Bio-inspired continual learning and credit assignment for neuromorphic computing

Axel Laborieux

Thèse de doctorat

NNT : 2021UPAST095

université PARIS-SACLAY

Bio-inspired Continual Learning and Credit Assignment for Neuromorphic Computing

Apprentissage continu et estimation du gradient inspirés de la biologie pour le calcul neuromorphique

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°575 : electrical, optical, bio : physics and engineering (EOBE)
Spécialité de doctorat : Physique
Unité de recherche : Université Paris-Saclay, CNRS, Centre de Nanosciences et de Nanotechnologies, 91120, Palaiseau, France.
Référent : Faculté des Sciences d'Orsay

Thèse présentée et soutenue à Paris-Saclay, le 06/10/2021, par

Axel LABORIEUX

Composition du Jury

Julie GROLLIER Directrice de recherche, Unité Mixte de Physique CNRS, Thales	Présidente
Emre NEFTCI Professeur, Forschungzentrum Jülich, Germany	Rapporteur & Examinateur
Daniel BRUNNER Chargé de recherche, HDR, FEMTO-ST, Besançon, France	Rapporteur & Examinateur
Robert LEGENSTEIN Professeur, Graz University of Technology Bestätigte	Examinateur

Direction de la thèse

Damien QUERLIOZ Chargé de recherche, Centre de Nanosciences et Nanotechnologie	Directeur de thèse
Liza HERRERA-DIEZ Chargée de recherche, Centre de Nanosciences et Nanotechnologie	Co-Encadrante & Examinatrice

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

Titre : Apprentissage continu et estimation du gradient inspirés de la biologie pour le calcul neuromorphique

Mots clés : Neuromorphique, apprentissage profond, réseaux de neurones quantifiés, apprentissage continu, estimation du gradient

Résumé : Les algorithmes d'apprentissage profond permettent aux ordinateurs de réaliser des tâches cognitives allant de la vision à la compréhension du langage naturel avec une performance comparable à celle des humains. Bien que ces algorithmes s'inspirent conceptuellement du cerveau, leur consommation énergétique est supérieure par plusieurs ordres de grandeur. La raison de cette surconsommation énergétique est à la fois architecturale et algorithmique. L'architecture des ordinateurs sépare physiquement les unités de calcul et de mémoire où les données sont stockées. Cette séparation provoque un déplacement de données particulièrement intense et coûteux en énergie pour les algorithmes d'apprentissage machine, ce qui limite les applications embarquées ou à faible budget énergétique. Une solution consiste à créer de nouvelles architectures neuromorphiques où la mémoire est au plus près des unités de calcul. Cependant, les algorithmes d'apprentissage existants possèdent des limitations qui rendent leur implémentation sur puce neuromorphique difficile. En particulier, les limitations algorithmiques au cœur de cette thèse sont l'oubli catastrophique et l'estimation non locale du gradient. L'oubli catastrophique concerne l'impossibilité de conserver la performance d'un réseau de neurones lorsqu'une nouvelle tâche est apprise. Le calcul du gradient dans les réseaux de neurones est effectué par la Backpropagation. Bien qu'efficace, cet algorithme est difficile à implémenter sur une puce neuromorphique car il nécessite deux types de calculs distincts. Ces concepts sont présentés en détail dans le chapitre 1 de la thèse. Le chapitre 2 présente un algorithme inspiré de la métaplasticité synaptique pour réduire l'oubli catastrophique dans les réseaux de neurones binaires. Les réseaux de neurones binaires sont des réseaux de neurones artificiels avec des poids et activation binaires, ce qui les rend attrayants pour les applications neuromorphiques. L'entraînement des poids synaptiques binaires nécessitent des variables cachées dont la signification est mal comprise. Nous montrons que ces variables cachées peuvent être utilisées pour consolider les synapses importantes. La règle de consolidation présentée est locale à la synapse, tout en étant aussi efficace qu'une méthode d'apprentissage continue établie dans la littérature. Le chapitre 3 s'intéresse à l'estimation locale du gradient pour l'apprentissage. Equilibrium Propagation est un algorithme d'apprentissage qui ne nécessite qu'un seul type de calcul pour estimer le gradient. Toutefois, son passage à l'échelle sur des tâches complexes et architectures profondes restent à démontrer. Dans ce chapitre, résultant d'une collaboration avec le Mila, nous montrons qu'un biais dans l'estimation du gradient empêche ce passage à l'échelle, et nous proposons un nouvel estimateur non biaisé qui permet de passer à l'échelle. Nous montrons aussi comment adapter l'algorithme pour optimiser l'entropie croisée au lieu du coût quadratique. Enfin, nous étudions le cas où les connexions synaptiques sont asymétriques. Ces résultats montrent que Equilibrium Propagation est un algorithme prometteur pour l'apprentissage sur puce. Enfin, dans le chapitre 4, nous présentons une architecture pour implémenter des synapses ternaires à l'aide de mémoires résistives à base d'oxyde d'Hafnium en collaboration avec l'université d'Aix Marseille et le CEA-Leti de Grenoble. Nous adaptons un circuit initialement prévu pour implémenter un réseau de neurone binaire en montrant qu'une troisième valeur de poids synaptique peut être codée en exploitant le un régime où la tension d'alimentation est basse, ce qui est particulièrement adapté pour les applications embarquées. Les résultats présentés dans cette thèse montrent que la conception jointe des algorithmes et des architectures de calcul est cruciale pour les applications neuromorphiques.

Maison du doctorat de l'Université Paris-Saclay
2ème étage aile ouest, Ecole normale supérieure Paris-Saclay
4 avenue des Sciences,
91190 Gif sur Yvette, France

Title: Bio-inspired Continual Learning and Credit Assignment for Neuromorphic Computing

Keywords: Neuromorphic, deep learning, quantized neural networks, continual learning, credit assignment

Abstract: Deep learning algorithms allow computers to perform cognitive tasks ranging from vision to natural language processing with performance comparable to humans. Although these algorithms are conceptually inspired by the brain, their energy consumption is orders of magnitude higher. The reason for this high energy consumption is both architectural and algorithmic. The architecture of computers physically separates the processor and the memory where data is stored. This separation causes particularly intense and energy-intensive data movement for machine learning algorithms, limiting on-board or low-energy budget applications. One solution consists in creating new neuromorphic architectures where the memory is as close as possible to the computation units. However, existing learning algorithms have limitations that make their implementation on neuromorphic chips difficult. In particular, the algorithmic limitations at the heart of this thesis are catastrophic forgetting and non-local credit assignment. Catastrophic forgetting concerns the inability to maintain the performance of a neural network when a new task is learned. Credit assignment in neural networks is performed by Backpropagation. Although efficient, this algorithm is challenging to implement on a neuromorphic chip because it requires two distinct types of computation. These concepts are presented in details in chapter 1 of this thesis. Chapter 2 presents an algorithm inspired by synaptic metaplasticity to reduce catastrophic forgetting in binarized neural networks. Binarized neural networks are artificial neural networks with binary weights and activation, which makes them attractive for neuromorphic applications. The training process of binarized synaptic weights requires hidden variables whose meaning is poorly understood. We show that these hidden variables can be used to consolidate important synapses. The presented consolidation rule is local to the synapse, while being as effective as an established continual learning method of the literature. Chapter 3 deals with the local estimation of the gradient for training. Equilibrium Propagation is a learning algorithm that requires only one type of computation to estimate the gradient. However, scaling it up to complex tasks and deep architectures remains to be demonstrated. In this chapter, resulting from a collaboration with the Mila, we show that a bias in the estimation of the gradient is responsible for this limitation, and we propose a new unbiased estimator that allows Equilibirum propagation to scale up. We also show how to adapt the algorithm to optimize the cross entropy loss instead of the quadratic cost. Finally, we study the case where synaptic connections are asymmetric. These results show that Equilibrium Propagation is a promising algorithm for on-chip learning. Finally, in Chapter 4, we present an architecture to implement ternary synapses using resistive memories based on Hafnium oxide in collaboration with the University of Aix Marseille and CEA-Leti in Grenoble. We adapt a circuit originally intended to implement a binarized neural network by showing that a third synaptic weight value can be encoded when exploiting the low supply voltage regime, which is particularly suitable for on-board applications. The results presented in this thesis show that the joint design of algorithms and computational architectures is crucial for neuromorphic applications.

Maison du doctorat de l'Université Paris-Saclay
2ème étage aile ouest, Ecole normale supérieure Paris-Saclay
4 avenue des Sciences,
91190 Gif sur Yvette, France

Acknowledgements

The scientific results presented in this manuscript would have not been possible without the support I received from many people inside and outside the lab, as well as the funding from the European Research Council. I would like to thank first my supervisor Dr. Damien Querlioz for his unconditional support and guidance throughout my PhD. I am grateful for your close yet non-invasive supervision which provided me with the right conditions to develop and grow as a scientist. I would like to thank Dr. Liza Herrera-Diez for her co-supervision. Although several unforeseen events limited the scope of the initial project, I still learned a lot from you and benefited from your positive influence. Many thanks to Prof. Emre Neftci, Dr. Daniel Brunner and Prof. Robert Legenstein for accepting to take part in my PhD committee and carefully reviewing this manuscript. Thanks also to the anonymous reviewers whose reviews have improved the quality of the research articles on which this thesis manuscript is built.

I would then like to acknowledge my collaborators outside the lab. Thanks to Dr. Julie Grollier, Benjamin Scellier, and Prof. Yoshua Bengio for collaborating remotely on the EqProp project in such troubled times. Thanks also to Drs. Marc Bocquet and Jean-Michel Portal from Aix Marseille University, as well as Drs. Elisa Vianello and Etienne Nowak from CEA Leti for collaborating on the Ternary synapse project.

Next I want to thank my dear colleagues and friends who made the office such an amazing workplace during those three and a half years. Thanks a lot Maxence for everything you have done for me, ranging from your invaluable scientific contributions all the way to the countless opportunities and suggestions you made to improve my work. Thanks Tifenn for your curiosity-driven experiments that led me to delve into BNNs at the beginning of the PhD, it is impressive to look back at the road travelled since then. Thanks Kamel for being such a wonderful person, you have a positive influence on everyone you encounter, and I am sure that it goes also for the chips you design. Thanks Guillaume for all the discussions about continual learning, I wish you had been in our office right from the start rather than later on. Thanks Bogdan for democratizing PyTorch in the team back then, and for occasionally teaching me chess the hard way. Thanks Mamour for bringing such a peaceful atmosphere in the office despite your insane commutes. Thanks Xing for your great contribution to the team despite such difficult times, I hope that we can meet again in the future. Thanks Guanda for your kind donations during the first wave of the pandemic. Many thanks Marie for your good mood everyday, it has been a pleasure to share an office with you while writing this manuscript. Thanks Atreya for being always ready to help your teammates, you really make a difference in the team. Thanks Clément for your expertise in data recovery, and for refraining from making fun of people who break servers (shame to them). Thanks Rohit for your contagious calm, I will miss having coffee with you after lunch. Thanks to Gyan for organising pizza lunches and the Euro 2020 Sweepstakes. Special thanks to Thibaut for taking care of streaming my PhD defense while having a master thesis to write, I wish you the best for your future plans. Big thanks to Maryam for your

help and support prior to the defense, it really did make a difference.

I would also like to acknowledge the people from Thales for organizing interesting meetings, workshops, and webinars, which always resulted in useful discussions. Thanks to Alice Mizrahi, Danijela Marković, Jérémie Laydevant, Nathan Leroux, and Erwann Martin.

I also want to thank all the people working at the C2N who contribute to making the lab run smoothly. Many thanks to Christophe Chassat and Alain Péan from IT, Lydia Andalon, Laurence Sidibé, and Bernadette Laborde from the administration, Sophie Bouchoule and Emilia Davodeau from the doctoral school. Thanks also to the employees cleaning the lab every morning. Thanks also to Valérie Fortuna, Elisabeth Delbecq and Sylvie Sikora from the MISS. It has been a truly fulfilling experience to introduce scientific research and critical thinking to so many kids during three years. I like to believe that it ignited a few vocations for scientific research. Finally, I want to thank my beloved wife Yawen for her unfailing support and affection, and my parents and all my family for encouraging me to pursue my dreams.

Contents

Introduction **1**

1 Neuromorphic Computing and Deep Learning **5**
1.1 Principles of Computation by Computers and the Brain 6
1.1.1 How Computers Work 6
1.1.2 The Birth of Neuromorphic Engineering 8
1.2 Artificial Neural Networks and Machine Learning 10
1.2.1 Early Networks 10
1.2.2 The Machine Learning Approach to AI 12
1.2.3 Modern Deep Networks 16
1.3 Major Differences Between Artificial Neural Networks and the Brain can Inspire Research in Deep Learning 18
1.3.1 Does the Brain use Back-propagation? 18
1.3.2 Beyond back-propagation in Rate-based Neural Networks 20
1.3.3 Memory and Forgetting in Artificial Neural Networks 26
1.4 Current Dedicated Hardware for AI 33
1.4.1 Neuroscience-based Hardware 33
1.4.2 Deep learning-based Hardware 34

2 Synaptic Metaplasticity in Binarized Neural Networks **41**
2.1 Background 42
2.2 Interpreting the hidden weights of binarized neural networks as metaplasticity states 46
2.3 Multitask learning with metaplastic binarized neural networks 48
2.4 Stream learning: learning one task from subsets of data 55
2.5 Mathematical interpretation 56
2.6 Increasing Synapse Complexity for Steady-State Continual Learning 59
2.7 Discussion 62
2.8 Methods 65

3 Scaling Equilibrium propagation to Deep ConvNets 69
3.1 Introduction . . . 70
3.2 Background . . . 72
3.2.1 Convergent RNNs With Static Input . . . 72
3.2.2 Training Procedures For Convergent RNNs . . . 72
3.2.3 Convolutional Architectures for Convergent RNNs . . . 74
3.2.4 Equilibrium Propagation with unidirectional synaptic connections . . . 75
3.3 Improving EP Training . . . 75
3.3.1 Reducing bias and variance in the gradient estimate of the loss function . 76
3.3.2 Changing the loss function . . . 77
3.3.3 Changing the learning rule of EP with unidirectional synaptic connections 78
3.4 Results . . . 79
3.4.1 ConvNets with bidirectional connections . . . 80
3.4.2 ConvNets with unidirectional connections . . . 81
3.5 Discussion . . . 82

4 Implementation of Ternary Weights 85
4.1 Background . . . 86
4.2 The Operation of A Precharge Sense Amplifier Can Provide Ternary Weights . . . 91
4.3 Impact of Process, Voltage, and Temperature Variations . . . 96
4.4 Programmability of Ternary Weights . . . 98
4.5 Network-Level Implications . . . 100
4.6 Comparison with Three-Level Programming . . . 103
4.7 Conclusion . . . 104

Conclusions and future work 105

Synthèse en Français 111

List of publications 117

Bibliography 140

A Synaptic Metaplasticity in Binarized Neural Networks 141
A.1 Forward and backward propagation in binarized neural networks . . . 141
A.2 Training parameters . . . 144
A.3 Implementation of Synaptic Intelligence . . . 144
A.4 Use of a metaplasticity function f_{meta} featuring a hard threshold . . . 147
A.5 Mathematical proofs . . . 148

A.6 Comparison with learning rate decay . . . 152
A.7 Sequential Training of the MNIST and Fashion-MNIST Datasets . . . 153
A.8 Sequential Training of the MNIST and USPS Datasets . . . 154
A.9 Class Incremental Learning . . . 155
A.10 Increasing Synapse Complexity . . . 157

B Scaling Equilibrium Propagation to Deep ConvNets 161
B.1 Gradients of BPTT . . . 161
B.2 Error terms in the estimates of the loss gradient . . . 161
B.3 Pseudo code . . . 163
B.3.1 Random one-sided estimation of the loss gradient . . . 163
B.3.2 Symmetric difference estimation of the loss gradient . . . 163
B.4 Convolutional Recurrent Neural Networks . . . 164
B.4.1 Definition of the operations . . . 164
B.4.2 Convolutional RNNs with symmetric connections . . . 165
B.4.3 Convolutional RNNs with asymmetric connections . . . 168
B.4.4 Random-sign estimate variance . . . 171
B.4.5 Adding dropout . . . 171
B.4.6 Changing the activation function . . . 172
B.5 Weight alignment for asymmetric connections . . . 173
B.6 Layer-wise comparison of EP estimates . . . 173

C Implementation of Ternary Weights 175
C.1 Training Algorithm of Binarized and Ternary Neural Networks . . . 175

List of Figures

1 Neuromorphic Computing and Deep Learning **5**

1.1 Comparison between a CPU and a GPU. 7
1.2 Cartoon of a biological neuron and synapse. 8
1.3 Perceptron and Hopfield Network. 10
1.4 Diagram of automatic differentiation. 15
1.5 Composition of convolutions. 17
1.6 Contrastive Hebbian learning. 22
1.7 Equilibrium Propagation and Target Propagation. 25
1.8 Continual learning in artificial neural networks. 27
1.9 Different types of memristors. 36
1.10 Schematic of the crossbar elementary unit. 37
1.11 Spintronics for neuromorphic computing. 38

2 Synaptic Metaplasticity in Binarized Neural Networks **41**

2.1 Problem setting and illustration of our approach. 45
2.2 Permuted MNIST learning task. 50
2.3 Influence of the network size on the number of tasks learned. 51
2.4 Sequential learning on various datasets. 54
2.5 Stream learning experiments. 56
2.6 Difference between standard and binarized optimization. 57
2.7 High hidden weights correspond to important parameters. 59
2.8 Complex synapse model. 62

3 Scaling Equilibrium propagation to Deep ConvNets **69**

3.1 Schematic of the convolutional architecture. 74
3.2 The symmetric gradient estimate. 76
3.3 Comparison between free dynamics for two loss functions. 77
3.4 Training curves on CIFAR-10. 80
3.5 Alignment between forward and backward weights. 82

4 Implementation of Ternary Weights **85**
4.1 Presentation of the device. . . . 90
4.2 Schematic of the precharge sense amplifier. . . . 91
4.3 Circuit simulation of the precharge sense amplifier. . . . 92
4.4 Four distinct programming conditions. . . . 94
4.5 Synaptic weights measured by the on-chip sense amplifier. . . . 95
4.6 Impact of process, voltage, and temperature variations. . . . 97
4.7 Distributions of resistance states. . . . 98
4.8 Comparison between BNN and TNN performance. . . . 100
4.9 Impact of bit error rate on TNNs and BNNs. . . . 102

A Appendix A **141**
A.1 Synaptic intelligence in BNNs. . . . 146
A.2 Comparison of different choices for f_{meta}. . . . 147
A.3 MNIST/Fashion-MNIST sequential learning. . . . 154
A.4 MNIST and USPS training examples. . . . 155
A.5 Class Incremental Learning. . . . 157
A.6 Stationary distributions of hidden variables. . . . 159

B Appendix B **161**
B.1 Random sign estimate training curves. . . . 172
B.2 EP symmetric estimates layer by layer. . . . 174

List of Tables

2 Synaptic Metaplasticity in Binarized Neural Networks **41**
2.1 Binarized neural network test accuracies on six permuted MNISTs. 49

3 Scaling Equilibrium propagation to Deep ConvNets **69**
3.1 Comparison between EP and BPTT on CIFAR-10. 80
3.2 Hyper-parameters used for the CIFAR-10 experiments. 81

4 Implementation of Ternary Weights **85**
4.1 Truth tables of the XNOR and GXNOR gates. 89
4.2 Error rates on ternary weights measured experimentally. 96
4.3 Gain in test accuracy of TNNs over BNNs. 101

A Appendix A **141**
A.1 Hyperparameters for the permuted MNISTs experiment. 143
A.2 Hyperparameters for the permuted FMNIST-MNIST experiment. 143
A.3 Hyperparameters for the stream learning experiment. 143
A.4 Permuted MNIST experiment with learning rate decay. 153
A.5 Ablation study of the feedback process for the complex synapse model. 158

B Appendix B **161**
B.1 Comparison between random-sign and symmetric estimates. 172

Nomenclature

Abbreviations

2T2R	Two transistors, two resistors.
BNN	Binarized neural network.
CMOS	Complementary metal oxide semi-conductor.
CPU	Central processing unit.
DRAM	Dynamic random access memory.
ECC	Error-correcting code.
EP	Equilibrium Propagation
GPU	Graphics processing unit.
HRS	High resistance state.
iCarl	Incremental classifier for representation learning.
LIF	Leaky integrate and fire.
LRS	Low resistance state.
MCU	Micro controller unit.
OPU	Optical processing unit.
PCM	Phase change memory.
PCSA	Precharge sense amplifier.
RRAM	Resistive random access memory.
SoC	System on chip.
SRAM	Static random access memory.

STT Spin-transfer torque.

TCAM Ternary content-addressable memory.

TNN Ternarized neural network.

TPU Tensor processing unit.

Introduction

COMPUTERS are now able to perform challenging cognitive tasks related to perception ranging from vision to natural language and speech understanding. The algorithms underlying these breakthroughs are deep artificial neural networks, whose principles are loosely inspired by the human brain. However, the energy consumption of deep networks is orders of magnitude higher than the brain. This inefficiency originates from the architecture of modern computers based on the physical separation of the processing unit and the data storage unit (the von Neumann architecture), which is not adapted to emulate artificial neural networks at a low energy cost: artificial neural networks require big amounts of data to transit back and forth between both units. This transfer makes up for the main part of the energy consumption in modern artificial neural networks. This high energy consumption limits the scope of applications to powerful and centralized devices. In particular, the deployment of artificial intelligence at the edge for medical applications or internet of things is made difficult by the energy budget.

One path to solve this challenge is to design computing architectures where memory storage is tightly integrated with processing units. This architecture principle is found in biological brains, where neurons are computing units, and synapses are responsible for long-term memory storage. The field of neuromorphic computing[1] aims at replicating this principle at the hardware level. This approach comes with the challenge of finding algorithms that are mindful of hardware constraints and easy to embed on dedicated hardware. The conventional algorithms of deep learning are to a large extent not appropriate for neuromorphic computing. In particular, two limitations of current deep artificial neural networks will be central to this thesis:

- The non locality of the learning rule provided by the error back-propagation algorithm makes it very non-optimal to implement in neuromorphic hardware associating tightly computing and memory.
- Catastrophic forgetting, which designate the tendency of artificial neural networks to forget quickly what they have learned in the past when trained on new data, is a major concern for neuromorphic hardware, limiting their ability to learn new information online.

By contrast, brains learn with local learning rules, based on neuron dynamics, and seem largely immune to the issue of catastrophic forgetting. Therefore, taking direct inspiration from the brain might be a major lead to adapt ideas of deep learning to neuromorphic hardware. The idea to mimic the functioning of neurons to design neuromorphic hardware dates back to the 1980s. Unfortunately, the specific algorithms implemented in the brain still remain largely unknown. The main idea of this thesis is that bridging the gap between biological inspiration and effective algorithms is a promising route towards modern neuromorphic hardware. Another aspect of neuromorphic computing is taking advantage of emerging materials to emulate the

[1] In this thesis, we take a broad definition of 'neuromorphic' as describing not only systems closely mimicking biological neurons, but also efficient systems performing meaningful global computation similar to neural ensembles.

computing circuits that are usually simulated by general purpose computers. In this thesis, I present three research projects addressing these two complementary aspects. Chapter 2 and 3 describe biologically inspired algorithms mindful of hardware constraints, and focus on the two limitations mentioned above. Chapter 4 takes advantage of emerging memories to implement an existing algorithm.

More specifically, Chapter 1 introduces in details the context of this thesis. After describing the architecture of modern computers and the principles of neuromorphic computing, I review artificial neural networks from their introduction to their recent breakthroughs. I go on to review two characteristics of artificial neural networks that limit the design of efficient hardware, and the existing solutions in the literature. The first one is the non-locality of error-backpropagation to perform credit assignment, which prevents efficient on-chip learning. The second one is the catastrophic forgetting property: when a network has already been trained to perform a task, learning a new task makes the network forget rapidly the first task. Finally, I review the state-of-the-art of current dedicated hardware for artificial intelligence.

In chapter 2, I present a work bridging computational neuroscience and deep learning that reduces catastrophic forgetting in binarized neural networks, a low precision version of artificial neural networks promising for neuromorphic applications. I show that the hidden variable associated with each binarized parameter during the training process is a relevant quantity for performing synaptic consolidation and reducing catastrophic forgetting. I study mathematically a toy problem of binarized optimization to provide insight as to why the hidden variables are correlated to the importance of the binarized weights. Based on this finding, I propose a simple consolidation mechanism to perform continual learning of several tasks. The method can do almost as well as elastic weight consolidation on the permuted MNIST continual learning benchmark, but without resorting to extra compute in-between tasks. The fact that this method does not need task boundaries allows me to explore a new learning setting where one task is learned by sequentially learning several sub-sets of the data, a relevant setting for edge applications. This work was published in Nature Communications [1] and presented as a poster contribution to Cosyne 2021 and a CVPR 2021 workshop on binarized networks.

In chapter 3, I present a work done in collaboration with the MILA on Equilibrium Propagation (EP), a more biologically plausible and hardware-friendly alternative to back-propagation through time with theoretical guarantees. We show that the gradient estimator of EP in its original formulation contains a bias that prevents EP from scaling to deeper networks. We propose a new unbiased gradient estimator and show that EP can successfully train deep architectures on challenging vision tasks, closely matching back-propagation through time. In addition, we show how to optimize the cross entropy loss function with EP. Finally, we study the setting where forward connections are distinct from backward connections and show that adding an alignment mechanism allows performance to be recovered. This work was published in Frontiers in Neuroscience [2] and presented as a poster contribution to a NeurIPS 2020 workshop.

In chapter 4, I present a work done in collaboration with CEA-Leti and Aix-Marseille Uni-

versity. We propose a method to implement ternarized weights using emerging resistive memories. To do so, we build on a design introduced in previous work to encode binarized weights and show that we can implement a third weight value when operating the circuit in the low supply voltage regime. We present simulations and experiments on a hybrid CMOS/RRAM chip using a 130 nm process. We show on a vision task that ternarized neural networks consistently outperform binarized neural networks. More importantly, we show that ternarized neural networks are resilient to the new type of read errors introduced by the third weight value, keeping their advantage over binarized neural networks. This work is in the proceedings of the IEEE International Conference on Artificial Intelligence Circuits And Systems 2020 (AICAS 2020) [3], and an extended version is published in the IEEE Transactions on Circuits And Systems I (TCAS I) [4].

Chapter 1

Neuromorphic Computing and Deep Learning

> "At the conceptual level, they take their overall strategy from the nervous systems of animals. On the implementation level, however, they are still a brute-force strategy, using little of the cleverness of nature."
>
> Carver MEAD on current commercial neural networks [5]

OVER the last decade, computer programs have been able to perform well at complex cognitive tasks for the first time in human history [6]. These programs can compete with and even outperform humans at very specific tasks. However, they consume orders of magnitude more energy than living agents to do so. In this first chapter, we review the fundamental reasons for this energy gap and introduce neuromorphic computing as a major route towards low-energy implementation of artificial intelligence. We review how this great challenge is tackled from both the hardware and algorithmic points of view, and how the right path will most likely be reached by acting on both aspects, calling for hardware and algorithmic co-design.

1.1 Principles of Computation by Computers and the Brain

In this section, we briefly describe how computation is performed by conventional processing units found in computers, and how their architectures is not adapted to efficiently implement deep neural network algorithms. The field of neuromorphic engineering is introduced as a path toward efficient neural processing.

1.1.1 How Computers Work

The conceptual ancestor behind modern computers is the Turing machine [7]. A Turing machine works with a tape of memory containing input data and a head moving on the tape according to a separated set of fixed instructions. Turing machines can perform almost any computation, provided the right set of instruction and enough memory. The universal Turing machine is a unique machine that can simulate any regular Turing machine by adding the instruction set into the memory along with input data [7].

This seminal concept paved the way for modern stored-programs computers. The hardware substrate that enabled the realisation of such machines at large-scale is the transistor, together with the digital encoding of the data with binary values. The architecture of computers follows the so-called von Neumann architecture, which is characterized by the physical separation of the processing unit and the memory unit. There exist two main types of processing units in modern computers whose simplified architectures are depicted in Fig. 1.1. While they both rely on evolutions of the von Neumann architecture, they are designed for different purposes.

1.1.1.1 The Central Processing Unit

The central processing unit (CPU) is the main processing unit of modern computers. Its architecture is optimized to excel at executing sequences of complex operations called threads. One major challenge for CPUs is to reduce the latency associated with memory accesses. This is done by devoting many transistors to flow control as well as dividing the memory into several layers trading off read speed, density and volatility. The memory in a computer can be

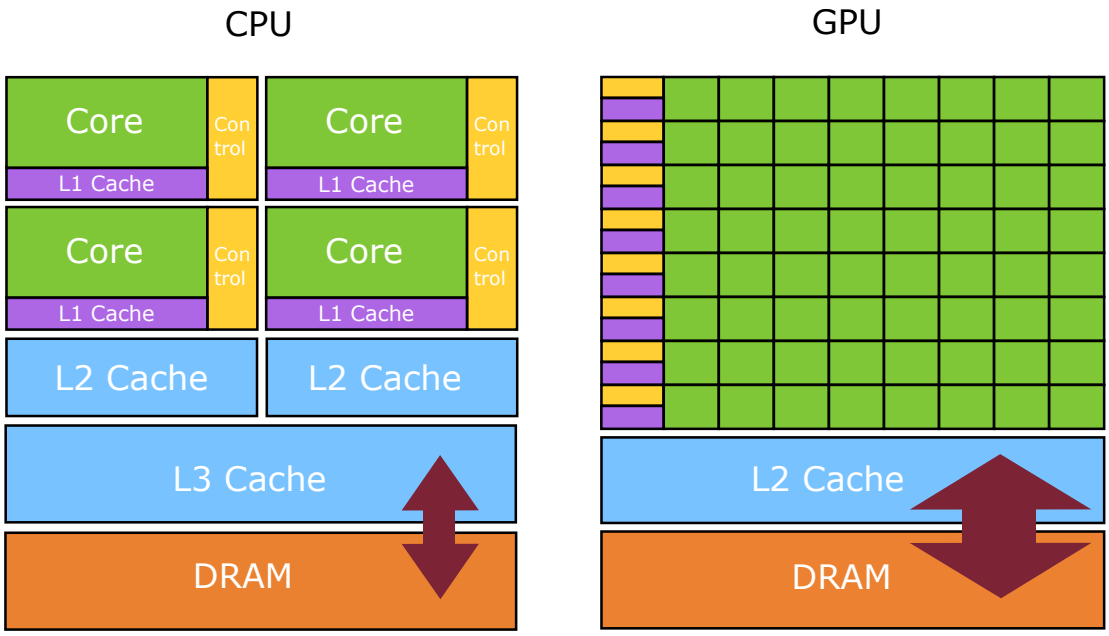

Figure 1.1: Comparison between simplified CPU and a GPU architectures, adapted from the NVIDIA documentation.

separated into three main groups. Non volatile memories such as magnetic hard drive and solid-state drives based on FLASH memory are used for static storage of programs and data when the power is off. They feature cheap storage but long read times, of 20 µs for FLASH and 1 ms for magnetic hard drives. The second group of memory is the so-called main memory or DRAM (dynamic random access memory) depicted in orange in Fig. 1.1. It is volatile and dense, because each memory cell is made of one transistor and one capacitor, and the reading time is 50 ns. The third group comprises the different levels of cache made of static random access memory (SRAM). They are less dense than DRAM, but the access times of the different caches are less than 5 ns. The CPU manages the data on these different memory groups in order to optimize the overall performance.

1.1.1.2 The Graphical Processing Unit

While an upscale multicore CPU can execute tens of threads in parallel, state-of-the-art graphical processing units (GPU), originally designed to render the pixel intensities on the screen, can execute thousands of slower threads in parallel. Far more resources are allocated for data processing rather than flow control and data caching, as we can see from Fig. 1.1. Therefore, the GPU takes advantage of parallelization and larger memory throughput at the cost of more latency for data access. The choice of using the CPU or a GPU to perform a computing task will thus depend on how much data is fed to the same instruction, which can then be executed in parallel. The Amdahl law describes the total computing time of an algorithm given

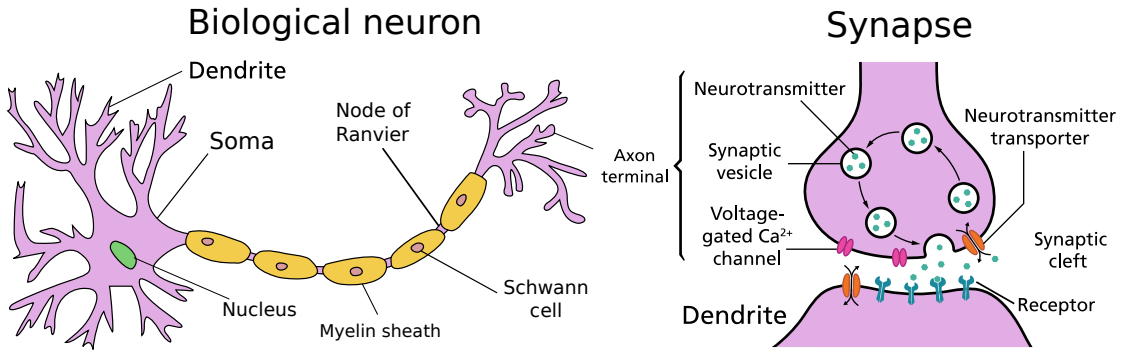

Figure 1.2: Biological neuron and synapse, adapted from wikipedia.

the amount of serial and parallel operations.

A new application for GPU appeared during the last decade in the name of deep learning [6]. This new use case is so important that the stock of the NVIDIA company went from USD 20 in April 2011 to USD 613 as of April 2021, and deep learning is now driving the development of dedicated GPUs like the Tesla V100, and even upscale servers like the NVIDIA DGX A100. However, while the state-of-the-art performance of AI models increases year after year, so does the energy required to train and deploy those models [8]. We can understand this energy issue by looking back at Fig. 1.1. The huge datasets (>>100GB) used in deep learning cannot possibly fit into the RAM of an upscale GPU (≈ 20GB); sometimes even deep learning models themselves cannot fit. Large-scale applications will typically use FLASH storage for the dataset and optimized batch scheduling to reduce latency between training iterations, but this does not reduce the energy cost of moving data back and forth. The model parameters also induce data movements as they need to be read from memory to compute the gradient of the metric one wish to optimize, and written back to memory afterward for each iteration. This data bottleneck is known as the von Neumann bottleneck and is the reason for the high energy consumption of modern deep learning.

However, artificial neural networks are merely simulated by computers, and could be more energy-efficient if emulated on dedicated physical substrates. This observation is at the root of neuromorphic engineering.

1.1.2 The Birth of Neuromorphic Engineering

Neuromorphic engineering was created in the 1980s by Carver Mead [5]. Mead had understood the limits of digital CMOS technology in terms of scaling for very large scale computing systems [9] and was looking for other computing approaches. The path he took was to use analog sub-threshold CMOS to replicate how biological tissues such as biological neurons and synapses process sensory inputs and information.

Neurons use action potentials or spikes to communicate with each other and process information. The majority of neurons act as integrators, and their main behavior can be described

in the following way: incoming spikes arrive to the dendrites of a given neuron (see Fig. 1.2) and are integrated by the soma. If the voltage reaches a certain threshold, the neuron will emit a spike through its axon. When the spike reaches the synapses at the axon terminal, the synaptic vesicles will release the neurotransmitters toward the synaptic cleft. The neurotransmitters will then be received into the receptors of subsequent neurons dendrites, causing the opening of ion channels and the propagation of the signal. There are many kinds of neurotransmitters and receptors, and they are not only responsible for the inhibition or excitation of the post synaptic neuron, but they also trigger changes on longer timescales. The simplistic view is that the tunable synaptic connections between neurons encode for our long-term memories.

Neuroscientists have developed theoretical models to describe how neurons work. A model already described by Louis Lapicque in 1907 is the 'leaky, integrate and fire' neuron [10]. However, biological neurons are more complex than this simple model and involve many biochemical processes. The resting membrane is the result of a difference in sodium and potassium ions concentrations between the intra and extra cellular mediums. This differences in concentration evolve thanks to ion channels who let specific ions flow in or out of the neuron. In 1952, Alan Hodgkin and Andrew Huxley described a model [11] for the evolution of the neuronal current as well as the ion channels and received the Nobel prize in Physiology or Medecine in 1963.

Although spikes are inherently digital, the neural information is believed, by many researchers, to be encoded by the relative timing between them in the analog domain, and the complex chemical reactions involved in neural circuits are also fully analog and noisy. Analog CMOS was thus a good candidate to implement processing systems inspired by biology. Mead and his team successfully designed a silicon retina and a cochlea for processing visual and audio inputs [12, 13]. Remarkably, they managed to design an artificial neuron in silicon [14] closely emulating the behavior of a real cat neocortex neuron.

While these implementations are event-driven and energy efficient, building higher-level processing systems requires off-chip communication, which are orders of magnitude more costly in energy consumption. Another challenge when emulating biological functions in silicon is how to decide how information should be processed and which algorithm should be implemented to perform meaningful computation.

Finding the algorithms performed by real neural networks remains an open problem. By contrast, researchers have designed more formal non-spiking models for neuronal computation as well as algorithms to adjust the synaptic connections between neurons. These algorithms have been developed in the context of artificial neural networks. In the next section, we review how these highly conceptual models of neurons and neural networks led to the recent breakthroughs in pattern recognition.

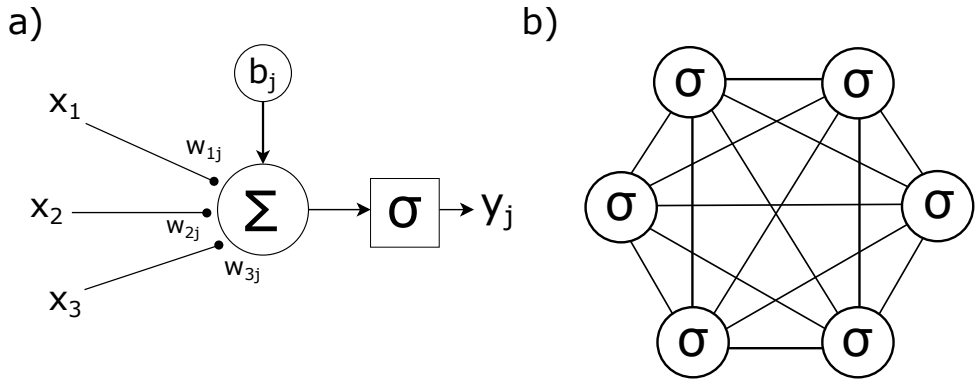

Figure 1.3: a) Schematic of the formal neuron or 'perceptron' [15]. b) Schematic of a Hopfield Network [16].

1.2 Artificial Neural Networks and Machine Learning

In this section, we review artificial neural networks from their inception in the 1950s to their recent breakthroughs for solving cognitive tasks. The framework of machine learning is introduced with a focus on neural network models. The mainstream pipeline of neural network training is detailed. We also describe the implementation of back-propagation for computing error gradients in modern deep learning frameworks.

1.2.1 Early Networks

1.2.1.1 The Perceptron

The first study of formal neurons dates back to 1943 with the work of Warren McCulloch and Walter Pitts [17]. They showed how a formal neuron receiving multiple inputs, summing them and producing an output by comparing the sum of inputs to a threshold could perform logical operations such as OR, AND, and NOT. However, their model did not include artificial synapses modulating incoming inputs. The formal neuron, or 'perceptron', introduced by Frank Rosenblatt [15] in 1958 included artificial synapses (see Fig. 1.3 a)). In this model, $x_1, x_2, ..., x_n$ represent the inputs of the neuron j as if coming from its dendrites, and $w_{1j}, w_{2j}, ..., w_{nj}$ account for the synaptic strengths, they are also referred to as 'synaptic weights' or simply 'weights'. The output of neuron j is given by the formula :

$$y_j = \sigma\left(\sum_{i=1}^{n} w_{ij}x_i + b_j\right) = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} w_{ij}x_i + b_j > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{1.1}$$

where b_j is the threshold of the neuron, or 'bias', and can also be viewed as a synaptic weight connecting an input clamped to 1. A negative synaptic weight can be viewed as a connection

with an inhibitory neuron.

This model, although simple, captures a main feature of a real neuron, which is integrating the inputs and producing a binary output. More importantly, Rosenblatt described a learning algorithm, that is a procedure to adjust the synaptic strengths in order to make the neuron perform a classification task. Taking a two-dimensional input space as an example, and noting x_1 and x_2 the coordinate of a point in the 2-D plane, the perceptron can perform binary classification, which means producing one of two outputs (0 or 1) for any given point. The boundary between the two possible outputs is given by the equation $\sum_{i=1}^{n} w_{ij}x_i + b_j = 0$, which is the equation of a straight line, or more generally a hyperplane in the input space. For this reason, the perceptron belongs to the class of linear classifiers. Thus, if we wish to classify two sets of points that can be separated by a hyperplane and have the perceptron output 0 for one set, and output 1 for the other set, the perceptron algorithm gives a way to adjust the synaptic strengths so as to make each set of points separated by a hyperplane.

More specifically, the learning rule for the weight w_{ij} is:

$$\Delta w_{ij} = \eta(\mathrm{t} - y_j)x_i, \tag{1.2}$$

where η is the 'learning rate', and t is the target associated with the input vector $\boldsymbol{x} = (x_1, ..., x_n)$. Provided that the data points are linearly separable, the perceptron algorithm is guaranteed to converge to a set of weights that effectively separate the data.

Therefore, the perceptron algorithm already contained powerful ideas such as learning under supervision from labeled data, with a theoretically-tractable training procedure inspired from the mechanisms of actual neurons. This algorithm was also implemented on a custom hardware called the 'Mark 1 perceptron' for image recognition [18]. It was made of an array of 400 photocells randomly connected to the neurons. Synaptic weights were encoded in potentiometers while weight updates during learning were carried out by electric motors.

However, the condition of linear separability of the data prevents the perceptron from classifying data produced by a non linear simple function such as XOR [19]. This limitation caused the field of AI to stagnate for about two decades.

1.2.1.2 Hopfield networks

Another type of artificial neural network, called Hopfield networks, was introduced by John Hopfield in 1982 [16]. This class of neural networks is also made of formal neurons similar to perceptrons (Eq. 1.1), but they are interconnected in an arbitrary fashion instead of a forward fashion (see Fig. 1.3 b)), and the dynamics is asynchronous, meaning that not all neurons are updated at every time step. Hopfield showed that these networks have the emerging property of content addressable memory. Given a set of specific patterns $\boldsymbol{\xi}^1, ..., \boldsymbol{\xi}^\mu$ for activation values,

made of ones and zeroes, Hopfield proposed to write the weights with the following formula:

$$w_{ij} = \sum_{p=1}^{\mu} (2\xi_i^p - 1)(2\xi_j^p - 1). \tag{1.3}$$

This rule is reminiscent of 'Hebbian' learning [20], because it only depends on neural activation directly adjacent to the synapse. This formula also implies that synapses are bidirectional, meaning that the synaptic weight of the synapse going from neuron i to neuron j is the same as the synaptic weight going from j to i: $w_{ij} = w_{ji}$. With this condition of symmetric weights, one can define the following energy function, analogous to the Ising model in physics:

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

where one step of dynamics (Eq. 1.1) leads to a decrease in the energy. The patterns, or 'memories' can thus be stored in the network as local minima of the energy function. If the neurons are set to a corrupted pattern $\widetilde{\boldsymbol{\xi}}$, the dynamics will lead the neurons to converge to the associated correct pattern. A network with N neurons can store $\approx 0.15N$ independent patterns [16]. Storing more patterns causes the local minima to merge into 'spurious patterns'. One striking feature about the Hopfield network is that the memory property emerges as a result of the high number of interacting neurons, and is quite independent from the specific details of each neuron. Indeed, Hopfield networks were subsequently adapted to neurons with graded outputs [21].

Modern Hopfield networks or 'Dense associative memories' [22] were introduced in 2016. They employ generalized energy functions that dramatically increase the capacity of the network to an exponential number of patterns as a function of the neuron number. While these new energy functions correspond to many-body interaction between neurons, the same properties can be recovered with two-body interaction with additional neurons [23]. These new Hopfield networks have been shown to have similar properties with the computation performed by the attention mechanism in transformers [24, 25], which are the current state-of-the-art models in natural language processing.

1.2.2 The Machine Learning Approach to AI

One of the reason that caused the study of artificial neural networks to be left aside during the 1970s was the lack of an algorithm to train neural networks with more than one layer. David Rumelhart, Geoffrey Hinton and Ronald Williams introduced the back-propagation algorithm in 1986 [26] to systematically compute the gradient of a difference measure between the output of the multi-layer network and a desired target. The idea of back-propagation is to apply the chain rule of differentiation to the composition of the neural networks layers. Rumelhart et al. showed that back-propagation could successfully train a multi-layer neural network, and

that doing so leads the hidden units to extract meaningful features of the input. Networks with hidden units, also called multi-layer perceptrons (MLP) were shown to be capable of approximating any function arbitrarily well in 1989 [27], as soon as they have one hidden layer and a sufficient number of hidden units. However, this theoretical guarantee is not satisfied with the number of neurons used in practice, and inductive biases need to be added in the architecture, as well as more layers. For image processing, a relevant inductive bias takes the form of convolutions [28], which are characterized by sparse connections and weight sharing so as to make the result invariant for a translated input. In 1989, LeCun et al. used back-propagation to train a convolutional neural neural network [29] on digit recognition.

Training an artificial neural network to perform digit recognition is an example of supervised learning application. The training process of an artificial neural network is the following: a dataset $\mathscr{D}_{\text{train}} = \{(\boldsymbol{x}_1, \text{t}_1), (\boldsymbol{x}_2, \text{t}_2), ..., (\boldsymbol{x}_n, \text{t}_n)\}$ of labelled data is provided where each $\boldsymbol{x}_i$ is associated with a label, the target t_i. We note $f : (\boldsymbol{\theta}, \boldsymbol{x}) \mapsto f(\boldsymbol{\theta}, \boldsymbol{x})$ the function implemented by the artificial neural network where $\boldsymbol{\theta}$ stands for the parameters of the neural network (synaptic weights and neuron biases) and $\boldsymbol{x}$ is a data point. At each training iteration, a small random subset of data points called a 'mini-batch' is sampled from $\mathscr{D}_{\text{train}}$ without replacement and an 'epoch' is completed when all the data points of $\mathscr{D}_{\text{train}}$ have been sampled. The function f is evaluated for all the data points of the mini-batch and the outputs y_i are compared against the targets t_i to compute a 'loss' denoted by $\mathscr{L}$. When the network has to predict a continuous target (regression task), the loss is usually the squared error, whereas when the target is a discrete label (classification task) the loss is the cross entropy [18]. The back-propagation algorithm is then used to compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}}$. The parameters are updated by taking a step in the opposite direction to the gradient in order to reduce the error measure. Because the error gradient is only estimated on the mini-batch and not on the whole dataset, this optimization procedure is called 'stochastic gradient descent'.

Other optimization procedures exist and consist in applying a function to the error gradient in order to accelerate or stabilize optimization. For instance, adding momentum consists in computing the parameter update as a linear combination of the gradient and the previous update to avoid oscillations in the parameter trajectory [26]. Adaptive moment estimation (Adam) [30] is another popular optimization algorithm for neural networks. The training process usually goes on for several epochs.

However, the goal of training a neural network is not only to perform well on training data, but also to generalize to *unseen* data. In order to verify that the network learns meaningful representations of the data, another dataset called the validation set and noted $\mathscr{D}_{\text{val}}$ is used to measure how the network performs on unseen data. When the neural network performs well on $\mathscr{D}_{\text{train}}$ but poorly on $\mathscr{D}_{\text{val}}$, it is said to 'overfit' to $\mathscr{D}_{\text{train}}$. Overfitting can be reduced by using a larger dataset. When obtaining more data is impossible, regularization techniques such as weight decay [18], dropout [31], or data augmentation can effectively reduce overfitting.

The training process also involves 'hyperparameters' such as the parameters of the optimization algorithm (learning rate, momentum...) or other regularization parameters. Hyperparameters are not optimized during a training loop but they are 'tuned' by the practitioner at the scale of the training process itself by comparing how they affect the performance on $\mathscr{D}_{\text{val}}$. This outer optimization loop can cause overfitting to $\mathscr{D}_{\text{val}}$. For this reason, a third set $\mathscr{D}_{\text{test}}$ should be used to evaluate the network before deploying it for real-world applications or deciding the winning model of a machine learning competition.

The training procedure described above was used at the 2012 edition of the ImageNet Large Scale Visual Recognition Challenge [32] (ILSVRC) when a convolutional neural network with 5 convolution layers followed by 3 fully connected layers won the competition by a large margin [33]. The ILSVRC consists in classifying real high definition images into 1,000 different classes. The training set contains 1.2 million images and is a subset of the ImageNet dataset which contains 14 million images belonging to 21,000 classes. The winning neural network, called AlexNet, achieved a top-5[1] test error rate of 15.4% while the runner up achieved 26.2% top-5 test error rate. To achieve such a feat, the authors used an efficient implementation of the convolution operation using GPUs. This milestone sparked the widespread interest in 'deep learning' [6] where many-layers neural networks trained in an end-to-end fashion on large datasets achieve state-of-the-art results.

The back-propagation algorithm for computing gradients of errors has been the workhorse of this revolution, and several software libraries (Tensorflow [34] from Google, PyTorch [35] from Facebook, and more recently Jax [36]) have been built to efficiently implement automatic differentiation. Figure 1.4 shows the main idea behind the implementation of automatic differentiation in PyTorch via the construction of a computational graph. In PyTorch, computation can be performed on data containers called 'tensors'. The graph of Fig. 1.4 takes the tensors $\boldsymbol{x}^T = (x_1, x_2)$ and $\boldsymbol{W}$ as inputs and computes

$$\mathscr{L} = \frac{1}{2}((y_1 - t_1)^2 + (y_2 - t_2)^2) \tag{1.5}$$

$$= \frac{1}{2}((\text{ReLU}(w_{11}x_1 + w_{12}x_1) - t_1)^2 + (\text{ReLU}(w_{21}x_1 + w_{22}x_2) - t_2)^2) \tag{1.6}$$

as an output, where ReLU is the rectified linear unit defined by $\text{ReLU}(x) = \max(x, 0)$. For each mathematical operation implemented in PyTorch, there is a corresponding backward operation (in blue in Fig. 1.4), which implements the partial derivative of the output with respect to the inputs, also called 'Jacobian'. These backward operations need the inputs to be cached during the forward computation. A computational graph in a typical training iteration ends with the computation of a single-valued tensor, such as the mean squared error (MSE) over the batch. When calling the 'backward' method on the tensor y, an initial gradient $\delta_0 = \frac{\partial y}{\partial y} = 1$ is

[1] Top-5 means that the correct label belongs to the 5 most likely labels predicted by the neural network.

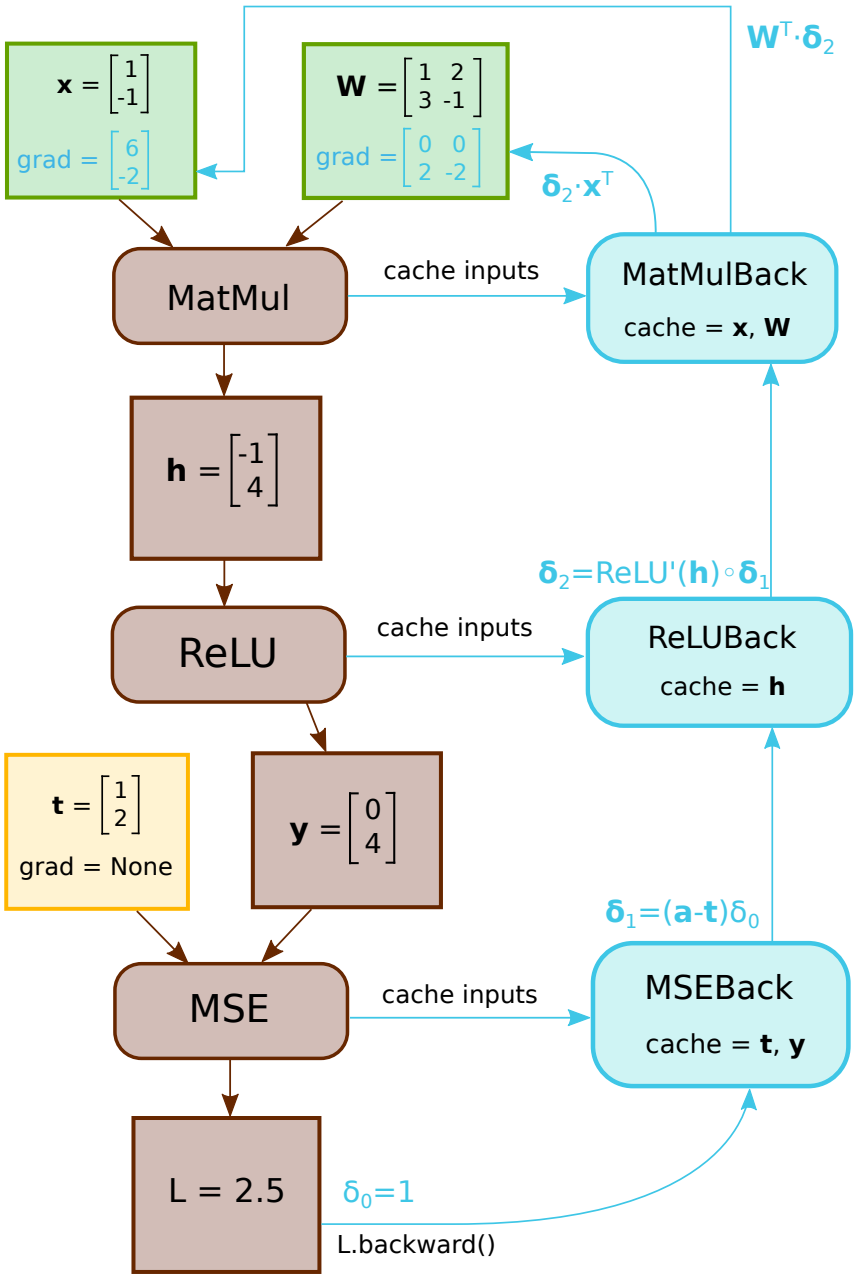

Figure 1.4: Toy example of how automatic differentiation is implemented in PyTorch. Adapted from a tutorial by Elliot Waite.

fed as an input to the backward graph. Gradients are then automatically accumulated in the 'grad' field of the leaf tensors of the graph and can be used to optimize the parameters. A direct computation shows for instance that, in the situation of Fig. 1.4:

$$\frac{\partial \mathscr{L}}{\partial w_{21}} = (y_2 - t_2) \times \mathrm{ReLU}'(h_2) \times x_1 = (4-2) \times 1 \times 1 = 2. \tag{1.7}$$

The backward method can in fact be called from any intermediary tensor by passing the right initial gradient vector. This is possible because each tensor contains a pointer to the backward version of the operation that produced it. However, gradients of intermediary tensors (brown rectangles in Fig. 1.4) are not stored by default. In more complicated graphs, tensors can be used through several paths, and, in this case, the backward graph accumulates gradients over all the paths.

Overall, automatic differentiation provides a way to compute derivatives of an arbitrary computational graph and gives researchers the possibility to explore the space of computational graphs to improve performance. In the next section, we review how deep neural architectures have evolved since 2012 to improve the state of the art.

1.2.3 Modern Deep Networks

After 2012 and AlexNet, new architectures have been designed to improve the classification accuracy on ImageNet, which remains the canonical benchmark for image recognition. The VGG architecture [37] builds on the fact that two consecutive convolutional layers with kernel size 3 have the same receptive field as one convolutional layer with a kernel size of 5, as illustrated in Fig. 1.5 a). By doing so, more layers can be stacked to achieve greater depth. The VGG architectures presented in [37] contained between 16 and 19 layers.

