higher than the one before, as shown in Fig 4.23a.

In this section, we have explored implementation possibilities of the EqSpike algorithm with memristor neurons, further in-depth studies are needed to train networks and assess the robustness of this approach. Additionally, extending the research to integrate memristor neurons and synapses, akin to Chapter 4's study, presents a valuable future direction.

There are also broader research avenues to consider. At the device level, improving memristor devices for analog use, with linear set and reset characteristics and reduced variability, can significantly improve network performance. From an algorithmic perspective, as outlined

in Chapter 4, adapting the Equilibrium algorithm to mitigate intra and inter-device variabilities holds potential. Moreover, while this thesis has primarily focused on supervised learning with labeled datasets, future work could explore unsupervised or semi-supervised learning with Equilibrium Propagation, addressing scenarios with limited or unlabeled data.

In conclusion, the convergence of highly parametrizable computing systems, especially neuromorphic computing, with physics-grounded algorithms offers great potential for the advancement of ultra-low power AI. Investigating the implementation of Equilibrium, of EqSpike, and enhancing algorithm robustness align with the increasing demand for efficient AI, promising innovative breakthroughs at the intersection of physics and artificial intelligence.

List of publications

Peer-Reviewed Journal Articles

✠ **Marie Drouhin**, Shuai Li, Matthieu Grelier, Sophie Collin, Florian Godel, Robert G Elliman, Bruno Dlubak, Juan Trastoy, Damien Querlioz and Julie Grollier , "Characterization and modeling of spiking and bursting in experimental NbOx neuron." , *Neuromorphic Computing and Engineering*. p. 044008, 2023. doi:10.1145/3566097.3567944

Peer-Reviewed Conference Proceedings

✠

Kamel-Eddine Harabi, Clément Turck , **Marie Drouhin**, Adrien Renaudineau , Thomas Bersani-Veroni, Tifenn Hirtzlin, Elisa Vianello , Marc Bocquet , Jean-Michel Portal and Damien Querlioz , "A Multimode Hybrid Memristor-CMOS Prototyping Platform Supporting Digital and Analog Projects" , *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, p. 184–185, 2023. doi:10.1088/2634-4386/ac969a

Conferences Without Proceedings

✠ **Marie Drouhin**, Clément Turck, Kamel-Eddine Harabi, Adrien Renaudineau, Thomas Bersani-Veroni, Elisa Vianello, Jean-Michel Portal, Julie Grollier and Damien Querlioz, "Nanoelectronic implementation of Equilibrium Propagation" , *Emerging Topics in Artificial Intelligence (ETAI)*.. 2023. *Oral presentation.*

✠ **Marie Drouhin**, Clément Turck, Kamel-Eddine Harabi, Adrien Renaudineau, Thomas Bersani-Veroni, Elisa Vianello, Jean-Michel Portal, Julie Grollier and Damien Querlioz, "Adapting Equilibrium Propagation to physical systems" , *Spintronics XVI 2023*.. 2023. *Oral presentation.*

✠ **Marie Drouhin**, Shuai Li, Matthieu Grelier, Sophie Collin, Florian Godel, Robert G. Elliman, Bruno Dlubak, Juan Trastoy, Damien Querlioz and Julie Grollier, "Neu-

ronal behaviors in niobium oxide memristors" , *AI, Neuroscience and Hardware: From Neural to Artificial Systems and Back Again.*, 2022. *Poster presentation.*

✠ **MARIE DROUHIN**, SHUAI LI, MATTHIEU GRELIER, SOPHIE COLLIN, FLORIAN GODEL, ROBERT G. ELLIMAN, BRUNO DLUBAK, JUAN TRASTOY, DAMIEN QUERLIOZ and JULIE GROLLIER, "Neuronal behaviors in niobium oxide memristors" , *MEMRISYS*, 2022. *Poster presentation.*

Bibliography

[1] Ke Yang, Qingxi Duan, Yanghao Wang, Teng Zhang, Yuchao Yang, and Ru Huang. Transiently chaotic simulated annealing based on intrinsic nonlinearity of memristors for efficient solution of optimization problems. *Science advances*, 6(33):eaba9901, 2020.

[2] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[3] GPU devotes more transistors to data processing. https://www.moleculardevices.com/applications/patch-clamp-electrophysiology/what-action-potential.

[4] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pages 10–14. IEEE, 2014.

[5] Action potential. https://www.moleculardevices.com/applications/patch-clamp-electrophysiology/what-action-potential.

[6] Wikipedia: Excitatory synapse. https://en.wikipedia.org/wiki/Excitatory_synapse.

[7] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007.

[8] Christopher A Del Negro, Chie-Fang Hsiao, Scott H Chandler, and Alan Garfinkel. Evidence for a novel bursting mechanism in rodent trigeminal neurons. *Biophysical journal*, 75(1):174–182, 1998.

[9] Wenqiang Zhang, Bin Gao, Jianshi Tang, Peng Yao, Shimeng Yu, Meng-Fan Chang, Hoi-Jun Yoo, He Qian, and Huaqiang Wu. Neuro-inspired computing chips. *Nature electronics*, 3(7):371–382, 2020.

[10] Zongjie Shen, Chun Zhao, Yanfei Qi, Wangying Xu, Yina Liu, Ivona Z Mitrovic, Li Yang, and Cezhou Zhao. Advances of rram devices: Resistive switching mechanisms, materials and bionic synaptic application. *Nanomaterials*, 10(8):1437, 2020.

[11] Suhas Kumar, Xinxin Wang, John Paul Strachan, Yuchao Yang, and Wei D Lu. Dynamical memristors for higher-complexity neuromorphic computing. *Nature Reviews Materials*, pages 1–17, 2022.

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

[13] Rui Liu, Debayan Mahalanabis, Hugh J Barnaby, and Shimeng Yu. Investigation of single-bit and multiple-bit upsets in oxide RRAM-based 1t1r and crossbar memory arrays. *IEEE Transactions on Nuclear Science*, 62(5):2294–2301, 2015.

[14] Valerio Milo, Artem Glukhov, Eduardo Pérez, Cristian Zambelli, Nicola Lepri, Mamathamba Kalishettyhalli Mahadevaiah, Emilio Pérez-Bosch Quesada, Piero Olivo, Christian Wenger, and Daniele Ielmini. Accurate program/verify schemes of resistive switching memory (RRAM) for in-memory neural network circuits. *IEEE Transactions on Electron Devices*, 68(8):3832–3837, 2021.

[15] Jacopo Frascaroli, Stefano Brivio, Erika Covi, and Sabina Spiga. Evidence of soft bound behaviour in analogue memristive devices for neuromorphic computing. *Scientific reports*, 8(1):7178, 2018.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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*, 6(1):52–63, 2023.

[21] Data centres and data transmission networks. https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks.

[22] Avita Katal, Susheela Dahiya, and Tanupriya Choudhury. Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing*, 26(3):1845–1875, 2023.

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

[24] Scott McCartney. *ENIAC: The triumphs and tragedies of the world's first computer*. Walker & Company, 1999.

[25] 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.

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

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

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

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

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

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

[32] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[33] Raghubir Singh and Sukhpal Singh Gill. Edge AI: a survey. *Internet of Things and Cyber-Physical Systems*, 2023.

[34] Yuanming Shi, Kai Yang, Tao Jiang, Jun Zhang, and Khaled B Letaief. Communication-efficient edge AI: Algorithms and systems. *IEEE Communications Surveys & Tutorials*, 22(4):2167–2191, 2020.

[35] 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.

[36] Anders Krogh. What are artificial neural networks? *Nature biotechnology*, 26(2):195–197, 2008.

[37] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

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

[39] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[40] Frank Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan books, 1962.

[41] Hans-Dieter Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

[42] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479:480, 1969.

[43] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[44] 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.

[45] John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

[46] John J Hopfield and David W Tank. "Neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[47] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[48] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

[49] Werbos. Backpropagation: Past and future. In *IEEE 1988 International Conference on Neural Networks*, pages 343–353. IEEE, 1988.

[50] David B Parker. Learning-logic. *Tech. Rep.*, 47, 1985.

[51] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.

[52] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

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

[54] Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[55] Zoubin Ghahramani. Unsupervised learning. In *Summer school on machine learning*, pages 72–112. Springer, 2003.

[56] M Emre Celebi and Kemal Aydin. *Unsupervised learning algorithms*, volume 9. Springer, 2016.

[57] Trevor Hastie, Robert Tibshirani, Jerome Friedman, Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised learning. *The elements of statistical learning: Data mining, inference, and prediction*, pages 485–585, 2009.

[58] O Chapelle, B Schölkopf, and A Zien. Semi-supervised learning. 2006.

[59] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005.

[60] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020.

[61] 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.

[62] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE transactions on knowledge and data engineering*, 35(1):857–876, 2021.

[63] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[64] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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

[66] Augustin Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[67] Haskell B Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[68] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

[69] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[70] 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.

[71] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.

[72] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[73] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[74] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26:289–315, 2007.

[75] 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.

[76] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

[77] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[79] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[80] 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.

[81] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[82] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[83] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[84] 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.

[85] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[86] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[87] 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.

[88] 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.

[89] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[90] Xue-Wen Chen and Xiaotong Lin. Big data deep learning: challenges and perspectives. *IEEE access*, 2:514–525, 2014.

[91] Yvan Saeys, Inaki Inza, and Pedro Larranaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[92] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[93] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[94] Laurens vd Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[95] Jie Ding, Vahid Tarokh, and Yuhong Yang. Model selection techniques: An overview. *IEEE Signal Processing Magazine*, 35(6):16–34, 2018.

[96] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

[97] Russell Reed. Pruning algorithms-a survey. *IEEE transactions on Neural Networks*, 4(5):740–747, 1993.

[98] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[99] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint arXiv:1907.10701*, 2019.

[100] Wm A Wulf and Sally A McKee. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[101] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

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

[103] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015.

[104] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[105] Garrick Orchard, E Paxon Frady, Daniel Ben Dayan Rubin, Sophia Sanborn, Sumit Bam Shrestha, Friedrich T Sommer, and Mike Davies. Efficient neuromorphic signal processing with Loihi 2. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 254–259. IEEE, 2021.

[106] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[107] Garrett E Alexander and Michael D Crutcher. Functional architecture of basal ganglia circuits: neural substrates of parallel processing. *Trends in neurosciences*, 13(7):266–271, 1990.

[108] Marcus E Raichle and Debra A Gusnard. Appraising the brain's energy budget. *Proceedings of the National Academy of Sciences*, 99(16):10237–10239, 2002.

[109] Christopher S Von Bartheld, Jami Bahney, and Suzana Herculano-Houzel. The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. *Journal of Comparative Neurology*, 524(18):3865–3895, 2016.

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

[111] Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in human neuroscience*, page 31, 2009.

[112] Robert Sylwester. *A celebration of neurons: An educator's guide to the human brain*. ERIC, 1995.

[113] Michael H Grider, Rishita Jessu, and Rian Kabir. Physiology, action potential. 2019.

[114] Thomas C Südhof and Robert C Malenka. Understanding synapses: past, present, and future. *Neuron*, 60(3):469–476, 2008.

[115] Thomas Trappenberg. *Fundamentals of computational neuroscience*. OUP Oxford, 2009.

[116] John Carew Eccles. *The physiology of synapses*. Academic Press, 2013.

[117] Larry F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999.

[118] Alan L Hodgkin. The local electric changes associated with repetitive action in a non-medullated axon. *The Journal of physiology*, 107(2):165, 1948.

[119] 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.

[120] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.

[121] Jinichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 1962.

[122] Eugene M Izhikevich and Richard FitzHugh. Fitzhugh-Nagumo model. *Scholarpedia*, 1(9):1349, 2006.

[123] Charles F Cadieu, Ha Hong, Daniel LK Yamins, Nicolas Pinto, Diego Ardila, Ethan A Solomon, Najib J Majaj, and James J DiCarlo. Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS computational biology*, 10(12):e1003963, 2014.

[124] Seyed-Mahdi Khaligh-Razavi and Nikolaus Kriegeskorte. Deep supervised, but not unsupervised, models may explain it cortical representation. *PLoS computational biology*, 10(11):e1003915, 2014.

[125] Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, et al. A deep learning framework for neuroscience. *Nature neuroscience*, 22(11):1761–1770, 2019.

[126] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[127] Pritish Narayanan, Alessandro Fumarola, Lucas L Sanches, Kohji Hosokawa, Scott C Lewis, Robert M Shelby, and Geoffrey W Burr. Toward on-chip acceleration of the backpropagation algorithm using nonvolatile memory. *IBM Journal of Research and Development*, 61(4/5):11–1, 2017.

[128] Alessandro Fumarola, Pritish Narayanan, Lucas L Sanches, Severin Sidler, Junwoo Jang, Kibong Moon, Robert M Shelby, Hyunsang Hwang, and Geoffrey W Burr. Accelerating machine learning with non-volatile memory: Exploring device and circuit tradeoffs. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. Ieee, 2016.

[129] James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in cognitive sciences*, 23(3):235–250, 2019.

[130] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[131] Geoffrey E Hinton. Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural computation*, 1(1):143–150, 1989.

[132] Javier R Movellan. Contrastive hebbian learning in the continuous hopfield model. In *Connectionist models*, pages 10–17. Elsevier, 1991.

[133] Pierre Baldi and Fernando Pineda. Contrastive learning and neural oscillations. *Neural computation*, 3(4):526–545, 1991.

[134] Geoffrey E Hinton and James McClelland. Learning representations by recirculation. In *Neural information processing systems*, 1987.

[135] Randall C O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, 8(5):895–938, 1996.

[136] Xiaohui Xie and H Sebastian Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural computation*, 15(2):441–454, 2003.

[137] Yoshua Bengio and Asja Fischer. Early inference in energy-based models approximates back-propagation. *arXiv preprint arXiv:1510.02777*, 2015.