The inception architecture [38] introduced modules (or subnetworks) such as the one depicted in Fig. 1.5 c). The idea is to further factorize the 3×3 convolution into the composition of a 3×1 and a 1×3 convolution to keep the memory footprint constant while increasing the size of the network (Fig. 1.5 b)). The 1×1 convolution keeps the spatial dimensions constant but changes the number of channels. One instance of such inception architecture, named GoogLeNet, had 22 layers and won the ILSVRC14 challenge. 1×1 convolutions have also been used in other architectures to reduce the number of operations required by usual convolutions. The SqueezeNet [39] architecture uses a stack of specific modules where 1×1 convolutions reduce the channel width, before applying 3×3 and 1×1 convolutions to increase the channel width. This topology reduces the computation because costly convolutions are performed on a reduced channel width. When combined with deep compression [40], a compression pipeline for reducing the memory footprint of the network, SqueezeNet can perform the same accuracy as AlexNet on ImageNet while being 500× lighter, taking up only 0.5MB of memory [39]. MobileNet [41] architectures use a combination of depth-wise convolutions where each channel is convolved with one kernel of depth 1, before applying 1×1 convolutions, also called point-wise

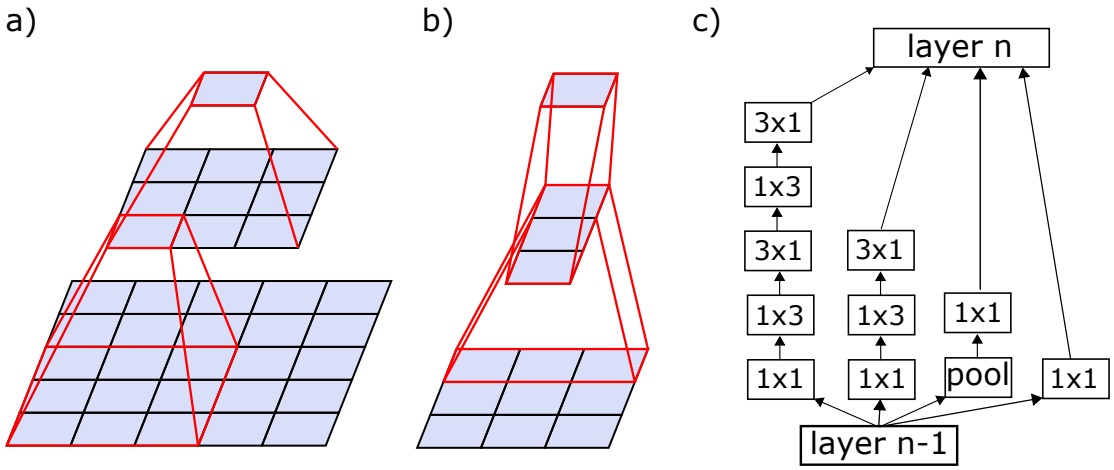

Figure 1.5: a) A 5×5 convolution has the same receptive field as two successive 3×3 convolution. b) A 3×3 conv can be simplified into a composition of 1×3 convolution and 3×1 convolution. c) An inception module [46], a 1×1 convolution only changes the channel dimension.

convolutions, to change the channel dimensions to the desired width.

Skip connections between layers of same dimensions were introduced in the ResNet architecture [42], with the idea to give the network the possibility to learn the residual transformation differing from the identity. Crucially, ResNets showed that deep architectures with skip connections outperformed the same architectures without skip connections. The benefit of skip connections is interpreted as providing an implicit ensemble of more shallow networks [43]. Recent state-of-the-art architectures include Efficient Nets [44, 45], which achieve increased performance by finding empirically how to properly scale the dimensions of the networks in terms of depth, width and resolution.

Deep networks have also revolutionized other data modalities such as speech recognition [47], games with reinforcement learning [48–51] and natural language processing with the long short term memory architecture [52], and more recently with the transformer architecture [53] based on attention [24]. A common pattern in deep learning research is the performance improvement when training bigger networks on bigger datasets. While AlexNet from 2012 had 16 million parameters and was trained on the 1.2 million images ILSVRC dataset, modern state-of-the-art networks for vision such as Efficient Net have 480 million parameters and involve around 30 million training images from the entire ImageNet dataset (14 million images) and 13 million images from the JFT-300M dataset. The current second biggest vision model has 829 million parameters and was trained on 3.5 billion Instagram images using the users hashtags as weak targets [54].

The scaling in the number of parameters is even more dramatic for natural language processing models for which parameters are counted in billions. One recent example is the Ope-

nAI GPT-3 language model [55] with 175 billion parameters trained on a dataset with 500 billion words, while the previous state of the art model GPT-2 from the year before had 'only' 1.5 billion parameters and was trained on 40 billion words [56]. A model like GPT-3 would have taken 355 years to train on a single V100 GPU and cost $4.6M in cloud computing time [2]. Perhaps surprisingly, the performance obtained by scaling up the dataset and number of parameters does not appear to plateau, which means that yet bigger models are likely to appear in the future.

However, upscaling will soon hit the barrier of unaffordable and unsustainable energy consumption. In their extensive study of more than 1,000 deep learning papers, Thompson et al. show that the relation between computation and performance when scaling up will not allow the top-1 test error rate of ImageNet, as well as other benchmarks, to be arbitrarily improved, given the current state of algorithms and hardware [8]. The scaling problem of deep learning thus calls for the co-design of new algorithms and hardware.

1.3 Major Differences Between Artificial Neural Networks and the Brain can Inspire Research in Deep Learning

Artificial neural networks were initially designed by taking inspiration from the brain and can now perform tasks similar to biological neural networks. However, some features of artificial neural networks are still very different from the brain, such as credit assignment by back-propagation and catastrophic forgetting, which we cover in this section. These biologically implausible features are obstacles for neuromorphic applications. Here, we review some features of the brain and the main approaches in the literature for solving the aforementioned limitations.

1.3.1 Does the Brain use Back-propagation?

While supercomputers achieve impressive results and even superhuman performance for specific cognitive tasks, the human brain can learn and perform many tasks with a far lower energy budget. The brain is composed of approximately 10^{15} synapses, 10^{11} neurons, and also non-neuronal cells called glial cells, which are responsible for many maintenance tasks such as recycling neurotransmitters, insulating neurons, providing them with energy, etc. Glial cells comes in many types, and are estimated to be as numerous as neurons [57], and are also suspected to play an active role in computation. The brain only consumes 20 W [58], which makes it extremely energy efficient, although making up 20% of the body power consumption when at rest. It is thus natural to study the brain in order to reproduce the key features responsible for its success [59]. However, the challenge is to tell apart the relevant features for information

[2] https://web.archive.org/web/20210413021003/https://lambdalabs.com/blog/demystifying-gpt-3/

processing from the irrelevant ones, which is all the more difficult as the mechanisms of the brain are far from being fully understood.

Perhaps the most striking differences between state-of-the-art feed forward deep networks and the brain are the use of spikes and dynamics. In fact, the real value of neural activation in deep networks can be thought of as a rate of spikes. The use of a real activation is convenient for automatic differentiation. Before the deep learning era, spiking neural networks where studied by neuroscientists with the sole goal of modeling the functions of biological neural networks. At that time, the brain was not believed to perform anything close to error back-propagation. After the success of rate-based deep networks at solving cognitive tasks with human-comparable performance [6], a new area of research has been dedicated to adapting spiking artificial neural networks for deep learning. The first works going in this direction aimed at translating deep neural networks into spiking networks by converting the activation into a constant spiking rates. By contrast, recent works have explored ways to train spiking networks where the latency between spikes is variable and computationally useful [60], similarly to the brain. These works adapt gradient-based techniques which have proven successful in training rate-based deep networks to spiking neural networks. They fall into two main categories [60]. In the *spike-timing based representation*, the actual real-valued spiking times of neurons are optimized by gradient descent. In the *activity-based representation*, the time step of the network is discrete, similar to recurrent neural networks, which makes the spiking times non differentiable. In this case, surrogate gradients are used to perform optimization [61, 62]. Interestingly, surrogate gradients methods can also optimize the parameters of the neurons such as the time constants and the connectivity [63], which could help to understand the different neuron types in the brain.

While gradient descent over an objective function successfully enables networks to perform complex tasks, the algorithm for computing gradients is equally important as far as brain understanding and hardware design are concerned. Some recent works in neuroscience suggest that deep networks trained with back-propagation account for the inferior temporal cortex representations found in real neural tissues better than models trained with other methods [64–66], which at least does not rule out the fact that the brain could do some sort of gradient based learning. However, it is not clear how the brain would perform this optimization. This main difficulty is known as the 'credit assignment' problem. Given a network with input units, output units and hidden units, how should hidden units be changed so as to drive the output units in the desired direction? This is a difficult problem, because each hidden unit influences the output in a very complex way. Back-propagation solves this problem, but in a highly non biologically-plausible way, for two main reasons:

- Firstly, we can see in Fig. 1.4 that an artificial neural network trained with back-propagation has to perform two types of computation: the forward pass which propagates neural ac-

tivation, and the backward pass which propagates error vectors. This is problematic because the δ quantities propagated during the backward pass are signed and potentially extreme-valued: either very small or very large. Furthermore, the computing graph of the backward pass does not fit the usual 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. It thus requires information about the forward pass to be stored.

- Secondly, back-propagation of error gradients needs to be performed with the transposed version of the forward synaptic weights (see 'MatMulBack' in Fig. 1.4). This issue is known as the 'weight transport' problem. For this reason, an efficient hardware implementation of back-propagation would require to use the same physical devices to encode the weights in both phases, but it shifts the problem to using the same circuit to perform the two different computations mentioned earlier [67, 68].

The counterpart of back-propagation for recurrent neural networks is back-propagation through time, which consists in unfolding the recurrent computational graph in time, hence creating 'copies' of neurons across time, and using back-propagation on the resulting forward graph. Back-propagation through time thus possesses the same biological implausibility.

Biologically plausible learning rules have been developed for recurrent spiking neural networks. Eligibility propagation [69] is a biologically plausible learning rule, which proved successful in approximating back-propagation through time for challenging tasks such as reinforcement learning of Atari games [48]. More recently, Payeur et al. [70] introduced 'burst propagation', a biologically plausible learning rule for deep spiking neural networks based on multiplexing simple spikes and burst of spikes to carry credit assignment. A coarse-grained ensemble version of burst propagation can achieve a top-5 test error rate of 56% on ImageNet, which nicely bridges the gap between neuroscience and machine learning neural networks.

1.3.2 Beyond back-propagation in Rate-based Neural Networks

We now describe learning strategies, which can achieve learning similarly to back-propagation, but using purely local computation. These theories, which use rate-base coding, may provide an alternative point of view to understand how the brain learns, and provide a path toward energy-efficient learning hardware.

1.3.2.1 Equilibrium Propagation

Equilibrium Propagation (EP), introduced in 2017 by Scellier and Bengio [71] provides a way to compute and propagate error gradients only with neural activities [71] in energy-based neural networks.

Before describing EP, it is useful to cover another training algorithm called 'contrastive Hebbian learning', introduced in 1985 to train Boltzmann machines [72], which are stochastic

counterparts to Hopfield networks. 'Hebbian learning' refers to the fundamental principle introduced by Hebb in 1949 [20], where repeated and persistent stimulation of the postsynaptic neuron from the presynaptic neuron leads to an increase in the synaptic strength. The contrastive Hebbian learning rule updates a synaptic weight according to the difference between the product of adjacent neural activation:

$$\Delta w_{ij} = \eta(\sigma_i^+ \sigma_j^+ - \sigma_i^- \sigma_j^-), \tag{1.8}$$

where the superscripts − and + respectively denote the activation in the negative and positive phases. In the negative phase, the network freely evolves according to its dynamics and some inputs, until it reaches an equilibrium. In the positive phase, the network also settles to an equilibrium but the output units are fully clamped to the targets. The update can then be separated into a 'negative update' increasing the energy of the pattern one wants to unlearn, and a 'positive update' decreasing the energy of the pattern one wants to learn, as illustrated in Fig. 1.6.

Contrastive Hebbian learning was adapted to deterministic networks [73–75] and became 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 contrastive Hebbian learning have been introduced to compute weight updates based on neural activations, such as Recirculation [76] and General Recirculation [77]. Xie et al. showed that back-propagation and contrastive Hebbian learning were equivalent in the case of feedback connections scaled by a small factor [78].

In 2015, Bengio and Fischer showed that when the output is slightly nudged in the second phase, the early change in neural activation correspond to the propagation of error derivatives. This idea was then taken further by Scellier and Bengio, when they introduced Equilibrium Propagation [71]. In Equilibrium Propagation, the free phase is the same as in contrastive Hebbian learning: the input x is clamped and the neurons evolve toward a low energy configuration that 'explains the data' given the synaptic weights of the network. By noting $\boldsymbol{u} = \{\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{y}\}$ the set of all units and $\boldsymbol{s} = \{\boldsymbol{h}, \boldsymbol{y}\}$ the set of neural units without the clamped input $\boldsymbol{x}$, the continuous Hopfield energy is defined over all units and is given by

$$E = \frac{1}{2}\sum_i u_i^2 - \frac{1}{2}\sum_{i \neq j} w_{ij}\sigma(u_i)\sigma(u_j) - \sum_i b_i \sigma(u_i), \tag{1.9}$$

and the neural dynamics follow the leaky integrator formula:

$$\frac{\mathrm{d}s_i}{\mathrm{d}t} = -\frac{\partial E}{\partial s_i} = -s_i + \sigma'(s_i)(\sum_{j \neq i} w_{ij}\sigma(u_j) + b_i). \tag{1.10}$$

The equilibrium state reached by this process is noted $\boldsymbol{s}_\star^0$.

The second phase differs from constrastive Hebbian learning because the outputs are only

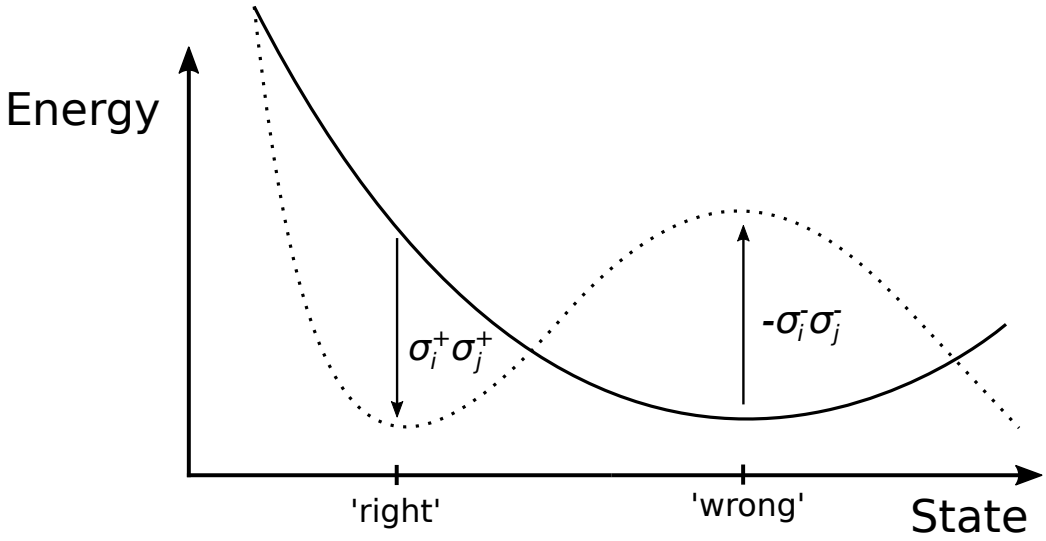

Figure 1.6: Effect of contrastive Hebbian learning on the energy landscape of the model. In the free phase, the model relaxes to an energy minimum that corresponds to a wrong prediction. In the clamped phase, the output units are clamped to the right target values, causing the model to relax in an new energy minimum corresponding to the desired behavior. The constrative Hebbian update can be viewed as updating the weights to increase the energy of the wrong neural configuration while decreasing the energy of the correct configuration.

weakly clamped to the target, similarly to a spring nudging the output $\boldsymbol{y}$ toward the target t (see Fig. 1.7 a)). This is done by adding a quadratic cost C scaled by a small 'nudging' factor β to the Hopfield energy function E, yielding a total energy F:

$$C = \|\boldsymbol{y} - \mathrm{t}\|_2^2,$$
$$F = E + \beta C.$$

The perturbation implicitly propagates from the output layer to the hidden layers by following the same dynamics. The second equilibrium in EP provably remains in the same energy mode thanks to the implicit function theorem, while in contrastive Hebbian learning, the second equilibrium can be in another mode, which leads to unstable training [74]. The second equilibrium is noted $\boldsymbol{s}_\star^\beta$. Given the two equilibrium states, the parameters are updated according to:

$$\Delta w_{ij} = \eta \frac{s_{\star,i}^\beta s_{\star,j}^\beta - s_{\star,i}^0 s_{\star,j}^0}{\beta}, \tag{1.11}$$

$$\Delta b_i = \eta \frac{s_{\star,i}^\beta - s_{\star,i}^0}{\beta}. \tag{1.12}$$

Scellier and Bengio show that this update performs gradient descent over the cost C when $\beta \to$

0:

$$\left.\frac{d}{d\beta}\right|_{\beta=0}\left(s^{\beta}_{\star,i}s^{\beta}_{\star,j}\right)=-\frac{\partial C}{\partial w_{ij}}, \tag{1.13}$$

and achieve 0.00% training error and test error between 2% and 3% [71] on the MNIST dataset [79]. Moreover, Ernoult et al. [80] showed that the neurons dynamics updates in the second phase are step-by-step equivalent to the cost gradients with respect to neurons that can be computed by back-propagation through time.

EP thus provides a solution for propagating error gradients with one type of computation (the neural dynamics), while also having theoretical guarantees regarding optimization. It only requires one type of neural computation, and the weight update can be expressed with local neural variables only. Owing to these strong features, EP is especially promising for designing physical systems which perform computation out of their own dynamics.

1.3.2.2 The Issue of Weight Transport

The issue of weight transport refers to the need for the exact transpose of forward weights in order to back-propagate the error gradients (see 'MatMulBack' in Fig. 1.4). In an architecture with weight transport, the weights are said to be 'symmetric'. EP also requires the synaptic strengths to be symmetric (see Fig. 1.7 a)) in order to be framed as an energy-based model. Scellier et al. have proposed a generalized version of EP called 'vector field' for which the symmetric weight condition is relaxed [81]. This generalization of EP can train MLPs on MNIST, however the theoretical guarantee to optimize the cost function does not hold in this case.

Recent works have shown that the symmetric weight condition for back-propagation can be relaxed to some extent. Lillicrap et al. showed that using fixed random backward weights instead of the weights transposed for credit assignment enables training a multi layer architecture on MNIST [82]. Feedback weights can also be set to link the output layer to each hidden layer directly [83] without degrading the performance on MNIST. However, these approaches do not appear to scale to more complex tasks [84].

Several approaches relax the constraint of symmetric weights while maintaining some similarity between forward and backward weights. Liao et al. show that having the same sign between forward and backward connections, but not the same magnitude, is enough for approximating back-propagation [85], but they require normalization mechanisms such as batch-normalization [86]. This sign symmetry between forward and backward weights was shown to match back-propagation performance on ImageNet [87, 88]. Akrout et al. [89] introduced the algorithms 'weight mirrors' and 'Kolen-Pollack' for solving the weight transport problem based on the fact that the covariance between the output and input of a given layer involves the transpose of the weight matrix. Specifically, if $\boldsymbol{y}=\boldsymbol{W}\boldsymbol{x}$ and the activation function is omitted, $E\left[\boldsymbol{x}\boldsymbol{y}^T\right]=E\left[\boldsymbol{x}\boldsymbol{x}^T\right]\boldsymbol{W}^T=\sigma^2\boldsymbol{W}^T$ with the last equality holding if the input are independent and identically distributed with variance σ^2. They thus design a local learning rule for feedback connections that make them become approximately proportional to the transposed forward

connections throughout learning and thereby closely match back-propagation. Their algorithms both outperform the sign symmetry algorithm [87, 88] on ResNets architecture trained on ImageNet. These approaches, although providing solutions for the problem of weight transport, still require signed errors to be linearly propagated by the feedback connections.

1.3.2.3 Target propagation and variants

Target propagation [90] is another biologically plausible algorithm, which updates the weights based on local neural activation, and does not require symmetry between forward and backward connections (see Fig. 1.7 b)). Although those features are desirable, Target propagation crucially relies on being able to invert the operation performed by one layer, which can be difficult when the dimension changes from one layer to the next. The idea of Target propagation is the following: if one could have inverse operations capable of inverting each layer of the neural networks, one could compute the inverse of the target at the output layer and also subsequent inverses until the first layer. If the network had obtained these inverted activation values in the forward pass, then it would have obtained the right output. If we have such target activations, we can update the forward weights to make the layer's output closer to its target activation. The learning procedure of target propagation is to learn the inverse functions by introducing trainable feedback weights. In the initial formulation of target propagation [90], the target of the penultimate layer is obtained by back-propagation of the global loss while subsequent targets are obtained by applying the approximated inverses to the already computed targets. Once all the targets are computed, the forward and backward parameters are updated by optimizing layer-wise, local, respectively forward and backward losses.

However, this formulation of target propagation does not perform well because the imperfection in the approximated inverse operations generate poor targets. This issue was mitigated by Lee et al. [91] who introduced 'difference target propagation'. They add a linear correction term to the targets to make up for the imperfect inversion, depicted as the orange arrows in Fig. 1.7 b), and obtain a more accurate target activation $\widetilde{\boldsymbol{h}}$ (see Fig. 1.7 b)). With this correction term, difference target propagation can successfully train MLPs on MNIST and closely match back-propagation. In terms of theoretical guarantees to optimize the global loss, Lee et al. [91] show that provided that the inverse operation are perfect, the angle between the updates provided by back-propagation and difference target propagation do not exceed 90°, which ensures that the global loss decreases. However, the issue of imperfect inverses is particularly pronounced in the last layer for classification problems with a low number of classes such as 10 for MNIST [79] and CIFAR-10 [93].

Bartunov et al. compared biologically plausible algorithms based on feedback alignment and target propagation [84] and introduced a simplified version of difference target propagation where backward weights are used even for the last layer such as in Fig. 1.7 b). They show that doing so decreases the performance of difference target propagation. Overall, Bartunov et al. show that feedback alignment and difference target propagation closely match back-

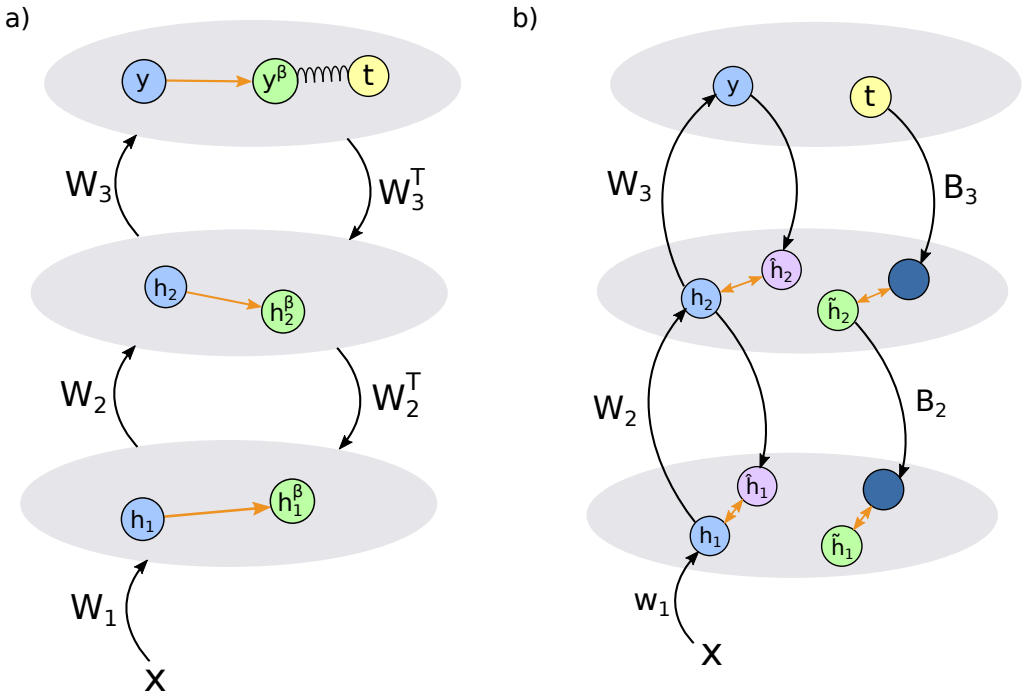

Figure 1.7: Grey ellipsoids denote the spaces of neuron layers, and circles denote specific vector values for each layer. Arrows denote synaptic connections. a) Description of Equilibrium Propagation [71]. In the first phase, neurons evolve by following the dynamics given by the gradient of the energy function, and settle to an equilibrium in blue. In the second phase, the target t is revealed and pulls the output unit towards itself, causing all the units to reach a second equilibrium. The weight update is computed based on the two equilibria. The input $\boldsymbol{x}$ is clamped throughout both phases. b) Difference Target Propagation [91], adapted from [92]. In the first phase, activation are propagated (blue) and approximate inverses are reconstructed (purple) by the backward weights. In the second phase, targets are generated by using the backward weights, and a correction term obtained obtained during the first phase, which corrects the imperfect inversion. Forward weights are updated to produce targets, and backward weights are updated to better approximate the inverse operation.

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. However, they show that these algorithms do not match back-propagation on ImageNet.

Despite the biological implausibility of back-propagation, it remains the best algorithm for training artificial neural networks on challenging real-world tasks. While the algorithms presented in this section provide ways to estimate error gradients in function of neural activation from simple artificial neurons, another line of research based on predictive coding [94–96] explore more complex neurons with compartments dedicated to encoding errors. These more complex architectures also possess theoretical guarantees with respect the computed gradients [97], but are less straightforward to implement in hardware.

Overall, a biologically plausible and/or hardware-friendly algorithm capable of matching back-propagation on real-world tasks has yet to be designed, but we argue that Equilibrium Propagation is the most promising of algorithm to reach this goal.

1.3.3 Memory and Forgetting in Artificial Neural Networks

Another biologically problematic feature of artificial neural networks is the interference between new and old representations when trained on a new task, which is very unlike the brain. This issue has been reported early on as 'catastrophic forgetting' or 'catastrophic inference' [98] and affects both content-addressable memories such as Hopfield networks [99–101] and deep networks [102–104]. The reason for catastrophic forgetting is sometimes referred to as the 'plasticity-stability dilemma' [105]: the network needs to update its synaptic connections in order to learn a task, but the connections should be stable when learning a second task, otherwise the connections will be erased by the new task. In the next subsections, we review how this common issue has been addressed in the literature for both Hopfield networks in the online learning regime and deep networks.

1.3.3.1 Forgetting in Hopfield networks

Hopfield networks have been introduced to store a number of pattern activation in the synaptic connections of a network. A pattern can be retrieved by the neural dynamics when a corrupted version is presented to the network. Although the metric of interest is initially the storage capacity, another concern is the biological plausibility of the writing rule 1.3:

$$w_{ij} = \sum_{p=1}^{\mu} (2\xi_i^p - 1)(2\xi_j^p - 1),$$

because the higher the number of patterns μ stored in the network, the higher the magnitude of the weights. Normalizing the weights shifts the problem to having arbitrary precision to describe the synapse, which is biologically implausible. This limitation motivated a line of re-

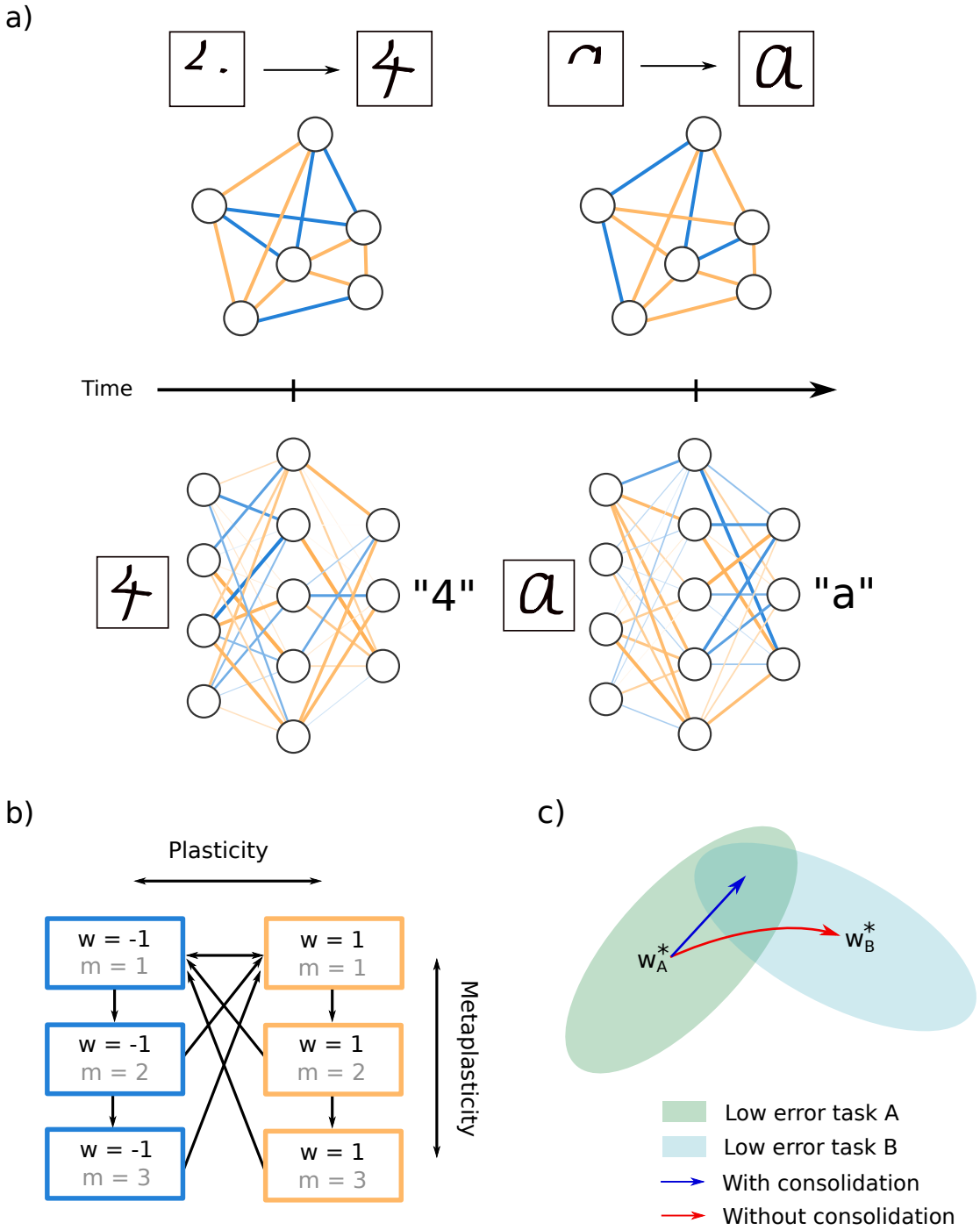

Figure 1.8: a) Continual learning in Hopfield network (top) and multilayer perceptron (bottom, generated with the software NN-SVG [106]). b) Metaplastic transitions introduced in [107] to alleviate forgetting in Hopfield networks. c) Regularization based methods for alleviating forgetting in deep networks pioneered by [104]

search studying the online learning setting of Hopfield network where patterns are presented sequentially and synapses are binarized as depicted at the top of Fig. 1.8 a) with binarized synapses in blue and orange. Each synapse has some probability to be written when a pattern is presented. The two metrics of interest in this setting are the time during which a pattern can be remembered, and the strength of the initial memory signal. Amit et al. [108] showed that if the number of states per synapse is bounded, then the time under which a pattern can be retrieved, or the number of retrievable patterns at a given time, is as low as $\log(N)$, where N is the number of neurons. This scaling property is very implausible with respect to the brain. Given the fact that the brain has 10^{11} neurons and biological synapses must have finite precision, it would mean that only tens of patterns can be stored and that forgetting is exponentially fast [107].

However, biological synapses are more complex than a mere scalar value and involve many molecular processes [109, 110]. Biological synapses have been found to be more complex in vertebrates than in invertebrates [111], which hints at the fact that synapse complexity is beneficial in large biological neural networks. Experiments on human subjects show that forgetting in the brain follows a power-law decay instead of an exponential one [112, 113]. Biological synapses are not only plastic, but also metaplastic [114, 115], which means that the plasticity of the synapse itself can change according to complex molecular machinery.

This concept was successfully adapted to Hopfield networks by the pioneering work of Fusi et al. [107]. The metaplastic synapse introduced in [107] is reproduced in Fig. 1.8 b). In this 'cascade' model, synaptic weights can have two states, +1 or -1, used for running the attractor dynamics of the network, but they are also equipped with a hidden metaplastic state denoted by m. The probability transitions between different (w, m) states depend on m in an exponential fashion. Higher m states correspond to consolidated synapses that are less likely to switch to the opposite w value. With this synaptic model, Fusi et al. showed that the initial memory signal and the memory lifetime scale as $\sqrt{N_{\text{syn}}}$ with N_{syn} the number of synapses in the network. In subsequent work building upon [107], Benna et al. [116] showed that a slightly more complex synapse model involving more hidden variables interacting in a bidirectional diffusion process can bring the memory lifetime to scale almost linearly with the number of synapses while maintaining the same scaling for the initial memory signal. In addition to metaplasticity, their model can also account for other phenomena found in real synapses such as delayed expression of long term potentiation and depression, and spacing effects for knowledge consolidation (optimal rehearsal interval).

Catastrophic forgetting is also an issue for deep neural networks introduced in section 1.2.3. Although connections with computational neuroscience models have been made to address catastrophic forgetting in deep networks [59], the mechanisms are quite different in practice.

1.3.3.2 Forgetting in deep networks

Catastrophic forgetting in deep networks is a major issue with consequences on model deployment to production. When a deep network has been trained on a dataset to perform a task, and we wish to use the same network to learn new data without keeping the initial dataset, the typical behavior is that the network forgets its knowledge on the initial task it was trained on. This is harmful in a variety of settings. Catastrophic forgetting prevents the network from learning several tasks in a row, but it can also be undesirable for transfer learning. Transfer learning consists in training the network on a large dataset to obtain a strong and general feature extractor before fine-tuning the network on a smaller dataset, on which the feature extractor could not have been obtained. However, catastrophic forgetting will cause the network to lose its performance on the initial dataset, even if the new dataset shares similarities with the first one [117, 118].

The continual learning literature aims at alleviating catastrophic forgetting in deep networks. Several reviews about this recent sub-field have already been written [119–121]. The complexity of continual learning lies in the wide variety of subtly different problem settings, several groups of very different approaches to achieving continual learning, and the variety of metrics to measure the performance of the models [122]. In this section, we give a general introduction to these aspects of continual learning.

Training a deep network to perform a classification task involves a training set as explained in section 1.2. Naturally, performing continual learning of successive classification tasks involve a sequence of training sets $(\mathscr{D}_1, ..., \mathscr{D}_n)$. The setting is depicted in Fig. 1.8 a) below the time axis, the color and intensity of the synaptic connections account for their values. The general rule of continual learning is that when the model is learning task i, it cannot access the datasets $\mathscr{D}_j$ for $j < i$. However, the validation accuracy over previous datasets should be maintained or even improve (a phenomenon called backward transfer). When the model has learned the whole sequence, it can be used to infer data on any dataset of the sequence.

Van de Ven et al. [121] point out a subtlety when testing the continually learned model: does it need the task index to infer the data at test time? If the model needs to be told somehow the task index, for example if it needs to use task specific parameters, then it belongs to the 'task incremental learning'. This setting is the easiest one because the model did not learn all the task in absolute, but relative to each other. The 'domain incremental learning', according to [121], corresponds to the case where the model does not need to know the task index to give the right output, but still cannot tell the task which the input belongs to. This setting is more difficult than the first one, but has limitation if common output units are shared between the tasks. The third and most difficult setting is the 'class incremental setting' and corresponds to the case where the model can give the right answer and tell the task from which the input comes from. If this third setting is solved, then learning the classes of a classification problem sequentially becomes possible. Other settings of continual learning include for example the absence of task boundaries where the model does not know when a task ends and a new task begins, or the

distribution of the training data being not independent and identically distributed as it is usually the case in machine learning. In the next paragraphs, we only review the main approaches to alleviate catastrophic forgetting in deep networks and refer the reader to existing reviews for further details [119–121].

Regularization-based methods. This type of continual learning algorithms aims at finding which synaptic weights are important for previous tasks, and adding a regularization term to the loss function in order to penalize the change of important parameters when subsequent tasks are learned. Suppose task A and task B are to be learned sequentially, then the loss optimized for task B is:

$$\widetilde{\mathscr{L}}_B(\boldsymbol{\theta}) = \mathscr{L}_B(\boldsymbol{\theta}) + c\sum_i \lambda_i \left(\theta_i - \theta^\star_{i,A}\right)^2, \tag{1.14}$$

where $\theta^\star_{i,A}$ are the parameters obtained after learning task A, and c is a hyperparameter trading off between rigidity and plasticity. The principle is depicted in Fig. 1.8 c). Although this class of method is often inspired by computational neuroscience models [107] (Fig. 1.8 b)), their concrete implementations are at odds with the principles of online learning and locality (see sub-section 1.3.3.1) that motivated [107].

The first work to have proposed such a method is Kirkpatrick et al. [104] with 'elastic weight consolidation'. The theoretical motivation for elastic weight consolidation is Bayesian. The idea is to use the posterior distribution of parameters after learning task A as a prior for learning task B. Because the posterior distribution over the parameters is intractable, it is approximated by a Gaussian distribution centered in the optimal parameters for Task A and with a diagonal precision matrix whose elements are determined by the Fisher information matrix. The negative log likelihood of the data for task B then takes the form of Eq. 1.14. Near the optimum for task A, the Fisher information matrix is equivalent to the second order derivative of the loss [123], and is in practice approximated by the squared parameter gradients averaged over the data:

$$\lambda_i = \mathbb{E}_{\mathscr{D}_A}\left[\left(\frac{\partial \mathscr{L}_A}{\partial \theta_i}\right)^2\right]. \tag{1.15}$$

λ_i is an approximation of the loss landscape curvature and parameters corresponding to high curvature dimensions must stay the same while parameters corresponding to flat dimensions can be used for learning subsequent tasks. Kirkpatrick et al. show that their algorithm enables learning sequentially pixel-permuted versions of the MNIST dataset, a benchmark introduced by Goodfellow et al. [117], as well as Atari Games in a reinforcement learning context [49]. When more tasks are learned, more terms are added to the loss, or it can be made online in order to keep the number of terms bounded [124]. A limitation is that a separate phase is needed between tasks in order to compute the parameters importance factors λ_i.

This restriction is relaxed by 'Synaptic intelligence' introduced by Zenke, Poole et al. [125]. Synaptic intelligence computes its importance factor in an online fashion during learning Task A. The theoretical insight for synaptic intelligence is the following: ideally one would like the

final model to be optimized over the total loss $\mathscr{L}_A + \mathscr{L}_B$. However, one cannot compute $\mathscr{L}_A$ during task B because the data is not available. $\mathscr{L}_A$ is thus replaced by a surrogate quadratic loss that will capture important aspects about the original loss: the optimum is the same and the net loss increase given the total parameter change is the same. This yields the following formula for λ_i:

$$\lambda_i = \frac{\omega_{i,A}}{\left(\theta^\star_{i,A} - \theta^0_i\right)^2 + \epsilon}, \tag{1.16}$$

where $\omega_{i,A}$ is the loss variation induced by the trajectory of the parameter θ_i computed during the training of task A. θ^0_i is the parameter before training task A and ϵ a damping factor to avoid division by zero. We this definition of λ_i, we see that if $\boldsymbol{\theta}$ was to go back to its initialization value while learning task B, it would increase accordingly the loss of task A. When learning more tasks, λ_i becomes the sum of several terms related to each task. Interestingly, on a toy problem with a quadratic loss landscape, they show that ω_i is linked to the curvature of the loss, but it is not guaranteed in general. They show that synaptic intelligence performs similarly to elastic weight consolidation on the permuted MNIST benchmark and they also investigate learning digit classes two by two with one binary output layer for each task. They do a similar experiment on CIFAR-10 and CIFAR-100 [93]. This setting is sometimes referred to as 'multi-head' setting and requires knowing the task label to read the classification from the right output head at test time. Finally, although the computation of $\boldsymbol{\omega}$ is carried online, task boundaries are still need to update $\boldsymbol{\lambda}$.

Aljundi et al. [126] propose another way of computing the importance of the parameters called 'Memory aware synapse'. If we call f the function learned by the neural network which outputs the last layer as a vector, a parameter is important for previous tasks if the squared norm of f has a high gradient with respect to it:

$$\lambda_i = \mathbb{E}_{\mathscr{D}}\left[\frac{\partial \|f(\boldsymbol{x},\boldsymbol{\theta})\|_2^2}{\partial \theta_i}\right], \tag{1.17}$$

where $\mathscr{D}$ can be a held out dataset without labels, contrary to the two previous approaches. They test this method over a wide range of complex vision tasks, but still in the multi-head setting with task oracle at test time.

Methods using (pseudo) data rehearsal. Another family of continual learning methods use data to alleviate catastrophic forgetting instead of consolidating parameters. However, the data used to rehearse while learning a new task is not necessary coming from previous training datasets. Li and Hoiem have introduced 'Learning without forgetting' [118]. They consider the setting where a convolutional neural network is used to learn several tasks sequentially. All the layers except the last one are common to all tasks, but a new linear classifier is introduced for each task. Before learning a new task, they use the new training dataset to generate soft targets (given by the predictions) with the classifiers of old tasks. When learning the new task,

convolutional layers and the new classifier are optimized for the current task, but convolutional layers and old classifiers are also optimized to *preserve* the predictions made before starting the new task. It is interesting that this technique works at all, because the old classifiers were not trained on the current dataset. Although Learning without forgetting is sometimes categorized as a regularization based method (see e.g. in [119]), the added loss depends on the generated data and not only on the parameters. The authors run experiments with AlexNet and VGG architectures on real world-visual tasks and report only slight degradation of performance with respect to joint-training of all tasks at the same time. However, it has been shown that Learning without forgetting is not robust to sequences of tasks with very different data distributions.

Rebuffi et al. have introduced 'Incremental Classifier and Representation Learning' (iCaRL) [127], which is specifically designed for class incremental learning with a minimum of two classes by tasks. The authors use a convolutional architecture which is trained incrementally. After learning a task, they store 'exemplars' based on the following principle: they compute the mean representation of training examples in the feature space (last convolutional layer), and choose a given amount of examples closest to the mean (for example 20). This amount of exemplars can be reduced in the future by popping the exemplars from the end of the list in order to meet a constant memory budget. When training a subsequent task, they merge the current data with the previous exemplars and their task label. The loss optimized is the sum of the cross entropy loss over the current task and a distillation loss. iCarl outperforms learning without forgetting on the incremental CIFAR-100 task.

Other approaches aim at training a generative model in order to generate previous tasks data and merge it to currently available data. This approach was introduced by Shin et al. [128] with 'Deep generative replay'. Their model consists of two neural networks: a generator and a solver. When a new task is learned, the training process requires the current and the previous generator-solver pair in order to produce the next generator-solver pair. Their method is shown to work well on the MNIST, Street View House Number (SVHN) task sequence. However, training a generator can become prohibitively complex for more difficult visual tasks. That is why Kemker et al. design their generative method FearNet [129] on embeddings obtained from a pre-trained ConvNet rather than raw inputs. They show better performance than iCaRL for the CIFAR-100 class incremental setting. Using one invertible neural network for each new class showed competitive performance with iCaRL and FearNet [130].

Interestingly, replay-based systems have a biological justification [131, 132]. In the brain, the hyppocampus is responsible for storing recent memories which are later consolidated in the neo cortex during rapid eye movement (REM) sleep. This is the complementary learning systems theory. Overall, current state-of-the-art methods rely on such complementary systems [121]. However, it should be noted that those methods can sometimes be combined with regularization based methods to improve performance.

1.4 Current Dedicated Hardware for AI

In the final section of this Chapter, we review the existing architectures for hardware dedicated to inference and learning. It will also appear that the bio-inspired algorithms presented in the previous section provide useful guiding principles for the design of energy efficient hardware using emergent technologies. When it comes to hardware dedicated to neural networks, it is important to distinguish between neuroscience-based hardware which aims at mimicking real neurons and synapses to study the similarity with the brain, and top-down approaches where the architecture design is driven by the physical implementation of an existing artificial intelligence algorithm [133].

1.4.1 Neuroscience-based Hardware

A wide range of neuromorphic hardware consists in simulating networks of complex neurons similar to biological neurons. Although the Hodgkin & Huxley model [11] mentioned in section 1.1 accurately describes biological neurons, simpler models like the Izhikevitch [134] or adaptive-exponential [135] models can still describe the complexity of biological neurons and are often chosen for the design of neuromorphic hardware. Another simple neuron model found in neuromorphic hardware is the Leaky-Integrate and Fire model (LIF). As far as synapses are concerned, different designs allow more or fewer bits to encode the synaptic weights and do not necessarily allow plasticity. When synaptic plasticity is implemented, it often takes the form of short term plasticity or long term plasticity rules such as spike-timing dependent plasticity [136] or spike-driven synaptic plasticity [137], which are inspired by experimental observation of specific types of synaptic plasticity in the brain. Another important distinction between approaches is the use of digital or mixed-signal analog-digital for the circuit architecture. In the following, we review only the main designs for this kind of neuromorphic hardware and refer the reader to [133] for a more complete review.

Mixed-signal approaches Among approaches using analog CMOS, a further distinction has to be made depending on the operating regime of the CMOS transistors. In the *sub-threshold* regime, the MOS transistors channels let the electrons flow in a diffusion process which is reminiscent of the ion channels of real neurons. Several approaches adopt this architecture principle. The Neurogrid chip [138] has 16 cores of 64k neurons with off-chip synapses. The ROLLS chip [139] employs a 180 nm process and contains a core with 256 adaptive-exponential neurons and 128k synapses implemented with capacitors. Half of synapses can implement short term plasticity and the other half implements spike-driven synaptic plasticity [137]. The ROLLS chip was up-scaled in the DYNAPs chip [140], although without plasticity.

Other analog approaches using the CMOS in the *above-threshold* regime can accelerate neuroscience simulations. The BrainScaleS wafer [141] consists of HICANN chips with 512 adaptive-exponential neurons and 112k 4-bit spike-timing dependent plastic synapses. The

wafer has 352 such chips and thus adds up to 180k neurons and 40 million synapses.

Digital approaches This line of neuromorphic hardware aims at simulating spiking neural networks more efficiently than conventional von-Neumann processing units. The SpiNNaker node developed by the university of Manchester [142] follows a distributed von-Neumann architecture with 18 ARM968 cores able to simulate few hundreds of LIF or Izhikevitch neurons in a time-multiplexed fashion, with up to 1,000 synaptic connections. The first version of SpiNNaker uses a 130 nm process. Nodes can be combined to simulate up to one billion neurons in the SpiNNaker project [143]. The communication between the micro-processors is asynchronous and relies on the transmission of data packets to simulate spikes.

Other approaches rely on full-custom designs. IBM developed the TrueNorth chip [144] using a 28 nm process, which consists of 4,096 cores each simulating 256 linear-leak integrate and fire neurons, a simplified version of the LIF neurons, for a total of approximately 1 million neurons. The synapses are binarized, non-plastic, and encoded in volatile SRAM.

On the other hand, Intel developed the Loihi chip [145] using a 14 nm process, consisting of 128 cores capable of simulating 1,024 LIF neurons with synapses encoded with SRAM of 1 up to 9 bits and capable of a programmable spike-based plasticity rule. The number of synapses per core ranges from 1 million to 114k depending on the encoding.

Because they are designed from neuroscience models, and credit assignment in the brain remains an open question, most chips can only learn simple classification tasks of small patterns or MNIST digits, using spike timing dependent plasticity or similar biologically derived learning rules. On a single layer architecture however, surrogate gradient based transfer learning can be implemented and solve more challenging tasks such as gesture recognition, as shown by [146] with the Loihi chip. They are also of great interest to run accelerated experiments on spiking neural networks. However, challenging AI benchmarks described in section 1.2 remain currently out of reach for this kind of chips.

1.4.2 Deep learning-based Hardware

This sub-section reviews dedicated hardware whose designs are based on existing deep learning algorithms and aim at the reducing the energy consumption of GPUs and CPUs. Those approaches are thus mostly based on non-spiking neural networks.

1.4.2.1 Dedicated hardware for inference

According to NVIDIA, 80% to 90% of the total energy cost in the lifetime of a neural network is taken by the inference phase when the model is deployed in production (as of 2019)[3]. It is thus important to design power efficient solution for inference. Although GPUs are more adapted

[3] http://web.archive.org/web/20210225104156/https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference/

to AI than CPUs, they are still general purpose in terms of the parallelizable tasks they can perform. To meet the computing needs of its new voice recognition service, Google designed its own digital Application Specific Integrated Circuit (ASIC) called the Tensor Processing Unit or TPU [147]. The solution provided by the TPU is to perform the tensor multiplication operations underpinning deep learning in a more efficient way using systolic arrays. A systolic array is a network of tightly coupled data processing units each independently computing a partial result from upstream inputs and passing it to downstream units. The data flow in a systolic array has a wave-like propagation, which is the reason for the medical term 'systolic'. The first generation of TPU introduced in 2016 used a 28 nm CMOS process and consisted in a 8-bit matrix multiplication engine using a systolic array, limiting the memory accesses of the traditional von-Neumann architecture. The TPU v1 could perform 23 TOPS for 28-40 W. However, it was only designed for inference. Learning in half precision (16-bit floating point) was enabled starting from the second generation of TPUs. More recently, Google introduced the Edge TPU, a system on chip (SoC) capable of 4 TOPS with only 2 W. (By comparison, IBM has reported a power consumption of 70 mW for the TrueNorth chip described in the previous section.) The Edge TPU only supports 8-bit precision, so it must either be used for inference or for basic training of a readout layer using the recently introduced Tensorflow quantization-aware training library. Overall, the impact of TPUs in terms of energy consumption has been to save Google from building a dozen more data centers[4].

Another path for low-power inference is using microcontrollers (MCU), which are purpose-specific computers with tiny memory budgets. Recent works have shown that high-end MCUs, can run compressed networks and achieve more than 70% top-1 accuracy on ImageNet [148], for a power consumption of the order of the watt and thus one order of magnitude better than an edge TPU. The MCU used in [148] has 512kB of SRAM for data and 2MB of flash memory for storing the parameters weights. While these results are impressive, analog/CMOS hybrid architectures described in the next paragraph have the potential to reduce the power consumption further.

More advanced designs make use of memristors, which are non volatile programmable memories that can be tightly integrated in the CMOS process. The resistance of a memristor can be programmed with the application of an external voltage and changed in a non-volatile way. The main types of memristors are depicted in Fig. 1.9 with the top row showing the states of low resistance and the bottom row showing the states of high resistance. The resistive random access memory (RRAM) in Fig. 1.9 a) relies on the electric-field induced creation and control of conductive filaments between two metallic electrodes separated by an insulating oxide such as hafnium or tantalum oxide [149]. Phase change memories, or PCMs (Fig. 1.9 b)), rely on the phase transition of the middle material. The spin-transfer torque magnetic RAM or STT-MRAM (Fig. 1.9 c)) consists of one magnetic layer with fixed magnetization and one magnetic

[4] https://web.archive.org/web/20210225125431/https://www.wired.com/2017/04/building-ai-chip-saved-google-building-dozen-new-data-centers/

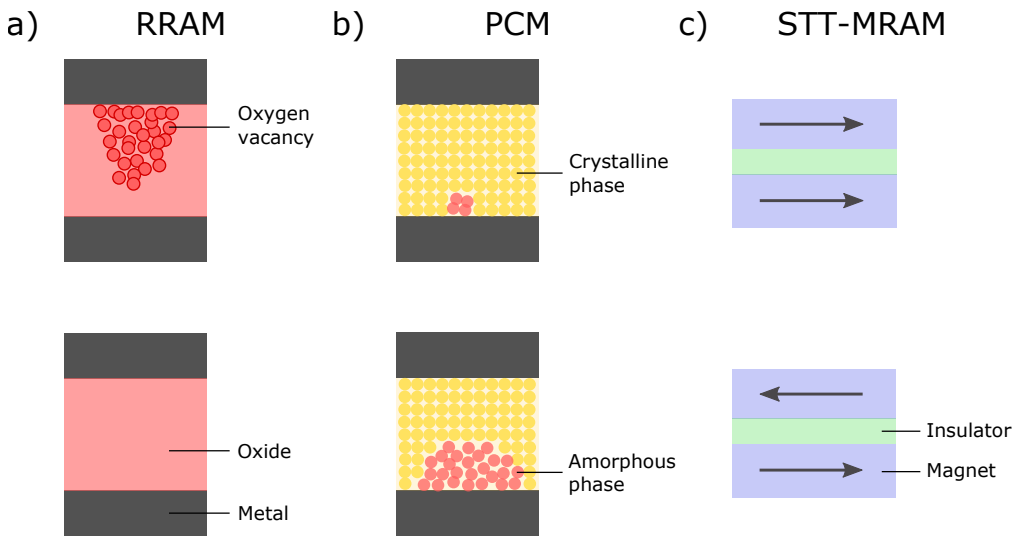

Figure 1.9: Different kinds of memristors, in each case the high resistance state is at the bottom and the low resistance state is on top. a) Resistive RAM based on the formation of oxygen vacancies upon voltage application. b) Phase change memory, based on a phase transition. c) Two magnets are separated by an insulator layer, the magnetization of the bottom magnet is fixed while the magnetization of the top magnet can be programmed by the spin transfer torque of a spin-polarized current tunneling through the junction.

layer with a switchable magnetization on top. Both layers are separated by an insulator forming a magnetic tunnel junction. A spin-polarized current tunneling through the junction applies a torque on the top magnetic layer and can switch the magnetization between both states in a non volatile way.

Memristors are promising for neuromorphic hardware because they can encode model parameters in a physically static and non volatile way. When arranged into *crossbars* like in Fig. 1.10 a), it is possible to implement the matrix multiplication operation out of Kirchhoff's laws. This is appealing because the data corresponding to the parameters, which are encoded as the conductances of the memristors, do not need to be moved from a cache memory to a computing unit, thereby saving energy. This principle has been demonstrated in 2015 by Prezioso et al. [150] for the classification of 3×3 black-and-white pixel images. A multi layer perceptron with one hidden layer was demonstrated on a simple task by Bayat et al. [151] by using one crossbar array for each layer. Ambrogio et al. [152] demonstrated a crossbar architecture capable of equivalent accuracy with software implementation on MNIST [79] and features extracted from CIFAR-10 and CIFAR-100 [93] by a pre-trained ResNet. They report a power consumption of 50 mW, which is a 100 fold power reduction with respect to GPUs.

That being said, crossbars of memristors come with new challenges. One challenge is the non-linear I-V characteristic of memristors which prevent weight updates from being accurately applied. Ref [152] addressed this issue by applying weight updates computed in software

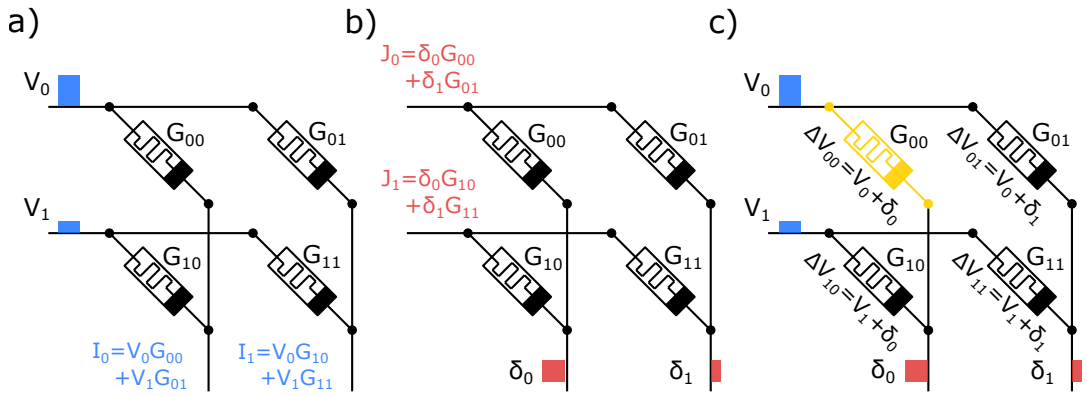

Figure 1.10: Crossbar of memristors for a) forward propagation, b) back-propagation, c) parameter update.

to capacitors with linear but volatile behaviors before transferring the weights to the PCMs. Another challenge is the need for a selection device together with the memristor to prevent the current from leaking in other branches of the crossbar. Finally, memristors operating in the analog domain are subject to device imprecision and variability. Binarized neural networks, a low precision counterpart of regular deep neural network [153] are outstanding candidates for crossbar implementation because the parameters assume binarized value. The memristors can thus be programmed to encode binarized weights, thereby reducing the device variability issue [154–156].