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

[139] Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*, 2014.

[140] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I 15*, pages 498–515. Springer, 2015.

[141] Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[142] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55(5), 2014.

[143] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):13276, 2016.

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

[145] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Advances in neural information processing systems*, 31, 2018.

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

[147] Friedemann Zenke, Sander M Bohté, Claudia Clopath, Iulia M Comşa, Julian Göltz, Wolfgang Maass, Timothée Masquelier, Richard Naud, Emre O Neftci, Mihai A Petrovici, et al. Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron*, 109(4):571–575, 2021.

[148] 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.

[149] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021.

[150] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. A history of spike-timing-dependent plasticity. *Frontiers in synaptic neuroscience*, 3:4, 2011.

[151] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. Spike-timing-dependent plasticity: a comprehensive overview. *Frontiers in synaptic neuroscience*, 4:2, 2012.

[152] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625, 2020.

[153] Erwann Martin, Maxence Ernoult, Jérémie Laydevant, Shuai Li, Damien Querlioz, Teodora Petrisor, and Julie Grollier. Eqspike: spike-driven equilibrium propagation for neuromorphic implementations. *Iscience*, 24(3), 2021.

[154] Alexandre Payeur, Jordan Guerguiev, Friedemann Zenke, Blake A Richards, and Richard Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nature neuroscience*, 24(7):1010–1019, 2021.

[155] Gordon E Moore et al. Progress in digital integrated electronics. In *Electron devices meeting*, volume 21, pages 11–13. Washington, DC, 1975.

[156] Mingyi Rao, Hao Tang, Jiangbin Wu, Wenhao Song, Max Zhang, Wenbo Yin, Ye Zhuo, Fatemeh Kiani, Benjamin Chen, Xiangqi Jiang, et al. Thousands of conductance levels in memristors integrated on CMOS. *Nature*, 615(7954):823–829, 2023.

[157] Taylor Simons and Dah-Jye Lee. A review of binarized neural networks. *Electronics*, 8(6):661, 2019.

[158] Paul Hasler, Chris Diorio, Bradley A Minch, and Carver Mead. Single transistor learning synapse with long term storage. In *Proceedings of ISCAS'95-International Symposium on Circuits and Systems*, volume 3, pages 1660–1663. IEEE, 1995.

[159] Chris Diorio, Paul Hasler, A Minch, and Carver A Mead. A single-transistor silicon synapse. *IEEE transactions on Electron Devices*, 43(11):1972–1980, 1996.

[160] Jingrui Wang and Fei Zhuge. Memristive synapses for brain-inspired computing. *Advanced Materials Technologies*, 4(3):1800544, 2019.

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

[162] R Stanley Williams. How we found the missing memristor. *IEEE spectrum*, 45(12):28–35, 2008.

[163] Leon O Chua and Sung Mo Kang. Memristive devices and systems. *Proceedings of the IEEE*, 64(2):209–223, 1976.

[164] Leon Chua. If it's pinched it's a memristor. *Semiconductor Science and Technology*, 29(10):104001, 2014.

[165] Lei Wang, CiHui Yang, Jing Wen, Shan Gai, and YuanXiu Peng. Overview of emerging memristor families from resistive memristor to spintronic memristor. *Journal of Materials Science: Materials in Electronics*, 26:4618–4628, 2015.

[166] Ee Wah Lim and Razali Ismail. Conduction mechanism of valence change resistive switching memory: A survey. *Electronics*, 4(3):586–613, 2015.

[167] Rainer Waser and Masakazu Aono. Nanoionics-based resistive switching memories. *Nature materials*, 6(11):833–840, 2007.

[168] Rainer Waser, Regina Dittmann, Georgi Staikov, and Kristof Szot. Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges. *Advanced materials*, 21(25-26):2632–2663, 2009.

[169] Ilia Valov, Rainer Waser, John R Jameson, and Michael N Kozicki. Electrochemical metallization memories—fundamentals, applications, prospects. *Nanotechnology*, 22(25):254003, 2011.

[170] Taehyun Kim, Seung-Hwan Kim, Jae-Hyeun Park, June Park, Euyjin Park, Seung-Geun Kim, and Hyun-Yong Yu. An artificial neuron using a bipolar electrochemical metallization switch and its enhanced spiking properties through filament confinement. *Advanced Electronic Materials*, 7(1):2000410, 2021.

[171] Yihao Chen, Yu Wang, Yuhao Luo, Xinwei Liu, Yuqi Wang, Fei Gao, Jianguang Xu, Ertao Hu, Subhranu Samanta, Xiang Wan, et al. Realization of artificial neuron using mxene bi-directional threshold switching memristors. *IEEE Electron Device Letters*, 40(10):1686–1689, 2019.

[172] Daniele Ielmini, Rainer Bruchhaus, and Rainer Waser. Thermochemical resistive switching: materials, mechanisms, and scaling projections. *Phase Transitions*, 84(7):570–602, 2011.

[173] Daniele Ielmini, F Nardi, and C Cagli. Physical models of size-dependent nanofilament formation and rupture in nio resistive switching memories. *Nanotechnology*, 22(25):254022, 2011.

[174] Stefan Slesazeck and Thomas Mikolajick. Nanoscale resistive switching memory devices: a review. *Nanotechnology*, 30(35):352003, 2019.

[175] Wei Yi, Kenneth K Tsang, Stephen K Lam, Xiwei Bai, Jack A Crowell, and Elias A Flores. Biological plausibility and stochasticity in scalable VO_2 active memristor neurons. *Nature communications*, 9(1):1–10, 2018.

[176] Rui Yuan, Qingxi Duan, Pek Jun Tiw, Ge Li, Zhuojian Xiao, Zhaokun Jing, Ke Yang, Chang Liu, Chen Ge, Ru Huang, et al. A calibratable sensory neuron based on epitaxial VO2 for spike-based neuromorphic multisensory system. *Nature Communications*, 13(1):3973, 2022.

[177] Parker Schofield, Adelaide Bradicich, Rebeca M Gurrola, Yuwei Zhang, Timothy D Brown, Matt Pharr, Patrick J Shamberger, and Sarbajit Banerjee. Harnessing the metal–insulator transition of VO2 in neuromorphic computing. *Advanced Materials*, page 2205294, 2022.

[178] Xinyi Li, Yanan Zhong, Hang Chen, Jianshi Tang, Xiaojian Zheng, Wen Sun, Yang Li, Dong Wu, Bin Gao, Xiaolin Hu, et al. A memristors-based dendritic neuron for high-efficiency spatial-temporal information processing. *Advanced Materials*, page 2203684, 2022.

[179] Shimul Kanti Nath. *Filamentary Threshold SwitchinIg in Niobium Oxides*. PhD thesis, The Australian National University (Australia), 2021.

[180] Yibo Li, Zhongrui Wang, Rivu Midya, Qiangfei Xia, and J Joshua Yang. Review of memristor devices in neuromorphic computing: materials sciences and device challenges. *Journal of Physics D: Applied Physics*, 51(50):503002, 2018.