Overall, memristor crossbars are good candidates for low power implementation of inference, although challenges inherent to memristors remain to be coped with in order to scale to big architectures. This will require designing error-tolerant algorithms and improving the technology. However, on-chip training calls for radical shifts in training algorithms and possibly hardware.

1.4.2.2 Dedicated hardware for learning

We can see from sections 1.2 and 1.3.2 that successful training of deep networks requires training algorithms that compute gradients or approximate gradients of a well-defined cost function. Local learning rules based on biological observations such as STDP, although already implemented in some hardware designs, have not been shown to scale in task complexity. However, local learning rules can emerge as the result of well designed training algorithm as we saw in section 1.3, though most of these algorithms have yet to be fully competitive with back-propagation. Crossbar challenges mentioned in the previous section also apply to on-chip learning, but implementing back-propagation in a crossbar brings a new set of challenges. Fig. 1.10 b) depicts how a crossbar can be used to perform the backward phase and propagate error gradients. While the weight transport problem mentioned in section 1.3 is not the major

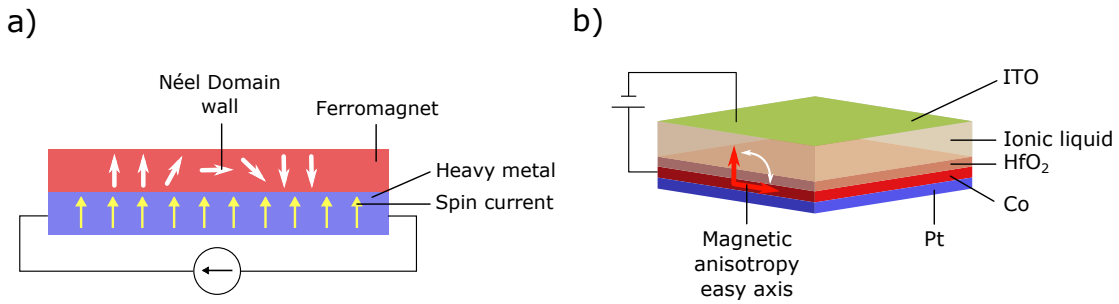

Figure 1.11: a) Magnetic Néel domain wall motion driven by current-induced spin orbit torque. b) Electric field control of magnetism [159, 160]

issue here because the transpose of the weight is naturally implemented by the same devices, the quantities δ are challenging to propagate because they do not have the same magnitude as the neural activation of the forward propagation [67], and require a different set of operations. The imprecision on the errors is likely to worsen as more layers are added. However, in principle, the weight update could be implemented by programming the memristors as shown in Fig. 1.10 c). Transferring data between layers still require costly analog to digital conversions.

A promising way toward on-chip learning is to implement algorithms which only require one type of computation like Equilibrium Propagation described in section 1.3.2.1. EP has the potential to be implemented on a physical substrate comprising of several layers with a parameterized energy function in an end-to-end fashion. A circuit implementing EP based on pseudo-power was simulated and shown to solve MNIST [157] with accuracy equivalent to a network trained in software.

Another attractive path for reducing the power consumption of deep learning is the Optical Processing Unit (OPU) introduced by the LightOn startup [158]. The OPU is a photonics co-processor which employs light scattering to perform fixed random projections in high dimension very efficiently. For specific applications, it can significantly speed up the computation compared to a von Neumann architecture. In particular, direct feedback alignment [83] mentioned in section 1.3 consists exactly in applying a random matrix multiplication to the loss gradient at the output of the network in order to obtain the errors of all layers in parallel. For such specific use cases, LightOn claims a power efficiency two to three orders of magnitude better than TPUs.

Overall, on-chip training will require algorithms and hardware co-design. Learning algorithms with strong theoretical guarantees with respect to optimization are paramount to scale to challenging tasks, and they need to be designed with potential hardware substrates in mind in order to be robust to, and ideally harness, device imperfections. On the other hand, emergent technologies such as non-volatile memories need to be as accurate as possible to allow coherent credit assignment of large network architectures.

New materials are also giving opportunities for novel hardware designs. Fig. 1.11 a) shows

an example of spintronics device consisting of a bi-layer structure with a heavy metal such as platinum for strong spin-orbit coupling, and a ferromagnetic metal such as cobalt on top. Applying an electric current in the heavy metal produces a pure spin current in the perpendicular direction due to spin Hall effect [161]. The spin current reaches the ferromagnetic layer and can move a magnetic domain wall separating two regions of opposite magnetization. The domain wall motion is also dependent on the magnetic anisotropy of the ferromagnet, which has been shown to be programmable in a non volatile and reversible way by the application of a voltage [159, 160, 162], as shown in Fig. 1.11 b). Such examples demonstrate that nanodevices can offer complex behaviors and functionalities for the implementation of learning algorithms.

The work presented in this thesis is a step toward bridging the gap between algorithms and hardware by taking the brain as a conceptual inspiration. In Chapter 2, we propose a continual learning method that bridges the gap between continual learning developed in deep learning [104, 125, 126] and computational neuroscience [107, 116]. More specifically, a local synaptic consolidation rule is found for binarized neural networks [153]. The method does not require boundaries between tasks and is thus promising for neuromorphic applications.

Chapter 3 is dedicated to credit assignment and more specifically to Equilibrium Propagation. We show that the original gradient estimate of EP contains a bias inherent in finite differentiation and propose a new estimate which cancels this bias. The gradient is estimated with three phases instead of two and we show that it unlocks the training of deeper neural architectures with EP. We also show how to adapt the training procedure to optimize the cross-entropy loss instead of the squared error loss. Finally, we study the impact of replacing the weight symmetry condition by an alignment mechanism throughout learning.

Finally, in Chapter 4 of this thesis, we build on the device used by Hirtzlin et al. [154–156] and show that the same device can be used to encode ternarized weights instead of binarized weights when operated in the near-threshold regime, leading to an increase in model performance without overhead. The errors inherent to the devices are incorporated into network simulations and shown to have little impact on the overall performance.

Chapter 2

Synaptic Metaplasticity in Binarized Neural Networks

> "For example, a synapse that is repeatedly potentiated to the point where it is not becoming any stronger should become more resistant to subsequent depotentiation protocols than a synapse that is potentiated to the same degree by a single tetanization."
>
> Stefano FUSI, Patrick DREW, and L.F. ABBOTT [107]

WHILE deep neural networks have surpassed human performance in multiple situations, they are prone to catastrophic forgetting: upon training a new task, they rapidly forget previously learned ones. Neuroscience studies, based on idealized tasks, suggest that in the brain, synapses overcome this issue by adjusting their plasticity depending on their past history. However, such 'metaplastic' behaviours do not transfer directly to mitigate catastrophic forgetting in deep neural networks. In this Chapter, adapted from an article published in Nature Communications [163], we interpret the hidden weights used by binarized neural networks, a low-precision version of deep neural networks, as metaplastic variables, and modify their training technique to alleviate forgetting. Building on this idea, we propose and demonstrate experimentally, in situations of multitask and stream learning, a training technique that reduces catastrophic forgetting without needing previously presented data, nor formal boundaries between datasets and with performance approaching more mainstream techniques with task boundaries. We support our approach with a theoretical analysis on a tractable task based on quadratic optimization. Finally, we present a more complex synaptic model that enables a stationary distribution of parameters, as well as graceful forgetting.

This Chapter bridges computational neuroscience and deep learning, and presents significant assets for future embedded and neuromorphic systems, especially when using novel nanodevices featuring physics analogous to metaplasticity.

2.1 Background

In recent years, deep neural networks have experienced incredible developments, outperforming the state-of-the-art, and sometimes human performance, for tasks ranging from image classification to natural language processing [6]. Nonetheless, these models suffer from catastrophic forgetting [104, 117] when learning new tasks: synaptic weights optimized during former tasks are not protected against further weight updates and are overwritten, causing the accuracy of the neural network on these former tasks to plummet [98, 132] (see Fig. 2.1 (a) and section 1.3.3). Balancing between learning new tasks and remembering old ones is sometimes thought of as a trade-off between plasticity and rigidity: synaptic weights need to be modified in order to learn, but also to remain stable in order to remember. This issue is particularly critical in embedded environments, where data is processed in real-time without the possibility of storing past data. Given the rate of synaptic modifications, most artificial neural networks were found to have exponentially fast forgetting [107]. This contrasts strongly with the capability of the brain, whose forgetting process is typically described with a power law decay [112], and which can naturally perform continual learning.

The neuroscience literature provides insights about underlying mechanisms in the brain that enable task retention. In particular, it was suggested by Fusi et al.[107, 116] that memory storage requires, within each synapse, hidden states with multiple degrees of plasticity (see section 1.3.3.1). For a given synapse, the higher the value of this hidden state, the less likely

this synapse is to change: it is said to be consolidated. These hidden variables could account for activity-dependent mechanisms regulated by intercellular signalling molecules occurring in real synapses [114, 115]. The plasticity of the synapse itself being plastic, this behaviour is named 'metaplasticity'. The metaplastic state of a synapse can be viewed as a criterion of importance with respect to the tasks that have been learned throughout and therefore constitutes one possible approach to overcome catastrophic forgetting.

Until now, the models of metaplasticity have been used for idealized situations in neuroscience studies, or for elementary machine learning tasks such as the Cart-Pole problem [164]. However, intriguingly, in the field of deep learning, binarized neural networks [153] (or the closely related XNOR-NETs [165]) have a remote connection with the concept of metaplasticity, also reminiscent, in neuroscience, of the multi state models with binary readout [166]. This connection has never been explored to perform continual learning in multi-layer networks. Binarized neural networks are neural networks whose weights and activations are constrained to the values $+1$ and -1. These networks were developed for performing inference with low computational and memory cost [154, 167, 168], and surprisingly, can achieve excellent accuracy on multiple vision [165, 169] and signal processing [170] tasks. The training procedure of binarized neural networks involves a real value associated to each synapse which accumulates the gradients of the loss computed with binary weights. This real value is said to be 'hidden', as during inference, we only use its sign to get the binary weight. In this Chapter, we interpret the hidden weight in binarized neural networks as a metaplastic variable that can be leveraged to achieve multitask learning. Based on this insight, we develop a learning strategy using binarized neural networks to alleviate catastrophic forgetting with strong biological-type constraints: previously-presented data can not be stored, nor generated, and the loss function is not task-dependent with weight penalties.

An important benefit of our synapse-centric approach is that it does not require a formal separation between datasets, which also allows the possibility to learn a single task in a more continuous fashion. Traditionally, if new data appears, the network needs to relearn incorporating the new data into the old data: otherwise the network will just learn the new data and forget what it had already learned. Through the example of the progressive learning of datasets, we show that our metaplastic binarized neural network, by contrast, can continue to learn a task when new data becomes available, without seeing the previously presented data of the dataset. This feature makes our approach particularly attractive for embedded contexts. The spatially and temporally local nature of the consolidation mechanism makes it also highly attractive for hardware implementations, in particular using neuromorphic approaches.

Our approach takes a remarkably different direction than the considerable research in deep learning that is now addressing the question of catastrophic forgetting (see section 1.3.3.2). Many proposals consist in keeping or retrieving information about the data or the model at previous tasks: using data generation [128], the storing of exemplars [127], or in preserving the initial model response in some components of the network [118]. These strategies do not seem

connected to how the brain avoids catastrophic forgetting, need a very formal separation of the tasks, and are not very appropriate for embedded contexts. A solution to solve the trade-off between plasticity and rigidity more connected to ours is to protect synaptic weights from further changes according to their 'importance' for the previous task. For example, elastic weight consolidation [104] uses an estimate of the diagonal elements of the Fisher information matrix of the model distribution with respect to its parameters to identify synaptic weights qualifying as important for a given task. Another work[126] uses the sensitivity of the network with respect to small changes in synaptic weights. Finally, in [125], the consolidation strategy consists in computing an importance factor based on path integral. This last approach is the closest to the biological models of metaplasticity, as all computations can be performed at the level of the synapse, and the importance factor is therefore reminiscent of a metaplasticity parameter. However, in all these techniques, the desired memory effect is enforced by optimizing a loss function with a penalty term which depends on the previous optimum, and does not emerge from the synaptic behaviour itself. This aspect requires a very formal separation of the tasks – the weight values at the end of task training need to be stored – and makes these models still largely incompatible with the constraints of biology and embedded contexts. The highly non-local nature of the consolidation mechanism also makes it difficult to implement in neuromorphic-type hardware. Specifically, the contributions of the present Chapter are the following:

- We interpret the hidden real value associated to each weight (or hidden weight) in binarized neural networks as a metaplastic variable, we propose a new training algorithm for these networks adapted to learning different tasks sequentially (Alg. 1).
- We show that our algorithm allows a binarized neural network to learn permuted MNIST tasks sequentially with an accuracy equivalent to elastic weight consolidation, but without any change to the loss function or the explicit computation of a task-specific importance factor. More complex sequences such as MNIST - Fashion-MNIST, MNIST - USPS and CIFAR-10/100 features can also be learned sequentially.
- We show that our algorithm enables to learn the Fashion-MNIST and the CIFAR-10 datasets by learning sequentially each subset of these datasets, which we call the stream-type setting.
- We show that our approach has a mathematical justification in the case of a tractable quadratic binary task where the trajectory of hidden weights can be derived explicitly.
- We adapt our approach to a more complex metaplasticity rule inspired by [116] and show that it can achieve steady-state continual learning. This allows us to discuss the merits and drawbacks of complex and simpler approaches to metaplasticity, especially for hardware implementations of deep learning.

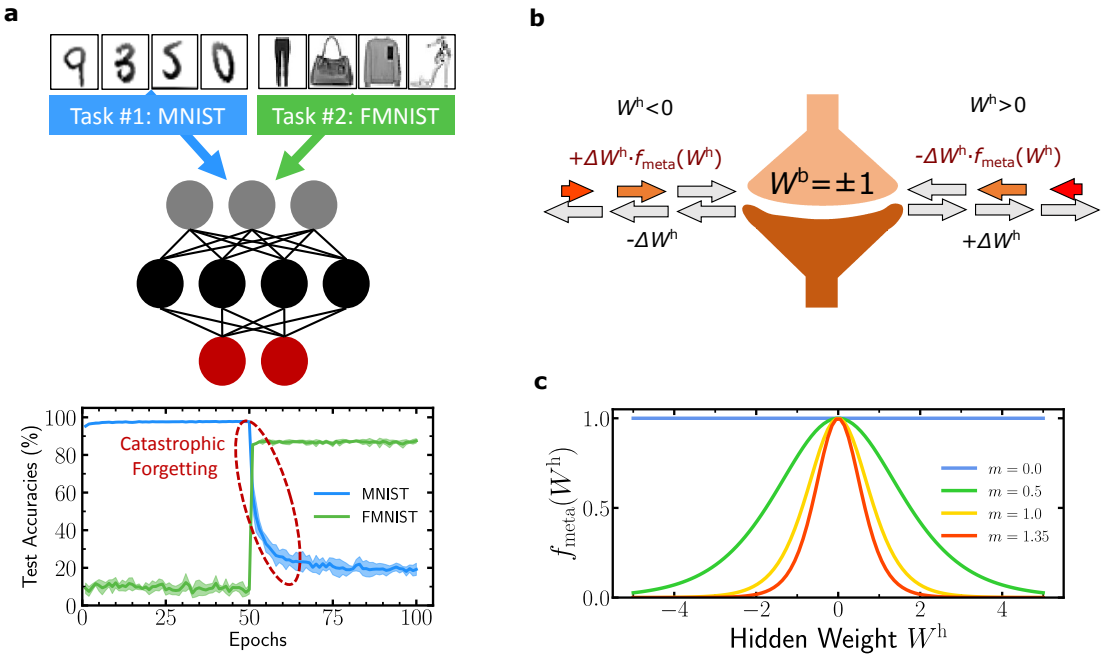

Figure 2.1: **Problem setting and illustration of our approach.** **a** Problem setting: two training sets (here MNIST and Fashion-MNIST) are presented sequentially to a fully connected neural network. When learning MNIST (epochs 0 to 50), the MNIST test accuracy reaches 97%, while the Fashion-MNIST accuracy stays around 10%. When learning Fashion-MNIST (epochs 50 to 100), the associated test accuracy reaches 85% while the MNIST test accuracy collapses to $\sim$ 20% in 25 epochs: this phenomenon is known as 'catastrophic forgetting'. **b** Illustration of our approach: in a binarized neural network, each synapse incorporates a hidden weight W^{h} used for learning and a binary weight $W^{\mathrm{b}} = \mathrm{sign}(W^{\mathrm{h}})$ used for inference. Our method, inspired by neuroscience works in the literature [107], amounts to regarding hidden weights as metaplastic states that can encode memory across tasks and thereby alleviate forgetting. With regards to the conventional training technique of binarized neural network, it consists in modulating some hidden weight updates by a function $f_{\mathrm{meta}}(W^{\mathrm{h}})$ whose shape is indicated in **c**. This modulation is applied to negative updates of positive hidden weights, and to positive updates of negative hidden weights. $f_{\mathrm{meta}}(|W^{\mathrm{h}}|)$ being a decreasing function, this modulation makes the hidden weights signs less likely to switch back when they grow in absolute value.

2.2 Interpreting the hidden weights of binarized neural networks as metaplasticity states

Synapses in binarized neural networks consist of binary switches that can take either +1 or −1 weights. Learning a task consists in finding a set of binary synaptic values that optimize an objective function related to the task at hand. All synapses share the same plasticity rule and are free to switch back and forth between the two weight values. When learning a second task after the first task, new synaptic transitions between +1 and −1 will overwrite the set of transitions found for the first task, leading to the fast forgetting of previous knowledge (Fig. 2.1 (a)). This scenario is reminiscent of the neural networks studied in [171] where all synapses are equally plastic and their probability to remain unchanged over a given period of time decreases exponentially for increasing time periods, leading to memory lifetimes scaling logarithmically with the size of the network. Synaptic metaplasticity models were introduced by Fusi et al[107] to provide long memory lifetimes, by endowing synapses with the ability to adjust their plasticity throughout time – making the plasticity itself plastic. In particular, in this vision, a synapse that is repeatedly potentiated should not increase its weight but rather become more resistant to further depression. In the cascade model [107], plasticity levels are discrete and the probability for a synapse to switch to the opposite strength value decreases exponentially with the depth of the plasticity level. This exponential scaling is introduced to obtain a large range of transition rates, ranging from fast synapses at the top of the cascade where the transition probability is unaffected, to slow synapses that are less likely to switch. Because the metaplastic state only controls the transition probability and not the synaptic strength (i.e., the weight value), it constitutes a 'hidden' state as far as synaptic currents are concerned.

The training process of conventional binarized neural networks relies on updating hidden real weights associated with each synapse, using loss gradients computed with binary weights. The binary weights are the signs of the hidden real weights, and are used in the equations of both the forward and backward passes. By contrast, the hidden weights are updated as a result of the learning rule, which therefore affects the binary weights only when the hidden weight changes sign – the detailed training algorithms are presented in Algorithms 2 and 3 of Appendix A.1. Once the hidden real weight is positive (respectively negative), the binary weight (synaptic strength) is set to +1 (respectively −1), but the synaptic strength will not change if the hidden weight continues to increase towards greater positive (respectively negative) values as a result of the training process. This feature means that hidden weights may be interpreted as analogues to the metaplastic states of the metaplasticity cascade model[107]. However, in conventional binarized neural networks, no mechanism guarantees that when the hidden weight gets updated farther away from zero, the transition to the opposite weight value gets less and less likely. Here, following the insight of [107], we show that introducing such a mechanism yields memory effects.

The mechanism that we propose is illustrated in Fig. 2.1 (b), where W^{h} is the hidden weight and ΔW^{h} is the update provided by the learning algorithm, and detailed in Algorithm 1. We introduce a set of functions f_{meta}, parameterized by a scalar m and depending on the hidden weight to modulate the strength of updates in the inverse direction to the sign of the hidden weights. The specific choice of this set of functions is motivated by the conceptual properties that we want our model to share with the cascade model[107]. First, the functions f_{meta} should be chosen so that the switching strength of the binary weight decreases exponentially with the amplitude of the hidden weight. On the other hand, the switching ability should remain unaffected when the hidden weight is close to zero, making the learning process of such weights analogous to the training of a conventional binarized neural network. We therefore choose a set of functions plotted in Fig. 2.1 (c) that decrease exponentially to zero as the hidden weight $|W^{\mathrm{h}}|$ approaches infinity, while being flat and equal to one around zero values of W^{h}:

$$f_{\mathrm{meta}}(m, W^{\mathrm{h}}) = 1 - \tanh^2(m \cdot W^{\mathrm{h}}). \tag{2.1}$$

The parameter m controls the speed at which the decay occurs and constitutes the only hyper-parameter introduced in our approach. More details about the choice of the f_{meta} function, as well as more implementation details are provided in the Methods section 2.8. All experiments in this Chapter use adaptive moment estimation (Adam) [30]. Momentum-based training and root mean square propagation showed equivalent results. However, pure stochastic gradient descent leads to lower accuracy, as usually observed in binarized neural networks, where momentum is an important element to stabilize training [153, 154, 165].

Algorithm 1 Our modification of the BNN training procedure to implement metaplasticity. $\mathbf{W^h}$ is the vector of hidden weights and W^{h} denotes one component (the same rule is applied for other vectors), $\boldsymbol{\theta}^{\mathrm{BN}}$ are Batch Normalization parameters, $\mathbf{U_W}$ and $\mathbf{U}_\theta$ are the parameter updates prescribed by the Adam algorithm [30], $(\mathbf{x},\mathbf{y})$ is a batch of labelled training data, m is the metaplasticity parameter, and η is the learning rate. ' $\cdot$ ' denotes the element-wise product of two tensors with compatible shapes. The difference between our implementation and the non-metaplastic implementation (recovered for $m = 0$) lies in the condition lines 6 to 9. f_{meta} is applied element-wise with respect to $\mathbf{W^h}$. 'cache' denotes all the intermediate layers computations needed to be stored for the backward pass. The details of the Forward and Backward functions are provided in Appendix A.1.

Input: $\mathbf{W^h}$, $\boldsymbol{\theta}^{\mathrm{BN}}$, $\mathbf{U_W}$, $\mathbf{U}_\theta$, $(\mathbf{x},\mathbf{y})$, m, η.
Output: $\mathbf{W^h}$, $\boldsymbol{\theta}^{\mathrm{BN}}$, $\mathbf{U_W}$, $\mathbf{U}_\theta$.

1: $\mathbf{W^b} \leftarrow \mathrm{Sign}(\mathbf{W^h})$ ▷ Computing binary weights
2: $\hat{\mathbf{y}}, \mathrm{cache} \leftarrow \mathrm{Forward}(\mathbf{x}, \mathbf{W^b}, \boldsymbol{\theta}^{\mathrm{BN}})$ ▷ Perform inference
3: $C \leftarrow \mathrm{Cost}(\hat{\mathbf{y}}, \mathbf{y})$ ▷ Compute mean loss over the batch
4: $(\partial_{\mathbf{W}} C, \partial_\theta C) \leftarrow \mathrm{Backward}(C, \hat{\mathbf{y}}, \mathbf{W^b}, \boldsymbol{\theta}^{\mathrm{BN}}, \mathrm{cache})$ ▷ Cost gradients
5: $(\mathbf{U_W}, \mathbf{U}_\theta) \leftarrow \mathrm{Adam}(\partial_{\mathbf{W}} C, \partial_\theta C, \mathbf{U_W}, \mathbf{U}_\theta)$
6: **for** W^{h} in $\mathbf{W^h}$ **do**
7: **if** $U_W \cdot W^{\mathrm{b}} > 0$ **then** ▷ If U_W prescribes to decrease $|W^{\mathrm{b}}|$
8: $W^{\mathrm{h}} \leftarrow W^{\mathrm{h}} - \eta U_W \cdot f_{\mathrm{meta}}(m, W^{\mathrm{h}})$ ▷ Metaplastic update
9: **else**
10: $W^{\mathrm{h}} \leftarrow W^{\mathrm{h}} - \eta U_W$
11: **end if**
12: **end for**
13: $\boldsymbol{\theta}^{\mathrm{BN}} \leftarrow \boldsymbol{\theta}^{\mathrm{BN}} - \eta \mathbf{U}_\theta$
14: **return** $\mathbf{W^h}$, $\boldsymbol{\theta}^{\mathrm{BN}}$, $\mathbf{U_W}$, $\mathbf{U}_\theta$

2.3 Multitask learning with metaplastic binarized neural networks

We first test the validity of our approach by learning sequentially multiple versions of the MNIST dataset where the pixels have been permuted, which constitutes a canonical benchmark for continual learning [117]. We train a binarized neural network with two hidden layers of 4,096 units using Algorithm 1 with several metaplasticity m values and 40 epochs per task (see section 2.8). Fig. 2.2 shows this process of learning six tasks. The conventional binarized neural network ($m = 0.0$) is subject to catastrophic forgetting: after learning a given task, the test accuracy quickly drops upon learning a new task. Increasing the parameter m gradually prevents the test accuracy on previous tasks from decreasing with eventually the $m = 1.35$ binarized neural network (Fig. 2.2 (d) managing to learn all six tasks with test accuracies comparable with the 97.4% test accuracy achieved by the BNN trained on one task only (see Table. 2.1).

Fig. 2.2 (g,h) show the distribution of the metaplastic hidden weights after learning Task 1 and Task 2 in the second layer. The consolidated weights of the first task correspond to hidden

weights between zero and five in magnitude. We observe in Fig. 2.2 (g) that around 10^7 of binary weights still have hidden weights near zero after learning one task. These weights correspond to synapses that repeatedly switched between +1 and −1 binary weights during the training of the first task, and are thus of little importance for the first task. These synapses were therefore not consolidated, and are then available for learning another task. After learning the second task, we can distinguish between hidden weights of synapses consolidated for Task 1 and for Task 2.

	No consolidation ($m = 0.0$)	Random Consolidation	Learning Rate Decay	Elastic Weight Consolidation	Metaplasticity ($m = 1.35$)
Task 1	9.2 ± 2.2	29.0 ± 2.9	71.1 ± 6.5	96.8 ± 0.7	96.9 ± 0.6
Task 2	7.8 ± 1.3	29.0 ± 4.2	87.2 ± 2.6	97.2 ± 0.2	97.2 ± 0.3
Task 3	9.3 ± 2.0	32.7 ± 4.7	86.1 ± 2.9	96.9 ± 0.2	96.9 ± 0.2
Task 4	9.0 ± 1.7	35.1 ± 4.1	63.7 ± 5.6	96.6 ± 0.2	96.4 ± 0.4
Task 5	13.2 ± 3.7	47.7 ± 8.8	75.1 ± 2.5	96.8 ± 0.3	96.7 ± 0.8
Task 6	97.4 ± 0.2	96.8 ± 0.2	93.9 ± 0.2	96.8 ± 0.3	97.3 ± 0.1

Table 2.1: Binarized neural network test accuracies on six permuted MNISTs at the end of training for different settings. We indicate mean and standard deviation over five trials, for a conventional (non-metaplastic) BNN ($m = 0.0$), a task-dependent learning rate decay scheduler, consolidation of synapses with random importance factors, elastic weight consolidation (EWC) [104] computed with parameter $\lambda_{\text{EWC}} = 5 \cdot 10^3$, and our metaplastic binarized neural network approach with parameter $m = 1.35$.

Table 2.1 presents a comparison of the results obtained using our technique with a random consolidation of weights, and with elastic weight consolidation [104], implemented on the same binarized neural network architecture (see section 2.8 for the details of EWC adaptation to BNNs). We see that the random consolidation approach does not allow multitask learning. On the other hand, our approach achieves a performance similar to elastic weight consolidation for learning six permuted MNISTs with the given architecture, although unlike elastic weight consolidation the consolidation does not require changing the loss function and thus does not require task boundaries.

We also perform a control experiment by decreasing the learning rate between each task. The initial learning rate is divided by ten for each new task, as this schedule provided the best results (see Appendix A.6). This technique achieves some memory effects but is not as effective as other consolidation methods: uniformly scaling down the learning rate for all synapses at once does not provide a wide range of synaptic plasticity where important synapses are consolidated and less important ones are more plastic.

Figure 2.3 shows a more detailed analysis of the performance of our approach when learning up to ten MNIST permutations, and for varying sizes of the binarized neural network, highlighting the connection between network size and its capacity in terms of number of tasks. We

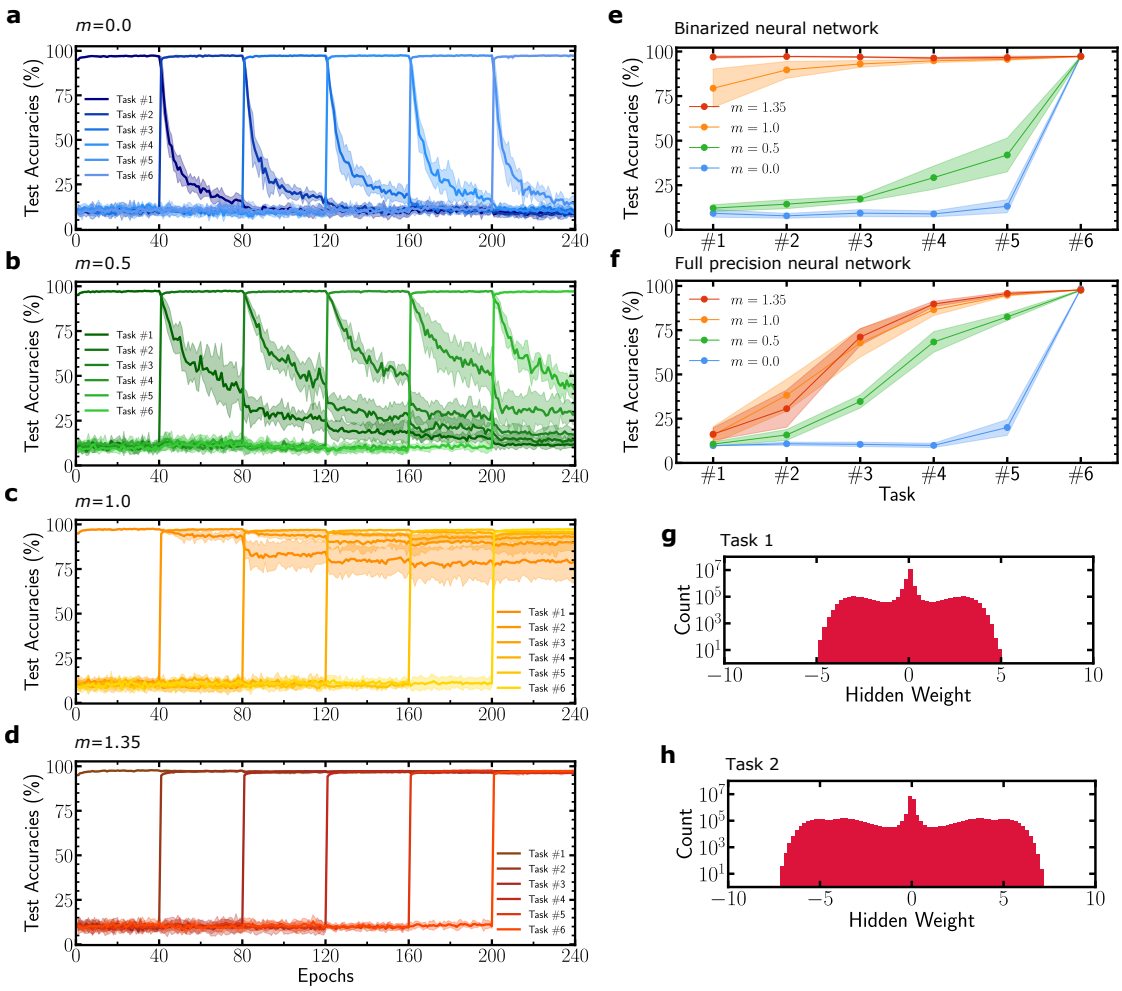

Figure 2.2: **Permuted MNIST learning task. a-d** Binarized neural network learning six tasks sequentially for several values of the metaplastic parameter m. (**a**) $m = 0$ corresponds to a conventional binarized neural network (**b**) $m = 0.5$ (**c**) $m = 1.0$ (**d**) $m = 1.35$. Curves are averaged over five runs and shadows correspond to one standard deviation. **e**,**f** Final test accuracy on each task after the last task has been learned. The dots indicate the mean values over five runs, and the shaded zone one standard deviation. (**e**) corresponds to a binarized neural network and (**f**) corresponds to our method applied to a real valued weights deep neural network with the same architecture. **g**,**h** Hidden weights distribution of a $m = 1.35$, two hidden layers of 4,096 units binarized neural network after learning for 40 epochs one permuted MNIST (**g**) and two permuted MNISTs (**h**).

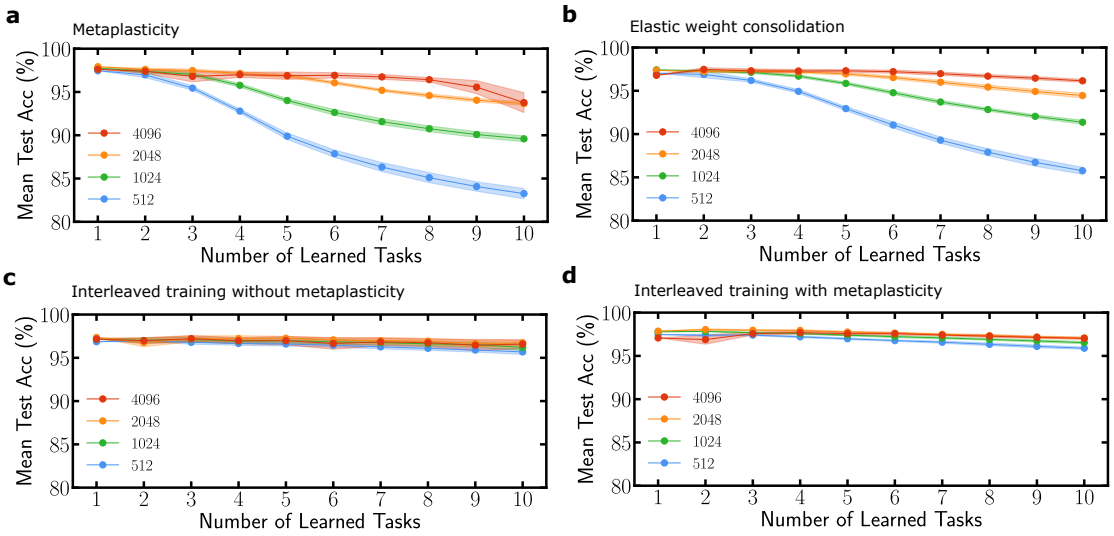

Figure 2.3: **Influence of the network size on the number of tasks learned.** **a**, **b** Mean test accuracy over tasks learned so far for up to ten tasks. Each task is a permuted version of MNIST learned for 40 epochs. The binarized neural network architecture consists of two hidden layers of a variable number of hidden units ranging from 512 to 4096. (**a**) uses metaplasticity with parameter $m = 1.35$ and (**b**) uses elastic weight consolidation with $\lambda_{\text{EWC}} = 5,000$. The decrease in mean test accuracy comes from the impossibility to learn new tasks because too many weights are consolidated. **c**,**d** Results for non-sequential (interleaved) training for (**c**) a non metaplastic and (**d**) a metaplastic binarized neural network. In this situation, each point is an independent training experiment performed on the corresponding number of tasks. All curves are averaged over five runs and shadow areas denote one standard deviation.

see that in this harder situation, elastic weight consolidation is more efficient with respect to the network size, especially for smaller networks. Figs. 2.3 c,d show the accuracy obtained when all tasks are learned at once, for a non-metaplastic and metaplastic binarized neural network. This result quantifies the capacity reduction induced by sequential learning. We also compare our approach with 'synaptic intelligence' introduced in [125] in Fig. A.1 in Appendix A.3. This approach features task boundaries as in the case of elastic weight consolidation, but can perform most operations locally, bringing it closer to biology, while retaining near-equivalent accuracy to elastic weight consolidation[125]. Contrary to elastic weight consolidation, synaptic intelligence does not adapt well to a binarized neural network: the importance factor involving a path integral cannot be computed in a natural manner using binarized weights (see Appendix A.3), leading to poor performance (Fig. A.1 (b) in Appendix A.3). On the other hand, synaptic intelligence applied to full precision neural networks requires less synapses than our binarized approach for equivalent accuracy (Fig. A.1 (a) in Appendix A.3), as binarized neural networks always require more synapses than full precision ones to reach equivalent accuracy [154, 172]. Our technique, therefore, approaches but does not match the accuracy of task-separated approaches. The major motivation of our approach are the possibilities allowed by the absence of task boundaries, such as the stream learning situation investigated in the next section.

Finally, as a control experiment, we also applied Algorithm 1 to a full precision network, except for the weight binarization step described in line one. Fig. 2.2 (e) and Fig. 2.2 (f) show the final accuracy of each task at the end of learning for a binarized neural network and a real valued weights deep neural network respectively, with the same architecture. The full precision network final test accuracy of each task for the same range of m values cannot retain more than three tasks with accuracy above 90%. This result highlights that our weight consolidation strategy is tied specifically to the use of hidden weights.

Hidden weights in a binarized neural network and real weights in a full precision neural network respectively possess fundamentally different meanings. In full precision networks, the inference is carried out using the real weights, in particular the loss function is also computed using these weights. Conversely in binarized neural networks, the inference is done with the binary weights and the loss function is also evaluated with these binary weights, which has two major consequences. First, the hidden weights do not undergo the same updates as the weights of a full precision network. Second, a change on a synapse whose hidden weight is positive and which is prescribed a positive update consequently will not affect the loss, nor its gradient at the next learning iteration since the loss only takes into account the sign of the hidden weights. Hidden weights in binarized neural networks consequently have a natural tendency to spread over time (Fig. 2.2 (g,h), and they are not weights properly speaking. Fig. 2.6 illustrates this difference visually. In a full precision neural network, 'important' weights for a task converge to an optimum value minimizing the loss. By contrast, in a binarized neural network, when a binarized weight has stabilized to its optimum value, its hidden weight keeps

increasing, thereby clearly indicating that the synapse should be consolidated. At the end of the paper, we provide a deeper mathematical interpretation of this intuition.

We also tested the capability of our binarized neural network to learn sequentially different datasets, in several situations. We first investigated the sequential training of the MNIST and the Fashion-MNIST dataset, presenting apparel items [173]. While a non-metaplastic network rapidly forgets the first dataset when the second one is trained (Fig. 2.4 (a), an optimized metaplastic network learns both tasks with accuracies near the ones achieved when the tasks are learned independently (Fig. 2.4 (b). More details and more results are presented in Appendix A.7. Fig. 2.4 (c) presents the sequential training of two closely related datasets: MNIST, and of a second handwritten digits dataset (United States Postal Services). A small amount of data is used in this experiment to keep the balance between the two datasets (see Appendix A.8). The baselines are non-metaplastic networks obtained by partitioning the metaplastic network into two equal parts (each featuring half the number of hidden neurons), and trained independently on each task. We see that the metaplastic network learns sequentially both datasets successfully with accuracies above the baselines, suggesting that for a fixed number of hidden neurons, metaplasticity can provide an increase in capacity. Fig. 2.4 (d) presents a variation of this situation with the same baselines, and where the metaplastic network is this time designed with a number of parameters doubled with regards to the baselines (see Appendix A.8). In that case, the accuracy of the sequentially trained metaplastic network still succeeds at matching, but does not exceed the non-sequentially trained baselines. Finally, we investigated a situation of class incremental learning of the CIFAR-10 (Figs. 2.4(e-f)) and CIFAR-100 (Figs. 2.4(g-h)) datasets. We use a convolutional neural network with convolutional layers pretrained on ImageNet, and a metaplastic classifier (see Appendix A.9). The classes of these datasets are divided into two subsets and trained sequentially. While in the non-metaplastic network (Figs. 2.4(e-g)), the first subset of classes is forgotten rapidly when the second is trained, in the metaplastic one (Figs. 2.4(f-h)), good accuracy is achieved, which remains below the one obtained with non sequentially trained classes. Better performance can be achieved if we allow the neurons to have independent thresholds for the two subsets (see Appendix A.9).

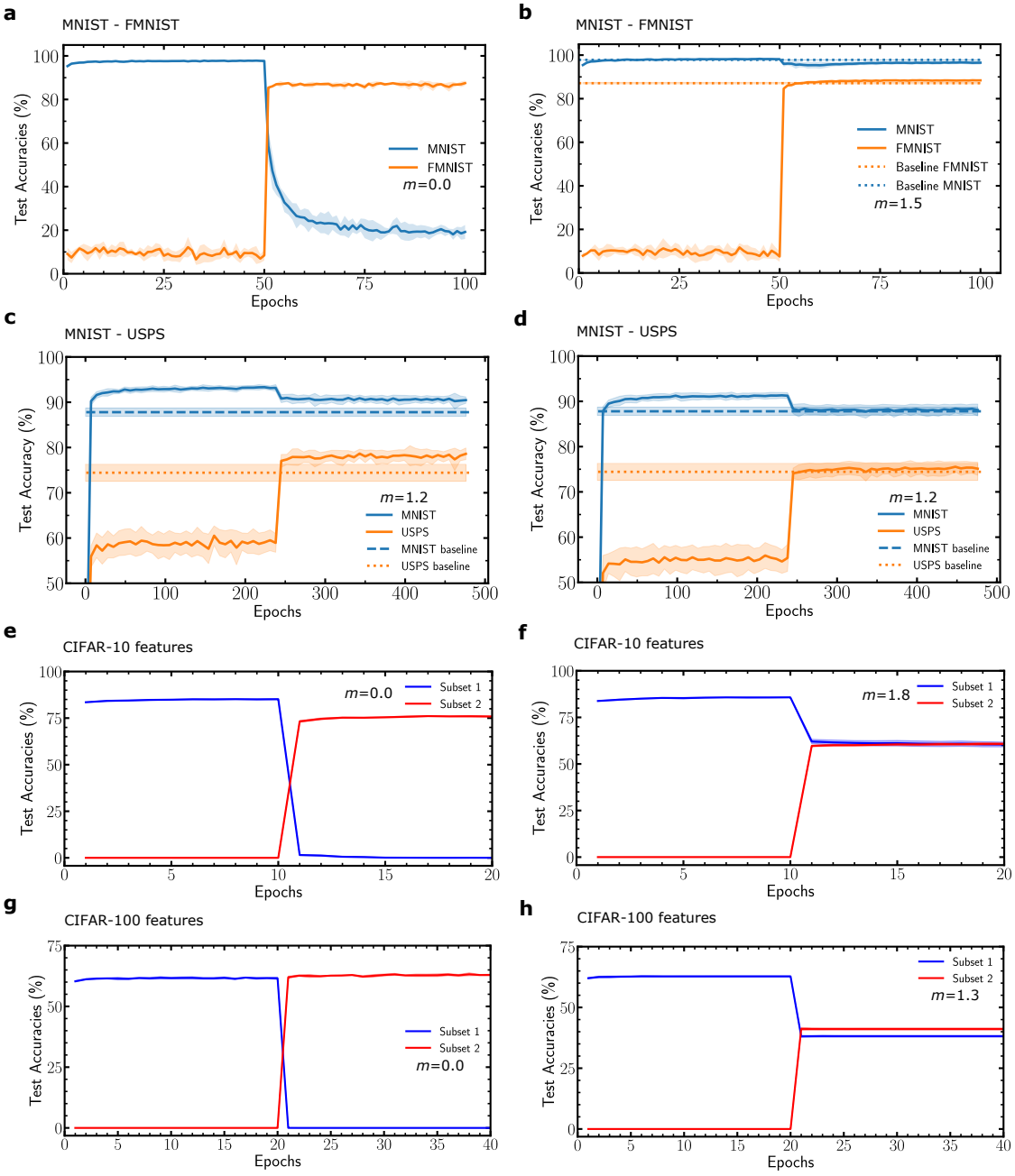

Figure 2.4: **Sequential learning on various datasets.** **a**, **b** Binarized neural network learning MNIST and Fashion-MNIST sequentially (**a**) without metaplasticity and (**b**) with metaplasticity. **c** Sequential training of the MNIST and USPS datasets of handwritten digits. The baselines correspond to the accuracy reached by non-metaplastic networks with half the number of neurons trained independently on each task. **d** presents the same experiment as **c**, with a metasplastic network featuring a doubled number of parameters with regards to the baselines. **e**, **f** Test accuracy when learning sequentially two subsets of CIFAR-10 classes from features extracted by a pretrained ResNet on ImageNet (see Appendix A.9). **g**, **h** Same experiment with CIFAR-100 features. All curves except **c**, **d** are averaged over five runs. **c** and **d** are averaged over fifty runs due to the small amount of data (see Appendix A.8). Shadows correspond to one standard deviation.

2.4 Stream learning: learning one task from subsets of data

We have shown that the hidden weights of binarized neural networks can be used as importance factors for synaptic consolidation. Therefore, in our approach, it is not required to compute an explicit importance factor for each synaptic weight: our consolidation strategy is carried out simultaneously with the weight update as consolidation only involves the hidden weights. The absence of formal dataset boundaries in our approach is important to tackle another aspect of catastrophic forgetting where all the training data of a given task is not available at the same time. In this section, we use our method to address this situation, which we call 'stream learning': the network learns one task but can only access one subset of the full dataset at a given time. Subsets of the full dataset are learned sequentially and the data of previous subsets cannot be accessed in the future.

We first consider the Fashion-MNIST dataset, split into 60 subsets presented sequentially during training (see section 2.8). The learning curves for regular and metaplastic binarized neural networks are shown in Fig. 2.5 (a), the dashed lines corresponding to the accuracy reached by the same architecture trained on the full dataset after full convergence. We observe that the metaplastic binarized neural network trained sequentially on subsets of data performs as well as the non-metaplastic binarized neural network trained on the full dataset. The difference in accuracy between the baselines can be explained by our consolidation strategy gradually reducing the number of weights able to switch, therefore acting as a learning rate decay (the mean accuracy achieved by a binarized neural network with $m = 0$ trained with a learning rate decay on all the data is 88.8%, equivalent to the metaplastic baseline in Fig. 2.5 (a).

In order to see if the advantage provided by metaplastic synapses holds for convolutional networks and harder tasks, we then consider the CIFAR-10 dataset, with a binarized version of a Visual Geometry Group (VGG) convolutional neural network (see section 2.8). CIFAR-10 is split into 20 sub datasets of 2,500 examples. The test accuracy curve of the metaplastic binarized neural network exhibits a gap with baseline accuracies smaller than the non-metaplastic one. Our metaplastic binarized neural network can thus gain new knowledge from new data without forgetting previously learned unavailable data. Because our consolidation strategy does not involve changing the loss function and the batch normalization settings are common across all subsets of data, the metaplastic binarized neural network gains new knowledge with each subset of data without any information about subsets boundaries. This feature is especially useful for embedded applications, and is not currently possible in alternative approaches of the literature to address catastrophic forgetting.

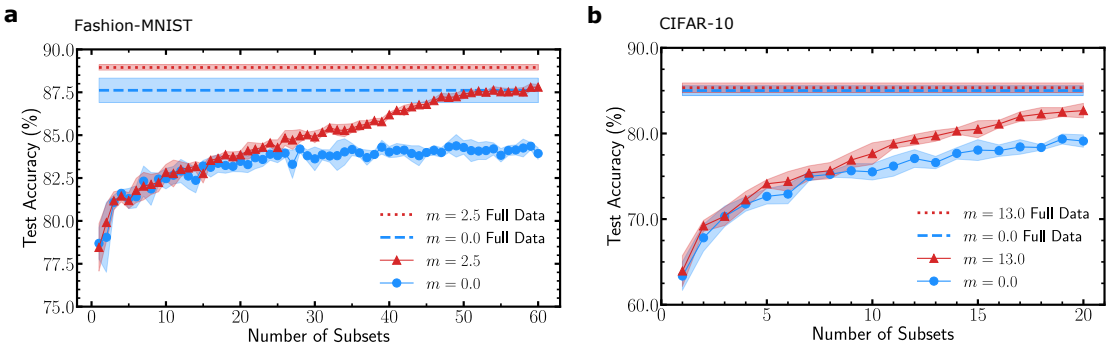

Figure 2.5: **Stream learning experiments.** **a** Progressive learning of the Fashion-MNIST dataset. The dataset is split into 60 parts consisting of only 1,000 examples, and containing all ten classes. Each sub dataset is learned for 20 epochs. The dashed lines represent the accuracies reached when the training is done on the full dataset for 20 epochs so that all curves are obtained with the same number of optimization steps. **b** Progressive learning of the CIFAR-10 dataset. The dataset is split into 20 parts, consisting of only 2,500 examples. Each sub dataset is learned for 200 epochs. The dashed lines represent the accuracies reached when the training is done on the full dataset for 200 epochs. Shadows correspond to one standard deviation around the mean over five runs.

2.5 Mathematical interpretation

We now provide a mathematical interpretation for the hidden weights of binarized neural networks, illustrated graphically in Fig 2.6. We show in archetypal situations that the larger a hidden weight gets while learning a given task, the bigger the loss increase upon flipping the sign of the associated binary weight, and consequently the more important they are with respect to this task. For this purpose, we define a quadratic binary task, an analytically tractable and convex counterpart of a binarized neural network optimization task.

Definition 1 (Quadratic Binary Task)*:* Consider the loss function:

$$\mathscr{L}(\mathbf{W}) = \frac{1}{2}(\mathbf{W}-\mathbf{W}^\star)^T \cdot \mathbf{H} \cdot (\mathbf{W}-\mathbf{W}^\star) \tag{2.2}$$

with a symmetric definite positive matrix $\mathbf{H} \in \mathbb{R}^{d\times d}$. Gradients are given by $\mathbf{g}(\mathbf{W}) = \mathbf{H}\cdot(\mathbf{W}-\mathbf{W}^\star)$. We assume the following optimization scheme:

$$\mathbf{W^h}_{t+1} = \mathbf{W^h}_t - \eta \mathbf{H} \cdot (\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star), \tag{2.3}$$

where sign returns the sign of a vector component-wise.

This task consists in finding the global optimum on a landscape featuring a uniform (Hessian) curvature. The gradient used for the optimization is evaluated using only the signs of the pa-

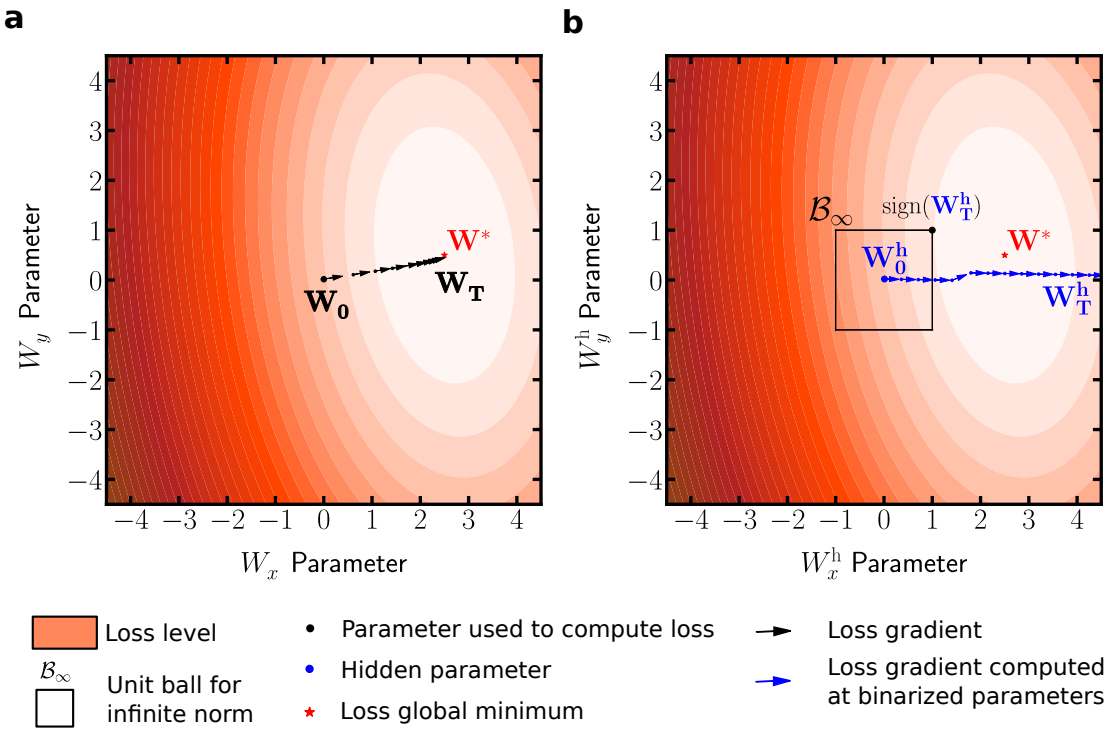

Figure 2.6: **Difference between standard and binarized optimization. a** Example of standard optimization in a two-dimensional quadratic binary task. The parameters converge to the global minimum. **b** Example of hidden weights trajectory in a two-dimensional quadratic binary task. One hidden weight W^{h}_x diverges because the optimal hidden weight vector $\mathbf{W}^\star$ has uniform norm greater than one (Lemma 1).

rameters $\mathbf{W^h}$ (Fig. 2.6 (b), in the same way that binarized neural networks employ only the sign of hidden weights for computing gradients during training. We demonstrate theoretically that throughout optimization on the quadratic binary task, if the uniform norm of the weight optimum vector is greater than one, the hidden weights vector diverges.

Lemma 1 (Condition for hidden Weight confinement)*:* Let $\mathbf{W^h}$ optimize a quadratic binary task according to the dynamics $\mathbf{W^h}_{t+1} = \mathbf{W^h}_t - \eta\mathbf{H}(\mathrm{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)$. Let $\mathscr{B}_\infty$ be the unit ball for the infinite norm and $\overline{\mathscr{B}}_\infty$ its closure. Then:

$$\mathbf{W}^\star \in \mathscr{B}_\infty \quad \Rightarrow \quad \exists C > 0, \quad \forall t \in \mathbb{N}, \quad \|\mathbf{W^h}_t\|_\infty < C \tag{2.4}$$

$$\mathbf{W}^\star \notin \overline{\mathscr{B}}_\infty \quad \Rightarrow \quad \lim_{t\to\infty} \|\mathbf{W^h}_t\|_\infty = \infty \tag{2.5}$$

Fig. 2.6 (b) shows an example in two dimensions where such a divergence is seen. The full proof is given in Appendix A.5. The idea of the proof for 2.4 is that if i is a dimension index for which $W^\star_i > 1$, and $\mathbf{e}_i$ is the canonical basis vector along dimension i, then the hidden weight update

projected on $\mathbf{H}^{-1}\mathbf{e}_i$ can only take two values that have the same sign and are non zero. The idea of the proof for 2.5 is to show that when the norm of $\mathbf{W^h}_t$ is large enough, then the $\|\mathbf{W^h}_t\|_{\mathbf{H}^{-1}}$ can only decrease. However, this idea does not work when $\mathbf{W}^\star$ is on the closure of $\mathscr{B}_\infty$, but we conjecture that the hidden weights also remain bounded in this case, possibly in an area that depend on the initialisation value.

In the particular case of a diagonal Hessian curvature, a correspondence exists between diverging hidden weights and components of the weight optimum greater than one in absolute value. We can derive an explicit form for the asymptotic evolution of the diverging hidden weights while optimizing (proof in Appendix A.5).

Lemma 2 (hidden Weight Trajectory)*:* Let $\mathbf{W^h}$ optimize a quadratic binary task according to the dynamics $\mathbf{W^h}_{t+1} = \mathbf{W^h}_t - \eta\mathbf{H}(\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)$ and assume $\mathbf{H} = \text{diag}(\lambda_1, \dots \lambda_d)$. Then:

$$|W_i^\star| > 1 \implies W^{\text{h}}_{i,t} \sim_{t\to+\infty} \underbrace{\text{sign}(W_i^\star)\eta\lambda_i(|W_i^\star| - 1)}_{=\widetilde{W}^{\text{h}}_i}\, t \tag{2.6}$$

The hidden weights diverge linearly: $W^{\text{h}}_{i,t} \sim \widetilde{W}^{\text{h}}_i\, t$ with a speed proportional to the curvature and the absolute magnitude of the global optimum. Given this result, we can prove the following theorem (proof in Appendix A.5):

Theorem 1: Let $\mathbf{W}$ optimize the quadratic binary task with optimum weight $\mathbf{W}^\star$ and curvature matrix $\mathbf{H}$, using the optimization scheme: $\mathbf{W^h}_{t+1} = \mathbf{W^h}_t - \eta\mathbf{H}\cdot(\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)$. We assume $\mathbf{H}$ equal to $\text{diag}(\lambda_1, \dots \lambda_d)$ with $\lambda_i > 0$, $\forall i \in [\![1, d]\!]$. Then, if $|W_i^\star| > 1$, the variation of loss resulting from flipping the sign of $W^{\text{b}}_{i,t}$ is:

$$\Delta_i\mathscr{L}(\mathbf{W}_t) \sim 2\lambda_i + 2\frac{|\widetilde{W}^{\text{h}}_i|}{\eta} \quad \text{as} \quad t \to +\infty. \tag{2.7}$$

This theorem states that the increase in the loss induced by flipping the sign of a diverging hidden weight is asymptotically proportional to the sum of the curvature and a term proportional to the hidden weight. Hence the correlation between high valued hidden weights and important binary weights.

Interestingly, this interpretation, established rigorously in the case of a diagonal Hessian curvature, may generalize to non-diagonal Hessian cases. Fig. 2.7 (a), for example, illustrates the correspondence between hidden weights and high impact on the loss by sign change on a quadratic binary task with a 500-dimensional non-diagonal Hessian matrix (see section 2.8 for the generation procedure). Fig. 2.7 (b,c,d) finally show that this correspondence extends to a practical binarized neural network situation, trained on MNIST. In this case, the cost variation $\mathbb{E}_{\text{data}}(\Delta\mathscr{L})$ upon switching binary weights signs increases monotonically with the magnitudes of the hidden weights (see section 2.8 for implementation details). These results provide an interpretation as to why hidden weights can be thought of as local importance factors useful for continual learning applications.

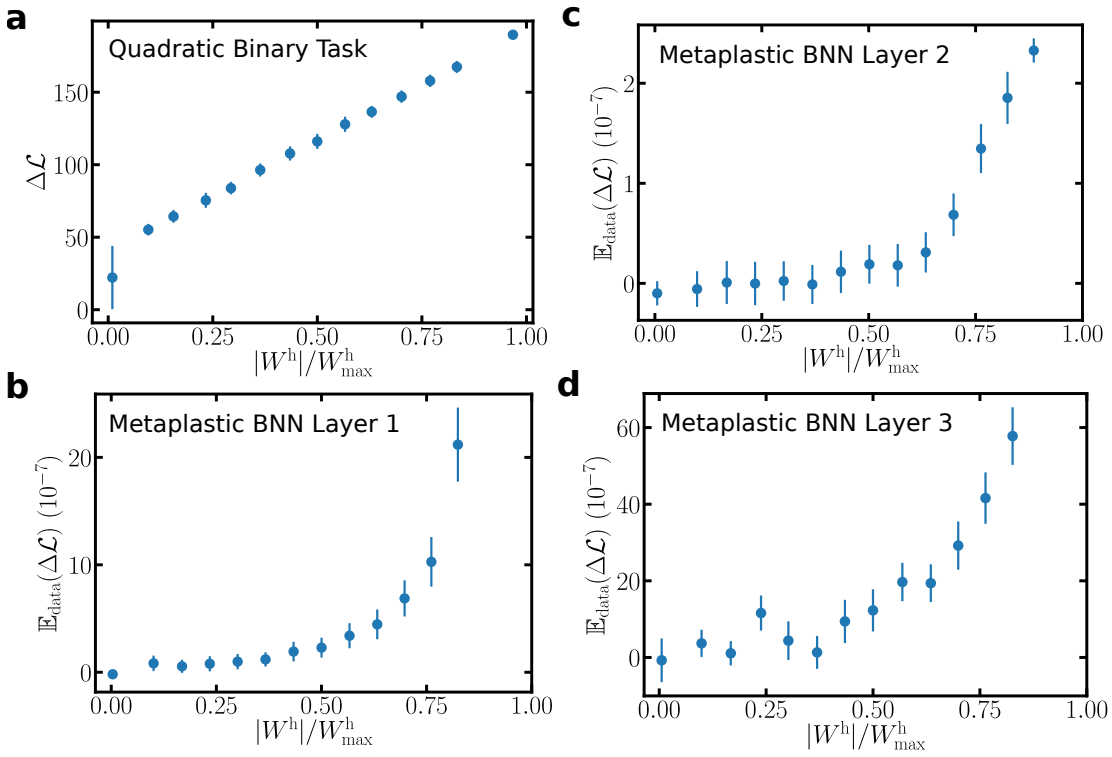

Figure 2.7: **High hidden weights correspond to important parameters.** **a** Mean increase in the loss occurred by switching the sign of a hidden weight as a function of the normalized value of the hidden weight, for a 500-dimensional quadratic binary task. The mean is taken by assigning hidden weights to bins of increasing absolute value and the error bars denote one standard deviation around the mean. The leftmost point corresponds to hidden weights staying bounded. **b**, **c**, **d** Increase in the loss occurred by switching the sign of hidden weights as a function of the normalized absolute value of the hidden weight in a binarized neural network trained on MNIST. Each dot is the mean increase over 100 realizations of weights to be switched and the error bars denote one standard deviation. The scales differ because the layers have different numbers of weights and thus different relative importance. See section 2.8 for implementation details.

2.6 Increasing Synapse Complexity for Steady-State Continual Learning

In this section, we show that one limitation of the metaplasticity model presented in section 2.2 can be alleviated by considering a more complex synaptic model inspired by the metaplasticity model of Benna and Fusi [116]. The metaplasticity model presented in section 2.2 has an aging property that prevents it from learning new tasks after learning a finite number of tasks (depending on the capacity of the network). For instance, Figure 2.8(b) shows the accuracies

of ten tasks learned by the metaplasticity model introduced in section 2.2. While the network successfully learns up to seven tasks, the last three tasks are not properly learned because all the weights have been consolidated. The type of continual learning achieved by this network is therefore non steady-state, consistently with much of the machine learning literature on continual learning[121], but unlike the brain.

The metaplasticity model introduced in [116] describes synapses with several hidden variables interacting over a wide range of timescales through diffusion processes. The slowest variable features a leakage term, allowing the possibility to reach a steady-state type of consolidated learning, where the newest memories can replace the firstly trained ones. Here, we propose training binarized neural networks with synapses featuring not a simple hidden weight, but a collection of them interacting over a wide range of timescales in a way inspired by [116]. This approach can have several benefits. First, the hidden weights tend to evolve stochastically in conventional binarized neural networks due to the stochastic nature of data batches. This means that in our original metaplasticity approach, if a hidden weight gets carried too far away from zero because of the noise, it will be consolidated. The more complicated synapses inspired by [116] can provide a cleaner signal to perform weight consolidation and constitute promising candidates to solve the issue of noise-induced consolidation. Second, and more importantly, thanks to the leakage on the slower variable, we hope to provide the binarized neural network with a truly steady-state form of continual learning.

In our model, each synapse features four hidden variables (W_1^{h}, W_2^{h}, W_3^{h}, and W_4^{h}), which evolve according to :

$$\begin{cases} W_1^{\mathrm{h}}(t+1) = W_1^{\mathrm{h}}(t) - \eta \dfrac{\partial \mathscr{L}}{\partial W_{\mathrm{b}}} + g_{1,2}(W_2^{\mathrm{h}}(t) - W_1^{\mathrm{h}}(t)) \quad \text{if} \quad (W_1^{\mathrm{h}}(t) - W_4^{\mathrm{h}}(t)) \cdot \mathrm{sign}(W_4^{\mathrm{h}}(t)) > 0 \\ W_1^{\mathrm{h}}(t+1) = W_1^{\mathrm{h}}(t) - \eta \dfrac{\partial \mathscr{L}}{\partial W_{\mathrm{b}}} + g_{1,2}(W_2^{\mathrm{h}}(t) - W_1^{\mathrm{h}}(t)) + \alpha(W_4^{\mathrm{h}}(t) - W_1^{\mathrm{h}}(t)) \quad \text{otherwise} \\ W_2^{\mathrm{h}}(t+1) = W_2^{\mathrm{h}}(t) + g_{1,2}(W_1^{\mathrm{h}}(t) - W_2^{\mathrm{h}}(t)) + g_{2,3}(W_3^{\mathrm{h}}(t) - W_2^{\mathrm{h}}(t)) \\ W_3^{\mathrm{h}}(t+1) = W_3^{\mathrm{h}}(t) + g_{2,3}(W_2^{\mathrm{h}}(t) - W_3^{\mathrm{h}}(t)) + g_{3,4}(W_4^{\mathrm{h}}(t) - W_3^{\mathrm{h}}(t)) \\ W_4^{\mathrm{h}}(t+1) = W_4^{\mathrm{h}}(t) + g_{3,4}(W_3^{\mathrm{h}}(t) - W_4^{\mathrm{h}}(t)) - \epsilon W_4^{\mathrm{h}}(t) \quad \text{if} \quad |W_3^{\mathrm{h}}(t)| > |W_4^{\mathrm{h}}(t)| \\ W_4^{\mathrm{h}}(t+1) = W_4^{\mathrm{h}}(t) + g_{3,4} \cdot f_{\mathrm{meta}}(W_4^{\mathrm{h}})(W_3^{\mathrm{h}}(t) - W_4^{\mathrm{h}}(t)) - \epsilon W_4^{\mathrm{h}}(t) \quad \text{otherwise} \end{cases} \tag{2.8}$$

The typical evolution of those hidden variables is described in Figure 2.8(a) and is plotted in Figure 2.8(d). The hidden variables evolve over a wide range of timescales through a diffusion chain process. The deepest variable W_4^{h} is a slower and smoother version of W_1^{h}, which is thus relevant for consolidation. These equations are analogous to the ones used in[116], with two additions. The addition of a f_{meta} factor in the last equation consolidates further the slowest variable W_4^{h}. Additionally, we introduce a direct feedback from W_4^{h} to W_1^{h} when W_1^{h} is smaller than W_4^{h} in absolute value or opposite sign, because we found that feedback through the path involving the intermediary variables was too slow to induce proper memory effects. The effect

of the direct feedback is shown in Figure 2.8(d) after 10^5 iterations. It forces a consolidated weight to be unconsolidated on a timescale governed by the ϵ decay term on W_4^{h}. We use $\eta = 5 \cdot 10^{-3}$, $g_{1,2} = 10^{-2}$, $g_{2,3} = 10^{-3}$, $g_{3,4} = 10^{-4}$, $\epsilon = 3 \cdot 10^{-5}$, $\alpha = 5 \cdot 10^{-3}$ and the same f_{meta} as in section 2.2 with $m = 10.0$.

These two additions are necessary as the dynamics of the synapses differ substantially when training binarized neural networks from the situation of [116]. In [116], synaptic updates occur following randomly presented patterns, in an independent and identically distributed fashion. Our continual learning situation is different, because there are two distinct timescales at play: a short timescale constituted by the training iterations within one task, and a long timescale constituted by the different tasks. The slowest variable evolves slowly at the intra-task timescale but rapidly with respect to the long timescale. We introduce f_{meta} to accommodate for this timescale asymmetry. Another difference comes from the sequential synaptic updates, which follow the gradient of a loss function and are therefore highly correlated on shorter time scales. For this reason, the influence of the slowest variable on W_1^{h} through the diffusion chain cannot effectively protect from the correlated gradients of the new task. We thus add a unidirectional feedback connection parameterized by α (Fig. 2.8 (a)) between the slowest variable and W_1^{h} to provide better consolidation. The two modifications of the model allow W_4^{h} to be more stable on the longer timescales of our setup, while allowing to W_1^{h} react on its shorter ones.

Our results, presented in Fig. 2.8 (c), show that binarized neural networks featuring such complex synapses can learn tasks sequentially similarly to our simpler synapse model, and in addition, new tasks can be learned while older tasks are gradually forgotten. The histograms of hidden variables (Fig. 2.8 (e)) also evidence that weights do not accumulate to high values.

We then show in Fig. 2.8 (c) how the model performs when a sequence of 20 tasks is learned. In this situation, the system reaches a 'true' steady state. This is observed by plotting the distributions of the hidden variables in Fig. 2.8 (b), superimposed over the three most recent tasks. We find that the capacity of the model in this true steady state regime is reduced compared to the more transient regime observed during the first ten tasks in Fig. 2.8, as the accuracy of the last learned tasks drops more rapidly in Fig. 2.8(a) than in Fig. 2.8 (c). This result is in accordance with the literature on this type of truly steady-state learning [107, 116, 174].

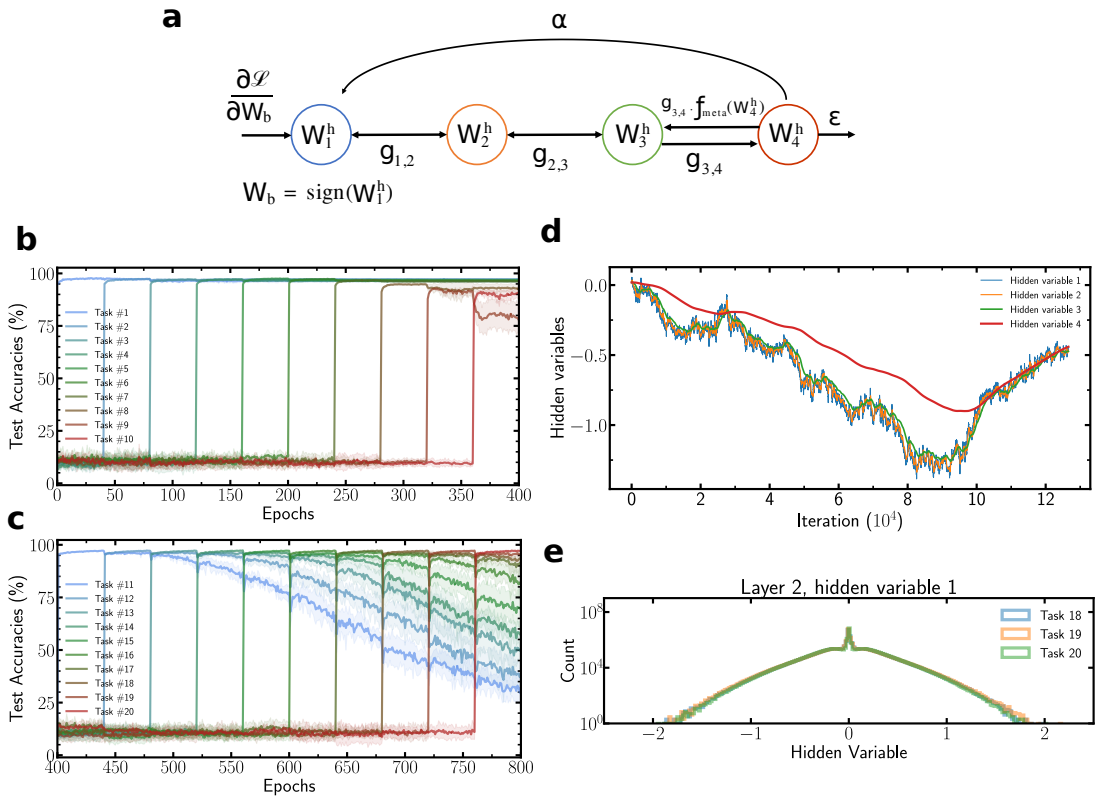

Figure 2.8: **Complex synapse model.** **(a)** Schematic of a more complex synapse model. **(b)** Test accuracies of ten tasks for a metaplastic BNN as introduced in section 2.2 with $m = 1.35$ and two hidden layers of 4,096 units. The tasks are learned until no further learning can be done (task #8 to #10 are not properly learned). **(c)** Same architecture but with our new algorithm with four hidden variables. The model is still able to learn several tasks sequentially but older tasks are gradually forgotten and new tasks can always be learned. The curves are averaged over five runs and shadows stand for one standard deviation. **(d)** Trajectories of the hidden variables as a function of training iterations. The deeper the hidden variable, the slower and smoother it behaves, providing a cleaner signal for consolidation. **(e)** Distribution of the hidden variables after learning 20 tasks. Unlike the distribution presented in section 2.3, hidden weights do not accumulate to ever increasing values and new tasks can always be learned. See Fig. A.6 of Appendix A.10 for more histograms.

2.7 Discussion

Addressing catastrophic forgetting with ideas from both neuroscience and machine learning has led us to find an artificial neural network with richer synapses behaviours that can perform continual learning without requiring an overhead computation of task-related importance factors. The continual learning capability of metaplastic binarized neural networks emerges from its intrinsic design, which is in stark contrast with other consolidation strategies

[104, 125, 126]. The resulting model is more autonomous because the optimized loss function is the same across all tasks. Metaplastic synapses enable binarized neural networks to learn several tasks sequentially similarly to related works, but more importantly, our approach takes the first steps beyond a more fundamental limitation of deep learning, namely the need for a full dataset to learn a given task. A single autonomous model able to learn a task from small amounts of data while still gaining knowledge, approaching to some extent the way the brain acquires new information, paves the way for widespread use of embedded hardware for which it is impossible to store large datasets. Other methods have been introduced to train binarized neural networks such as [175] or [176] and provide valuable insights to understand the specificity of binarized networks with respect to continual learning. Helwegen et al. [175] interpret the hidden weight as inertia, which is coherent with the fact that high inertia might correspond to important weights, while Meng et al. [176] link the hidden weight to the natural parameter of a probability distribution over binarized weights which can be used as a relevant prior to perform continual learning.

A distinctive aspect of continual learning approaches is their behaviour when the neural network reaches its capacity in terms of number of tasks. The behaviour in the case of our approach can be anticipated from the mathematical interpretation in the previous section: when all hidden weights have started to diverge, i.e., are consolidated for a given task, no weights should be able to learn new tasks. The consequence of this situation is well seen in Fig. 2.8 (b): when learning ten permuted MNIST tasks, the last task has reduced accuracy, while the first trained tasks retain their original accuracy. This behaviour fits well with a large section of the literature on continual learning, multitask learning, where the goal is to learn a given number of tasks [121]. Fig. A.2 in Appendix A.4 also highlights the relative definitive nature of synaptic consolidation in our approach. We implemented a variation, where the metaplasticity function reaches a hard zero after a given threshold. We see that the performance on the ten permuted MNIST tasks is only modestly reduced by this change.

This behaviour also differentiates our approach from the brain, where a more natural behaviour for most networks would be to forget the earliest trained tasks, and replace them with the newly trained ones. In recent years, the literature about metaplasticity has aimed at reproducing this behaviour, i.e., a type of 'steady-state' continual learning [116, 164]. This recent literature can therefore provide leads to implement such behaviour in our network. In particular Benna et al. proposed a metaplasticity model where synapses feature a network of different elements, which all evolve at different time scales [116]. This model can feature a sophisticated memory effect, and one work successfully used this type of synapses in the context of an elementary continual reinforcement learning task related to the Cart-Pole problem [164].

We found that directly applying the metaplasticity rule of [116] in our context does not yield proper memory effects. The explanation stems from the specificity of deep networks: in [116], synaptic updates occur following randomly presented patterns, in an independent and identically distributed fashion. In our continual learning situation, sequential synaptic updates are

highly correlated. However, the rule of [116] can still be used as an inspiration to allow steady-state continual learning in our approach. In section 2.6 and the associated Fig. 2.8, we provided a learning rule where synapses also feature a network of elements evolving at different time scale adapted for the training of binarized neural networks, leading to a natural forgetting of tasks trained a long time ago when new tasks are trained. Our adaptation consists in modulating the flow between hidden variables, an idea suggested as a perspective in [164] as a way to bridge the gap between conventional continual learning methods and neuroscience based approaches. We can see in Fig. 2.8 (c) that in this case, when training ten permuted MNIST tasks, the last trained task features the highest accuracy, while the accuracy of the first trained tasks starts to decrease.

This discussion highlights an interplay between the level of continual learning feature and of synaptic complexity. Highly complicated synapses, featuring many equations and hyperparameters, as the ones of [116, 164] or the one that we just introduced, can achieve advanced continual learning behaviours. For an artificial system, the richness of highly complex synapses needs to be counterbalanced with their implementation cost. Biology might have experienced a similar dilemma. Evolution seems to have favored synapses exhibiting highly complex metaplastic behaviours[115], although simpler synapses might have been more efficient to implement, suggesting the high computational benefits of complex synapses.

This discussion is natural for software implementations of metaplasticity, and also exists for hardware. In particular, the fact that metaplastic approaches build on synapses with rich behaviour resonates with the progress of nanotechnologies, which can provide compact and energy-efficient electronic devices able to mimic neuroscience-inspired models, employing 'memristive' technologies [152, 177–179]. Many works in nanotechnologies have shown that a single nanometer-scale device can provide metaplastic behaviour [180–184]. The metaplasticity features of these nanodevices vary greatly depending on their underlying physics and technology, but their complexity is analogous to our proposal here. Typically, metaplasticity occurs by transforming the shape of a conductive filament in a continuous fashion. These changes make the device harder to program, and therefore provide a feature that can be analogous to our continuous metaplasticity function f_{meta}. On the other hand, the complicated version of section 2.6 would be highly challenging to implement with a single nanodevice, based on the current state of nanotechnologies, as these metaplasticity models require many different states with different time dynamics. Our proposal, as other proposals of complex synapses with multiple variables [174] or stochastic behaviours [185], could therefore be an outstanding candidate for nanotechnological implementations, as it provides rich features at the network level, while remaining compatible with the constraints of technology.

Additionally, taking inspiration from the metaplastic behaviour of actual synapses of the brain resulted in a strategy where the consolidation is local in space and time. This makes this approach particularly suited for artificial intelligence dedicated hardware and neuromorphic computing approaches, which can save considerable energy by employing circuit architectures

optimized for the topology of neural network models, and therefore limiting data movements [186]. The fact that our metaplasticity approach is entirely local should be put into perspective into the non-local aspects of the overall learning algorithms. First, all our simulations use batch-normalization, as it is known to efficiently stabilize the training of binarized neural networks [153, 165]. Batch-normalization is not, however, a fundamental element of the scheme. Normalization technique that do not involve batches, such as instance normalization [187], layer normalization [188], or online normalization [189] provide more hardware-friendly alternatives. More profoundly, error backpropagation itself is of course non-local. Currently, multiple efforts aim at developing more local alternatives to backpropagation [66, 69, 71], or at relying on directly bioinspired learning rules [139, 190]. We have seen that alternative approaches of the literature to overcome catastrophic forgetting typically rely on the use of additional terms in the loss, are therefore strongly tied to the use of error backpropagation. On the other hand, as our metaplasticity approach is entirely synaptic-centric, it is largely agnostic to the learning rule, and should be adaptable to all these emerging learning approaches. This discussion also evidences the benefit of taking inspiration from biology with regards to purely mathematically-motivated approaches: they tend to be naturally compatible with the constraints of hardware developments and can be amenable for the development of energy-efficient artificial intelligence.

In conclusion, we have shown that the hidden weights involved in the training of binarized neural networks are excellent candidates as metaplastic variables that can be efficiently leveraged for continual learning. We have implemented long term memory into binarized neural networks by modifying the hidden weight update of synapses. Our work highlights that binarized neural networks can be more than a low precision version of deep neural networks, as well as the potential benefits of the synergy between neurosciences and machine learning research, which for instance aims to convey long term memory to artificial neural networks. We have also mathematically justified our technique in a tractable quadratic binary problem. Our method allows for online synaptic consolidation directly from model behaviour, which is important for neuromorphic dedicated hardware, and is also useful for a variety of settings subject to catastrophic forgetting.

2.8 Methods

Metaplasticity-inspired training of binarized neural networks. The binarized neural networks studied in this Chapter are designed and trained following the principles introduced in [191] - specific implementation details are provided in Appendix A.2. These networks consist of binarized layers where both weight values and neuron activations assume binary values meaning $\{+1,-1\}$. Binarized neural networks can achieve high accuracy on vision tasks [165, 169], provided that the number of neurons is increased with regards to real neural networks. Binarized neural networks are especially promising for AI hardware because unlike conventional

deep networks which rely on costly matrix-vector multiplications, these operations for binarized neural networks can be done in hardware with XNOR logic gates and pop-count operations, reducing the power consumption by several orders of magnitude [154].

In this Chapter, we propose an adaptation of the conventional binarized neural network training technique to provide binarized neural networks with metaplastic synapses. We introduce the function $f_{\text{meta}} : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$ to provide an asymmetry, at equivalent gradient value and for a given weight, between updates towards zero hidden value and away from zero. Alg. 1 describes our optimization update rule and the unmodified version of the update rule is recovered when $m = 0.0$ due to condition (2.9) satisfied by f_{meta}. f_{meta} is defined such that:

$$\forall x \in \mathbb{R},\ f_{\text{meta}}(0, x) = 1, \tag{2.9}$$

$$\forall m \in \mathbb{R}^+,\ f_{\text{meta}}(m, 0) = 1, \tag{2.10}$$

$$\forall m \in \mathbb{R}^+,\ \partial_x f_{\text{meta}}(m, 0) = 0, \tag{2.11}$$

$$\forall m \in \mathbb{R}^+,\ \lim_{|x| \to +\infty} f_{\text{meta}}(m, x) = 0. \tag{2.12}$$

Conditions (2.10) and (2.11) ensure that near-zero real values, the weights are free to switch in order to learn. Condition (2.12) ensures that the farther from zero a real value is, the more difficult it is to make the corresponding weight switch back. In all the experiments of this paper, we use :

$$f_{\text{meta}}(m, x) = 1 - \tanh^2(m \cdot x). \tag{2.13}$$

The parameter m controls how fast binary weights are consolidated (Fig. 2.1 (c). The specific choice of f_{meta} is made to have a variety of plasticity over large ranges of time steps (iteration steps) with an exponential dependence as in [107]. Specific values of the hyperparameters can be found in Appendix A.2.

Multitask training experiments. A permuted version of the MNIST dataset consists of a fixed spatial permutation of pixels applied to each example of the dataset. We also train a full precision (32-bits floating point) version of our network with the same architecture for comparison, but with tanh activation function instead of sign. The learned parameters in batch normalization are not binary and therefore cannot be consolidated by our metaplastic strategy. Therefore, in our experiments, the binarized and full precision neural networks have task-specific batch normalization parameters in order to isolate the effect of weight consolidation on previous tasks test accuracies.

For the control, elastic weight consolidation is applied to binarized neural networks by consolidating the binary weights (and not the hidden weights as the response of the network is determined by the binary weights): both the surrogate loss term, and the Fisher information estimates are computed using the binary weight values. The EWC regularization strength pa-

rameter is $\lambda_{\text{EWC}} = 5 \cdot 10^3$. The random consolidation presented in Tab. 2.1 consists in computing the same importance factors as elastic weight consolidation but then randomly shuffling the importance factors of the synapses.

Stream learning experiments. For Fashion-MNIST experiments, we use a metaplastic binarized neural network of two 1,024 units hidden layers. The dataset is split into 60 subsets of 1,000 examples each, and each subset is learned for 20 epochs. (All classes are represented in each subset.)

For CIFAR-10 experiments, we use a binary version of VGG-7 similarly to [191], with six convolution layers of 128-128-256-256-512-512 filters and kernel sizes of 3. Dropout with probability 0.5 is used in the last two fully connected layers of 2,048 units. Data augmentation is used within each subset with random crop and random rotation.

Sign Switch in a binarized neural network. Two major differences between the quadratic binary task and the binarized neural network are the dependence on the training data and the relative contribution of each parameter which is lower in the case of the BNN than in the quadratic binary task. The procedure for generating Fig.2.7 (b,c,d) have to be adapted accordingly. Bins of increasing normalised hidden weights are created, but instead of computing the cost variation for a single sign switch, a fixed amount of weights are switched within each bin so as to increase the contribution of the sign switch on the cost variation. The resulting cost variation is then normalised with respect to the number of switched weights. An average is done over several realizations of the hidden weights to be switched. Given the different sizes of the three layers, the amounts of switched weights per bins for each layer are respectively 1,000, 2,000, and 100.

Positive symmetric definite matrix generation. To generate random positive symmetric definite matrices we first generate the diagonal matrix of eigenvalues $\mathbf{D} = \text{diag}(\lambda_1, ..., \lambda_d)$ with a uniform or normal distribution of mean μ and variance σ and ensure that all eigen values are positive. We then use the subgroup algorithm described in [192] to generate a random rotation $\mathbf{R}$ in dimension d. This is done by first generating a random rotation $\mathbf{R}_2$ in 2D and iteratively increasing the dimension by sampling a random unitary vector $\mathbf{v}$, then computing $\mathbf{x} = (\mathbf{e}_1 - \mathbf{v})/\|\mathbf{e}_1 - \mathbf{v}\|$ with $\mathbf{e}_1 = (1, 0, ..., 0)^T$, and finally computing $\mathbf{R}_{n+1} = (\mathbf{I} - 2\mathbf{x}^T\mathbf{x}) \cdot \hat{\mathbf{R}}_n$, where $\hat{\mathbf{R}}_n$ is a $n+1 \times n+1$ matrix where $\hat{\mathbf{R}}_{n,0,0} = 1$, $\hat{\mathbf{R}}_{n,1:,1:} = \mathbf{R}_n$ and $\hat{\mathbf{R}}_{n,0,j} = \hat{\mathbf{R}}_{n,i,0} = 0$. We then compute $\mathbf{H} = \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R}$.

Chapter 3

Scaling Equilibrium Propagation to Deep ConvNets by Drastically Reducing its Gradient Estimator Bias

> "Rather than search for machine-independent algorithms, one should search for just the opposite - dynamical algorithms that can fully exploit the collective behavior of physical hardware."
>
> Fernando J. PINEDA [193]

WHILE Chapter 2 presented a local consolidation rule for continual learning in binarized neural networks, the learning algorithm itself still relied on the highly biologically implausible back-propagation presented in Section 1.2 of Chapter 1. In this Chapter, we make a contribution to Equilibrium Propagation, introduced in Section 1.3.2.1, which provides a way to perform credit assignment locally in space. In practice, however, the standard implementation of Equilibrium Propagation does not scale to visual tasks harder than MNIST. In this Chapter, we show that a bias in the gradient estimate of equilibrium propagation, inherent in the use of finite nudging, is responsible for this phenomenon and that cancelling it allows training deep convolutional neural networks. We show that this bias can be greatly reduced by using symmetric nudging (a positive nudging and a negative one). We also generalize Equilibrium Propagation to the case of cross-entropy loss (by opposition to squared error). As a result of these advances, we are able to achieve a test error of 11.7% on CIFAR-10, which approaches the one achieved by BPTT and provides a major improvement with respect to the standard Equilibrium Propagation that gives 86% test error. We also apply these techniques to train an architecture with unidirectional forward and backward connections, yielding a 13.2% test error. These results highlight equilibrium propagation as a compelling biologically-plausible approach to compute error gradients in deep neuromorphic systems. The networks studied here are not quantized, but can be extended to quantized networks [194]. This Chapter is adapted from an article published in Frontiers in Neuroscience [2], done in collaboration with the Mila.

3.1 Introduction

How synapses in hierarchical neural circuits are adjusted throughout learning a task remains a challenging question called the credit assignment problem [66]. Equilibrium Propagation (EP) [71] provides a biologically plausible solution to this problem in artificial neural networks (see section 1.3.2.1). EP is an algorithm for convergent recurrent neural networks (RNNs) which, by definition, are given a static input and whose recurrent dynamics converge to a steady state corresponding to the prediction of the network. EP proceeds in two phases, bringing the network to a first steady state, then nudging the output layer of the network towards a ground-truth target until reaching a second steady state. During the second phase of EP, the perturbation originating from the output layer propagates forward in time to upstream layers, creating local error signals that match exactly those that are computed by Backpropagation Through Time (BPTT), the canonical approach for training RNNs [80]. We refer to Scellier et al. [195] for a comparison between EP and recurrent backpropagation [196, 197]. Owing to this strong theoretical guarantee, EP can provide leads for understanding biological learning [92]. Moreover, the spatial locality of the learning rule prescribed by EP and the possibility to make it also local in time [198] is highly attractive for designing energy-efficient neuromorphic hardware implementations of gradient-based learning algorithms [157, 198–202].

To meet these expectations, however, EP should be able to scale to complex tasks. Until

now, works on EP [71, 80, 198, 203, 204] limited their experiments to the MNIST classification task and shallow network architectures. Despite the theoretical guarantees of EP, the literature suggests that no implementation of EP has thus far succeeded to match the performance of standard deep learning approaches to train deep networks on hard visual tasks. This problem is even more challenging when using a more bio-plausible topology where the synaptic connections of the network are unidirectional: existing proposals of EP in this situation [81, 198] lead to a degradation of accuracy on MNIST compared to standard EP. In this Chapter, we show that performing the second phase of EP with nudging strength of constant sign induces a systematic first order bias in the EP gradient estimate which, once cancelled, unlocks the training of deep convolutional neural networks (ConvNets), with bidirectional or unidirectional connections and with performance closely matching that of BPTT on CIFAR-10. We also propose to implement the neural network predictor as an external softmax readout. This modification preserves the local nature of EP and allows us to use the cross-entropy loss, contrary to previous approaches using the squared error loss and where the predictor takes part in the free dynamics of the system.

Other biologically plausible alternatives to backpropagation (BP) have attempted to scale to hard vision tasks. Bartunov et al. [84] investigated the use of feedback alignment [82] and variants of target propagation [90, 205] on CIFAR-10 and ImageNet, showing that they perform significantly worse than backpropagation. When the alignment between forward and backward weights is enhanced with extra mechanisms [89], feedback alignment performs better on ImageNet than sign-symmetry [87], where feedback weights are taken to be the sign of the forward weights, and almost as well as backpropagation. However, in feedback alignment and target propagation, the error feedback does not affect the forward neural activity and is instead routed through a distinct backward pathway, an issue that EP avoids. Payeur et al. [70] proposed a burst-dependent learning rule that also addresses this problem and whose rate-based equivalent, relying on the use of specialized synapses and complex network topology, has been benchmarked against CIFAR-10 and ImageNet. Related works on implicit models [206] have shown that training deep networks can be framed as solving a fixed point (steady state) equation, leading to an analytical backward pass. This framework was shown to solve challenging vision tasks [207]. While the use of a steady state is common with EP, the process to reach the steady state as well as the learning rule are different. In comparison with these approaches, EP offers a minimalistic circuit requirement to handle both inference and gradient computation, which makes it an outstanding candidate for energy-efficient neuromorphic learning hardware design.

More specifically, the contributions of this Chapter are the following:

- We introduce a new method to estimate the gradient of the loss based on three steady states instead of two (section 3.3.1). This approach enables us to achieve 11.68% test error on CIFAR-10, with 0.6 % performance degradation only with respect to BPTT. Conversely, we show that using a nudging strength of constant sign yields 86.64% test error.

- We propose to implement the output layer of the neural network as a softmax readout, which subsequently allows us to optimize the cross-entropy loss function with EP. This method improves the classification performance on CIFAR-10 with respect to the use of the squared error loss and is also closer to the one achieved with BPTT (section 3.3.2).

- Finally, based on ideas of Scellier et al. [81] and Kolen and Pollack [208], we adapt the learning rule of EP for architectures with distinct (unidirectional) forward and backward connections, yielding only 1.5% performance degradation on CIFAR-10 compared to bidirectional connections (section 3.2.4).

3.2 Background

3.2.1 Convergent RNNs With Static Input

We consider the setting of supervised learning where we are given an input x (e.g., an image) and want to predict a target y (e.g., the class label of that image). To solve this type of task, Equilibrium Propagation (EP) relies on convergent RNNs, where the input of the RNN at each time step is static and equal to x, and the state s of the neural network converges to a steady-state $s_\star$. EP applies to a wide class of convergent RNNs, where the transition function derives from a scalar primitive[1] Φ [80]. In this situation, the dynamics of a network with parameters θ, usually synaptic weights, is given by

$$s_{t+1} = \frac{\partial \Phi}{\partial s}(x, s_t, \theta), \tag{3.1}$$

where s_t is the state of the RNN at time step t. After the dynamics have converged at some time step T, the network reaches the steady state $s_T = s_\star$, which, by definition, satisfies:

$$s_\star = \frac{\partial \Phi}{\partial s}(x, s_\star, \theta). \tag{3.2}$$

Formally, the goal of learning is to optimize θ to minimize the loss at the steady state $\mathscr{L}^\star = \ell(s_\star, y)$, where ℓ is a differentiable cost function. While we did not investigate theoretical guarantees ensuring the convergence of the dynamics, we refer the reader to Scarselli et al. [209] for sufficient conditions on the transition function to ensure convergence. In practice, we always observe the convergence to a steady-state.

3.2.2 Training Procedures For Convergent RNNs

[1] In the original version of EP for real-time dynamical systems [71], the dynamics derive from an energy function E, which plays a similar role to the primitive function Φ in the discrete-time setting studied here.

3.2.2.1 Equilibrium Propagation (EP)

Scellier and Bengio [71] introduced Equilibrium Propagation in the case of real time dynamics, as presented in section 1.3.2.1. Subsequent work adapted it to discrete-time dynamics, bringing it closer to conventional deep learning [80]. EP consists of two distinct phases. During the first ('free') phase, the RNN evolves according to Eq. (3.1) for T time steps to ensure convergence to a first steady state $s_\star$. During the second ('nudged') phase of EP, a nudging term $-\beta\frac{\partial \ell}{\partial s}$ is added to the dynamics, with β a small scaling factor. Denoting s_0^β, s_1^β, s_2^β... the states during the second phase, the dynamics reads

$$s_0^\beta = s_\star, \quad \text{and} \quad \forall\, t>0, \quad s_{t+1}^\beta = \frac{\partial \Phi}{\partial s}(x, s_t^\beta, \theta) - \beta \frac{\partial \ell}{\partial s}(s_t^\beta, y). \tag{3.3}$$

The RNN then reaches a new steady state denoted $s_\star^\beta$. Scellier and Bengio [71] proposed the EP learning rule, denoting η the learning rate applied:

$$\Delta\theta = \eta \widehat{\nabla}^{\text{EP}}(\beta), \qquad \text{where} \qquad \widehat{\nabla}^{\text{EP}}(\beta) \triangleq \frac{1}{\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s_\star^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s_\star, \theta)\right). \tag{3.4}$$

They proved that this learning rule performs stochastic gradient descent in the limit $\beta \to 0$:

$$\lim_{\beta \to 0} \widehat{\nabla}^{\text{EP}}(\beta) = -\frac{\partial \mathscr{L}^\star}{\partial \theta}. \tag{3.5}$$

3.2.2.2 Equivalence of Equilibrium Propagation and Backpropagation Through Time (BPTT)

The convergent RNNs considered by EP can also be trained by Backpropagation Through Time (BPTT). At each BPTT training iteration, the first phase is performed for T time steps until the network reaches the steady state $s_T = s_\star$. The loss at the final time step is computed and the gradients are subsequently backpropagated through the computational graph of the first phase, backward in time.

Let us denote $\nabla^{\text{BPTT}}(t)$ the gradient computed by BPTT truncated to the last t time steps $(T-t, \ldots, T)$, which we define formally in Appendix B.1.

A theorem derived by Ernoutl et al. [80], inspired from Scellier et al. [195], shows that, provided convergence in the first phase has been reached after $T-K$ time steps (i.e., $s_{T-K} = s_{T-K+1} = \ldots = s_T = s_\star$), the gradients of EP match those computed by BPTT in the limit $\beta \to 0$, in the first K time steps of the second phase for fully connected and convolutional architectures including pooling operations:

$$\forall\, t = 1, 2, \ldots, K, \qquad \widehat{\nabla}^{\text{EP}}(\beta, t) \triangleq \frac{1}{\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s_t^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s_\star, \theta)\right) \xrightarrow[\beta \to 0]{} \nabla^{\text{BPTT}}(t). \tag{3.6}$$

3.2.3 Convolutional Architectures for Convergent RNNs

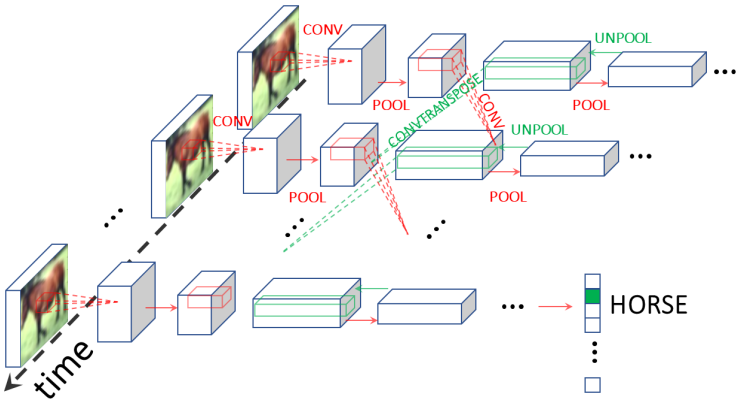

Figure 3.1: Schematic of the architecture used. We use Equilibrium Propagation (EP) to train a recurrent ConvNet receiving a static input. Red (resp. green) arrows depict forward (resp. backward) operations, with convolutions and transpose convolutions happening through time. At the final time step, the class prediction is carried out. The use of RNNs is inherent in the credit assignment of EP which uses of the temporal variations of the system as error signals for the gradient computation.

A convolutional architecture for convergent RNNs with static input was introduced by Ernoult et al. [80] and successfully trained with EP on the MNIST dataset. In this architecture, presented in Fig. 3.1, we define N^{conv} and N^{fc} the number of convolutional and fully connected layers respectively, and $N^{\text{tot}} \overset{\Delta}{=} N^{\text{conv}} + N^{\text{fc}}$. w_{n+1} denotes the weights connecting s^n to s^{n+1}, with $s_0 = x$. To simplify notations, we use distinct operators to differentiate whether w_n is a convolutional layer or a fully connected layer: respectively $\star$ for convolutions and $\cdot$ for linear layers. The primitive function can therefore be defined as:

$$\Phi(x,\{s^n\}) = \sum_{n=0}^{N^{\text{conv}}-1} s^{n+1} \bullet \mathscr{P}\left(w_{n+1} \star s^n\right) + \sum_{n=N^{\text{conv}}}^{N^{\text{tot}}-1} s^{n+1\top} \cdot w_{n+1} \cdot s^n, \tag{3.7}$$

where $\bullet$ is the Euclidean scalar product generalized to pairs of tensors with same arbitrary dimension, and $\mathscr{P}$ is a pooling operation. Combining Eqs. (3.1) and (3.7), and restricting the space of the state variables to $[0,1]$, yield the dynamics:

$$\begin{cases} s^n_{t+1} &= \sigma\left(\mathscr{P}\left(w_n \star s^{n-1}_t\right) + \tilde{w}_{n+1} \star \mathscr{P}^{-1}\left(s^{n+1}_t\right)\right), \qquad 1 \le n \le N^{\text{conv}} \\ s^n_{t+1} &= \sigma\left(w_n \cdot s^{n-1}_t + w^\top_{n+1} \cdot s^{n+1}_t\right), \qquad N^{\text{conv}} < n < N^{\text{tot}} \end{cases} \tag{3.8}$$

where σ is an activation function bounded between 0 and 1. Transpose convolution and inverse pooling are respectively defined through the convolution by the flipped kernel $\tilde{w}$ and $\mathscr{P}^{-1}$. Plugging Eq. (3.7) into Eq. (3.4) yields the local learning rule mentioned in Section 1.3.2.1 $\Delta\theta_{ij} = \eta(s^\beta_{i,\star} s^\beta_{j,\star} - s_{i,\star} s_{j,\star})/\beta$ for a parameter θ_{ij} linking neurons i and j. Appendix B.4 provides the

implementation details of this model.

3.2.4 Equilibrium Propagation with unidirectional synaptic connections

We have seen that in the standard formulation of EP, the dynamics of the neural network derive from a function Φ (Eq. (3.1)) called the primitive function. This formulation implies the existence of bidirectional synaptic connections between neurons. For better biological plausibility, a more general formulation of EP circumvents this requirement and allows training networks with distinct (unidirectional) forward and backward connections [81, 198]. This feature is also desirable for hardware implementations of EP. Although some analog implementations of EP naturally lead to symmetric weights [157], neural networks with unidirectional weights are in general easier to implement in neuromorphic hardware.

In this setting, the dynamics of Eq. (3.1) is changed into the more general form:

$$s_{t+1} = F(x, s_t, \theta), \tag{3.9}$$

and the conventionally proposed learning rule reads:

$$\Delta\theta = \eta \widehat{\nabla}^{\text{VF}}(\beta), \qquad \text{where} \qquad \widehat{\nabla}^{\text{VF}}(\beta) \triangleq \frac{1}{\beta} \frac{\partial F}{\partial \theta}(x, s_\star, \theta)^\top \cdot \left(s_\star^\beta - s_\star\right), \tag{3.10}$$

where VF stands for Vector Field [81]. If the transition function F derives from a primitive function Φ (i.e., if $F = \frac{\partial \Phi}{\partial s}$), then $\widehat{\nabla}^{\text{VF}}(\beta)$ is equal to $\widehat{\nabla}^{\text{EP}}(\beta)$ in the limit $\beta \to 0$ (i.e. $\lim_{\beta \to 0} \widehat{\nabla}^{\text{VF}}(\beta) = \lim_{\beta \to 0} \widehat{\nabla}^{\text{EP}}(\beta)$).

3.3 Improving EP Training

We have seen in Eq. (3.6) that the temporal variations of the network over the second phase of EP exactly compute BPTT gradients in the limit $\beta \to 0$. This result appears to underpin the use of two phases as a fundamental element of EP, but is it really the case? In this section, we revisit EP as a gradient estimation procedure and propose an implementation in three phases instead of two. Moreover, we show how to optimize the cross-entropy loss function with EP. Combining these two new techniques enabled us to achieve the best performance on CIFAR-10 by EP, on architectures with bidirectional and unidirectional forward and backward connections (section 3.4).

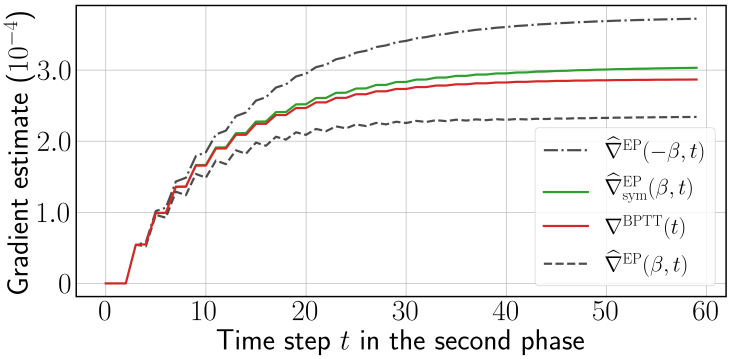

Figure 3.2: One-sided EP gradient estimate for opposite values of $\beta = 0.1$ (black dashed curves), symmetric EP gradient estimate (green curve) and reference gradients computed by BPTT (red curve) computed over the second phase, for a single weight chosen at random. The time step t is defined for BPTT and EP according to Eq. (3.6). More instances can be found in Appendix B.6.

3.3.1 Reducing bias and variance in the gradient estimate of the loss function

In the foundational work on EP, Scellier and Bengio [71] demonstrate that:

$$\left.\frac{d}{d\beta}\right|_{\beta=0} \frac{\partial \Phi}{\partial \theta}(x, s^{\beta}_{\star}, \theta) = -\frac{\partial \mathscr{L}^{\star}}{\partial \theta}. \tag{3.11}$$

The traditional implementation of EP evaluates the left-hand side of Eq. (3.11) using the estimate $\widehat{\nabla}^{\mathrm{EP}}(\beta)$ with two points $\beta = 0$ and $\beta > 0$, thereby calling for the need of two phases – the free phase and the nudged phase. However, the use of $\beta > 0$ in practice induces a systematic first order bias in the gradient estimation provided by EP. In order to eliminate this bias, we propose to perform a third phase with $-\beta$ as the nudging factor, keeping the first and second phases unchanged. We then estimate the gradient of the loss using the following symmetric difference estimate:

$$\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta) \triangleq \frac{1}{2\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s^{\beta}_{\star}, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s^{-\beta}_{\star}, \theta)\right). \tag{3.12}$$

Indeed, under mild assumptions on the function $\beta \mapsto \frac{\partial \Phi}{\partial \theta}(x, s^{\beta}_{\star}, \theta)$, we can show that, as $\beta \to 0$:

$$\frac{\widehat{\nabla}^{\mathrm{EP}}(\beta) + \widehat{\nabla}^{\mathrm{EP}}(-\beta)}{2} = -\frac{\partial \mathscr{L}^{\star}}{\partial \theta} + O(\beta^2), \tag{3.13}$$

$$\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta) = -\frac{\partial \mathscr{L}^{\star}}{\partial \theta} + O(\beta^2). \tag{3.14}$$

This result is proved in Lemma 3 of Appendix B.2. Eq. (3.13) shows that the estimate $\widehat{\nabla}^{\mathrm{EP}}(\beta)$ possesses a first-order error term in β which the symmetric estimate $\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta)$ eliminates (Eq. (3.14)).

Note that the first-order term of $\widehat{\nabla}^{\text{EP}}(\beta)$ could also be cancelled out on average by choosing the sign of β at random with even probability (so that $\mathbb{E}(\beta) = 0$, see Alg. 4 of Appendix B.3.1). Although not explicitly stated in this purpose, the use of such randomization has been reported in some earlier publications on the MNIST task [71, 198]. However, in this work, we show that this method exhibits high variance in the training procedure.

We call $\widehat{\nabla}^{\text{EP}}(\beta)$ and $\widehat{\nabla}^{\text{EP}}_{\text{sym}}(\beta)$ the one-sided and symmetric EP gradient estimates respectively. The qualitative difference between these estimates is depicted in Fig. 3.2, and the full training procedure is depicted in Alg. 5 of Appendix B.3.2.

Finally, this technique can also be applied to the Vector Field setting introduced in section 3.2.4 and we denote $\widehat{\nabla}^{\text{VF}}_{\text{sym}}(\beta)$ the resulting symmetric estimate — see Appendix B.4.3 for details.

3.3.2 Changing the loss function

We also introduce a novel architecture to optimize the cross-entropy loss with EP, narrowing the gap with conventional deep learning architectures for classification tasks. In the next paragraph, we denote $\hat{y}$ the set of neurons that carries out the prediction of the neural network.

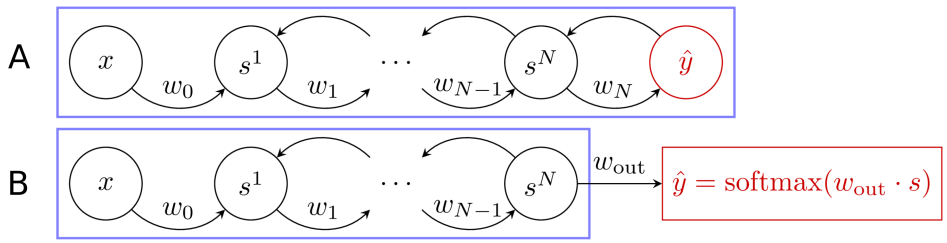

Figure 3.3: Free dynamics of the architectures used for the two loss functions where the blue frame delimits the system. **A Squared Error loss function.** The usual setting where the predictor $\hat{y}$ (in red) takes part in the free dynamics of the neural network through bidirectional synaptic connections. **B Cross-Entropy loss function.** The new approach proposed in this Chapter where the predictor $\hat{y}$ (also in red) is no longer involved in the system free dynamics and is implemented as a softmax readout.

3.3.2.1 Squared Error loss function.

Previous implementations of EP used the squared error loss. Using this loss function for EP is natural, as in this setting, the output $\hat{y}$ is viewed as a part of s (the state variable of the network), which can influence the state of the network through bidirectional synaptic connections (see Fig. 3.3). Moreover, the nudging term in this case can be physically interpreted since it reads as an elastic force. The state of the network is of the form $s = (s^1, \dots, s^N, \hat{y})$ where $h = (s^1, \dots, s^N)$ represent the 'hidden layers', and the corresponding cost function is

$$\ell(\hat{y}, y) = \frac{1}{2} \left\| \hat{y} - y \right\|^2 . \tag{3.15}$$

The second phase dynamics of the hidden state and output layer given by Eq. (3.3) read, in this context:

$$h_{t+1}^{\beta} = \frac{\partial \Phi}{\partial h}(x, h_t^{\beta}, \widehat{y}_t^{\beta}, \theta), \qquad \widehat{y}_{t+1}^{\beta} = \frac{\partial \Phi}{\partial \widehat{y}}(x, h_t^{\beta}, \widehat{y}_t^{\beta}, \theta) + \beta\,(y - \widehat{y}_t^{\beta}). \tag{3.16}$$

3.3.2.2 Softmax readout, Cross-Entropy loss function.

In this paper, we propose an alternative approach, where the output $\widehat{y}$ is not a part of the state variable s but is instead implemented as a read-out (see Fig. 3.3), which is a function of s and of a weight matrix w_{out} of size $\dim(y) \times \dim(s)$. In practice, w_{out} reads out the last convolutional layer. At each time step t we define:

$$\widehat{y}_t = \text{softmax}(w_{\text{out}} \cdot s_t). \tag{3.17}$$

The cross-entropy cost function associated with the softmax readout is then:

$$\ell(s, y, w_{\text{out}}) = -\sum_{c=1}^{C} y_c \log(\text{softmax}_c(w_{\text{out}} \cdot s)). \tag{3.18}$$

Using $\frac{\partial \ell}{\partial s}(s, y, w_{\text{out}}) = w_{\text{out}}^{\top} \cdot \big(\text{softmax}(w_{\text{out}} \cdot s) - y\big)$, the second phase dynamics given by Eq. (3.3) read in this context:

$$s_{t+1}^{\beta} = \frac{\partial \Phi}{\partial s}(x, s_t^{\beta}, \theta) + \beta\, w_{\text{out}}^{\top} \cdot \Big(y - \widehat{y}_t^{\beta}\Big). \tag{3.19}$$

Note here that the loss $\mathscr{L}^{\star} = \ell(s_{\star}, y, w_{\text{out}})$ also depends on the parameter w_{out}. The Appendix B.4.2.2 provides the learning rule applied to w_{out}.

3.3.3 Changing the learning rule of EP with unidirectional synaptic connections

In the case of architectures with unidirectional connections, applying the traditional EP learning rule directly, as given by Eq. (3.10), prescribes different forward and backward weights updates, resulting in significantly different forward and backward weights throughout learning. However, the theoretical equivalence between EP and BPTT only holds for bidirectional connections. Until now, training experiments of unidirectional weights EP have performed worse than bidirectional weights EP [198]. In this work, therefore, we tailor a new learning rule for unidirectional weights, described in detail Appendix B.4.3, where the forward and backward weights undergo the same weight updates, incorporating an equal leakage term. This way, forward and backward weights, although they are independently initialized, naturally converge to identical values throughout the learning process. A similar methodology, adapted from Kolen and Pollack [208], has been shown to improve the performance of Feedback Alignment in Deep ConvNets [89].