[181] Ilia Valov. Redox-based resistive switching memories (rerams): Electrochemical systems at the atomic scale. *ChemElectroChem*, 1(1):26–36, 2014.

[182] Sridhar Chandrasekaran, Firman Mangasa Simanjuntak, R Saminathan, Debashis Panda, and Tseung-Yuen Tseng. Improving linearity by introducing al in HfO_2 as a memristor synapse device. *Nanotechnology*, 30(44):445205, 2019.

[183] Ximeng Guan, Shimeng Yu, and H-S Philip Wong. A SPICE compact model of metal oxide resistive switching memory with variations. *IEEE electron device letters*, 33(10):1405–1407, 2012.

[184] Peng Huang, Dongbin Zhu, Sijie Chen, Zheng Zhou, Zhe Chen, Bin Gao, Lifeng Liu, Xiaoyan Liu, and Jinfeng Kang. Compact model of HfO_x-based electronic synaptic devices for neuromorphic computing. *IEEE Transactions on Electron Devices*, 64(2):614–621, 2017.

[185] Jiyong Woo, Dongwook Lee, Yunmo Koo, and Hyunsang Hwang. Dual functionality of threshold and multilevel resistive switching characteristics in nanoscale HfO_2-based RRAM devices for artificial neuron and synapse elements. *Microelectronic Engineering*, 182:42–45, 2017.

[186] JJ Wang, SG Hu, XT Zhan, Q Yu, Z Liu, Tu Pei Chen, Y Yin, Sumio Hosaka, and Y Liu. Handwritten-digit recognition by hybrid convolutional neural network based on HfO_2 memristive spiking-neuron. *Scientific reports*, 8(1):1–7, 2018.

[187] Jiadi Zhu, Teng Zhang, Yuchao Yang, and Ru Huang. A comprehensive review on emerging artificial neuromorphic devices. *Applied Physics Reviews*, 7(1), 2020.

[188] Julie Grollier, Damien Querlioz, KY Camsari, Karin Everschor-Sitte, Shunsuke Fukami, and Mark D Stiles. Neuromorphic spintronics. *Nature electronics*, 3(7):360–370, 2020.

[189] Xiaobin Wang, Yiran Chen, Haiwen Xi, Hai Li, and Dimitar Dimitrov. Spintronic memristor through spin-torque-induced magnetization motion. *IEEE electron device letters*, 30(3):294–297, 2009.

[190] Akinobu Yamaguchi, Teruo Ono, Saburo Nasu, Kousaku Miyake, Ko Mibu, and Teruya Shinjo. Real-space observation of current-driven domain wall motion in submicron magnetic wires. *Physical review letters*, 92(7):077205, 2004.

[191] A Chanthbouala, R Matsumoto, J Grollier, V Cros, A Anane, A Fert, AV Khvalkovskiy, KA Zvezdin, K Nishimura, Y Nagamine, et al. Vertical-current-induced domain-wall motion in mgo-based magnetic tunnel junctions with low current densities. *Nature Physics*, 7(8):626–630, 2011.

[192] Steven Lequeux, Joao Sampaio, Vincent Cros, Kay Yakushiji, Akio Fukushima, Rie Matsumoto, Hitoshi Kubota, Shinji Yuasa, and Julie Grollier. A magnetic synapse: multilevel spin-torque memristor with perpendicular anisotropy. *Scientific reports*, 6(1):31510, 2016.

[193] Shehzaad Kaka, Matthew R Pufall, William H Rippard, Thomas J Silva, Stephen E Russek, and Jordan A Katine. Mutual phase-locking of microwave spin torque nano-oscillators. *Nature*, 437(7057):389–392, 2005.

[194] Sergei Urazhdin, Phillip Tabor, Vasil Tiberkevich, and Andrei Slavin. Fractional synchronization of spin-torque nano-oscillators. *Physical review letters*, 105(10):104101, 2010.

[195] Nicolas Locatelli, Vincent Cros, and Julie Grollier. Spin-torque building blocks. *Nature materials*, 13(1):11–20, 2014.

[196] Tingsu Chen, Randy K Dumas, Anders Eklund, Pranaba K Muduli, Afshin Houshang, Ahmad A Awad, Philipp Dürrenfeld, B Gunnar Malm, Ana Rusu, and Johan Åkerman. Spin-torque and spin-hall nano-oscillators. *Proceedings of the IEEE*, 104(10):1919–1945, 2016.

[197] Uwe Schroeder, Min Hyuk Park, Thomas Mikolajick, and Cheol Seong Hwang. The fundamentals and applications of ferroelectric HfO_2. *Nature Reviews Materials*, 7(8):653–669, 2022.

[198] Erika Covi, Halid Mulaosmanovic, Benjamin Max, Stefan Slesazeck, and Thomas Mikolajick. Ferroelectric-based synapses and neurons for neuromorphic computing. *Neuromorphic Computing and Engineering*, 2(1):012002, 2022.

[199] Abu Sebastian, Manuel Le Gallo, Geoffrey W Burr, Sangbum Kim, Matthew BrightSky, and Evangelos Eleftheriou. Tutorial: Brain-inspired computing using phase-change memory devices. *Journal of Applied Physics*, 124(11), 2018.

[200] Ming Xu, Xianliang Mai, Jun Lin, Wei Zhang, Yi Li, Yuhui He, Hao Tong, Xiang Hou, Peng Zhou, and Xiangshui Miao. Recent advances on neuromorphic devices based on chalcogenide phase-change materials. *Advanced Functional Materials*, 30(50):2003419, 2020.

[201] Tomas Tuma, Angeliki Pantazi, Manuel Le Gallo, Abu Sebastian, and Evangelos Eleftheriou. Stochastic phase-change neurons. *Nature nanotechnology*, 11(8):693–699, 2016.

[202] 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.

[203] Qiangfei Xia and J Joshua Yang. Memristive crossbar arrays for brain-inspired computing. *Nature materials*, 18(4):309–323, 2019.

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

[205] Huihan Li, Shaocong Wang, Xumeng Zhang, Wei Wang, Rui Yang, Zhong Sun, Wanxiang Feng, Peng Lin, Zhongrui Wang, Linfeng Sun, et al. Memristive crossbar arrays for storage and computing applications. *Advanced Intelligent Systems*, 3(9):2100017, 2021.

[206] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, 12:774, 2018.

[207] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, page 891, 2018.

[208] Jack D Kendall and Suhas Kumar. The building blocks of a brain-inspired computer. *Applied Physics Reviews*, 7(1):011305, 2020.

[209] Mohammed A Zidan, John Paul Strachan, and Wei D Lu. The future of electronics based on memristive systems. *Nature electronics*, 1(1):22–29, 2018.

[210] Suhas Kumar, R Stanley Williams, and Ziwen Wang. Third-order nanocircuit elements for neuromorphic engineering. *Nature*, 585(7826):518–523, 2020.

[211] Yuting Wu, Xinxin Wang, and Wei Lu. Dynamic resistive switching devices for neuromorphic computing. *Semiconductor Science and Technology*, 2021.