Assuming general dynamics of the form of Eq. (3.9), we distinguish forward connections θ_{f} from backward connections θ_{b} so that $\theta = \{\theta_{\mathrm{f}}, \theta_{\mathrm{b}}\}$, with θ_{f} and θ_{b} having same dimension. Assuming a first phase, a second phase with $\beta > 0$ and a third phase with $-\beta$, we define:

$$\forall \mathrm{i} \in \{\mathrm{f}, \mathrm{b}\}, \qquad \overline{\nabla^{\mathrm{VF}}_{\theta_{\mathrm{i}}}}(\beta) = \frac{1}{2\beta}\left(\frac{\partial F}{\partial \theta_{\mathrm{i}}}^{\top}(x, s^{\beta}_{\star}, \theta) \cdot s^{\beta}_{\star} - \frac{\partial F}{\partial \theta_{\mathrm{i}}}^{\top}(x, s^{-\beta}_{\star}, \theta) \cdot s^{-\beta}_{\star}\right) \tag{3.20}$$

and we propose the following update rules:

$$\begin{cases} \Delta\theta_{\mathrm{f}} = \eta\left(\widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta) - \lambda\theta_{\mathrm{f}}\right) \\ \Delta\theta_{\mathrm{b}} = \eta\left(\widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta) - \lambda\theta_{\mathrm{b}}\right) \end{cases}, \quad \text{with} \quad \widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta) = \frac{1}{2}(\overline{\nabla^{\mathrm{VF}}_{\theta_{\mathrm{f}}}}(\beta) + \overline{\nabla^{\mathrm{VF}}_{\theta_{\mathrm{b}}}}(\beta)) \tag{3.21}$$

where η is the learning rate and λ a leakage parameter. The estimate $\widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta)$ can be thought of a generalization of Eq. (3.12), as highlighted in Appendix B.4.3 with an explicit application of Eq. (3.21) to a ConvNet. In the case of a fully connected layer, both terms in the sum in the right hand side of Eq. 3.21 are equal: $\partial F/\partial\theta_{\mathrm{i}}$ only depends on the neuron activations and not on θ_{i}, in the same way, as seen at the end of section 2.3, that Eq. (8) yields a fully local learning rule. The case of convolutional layers is a little more subtle, due to presence of the maximum pooling operations. The forward weights are involved in a pooling operation while the backward weights are involved in an unpooling operation. However, for the parameter update to be the same, the pooling and unpooling operations need to share information regarding the indices of maxima. Therefore, there is indeed a need for information transfer between the backward forward parameters, but this exchange is limited to the index of the maximum identified in the maximum pooling operation (this can be seen from Eq. B.24)

3.4 Results

In this section, we implement EP with the modifications described in section 3.3 and successfully train deep ConvNets on the CIFAR-10 vision task [210]. The convolutional architecture used consists of four 3×3 convolutional layers of respective feature maps 128 - 256 - 512 - 512. We use a stride of one for each convolutional layer, and zero-padding of one for each layer except for the last layer. Each layer is followed by a 2×2 Max Pooling operation with a stride of two. The resulting flattened feature vector is of size 512. The weights are initialized using the default initialization of PyTorch, which is the uniform Kaiming initialization of [211]. The data is normalized and augmented with random horizontal flips and random crops. The training is performed with stochastic gradient descent with momentum and weight decay. We use the learning rate scheduler introduced by [212] to speed up convergence.

The hyper-parameters are reported in Tab. 3.2. All experiments are performed using PyTorch 1.4.0. [35]. The simulations were carried across several servers consisting of 14 Nvidia GeForce RTX 2080 TI GPUs in total. Each run was performed on a single GPU for an average

run time of 2 days.

3.4.1 ConvNets with bidirectional connections

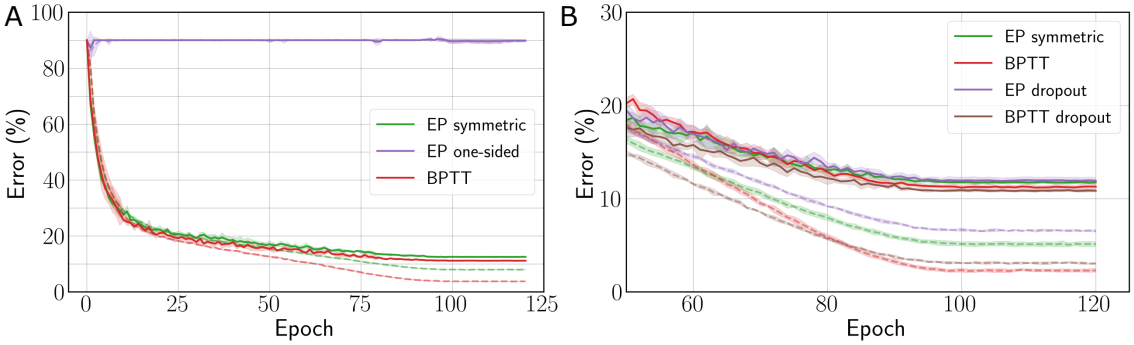

Figure 3.4: **A** Train (dashed) and test (solid) errors on CIFAR-10 with the Squared Error loss function. **B** Train (dashed) and test (solid) errors on CIFAR-10 with the Cross-Entropy loss function. The curves are averaged over 5 runs and shadows stand for $\pm 1 \times$ standard deviation. The change in error rate around epochs 85-90 is due to the end of the learning rate scheduler decay phase (Cosine annealing).

Table 3.1: Performance comparison on CIFAR-10 between BPTT and EP with several gradient estimation schemes. Note that the different gradient estimates only apply to EP. We indicate over five trials the mean and standard deviation in parenthesis for the test error, and the mean train error.

Loss Function	EP Gradient Estimate	EP Error (%)		BPTT Error (%)	
		Test	Train	Test	Train
Squared Error	2-Phase / $\hat{\nabla}^{\text{EP}}$	86.64 (5.82)	84.90	11.10 (0.21)	3.69
	Random Sign	21.55 (20.00)	20.01		
	3-Phase / $\hat{\nabla}^{\text{EP}}_{\text{sym}}$	12.45 (0.18)	7.83		
Cross-Ent.	3-Phase / $\hat{\nabla}^{\text{EP}}_{\text{sym}}$	**11.68** (**0.17**)	**4.98**	11.12 (0.21)	2.19
Cross-Ent. (Dropout)	3-Phase / $\hat{\nabla}^{\text{EP}}_{\text{sym}}$	11.87 (0.29)	6.46	10.72 (0.06)	2.99
Cross-Ent.	3-Phase / $\hat{\nabla}^{\text{VF}}_{\text{sym}}$	75.47 (4.72)	78.04	9.46 (0.17)	0.80
	3-Phase / $\hat{\nabla}^{\text{KP-VF}}_{\text{sym}}$	**13.15** (**0.49**)	8.87		

We first consider the bidirectional weight setting of section 3.2.3. In Table 3.1, we compare the performance achieved by the ConvNet for each EP gradient estimate introduced in section 3.3.1 with the performance achieved by BPTT.

The one-sided gradient estimate leads to unstable training behavior where the network is unable to fit the data, as shown by the purple curve of Fig. 3.4A, with 86.64% test error on CIFAR-10. When the bias in the gradient estimate is averaged out by choosing at random the sign of β during the second phase, the average test error over five runs goes down to 21.55% (see Table 3.1). However, one run among the five yielded instability similar to the one-sided estimate, whereas the four remaining runs lead to 12.61% test error and 8.64% train error. This method

for estimating the loss gradient thus presents high variance — further experiments shown in Appendix B.4.4 confirm this trend.

Conversely, the three-phase symmetric estimate enables EP to consistently reach 12.45% test error, with only 1.35% degradation with respect to BPTT (see Fig. 3.4A). Therefore, removing the first-order error term in the gradient estimate is critical for scaling to deeper architectures. Proceeding to this end deterministically (with three phases) rather than stochastically (with a randomized nudging sign) appears more reliable.

The results of Table 3.1 also show that the readout scheme introduced in section 3.3.2 to optimize the cross-entropy loss function enables EP to narrow the performance gap with BPTT down to 0.56% while outperforming the Squared Error setting by 0.77%. However, we observe that the test errors reached by BPTT are similar for the squared error and the cross-entropy loss. The fact that only EP benefits from the cross-entropy loss is due to the output not being part of the dynamics, which reduces the number of layers following the dynamics by one.

We also adapted dropout [31] to convergent RNNs (see Appendix B.4.5 for implementation details) to see if the performance could be improved further. However, we can observe from Table 3.1 and Fig. 3.4B that contrary to BPTT, the EP test error is not improved by adding a 0.1 dropout probability in the neuron layer after the convolutions.

Table 3.2: Hyper-parameters used for the CIFAR-10 experiments.

Hyper-parameter	Squared Error	Cross-Entropy
T	250	250
K	30	25
β	0.5	1.0
Batch Size	128	128
Initial learning rates (Layer-wise)	0.25 - 0.15 - 0.1 - 0.08 - 0.05	0.25 - 0.15 - 0.1 - 0.08 - 0.05
Final learning rates	10^{-5}	10^{-5}
Weight decay (All layers)	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
Momentum	0.9	0.9
Epoch	120	120
Cosine Annealing Decay time (epochs)	100	100

3.4.2 ConvNets with unidirectional connections

We now present the accuracy achieved by EP when the architecture uses distinct forward and backward weights, using a softmax readout. For this architecture, the backward weights are defined for all convolutional layers, except the first convolutional layer connected to the static input. The forward and backward weights are initialized randomly and independently at the beginning of training. The backward weights have no bias contrary to their forward coun-

terparts. The hyper-parameters such as learning rate, weight decay and momentum are shared between forward and backward weights.

As seen in Table 3.1, we find that the estimate $\widehat{\nabla}^{\text{VF}}_{\text{sym}}(\beta)$ leads to a poor performance with 75.47% test-error. We concomitantly observed that forward and backward weight did not align well, as shown by the dashed curves in Fig. 3.5. Conversely, when using our new estimate $\widehat{\nabla}^{\text{KP-VF}}_{\text{sym}}(\beta)$ defined in section 3.3.3, a good performance is recovered with only 1.5% performance degradation with respect to the architecture with bidirectional connections, and a 3% degradation with respect to BPTT (see Table 3.1). The discrepancy between the BPTT test error achieved by the architecture with bidirectional (11.12%) and unidirectional (9.46%) connections comes from the increase in parameters provided by backward weights. As observed in the weight alignment curves in Fig. 3.5, forward and backward weights are well aligned by epoch 50 when using the new estimate. These results suggest that enhancing forward and backward weights alignment can help EP training in deep ConvNets.

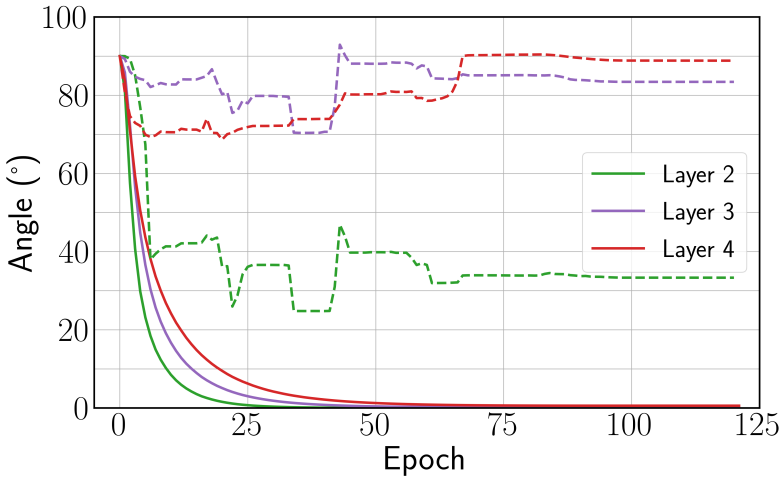

Figure 3.5: Angle between forward and backward weights for the new estimate $\widehat{\nabla}^{\text{KP-VF}}_{\text{sym}}$ introduced (solid) and $\widehat{\nabla}^{\text{VF}}_{\text{sym}}$ (dashed). The angle is not defined for the first layer because the input layer is clamped.

3.5 Discussion

Our results unveil the necessity, in order to scale EP to deep convolutional neural networks on hard visual tasks, to compute better gradient estimates than the conventional implementation of EP. This traditional implementation incorporates a first order gradient estimate bias, which severely impedes the training of deep architectures. Conversely, we saw that the three-phase EP proposed here removes this bias and brings EP performance on CIFAR-10 close to the one achieved by BPTT. Additionally, our new technique to train EP with softmax readout reduces the gap between EP and BPTT further down to 0.56%, while maintaining the locality of

the learning rule of all parameters.

While the test accuracy of BPTT and our adapted EP are very close, we can notice in Table 3.1 that BPTT fits the training data better than EP by at least 2.8%. Also, the introduction of dropout improves BPTT performance, while it has no significant effect on the test accuracy of EP. These two insights combined suggest that EP training may have a self-regularizing effect applied throughout the network, similar to the effects of dropout. We hypothesize this effect to be not only due to the residual estimation bias of the BPTT gradients by EP, but also to an additional inherent error term due to the fact that in practice, the fixed point is approached with a precision that depends on the number of time steps at inference. While the exactness of the fixed point is crucial for EP, BPTT computes exact gradients regardless of whether the fixed point is not exactly reached.

We also saw that employing a new training technique that still preserves the spatial locality of EP computations – and therefore its suitability for neuromorphic implementations – our results extend to the case of an architecture with distinct forward and backward synaptic connections. We only observe a 1.5% performance degradation with respect to the bidirectional architecture. This result demonstrates the scalability of EP without the biologically implausible requirement of a bidirectional connectivity pattern.

Our three steady states-based gradient estimate comes at a computational cost with regards to the conventional EP implementation, as an additional phase is needed. Even though the steady state of the free phase $s_\star$ is not used to compute the gradient estimate in Eq. (3.12), we experimentally found that $s_\star$ is needed as a starting point for the second and third phases. In terms of simulation time, EP is 20% slower than BPTT due to the dynamics performed in second and third phases. However, the memory requirement to store the computational graph unfolded in time in the case of BPTT far outweighs the memory needed by EP, which consists only of the steady states reached by the neurons.

The full potential of EP will be best envisioned on neuromorphic hardware. Multiple works have investigated the implementation of EP on such systems [80, 198, 199, 201, 202], in both rate based [157] and spiking approaches [200]. Most of these approaches employ analog circuits that exploit device physics to implement the dynamics of EP intrinsically. The spatially local nature of EP computations, on top of its connection with physical equations, make this mapping between EP and neuromorphic hardware natural. Our prescription to run two nudging phases with opposite nudging strengths could be implemented naturally in neuromorphic systems. In fact, the use of differential operation to cancel inherent biases is a technique widely used in electronics, and in neuromorphic computing in particular [154]. Overall, our work provides evidence that EP is a compelling approach to scale neuromorphic on-chip training to real-world tasks in a fully local fashion.

Chapter 4

Implementation of Ternary Weights with Resistive RAM Using a Single Sense Operation per Synapse

CHAPTERS 2 and 3 focused on designing algorithms mindful of hardware constraints for neuromorphic applications. In this Chapter, adapted from an article published in IEEE Transactions on Circuits And Systems I [4], we propose a hardware architecture to implement a neural network inference algorithm, using emerging memories. This work is done in collaboration with CEA-Leti and Aix-Marseille University.

The design of systems implementing low precision neural networks with emerging memories such as resistive random access memory (RRAM) is a significant lead for reducing the energy consumption of artificial intelligence (see section 1.4.2.1). To achieve maximum energy efficiency in such systems, logic and memory should be integrated as tightly as possible. We propose a two-transistor/two-resistor memory architecture employing a precharge sense amplifier, where the weight value can be extracted in a single sense operation. Based on experimental measurements on a hybrid 130 nm CMOS/RRAM chip featuring this sense amplifier, we show that this technique is particularly appropriate at low supply voltage, and that it is resilient to process, voltage, and temperature variations. We characterize the bit error rate in our scheme. We show based on neural network simulation on the CIFAR-10 image recognition task that the use of ternary neural networks significantly increases neural network performance, with regards to binary ones, which are often preferred for inference hardware. We finally evidence that the neural network is immune to the type of bit errors observed in our scheme, which can therefore be used without error correction.

4.1 Background

Artificial Intelligence has made tremendous progress in recent years due to the development of deep neural networks. Its deployment at the edge, however, is currently limited by the high power consumption of the associated algorithms [213]. Low precision neural networks are currently emerging as a solution, as they allow the development of low power consumption hardware specialized in deep learning inference [214]. The most extreme case of low precision neural networks, the Binarized Neural Network (BNN), also called XNOR-NET, is receiving particular attention as it is especially efficient for hardware implementation: both synaptic weights and neuronal activations assume only binary values [153, 165]. Remarkably, this type of neural network can achieve high accuracy on vision tasks [169]. One particularly investigated lead is to fabricate hardware BNNs with emerging memories such as resistive RAM or memristors [215–222]. The low memory requirements of BNNs, as well as their reliance on simple arithmetic operations, make them indeed particularly adapted for 'in-memory' or 'near-memory' computing approaches, which achieve superior energy-efficiency by avoiding the von Neumann bottleneck entirely.

Ternary neural networks [223] (TNN, also called Gated XNOR-NET, or GXNOR-NET [224]), which add the value 0 to synaptic weights and activations, are also considered for hardware implementations [225–228]. They are comparatively receiving less attention than binarized neu-

ral networks, however. In this Chapter, we highlight that implementing TNNs does not necessarily imply considerable overhead with regards to BNNs. We introduce a two-transistor/two-resistor memory architecture for TNN implementation. The array uses a precharge sense amplifier for reading weights, and the ternary weight value can be extracted in a single sense operation, by exploiting the fact that latency of the sense amplifier depends on the resistive states of the memory devices. This work extends a hardware developed for the energy-efficient implementation of BNNs [215], where the synaptic weights are implemented in a differential fashion. We, therefore, show that it can be extended to TNNs without overhead on the memory array.

The contribution of this work is as follows. After presenting the background of the work (section 4.1):

- We demonstrate experimentally, on a fabricated 130 nm RRAM/CMOS hybrid chip, a strategy for implementing ternary weights using a precharge sense amplifier, which is particularly appropriate when the sense amplifier is operated at low supply voltage (section 4.2).
- We analyze the bit errors of this scheme experimentally and their dependence on the RRAM programming conditions (section 4.4).
- We verify the robustness of the approach to process, voltage, and temperature variations (section 4.3).
- We carry simulations that show the superiority of TNNs over BNNs on the canonical CIFAR-10 vision task, and evidence the error resilience of hardware TNNs (section 4.5).
- We discuss the results, and compare our approach with the idea of storing three resistance levels per device.

The main equation in conventional neural networks is the computation of the neuronal activation $A_j = f\left(\sum_i W_{ji} X_i\right)$, where A_j, the synaptic weights W_{ji}, and input neuronal activations X_i assume real values, and f is a non-linear activation function. Binarized neural networks (BNNs) are a considerable simplification of conventional neural networks, in which all neuronal activations (A_j, X_i) and synaptic weights W_{ji} can only take binary values meaning +1 and −1. Neuronal activation then becomes:

$$A_j = \text{sign}\left(\sum_i XNOR\left(W_{ji}, X_i\right) - T_j\right), \tag{4.1}$$

where sign is the sign function, T_j is a threshold associated with the neuron, and the $XNOR$ operation is defined in Table 4.1. Training BNNs is a relatively sophisticated operation, during which each synapse needs to be associated with a real value in addition to its binary value (see Appendix C). Once training is finished, these real values can be discarded, and the neural network is entirely binarized. Due to their reduced memory requirements, and reliance

on simple arithmetic operations, BNNs are especially appropriate for in- or near- memory implementations. In particular, multiple groups investigate the implementation of BNN inference with resistive memory tightly integrated at the core of CMOS [215–222]. Usually, resistive memory stores the synaptic weights W_{ji}. However, this comes with a significant challenge: resistive memory is prone to bit errors, and in digital applications, is typically used with strong error-correcting codes (ECC). ECC, which requires large decoding circuits [229], goes against the principles of in- or near- memory computing. For this reason, [215] proposes a two-transistor/two-resistor (2T2R) structure, which reduces resistive memory bit errors, without the need for ECC decoding circuit, by storing synaptic weights in a differential fashion. This architecture allows the extremely efficient implementation of BNNs, and using the resistive memory devices in very favorable programming conditions (low energy, high endurance). It should be noted that systems using this architecture function with row-by-row read operations, and do not use the in-memory computing technique of using the Kirchhoff current law to perform the sum operation of neural networks, while reading all devices at the same time [150, 152]. This choice limits the parallelism of such architectures, while at the same time avoiding the need of analog-to-digital conversion and analog circuits such as operational amplifiers, as discussed in detail in [154].

In this Chapter, we show that the same architecture can be used for a generalization of BNNs – ternary neural networks (TNNs)[1], where neuronal activations and synaptic weights A_j, X_i, and W_{ji} can now assume three values: +1, −1, and 0. Equation (4.1) now becomes:

$$A_j = \phi\left(\sum_i GXNOR\left(W_{ji}, X_i\right) - T_j\right). \tag{4.2}$$

$GXNOR$ is the 'gated' XNOR operation that realizes the product between numbers with values +1, −1 and 0 (Table 4.1). ϕ is an activation function that outputs +1 if its input is greater than a threshold Δ, −1 if the input is lesser than $-\Delta$ and 0 otherwise. We show experimentally and by circuit simulation in sec. 4.2 how the 2T2R BNN architecture can be extended to TNNs with practically no overhead, in sec. 4.4 its bit errors, and in sec. 4.5 the corresponding benefits in terms of neural network accuracy.

[1] In the literature, the name 'Ternary Neural Networks' is sometimes also used to refer to neural networks where the synaptic weights are ternarized, but the neuronal activations remain real or integer [230, 231].

Table 4.1: Truth Tables of the XNOR and GXNOR Gates.

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

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

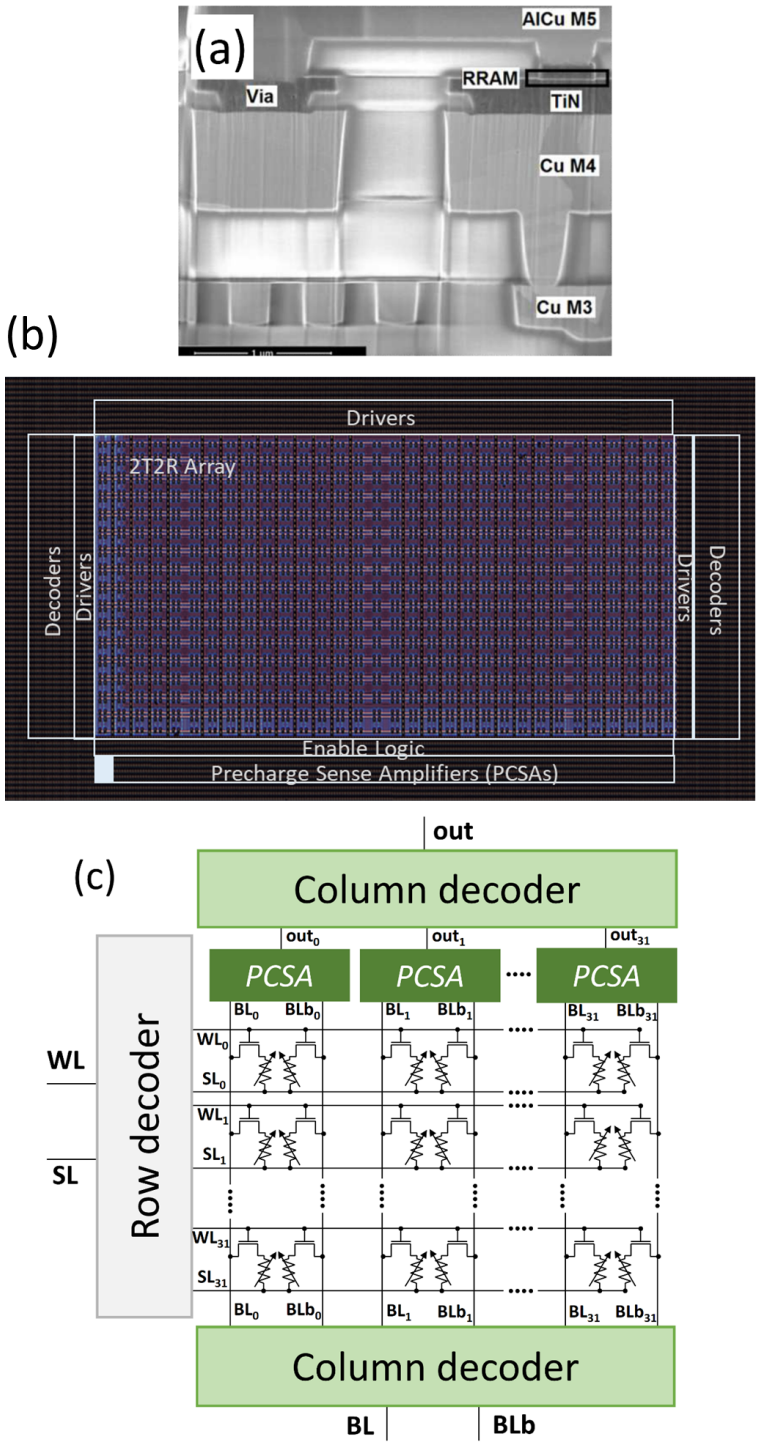

Figure 4.1: (a) Electron microscopy image of a hafnium oxide resistive memory cell (RRAM) integrated in the backend-of-line of a 130 nm CMOS process. (b) Photograph and (c) simplified schematic of the hybrid CMOS/RRAM test chip characterized in this work. The white rectangle in (b) materializes a single PCSA.

4.2 The Operation of A Precharge Sense Amplifier Can Provide Ternary Weights

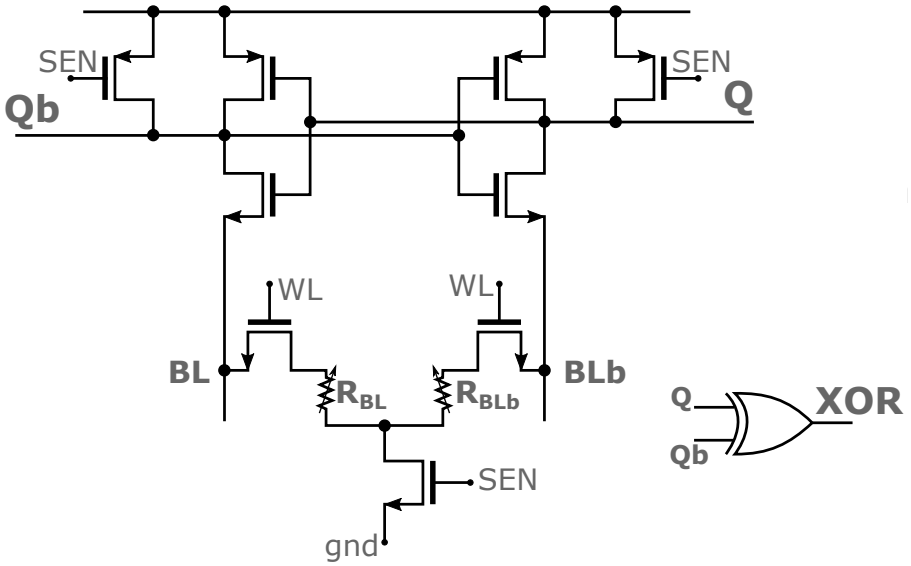

Figure 4.2: Schematic of the precharge sense amplifier fabricated in the test chip.

In this Chapter, we use the architecture of [215], where synaptic weights are stored in a differential fashion. Each bit is implemented using two devices programmed either as low resistance state (LRS) / high resistance state (HRS) to mean weight +1 or HRS/LRS to mean weight −1. Fig. 4.1 presents the test chip used for the experiments. This chip cointegrates 130 nm CMOS and resistive memory in the back-end-of-line, between levels four and five of metal. The resistive memory cells are based on 10 nm thick hafnium oxide (Fig. 4.1(a)). All devices are integrated with a series NMOS transistor. After an initial forming step (consisting in the application of a voltage ramp from zero volts to 3.3 V at a rate of $1000\,\mathrm{V}\cdot\mathrm{s}^{-1}$, and with a current limited to a compliance of 200 µA), the devices can switch between high resistance state (HRS) and low resistance state (LRS), through the dissolution or creation of conductive filaments of oxygen vacancies. Programming into the HRS is obtained by the application of a negative RESET voltage pulse (typically between 1.5 V and 2.5 V during 1 µs). Programming into the LRS is obtained by the application of a positive SET pulse (also typically between 1.5 V and 2.5 V during 1 µs), with current limited to a compliance current through the choice of the voltage applied on the transistor gate through the word line (WL). This test chip is designed with highly conservative sizing, allowing the application of a wide range of voltages and electrical currents to the RRAM cells. The area of each bit cell is 6.6 µm × 6.9 µm. More details on the RRAM technology are provided in [154].

Our experiments are based on a 2,048 devices array incorporating all sense and periph-

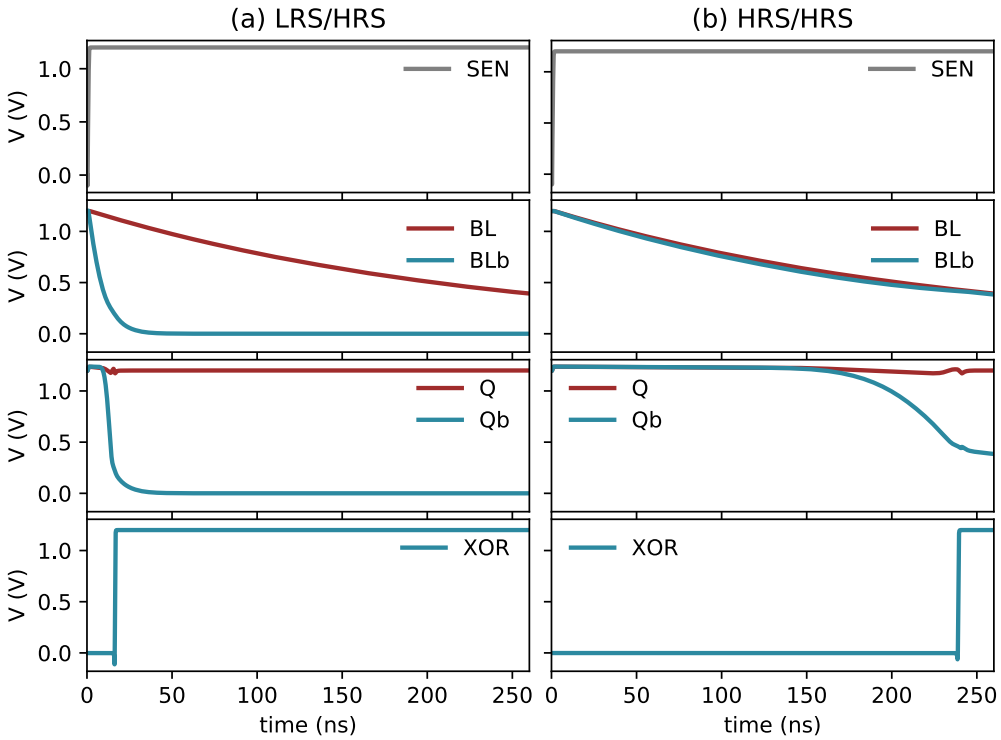

Figure 4.3: Circuit simulation of the precharge sense amplifier of Fig. 4.2 with a supply voltage of 1.2 V, using thick oxide transistors (nominal voltage of 5 V), if the two devices are programmed in an (a) LRS / HRS (5 kΩ/350 kΩ) or (b) HRS/HRS (320 kΩ/350 kΩ) configuration.

ery circuitry, illustrated in Fig. 4.1(b-c). The ternary synaptic weights are read using on-chip precharge sense amplifiers (PCSA), presented in Fig. 4.2, and initially proposed in [232] for reading spin-transfer magnetoresistive random access memory. Fig. 4.3(a) shows an electrical simulation of this circuit to explain its working principle, using the Mentor Graphics Eldo simulator. These first simulations are presented in the commercial 130 nm ultra-low leakage technology, used in our test chip, with a low supply voltage of 1.2 V [233], with thick oxide transistors (the nominal voltage in this process for thick oxide transistor is 5 V). Since the technology targets ultra-low leakage applications the threshold voltages are significantly high (around 0.6 V), thus a supply voltage of 1.2 V significantly reduces the overdrive of the transistors ($V_{GS} - V_{TH}$).

In the first phase (SEN=0), the outputs Q and Qb are precharged to the supply voltage V_{DD}. In the second phase (SEN=V_{DD}), each branch starts to discharge to the ground. The branch that has the resistive memory (BL or BLb) with the lowest electrical resistance discharges faster and causes its associated inverter to drive the output of the other inverter to the supply voltage. At the end of the process, the two outputs are therefore complementary and can be used to tell which resistive memory has the highest resistance and therefore the synaptic weight. We observed that the convergence speed of a PCSA depends heavily on the resistance state of the two resistive memories. This effect is particularly magnified when the PCSA is used with a reduced overdrive, as presented here: the operation of the sense amplifier is slowed down, with regards to nominal voltage operation, and the convergence speed differences between resistance values become more apparent. Fig. 4.3(b) shows a simulation where the two devices, BL and BLb, were programmed in the HRS. We see that the two outputs converge to complementary values in more than 200 ns, whereas less than 50 ns were necessary in Fig. 4.3(a), where the devices are programmed in complementary LRS/HRS states.

These first simulations suggest a technique for implementing ternary weights using the memory array of our test chip. Similarly to when this array is used to implement BNN, we propose to program the devices in the LRS/HRS configuration to mean the synaptic weight 1, and HRS/LRS to mean the synaptic weight −1. Additionally, we use the HRS/HRS configuration to mean synaptic weight 0, while the LRS/LRS configuration is avoided. The sense operation is performed during a duration of 50 ns. If at the end of this period, outputs Q and Qb have differentiated, causing the output of the XOR gate to be 1, output Q determines the synaptic weight (1 or −1). Otherwise, the output of the XOR gate is 0, and the weight is determined to be 0.

This type of coding is reminiscent to the one used by the 2T2R ternary content-addressable memory (TCAM) cell of [234], where the LRS/HRS combination is used for coding 0, the HRS/LRS combination for coding 1, and the HRS/HRS combination for coding 'don't care' (or X).

Experimental measurements on our test chip confirm that the PCSA can be used in this fashion. We first focus on one synapse of the memory array. We program one of the two devices (BLb) to a resistance of 100 kΩ. We then program its complementary device BL to several resistance values, and for each of them perform 100 read operations of duration 50 ns, using

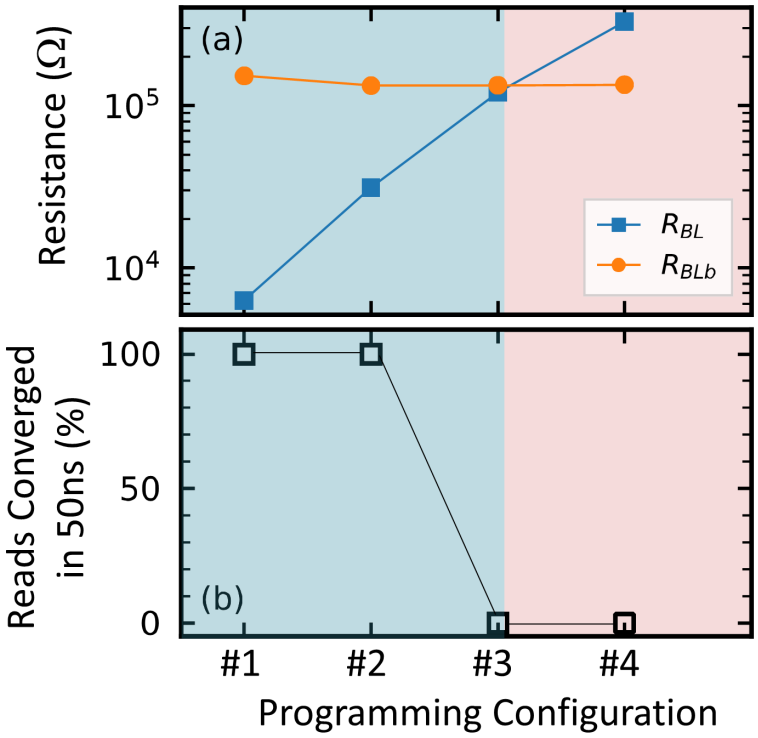

Figure 4.4: Two devices have been programmed in four distinct programming configurations given by the resistance of each device presented in (a), and measured using an on-chip sense amplifier. (b) Proportion of read operations that have converged in 50 ns, over 100 trials.

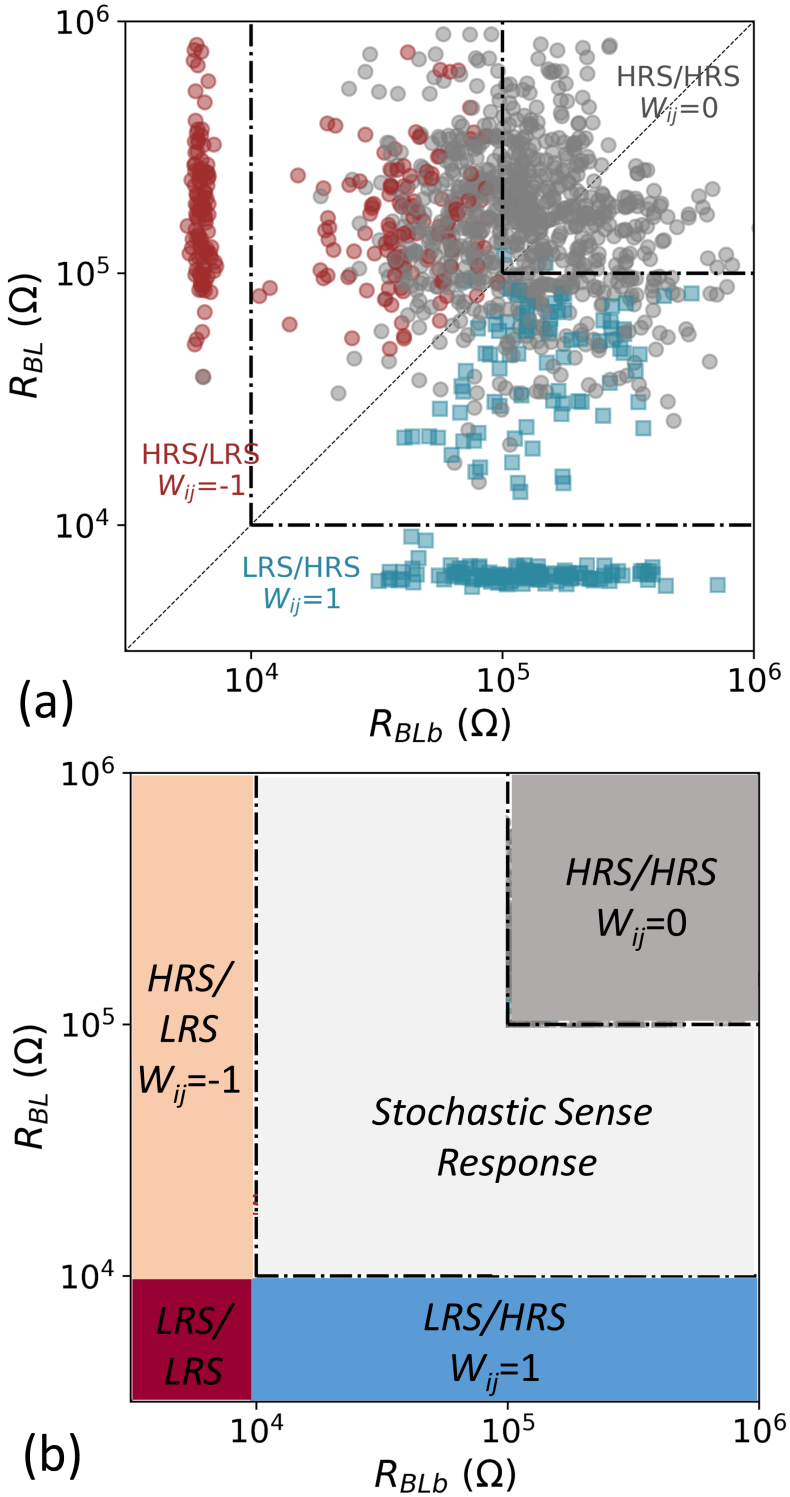

Figure 4.5: For 109 device pairs programmed with multiple R_{BL}/R_{BLb} configuration, value of the synaptic weight measured by the on-chip sense amplifier using the strategy described in body text and 50 ns reading time.

on-chip PCSAs.

These PCSAs are fabricated using thick-oxide transistors, designed for a nominal supply voltage of $5V$, and here used with a supply voltage of $1.2V$, close to their threshold voltage ($0.6V$), to reduce their overdrive, and thus to exacerbate the PCSA delay variations. In the test chip, they are sized conservatively with a total area of $290\,\mu m^2$. The use of thick oxide transistors in this test chip allows us to investigate the behavior of the devices at high voltages, without the concern of damaging the CMOS periphery circuits. Fig. 4.4 plots the probability that the sense amplifier has converged during the read time. In 50 ns, the read operation is only converged if the resistance of the BL device is significantly lower than 100 kΩ.

To evaluate this behavior in a broader range of programming conditions, we repeated the experiment on 109 devices and their complementary devices of the memory array programmed, every 14 times, with various resistance values in the resistive memory, and performed a read operation in 50 ns with an on-chip PCSA. The memory array of our test chip features one separate PCSA per column. Therefore, 32 different PCSAs are used in our results. Fig. 4.5(a) shows, for each couple of resistance values R_{BL} and R_{BLb} if the read operation had converged with $Q = V_{DD}$ (blue), meaning a weight of 1, converged with $Q = 0$ (red), meaning a weight of -1, or not converged (grey) meaning a weight of 0.

The results confirm that LRS/HRS or HRS/LRS configurations may be used to mean weights 1 and -1, and HRS/HRS for weight 0. When both devices are in HRS (resistance higher than 100 kΩ, the PCSA never converges within 50 ns (weight of 0). When one device is in LRS (resistance lower than 10 kΩ, the PCSA always converges within 50 ns (weight of ± 1). The separation between the 1 (or -1) and 0 regions is not strict, and for intermediate resistance values, we see that the read operation may or may not converge in 50 ns. Fig. 4.5(b) summarizes the different operation regimes of the PCSA.

Table 4.2: Error Rates on Ternary Weights Measured Experimentally

Programming Conditions	Type 1 ($1 \longleftrightarrow -1$)	Type 2 ($\pm 1 \rightarrow 0$)	Type 3 ($0 \rightarrow \pm 1$)
Fig. 4.7(a)	$< 10^{-6}$	$< 1\%$	6.5%
Fig. 4.7(b)	$< 10^{-6}$	$< 1\%$	18.5%

4.3 Impact of Process, Voltage, and Temperature Variations

We now verify the robustness of the proposed scheme to process, voltage, and temperature variation. For this purpose, we performed extensive circuit simulations of the operation of the

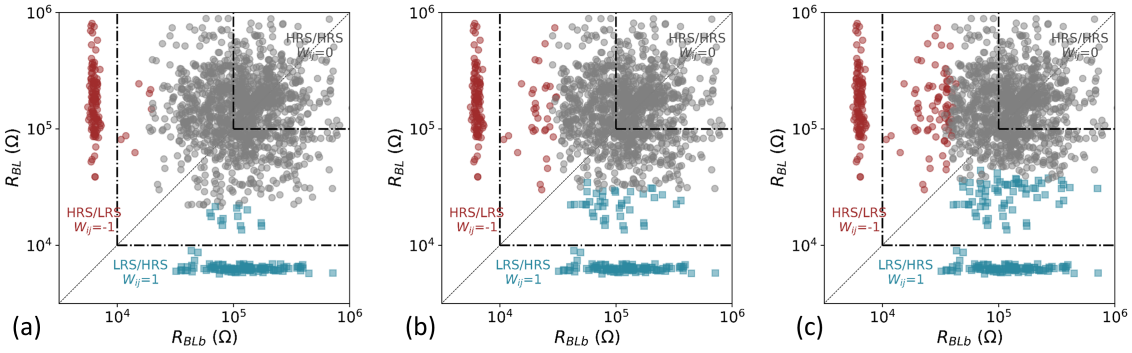

Figure 4.6: Three Monte Carlo SPICE-based simulation of the experiments of Fig. 4.5, in three situations: (a) slow transistors (0 °C temperature, 1.1 V supply voltage), (b) experimental conditions (27 °C temperature, 1.2 V supply voltage), (c) fast transistors (60 °C temperature, 1.3 V supply voltage). The simulations include local and global process variations, as well as transistor mismatch, in a way that each point in the Figure is obtained using different transistor parameters. All results are plotted in the same manner and with the same conventions as Fig. 4.5.

sense amplifier, reproducing the conditions of the experiments of Fig. 4.5, using the same resistance values for the RRAM devices, and including process, voltage, and temperature variations. The results of the simulations are processed and plotted using the same format as the experimental results of Fig. 4.5, to ease comparison.

These simulations are obtained using the Monte Carlo simulator provided by the Mentor Graphics Eldo tool with parameters validated on silicon, provided by the design kit of our commercial CMOS process. Each point in the graphs of Fig. 4.6 therefore features different transistor parameters. We included global and local process variations, as well as transistor mismatch, in order to capture the whole range of transistor variabilities observed in silicon. In order to assess the impact of voltage and temperature variations, these simulations are presented in three conditions: slow transistors (0 °C temperature, and 1.1 V supply voltage, Fig. 4.6(a)), experimental conditions (27 °C temperature, and 1.2 V supply voltage, Fig. 4.6(b)), and fast transistors (60 °C temperature, and 1.3 V supply voltage, Fig. 4.6(c)). The RRAM devices are modeled by resistors. Their process variations are naturally included through the use of different resistance values in Fig. 4.6. The impact of voltage variation on RRAM is naturally included through Ohm's law, and the impact of temperature variation, which is smaller than on transistors, is neglected.

In all three conditions, the simulation results appear very similar to the experiments. Three clear regions are observed: non-convergence of the sense amplifier within 50 ns for devices in HRS/HRS, and convergence within this time to a +1 or −1 value for devices in LRS/HRS and HRS/LRS, respectively. However, the frontier between these regimes is much sharper in the simulations than in the experiments. As the different data points in Fig. 4.6 differ by process and mismatch variations, this suggests that process variation does not cause the stochasticity observed in the experiments of Fig. 4.5, and that they have little impact in our scheme.

We also see that the frontier between the different sense regimes in all three operating con-

ditions remains firmly within the $10-100k\Omega$ range, suggesting that even high variations of voltage ($\pm 0.1V$) and temperature ($\pm 30°C$) do not endanger the functionality of our scheme. Logically, in the case of fast transistors, the frontier is shifted toward higher resistances, whereas in the case of slow transistors, it is shifted toward lower resistances. Independent simulations allowed verifying that this change is mostly due to the voltage variations: the temperature variations have an almost negligible impact on the proposed scheme.

We also observed that the impact of voltage variations increased importantly when reducing the supply voltage. For example, with a supply voltage of $0.7V$ instead of the $1.2V$ value considered here, variations of the supply voltage of $\pm 0.1V$ can impact the mean switching delay of the PCSA, by a factor two. The thick oxide transistors used in this work have a nominal voltage of $5V$, and a typical threshold voltage of approximately $0.6V$. Therefore, although our scheme is especially appropriate for supply voltages far below the nominal voltage, it is not necessarily appropriate for voltages in the subthreshold regime, or very close to the threshold voltage.

4.4 Programmability of Ternary Weights

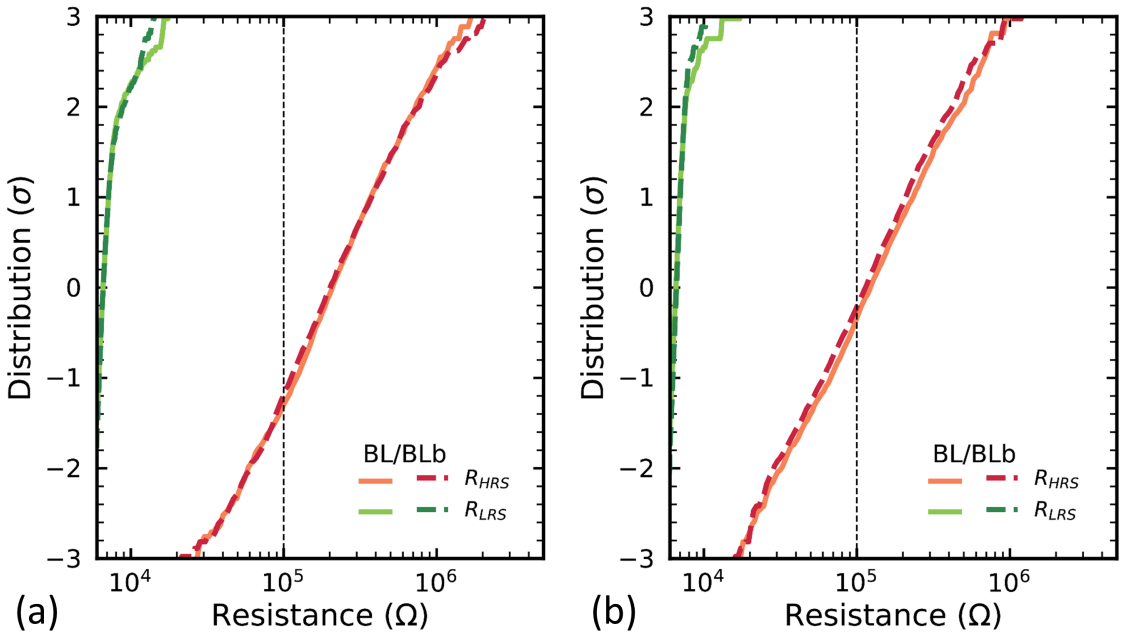

Figure 4.7: Distribution of the LRS and HRS states programmed with a SET compliance of 200 µA, RESET voltage of 2.5 V and programming pulses of (a) 100 µs and (b) 1 µs. Measurements are performed 2,048 RRAM devices, separating bit line (full lines) and bit line bar (dashed lines) devices.

To ensure reliable functioning of the ternary sense operation, we have seen that devices in LRS should be programmed to electrical resistance below 10 kΩ, and devices in HRS to resis-

tances above 100 kΩ (Fig. 4.5(b)). The electrical resistance of resistive memory devices depends considerably on their programming conditions [154, 235]. Fig. 4.7 shows the distributions of LRS and HRS resistances using two programming conditions, over the 2,048 devices of the array, differentiating devices connected to bit lines and to bit lines bar. We see that in all cases, the LRS features a tight distribution. The SET process is indeed controlled by a compliance current that naturally stops the filament growth at a targeted resistance value [236]. An appropriate choice of the compliance current can ensure LRS below 10 kΩ in most situations.

On the other hand, the HRS shows a broad statistical distribution. In the RESET process, the filament indeed breaks in a random process, making it extremely hard to control the final state [236, 237]. The use of stronger programming conditions leads to higher values of the HRS.

This asymmetry between the variability of LRS and HRS means that in our scheme, the different ternary weight values feature different error rates naturally. The ternary error rates in the two programming conditions of Fig. 4.7(a) are listed in Table 4.2. Errors of Type 1, where weight values of 1 and −1 are inverted are the least frequent. Errors of Type 2, where a weight value of 1 or −1 is replaced by a weight value of 0 are infrequent as well. On the other hand, due to the large variability of the HRS, weight values 0 have a significant probability to be measured as 1 or −1 (Type 3 errors): 6.5% in the conditions of Fig. 4.7(a), and 18.5% in the conditions of Fig. 4.7(b).

Some resistive memory technologies with large memory windows, such as specifically optimized conductive bridge memories [238], would feature lower Type 3 error rates. Similarly, program-and-verify strategies [239–241] may reduce this error rate. Nevertheless, the higher error rate for zeros than for 1 and −1 weights is an inherent feature of our architecture. Therefore, in the next section, we assess the impact of these errors on the accuracy of neural networks.

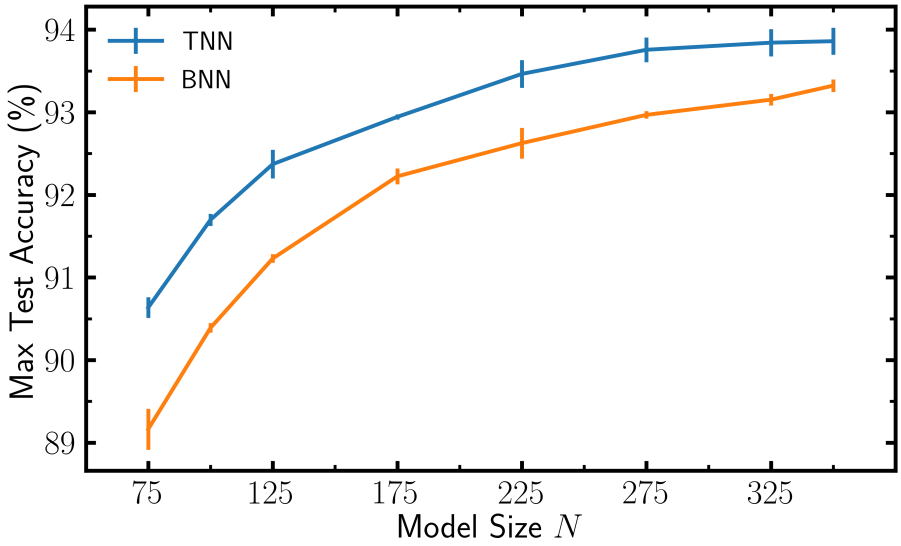

Figure 4.8: Simulation of the maximum test accuracy reached during one training procedure, averaged over five trials, for BNNs and TNNs with various model sizes on the CIFAR-10 dataset. Error bar is one standard deviation.

4.5 Network-Level Implications

We first investigate the accuracy gain when using ternarized instead of binarized networks. We trained BNN and TNN versions of networks with Visual Geometry Group (VGG) type architectures [37] on the CIFAR-10 task of image recognition, consisting in classifying 1,024 pixels color images among ten classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck) [210]. Simulations are performed using PyTorch 1.1.0 [35] on a cluster of eight Nvidia GeForce RTX 2080 GPUs.

The architecture of our networks consists of six convolutional layers with kernel size three. The number of filters at the first layer is called N and is multiplied by two every two layers. Maximum-value pooling with kernel size two is used every two layers and batch-normalization [86] every layer. The classifier consists of one hidden layer of 512 units. For the TNN, the activation function has a threshold $\Delta = 5 \cdot 10^{-2}$ (as defined in section 4.1). The training methods for both the BNN and the TNN are described in the Appendix C. The training is performed using the AdamW optimizer [30, 242], with minibatch size 128. The initial learning rate is set to 0.01, and the learning rate schedule from [212, 242] (Cosine annealing with two restarts, for respectively 100, 200, 400 epochs) is used, resulting in a total of 700 epochs. Training data is augmented using random horizontal flip, and random choice between cropping after padding and random small rotations.

No error is added during the training procedure, as our device is meant to be used for inference. The synaptic weights encoded by device pairs would be set after the model has been trained on a computer.

Fig. 4.8 shows the maximum test accuracy resulting from these training simulations, for different sizes of the model. The error bars represent one standard deviation of the training accuracies. TNNs always outperform BNNs with the same model size (and, therefore, the same number of synapses). The most substantial difference is seen for smaller model size, but a significant gap remains even for large models. Besides, the difference in the number of parameters required to reach a given accuracy for TNNs and BNNs increases with higher accuracies. There is, therefore, a definite advantage to use TNNs instead of BNNs.

Fig. 4.8 compared fully ternarized (weights and activations) with regards to fully binarized (weights and activations) ones. Table 4.3 lists the impact of weight ternarization for different types of activations (binary, ternary, and real activation). All results are reported on a model of size $N = 128$, trained on CIFAR-10, and are averaged over five training procedures. We observe that for BNNs and TNNs with quantized activations, the accuracy gains provided by ternary weights over binary weights are 0.84 and 0.86 points and are statistically significant over the standard deviations. This accuracy gain is more important than the gain provided by ternary activations over binary activations, which is about 0.3 points. This bigger impact of weight ternarization over ternary activation may come from the ternary kernels having a better expressing power over binary kernels, which are often redundant in practical settings [153]. The gain of ternary weights drops to 0.26 points if real activation is allowed (using rectified linear unit, or ReLU, as activation function, see Appendix C), and is not statistically significant considering the standard deviations.

Quantized activations are vastly more favorable in the context of hardware implementations, and in this situation, there is thus a statistically significant benefit provided by ternary weights over binary weights.

Table 4.3: Comparison of the gain in test accuracy for a $N = 128$ model size on CIFAR-10 obtained by weight ternarization instead of binarization for three types of activation quantization.