[212] Dahye Kim, Beomki Jeon, Yunseok Lee, Doohyung Kim, Youngboo Cho, and Sungjun Kim. Prospects and applications of volatile memristors. *Applied Physics Letters*, 121(1):010501, 2022.

[213] Joon-Kyu Han, Seong-Yun Yun, Sang-Won Lee, Ji-Man Yu, and Yang-Kyu Choi. A review of artificial spiking neuron devices for neural processing and sensing. *Advanced Functional Materials*, page 2204102, 2022.

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

[215] D Ielmini, Z Wang, and Y Liu. Brain-inspired computing via memory device physics. *APL Materials*, 9(5), 2021.

[216] Zhongrui Wang, Huaqiang Wu, Geoffrey W Burr, Cheol Seong Hwang, Kang L Wang, Qiangfei Xia, and J Joshua Yang. Resistive switching materials for information processing. *Nature Reviews Materials*, 5(3):173–195, 2020.

[217] Seung Hwan Lee, Xiaojian Zhu, and Wei D Lu. Nanoscale resistive switching devices for memory and computing applications. *Nano Research*, 13(5):1228–1243, 2020.

[218] Yue Xi, Bin Gao, Jianshi Tang, An Chen, Meng-Fan Chang, Xiaobo Sharon Hu, Jan Van Der Spiegel, He Qian, and Huaqiang Wu. In-memory learning with analog resistive switching memory: A review and perspective. *Proceedings of the IEEE*, 109(1):14–42, 2020.

[219] Sanghyeon Choi, Jehyeon Yang, and Gunuk Wang. Emerging memristive artificial synapses and neurons for energy-efficient neuromorphic computing. *Advanced Materials*, 32(51):2004659, 2020.

[220] Rui Yang, He-Ming Huang, and Xin Guo. Memristive synapses and neurons for bioinspired computing. *Advanced Electronic Materials*, 5(9):1900287, 2019.

[221] Ke Yang, J Joshua Yang, Ru Huang, and Yuchao Yang. Nonlinearity in memristors for neuromorphic dynamic systems. *Small Science*, page 2100049, 2021.

[222] Xumeng Zhang, Wei Wang, Qi Liu, Xiaolong Zhao, Jinsong Wei, Rongrong Cao, Zhihong Yao, Xiaoli Zhu, Feng Zhang, Hangbing Lv, et al. An artificial neuron based on a threshold switching memristor. *IEEE Electron Device Letters*, 39(2):308–311, 2017.

[223] 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):137–145, 2018.

[224] Pablo Stoliar, Julien Tranchant, Benoit Corraze, Etienne Janod, Marie-Paule Besland, Federico Tesler, Marcelo Rozenberg, and Laurent Cario. A leaky-integrate-and-fire neuron analog realized with a Mott insulator. *Advanced Functional Materials*, 27(11):1604740, 2017.

[225] Shuai Li, Xinjun Liu, Sanjoy Kumar Nandi, Dinesh Kumar Venkatachalam, and Robert Glen Elliman. High-endurance megahertz electrical self-oscillation in Ti/NbO_x bilayer structures. *Applied Physics Letters*, 106(21):212902, 2015.

[226] Suhas Kumar, John Paul Strachan, and R Stanley Williams. Chaotic dynamics in nanoscale NbO_2 Mott memristors for analogue computing. *Nature*, 548(7667):318–321, 2017.

[227] Qingxi Duan, Zhaokun Jing, Xiaolong Zou, Yanghao Wang, Ke Yang, Teng Zhang, Si Wu, Ru Huang, and Yuchao Yang. Spiking neurons with spatiotemporal dynamics and gain modulation for monolithically integrated memristive neural networks. 11(1):1–13, 2020.

[228] Marie Drouhin, Shuai Li, Matthieu Grelier, Sophie Collin, Florian Godel, Robert G Elliman, Bruno Dlubak, Juan Trastoy, Damien Querlioz, and Julie Grollier. Characterization and modeling of spiking and bursting in experimental nbo x neuron. *Neuromorphic Computing and Engineering*, 2(4):044008, 2022.

[229] Gary A Gibson. Designing negative differential resistance devices based on self-heating. *Advanced Functional Materials*, 28(22):1704175, 2018.

[230] Matthew D Pickett and R Stanley Williams. Sub-100 fJ and sub-nanosecond thermally driven threshold switching in niobium oxide crosspoint nanodevices. *Nanotechnology*, 23(21):215202, 2012.

[231] Stefan Slesazeck, Hannes Mähne, Helge Wylezich, Andre Wachowiak, Janaki Radhakrishnan, Alon Ascoli, Ronald Tetzlaff, and Thomas Mikolajick. Physical model of threshold switching in NbO_2 based memristors. *RSC advances*, 5(124):102318–102322, 2015.

[232] Ziwen Wang, Suhas Kumar, Yoshio Nishi, and H-S Philip Wong. Transient dynamics of NbO_x threshold switches explained by Poole-Frenkel based thermal feedback mechanism. *Applied Physics Letters*, 112(19):193503, 2018.

[233] HH Poole. Viii. on the dielectric constant and electrical conductivity of mica in intense fields. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 32(187):112–129, 1916.

[234] J Frenkel. On pre-breakdown phenomena in insulators and electronic semi-conductors. *Physical Review*, 54(8):647, 1938.

[235] Shuai Li, Xinjun Liu, Sanjoy Kumar Nandi, Shimul Kanti Nath, and Robert Glen Elliman. Origin of current-controlled negative differential resistance modes and the emergence of composite characteristics with high complexity. *Advanced Functional Materials*, 29(44):1905060, 2019.

[236] Yaxin Ding, Peng Yuan, Jie Yu, Yuting Chen, Pengfei Jiang, Yuan Wang, Yannan Xu, Shuxian Lv, Zhiwei Dang, Boping Wang, et al. Forming-free NbO_x-based memristor enabling low-energy-consumption artificial spiking afferent nerves. *IEEE Transactions on Electron Devices*, 2022.

[237] Xumeng Zhang, Ye Zhuo, Qing Luo, Zuheng Wu, Rivu Midya, Zhongrui Wang, Wenhao Song, Rui Wang, Navnidhi K Upadhyay, Yilin Fang, et al. An artificial spiking afferent nerve based on Mott memristors for neurorobotics. *Nature communications*, 11(1):1–9, 2020.

[238] Mark D McDonnell and Derek Abbott. What is stochastic resonance? definitions, misconceptions, debates, and its relevance to biology. *PLoS computational biology*, 5(5):e1000348, 2009.

[239] Sanjoy Kumar Nandi, Sujan Kumar Das, Yubo Cui, Assaad El Helou, Shimul Kanti Nath, Thomas Ratcliff, Peter Raad, and Robert G Elliman. Thermal conductivity of amorphous NbO_xs thin films and its effect on volatile memristive switching. *ACS Applied Materials & Interfaces*, 2022.

[240] Frank C Hoppensteadt and Eugene M Izhikevich. Oscillatory neurocomputers with dynamic connectivity. *Physical Review Letters*, 82(14):2983, 1999.

[241] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[242] Rodrigo Laje and Dean V Buonomano. Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nature neuroscience*, 16(7):925–933, 2013.

[243] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Updates of equilibrium prop match gradients of backprop through time in an rnn with static input. *Advances in neural information processing systems*, 32, 2019.

[244] 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.

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

[246] Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu. STDP as presynaptic activity times rate of change of postsynaptic activity. *arXiv preprint arXiv:1509.05936*, 2015.

[247] Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu. STDP as presynaptic activity times rate of change of postsynaptic activity approximates backpropagation. *Neural Computation*, 10, 2017.

[248] Jack Kendall, Ross Pantone, Kalpana Manickavasagam, Yoshua Bengio, and Benjamin Scellier. Training end-to-end analog neural networks with equilibrium propagation. *arXiv preprint arXiv:2006.01981*, 2020.

[249] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Equilibrium propagation with continual weight updates. *arXiv preprint arXiv:2005.04168*, 2020.

[250] Benjamin Scellier, Anirudh Goyal, Jonathan Binas, Thomas Mesnard, and Yoshua Bengio. Generalization of equilibrium propagation to vector field dynamics. *arXiv preprint arXiv:1808.04873*, 2018.

[251] Axel Laborieux and Friedemann Zenke. Holomorphic equilibrium propagation computes exact gradients through finite size oscillations. *Advances in Neural Information Processing Systems*, 35:12950–12963, 2022.

[252] Jérémie Laydevant, Maxence Ernoult, Damien Querlioz, and Julie Grollier. Training dynamical binary neural networks with equilibrium propagation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4640–4649, 2021.

[253] Benjamin Scellier, Siddhartha Mishra, Yoshua Bengio, and Yann Ollivier. Agnostic physics-driven deep learning. *arXiv preprint arXiv:2205.15021*, 2022.

[254] Sam Dillavou, Menachem Stern, Andrea J Liu, and Douglas J Durian. Demonstration of decentralized physics-driven learning. *Physical Review Applied*, 18(1):014040, 2022.

[255] Menachem Stern, Daniel Hexner, Jason W Rocks, and Andrea J Liu. Supervised learning in physical networks: From machine learning to learning machines. *Physical Review X*, 11(2):021045, 2021.

[256] Su-in Yi, Jack D Kendall, R Stanley Williams, and Suhas Kumar. Activity-difference training of deep neural networks using memristor crossbars. *Nature Electronics*, 6(1):45–51, 2023.

[257] Jérémie Laydevant, Danijela Markovic, and Julie Grollier. Training an ising machine with equilibrium propagation. *arXiv preprint arXiv:2305.18321*, 2023.

[258] 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.

[259] Stefano Brivio, Sabina Spiga, and Daniele Ielmini. HfO_2-based resistive switching memorydevices for neuromorphic computing. *Neuromorphic Computing and Engineering*, 2022.

[260] S Brivio and S Spiga. Stochastic circuit breaker network model for bipolar resistance switching memories. *Journal of Computational Electronics*, 16(4):1154–1166, 2017.

[261] Karsten Fleck, Camilla La Torre, Nabeel Aslam, Susanne Hoffmann-Eifert, Ulrich Böttger, and Stephan Menzel. Uniting gradual and abrupt set processes in resistive switching oxides. *Physical review applied*, 6(6):064015, 2016.

[262] K-E Harabi, Clement Turck, Marie Drouhin, Adrien Renaudineau, T Bersani-Veroni, D Querlioz, T Hirtzlin, E Vianello, M Bocquet, and J-M Portal. A multimode hybrid

memristor-CMOS prototyping platform supporting digital and analog projects. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pages 184–185, 2023.

[263] Kamel-Eddine Harabi. *Energy Efficient Memristor-Based Artificial Intelligence Accelerators using In/NearMemory Computing*. PhD thesis, Université Paris Saclay, 2023.

[264] T Hirtzlin, Marc Bocquet, M Ernoult, J-O Klein, E Nowak, E Vianello, J-M Portal, and D Querlioz. Hybrid analog-digital learning with differential rram synapses. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 22–6. IEEE, 2019.

[265] Preben Alstrøm and Mogens T Levinsen. Phase-locking structure of "integrate-and-fire" models with threshold modulation. *Physics Letters A*, 128(3-4):187–192, 1988.

[266] Thomas Nowotny, Ramon Huerta, and Mikhail I Rabinovich. Neuronal synchrony: peculiarity and generality. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3), 2008.

[267] Leon O Chua, Y Yao, and Q Yang. Devil's staircase route to chaos in a non-linear circuit. *International Journal of Circuit Theory and Applications*, 14(4):315–329, 1986.

[268] Per Bak. Commensurate phases, incommensurate phases and the devil's staircase. *Reports on Progress in Physics*, 45(6):587, 1982.

Résumé en Français

Introduction et contexte

À l'ère numérique actuelle, la technologie de l'information a été un moteur du progrès mondial. Cependant, cela s'accompagne d'un coût énergétique significatif. La demande croissante de services intensifs en données, tels que l'intelligence artificielle et le "cloud computing", a souligné le besoin urgent de solutions informatiques économes en énergie tout en favorisant l'innovation. Les ordinateurs nous ont propulsés dans une ère de croissance exponentielle en puissance de calcul et en mémoire. Cependant, de nouveaux défis sont présents, en particulier les demandes énergétiques croissantes liées à l'avènement de l'apprentissage profond.

Le calcul neuromorphique, inspiré du cerveau humain, offre des perspectives en tant que solution matérielle écoénergétique. Les algorithmes d'apprentissage et les neurones ou synapses hardware doivent être co-conçus pour obtenir les systèmes les plus efficaces possibles. Les memristors, compatibles avec la technologie CMOS, offrent efficacité, rapidité, mais aussi une non-volatilité pour les synapses et un comportement impulsionnel pour les neurones. L'étude de ces technologies émergentes et de leur compatibilité avec les algorithmes d'apprentissage est une étape importante vers la réalisation de l'apprentissage sur puce. Dans ce travail, nous avons suivi diverses approches :

- **Modélisation et Caractérisation de Dispositifs :** Comme les réseaux neuronaux impulsionnels offrent la promesse d'un apprentissage à faible consommation d'énergie, nous avons exploré des dispositifs neuronaux émergents en tant qu'alternatives au CMOS. Ces dispositifs évolutifs reproduisant des comportements biologiques offrent le potentiel de créer des systèmes complexes imitant le cerveau. Les neurones memristifs basés sur de l'oxyde de niobium ont été étudiés dans le chapitre 2.

- **Adaptation des Algorithmes d'Apprentissage pour l'Implémentation Matérielle :** La rétropropagation du gradient, appelée Back Propagation en anglais, n'est pas un algorithme adapté à l'apprentissage dans un système physique en raison de la non-localité des mises à jour. La propagation de l'équilibre ou Equilibrium Propagation est un algorithme basé sur la physique plutôt que le calcul. Il ouvre la voie à l'utilisation de la physique inhérente à ces systèmes matériels pour l'apprentissage sur puce. Adapter un

algorithme purement mathématique pour une future implémentation matérielle est une étape cruciale vers l'apprentissage sur puce, et cela est étudié dans le chapitre 3.