	Activations		
	Binary	Ternary	Full Precision
Weights			
Binary	91.19 ± 0.08	91.51 ± 0.09	93.87 ± 0.19
Ternary	92.03 ± 0.12	92.35 ± 0.05	94.13 ± 0.10
Gain of ternarization	*0.84*	*0.86*	*0.26*

We finally investigate the impact of bit errors in BNNs and TNNs to see if the advantage provided by using TNNs in our approach remains constant when errors are taken into account. Consistently with the results reported in section 4.4, three types of errors are investigated: Type 1 errors are sign switches, e.g., $+1$ mistaken for -1, Type 2 errors are only defined for TNNs and

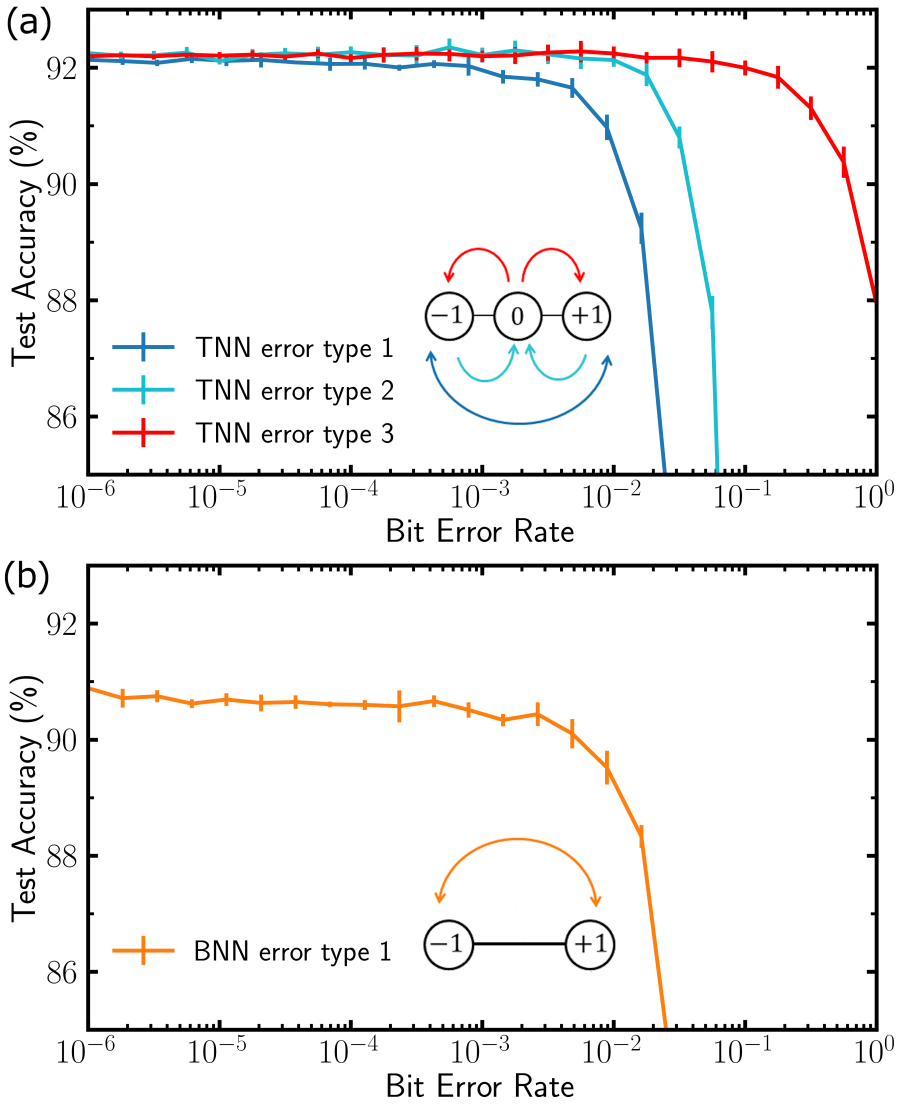

Figure 4.9: Simulation of the impact of Bit Error Rate on the test accuracy at inference time for model size $N = 128$ TNN in (a) and BNN in (b). Type 1 errors are sign switches (e.g. $+1$ mistaken for -1), Type 2 errors are ± 1 mistaken for 0, and Type 3 errors are 0 mistaken for ± 1, as described in the inset schematics. Errors are sampled at each mini batch and the test accuracy is averaged over five passes through the test set. Error bars are one standard deviation. The bit error rate is given as an absolute rate.

correspond to ±1 mistaken for 0, and Type 3 errors are 0 mistaken for ±1, as illustrated in the inset schematic of Fig. 4.9(a).

Fig. 4.9(a) shows the impact of these errors on the test accuracy for different values of the error rate at inference time. These simulation results are presented on CIFAR-10 with a model size of $N = 128$. Errors are randomly and artificially introduced in the weights of the neural network . Bit errors are included at the layer level and sampled at each mini-batch of the test set. Type 1 errors switch the sign of a synaptic weight with a probability equal to the rate of type 1 errors. Type 2 errors set a non-zero synaptic weight to 0 with a probability equal to the type 2 error rate. Type 3 errors set a synaptic weight of 0 to ±1 with a probability equal to the type 3 error rate, the choice of the sign (+1 or −1) is made with 0.5 probability. Fig. 4.9 is obtained by averaging the test accuracy obtained for five passes through the test set for increasing bit error rate.

Type 1 errors have the most impact on neural network accuracy. As seen in Fig. 4.9(b), the impact of these errors is similar to the impact of weight errors in a BNN. On the other hand, Type 3 errors have the least impact, with bit error rates as high as 20% degrading surprisingly little the accuracy. This result is fortunate, as we have seen in section 4.4 that Type 3 errors are the most frequent in our architecture.

We also performed simulations considering all three types of error at the same time, with error rates reported in Table 4.2 corresponding to the programming conditions of Fig. 4.7(a) and 4.7(b). For Type 1 and Type 2 errors, we considered the upper limits listed in Table 4.2. For the conditions of Fig. 4.7(a) (Type 3 error rate of 6.5%), the test accuracy was degraded from 92.2% to $92.05 \pm 0.14\%$, and to $92.02 \pm 0.17\%$ for the conditions of 4.7(b) (Type 3 error rate of 18.5%), where the average and standard deviation is performed over 100 passes through the test set. We found that the slight degradation on CIFAR-10 test accuracy was mostly due to the Type 2 errors, although Type 3 errors are much more frequent.

The fact that mistaking a 0 weight for a ±1 weight (Type 3 error) has much less impact than mistaking a ±1 weight for a 0 weight (Type 2 error) can seem surprising. However, it is known, theoretically and practically, that in BNNs, some weights have little importance to the accuracy of the neural networks [1]. They typically correspond to synapses that feature a 0 weight in a TNN, whereas synapses with ±1 weights in a TNN correspond to 'important' synapses of a BNN. It is thus understandable that errors on such synapses have more impact on the final accuracy of the neural network.

4.6 Comparison with Three-Level Programming

An alternative approach to implementing ternary weights with resistive memory can be to program the individual devices into there separate levels. This idea is feasible, as the resistance level of the LRS can to a large extent be controlled through the choice of the compliance current during the SET operation in many resistive memory technologies [154, 236].

The obvious advantage of this approach is that it requires a single device per synapse. This idea also brings several challenges. First, the sense operation has to be more complex. The most natural technique is to perform two sense operations, comparing the resistance of a device under test to two different thresholds. Second, this technique is much more prone to bit errors than our technique, as states are not programmed in a differential fashion [154]. Additionally, this approach does not feature the natural resilience to Type 1 and Type 2 errors, and Type 2 and Type 3 errors will typically feature similar rates. Finally, unlike ours, this approach is prone to resistive drift, inherent to some resistive memory technologies [243].

These comments suggest that the choice of a technique for storing ternary weights should be dictated by technology. Our technique is especially appropriate for resistive memories not supporting single-device multilevel storage, with high error rates, or resistance drift. The three-levels per devices approach would be the most appropriate with devices with well controlled analog storage properties.

4.7 Conclusion

In this Chapter, we revisited a differential memory architecture for BNNs. We showed experimentally on a hybrid CMOS/RRAM chip that, its sense amplifier can differentiate not only the LRS/HRS and HRS/LRS states, but also the HRS/HRS states in a single sense operation. This feature allows the architecture to store ternary weights, and to provide a building block for ternary neural networks. We showed by neural network simulation on the CIFAR-10 task the benefits of using ternary instead of binary networks, and the high resilience of TNNs to weights errors, as the type of errors observed experimentally in our scheme is also the type of errors to which TNNs are the most immune. This resilience allows the use of our architecture without relying on any formal error correction. Our approach also appears resilient to process, voltage, and temperature variation if the supply voltage remains reasonably higher than the threshold voltage of the transistors.

As this behavior of the sense amplifier is exacerbated at supply voltages below the nominal voltage, our approach especially targets extremely energy-conscious applications such as uses within wireless sensors or medical applications. This work opens the way for increasing the edge intelligence in such contexts, and also highlights that the low voltage operation of circuits may sometimes provide opportunities for new functionalities.

Conclusions and future work

Summary

This thesis started with the following fascinating paradox: deep learning algorithms based on artificial neural networks enable modern computers to perform cognitive tasks, but computers consume orders of magnitude more energy than living creatures, limiting progress, and deployment in resource-constrained environments. In Chapter 1, we established the hardware reasons for this discrepancy. While neural tissues closely intertwine computation and memory, those are conceptually and physically separated in the von Neumann architecture of modern computers, leading to costly data movements and the so-called von Neumann bottleneck. We reviewed the deep neural networks algorithms responsible for the recent breakthroughs and pointed out some of their limitations when designing neuromorphic hardware:

- Catastrophic forgetting, i.e. forgetting the previous task when learning a new one. This would prevent a dedicated chip to learn continually several tasks or from a stream of data.
- Credit assignment, i.e. how synapses must change to improve an output located far away in the network. Back-propagation solves credit assignment with explicit computation, which complicates the implementation on a circuit of artificial neurons.

We then presented biological inspiration as a path toward solving those limitations. Finally, we reviewed the current hardware solutions for more energy-efficient AI with respect to inference only, and learning on chip. While dedicated hardware based on neuroscience (bottom-up) are low-power, they lack powerful algorithms to solve complex tasks. On the other hand, hardware based on successful deep learning algorithms (top-down) somewhat reduces the energy requirements of conventional CPU/GPU, but remains more power-hungry than biological neural tissues.

The task of neuromorphic computing as a field is to find hardware substrates and algorithms to efficiently emulate neural systems. New hardware is needed to perform computation beyond the von Neumann paradigm, but new algorithms are also needed because the ones responsible for the recent milestones still have striking differences with the biological neural networks, which make them challenging to implement on dedicated substrates. This thesis tackles both aspects with Chapter 2 and 3 focused on the algorithm side, and Chapter 4 focused on the hardware side.

In Chapter 2, we present a way to reduce catastrophic forgetting in binarized neural networks, published in Nature Communications [163]. This work bridges computational neuroscience models of synaptic metaplasticity done in the ideal observer framework in Hopfield-like networks, and continual learning approaches based on regularization. The training process of binarized neural networks involves a hidden quantity for each synapse, which is usually discarded after training, begging the question of the difference between equal binary parameters with different hidden magnitudes. In this work, we essentially show that this quan-

tity, obtained as a result of the training process, is relevant for performing continual learning. We introduce a simple consolidation rule based on the hidden magnitude and test our approach on several tasks sequences such as permuted MNIST and MNIST/Fashion MNIST. Our approach performs almost identical to elastic weight consolidation, an established concurrent regularization-based approach for continual learning, but does not need updating the loss function nor performing additional computation between the tasks, allowing for more online settings. We explore a setting that we call 'stream learning', in which a task is learned as a sequence of sub-datasets, and show that our metaplastic binarized neural network outperforms its non-metaplastic counterpart. We propose a mathematical derivation on a simplified binarized optimization of a quadratic cost, which provides insight for understanding why high hidden magnitude correspond to an important binarized weight. Finally, we present a more complex synaptic models to allow graceful long-term forgetting and stationary distribution of parameters.

While Chapter 2 provided a local and hardware-friendly consolidation rule for continual learning, the credit assignment was still performed by back-propagation. To go toward on-chip learning with local credit assignment, the Chapter 3 focuses on Equilibrium Propagation (EP), in collaboration with the Mila, published in Frontiers in Neuroscience [2]. We uncover a bias in the gradient estimate of EP, which prevents it from training deeper architectures on more complex tasks. This bias is inherent in one-sided derivative estimation, and can be removed by a symmetric estimate. With this new estimate, we show for the first time that EP can train deep convolutional architectures on natural images (CIFAR-10), closely matching back-propagation through time. This result propels EP as a strong candidate to design learning-capable chips. We also present several optimizations to EP: we show how to optimize another loss function while preserving local credit assignment, making EP more flexible. Finally, we study the setting of asymmetric synaptic connections, and show that an alignment mechanism is sufficient to preserve learning.

Finally, Chapter 4 is more hardware oriented. We present a hardware solution to implement a trained ternarized neural networks for low-power inference, first published in the proceedings of IEEE AICAS 2020 [3], and published in the IEEE Transactions on Circuits And Systems I [4] as an extended version. The work is done in collaboration with CEA-Leti and Aix-Marseille University. The architecture consists of a hybrid CMOS/RRAM crossbar using a two transistors/two resistors encoding of parameters. The hafnium oxide RRAM can be set either in high or low resistance states (HRS or LRS). This architecture was studied in previous works to implement binarized neural networks with great resilience to device errors by encoding parameters either as HRS/LRS pairs or LRS/HRS pairs. In this work, we show that the same architecture can be used to encode ternarized parameters when using the HRS/HRS pair and operating the transistors in the low supply voltage regime. The precharge sense amplifier used to read the value of the parameters exhibit a slower convergence time for HRS/HRS pairs, which makes it possible to discriminate between zero and non-zero parameters. We present CMOS simu-

lations and experiments on a fabricated chip using a 130 nm process to prove the feasibility of our approach. Neural networks simulations show the consistent benefit in accuracy when using ternarized networks over binarized networks. Finally, the impact of the device errors introduced by this new encoding scheme is investigated in simulations. The impact of error is found to be minor, which preserves the accuracy benefit provided by our approach.

Perspectives

The projects presented in this thesis are independent but complementary, and combining the results of different chapters opens exciting research directions. For instance, EP has been adapted to binarized neural networks [194], and could therefore benefit from the results of Chapter 2 on metaplasticity. This would produce a model with a learning rule and a consolidation rule that are both local in space. In addition, the performance increase brought by ternarization could be investigated in the contexts of metaplastic binarized neural networks and binarized EP.

Future works should also take advantage of the algorithms studied in this thesis to design neuromorphic hardware. Using the emerging materials presented in section 1.4.2.2, in particular the non volatile electric field control magnetism, one could design a metaplastic synapse in the spirit of Chapter 2. The position of a magnetic domain wall could encode the hidden magnitude of a binarized parameter, and magnetic anisotropy gradients in the materials could provide metaplastic properties out of device behavior, without overhead and independent from the learning rule.

Based on the results presented in Chapter 3, designing a hardware substrate that can implement Equilibrium Propagation is even more promising. The design could take advantage of a physical energy to implement neural dynamics, as in [157], or implement the dynamics explicitly. For such a design, a proof of concept on a easy task would be a first step giving time to work on some limitations of EP. Indeed, although image recognition on CIFAR-10 proves that EP can train a network to extract meaningful features, there are examples of algorithms working on CIFAR-10 but failing on real world tasks such as ImageNet. Further works should thus show whether EP can cross this gap. EP should also be adapted to work on sequential data, which is not straightforward since the time is already used to reach a fixed point.

Neuromorphic computing is a complex topic involving physics, neuroscience, and machine learning. As shown in this thesis, it is fruitful to merge ideas from different fields. However, many motivations and goals can fall under the term 'neuromorphic computing'. It could be studying the brain by trying to reproduce it in silicon, accelerating neuroscience simulations, solving practical tasks more efficiently, or in a resource-constrained setting. For this reason, it can be difficult to identify a clear way forward in the field.

Given the state of the art of deep learning algorithms, valid short term objectives of neuromorphic hardware could be to reproduce the inference of deep neural networks algorithms into

dedicated hardware, especially since simpler architectures based on multi-layer perceptrons appear to be effective for large scale tasks [244]. However, low-power inference hardware will have to provide a significant edge over compressed models running on micro controller units [148], which may already achieve satisfying energy gains with a well-established technology. On a longer timescale, hardware-friendly credit assignment with strong theoretical guarantees will be key to bring the learning part of deep learning onto dedicated chips [61, 84, 95]. Given that successful models are trained with back-propagation, learning rules for dedicated hardware should not only be driven by locality, but also by guarantees with respect to the estimated gradients.

Ultimately, neuromorphic computing is by definition bounded by our understanding of the brain, especially at the circuit level. Hopefully, probing the brain will become easier with technological progress. Beyond the specific milestones reached by deep learning, powerful ideas have emerged for studying computational circuits, such as optimizing a cost function with a specific learning rule on a specific computational graph [66]. This framework, when coupled with biologically-plausible neural networks trained with surrogate gradients [62], and compared against experimental data, will be invaluable to understand the brain. In particular, taking advantage of end-to-end optimization to study cell types and circuit motifs could lead to the discovery of more powerful models.

In the long term, understanding the brain will require departing from the easily-quantified supervised learning to self-supervised learning. Current metrics of self-supervised learning consist in measuring the accuracy reached by transfer learning on downstream tasks, or comparing the learned representations to neural data through representational similarity analysis, a method based on correlations. These metrics have limitations because the choice of a downstream task is arbitrary, and correlations only capture linear dependencies. More appropriate and robust metrics will have to be designed.

In the future, successful neuromorphic chips are more likely to be heterogeneous and modular rather than monolithic. This will bring the challenge of understanding how to achieve efficient communications between the chips, an issue already identified by the pioneers of the field [5]. Studying how the brain encode information sparsely and efficiently using temporal coding will provide valuable answers. Fortunately, end-to-end optimization will be a powerful tool to find such encodings, provided that the right objective functions are optimized with the right models.

Deep learning was initially inspired by the brain, and its recent progress resulted in a framework that will be invaluable to study the brain. It would not be surprising to see a virtuous cycle where new knowledge about the brain provides ways to improve deep learning and facilitates the design of neuromorphic systems.

Synthèse en Français

Introduction et contexte

Le domaine de l'apprentissage machine qui consiste à faire apprendre des tâches intelligentes aux ordinateurs en fonction de données a connu une révolution dans la dernière décennie. Les ordinateurs sont désormais capables de réaliser des tâches cognitives de perception (vision, compréhension et traitement du langage naturel ...) avec une performance comparable aux humains, un sous domaine appelé 'apprentissage profond' qui utilise des réseaux de neurones artificiels inspirés du cerveau. Alors que les algorithmes qui sous-tendent cette révolution ont été inventés dans les années 1980-1990, la puissance de calcul et la quantité de donnée nécessaires pour réaliser des tâches de perception complexes ont vu le jour dans les années 2010.

Bien que ces algorithmes réalisent des tâches de perception similaires à ce dont sont capables les humains, leur consommation énergétique dépasse celle du cerveau par plusieurs ordre de grandeurs. Cette différence est en partie dûe à l'architecture des ordinateurs, appelée architecture de von Neumann, dans laquelle le processeur où les calculs sont effectués est séparé physiquement de la mémoire où les données sont stockées. L'apprentissage machine à grande échelle nécessite un déplacement de donnée incessant entre ces unités, causant un goulot d'étranglement énergétique. Au contraire, une simple observation du cerveau suggère que les neurones en tant qu'unité de calcul sont au plus proche de leur connexions synaptiques qui encodent la mémoire. La haute consommation énergétique de l'apprentissage profond limite l'exécution des réseaux de neurones artificiels à des ordinateurs suffisamment puissants, rendant impossible les applications embarquées ou sur des appareils disposant de peu de puissance.

Une solution pour réduire le coût énergétique de l'intelligence artificielle est de construire des architectures 'neuromorphiques', c'est à dire inspirées du cerveau humain où la mémoire est proche physiquement du calcul. Une possibilité est de concevoir une puce dédiée à l'inférence, c'est à dire que l'apprentissage est fait sur un ordinateur classique et le modèle résultant est encodé dans une puce dédiée. Un objectif plus long terme est de concevoir des architectures prenant en charge l'apprentissage à partir des données. Deux défis complémentaires apparaissent :

- **Défi 1:** Trouver des composants mémoires adaptés pour émuler les algorithmes de réseaux de neurones artificiels. Les caractéristiques importantes sont l'encodage (digital ou analogue), la volatilité (persistence en absence d'alimentation), la variabilité entre les composants, la cyclabilité (le nombre de réécritures possibles).

- **Défi 2:** Concevoir des algorithmes de réseaux de neurones artificiels mieux adaptés à la conception de circuit avec des composants dédiés. En effet, les algorithmes d'inférence et d'apprentissage ont des contraintes plus ou moins simples à satisfaire avec des composants dédiés consommant peu d'énergie.

Dans cette thèse, je présente des travaux qui apportent des solutions à ces deux défis. Dans le chapitre 2, je présente une procédure pour réduire l'oubli catastrophique dans les réseaux de neurones binaires, publié dans Nature Communications [1]. Dans le chapitre 3, je montre comment un algorithme d'apprentissage alternatif et plus adapté aux contraintes hardware peut passer à l'échelle sur une tâche de vision complexe, en collaboration avec le Mila, publié dans Frontiers in Neuroscience [2]. Dans le chapitre 4, je présente en collaboration avec le CEA Leti et l'université Aix Marseille une architecture pour encoder des réseaux de neurones ternaires avec des composants mémoires émergents, publié à la conférence AICAS 2020 [3] et en version étendue dans TCAS I [4].

Résultats

Chapitre 2 Dans ce chapitre, je présente un moyen de réduire l'oubli catastrophique dans les réseaux de neurones binaires pour la conception de systèmes neuromorphiques. L'oubli catastrophique désigne une limitation des réseaux de neurones artificiels qui empêche l'apprentissage successif de plusieurs tâches. Si on veut qu'un réseau apprenne à reconnaître des chiffres manuscrits, on peut l'entrainer sur une base de données d'exemples de chiffres. Si ensuite on souhaite utiliser le même réseau pour apprendre une seconde tâche, par exemple reconnaître des lettres, le réseau apprendra cette seconde tâche au prix d'un oubli quasiment instantané des chiffres. Cette caractéristique très différente du cerveau humain est aussi une limitation majeure pour le déploiement des réseaux de neurones, notamment dans un contexte embarqué. On souhaiterait qu'un réseau puisse continuer à apprendre lorsque de nouvelles données deviennent disponibles ou qu'une nouvelle tâche survient, et ce sans avoir à apprendre à nouveau les anciennes données. Le problème de l'oubli catastrophique s'explique par le fait qu'un réseau de neurones doit modifier ses paramètres pour apprendre une tâche, mais aussi empêcher les paramètres de bouger de manière à protéger ses souvenirs, ce qui semble contradictoire. Dans le cerveau, les synapses sont plastiques, c'est à dire que leur efficacité peut être modifiée pour réaliser un apprentissage, mais elles sont aussi métaplastiques : des mécanismes biologiques existent pour rendre la synapse plus ou moins plastique. Des modèles de neuroscience computationelle ont montré que la métaplasticité des synapses peut s'expliquer par

l'introduction de variables cachées en plus de l'efficacité synaptique. Ces variables cachées changent au fil de l'apprentissage et consolident les synapses. Cependant les solutions proposées dans le domaine de l'apprentissage profond sont très différentes dans leur implémentation, et peu adaptées à la conception de systèmes neuromorphiques, car elles induisent des calculs supplémentaires et non locaux en espace pour déterminer l'importance des différents paramètres pour la tâche précédente. Le problème de l'oubli catastrophique est partagé par tous les réseaux de neurones, y compris les réseaux de neurones binaires.

Les réseaux de neurones binaires sont des réseaux de neurones artificiels basse précision, n'utilisant que des valeurs binaires pour les activations neuronales et les paramètres synaptiques. Ces réseaux de neurones sont donc très étudiés pour la conception de systèmes à basse consommation énergétique car ils consomment moins de mémoire tout en étant presque aussi performants que les réseaux de neurones utilisant une précision de 32 bits. Toutefois, le processus d'apprentissage des réseaux de neurones binaires nécessite une variable cachée par connexion synaptique. Cette variable cachée est mise à jour à chaque itération d'apprentissage par le gradient de la fonction à optimiser, évaluée en les valeurs binaires des paramètres. Dans ce chapitre, je montre que cette variable cachée peut être interprétée comme une variable utile pour la métaplasticité et l'apprentissage continu sans calcul supplémentaire. Ce travail crée un lien entre les modèles de neurosciences computationelles et d'apprentissage continu en apprentissage profond.

Dans un premier temps, je propose une modification du processus d'apprentissage des réseaux de neurones binaires pour consolider les synapses qui ont une variable cachée élevée, en accord avec les principes développés en neuroscience computationelle. Je montre ensuite expérimentalement que cette modification simple permet de réduire l'oublie catastrophique des réseaux de neurones binaires quand plusieurs tâches sont apprises successivement. Les séquences de tâches étudiées sont des versions permutées des chiffres manuscrits, standard dans l'apprentissage continu, et aussi des séquences plus compliquées comme chiffres et vêtements. Je montre que ma méthode obtient des résultats similaires à la méthode Elastic Weight Consolidation (EWC) développée pour réduire l'oubli catastrophique en calculant les facteurs d'importance de chaque synapse. Contrairement à EWC, ma méthode ne nécessite pas de faire de pause entre les tâches pour calculer les coefficients d'importance, ce qui est favorable dans un contexte embarqué et basse consommation énergétique.

Grâce à cet avantage, j'explore ensuite un contexte d'apprentissage continu non exploré par les autres méthodes que j'appelle 'stream learning'. Dans ce contexte, une seule tâche est apprise, mais en apprenant successivement des fractions de l'ensemble total des données. Ce contexte est pertinent dans un contexte embarqué où une base de donnée entière ne peut être stockée. J'étudie dans un premier temps une base de donnée constituée d'images de vêtements (Fashion MNIST). Je montre que le réseau de neurones binaire métaplastique entrainé séquentiellement peut atteindre une performance similaire à un réseau non métaplastique entrainé

sur tout l'ensemble des données, alors que le réseau non métaplastique n'arrive pas à apprendre séquentiellement à cause de l'oublie catastrophique. Je montre ensuite que cette propriété persiste dans le cas d'un apprentissage plus compliqué d'images naturelles (CIFAR-10).

Ensuite, j'ai étudié une version simplifiée de l'optimisation binaire dans laquelle des calculs mathématiques sont possibles. La fonction optimisée dans cet exemple est une fonction quadratique convexe. Des résultats mathématiquement prouvés montrent que la variable cachée est une quantité corrélée à l'importance du paramètre binaire associé. L'importance du paramètre étant défini comme l'augmentation de la fonction à minimiser lorsque le paramètre est changé en son opposé. Je montre ensuite par des simulations que cette interpration de l'importance est vérifiée dans un réseau de neurone binaire entrainé sur MNIST.

La méthode introduite permet d'apprendre plusieurs tâches, mais les synapses finissent par être toutes consolidées de sorte qu'il devient impossible d'apprendre de nouvelles tâches. Pour obtenir un état d'apprentissage continu stationnaire, j'adapte un modèle de synapse plus complexe issue des neurosciences computationelles avec plusieurs variables cachées. Ce modèle contient un mécanisme de déconsolidation graduel qui permet d'atteindre un état stationnaire tout en limitant l'oubli catastrophique.

Ces résultats constituent un pont entre différentes études de l'oubli catastrophique dans l'apprentissage machine et les neurosciences. Ce chapitre propose une solution au défi 2 mentionné en introduction. En effet, la méthode présentée prend en compte les contraintes hardware pour la conception de systèmes neuromorphiques car la règle de consolidation ne dépend que de l'état interne de la synapse obtenu par la processus d'apprentissage. Dans de futurs travaux, des composants magnétiques contrôlables par champs électrique seront utilisés pour prouver expérimentalement la réalisation de synapses métaplastiques.

Chapitre 3 Ce chapitre est le résultat d'une collaboration avec le Mila. Les réseaux de neurones artificiels réalisent des tâches cognitives en apprenant à partir de données. Plus précisément, une fonction qui quantifie la qualité de la prédiction associée à un exemple est définie. Cette fonction est optimisée en changeant les paramètres du réseau de neurone au cours de l'apprentissage. Les paramètres sont changés en utilisant le gradient de la fonction à optimiser. Ce gradient est calculé en pratique en utilisant la rétro propagation du gradient. Cependant, la rétro propagation du gradient est problématique d'un point de vue biologique car elle suppose que les neurones peuvent réaliser deux types de calcul distincts. Un autre problème est l'hypothèse que les synapses sont bidirectionnelles. Ces problèmes rendent difficiles l'implémentation de la rétro propagation du gradient sur une puce dédiée à l'apprentissage.

Equilibrium Propagation (EP) est un algorithme introduit en 2017 pour calculer le gradient de la fonction à optimiser de manière plus plausible biologiquement. EP permet d'estimer le gradient avec un seul type de calcul neuronal dans un réseau de neurones à base d'énergie. Plus généralement, EP peut être utilisé dans n'importe quel système physique dans lequel une fonc-

tion d'énergie peut être définie. Un autre avantage d'EP est l'existence de garanties théoriques quant à l'estimation du gradient. Cependant, EP n'a été employé que sur des tâches simples (reconnaissance de chiffres manuscrits), et sur des réseaux peu profonds. La fonction optimisée par EP dans sa formulation originale est le coût quadratique, mais n'est pas la fonction la plus adaptée pour des problèmes de classification.

Dans ce chapitre, nous montrons que la formulation originale d'EP estime le gradient de manière biaisée, ce qui empêche l'utilisation d'EP dans des architectures plus profondes. Nous proposons un nouvel estimateur qui permet de réduire ce biais. Avec ce nouvel estimateur, nous montrons pour la première fois que EP peut entrainer des réseaux de neurones profonds sur des images naturelles (CIFAR-10). La performance atteinte par EP est proche de celle atteinte par la rétro propagation du gradient. Dans un deuxième temps, nous proposons un moyen d'optimiser l'entropie croisée, qui est plus adaptée à la classification que le coût quadratique. Cette méthode permet d'augmenter la performance de classification du réseau entrainé par EP sur CIFAR-10. Enfin, nous étudions le cas où les synapses ne sont pas bidirectionnelles. Bien qu'une généralisation de EP existe pour ce cas de figure, la garantie théorique quant à l'estimation du gradient n'est plus valable, et l'optimisation est instable en pratique sur des réseaux de neurones profonds. Nous montrons qu'ajouter un mécanisme simple d'alignement entre les paramètres de direction opposée permet d'entrainer le réseau malgré une légère baisse de performance.

Ces résultats apportent une solution au défi 2 mentionné en introduction car ils montrent qu'un algorithme d'apprentissage plus adapté à la conception de cicruit que la rétro propagation du gradient peut passer à l'échelle sur des tâches complexes. Concevoir des circuits implémentant EP grâce à la physique du système ou explicitement constitue une piste de recherche sérieuse pour le calcul neuromorphique.

Chapitre 4 Ce chapitre est le résultat d'une collaboration avec le CEA-Leti et Aix-Marseille université. Nous proposons une implémentation de réseau de neurone ternaire, c'est à dire trois valeurs (-1, 0, 1) pour les paramètres synaptiques et les activations neuronales, en utilisant des mémoires résistives émergentes (RRAM). Nous proposons une architecture hybride entre CMOS et RRAM avec deux transistors et deux résistances dans laquelle le paramètre peut être mesurée avec une seule opération de détection en utilisant un amplificateur de détection à pré-charge (PCSA). Ce circuit avait été utilisé dans des travaux précédents pour implémenter un réseau de neurone binaire. Dans ce travail précédent, les paramètres binaires (-1 et 1) étaient encodés par des paires de RRAM codées de manière différentielle. Une RRAM peut être programmée dans deux états : un état de haute résistance (HRS) et un état de basse résistance (LRS). Le travail précédent utilisait une paire HRS/LRS pour encoder un 1 et LRS/HRS pour encoder un -1, ce qui permettait de réduire les erreurs de lectures. Dans ce travail, nous proposons une méthode pour implémenter un réseau de neurone ternaire avec ce même circuit en exploitant un régime où la tension d'alimentation est basse, ce qui est particulièrement

adapté pour les applications embarquées.

L'architecture proposée est d'abord étudiée avec des simulations SPICE. Nous montrons en simulations que lorsque la tension d'alimentation est basse, l'opération du PCSA permet de différencier les paires HRS/LRS et LRS/HRS, mais aussi de différencier la paire HRS/HRS, qui peut donc être utilisée pour encoder la troisième valeur de paramètre : 0. Nous montrons ensuite expérimentalement la réalisation pratique de cette puce en utilisant une technologie 130 nm. Nous montrons que l'on peut effectivement encoder les trois valeurs de paramètres utilisées dans un réseau de neurones ternaire. Enfin, des simulations de réseaux de neurones ternaires sont faites pour quantifier le gain en performance obtenue en passant de réseaux binaires à ternaires sur une tâche de vison (CIFAR-10). Les erreurs d'encodage inhérentes aux RRAM utilisées dans le circuit sont incorporées dans les simulations de réseaux de neurones pour étudier l'impact de celles-ci. Nous montrons que les erreurs de lecture où un paramètre est changé en son opposé (-1 devient 1 ou inversement) restent rares et sans impact, comme dans l'architecture proposée pour les réseaux de neurones binaires. Cependant, un nouveau type d'erreur survient dans cette architecture lorsqu'un 0 est lu comme un 1 ou un -1. Les simulations de réseaux de neurones ternaires avec ces nouvelles erreurs incorporées restent plus performants que les réseaux de neurones binaires de même taille, ce qui valide notre approche.

Ces résultats sont un exemple de solution au défi 1 mentionné en introduction, car nous proposons une architecture à base de composants émergents pour implémenter un algorithme pré-existant (réseau de neurone ternaire).

Perspectives

Dans cette thèse, j'ai présenté des résultats sur des algorithmes d'apprentissage qui permettent de mieux définir les besoins des architectures neuromorphiques en terme de composants. J'ai aussi montré comment des mémoires émergentes peuvent être utilisées pour concevoir des circuits dédiés à l'inférence. Dans les travaux futurs, cette logique de co-développement entre composants et algorithmes devra être au centre de la recherche pour aboutir à la conception de circuits intelligents consommant peu d'énergie. Bien que les objectifs d'un circuit dédié à l'IA et du cerveau humain ne soient pas identiques, une meilleure compréhension des mécanismes d'apprentissage dans le cerveau ne sera que bénéfique à la conception de circuit. Quelles sont les fonctions optimisées par le cerveau ? Quels schémas architecturaux ou micro-cricuits permettent une implémentation efficace ? Et enfin, comment le cerveau orchestre un apprentissage cohérent sur des connexions synaptiques dispersées dans l'espace tout en utilisant des calculs locaux ? Des réponses à ces questions aideront non seulement l'apprentissage profond, mais aussi la conception de circuits neuromorphiques.

List of publications

Peer-Reviewed Journal Articles

✠ **Axel Laborieux**, Maxence Ernoult, Tifenn Hirtzlin and Damien Querlioz, "Synaptic Metaplasticity in Binarized Neural Networks" , *Nature Communications*, vol. 12, p. 1–12, 2021. doi:10.1038/s41467-021-22768-y

✠ **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*, vol. 15, p. 129, 2021. doi:10.3389/fnins.2021.633674

✠ **Axel Laborieux**, Marc Bocquet, Tifenn Hirtzlin, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal and Damien Querlioz, "Implementation of Ternary Weights With Resistive RAM Using a Single Sense Operation Per Synapse" , *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, p. 138–147, 2020. doi:10.1109/TCSI.2020.3031627

✠ L. Herrera Diez, Y.T. Liu, D.A. Gilbert, M. Belmeguenai, J. Vogel, S. Pizzini, E. Martinez, A. Lamperti, J.B. Mohammedi, **A. Laborieux**, Y. Roussigné, A.J. Grutter, E. Arenholtz, P. Quarterman, B. Maranville, S. Ono, M. Salah El Hadri, R. Tolley, E.E. Fullerton, L. Sanchez-Tejerina, A. Stashkevich, S.M. Chérif, A.D. Kent, D. Querlioz, J. Langer, B. Ocker and D. Ravelosona, "Nonvolatile Ionic Modification of the Dzyaloshinskii-Moriya Interaction" , *Physical Review Applied*, vol. 12, No. 3, p. 034005, 2019. doi:10.1103/PhysRevApplied.12.034005

Peer-Reviewed Conference Proceedings

✠ Fadi Jebali, Atreya Majumdar, **Axel Laborieux**, Tifenn Hirtzlin, Elisa Vianello, Jean-Pierre Walder, Marc Bocquet, Damien Querlioz and Jean-Michel Portal, "CAPC: A Configurable Analog Pop-Count Circuit for Near-Memory Binary Neural Networks" , *64th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, ***Accepted.***

✠ **AXEL LABORIEUX**, MARC BOCQUET, TIFENN HIRTZLIN, JACQUES-OLIVIER KLEIN, LIZA HERRERA-DIEZ, ETIENNE NOWAK, ELISA VIANELLO, JEAN-MICHEL PORTAL and DAMIEN QUERLIOZ, "Low Power In-Memory Implementation of Ternary Neural Networks with Resistive RAM-Based Synapse" , *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, p. 136–140, 2020. ***2nd best paper award.***

doi:10.1109/AICAS48895.2020.9073877

✠ L. HERRERA DIEZ, Y.T. LIU, D.A. GILBERT, M. BELMEGUENAI, J. VOGEL, S. PIZZINI, E. MARTINEZ, A. LAMPERTI, J.B. MOHAMMEDI, **A. LABORIEUX**, Y. ROUSSIGNÉ, A.J. GRUTTER, E. ARENHOLTZ, P. QUARTERMAN, B. MARANVILLE, S. ONO, M. SALAH EL HADRI, R. TOLLEY, E.E. FULLERTON, L. SANCHEZ-TEJERINA, A. STASHKEVICH, S.M. CHÉRIF, A.D. KENT, D. QUERLIOZ, J. LANGER, B. OCKER and D. RAVELOSONA, "Electric field control of magnetism" , *Spintronics XIII*, vol. 11470, p. 114703G, 2020.

doi:10.1117/12.2567644

Conferences Without Proceedings

✠ **AXEL LABORIEUX**, MAXENCE ERNOULT, TIFENN HIRTZLIN and DAMIEN QUERLIOZ, "Synaptic Metaplasticity in Binarized Neural Networks" , *CVPR 2021 : Workshop on Binary Networks, Poster presentation.*

✠ **AXEL LABORIEUX**, MAXENCE ERNOULT, TIFENN HIRTZLIN and DAMIEN QUERLIOZ, "Synaptic Metaplasticity in Binarized Neural Networks" , *Cosyne 2021, Poster presentation.*

✠ **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" , *NeurIPS 2020 Workshop : Beyond Backpropagation Novel Ideas for Training Neural Architectures, Poster presentation.*

✠ **AXEL LABORIEUX**, TIFENN HIRTZLIN, LIZA HERRERA-DIEZ and DAMIEN QUERLIOZ, "Memory Effects in Metaplastic Binarized Neural Networks" , *Ecole d'été : X Data Science Summer School (XDS3) 2019, Poster presentation.*

Bibliography

[1] Axel Laborieux, Maxence Ernoult, Tifenn Hirtzlin, and Damien Querlioz. Synaptic metaplasticity in binarized neural networks. *arXiv preprint arXiv:2003.03533*, 2020.

[2] 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*, 15:129, 2021.

[3] Axel Laborieux, Marc Bocquet, Tifenn Hirtzlin, Jacques-Olivier Klein, L Herrera Diez, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Low power in-memory implementation of ternary neural networks with resistive ram-based synapse. In *2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020.

[4] Axel Laborieux, Marc Bocquet, Tifenn Hirtzlin, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Implementation of ternary weights with resistive ram using a single sense operation per synapse. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(1):138–147, 2020.

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

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[7] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[8] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

[9] C. Mead and L. Conway. Introduction to vlsi systems. In *Introduction to VLSI systems*, 1978.

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

[11] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[12] C. Mead. Analog vlsi and neural systems. In *Analog VLSI and neural systems*, 1989.

[13] R. Lyon and C. Mead. An analog electronic cochlea. *IEEE Trans. Acoust. Speech Signal Process.*, 36:1119–1134, 1988.

[14] Misha Mahowald and Rodney Douglas. A silicon neuron. *Nature*, 354(6354):515–518, 1991.

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

[16] 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.

[17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[18] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[19] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.

[20] Donald Olding Hebb. The organization of behavior; a neuropsycholocigal theory. *A Wiley Book in Clinical Psychology*, 62:78, 1949.

[21] 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.

[22] Dmitry Krotov and John J Hopfield. Dense associative memory for pattern recognition. *arXiv preprint arXiv:1606.01164*, 2016.

[23] Dmitry Krotov and John Hopfield. Large associative memory problem in neurobiology and machine learning. *arXiv preprint arXiv:2008.06996*, 2020.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[25] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

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

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

[28] 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.

[29] 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.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] 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.

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[34] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[35] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[36] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. Jax: composable transformations of python+numpy programs, 2018.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

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

[40] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[41] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[42] 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.

[43] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *arXiv preprint arXiv:1605.06431*, 2016.

[44] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[45] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training, 2021.

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[47] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[48] 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.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[50] 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.

[51] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

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

[53] 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.

[54] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 181–196, 2018.

[55] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[56] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[57] 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.

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

[59] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

[60] 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.

[61] 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.

[62] 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.

[63] Bojian Yin, Federico Corradi, and Sander M Bohté. Effective and efficient computation with multiple-timescale spiking recurrent neural networks. In *International Conference on Neuromorphic Systems 2020*, pages 1–8, 2020.

[64] 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 Comput Biol*, 10(12):e1003963, 2014.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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. *bioRxiv*, page 738385, 2020.

[70] Alexandre Payeur, Jordan Guerguiev, Friedemann Zenke, Blake A Richards, and Richard Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nature Neuroscience*, pages 1–10, 2021.

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

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

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

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

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

[76] Geoffrey E Hinton and James L McClelland. Learning representations by recirculation. In *Neural information processing systems*, volume 1987, pages 358–366, 1988.

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

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

[79] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998. *URL http://yann. lecun. com/exdb/mnist*, 10:34, 1998.

[80] 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. In *Advances in Neural Information Processing Systems*, pages 7081–7091, 2019.

[81] 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.

[82] 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):1–10, 2016.

[83] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045, 2016.

[84] 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. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

[85] Qianli Liao, Joel Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[86] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[87] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. *arXiv preprint arXiv:1811.03567*, 2018.

[88] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[89] Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, pages 974–982, 2019.

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

[91] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

[92] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, pages 1–12, 2020.

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

[94] James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5):1229–1262, 2017.

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

[96] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. *arXiv preprint arXiv:1810.11393*, 2018.

[97] Beren Millidge, Alexander Tschantz, and Christopher L Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *arXiv preprint arXiv:2006.04182*, 2020.

[98] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[99] Neil Burgess, JL Shapiro, and MA Moore. Neural network models of list learning. *Network: Computation in Neural Systems*, 2(4):399–422, 1991.

[100] JP Nadal, G Toulouse, JP Changeux, and S Dehaene. Networks of formal neurons and memory palimpsests. *EPL (Europhysics Letters)*, 1(10):535, 1986.

[101] Michael McCloskey and Neal J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 1989.

[102] Stephan Lewandowsky and Shu-Chen Li. Catastrophic interference in neural networks: Causes, solutions, and data. In *Interference and inhibition in cognition*, pages 329–361. Elsevier, 1995.

[103] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

[104] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[105] Wickliffe C Abraham and Anthony Robins. Memory retention–the synaptic stability versus plasticity dilemma. *Trends in neurosciences*, 28(2):73–78, 2005.

[106] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[107] Stefano Fusi, Patrick J Drew, and Larry F Abbott. Cascade models of synaptically stored memories. *Neuron*, 45(4):599–611, 2005.

[108] Daniel J Amit and Stefano Fusi. Constraints on learning in dynamic synapses. *Network: Computation in Neural Systems*, 3(4):443–464, 1992.

[109] David S Bredt and Roger A Nicoll. Ampa receptor trafficking at excitatory synapses. *Neuron*, 40(2):361–379, 2003.

[110] Joshua R Sanes and Jeff W Lichtman. Can molecules explain long-term potentiation? *Nature neuroscience*, 2(7):597–604, 1999.

[111] Richard D Emes, Andrew J Pocklington, Christopher NG Anderson, Alex Bayes, Mark O Collins, Catherine A Vickers, Mike DR Croning, Bilal R Malik, Jyoti S Choudhary, J Douglas Armstrong, et al. Evolutionary expansion and anatomical specialization of synapse proteome complexity. *Nature neuroscience*, 11(7):799, 2008.

[112] John T Wixted and Ebbe B Ebbesen. On the form of forgetting. *Psychological science*, 2(6):409–415, 1991.

[113] John T Wixted and Ebbe B Ebbesen. Genuine power curves in forgetting: A quantitative analysis of individual subject forgetting functions. *Memory & cognition*, 25(5):731–739, 1997.

[114] Wickliffe C Abraham and Mark F Bear. Metaplasticity: the plasticity of synaptic plasticity. *Trends in neurosciences*, 19(4):126–130, 1996.

[115] Wickliffe C Abraham. Metaplasticity: tuning synapses and networks for plasticity. *Nature Reviews Neuroscience*, 9(5):387–387, 2008.

[116] Marcus K Benna and Stefano Fusi. Computational principles of synaptic memory consolidation. *Nature neuroscience*, 19(12):1697–1706, 2016.

[117] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradientbased neural networks. In *In Proceedings of International Conference on Learning Representations (ICLR)*, 2014.

[118] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[119] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[120] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[121] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[122] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[123] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[124] Ferenc Huszár. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences*, page 201717042, 2018.

[125] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[126] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[127] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[128] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[129] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[130] Guillaume Hocquet, Olivier Bichler, and Damien Querlioz. Ova-inn: Continual learning with invertible neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[131] Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

[132] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[133] Charlotte Frenkel, David Bol, and Giacomo Indiveri. Bottom-up and top-down neural processing systems design: Neuromorphic intelligence as the convergence of natural and artificial intelligence. *arXiv preprint arXiv:2106.01288*, 2021.

[134] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[135] Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of neurophysiology*, 94(5):3637–3642, 2005.

[136] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 18(24):10464–10472, 1998.

[137] Joseph M Brader, Walter Senn, and Stefano Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation*, 19(11):2881–2912, 2007.

[138] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[139] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141, 2015.

[140] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017.

[141] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950. IEEE, 2010.

[142] Eustace Painkras, Luis A Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, and Steve B Furber. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

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

[144] 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.

[145] 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.

[146] Kenneth Stewart, Garrick Orchard, Sumit Bam Shrestha, and Emre Neftci. On-chip few-shot learning with surrogate gradient descent on a neuromorphic processor. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 223–227. IEEE, 2020.

[147] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[148] Ji Lin, Wei-Ming Chen, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[149] Daniele Ielmini and Rainer Waser. *Resistive switching: from fundamentals of nanoionic redox processes to memristive device applications*. John Wiley & Sons, 2015.

[150] Mirko Prezioso et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61, 2015.

[151] F Merrikh Bayat, Mirko Prezioso, Bhaswar Chakrabarti, H Nili, I Kataeva, and D Strukov. Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. *Nature communications*, 9(1):1–7, 2018.

[152] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M Shelby, Irem Boybat, Carmelo di Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan CP Farinha, et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60–67, 2018.

[153] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[154] Tifenn Hirtzlin, Marc Bocquet, Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Frontiers in Neuroscience*, 13:1383, 2020.

[155] Tifenn Hirtzlin, Marc Bocquet, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Outstanding bit error tolerance of resistive ram-based binarized neural networks. *arXiv preprint arXiv:1904.03652*, 2019.

[156] Tifenn Hirtzlin, Bogdan Penkovsky, Marc Bocquet, Jacques-Olivier Klein, Jean-Michel Portal, and Damien Querlioz. Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access*, 7:76394, 2019.

[157] 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.

[158] Alaa Saade, Francesco Caltagirone, Igor Carron, Laurent Daudet, Angélique Drémeau, Sylvain Gigan, and Florent Krzakala. Random projections through multiple optical scattering: Approximating kernels at the speed of light. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6215–6219. IEEE, 2016.

[159] Y. T. Liu, S. Ono, G. Agnus, J. P. Adam, S. Jaiswal, J. Langer, B. Ocker, D. Ravelosona, and L. Herrera Diez. Electric field controlled domain wall dynamics and magnetic easy axis switching in liquid gated CoFeB/MgO films. *Journal of Applied Physics*, 2017.

[160] L Herrera Diez, YT Liu, Dustin Allen Gilbert, M Belmeguenai, J Vogel, S Pizzini, E Martinez, A Lamperti, JB Mohammedi, A Laborieux, et al. Nonvolatile ionic modification of the dzyaloshinskii-moriya interaction. *Physical Review Applied*, 12(3):034005, 2019.

[161] JE Hirsch. Spin hall effect. *Physical review letters*, 83(9):1834, 1999.

[162] Uwe Bauer, Lide Yao, Aik Jun Tan, Parnika Agrawal, Satoru Emori, Harry L. Tuller, Sebastiaan Van Dijken, and Geoffrey S.D. Beach. Magneto-ionic control of interfacial magnetism. *Nature Materials*, 2015.

[163] Axel Laborieux, Maxence Ernoult, Tifenn Hirtzlin, and Damien Querlioz. Synaptic metaplasticity in binarized neural networks. *Nature communications*, 12(1):1–12, 2021.

[164] Christos Kaplanis, Murray Shanahan, and Claudia Clopath. Continual reinforcement learning with complex synapses. *arXiv preprint arXiv:1802.07239*, 2018.

[165] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[166] Subhaneil Lahiri and Surya Ganguli. A memory frontier for complex synapses. *Advances in neural information processing systems*, 26:1034–1042, 2013.

[167] Francesco Conti, Pasquale Davide Schiavone, and Luca Benini. Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2940–2951, 2018.

[168] Daniel Bankman, Lita Yang, Bert Moons, Marian Verhelst, and Boris Murmann. An always-on 3.8μj/86% cifar-10 mixed-signal binary cnn processor with all memory on chip in 28-nm cmos. *IEEE Journal of Solid-State Circuits*, 54(1):158–172, 2018.

[169] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 345–353, 2017.

[170] Bogdan Penkovsky, Marc Bocquet, Tifenn Hirtzlin, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. In-memory resistive ram implementation of binarized neural networks for medical applications. In *Design, Automation and Test in Europe Conference (DATE)*, 2020.

[171] Daniel J Amit and Stefano Fusi. Learning in neural networks with material synapses. *Neural Computation*, 6(5):957–982, 1994.

[172] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[173] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[174] Marcus K Benna and Stefano Fusi. Efficient online learning with low-precision synaptic variables. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 1610–1614. IEEE, 2017.

[175] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. In *Advances in neural information processing systems*, pages 7533–7544, 2019.

[176] Xiangming Meng, Roman Bachmann, and Mohammad Emtiyaz Khan. Training binary neural networks using the bayesian learning rule. *arXiv preprint arXiv:2002.10778*, 2020.

[177] Sören Boyn, Julie Grollier, Gwendal Lecerf, Bin Xu, Nicolas Locatelli, Stéphane Fusil, Stéphanie Girod, Cécile Carrétéro, Karin Garcia, Stéphane Xavier, et al. Learning through

ferroelectric domain dynamics in solid-state synapses. *Nature communications*, 8(1):1–7, 2017.

[178] Miguel Romera, Philippe Talatchian, Sumito Tsunegi, Flavio Abreu Araujo, Vincent Cros, Paolo Bortolotti, Juan Trastoy, Kay Yakushiji, Akio Fukushima, Hitoshi Kubota, Shinji Yuasa, Maxence Ernoult, Damir Vodenicarevic, Tifenn Hirtzlin, Nicolas Locatelli, Damien Querlioz, and Julie Grollier. Vowel recognition with four coupled spin-torque nano-oscillators. *Nature*, 563(7730):230, 2018.

[179] Jacob Torrejon et al. Neuromorphic computing with nanoscale spintronic oscillators. *Nature*, 547(7664):428, 2017.

[180] Quantan Wu, Hong Wang, Qing Luo, Writam Banerjee, Jingchen Cao, Xumeng Zhang, Facai Wu, Qi Liu, Ling Li, and Ming Liu. Full imitation of synaptic metaplasticity based on memristor devices. *Nanoscale*, 10(13):5875–5881, 2018.

[181] Xiaojian Zhu, Chao Du, YeonJoo Jeong, and Wei D Lu. Emulation of synaptic metaplasticity in memristors. *Nanoscale*, 9(1):45–51, 2017.

[182] Tae-Ho Lee, Hyun-Gyu Hwang, Jong-Un Woo, Dae-Hyeon Kim, Tae-Wook Kim, and Sahn Nahm. Synaptic plasticity and metaplasticity of biological synapse realized in a knbo3 memristor for application to artificial synapse. *ACS applied materials & interfaces*, 10(30):25673–25682, 2018.

[183] Bo Liu, Zhiwei Liu, In-Shiang Chiu, MengFu Di, YongRen Wu, Jer-Chyi Wang, Tuo-Hung Hou, and Chao-Sung Lai. Programmable synaptic metaplasticity and below femtojoule spiking energy realized in graphene-based neuromorphic memristor. *ACS applied materials & interfaces*, 10(24):20237–20243, 2018.

[184] Zheng-Hua Tan, Rui Yang, Kazuya Terabe, Xue-Bing Yin, Xiao-Dong Zhang, and Xin Guo. Synaptic metaplasticity realized in oxide memristive devices. *Advanced Materials*, 28(2):377–384, 2016.

[185] David Kappel, Stefan Habenschuss, Robert Legenstein, and Wolfgang Maass. Network plasticity as bayesian inference. *PLoS Comput Biol*, 11(11):e1004485, 2015.

[186] Editorial. Big data needs a hardware revolution. *Nature*, 554(7691):145, February 2018.

[187] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[188] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[189] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online normalization for training neural networks. In *Advances in Neural Information Processing Systems*, pages 8433–8443, 2019.

[190] Damien Querlioz, Olivier Bichler, Adrien Francis Vincent, and Christian Gamrat. Bioinspired programming of memory devices for implementing an inference engine. *Proceedings of the IEEE*, 103(8):1398–1416, 2015.

[191] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[192] Persi Diaconis and Mehrdad Shahshahani. The subgroup algorithm for generating uniform random variables. *Probability in the engineering and informational sciences*, 1(1):15–32, 1987.

[193] Fernando J Pineda. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1(2):161–172, 1989.

[194] 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.

[195] Benjamin Scellier and Yoshua Bengio. Equivalence of equilibrium propagation and recurrent backpropagation. *Neural computation*, 31(2):312–329, 2019.

[196] Luis B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings of the IEEE First International Conference on Neural Networks* (San Diego, CA), volume II, pages 609–618. Piscataway, NJ: IEEE, 1987.

[197] Fernando J. Pineda. Generalization of Back-Propagation to Recurrent Neural Networks. *Physical Review Letters*, 59(19):2229–2232, November 1987.

[198] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Equilibrium propagation with continual weight updates. *arXiv preprint arXiv:2005.04168*, 2020.

[199] Gianluca Zoppo, Francesco Marrone, and Fernando Corinto. Equilibrium propagation for memristor-based recurrent neural networks. *Frontiers in neuroscience*, 14:240, 2020.

[200] Erwann Martin, Maxence Ernoult, Jérémie Laydevant, Shuai Li, Damien Querlioz, Teodora Petrisor, and Julie Grollier. Eqspike: Spike-driven equilibrium propagation for neuromorphic implementations. *arXiv preprint arXiv:2010.07859*, 2020.

[201] Zhengyun Ji and Warren Gross. Towards efficient on-chip learning using equilibrium propagation. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[202] Armin Najarpour Foroushani, Hussein Assaf, Fereidoon Hashemi Noshahr, Yvon Savaria, and Mohamad Sawan. Analog circuits to accelerate the relaxation process in the equilibrium propagation algorithm. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[203] Peter O'Connor, Efstratios Gavves, and Max Welling. Initialized equilibrium propagation for backprop-free training. 2018.

[204] Peter O'Connor, Efstratios Gavves, and Max Welling. Training a spiking neural network with equilibrium propagation. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1516–1523, 2019.

[205] Yann Lecun. Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models). 1987.

[206] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pages 690–701, 2019.

[207] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. *arXiv preprint arXiv:2006.08656*, 2020.

[208] John F Kolen and Jordan B Pollack. Backpropagation without weight transport. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 3, pages 1375–1380. IEEE, 1994.

[209] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[210] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[211] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[212] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[213] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1(4):216, 2018.

[214] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[215] M. Bocquet, T. Hirztlin, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz. In-memory and error-immune differential rram implementation of binarized deep neural networks. In *IEDM Tech. Dig.*, page 20.6.1. IEEE, 2018.

[216] Shimeng Yu, Zhiwei Li, Pai-Yu Chen, Huaqiang Wu, Bin Gao, Deli Wang, Wei Wu, and He Qian. Binary neural network with 16 mb rram macro chip for classification and online training. In *IEDM Tech. Dig.*, pages 16–2. IEEE, 2016.

[217] Edouard Giacomin, Tzofnat Greenberg-Toledo, Shahar Kvatinsky, and Pierre-Emmanuel Gaillardon. A robust digital rram-based convolutional block for low-power image processing and learning applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(2):643–654, 2019.