- **Formation des Algorithmes d'Apprentissage avec des Mesures de Vrais Dispositifs :** Les dispositifs réels sont caractérisés par du bruit, de la variabilité, de la non-linéarité et des comportements que les modèles théoriques ont souvent du mal à capturer. Ainsi, nous soulignons l'importance de travailler avec des données expérimentales pour entraîner des réseaux de neurones, en particulier lorsqu'il s'agit d'applications sur puce. Le chapitre 4 vise à former un réseau de neurones où les synapses sont basées sur des dispositifs réels, ouvrant la voie à l'apprentissage sur puce.

Résultats

Chapitre 2

Le chapitre 2 se concentre sur la caractérisation et la modélisation de neurones composés de memristors volatils filamenteraires à base de NbOx. Ces memristors se révèlent être des candidats prometteurs en raison de leur scalabilité et de leur compatibilité avec les memristors non volatils et les technologies CMOS. L'empilement $Pt/Nb_20_5/Ti/Pt$ présente des caractéristiques I-V correspondant à une résistance différentielle négative en forme de S contrôlée en courant et à une hystérèse ("threshold switching") contrôlée en tension, des phénomènes qui peuvent être efficacement modélisés en considérant la conduction de Poole-Frenkel. Les propriétés dynamiques de ces dispositifs sont particulièrement intéressantes, car leur comportement impulsionnel rappelle celui des neurones biologiques. La forme des impulsions est caractérisée par une dépolarisation initiale suivie d'une hyperpolarisation due à la présence d'une inductance. Ces dispositifs présentent des comportements tels que des caractéristiques leaky-integrate-and-fire (LIF), des impulsions tout-ou-rien et du "phasic bursting". Nous explorons l'origine de ce dernier comportement avec un modèle de dynamique non linéaire. Il émerge comme une interaction complexe entre un point fixe instable (cycle limite) et un point fixe stable (équilibre), provenant de l'effet Poole-Frenkel. Dans ce chapitre, nous décrivons la fabrication et la caractérisation ces neurones, et nous développons un modèle simple basé sur la dynamique non linéaire qui reproduit avec précision les comportements neuronaux mentionnés ci-dessus. Il s'agit d'un outil particulièrement intéressant pour la conception de systèmes informatiques neuromorphiques à impulsions.

Cette figure présente en a les caractéristiques I-V des neurones, en b un exemple de train d'impulsions et en c le phénomène de "phasic bursting".

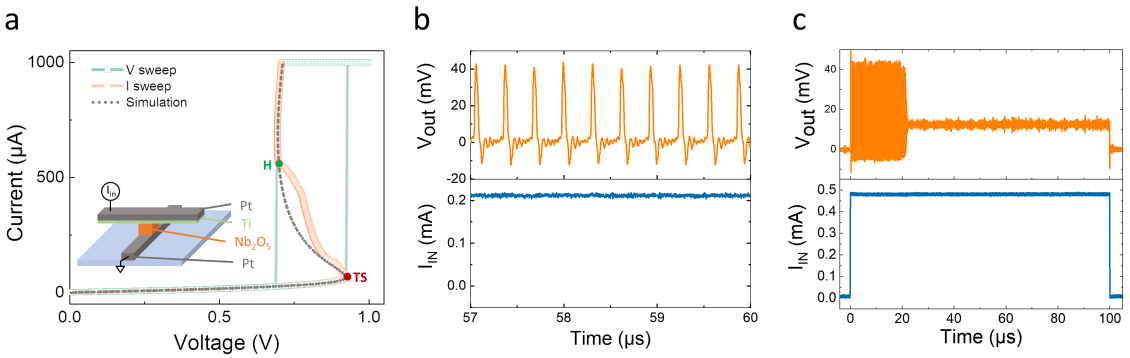


Figure 4.24: a : Caractéristiques I-V controlées en tension (vert) et en courant (orange). La structure des neurones memristifs est présentée en insert. b: Trains d'impulsions lors de l'application d'un courant constant. c: Illustration du phasic busrting lors de l'application d'un courant constant : les impulsions s'arrêtent même sans variation du courant d'entére.

Chapitre 3

Le chapitre 3 adopte une approche différente du calcul neuromorphique, explorant cette fois-ci l'adaptation d'un algorithme d'apprentissage, Equilibrium Propagation (EqProp), à des systèmes physiques. En particulier, nous nous concentrons sur le défi de manipuler des gradients à valeurs réelles dans un environnement basé sur des memristors, plus adapté à une écriture basée sur un nombre entier d'impulsions. Pour relever ce défi, nous étudions différentes approches de discrétisation du gradient. Nous choisissons d'abord d'explorer la discrétisation ternaire, où toutes les synapses au-dessus ou en dessous d'un seuil sont mises à jour. Cette approche donne des précisions très proches de celles de l'algorithme EqProp conventionnel (qui correspond à 99,06 % de précision). Au cours de cette exploration, nous examinons le rôle des hyperparamètres et leur impact sur les performances du réseau. Une autre approche que nous explorons consiste à introduire des probabilités dans le processus de mise à jour. Cette modification améliore non seulement les performances, mais donne également une distribution d'impulsions similaire au scénario idéal non discrétisé. Pour analyser davantage l'impact de la discrétisation, une autre approche consiste à quantifier en davantage d'états pour la discrétisation du gradient. Bien que cette voie montre du potentiel et surpasse l'approche ternaire non probabiliste en termes de performances, elle implique un compromis - une plus grande dispersion des impulsions et, finalement, une consommation d'énergie plus élevée que l'approche ternaire probabiliste.

La figure suivante présente les performances obtenues dans les trois scénarios: en a le cas ternaire non probabiliste, en b le cas ternaire probabiliste et en c le cas d'une quantification à 19 états.

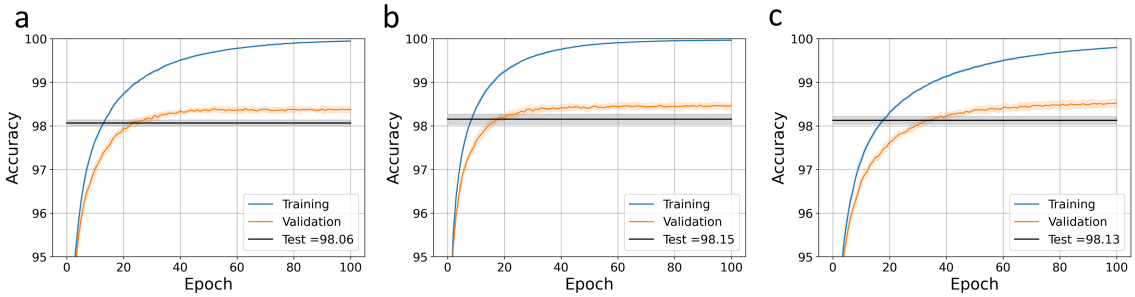


Figure 4.25: Précisions obtenues pour différentes méthodes de discrétisation du gradient. a : Ternarisation du gradient. b : Ternarisation avec mises à jour probabilistes. c : Quantification du gradient (dix-neuf états différents).