[218] Xiaoyu Sun, Shihui Yin, Xiaochen Peng, Rui Liu, Jae-sun Seo, and Shimeng Yu. Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks. *algorithms*, 2:3, 2018.

[219] Z Zhou, P Huang, YC Xiang, WS Shen, YD Zhao, YL Feng, B Gao, HQ Wu, H Qian, LF Liu, et al. A new hardware implementation approach of bnns based on nonlinear 2t2r synaptic cell. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 20–7. IEEE, 2018.

[220] Masanori Natsui, Tomoki Chiba, and Takahiro Hanyu. Design of mtj-based nonvolatile logic gates for quantized neural networks. *Microelectronics journal*, 82:13–21, 2018.

[221] Tianqi Tang, Lixue Xia, Boxun Li, Yu Wang, and Huazhong Yang. Binary convolutional neural network on rram. In *Proc. ASP-DAC*, pages 782–787. IEEE, 2017.

[222] Jaeheum Lee, Jason K Eshraghian, Kyoungrok Cho, and Kamran Eshraghian. Adaptive precision cnn accelerator using radix-x parallel connected memristor crossbars. *arXiv preprint arXiv:1906.09395*, 2019.

[223] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. Ternary neural networks for resource-efficient ai applications. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2547–2554. IEEE, 2017.

[224] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*, 100:49–58, 2018.

[225] Kota Ando, Kodai Ueyoshi, Kentaro Orimo, Haruyoshi Yonekawa, Shimpei Sato, Hiroki Nakahara, Masayuki Ikebe, Tetsuya Asai, Shinya Takamaeda-Yamazaki, Tadahiro Kuroda, et al. Brein memory: A 13-layer 4.2 k neuron/0.8 m synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm cmos. In *Proc. VLSI Symp. on Circuits*, pages C24–C25. IEEE, 2017.

[226] Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot, Hande Alemdar, Nicholas Caldwell, and Vincent Leroy. Scalable high-performance architecture for convolutional ternary neural networks on fpga. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[227] Zhiwei Li, Pai-Yu Chen, Hui Xu, and Shimeng Yu. Design of ternary neural network with 3-d vertical rram array. *IEEE Transactions on Electron Devices*, 64(6):2721–2727, 2017.

[228] Biao Pan, Deming Zhang, Xueying Zhang, Haotian Wang, Jinyu Bai, Jianlei Yang, Youguang Zhang, Wang Kang, and Weisheng Zhao. Skyrmion-induced memristive magnetic tunnel junction for ternary neural network. *IEEE Journal of the Electron Devices Society*, 7:529–533, 2019.

[229] Stefano Gregori, Alessandro Cabrini, Osama Khouri, and Guido Torelli. On-chip error correcting techniques for new-generation flash memories. *Proc. IEEE*, 91(4):602–616, 2003.

[230] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462*, 2017.

[231] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, et al. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pages 5–14. ACM, 2017.

[232] Weisheng Zhao, Claude Chappert, Virgile Javerliac, and Jean-Pierre Noziere. High speed, high stability and low power sensing amplifier for mtj/cmos hybrid logic circuits. *IEEE Transactions on Magnetics*, 45(10):3784–3787, 2009.

[233] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proc. IEEE*, 98(2):253–266, Feb 2010.

[234] Rui Yang, Haitong Li, Kirby KH Smithe, Taeho R Kim, Kye Okabe, Eric Pop, Jonathan A Fan, and H-S Philip Wong. Ternary content-addressable memory with mos 2 transistors for massively parallel data search. *Nature Electronics*, 2(3):108–114, 2019.

[235] Alessandro Grossi, E Nowak, Cristian Zambelli, C Pellissier, S Bernasconi, G Cibrario, K El Hajjam, R Crochemore, JF Nodin, Piero Olivo, et al. Fundamental variability limits of filament-based rram. In *IEDM Tech. Dig.*, pages 4–7. IEEE, 2016.

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

[237] Denys Riwan Bunsothy Ly et al. Role of synaptic variability in resistive memory-based spiking neural networks with unsupervised learning. *J. Phys. D: Applied Physics*, 2018.

[238] E Vianello, O Thomas, G Molas, O Turkyilmaz, N Jovanović, D Garbin, G Palma, M Alayan, C Nguyen, J Coignus, et al. Resistive memories for ultra-low-power embedded computing design. In *2014 IEEE International Electron Devices Meeting*, pages 6–3. IEEE, 2014.

[239] Seung Ryul Lee, Young-Bae Kim, Man Chang, Kyung Min Kim, Chang Bum Lee, Ji Hyun Hur, Gyeong-Su Park, Dongsoo Lee, Myoung-Jae Lee, Chang Jung Kim, et al. Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory. In *2012 Symposium on VLSI Technology (VLSIT)*, pages 71–72. IEEE, 2012.

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

[241] Cong Xu, Dimin Niu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. Understanding the trade-offs in multi-level cell reram memory design. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2013.

[242] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[243] Jing Li, Binquan Luan, and Chung Lam. Resistance drift in phase change memory. In *2012 IEEE International Reliability Physics Symposium (IRPS)*, pages 6C–1. IEEE, 2012.

[244] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

[245] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207, 2013.

[246] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[247] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Appendix A

Synaptic Metaplasticity in Binarized Neural Networks

A.1 Forward and backward propagation in binarized neural networks

Algorithm 2 Forward function of the BNN reproduced from [153]. $\mathbf{W^b} = (\mathbf{W^b}_l)_{l=1...L}$ are the binary weights, $\boldsymbol{\theta}^{\text{BN}} = \{(\boldsymbol{\gamma}_l,\ \boldsymbol{\beta}_l) \mid l = 1...L\}$ are Batch Normalization parameters. L is the total number of layers and the subscript l when specified is the layer index. $\mathbf{x}$ is a batch of input data with dimensions (P, N) with P the number of pixels and N the number of examples in the batch. $\text{E}(\cdot)$ and $\text{Var}(\cdot)$ are batch-wise mean and variance. While they are computed during training with the statistics of the batches, running averages of the mean and variance are stored to be used at test time. This enables the network to infer on a single example at test time. ϵ is a small number to avoid division by zero, it was set to 10^{-5} in all the experiments.

Input: $\mathbf{W^b}$, $\boldsymbol{\theta}^{\text{BN}}$, $\mathbf{x}$.
Output: $\hat{\mathbf{y}}$, cache.

$\mathbf{a}_0 \leftarrow \mathbf{x}$ ▷ Input is not binarized
for $l = 1$ to L **do** ▷ For loop over the layers
  $\mathbf{z}_l \leftarrow \mathbf{W^b}_l \mathbf{a}_l$ ▷ Matrix multiplication
  $\mathbf{a}_l \leftarrow \boldsymbol{\gamma}_l \cdot \frac{\mathbf{z}_l - \text{E}(\mathbf{z}_l)}{\sqrt{\text{Var}(\mathbf{z}_l) + \epsilon}} + \boldsymbol{\beta}_l$ ▷ Batch Normalization [86]
  if $l < L$ **then** ▷ If not the last layer
    $\mathbf{a}^{\text{b}}_l \leftarrow \text{Sign}(\mathbf{a}_l)$ ▷ Activation is binarized
  end if
end for
$\hat{\mathbf{y}} \leftarrow \mathbf{a}_L$
return $\hat{\mathbf{y}}$, cache

The optimization is performed using Adaptive Moment Estimation (Adam) algorithm [30]. As the sign function is not differentiable in zero and the derivative is zero on $\mathbb{R}^\star$, during error backpropagation the derivative of hardtanh function is used as a replacement for the derivative

Algorithm 3 Backward function of the BNN reproduced from [153]. $\mathbf{W^b} = (\mathbf{W^b}_l)_{l=1...L}$ are the binary weights, $\boldsymbol{\theta}^{\text{BN}} = \{(\boldsymbol{\gamma}_l,\ \boldsymbol{\beta}_l) \mid l = 1...L\}$ are Batch Normalization parameters. BackBatchNorm$(\cdot)$ specifies how to backpropagate through the Batch normalization [86]. L is the total number of layers and the subscript l when specified is the layer index. $1_{|a_l|\leq 1}$ is the derivative of Hardtanh taken as a replacement for back propagating through Sign activation.

Input: $C, \hat{\mathbf{y}}, \mathbf{W^b}, \boldsymbol{\theta}^{\text{BN}}$, cache.
Output: $(\partial_W C, \partial_\theta C)$.

$\mathbf{g}_{a_L} \leftarrow \frac{\partial C}{\partial \hat{\mathbf{y}}}$ ▷ Cost gradient with respect to output
for $l = L$ to 1 **do** ▷ For loop backward over the layers
  if $l < L$ **then** ▷ If not the last layer
    $\mathbf{g}_{a_l} \leftarrow \mathbf{g}_{a_l^{\text{b}}} \cdot 1_{|a_l|\leq 1}$ ▷ Back Prop through Sign
  end if
  $(\mathbf{g}_{z_l}, \mathbf{g}_{\gamma_l}, \mathbf{g}_{\beta_l}) \leftarrow \text{BackBatchNorm}(\mathbf{g}_{a_l}, \mathbf{z}_l, \boldsymbol{\gamma}_l, \boldsymbol{\beta}_l)$ ▷ See [86]
  $\mathbf{g}_{a^{\text{b}}_{l-1}} \leftarrow \mathbf{W^b}_l \mathbf{g}_{z_l}$
  $\mathbf{g}_{W^{\text{b}}_l} \leftarrow \mathbf{a}^{\text{b}}_{l-1}{}^{\top} \mathbf{g}_{z_l}$
end for
$\partial_W C \leftarrow \{\mathbf{g}_{W^{\text{b}}_l} \mid l = 1...L\}$
$\partial_\theta C \leftarrow \{\mathbf{g}_{\gamma_l}, \mathbf{g}_{\beta_l} \mid l = 1...L\}$
return $(\partial_W C, \partial_\theta C)$

of the Sign function. The activation function is the sign function except for the output layer. The input neurons are not binarized. We use batch normalization [86] at all layers as detailed in Alg. 2. The following derivation for layer l,

$$\gamma_l \cdot \frac{z - \text{E}(z)}{\sqrt{\text{Var}(z) + \epsilon}} + \beta_l = \frac{\gamma_l}{\sqrt{\text{Var}(z) + \epsilon}} \left(z - \left[\text{E}(z) - \frac{\beta_l \sqrt{\text{Var}(z) + \epsilon}}{\gamma_l} \right] \right)$$

$$a = \text{Sign}(\gamma_l) \text{Sign} \left(z - \left[\text{E}(z) - \frac{\beta_l \sqrt{\text{Var}(z) + \epsilon}}{\gamma_l} \right] \right)$$

shows that because the Sign function is invariant by any multiplicative constant in the input, the only task dependent parameters we need to store for an inference hardware chip is the term between square brackets, along with the sign of γ_l. The amount of task dependent parameters scales as the number of neurons and is order of magnitudes smaller than the number of synapses.

Adam optimizer updates the hidden weight with loss gradients computed using binary weights only. We use a small weight decay of 10^{-7} in the Adam optimizer to make zero floating values more stable. However, consolidated weights are not subject to weight decay, as we implement weight decay as a modification of the loss gradient, which is gradually suppressed by f_{meta}.

	pMNISTs		
Network	Binarized meta	Binarized EWC	Full precision
Layers	784-4096-4096-10	784-4096-4096-10	784-4096-4096-10
Learning rate	0.005	0.005	0.005
Minibatch size	100	100	100
Epochs/task	40	40	40
m	1.35	0.0	1.35
λ_{EWC}	0.0	5,000	0.0
Weight decay	1e-7	1e-7	1e-7
Initialization	Uniform width = 0.1	Uniform width = 0.1	Uniform width = 0.1

Table A.1: Hyperparameters for the permuted MNISTs experiment.

	FMNIST - MNIST
Network	Binarized meta
Layers	784-4096-4096-10
Learning rate	0.005
Minibatch size	100
Epochs/task	50
m	1.5
Weight decay	1e-8
Initialization	Uniform width = 0.1

Table A.2: Hyperparameters for the permuted FMNIST-MNIST experiment.

	Stream FMNIST	Stream CIFAR-10
Network	Binarized meta	Binarized meta
Layers	784-1024-1024-10	VGG-7
Sub Parts	60	20
Learning rate	0.005	0.0001
Minibatch size	100	64
Epochs/subset	20	200
m	2.5	13.0
Weight decay	1e-7	0.0
Initialization	Uniform width = 0.1	Gauss width = 0.007

Table A.3: Hyperparameters for the stream learning experiment.

A.2 Training parameters

The batch normalization layers parameters were not learned for the Fashion MNIST experiment whereas they were learned for the CIFAR-10 experiment. The batch normalization parameters are set to $\beta = 0$, $\gamma = 1$ for the Fashion MNIST experiment. The performance of the BNN with learned batch normalization parameters was inferior, as batch normalization parameters appear to overfit to the subsets of data. In the CIFAR-10 experiment the performance was higher with learned batch normalization parameters. The architecture of VGG-7 network consists of 6 convolutional layers of 3×3 sized kernels with kernel number per layer following the sequence 128-128-256-256-512-512. The classifier consists of two hidden layers of 2048-1024 hidden units. Dropout was used in the classifier with value 0.5.

A.3 Implementation of Synaptic Intelligence

In this section, we discuss the implementation of the synaptic intelligence algorithm [125], designed for continual learning in full precision neural networks. The algorithm consists in optimizing the loss function

$$\widetilde{L}_\mu = L_\mu + c\sum_k \Omega_k^\mu (\widetilde{\theta}_k - \theta_k)^2 \tag{A.1}$$

when learning the task μ, where L_μ is the loss function associated with the current task and $c\sum_k \Omega_k^\mu (\widetilde{\theta}_k - \theta_k)^2$ is a "surrogate loss"[125] compelling the current parameters θ_k to stay close to the parameters $\widetilde{\theta}_k$ optimized for previous tasks. Ω_k^μ is the importance factor for parameter $\widetilde{\theta}_k$ and is updated between each task by

$$\Omega_k^\mu = \sum_{\nu<\mu} \frac{\omega_k^\nu}{(\Delta_k^\nu)^2 + \xi}. \tag{A.2}$$

Δ_k^ν is a normalization factor equal to the total parameter change over the latest learned task, and ξ is a small constant number avoiding any division by zero. ω_k^ν is computed in an online fashion by approximating the path integral of the parameters and can be interpreted as the parameter specific contribution to changes in the total loss.

$$\omega_k^\mu = -\sum_t \frac{\partial L}{\partial \theta_k(t)} (\theta_k(t+1) - \theta_k(t)). \tag{A.3}$$

As a control experiment, we reproduce the results of [125] for the permuted MNIST benchmark in Fig. A.1a, with $c = 0.1$ and $\xi = 0.1$. In the case of binarized neural networks, we tried several ways of computing the importance factor Ω_k^μ by employing either the binarized weight or the hidden weight for ω_k^μ and Δ_k^ν. The best performance was achieved by using the binarized weight values for ω_k^μ and the hidden weight values for Δ_k^ν and $c = 1.0$, $\xi = 0.1$. The results are

shown in Supp. Fig. A.1b.

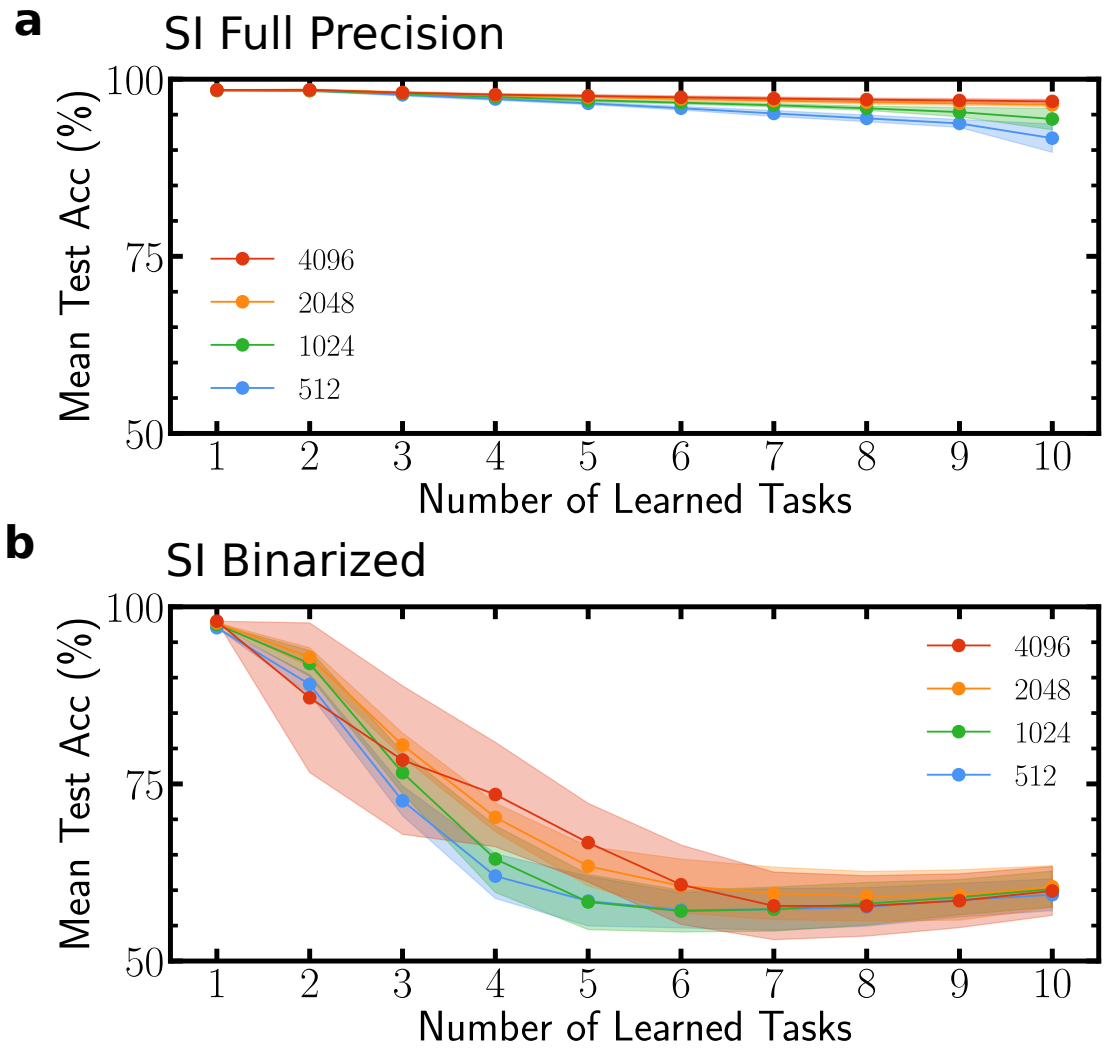

Figure A.1: **Synaptic Intelligence a** applied to full precision neural networks with two hidden ReLU layers of increasing size ranging from 512 to 4,096 for the permuted MNIST benchmark, results reproduced from [125]. **b** The best performing adaptation of Synaptic Intelligence to binarized neural networks.) The curves are averaged over five runs and shadows stand for one standard deviation.

A.4 Use of a metaplasticity function f_{meta} featuring a hard threshold

In this section, we present a control experiment where the modulating function f_{meta} is a hard threshold function such that $f_{\text{meta}}(W^{\text{h}}) = 1$ if $|W^{\text{h}}| < m$, and $f_{\text{meta}}(W^{\text{h}}) = 0$ if $|W^{\text{h}}| > m$. The hyperparameter m is, in this case, the threshold value above which f_{meta} is zero. The value of m is obtained by hyperparameter tuning and set to $m = 0.4$.

We observe that the performance is degraded modestly when using such threshold mechanism, in accordance with the theoretical evidence that high hidden weights correspond to important binarized weights for consolidation. The most degradation is observed in the regime where the neural network exhibits the highest capacity in number of tasks (network with 4,096-wide layers trained with nine or ten tasks).

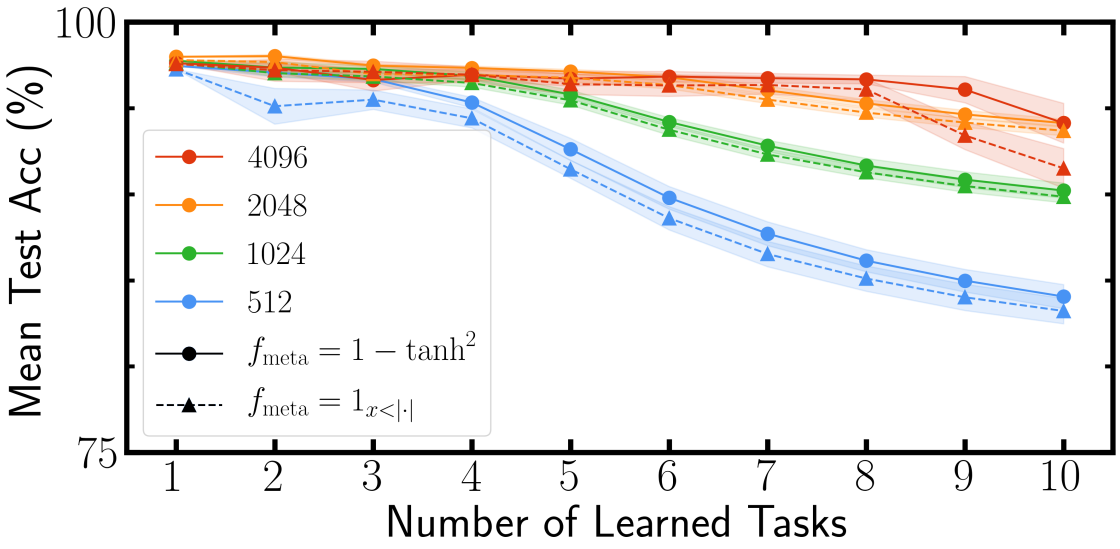

Figure A.2: **Comparison of different choices for f_{meta}**, when ten training ten permuted MNIST tasks. This plot shows the comparison between two classes of f_{meta} functions. The bullets represent the metaplastic BNNs with the function class introduced in the body text with $m = 1.35$, while the squares denote an f_{meta} function with a hard threshold above which a weight is irreversibly consolidated. The threshold value is tuned to be 0.4. The colors denote increasing network sizes. The curves are averaged over five runs and shadows stand for one standard deviation.

A.5 Mathematical proofs

Proof of Lemma 1. We first prove Eq. (2.5). Let us assume that $\mathbf{W}^\star \notin \overline{\mathscr{B}}_\infty$ so that there exists at least one component $i \in [\![1,d]\!]$ such that $|W_i^\star| > 1$. Since H is symmetric definite positive, it is invertible. Taking the euclidian scalar product between $\mathbf{H}^{-1}\mathbf{e}_i$ and the update $(\mathbf{W^h}_{t+1} - \mathbf{W^h}_t)$ yields:

$$\begin{aligned}
\langle \mathbf{H}^{-1}\mathbf{e}_i,\, \mathbf{W^h}_{t+1} - \mathbf{W^h}_t \rangle &= (\mathbf{H}^{-1}\mathbf{e}_i)^T \cdot (\mathbf{W^h}_{t+1} - \mathbf{W^h}_t) \\
&= -\eta (\mathbf{H}^{-1}\mathbf{e}_i)^T \cdot \mathbf{H}(\mathrm{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star) \\
&= -\eta\, \mathbf{e}_i^T \cdot (\mathbf{H}^{-1})^T \mathbf{H}(\mathrm{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star) \\
&= -\eta\, \mathbf{e}_i^T \cdot \mathbf{H}^{-1}\mathbf{H}(\mathrm{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star) \\
&= -\eta\, \mathbf{e}_i^T \cdot (\mathrm{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star) \\
&= -\eta(\mathrm{sign}(W^{\mathrm{h}}_{i,t}) - W_i^\star),
\end{aligned}$$

where we have used at the fourth equality that $\mathbf{H}^{-1}$ is also symmetric. Since $|W_i^\star| > 1$, the sign of $\mathrm{sign}(W^{\mathrm{h}}_{i,t}) - W_i^\star$ is constant (and $\neq 0$), so the component of $\mathbf{W}$ along $\mathbf{H}^{-1}\mathbf{e}_i$ is expected to diverge. More precisely, let us assume $W_i^\star > 1$ so that $\mathrm{sign}(W^{\mathrm{h}}_{i,t}) - W_i^\star < 1 - W_i^\star$ and:

$$\langle \mathbf{H}^{-1}\mathbf{e}_i, \mathbf{W^h}_{t+1} - \mathbf{W}^{\mathrm{h}}_t \rangle \geq -\eta(1 - W_i^\star). \tag{A.4}$$

Summing Eq. (A.4) from time step 0 to t yields:

$$\langle \mathbf{H}^{-1}\mathbf{e}_i, \mathbf{W^h}_t \rangle \geq -\eta(1 - W_i^\star)\, t + \langle \mathbf{H}^{-1}\mathbf{e}_i, \mathbf{W^h}_0 \rangle, \tag{A.5}$$

showing that $\lim_{t\to+\infty} \langle \mathbf{H}^{-1}\mathbf{e}_i, \mathbf{W^h}_t \rangle = +\infty$. Consequently there exists $j \in [\![1,d]\!]$ such that $\lim_{t\to+\infty} \langle \mathbf{e}_j, \mathbf{W^h}_t \rangle = +\infty$ and therefore $\lim_{t\to\infty} \|\mathbf{W^h}_t\|_\infty = +\infty$. Similarly if $W_i^\star < -1$, we show that:

$$\langle \mathbf{H}^{-1}\mathbf{e}_i, \mathbf{W^h}_t \rangle \leq \eta(1 + W_i^\star)\, t + \langle \mathbf{H}^{-1}\mathbf{e}_i, \mathbf{W^h}_0 \rangle, \tag{A.6}$$

giving the same conclusion as above.

We now prove Eq. (2.4). Let us assume that $\mathbf{W}^\star \in \mathscr{B}_\infty$, i.e. $\forall i \in [\![1,d]\!]$, $|W_i^\star| < 1$. We have:

$$
\begin{aligned}
&\|\mathbf{W^h}_{t+1}\|^2_{\mathbf{H}^{-1}} = \langle \mathbf{W^h}_{t+1}, \mathbf{W^h}_{t+1}\rangle_{\mathbf{H}^{-1}} \\
&= \langle \mathbf{W^h}_t + \Delta\mathbf{W^h}_t, \mathbf{W^h}_t + \Delta\mathbf{W^h}_t\rangle_{\mathbf{H}^{-1}} \\
&= \|\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} + 2\langle \Delta\mathbf{W^h}_t, \mathbf{W^h}_t\rangle_{\mathbf{H}^{-1}} + \langle \Delta\mathbf{W^h}_t, \Delta\mathbf{W^h}_t\rangle_{\mathbf{H}^{-1}} \\
&= \|\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} + 2\langle \mathbf{H}^{-1}\Delta\mathbf{W^h}_t, \mathbf{W^h}_t\rangle + \|\Delta\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} \\
&= \|\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} - 2\eta(\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)^T\mathbf{W^h}_t + \|\Delta\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} \\
&= \|\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} - 2\eta(\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)^T\mathbf{W^h}_t + \|\Delta\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}},
\end{aligned}
$$

so that :

$$
\begin{aligned}
&\|\mathbf{W^h}_{t+1}\|^2_{\mathbf{H}^{-1}} - \|\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} \leq 0 \\
&\Leftrightarrow 2(\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)^T \cdot \mathbf{W^h}_t \geq \|\Delta\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}}.
\end{aligned}
\tag{A.7}
$$

We want to show that if $\mathbf{W^h}_t$ is large enough in norm $\|\cdot\|_{\mathbf{H}^{-1}}$, Eq. (A.7) will be met. First note that, because the dimension is finite there exist two constants $\alpha > 0$ and $\beta > 0$ such that $\forall \mathbf{x} \in \mathbb{R}^d$,

$$
\alpha\|\mathbf{x}\|_{\mathbf{H}^{-1}} < \|\mathbf{x}\|_\infty < \beta\|\mathbf{x}\|_{\mathbf{H}^{-1}}
$$

and also that:

$$
\|\Delta\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}} = \eta^2\|\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star\|^2_{\mathbf{H}}.
$$

Then, by triangular inequality:

$$
\eta\|\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star\|_{\mathbf{H}} \leq \eta(\|\text{sign}(\mathbf{W^h}_t)\|_{\mathbf{H}} + \|\mathbf{W}^\star\|_{\mathbf{H}}).
$$

Denoting $(\mathbf{e}_\alpha)_\alpha$ and $(\lambda_\alpha)_\alpha$ the eigenbasis of $\mathbf{H}$ and their associated eigenvalues, we have by Cauchy Schwarz inequality:

$$
\begin{aligned}
\|\text{sign}(\mathbf{W^h}_t)\|^2_{\mathbf{H}} &= \langle \mathbf{H}\cdot\text{sign}(\mathbf{W^h}_t), \text{sign}(\mathbf{W^h}_t)\rangle \\
&= \sum_{\alpha=1}^{d} \lambda_\alpha |\langle \text{sign}(\mathbf{W^h}_t), \mathbf{e}_\alpha\rangle|^2 \\
&\leq \sum_{\alpha=1}^{d} \lambda_\alpha \underbrace{\|\text{sign}(\mathbf{W^h}_t)\|_2^2}_{=d} \cdot \underbrace{\|\mathbf{e}_\alpha\|_2^2}_{=1} \\
&\leq d^2\lambda_{\alpha,max},
\end{aligned}
$$

so that:

$$
\|\Delta\mathbf{W^h}_t\|_{\mathbf{H}^{-1}} \leq \eta(d\sqrt{\lambda_{\alpha,max}} + \|\mathbf{W}^\star\|_{\mathbf{H}}).
\tag{A.8}
$$

Thus the right hand side of Eq. A.7 is bounded. Also note that:

$$\begin{aligned}
2(\text{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)^T \cdot \mathbf{W^h}_t &= 2\sum_{i=1}^{d}(1 - \text{sign}(W^{\text{h}}_{i,t})W^\star_i)|W^{\text{h}}_{i,t}| \\
&\geq 2\sum_{i=1}^{d}(1 - |W^\star_i|)|W^{\text{h}}_{i,t}| \\
&\geq 2(1 - \|\mathbf{W}^\star\|_\infty)\sum_{i=1}^{d}|W^{\text{h}}_{i,t}| \\
&\geq 2(1 - \|\mathbf{W}^\star\|_\infty) \cdot \|\mathbf{W^h}_t\|_\infty,
\end{aligned}$$

So far we have shown that the left hand side of Eq.A.7 is lower bounded by a constant ($\neq 0$) times the infinite norm of $\mathbf{W^h}_t$, while the right hand side is bounded. Therefore to ensure Eq. (A.7) it suffices that:

$$\begin{aligned}
&2(1 - \|\mathbf{W}^\star\|_\infty) \cdot \|\mathbf{W^h}_t\|_\infty \geq \eta(d\sqrt{\lambda_{\alpha,max}} + \|\mathbf{W}^\star\|_{\mathbf{H}}) \\
\Leftrightarrow\ &\|\mathbf{W^h}_t\|_\infty \geq \frac{\eta(d\sqrt{\lambda_{\alpha,max}} + \|\mathbf{W}^\star\|_{\mathbf{H}})}{2(1 - \|\mathbf{W}^\star\|_\infty)}.
\end{aligned}$$

And thus to ensure Eq. (A.7) it suffices that:

$$\|\mathbf{W^h}_t\|_{\mathbf{H}^{-1}} \geq \frac{\eta(d\sqrt{\lambda_{\alpha,max}} + \|\mathbf{W}^\star\|_{\mathbf{H}})}{2\alpha(1 - \|\mathbf{W}^\star\|_\infty)}.$$

Denoting $M = \frac{\eta(d\sqrt{\lambda_{\alpha,max}} + \|\mathbf{W}^\star\|_{\mathbf{H}})}{2\alpha(1-\|\mathbf{W}^\star\|_\infty)}$, we can conclude that $\|\mathbf{W^h}_t\|_{\mathbf{H}^{-1}} \geq M \Rightarrow \|\mathbf{W^h}_{t+1}\|^2_{\mathbf{H}^{-1}} < \|\mathbf{W^h}_t\|^2_{\mathbf{H}^{-1}}$. And because the update $\Delta\mathbf{W^h}_t$ is bounded in norm $\|\cdot\|_{\mathbf{H}^{-1}}$, an absolute upper bound of $\mathbf{W^h}_t$ is :

$$C = \beta\max(\|\mathbf{W^h}_0\|_{\mathbf{H}^{-1}},\ M + \eta(d\sqrt{\lambda_{\alpha,max}} + \|\mathbf{W}^\star\|_{\mathbf{H}})).$$

Thus we have proven that $\mathbf{W}^\star \in \mathscr{B}_\infty \Rightarrow \exists C > 0, \forall t \in \mathbb{N}, \|\mathbf{W^h}_t\|_\infty < C$

□

Proof of Lemma 2. If $\mathbf{H} = \text{diag}(\lambda_1, \ldots \lambda_d)$, the dynamics of $\mathbf{W^h}_t$ defined in Eq. (2.3) simply rewrites component-wise:

$$\forall i \in [\![1, d]\!]\ ,\ \Delta W^{\text{h}}_{i,t} = W^{\text{h}}_{i,t+1} - W^{\text{h}}_{i,t} = -\eta\lambda_i(\text{sign}(W^{\text{h}}_{i,t}) - W^\star_i). \tag{A.9}$$

By Lemma 1, components W_i such that $|W^\star_i| < 1$ are bounded.

For components i where $|W^\star_i| > 1$, $\Delta W^{\text{h}}_{i,t}$ has the sign of $W^\star_i$ since Eq. (A.9) rewrites:

$$\Delta W_{i,t}^{\mathrm{h}} = \mathrm{sign}(W_i^\star) \underbrace{\eta\lambda_i(|W_i^\star| - \mathrm{sign}(W_i^\star W_{i,t}^{\mathrm{h}}))}_{>0}, \tag{A.10}$$

so that $W_{i,t}^{\mathrm{h}}$ necessarily ends up having the same sign as $W_i^\star$, hence there exists $t_{0,i} \in \mathbb{N}$ such that :

$$\forall\, t > t_{0,i}, \quad \Delta W_{i,t}^{\mathrm{h}} = \mathrm{sign}(W_i^\star)\eta\lambda_i(|W_i^\star| - 1). \tag{A.11}$$

By definition of $t_{0,i}$, $W_{i,t}^{\mathrm{h}}$ and $W_i^\star$ have opposite sign before $t_{0,i}$ so that:

$$\forall\, t \le t_{0,i}, \quad \Delta W_{i,t}^{\mathrm{h}} = \mathrm{sign}(W_i^\star)\eta\lambda_i(1 + |W_i^\star|). \tag{A.12}$$

Therefore, summing Eq. (A.9) between 0 and t yields :

$$\begin{aligned} W_{i,t}^{\mathrm{h}} &= W_{i,0}^{\mathrm{h}} + \sum_{u=0}^{t_{0,i}} \mathrm{sign}(W_i^\star)\eta\lambda_i(|W_i^\star| + 1) \\ &\quad + \sum_{u=t_{0,i}+1}^{t} \mathrm{sign}(W_i^\star)\eta\lambda_i(|W_i^\star| - 1) \\ &= W_{i,0}^{\mathrm{h}} + \mathrm{sign}(W_i^\star)\eta\lambda_i(|W_i^\star| + 1)\, t_{0,i} \\ &\quad + \mathrm{sign}(W_i^\star)\eta\lambda_i(|W_i^\star| - 1)(t - t_{0,i}) \\ &\sim_{t\to+\infty} \underbrace{\mathrm{sign}(W_i^\star)\eta\lambda_i(|W_i^\star| - 1)}_{=\widetilde{W_i^{\mathrm{h}}}}\, t \end{aligned} \tag{A.13}$$

□

Proof of Theorem . Using Eq. (2.2), the loss reads:

$$\begin{aligned} \mathscr{L}(\mathbf{W^h}_t) &= \frac{1}{2}(\mathrm{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star)^T \mathbf{H}(\mathrm{sign}(\mathbf{W^h}_t) - \mathbf{W}^\star) \\ &= \frac{1}{2}\sum_{i=1}^{n} \lambda_i(\mathrm{sign}(W_{i,t}^{\mathrm{h}}) - W_i^\star)^2 \\ &= \frac{1}{2}\sum_{i,|W_i^\star|\le 1} \lambda_i(\mathrm{sign}(W_{i,t}^{\mathrm{h}}) - W_i^\star)^2 \\ &\quad + \frac{1}{2}\sum_{i,|W_i^\star|>1} \lambda_i(\mathrm{sign}(W_{i,t}^{\mathrm{h}}) - W_i^\star)^2. \end{aligned}$$

Using Lemma 2, for all components i such that $|W_i^\star| > 1$, there exists $t_{0,i}$ such that for all $t > t_{0,i}$, $\mathrm{sign}(W_{i,t}^{\mathrm{h}}) = \mathrm{sign}(W_i^\star)$ and therefore $\frac{1}{2}\lambda_i(\mathrm{sign}(W_{i,t}^{\mathrm{h}}) - W_i^\star)^2 = \frac{1}{2}\lambda_i(1 - |W_i^\star|)^2$. Defining $T = \max_{i||W_i^\star|>1}(t_{0,i})$, the loss rewrites for $t > T$:

$$\mathscr{L}(\mathbf{W^h}_t) = \frac{1}{2}\sum_{i,|W_i^\star|\leq 1}\lambda_i(\text{sign}(W^{\text{h}}_{i,t}) - W_i^\star)^2 + \frac{1}{2}\sum_{i,|W_i^\star|>1}\lambda_i(|W_i^\star|-1)^2$$

Then, the increase in energy if a binary component in the $|W_i^\star| > 1$ sum is switched is :

$$\Delta_i\mathscr{L}(\mathbf{W^h}_t) = \frac{\lambda_i}{2}((|W_i^\star|+1)^2 - (|W_i^\star|-1)^2) = 2\lambda_i|W_i^\star| \tag{A.14}$$

Using the explicit form of $W^{\text{h}}_{i,t}$ in Eq. (A.13) along with Eq. (A.14), we get:

$$\begin{aligned}
W^{\text{h}}_{i,t} &= W^{\text{h}}_{i,0} + \text{sign}(W_i^\star)\eta\lambda_i(|W_i^\star|+1)\,t_{0,i} \\
&\quad + \text{sign}(W_i^\star)\eta\lambda_i(|W_i^\star|-1)(t-t_{0,i}) \\
&= W^{\text{h}}_{i,0} + \text{sign}(W_i^\star)\eta\lambda_i\left(\frac{\Delta_i\mathscr{L}}{2\lambda_i}+1\right)t_{0,i} \\
&\quad + \text{sign}(W_i^\star)\eta\lambda_i\left(\frac{\Delta_i\mathscr{L}}{2\lambda_i}-1\right)(t-t_{0,i}) \\
&= W^{\text{h}}_{i,0} + \text{sign}(W_i^\star)\eta\frac{\Delta_i\mathscr{L}}{2}t + \text{sign}(W_i^\star)\eta\lambda_i(2t_{0,i}-t) \\
&= \text{sign}(W_i^\star)\eta\left(\frac{\Delta_i\mathscr{L}}{2}-\lambda_i\right)t + W^{\text{h}}_{i,0} + \text{sign}(W_i^\star)\eta\lambda_i 2t_{0,i}.
\end{aligned}$$

Since $W^{\text{h}}_{i,t}$ has the same sign as $W_i^\star$ for t being large enough, multiplying both sides for the last equation and dividing by t yields:

$$\Delta_i\mathscr{L}(\mathbf{W^h}_t) = 2\left(\lambda_i + \frac{\widetilde{|W_i^{\text{h}}|}}{\eta}\right)\underbrace{-2\frac{|W^{\text{h}}_{i,0}| + \eta\lambda_i 2t_{0,i}}{\eta t}}_{=\mathscr{O}(\frac{1}{t})} \tag{A.15}$$

□

A.6 Comparison with learning rate decay

In this section, we investigate the performance of a learning rate decay scheduler to see how it compares to our metaplastic binarized neural network approach. We study the setting of learning six permuted MNISTs and investigate learning rate schedulers where the learning rate is divided by a constant factor between each task. We list in Table A.4 the performance for several values of initial learning rates and dividing factors. For instance, an initial learning rate of 10^{-2} and dividing factor of 10 means that the six tasks are learned respectively with the learning rates : 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, and 10^{-7}.

(initial LR, dividing factor)	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
(0.001, 5.0)	60.93(9.5)	74.49(6.2)	73.28(7.1)	63.47(2.6)	60.54(3.9)	95.71(0.2)
(0.01, 5.0)	27.01(5.8)	51.15(7.2)	79.86(4.5)	86.53(0.9)	81.45(4.1)	96.62(0.1)
(0.1, 5.0)	9.70(0.5)	11.94(2.4)	11.50(3.1)	18.26(11.3)	72.47(14.4)	96.98(0.2)
(0.001, 10.0)	84.98(6.5)	86.02(3.3)	62.58(8.7)	58.86(5.0)	82.77(1.7)	91.42(0.2)
(0.01, 10.0)	57.21(8.2)	86.35(3.9)	84.13(2.5)	67.00(4.6)	75.12(5.7)	94.77(0.2)
(0.1, 10.0)	10.58(1.0)	14.49(7.8)	11.64(2.8)	28.51(24.5)	66.67(26.5)	95.88(0.2)
(0.001, 20.0)	**93.23(2.4)**	**86.85(1.3)**	**73.37(1.8)**	**82.83(3.6)**	**87.76(0.7)**	**73.29(2.3)**
(0.01, 20.0)	**89.28(1.4)**	**93.40(1.3)**	**80.13(5.0)**	**76.71(7.4)**	**88.67(0.9)**	**82.15(1.0)**
(0.1, 20.0)	10.16(0.7)	13.16(4.1)	16.46(3.4)	40.85(31.0)	41.88(43.6)	88.77(0.4)
(0.005, 5.0)	38.38(3.0)	65.13(5.5)	78.35(6.8)	82.42(2.0)	77.61(5.2)	96.56(0.0)
(0.05, 5.0)	9.90(0.1)	11.60(1.6)	11.03(3.5)	24.04(16.6)	69.18(13.6)	96.93(0.1)
(0.5, 5.0)	10.03(0.1)	10.04(0.3)	11.29(2.0)	12.73(6.6)	28.34(15.0)	97.12(0.1)
(0.005, 10.0)	71.14(6.4)	87.20(2.6)	86.06(2.8)	63.66(5.6)	75.13(2.5)	93.91(0.1)
(0.05, 10.0)	10.14(0.1)	10.58(1.7)	9.79(0.5)	13.72(3.4)	51.86(34.4)	95.58(0.2)
(0.5, 10.0)	10.09(0.3)	9.99(0.2)	9.74(0.5)	10.14(0.5)	38.81(22.7)	96.79(0.2)
(0.005, 20.0)	**91.51(2.1)**	**93.67(1.1)**	**75.41(6.1)**	**80.62(1.3)**	**88.69(0.4)**	**80.89(0.9)**
(0.05, 20.0)	10.01(1.2)	9.90(1.3)	10.08(0.8)	39.40(37.4)	71.63(34.5)	85.22(1.3)
(0.5, 20.0)	10.32(0.7)	10.27(1.1)	9.89(0.2)	15.57(4.0)	14.13(4.6)	91.93(0.5)

Table A.4: Permuted MNIST experiment with learning rate decay. The accuracy for each task is averaged over five runs and standard deviation is given between parenthesis. Best settings are in bold font.

A.7 Sequential Training of the MNIST and Fashion-MNIST Datasets

To test the ability of our binarized neural network to learn several tasks sequentially, we train a binarized neural network sequentially on two tasks in a more difficult situation than permuted MNISTs. When learning permuted versions of MNIST, the relevant input features do not overlap extensively between tasks which makes it easier for the network to learn sequentially. For this reason, we now train a binarized neural network with two hidden layers of 4,096 units to learn sequentially the MNIST dataset and the Fashion-MNIST dataset [173] which consists of fashion items images belonging to ten classes. Fig. A.3(b) shows the result of the training of a $m = 1.5$ binarized neural network, with 50 epochs on MNIST and 50 epochs on Fashion-MNIST (Fig. A.3(d) shows the reverse training order). Figs. A.3(a) and (c) also show the result for the conventional binarized neural network ($m = 0$). Baselines define the accuracies the binarized neural network would have obtained had it been trained on each of these tasks separately. The baseline of Fashion-MNIST is taken in Fig. A.3(a) (orange curve after 100 epochs) and the baseline of MNIST in Fig. A.3(c) (blue curve after 100 epochs). We observe that the metaplas-

tic binarized neural network is able to learn both tasks sequentially with baseline accuracies regardless of the order chosen to learn the tasks.

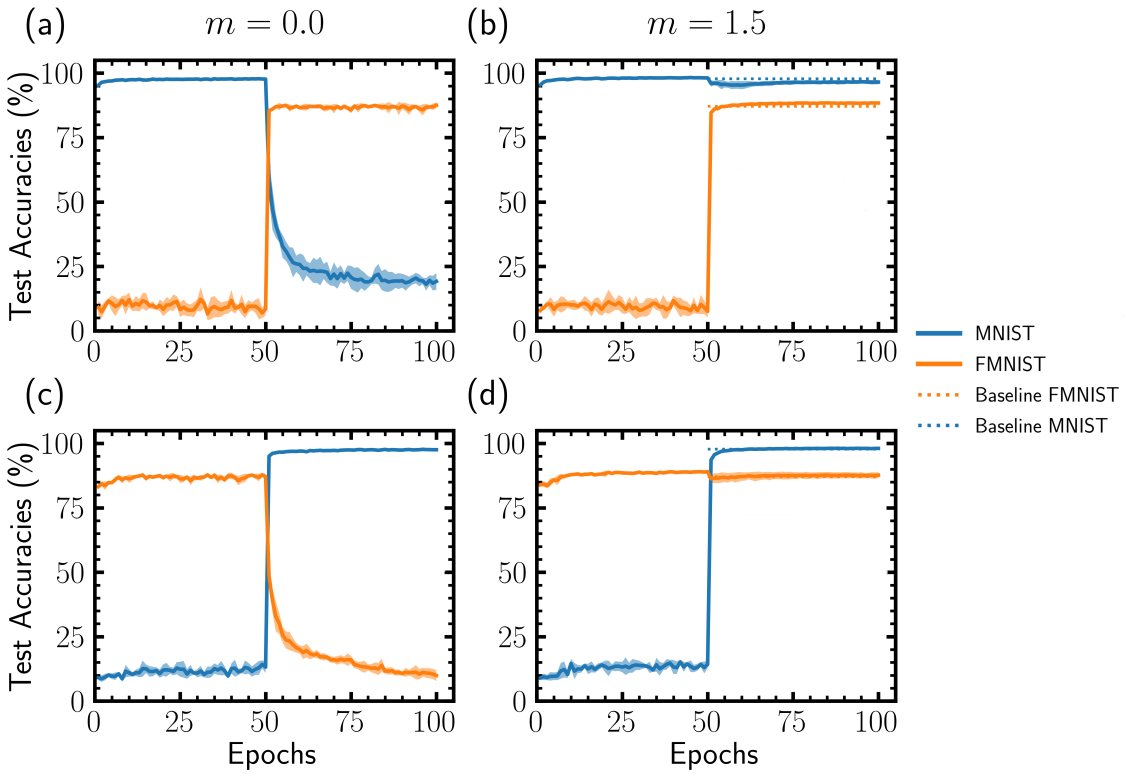

Figure A.3: **MNIST/Fashion-MNIST sequential learning.** Binarized neural network learning MNIST and Fashion-MNIST sequentially ((a) and (b)) or Fashion-MNIST and MNIST ((c) and (d)) for two values of the metaplastic parameter m. $m = 0$ corresponds to a conventional BNN ((a) and (c)), $m = 1.5$ is a metaplastic BNN ((b) and (d)). Curves are averaged over five runs and shadows correspond to one standard deviation.

A.8 Sequential Training of the MNIST and USPS Datasets

In this section, we investigate the sequential training of two closely related tasks: the handwritten digits of the MNIST (Supp. Fig. A.4(a)) and of the United States Postal Services (USPS, Supp. Fig. A.4(b)) datasets. This situation differs from permuted MNIST (Fig. 2 in the main body text), sequential Fashion-MNIST / MNIST (section A.7) and incremental CIFAR-10/CIFAR-100 (section A.9), where the incrementally trained tasks were always largely uncorrelated in nature.

We compare the accuracy of a metaplastic binarized neural network trained sequentially on MNIST and USPS, with the two networks trained independently on each task and each featuring half the number of hidden neurons (Fig. 4(c) in the main body text) and half the number

of parameters (Fig. 4(d) in the main body text) of the metaplastic network. This choice allows verifying in this situation whether a metaplastic network performs better than a network partitioned into two parts, with each partition trained on one task, independently from the other. As the MNIST dataset is much larger than the USPS one, we follow the training protocol introduces in [245] and [246], where 2,000 training examples are used for MNIST and 1,800 for USPS. For this reason, we focus on relatively small neural networks. The small network is a convolutional neural network with three layers of 4×4 kernels with increasing feature maps of 6-10-15 (4,056 parameters) while the neural network with twice more neurons (Fig. 4(c)) has feature maps 12-20-30 (14,832 parameters). We can choose the dimensions to be 10-15-20 to obtain a network with approximately twice more parameters (8,160) (Fig. 4(d)).

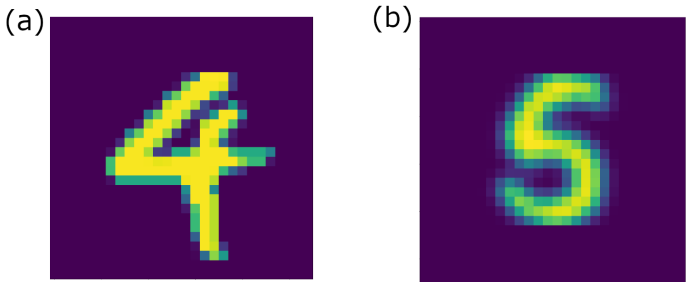

Figure A.4: (a) One MNIST training example, (b) One USPS training example.

A.9 Class Incremental Learning on CIFAR-10 and CIFAR-100 Features

In this section, we investigate a setting of class incremental learning, where a network learns different subsets of classes of the CIFAR-10 and CIFAR-100 datasets sequentially. We focus on the sequential training of the fully-connected layers of a convolutional neural network. This choice is motivated by the fact that the ability to extract features from visual input does not change across time presumably: for instance, one does not usually forget how to recognize shapes, but rather we can forget abstracted concepts.

To extract relevant features from the CIFAR-10 and CIFAR-100 datasets, we therefore use the convolutional layers of a ResNet-18 network [42], pretrained on the ImageNet dataset, and available in the PyTorch 1.1.0 library. This choice ensures that the feature extractor is fairly general, without having been trained on CIFAR images. We create a feature-extracted dataset of CIFAR-10 and CIFAR-100 by resizing CIFAR images from 32×32 to 220×220 pixels, and applying random crops of a 200×200 window, as well as random horizontal flips. We then perform ten passes through the training set of each dataset, resulting in 500,000 training images for each training sets. We perform only one pass through the test sets and do not apply data augmentation (we only resize the test images to 220×220 pixels and center-crop them to 200×200 pixels).

The features obtained by this procedure are 512-dimensional vectors.

The architectures we use for learning the extracted features are binarized multilayer perceptrons of dimensions 512-2048-10 for CIFAR-10, and 512-2048-2048-100 for CIFAR-100. The results are shown in Fig. A.5 for datasets split into two subsets of classes. The subsets for CIFAR-10 are chosen by grouping together similar classes : subset 1 consists of vehicle classes and the horse class, while subset 2 consists of the remaining animal classes. For CIFAR-100, the subsets of classes are chosen randomly.

We consider three settings for CIFAR-10 and CIFAR-100. Figs. A.5(a) and (d) show the training results for a non-metaplastic setting. We see that, when the network starts learning the second subset of classes, it forgets the first subset of classes rapidly and entirely.

The results for a metaplastic network with task dependent thresholds are shown in Figs. A.5(b) and (e). The metaplasticity parameter m was optimized in each case by hyperparameter grid search. Learning in this situation is highly successful. For CIFAR-10, at the end of learning, accuracy on both subsets approaches the maximum accuracies at the end of subphases of Fig. A.5(a). For CIFAR-100, accuracy on the first subset approaches the maximum one reached in Fig. A.5(d). The accuracy on the second subset is also very high, but remains below the maximum one reached in Fig. A.5(d). These results highlight the applicability of our metaplasticity approach to datasets more sophisticated than MNIST. However, this situation does not correspond to a truly incremental task learning situation, as the output is computed given information on the subset at hand.

For this reason, in Figs. A.5(c) and (f), we use the technique of 'instance normalization' [187] to avoid the task dependency through neurons thresholds. In this situation, during testing, it is not necessary for the network to know to which subset of classes the presented image belongs. The metaplasticity parameter m was again optimized in each case by hyperparameter grid search. We see that incremental learning is achieved, successfully, with final accuracies that do not, however, match the ones seen with task-dependent thresholds in Figs. A.5(b) and (e), highlighting the difficulty of this training situation.

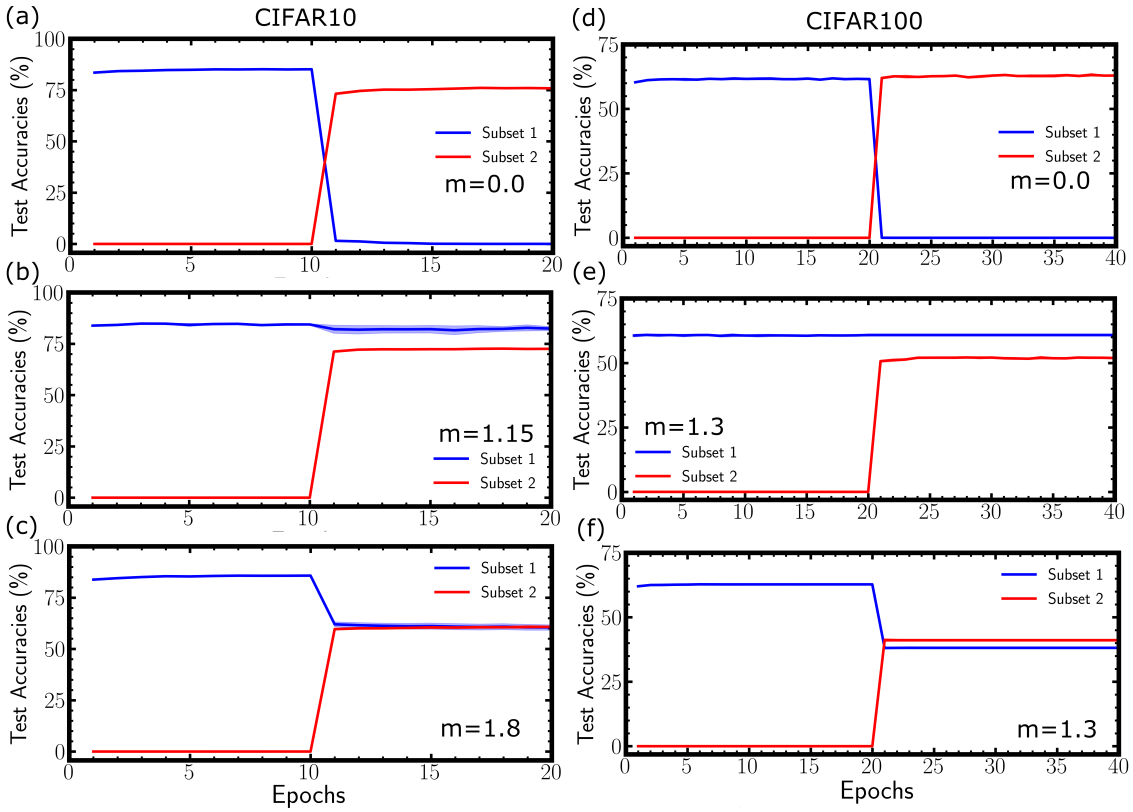

Figure A.5: **Class Incremental Learning** on CIFAR-10 features (a,b,c) and CIFAR-100 features (d,e,f) with the following settings : (a,d) Non-metaplastic (b,e) Metaplastic with task dependent neurons activation thresholds through Batch Normalization (c,f) No dependency on task (Instance Normalization). The curves are averaged over five runs and shadows stand for one standard deviation.