Chapitre 4

Le chapitre 4 présente une plateforme hardware composée de memristors filamentaires à base de HfOx. Ces dispositifs peuvent être adressés individuellement, et les mesures dans le régime de "weak reset" sont utilisées pour effectuer l'apprentissage avec de vrais dispositifs. Les neurones sont émulés grâce à un ordinateur, mais les synapses sont basées sur des dispositifs réels. Des contrôles sont d'abord effectués, plus précisément un perceptron à couche unique et un réseau à 2 couches avec la première couche gelée. La précision était respectivement de 75 % et 70 %. Ensuite, un réseau entièrement connecté à une couche cachée a été formé. Deux définitions de poids différentes ont été explorées : la différence linéaire de conductances et la différence logarithmique de conductances. La première a donné de meilleurs résultats (précision de 91 %) que la seconde (précision de 89,5 %). En introduisant un seuil de conductance pour limiter le régime de bruit élevé, la performance a pu être augmentée à 91,75 % pour la définition linéaire et à 92,14 % pour la définition logarithmique.

La figure suivante présente en a et b les performances obtenues respectivement pour la définition linéraire en conductance des poids, et celle logarithmique dans le cas où aucun seuil de conductance n'est imposé. Les figures b) et c présentent les résultats obtenus dans les cas où des seuils sont mis en place pour la définition linéaire ou logarithmique des poids respectivement.

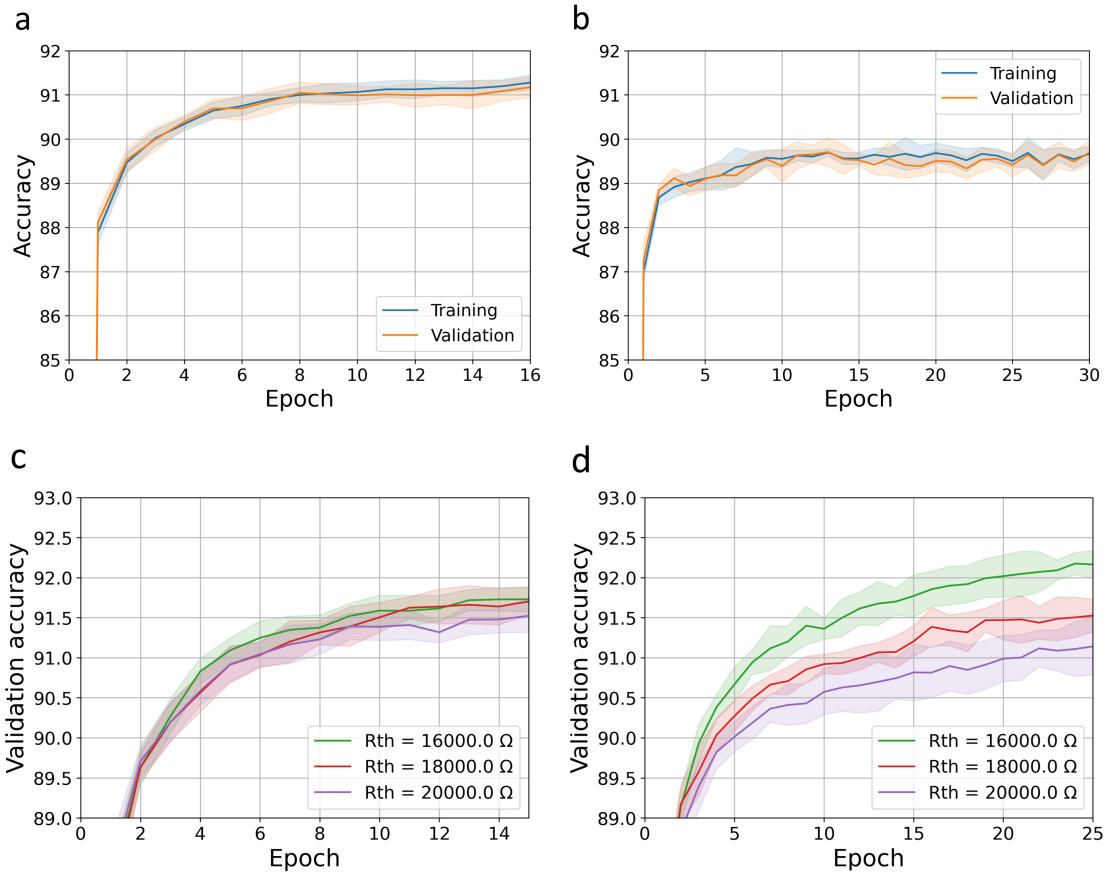


Figure 4.26: Performances obtenues pour un réseau de neurones à une couche cachée. a : Performances obtenues avec une définition linéaire en conductance des poids. b : Performances obtenues avec une définition logarithmique en conductance des poids. c : Performances obtenues avec une définition linéaire en conductance des poids avec seuil sur la conductance, pour trois différentes valeurs du seuil. d: Performances obtenues avec une définition logarithmique en conductance des poids avec seuil sur la conductance, pour trois différentes valeurs du seuil.

En résumé, notre étude a révélé les promesses du calcul neuromorphique et des memristors. Atteindre l'efficacité énergétique et débloquer de nouvelles possibilités computationnelles nécessite une synergie harmonieuse entre le hardawre et le software. En regardant vers le futur, notre quête d'un avenir numérique plus vert et plus innovant se poursuit.

Conclusion

Plusieurs grands axes de recherches pourraient être explorés. Au niveau des composants, l'amélioration des dispositifs memristifs pour une utilisation analogique, avec des évolutions linéaires et une variabilité réduite, peut significativement améliorer les performances du réseau. D'un point de vue algorithmique, comme mentionné dans le chapitre 4, l'adaptation de l'algorithme Equilibrium Propagation pour atténuer les variabilités intra et inter-dispositifs offre un grand potentiel. De plus, bien que cette thèse se soit principalement concentrée sur l'apprentissage supervisé avec des ensembles de données étiquetés, les travaux futurs pourraient explorer l'apprentissage non supervisé ou semi-supervisé avec l'Équilibrium Propagation, abordant des scénarios avec des données limitées ou non étiquetées.

En conclusion, la convergence de systèmes informatiques hautement paramétrables, en particulier le calcul neuromorphique, avec des algorithmes ancrés dans la physique offre un grand potentiel pour l'avancement de l'IA ultra-basse consommation. L'investigation de la mise en œuvre d'Equilibrium Propagation et l'amélioration de la robustesse de l'algorithme sont en phase avec la demande croissante d'une IA efficace, promettant des percées innovantes à l'intersection de la physique et de l'intelligence artificielle.