A.10 Increasing Synapse Complexity for Steady-State Continual Learning

In this appendix, we investigate the impact of removing the feedback process linking the slowest hidden variable to the first one, in multiple situations. We let the hidden variables evolve only through the main connections and remove the feedback process: we set $\alpha = 0$ and $f_{\text{meta}} = 1$ in Fig. 2.8(a). The results are listed in the Table 5 for 21 values of the parameters of the synapses, covering cases with more hidden variables and/or slower time scales. In all these situations, we observe some memory signal for Tasks 8 and 9, with varying accuracy depending on the parameter choice. However, the accuracy of Task 7 is always back to near-random guess, suggesting that catastrophic forgetting remains strong in the absence of our model modifications. This result is consistent with our interpretation that the influence of the slowest (last)

Hidden Variables number	Parameters $g_{i,i+1}, \epsilon$	Task 7 Test Acc. (%)	Task 8 Test Acc. (%)	Task 9 Test Acc. (%)	Task 10 Test Acc. (%)
4	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$	10.44	14.86	31.1	97.57
	$2\cdot10^{-2}, 2\cdot10^{-3}, 2\cdot10^{-4}, 2\cdot10^{-5}$	6.92	15.6	37.53	97.77
	$3\cdot10^{-2}, 3\cdot10^{-3}, 3\cdot10^{-4}, 3\cdot10^{-5}$	13.26	16.56	36.73	95.76
	$4\cdot10^{-2}, 4\cdot10^{-3}, 4\cdot10^{-4}, 4\cdot10^{-5}$	11.02	11.31	40.08	97.52
	$7\cdot10^{-2}, 7\cdot10^{-3}, 7\cdot10^{-4}, 7\cdot10^{-5}$	11.0	13.77	31.25	97.07
	$9\cdot10^{-2}, 9\cdot10^{-3}, 9\cdot10^{-4}, 9\cdot10^{-5}$	8.57	12.94	23.42	96.01
	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$	11.06	6.42	31.4	96.99
5	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$	10.1	21.21	45.19	97.49
	$2\cdot10^{-2}, 2\cdot10^{-3}, 2\cdot10^{-4}, 2\cdot10^{-5}, 2\cdot10^{-6}$	12.94	12.72	48.04	97.61
	$3\cdot10^{-2}, 3\cdot10^{-3}, 3\cdot10^{-4}, 3\cdot10^{-5}, 3\cdot10^{-6}$	9.62	14.03	35.06	96.79
	$4\cdot10^{-2}, 4\cdot10^{-3}, 4\cdot10^{-4}, 4\cdot10^{-5}, 4\cdot10^{-6}$	17.29	16.49	42.48	97.5
	$7\cdot10^{-2}, 7\cdot10^{-3}, 7\cdot10^{-4}, 7\cdot10^{-5}, 7\cdot10^{-6}$	10.06	13.86	38.81	96.7
	$9\cdot10^{-2}, 9\cdot10^{-3}, 9\cdot10^{-4}, 9\cdot10^{-5}, 9\cdot10^{-6}$	15.16	15.23	45.78	97.07
	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$	10.05	18.86	38.99	97.13
6	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}$	7.17	13.07	31.76	96.35
	$2\cdot10^{-2}, 2\cdot10^{-3}, 2\cdot10^{-4}, 2\cdot10^{-5}, 2\cdot10^{-6}, 2\cdot10^{-7}$	11.00	18.91	35.88	96.93
	$3\cdot10^{-2}, 3\cdot10^{-3}, 3\cdot10^{-4}, 3\cdot10^{-5}, 3\cdot10^{-6}, 3\cdot10^{-7}$	13.09	18.05	45.85	97.44
	$4\cdot10^{-2}, 4\cdot10^{-3}, 4\cdot10^{-4}, 4\cdot10^{-5}, 4\cdot10^{-6}, 4\cdot10^{-7}$	11.45	17.47	44.48	97.53
	$7\cdot10^{-2}, 7\cdot10^{-3}, 7\cdot10^{-4}, 7\cdot10^{-5}, 7\cdot10^{-6}, 7\cdot10^{-7}$	11.15	17.08	58.89	97.72
	$9\cdot10^{-2}, 9\cdot10^{-3}, 9\cdot10^{-4}, 9\cdot10^{-5}, 9\cdot10^{-6}, 9\cdot10^{-7}$	9.54	16.29	42.53	96.75
	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$	8.7	14.87	44.74	97.6

Table A.5: Control experiment to verify that the feedback process we introduce on the slowest hidden variable is required. The results in this table correspond to the ten permuted MNISTs experiment with hidden variables evolving through the main connections only (as in [116]) for a wide range of parameters.

hidden variable over the fastest one through the main connections is too weak to protect the first variable from the strongly correlated gradients related to the current task.

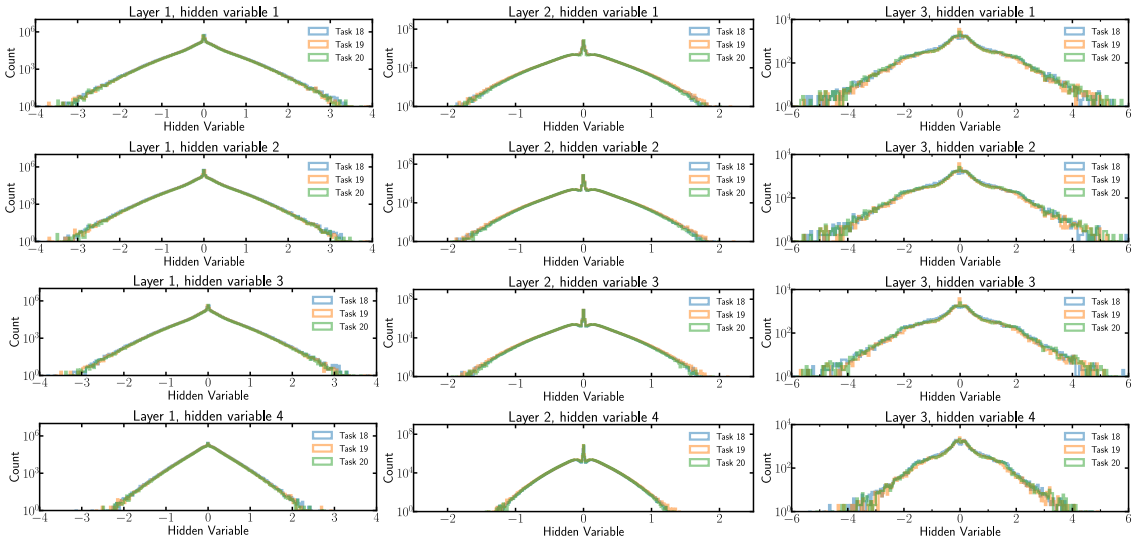

Figure A.6: **Steady-state regime** Distribution of hidden variables for each layer (horizontally) and each hidden variable (vertically). Distributions are superimposed over the three most recent tasks. We observe that the steady state have been reached.

Appendix B

Scaling Equilibrium Propagation to Deep ConvNets by Drastically Reducing its Gradient Estimator Bias

B.1 Gradients of BPTT

In this appendix, we define $\nabla^{\text{BPTT}}(t)$, the gradient computed by BPTT truncated to the last t time steps $(T-t,\dots,T)$. To do this, let us rewrite Eq. (3.1) as $s_{t+1} = \frac{\partial \Phi}{\partial s}(x, s_t, \theta_t = \theta)$, where θ_t denotes the parameter at time step t, the value θ being shared across all time steps. We consider the loss after T time steps $\mathscr{L} = \ell(s_T, y)$. Rewriting the dynamics in such a way enables us to define $\frac{\partial \mathscr{L}}{\partial \theta_t}$ as the sensitivity of the loss with respect to θ_t, when $\theta_0,\dots,\theta_{t-1},\theta_{t+1},\dots,\theta_{T-1}$ remain fixed (set to the value θ). With these notations, the gradient computed by BPTT truncated to the last t time steps is

$$\nabla^{\text{BPTT}}(t) = \frac{\partial \mathscr{L}}{\partial \theta_{T-t}} + \dots + \frac{\partial \mathscr{L}}{\partial \theta_{T-1}}. \tag{B.1}$$

B.2 Error terms in the estimates of the loss gradient

In this appendix, we prove Lemma 3 which shows that $\widehat{\nabla}^{\text{EP}}_{\text{sym}}(\beta)$ is a better estimate of $-\frac{\partial \mathscr{L}^\star}{\partial \theta}$ than $\widehat{\nabla}^{\text{EP}}(\beta)$. First, we recall the theorem proved in [71].

Theorem 1 ([71])*:*

$$\frac{d}{d\beta}\bigg|_{\beta=0} \frac{\partial \Phi}{\partial \theta}(x, s^\beta_\star, \theta) = -\frac{\partial \mathscr{L}^\star}{\partial \theta}. \tag{B.2}$$

We also recall that the two estimates (one-sided and symmetric) are, by definition:

$$\widehat{\nabla}^{\mathrm{EP}}(\beta) \stackrel{\Delta}{=} \frac{1}{\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s_\star^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s_\star, \theta)\right),$$
$$\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta) \stackrel{\Delta}{=} \frac{1}{2\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s_\star^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s_\star^{-\beta}, \theta)\right).$$

Finally we recall Lemma 3, for readability.

Lemma 3*:* Provided the function $\beta \mapsto \frac{\partial \Phi}{\partial \theta}(x, s_\star^\beta, \theta)$ is three times differentiable, we have, as $\beta \to 0$:

$$\widehat{\nabla}^{\mathrm{EP}}(\beta) = -\frac{\partial \mathscr{L}^\star}{\partial \theta} + \frac{\beta}{2}\frac{d^2}{d\beta^2}\bigg|_{\beta=0} \frac{\partial \Phi}{\partial \theta}(s_\star^\beta, \theta) + O(\beta^2),$$
$$\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta) = -\frac{\partial \mathscr{L}^\star}{\partial \theta} + O(\beta^2).$$

Proof of Lemma 3. Let us define

$$f(\beta) \stackrel{\Delta}{=} \frac{\partial \Phi}{\partial \theta}(x, s_\star^\beta, \theta).$$

The formula of Theorem rewrites

$$f'(0) = -\frac{\partial \mathscr{L}^\star}{\partial \theta}.$$

As $\beta \to 0$, we have the Taylor expansion

$$f(\beta) = f(0) + \beta f'(0) + \frac{\beta^2}{2} f''(0) + O(\beta^3). \tag{B.3}$$

With these notations, the one-sided estimate reads

$$\begin{aligned}\widehat{\nabla}^{\mathrm{EP}}(\beta) &= \frac{1}{\beta}\big(f(\beta) - f(0)\big)\\ &= f'(0) + \frac{\beta}{2} f''(0) + O(\beta^2)\\ &= -\frac{\partial \mathscr{L}^\star}{\partial \theta} + \frac{\beta}{2}\frac{d^2}{d\beta^2}\bigg|_{\beta=0} \frac{\partial \Phi}{\partial \theta}(x, s_\star^\beta, \theta) + O(\beta^2).\end{aligned}$$

We can also write a Taylor expansion around 0 at the point $-\beta$. We have

$$f(-\beta) = f(0) - \beta f'(0) + \frac{\beta^2}{2} f''(0) + O(\beta^3). \tag{B.4}$$

Subtracting Eq. B.4 from Eq. B.3, we can rewrite the symmetric difference estimate as

$$\begin{aligned}\widehat{\nabla}^{\text{EP}}_{\text{sym}}(\beta) &= \frac{1}{2\beta}\big(f(\beta) - f(-\beta)\big)\\ &= f'(0) + O(\beta^2)\\ &= -\frac{\partial \mathscr{L}^\star}{\partial \theta} + O(\beta^2).\end{aligned}$$

The derivative to the third order of f is only used to get the $O(\beta^3)$ term in the expansion Eq. (B.3), it can be changed into $o(\beta^2)$ if we only assume f twice differentiable. □

B.3 Pseudo code

B.3.1 Random one-sided estimation of the loss gradient

In this appendix, we define the random one-sided estimation used in this work and by [71, 198].

Algorithm 4 EP with random one-sided estimation of the loss gradient. We omit the activation function σ for clarity.

Input: x, y, θ, η.

Output: θ.

$s_0 \leftarrow 0$

for $t = 0$ to T **do** ▷ First phase.

  $s_{t+1} \leftarrow \frac{\partial \Phi}{\partial s}(x, s_t, \theta)$

end for

$s_\star \leftarrow s_T$

$\beta \leftarrow \beta \times \text{Bernoulli}(1, -1)$ ▷ Random sign.

$s_0^\beta \leftarrow s_\star$

for $t = 0$ to K **do** ▷ Second phase.

  $s_{t+1}^\beta \leftarrow \frac{\partial \Phi}{\partial s}(x, s_t^\beta, \theta) - \beta \frac{\partial \ell}{\partial s}(s_t^\beta, y)$

end for

$s_\star^\beta \leftarrow s_K^\beta$

$\nabla_\theta^{\text{EP}} \leftarrow \frac{1}{\beta}\left(\frac{\partial \Phi}{\partial \theta}(s_\star^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(s_\star, \theta)\right)$

$\theta \leftarrow \theta + \eta \nabla_\theta^{\text{EP}}$

return θ

B.3.2 Symmetric difference estimation of the loss gradient

In this appendix, we define the estimation procedure using a symmetric difference estimate introduced in this work.

Algorithm 5 EP with symmetric difference estimation of the loss gradient. We omit the activation function σ for clarity.

Input: x, y, θ, η.

Output: θ.

$s_0 \leftarrow 0$
for $t = 0$ to T **do**
  $s_{t+1} \leftarrow \frac{\partial \Phi}{\partial s}(x, s_t, \theta)$ ▷ First phase.
end for
$s_\star \leftarrow s_T$ ▷ Store the free steady state.
$s_0^\beta \leftarrow s_\star$
for $t = 0$ to K **do**
  $s_{t+1}^\beta \leftarrow \frac{\partial \Phi}{\partial s}(x, s_t^\beta, \theta) - \beta \frac{\partial \ell}{\partial s}(s_t^\beta, y)$ ▷ Second phase.
end for
$s_\star^\beta \leftarrow s_K^\beta$
$s_0^{-\beta} \leftarrow s_\star$ ▷ Back to the free steady state.
for $t = 0$ to K **do**
  $s_{t+1}^{-\beta} \leftarrow \frac{\partial \Phi}{\partial s}(x, s_t^{-\beta}, \theta) + \beta \frac{\partial \ell}{\partial s}(s_t^{-\beta}, y)$ ▷ Third phase.
end for
$s_\star^{-\beta} \leftarrow s_K^{-\beta}$
$\widehat{\nabla}_\theta^{\text{EP}} \leftarrow \frac{1}{2\beta}\left(\frac{\partial \Phi}{\partial \theta}(s_\star^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(s_\star^{-\beta}, \theta)\right)$
$\theta \leftarrow \theta + \eta \widehat{\nabla}_\theta^{\text{EP}}$
return θ

B.4 Convolutional Recurrent Neural Networks

Throughout this section, N^{conv} and N^{fc} denote respectively the number of convolutional layers and fully connected layers in the convolutional RNN, and $N^{\text{tot}} \triangleq N^{\text{conv}} + N^{\text{fc}}$. The neuron layers are denoted by s and range from $s^0 = x$ the input to the output $s^{N^{\text{tot}}}$ in the case of squared error, or $s^{N^{\text{tot}}-1}$ in the case of softmax read-out.

B.4.1 Definition of the operations

In this subsection we detail the operations involved in the dynamics of a convolutional RNN.

- The 2-D convolution between w with dimension $(C_{\text{out}}, C_{\text{in}}, F, F)$ and an input x of dimensions $(C_{\text{in}}, H_{\text{in}}, W_{\text{in}})$ and stride one is a tensor y of size $(C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ defined by:

$$y_{c,h,w} = (w \star x)_{c,h,w} = B_c + \sum_{i=0}^{C_{\text{in}}-1} \sum_{j=0}^{F-1} \sum_{k=0}^{F-1} w_{c,i,j,k} x_{i,j+h,k+w}, \tag{B.5}$$

  where B_c is a channel-wise bias.

- The 2-D transpose convolution of y by $\tilde{w}$ is then defined in this work as the gradient of the 2-D convolution with respect to its input:

$$(\tilde{w} \star y) \triangleq \frac{\partial (w \star x)}{\partial x} \cdot y \tag{B.6}$$

- The dot product "$\bullet$" generalized to pairs of tensors of same shape (C, H, W):

$$a \bullet b = \sum_{c=0}^{C-1} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} a_{c,h,w} b_{c,h,w}. \tag{B.7}$$

- The pooling operation $\mathscr{P}$ with stride F and filter size F of x:

$$\mathscr{P}_F(x)_{c,h,w} = \max_{i,j \in [0,F-1]} \left\{ x_{c,F(h-1)+1+i,F(w-1)+1+j} \right\}, \tag{B.8}$$

  with relative indices of maximums within each pooling zone given by:

$$\mathrm{ind}_{\mathscr{P}}(x)_{c,h,w} = \operatorname*{argmax}_{i,j \in [0,F-1]} \left\{ x_{c,F(h-1)+1+i,F(w-1)+1+j} \right\} = (i^\star(x,h), j^\star(x,w)). \tag{B.9}$$

- The unpooling operation $\mathscr{P}^{-1}$ of y with indices $\mathrm{ind}_{\mathscr{P}}(x)$ is then defined as:

$$\mathscr{P}^{-1}(y, \mathrm{ind}_{\mathscr{P}}(x))_{c,h,w} = \sum_{i,j} y_{c,i,j} \cdot \delta_{h,F(i-1)+1+i^\star(x,h)} \cdot \delta_{w,F(j-1)+1+j^\star(x,w)}, \tag{B.10}$$

  which consists in filling a tensor with the same dimensions as x with the values of y at the indices $\mathrm{ind}_{\mathscr{P}}(x)$, and zeroes elsewhere. For notational convenience, we omit to write explicitly the dependence on the indices except when appropriate.

- The flattening operation $\mathscr{F}$ is defined as reshaping a tensor of dimensions (C, H, W) to $(1, CHW)$. We denote by $\mathscr{F}^{-1}$ its inverse.

B.4.2 Convolutional RNNs with symmetric connections

In this section, we write explicitly the dynamics and the learning rules applied for the convolutional architecture with symmetric connections, for the Squared loss function and the Cross-Entropy loss function, for the one-sided and symmetric estimates.

B.4.2.1 Squared Error loss

Equations of the dynamics. In this case, the dynamics read:

$$\begin{cases} s_{t+1}^{n+1} = \sigma\left(\mathscr{P}(w_{n+1} \star s_t^n) + \tilde{w}_{n+2} \star \mathscr{P}^{-1}(s_t^{n+2})\right), & \forall n \in [0, N^{\text{conv}} - 2] \\ s_{t+1}^{N^{\text{conv}}} = \sigma\left(\mathscr{P}(w_{N^{\text{conv}}} \star s_t^{N^{\text{conv}}-1}) + \mathscr{F}^{-1}(w_{N^{\text{conv}}+1}{}^{\top} \cdot s_t^{N^{\text{conv}}+1})\right), \\ s_{t+1}^{N^{\text{conv}}+1} = \sigma\left(w_{N^{\text{conv}}+1} \cdot \mathscr{F}(s_t^{N^{\text{conv}}}) + w_{N^{\text{conv}}+2}{}^{\top} \cdot s_t^{N^{\text{conv}}+2}\right), \\ s_{t+1}^{n+1} = \sigma\left(w_{n+1} \cdot s_t^n + w_{n+2}{}^{\top} \cdot s_t^{n+2}\right), & \forall n \in [N^{\text{conv}} + 1, N^{\text{tot}} - 2] \\ s_{t+1}^{N^{\text{tot}}} = \sigma\left(w_{N^{\text{tot}}} \cdot s_t^{N^{\text{tot}}-1}\right) + \beta(y - s^{N^{\text{tot}}}), & \text{with } \beta = 0 \text{ during the first phase,} \end{cases} \tag{B.11}$$

where we take the convention $s^0 = x$. In this case, we have $\hat{y} = s_{t+1}^{N^{\text{tot}}}$. Considering the function:

$$\begin{aligned} \Phi(x, s^1, \cdots, s^{N^{\text{tot}}}) = &\sum_{n=N_{\text{conv}}+2}^{N_{\text{tot}}-1} s^{n+1\top} \cdot w_{n+1} \cdot s^n + s^{N_{\text{conv}}+1\top} \cdot w_{N_{\text{conv}}+1} \cdot \mathscr{F}(s_t^{N_{\text{conv}}}) \\ &+ \sum_{n=1}^{N_{\text{conv}}-1} s^{n+1} \bullet \mathscr{P}\left(w_{n+1} \star s^n\right) + s^1 \bullet \mathscr{P}(w_1 \star x), \end{aligned}$$

when ignoring the activation function, we have:

$$\forall n \in [1, N^{\text{tot}}]: \quad s_t^n \approx \frac{\partial \Phi}{\partial s^n}. \tag{B.12}$$

Note that in the case of the Squared Error loss function, the dynamics of the output layer derive from Φ as it can be seen by Eq. (B.12).

Learning rules for the one-sided EP estimator. In this case, the learning rules read:

$$\begin{cases} \forall n \in [N_{\text{conv}} + 2, N_{\text{tot}} - 1]: \quad \Delta w_n = \frac{1}{\beta}\left(s_\star^{n+1,\beta} \cdot s_\star^{n,\beta^\top} - s_\star^{n+1} \cdot s_\star^{n^\top}\right) \\ \Delta w_{N_{\text{conv}}+1} = \frac{1}{\beta}\left(s_\star^{N_{\text{conv}}+1,\beta} \cdot \mathscr{F}\left(s_\star^{N_{\text{conv}},\beta}\right)^\top - s_\star^{N_{\text{conv}}+1} \cdot \mathscr{F}\left(s_\star^{N_{\text{conv}}}\right)^\top\right) \\ \forall n \in [1, N_{\text{conv}} - 1]: \quad \Delta w_{n+1} = \frac{1}{\beta}\left(\mathscr{P}^{-1}(s_\star^{n+1,\beta}) \star s_\star^{n,\beta} - \mathscr{P}^{-1}(s_\star^{n+1}) \star s_\star^n\right) \\ \Delta w_1 = \frac{1}{\beta}\left(\mathscr{P}^{-1}(s_\star^{1,\beta}) \star x - \mathscr{P}^{-1}(s_\star^1) \star x\right) \end{cases}, \tag{B.13}$$

Learning rules for the symmmetric EP estimator. In this case, the learning rules read:

$$\begin{cases} \forall n \in [N_{\text{conv}} + 2, N_{\text{tot}} - 1]: \quad \Delta w_n = \frac{1}{2\beta}\left(s_\star^{n+1,\beta} \cdot s_\star^{n,\beta^\top} - s_\star^{n+1,-\beta} \cdot s_\star^{n,-\beta^\top}\right) \\ \Delta w_{N_{\text{conv}}+1} = \frac{1}{2\beta}\left(s_\star^{N_{\text{conv}}+1,\beta} \cdot \mathscr{F}\left(s_\star^{N_{\text{conv}},\beta}\right)^\top - s_\star^{N_{\text{conv}}+1,-\beta} \cdot \mathscr{F}\left(s_\star^{N_{\text{conv}},-\beta}\right)^\top\right) \\ \forall n \in [1, N_{\text{conv}} - 1]: \quad \Delta w_{n+1} = \frac{1}{2\beta}\left(\mathscr{P}^{-1}(s_\star^{n+1,\beta}) \star s_\star^{n,\beta} - \mathscr{P}^{-1}(s_\star^{n+1,-\beta}) \star s_\star^{n,-\beta}\right) \\ \Delta w_1 = \frac{1}{2\beta}\left(\mathscr{P}^{-1}(s_\star^{1,\beta}) \star x - \mathscr{P}^{-1}(s_\star^{1,-\beta}) \star x\right) \end{cases}, \tag{B.14}$$

B.4.2.2 Cross-Entropy loss

Equations of the dynamics. In this case, the dynamics read:

$$\begin{cases} s_{t+1}^{n+1} = \sigma\left(\mathscr{P}(w_{n+1} \star s_t^n) + \tilde{w}_{n+2} \star \mathscr{P}^{-1}(s_t^{n+2})\right), \qquad \forall n \in [0, N^{\text{conv}} - 2] \\ s_{t+1}^{N^{\text{conv}}} = \sigma\left(\mathscr{P}(w_{N^{\text{conv}}} \star s_t^{N^{\text{conv}}-1}) + \mathscr{F}^{-1}({w_{N^{\text{conv}}+1}}^{\top} \cdot s_t^{N^{\text{conv}}+1})\right), \\ s_{t+1}^{N^{\text{conv}}+1} = \sigma\left(w_{N^{\text{conv}}+1} \cdot \mathscr{F}(s_t^{N^{\text{conv}}}) + {w_{N^{\text{conv}}+2}}^{\top} \cdot s_t^{N^{\text{conv}}+2}\right), \\ s_{t+1}^{n+1} = \sigma\left(w_{n+1} \cdot s_t^n + {w_{n+2}}^{\top} \cdot s_t^{n+2}\right), \qquad \forall n \in [N^{\text{conv}} + 1, N^{\text{tot}} - 3] \\ s_{t+1}^{N^{\text{tot}}-1} = \sigma\left(w_{N^{\text{tot}}-1} \cdot s_t^{N^{\text{tot}}-2}\right) + \beta {w_{\text{out}}}^{\top} \cdot (y - \hat{y}) \quad \text{with } \beta = 0 \text{ during the first phase,} \\ \hat{y} = \text{softmax}(w_{\text{out}} \cdot s_t^{N^{\text{tot}}-1}), \end{cases} \tag{B.15}$$

where we keep again the convention $s^0 = x$. Considering the function:

$$\begin{aligned} \Phi(x, s^1, \cdots, s^{N^{\text{tot}}-1}) = &\sum_{n=N_{\text{conv}}+1}^{N_{\text{tot}}-2} {s^{n+1}}^{\top} \cdot w_n \cdot s^n + s^{N_{\text{conv}}+1} \cdot w_{N_{\text{conv}}+1} \cdot \mathscr{F}(s_t^{N_{\text{conv}}}) \\ &+ \sum_{n=1}^{N_{\text{conv}}-1} s^{n+1} \bullet \mathscr{P}\left(w_{n+1} \star s^n\right) + s^1 \bullet \mathscr{P}\left(w_1 \star x\right), \end{aligned}$$

when ignoring the activation function, we have:

$$\forall n \in [1, N^{\text{tot}} - 1]: \quad s_t^n \approx \frac{\partial \Phi}{\partial s^n}, \qquad \hat{y} = \text{softmax}(w_{\text{out}} \cdot s_t^{N^{\text{tot}}-1}). \tag{B.16}$$

Note that in this case and contrary to the Squared Error loss function, the dynamics of the output layer do not derive from the primitive function Φ, as it can be seen from Eq. (B.16)

Learning rules for the one-sided EP estimator. In this case, the learning rules read:

$$\begin{cases} \Delta w_{\text{out}} = -\left(\hat{y}_\star^\beta - y\right) \cdot s_\star^{\beta, N^\top}. \\ \forall n \in [N_{\text{conv}} + 2, N_{\text{tot}} - 2]: \quad \Delta w_n = \frac{1}{\beta}\left(s_\star^{n+1,\beta} \cdot s_\star^{n,\beta^\top} - s_\star^{n+1} \cdot s_\star^{n^\top}\right) \\ \Delta w_{N_{\text{conv}}+1} = \frac{1}{\beta}\left(s_\star^{N_{\text{conv}}+1,\beta} \cdot \mathscr{F}\left(s_\star^{N_{\text{conv}},\beta}\right)^\top - s_\star^{N_{\text{conv}}+1} \cdot \mathscr{F}\left(s_\star^{N_{\text{conv}}}\right)^\top\right) \\ \forall n \in [1, N_{\text{conv}} - 1]: \quad \Delta w_{n+1} = \frac{1}{\beta}\left(\mathscr{P}^{-1}(s_\star^{n+1,\beta}) \bullet s_\star^{n,\beta} - \mathscr{P}^{-1}(s_\star^{n+1}) \bullet s_\star^n\right) \\ \Delta w_1 = \frac{1}{\beta}\left(\mathscr{P}^{-1}(s_\star^{1,\beta}) \bullet x - \mathscr{P}^{-1}(s_\star^1) \bullet x\right) \end{cases}. \tag{B.17}$$

Learning rules for the symmetric EP estimator. In this case, the learning rules read:

$$\begin{cases} \Delta w_{\text{out}} = -\frac{1}{2}\left(\left(\widehat{y}_\star^{\beta} - y\right)\cdot s_\star^{\beta,N^\top} + \left(\widehat{y}_\star^{-\beta} - y\right)\cdot s_\star^{-\beta,N^\top}\right). \\ \forall n \in [N_{\text{conv}}+2, N_{\text{tot}}-2]: \quad \Delta w_n = \frac{1}{2\beta}\left(s_\star^{n+1,\beta}\cdot s_\star^{n,\beta^\top} - s_\star^{n+1,-\beta}\cdot s_\star^{n,-\beta^\top}\right) \\ \Delta w_{N_{\text{conv}}+1} = \frac{1}{2\beta}\left(s_\star^{N_{\text{conv}}+1,\beta}\cdot \mathscr{F}\left(s_\star^{N_{\text{conv}},\beta}\right)^\top - s_\star^{N_{\text{conv}}+1,-\beta}\cdot \mathscr{F}\left(s_\star^{N_{\text{conv}},-\beta}\right)^\top\right) \\ \forall n \in [1, N_{\text{conv}}-1]: \quad \Delta w_{n+1} = \frac{1}{2\beta}\left(\mathscr{P}^{-1}(s_\star^{n+1,\beta})\bullet s_\star^{n,\beta} - \mathscr{P}^{-1}(s_\star^{n+1,-\beta})\bullet s_\star^{n,-\beta}\right) \\ \Delta w_1 = \frac{1}{2\beta}\left(\mathscr{P}^{-1}(s_\star^{1,\beta})\bullet x - \mathscr{P}^{-1}(s_\star^{1,-\beta})\bullet x\right) \end{cases} . \tag{B.18}$$

B.4.2.3 Implementation details in PyTorch.

The equation of the dynamics as well as the EP estimates computation can be expressed as derivatives of the primitive function Φ. Therefore, the automatic differentiation framework provided by PyTorch can be leveraged to implement implicitly the equations of the dynamics and the EP estimates computation by differentiating Φ. Although this implementation is slower than explicitly implementing the equations of the dynamics, it is more flexible in terms of network architecture as Φ is relatively easy to compute.

B.4.3 Convolutional RNNs with asymmetric connections

In this section, we write the explicit definition of the dynamics and the learning rule of a convolutional architecture with asymmetric connections where forward and backward connections are no longer constrained to be equal-valued. In this setting, we use the Cross-Entropy loss function along with a softmax readout to implement the output layer of the network.

Equations of the dynamics. In this setting, the dynamics Eq. (B.15) have simply to be changed into:

$$\begin{cases} s_{t+1}^{n+1} = \sigma\left(\mathscr{P}(w_{n+1}^{\text{f}} \star s_t^n) + \tilde{w}_{n+2}^{\text{b}} \star \mathscr{P}^{-1}(s_t^{n+2})\right), \qquad \forall n \in [0, N^{\text{conv}}-2] \\ s_{t+1}^{N^{\text{conv}}} = \sigma\left(\mathscr{P}(w_{N^{\text{conv}}}^{\text{f}} \star s_t^{N^{\text{conv}}-1}) + \mathscr{F}^{-1}(w_{N^{\text{conv}}+1}^{\text{b}})^\top \cdot s_t^1)\right), \\ s_{t+1}^{N^{\text{conv}}+1} = \sigma\left(w_{N^{\text{conv}}+1}^{\text{f}} \cdot \mathscr{F}(s_t^{N^{\text{conv}}}) + w_{N^{\text{conv}}+2}{}^{\text{b}^\top} \cdot s_t^{N^{\text{conv}}+2}\right), \\ s_{t+1}^{n+1} = \sigma\left(w_{n+1}^{\text{f}} \cdot s_t^n + w_{n+2}^{\text{b}}{}^\top \cdot s_t^{n+2}\right), \qquad \forall n \in [N^{\text{conv}}+1, N^{\text{tot}}-3] \\ s_{t+1}^{N^{\text{tot}}-1} = \sigma\left(w_{N^{\text{tot}}-1}^{\text{f}} \cdot s_t^{N^{\text{tot}}-2}\right) + \beta w_{\text{out}}{}^\top \cdot (y - \hat{y}), \\ \hat{y} = \text{softmax}(w_{\text{out}} \cdot s_t^{N^{\text{tot}}-1}), \end{cases} \tag{B.19}$$

where we distinguish now between forward and backward connections: $w_n^{\text{f}} \neq w_n^{\text{b}} \quad \forall n \in [1, N_{\text{tot}}-2]$.

Original Vector Field learning rule (VF). The symmetric version of the original Vector Field learning rule is defined as:

$$\widehat{\nabla}^{\text{VF}}_{\text{sym}}(\beta) \triangleq \frac{1}{2\beta}\frac{\partial F}{\partial \theta}(x, s_\star, \theta)^\top \cdot \left(s_\star^{\beta} - s_\star^{-\beta}\right), \tag{B.20}$$

which yields in the case of softmax read-out:

$$\begin{cases} \Delta w_{\text{out}} = -\frac{1}{2}\left(\left(\widehat{y}_\star^{\beta} - y\right)\cdot s_\star^{\beta,N^\top} + \left(\widehat{y}_\star^{-\beta} - y\right)\cdot s_\star^{-\beta,N^\top}\right). \\ \forall n \in [N_{\text{conv}}+2, N_{\text{tot}}-2]: \quad \Delta w_n^{\text{f}} = \frac{1}{2\beta}\left(s_\star^{n+1,\beta} - s_\star^{n+1,-\beta}\right)\cdot s_\star^{n^\top} \\ \forall n \in [N_{\text{conv}}+2, N_{\text{tot}}-2]: \quad \Delta w_n^{\text{b}} = \frac{1}{2\beta} s_\star^{n+1}\cdot\left(s_\star^{n,\beta} - s_\star^{n,-\beta}\right)^\top \\ \Delta w^{\text{f}}_{N_{\text{conv}}+1} = \frac{1}{2\beta}\left(s_\star^{N_{\text{conv}}+1,\beta} - s_\star^{N_{\text{conv}}+1,-\beta}\right)\cdot \mathscr{F}\left(s_\star^{N_{\text{conv}}}\right)^\top \\ \Delta w^{\text{b}}_{N_{\text{conv}}+1} = \frac{1}{2\beta} s_\star^{N_{\text{conv}}+1}\cdot\left(\mathscr{F}\left(s_\star^{N_{\text{conv}},\beta}\right) - \mathscr{F}\left(s_\star^{N_{\text{conv}},-\beta}\right)\right)^\top \\ \forall n \in [1, N_{\text{conv}}-1]: \quad \Delta w_{n+1}^{\text{f}} = \frac{1}{2\beta}\left(\mathscr{P}^{-1}(s_\star^{n+1,\beta}) - \mathscr{P}^{-1}(s_\star^{n+1,-\beta})\right)\bullet s_\star^{n} \\ \forall n \in [1, N_{\text{conv}}-1]: \quad \Delta w_{n+1}^{\text{b}} = \frac{1}{2\beta}\mathscr{P}^{-1}(s_\star^{n+1})\bullet\left(s_\star^{n,\beta} - s_\star^{n,-\beta}\right) \\ \Delta w_1 = \frac{1}{2\beta}\left(\mathscr{P}^{-1}(s_\star^{1,\beta})\bullet x - \mathscr{P}^{-1}(s_\star^{1,-\beta})\bullet x\right) \end{cases}. \tag{B.21}$$

Importantly, note that $\Delta w_n^{\text{f}} \neq \Delta w_n^{\text{b}} \quad \forall n \in [1, N_{\text{tot}}-2]$.

Kolen-Pollack algorithm. When forward and backward weights have a common gradient estimate, and a weight decay term λ, they converge to the same values. We recall the proof, noting t the iteration step and taking the notations of section 3.3.3, the update rule follows:

$$\begin{cases} \theta_{\text{f}}(t+1) = \theta_{\text{f}}(t) + \Delta\theta_{\text{f}} \\ \theta_{\text{b}}(t+1) = \theta_{\text{b}}(t) + \Delta\theta_{\text{b}} \end{cases}.$$

We can then write

$$\begin{aligned} \theta_{\text{f}}(t+1) - \theta_{\text{b}}(t+1) &= \theta_{\text{f}}(t) - \theta_{\text{b}}(t) + \Delta\theta_{\text{f}} - \Delta\theta_{\text{b}} \\ &= \theta_{\text{f}}(t) - \theta_{\text{b}}(t) - \eta\lambda\left(\theta_{\text{f}}(t) - \theta_{\text{b}}(t)\right) \\ &= (1-\eta\lambda)\left(\theta_{\text{f}}(t) - \theta_{\text{b}}(t)\right), \end{aligned}$$

where we use the fact that the estimates are the same for both parameters, such that they cancel out. Then by recursion :

$$\theta_{\text{f}}(t) - \theta_{\text{b}}(t) = (1-\eta\lambda)^t\left(\theta_{\text{f}}(0) - \theta_{\text{b}}(0)\right) \underset{t\to\infty}{\to} 0, \quad \text{since} \quad |1-\eta\lambda| < 1.$$

Kolen-Pollack Vector Field learning rule (KP-VF). We remind here that the new learning rule proposed in this paper to train convNets with asymmetric connections is defined as:

$$\begin{cases} \Delta\theta_{\mathrm{f}} = \eta\left(\widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta) - \lambda\theta_{\mathrm{f}}\right) \\ \Delta\theta_{\mathrm{b}} = \eta\left(\widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta) - \lambda\theta_{\mathrm{b}}\right) \end{cases}, \quad \text{with} \quad \widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta) = \frac{1}{2}(\overline{\nabla^{\mathrm{VF}}_{\theta_{\mathrm{f}}}}(\beta) + \overline{\nabla^{\mathrm{VF}}_{\theta_{\mathrm{b}}}}(\beta)), \tag{B.22}$$

where:

$$\forall \mathrm{i} \in \{\mathrm{f},\mathrm{b}\}, \qquad \overline{\nabla^{\mathrm{VF}}_{\theta_{\mathrm{i}}}}(\beta) = \frac{1}{2\beta}\left(\frac{\partial F}{\partial \theta_{\mathrm{i}}}^{\top}(x, s^{\beta}_{\star}, \theta)\cdot s^{\beta}_{\star} - \frac{\partial F}{\partial \theta_{\mathrm{i}}}^{\top}(x, s^{-\beta}_{\star}, \theta)\cdot s^{-\beta}_{\star}\right). \tag{B.23}$$

More specifically, applying Eq. (B.23) to Eq. (B.19) yields:

$$\begin{cases} \forall n \in [N_{\mathrm{conv}}+2, N_{\mathrm{tot}}-2]: \\ \qquad \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{f}}_n}}(\beta) = \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{b}}_n}}(\beta) = \frac{1}{2\beta}\left(s^{n+1,\beta}_{\star}\cdot s^{n,\beta^{\top}}_{\star} - s^{n+1,-\beta}_{\star}\cdot s^{n,-\beta^{\top}}_{\star}\right) \\ \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{f}}_{N_{\mathrm{conv}}+1}}}(\beta) = \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{b}}_{N_{\mathrm{conv}}+1}}}(\beta) = \\ \qquad \frac{1}{2\beta}\left(s^{N_{\mathrm{conv}}+1,\beta}_{\star}\cdot \mathscr{F}\left(s^{N_{\mathrm{conv}},\beta}_{\star}\right)^{\top} - s^{N_{\mathrm{conv}}+1,-\beta}_{\star}\cdot \mathscr{F}\left(s^{N_{\mathrm{conv}},-\beta}_{\star}\right)^{\top}\right) \\ \forall n \in [1, N_{\mathrm{conv}}-1]: \\ \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{f}}_{n+1}}}(\beta) = \frac{1}{2\beta}\left(\mathscr{P}^{-1}\left(s^{n+1,\beta}_{\star}, \mathrm{ind}_{\mathscr{P}}\left(w^{\mathrm{f}}_{n+1}\star s^{n,\beta}_{\star}\right)\right)\bullet s^{n,\beta}_{\star}\right. \\ \qquad\qquad \left. -\mathscr{P}^{-1}\left(s^{n+1,-\beta}_{\star}, \mathrm{ind}_{\mathscr{P}}\left(w^{\mathrm{f}}_{n+1}\star s^{n,-\beta}_{\star}\right)\right)\bullet s^{n,-\beta}_{\star}\right) \\ \forall n \in [1, N_{\mathrm{conv}}-1]: \\ \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{b}}_{n+1}}}(\beta) = \frac{1}{2\beta}\left(\mathscr{P}^{-1}\left(s^{n+1,\beta}_{\star}, \mathrm{ind}_{\mathscr{P}}\left(w^{\mathrm{b}}_{n+1}\star s^{n,\beta}_{\star}\right)\right)\bullet s^{n,\beta}_{\star}\right. \\ \qquad\qquad \left. -\mathscr{P}^{-1}\left(s^{n+1,-\beta}_{\star}, \mathrm{ind}_{\mathscr{P}}\left(w^{\mathrm{b}}_{n+1}\star s^{n,-\beta}_{\star}\right)\right)\bullet s^{n,-\beta}_{\star}\right) \end{cases}. \tag{B.24}$$

Combining Eqs. (B.24) with Eq. (B.22) gives the associated parameter updates. The updates for w_1 and w_{out} are the same than those of Eq. (B.21). Importantly, note that while $\forall n \in [N_{\mathrm{conv}}+1, N_{\mathrm{tot}}-2]: \quad \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{f}}_n}}(\beta) = \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{b}}_n}}(\beta)$, we have $\forall n \in [1, N_{\mathrm{conv}}-1]: \quad \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{f}}_n}}(\beta) \neq \overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{b}}_n}}(\beta)$ because of inverse pooling. In other words, the updates of the convolutional filters do not solely depend on the pre and post synaptic activations but also on the location of the maximal elements within each pooling window, itself depending on the filter considered. Hence the motivation to average $\overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{f}}_n}}(\beta)$ and $\overline{\nabla^{\mathrm{VF}}_{w^{\mathrm{b}}_n}}(\beta)$ and use this quantity to update to w^{b}_n and w^{f}_n and apply the Kolen-Pollack technique.

Implementation details in PyTorch. The dynamics in the case of asymmetric connections does not derive from a primitive function Φ. Therefore, it is not possible to implicitly get the dynamics by differentiating one primitive function. A way around is to get the asymmetric dynamics by differentiating one quantity $\tilde{\Phi}^n$ by layer. This quantity is not a primitive function and is especially designed to get the right equations once differentiated. We define $\tilde{\Phi}^n(w^{\mathrm{f}}_n, w^{\mathrm{b}}_{n+1}, s^{n-1}, s^n)$ by:

$$\begin{cases} \forall n \in [1, N_{\text{conv}} - 1] : \tilde{\Phi}^n = s^n \bullet \mathscr{P}(w_n^{\text{f}} \star s^{n-1}) + s^{n+1} \bullet \mathscr{P}(w_{n+1}^{\text{b}} \star s^n) \\ \tilde{\Phi}^{N_{\text{conv}}} = s^{N_{\text{conv}}} \bullet \mathscr{P}(w_{N_{\text{conv}}}^{\text{f}} \star s^{N_{\text{conv}}-1}) + s^{N_{\text{conv}}+1} \cdot w_{N_{\text{conv}}+1}^{\text{b}} \cdot \mathscr{F}(s^{N_{\text{conv}}}) \\ \forall n \in [N_{\text{conv}} + 1, N_{\text{tot}} - 1] : \tilde{\Phi}^n = s^n \cdot w_n^{\text{f}} \cdot s^{n-1} + s^{n+1} \cdot w_{n+1}^{\text{b}} \cdot s^n \\ \tilde{\Phi}^{N_{\text{tot}}-1} = s^{N_{\text{tot}}-1} \cdot w_{N_{\text{tot}}-1}^{\text{f}} \cdot s^{N_{\text{tot}}-2} + \beta \ell(s^{N_{\text{tot}}-1}, y, w_{\text{out}}) \end{cases}, \tag{B.25}$$

where ℓ is defined by Eq. (3.18), and $\beta = 0$ in the first phase. Then, $\forall n \in [1, N^{\text{tot}} - 1]$, the dynamics of Eq. (B.19) read:

$$s_{t+1}^n = \sigma\left(\frac{\partial \tilde{\Phi}^n}{\partial s^n}(w_n^{\text{f}}, w_{n+1}^{\text{b}}, s_t^{n-1}, s_t^n)\right). \tag{B.26}$$

The original VF update of Eq. (B.21) can be written as $\forall n \in [1, N^{\text{tot}} - 1], \forall \text{i} \in \{\text{f}, \text{b}\}$:

$$\Delta w_n^{\text{i}} = \frac{1}{2\beta}\left(\frac{\partial \tilde{\Phi}^n}{\partial w_n^{\text{i}}}(s_\star^{n,\beta}, s_\star^{n-1}) - \frac{\partial \tilde{\Phi}^n}{\partial w_n^{\text{i}}}(s_\star^{n,-\beta}, s_\star^{n-1})\right), \tag{B.27}$$

and Eq. (B.24) as:

$$\overline{\nabla_{w_n^{\text{i}}}^{\text{VF}}}(\beta) = \frac{1}{2\beta}\left(\frac{\partial \tilde{\Phi}^n}{\partial w_n^{\text{i}}}(s_\star^{n,\beta}, s_\star^{n-1,\beta}) - \frac{\partial \tilde{\Phi}^n}{\partial w_n^{\text{i}}}(s_\star^{n,-\beta}, s_\star^{n-1,-\beta})\right). \tag{B.28}$$

B.4.4 Random-sign estimate variance

The results presented in Table 3.1 consists of five runs. In the case of the EP random-sign estimate, one run among the five collapses to random guess similar to the one-sided estimate. In order to test the frequency of such a phenomenon, we performed another five runs with both symmetric and random-sign estimates. The results for each run presented in Table B.1 show that two trials among ten are unstable, confirming further the high variance nature of the random-sign estimate.

B.4.5 Adding dropout

We adapt dropout [31] for convergent RNNs by shutting some units to zero with probability $p < 1$ when computing $\Phi(x, s_t, \theta)$. We multiply the remaining active units by the factor $\frac{1}{1-p}$ to keep the same neural activity on average, so that the learning rule is rescaled by $\left(\frac{1}{1-p}\right)^2$. The dropped out units are the same within one training iteration but they differ across the examples of one mini batch. In our experiments we use $p = 0.1$ on the last convolutional layer before the linear classifier. The results are reported in Table 3.1.

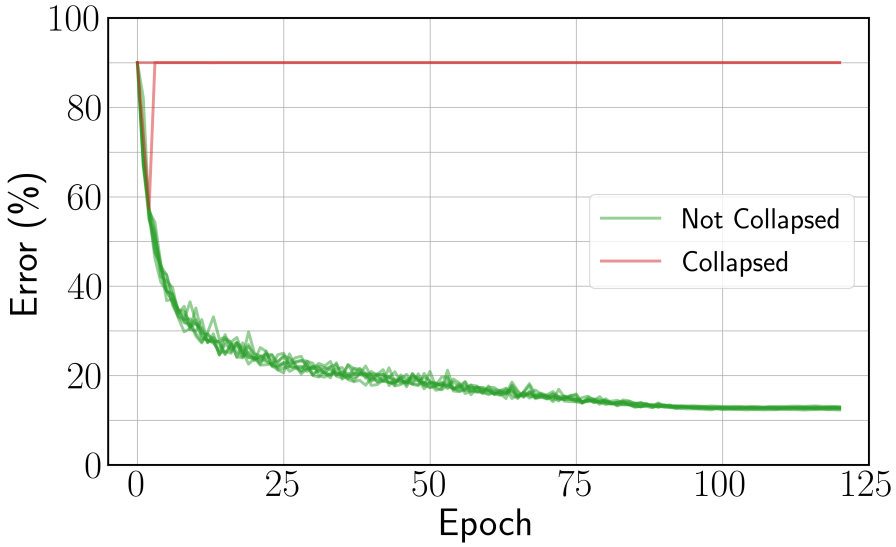

Figure B.1: Test error curve of each run with the Squared Error loss function and random-sign estimate. The two collapsed runs among the ten trials are steady to 90% because in such cases the network typically outputs the same class for each data point.

Table B.1: Best test error comparison between random-sign and symmetric estimates, for ten runs.

Run index	EP random-sign	EP symmetric
1	12.97	12.24
2	12.72	12.31
3	12.30	12.68
4	12.45	12.43
5	12.78	12.57
6	12.66	12.55
7	12.84	12.44
8	12.59	12.52
9	57.32	12.85
10	89.98	12.60
Mean	24.86	**12.52**
w/o collapse	**12.66**	N.A

B.4.6 Changing the activation function

Previous implementations of EP used a shifted hard sigmoid activation function:

$$\sigma(x) = \max(0, \min(x, 1)). \tag{B.29}$$

In their experiments with ConvNets on MNIST, [80] observed saturating units that cannot pass error signals during the second phase. In this work, to mitigate this effect, we have rescaled by a factor $1/2$ the slope of the activation function to ease signal propagation and prevent satura-

tion, therefore changing Eq. (B.29) into:

$$\sigma(x) = \max\left(0, \min\left(\frac{x}{2}, 1\right)\right). \tag{B.30}$$

B.5 Weight alignment for asymmetric connections

The angle α between forward and backward weights is defined as :

$$\alpha = \frac{180}{\pi}\text{Acos}\left(\frac{w^{\text{b}} \bullet w^{\text{f}}}{\|w^{\text{b}}\|\|w^{\text{f}}\|}\right), \quad \text{where} \quad \|w\| = \sqrt{w \bullet w}. \tag{B.31}$$

Fig. 3.5 shows the angle between forward and backward weights during training on CIFAR-10 for both the original VF learning rule (dashed) and the new learning rule inspired by [208].

B.6 Layer-wise comparison of EP estimates

In this section we show on Fig. B.2 more instances of Fig. 3.2 for each layer of the convolutional architecture.

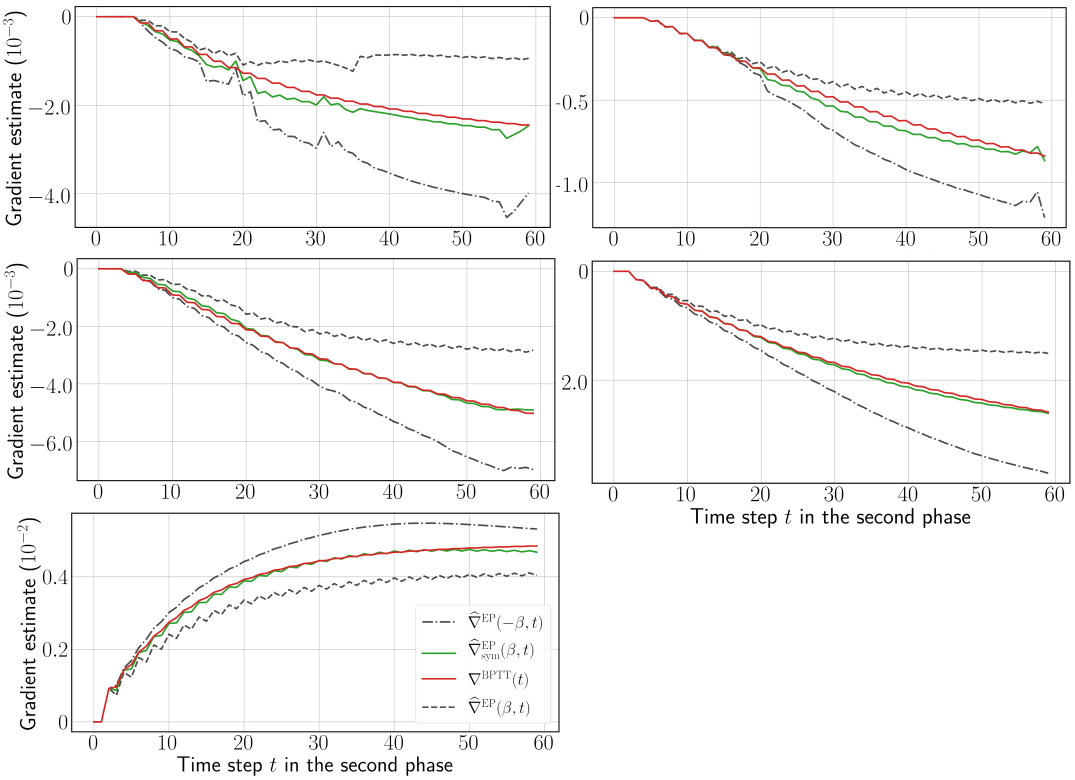

Figure B.2: Layer-wise comparison between EP gradient estimates and BPTT gradients for 5 layers deep CNN on CIFAR-10 Data. Layer index increases from top to bottom, left to right, top-left being the first layer.

Appendix C

Implementation of Ternary Weights with Resistive RAM Using a Single Sense Operation per Synapse

C.1 Training Algorithm of Binarized and Ternary Neural Networks

During the training of BNNs and TNNs, each quantized (binary or ternary) weight is associated with a real hidden weight. This approach to training quantized neural network was introduced in [153] and is presented in Algorithm 6.

The quantized weights are used for computing neuron values (equations (4.1) and (4.2)), as well as the gradients values in the backward pass. However, training steps are achieved by updating the real hidden weights. The quantized weight is then determined by applying to the real value the quantizing function Quantize, which is ϕ for ternary or sign for binary as defined in section 4.1. The quantization of activations is done by applying the same function Quantize, except for real activation, which is done by applying a rectified linear unit ($\text{ReLU}(x) = \max(0, x)$).

Quantized activation functions (ϕ or sign) have zero derivatives almost everywhere, which is an issue for backpropagating the error gradients through the network. A way around this issue is the use of a straight-through estimator [247], which consists in taking the derivative of another function instead of the almost everywhere zero derivatives. Throughout this work, we take the derivative of Hardtanh, which is 1 between -1 and 1 and 0 elsewhere, both for binary and ternary activations.

The simulation code used in this work is available publicly in the Github repository: https://github.com/Laborieux-Axel/Quantized_VGG

Algorithm 6 Training procedure for binary and ternary neural networks. W^{h} are the hidden weights, $\theta^{\mathrm{BN}} = (\gamma_l, \beta_l)$ are Batch Normalization parameters, U_W and U_θ are the parameter updates prescribed by the Adam algorithm [30], (X, y) is a batch of labelled training data, and η is the learning rate. "cache" denotes all the intermediate layers computations needed to be stored for the backward pass. Quantize is either ϕ or sign as defined in section 4.1. " $\cdot$ " denotes the element-wise product of two tensors with compatible shapes.

Input: W^{h}, $\theta^{\mathrm{BN}} = (\gamma_l, \beta_l)$, U_W, U_θ, (X, y), η.
Output: W^{h}, θ^{BN}, U_W, U_θ.

$W^{\mathrm{Q}} \leftarrow \mathrm{Quantize}(W^{\mathrm{h}})$ ▷ Computing quantized weights
$A_0 \leftarrow X$ ▷ Input is not quantized
for $l = 1$ to L **do** ▷ For loop over the layers
  $z_l \leftarrow W_l^{\mathrm{Q}} A_l$ ▷ Matrix multiplication
  $A_l \leftarrow \gamma_l \cdot \frac{z_l - \mathrm{E}(z_l)}{\sqrt{\mathrm{Var}(z_l) + \epsilon}} + \beta_l$ ▷ Batch Normalization [86]
  if $l < L$ **then** ▷ If not the last layer
    $A_l \leftarrow \mathrm{Quantize}(A_l)$ ▷ Activation is quantized
  end if
end for
$\hat{y} \leftarrow A_L$
$C \leftarrow \mathrm{Cost}(\hat{y}, y)$ ▷ Compute mean loss over the batch
$(\partial_W C, \partial_\theta C) \leftarrow \mathrm{Backward}(C, \hat{y}, W^{\mathrm{Q}}, \theta^{\mathrm{BN}}, \mathrm{cache})$ ▷ Cost gradients
$(U_W, U_\theta) \leftarrow \mathrm{Adam}(\partial_W C, \partial_\theta C, U_W, U_\theta)$
$W^{\mathrm{h}} \leftarrow W^{\mathrm{h}} - \eta U_W$
$\theta^{\mathrm{BN}} \leftarrow \theta^{\mathrm{BN}} - \eta U_\theta$
return W^{h}, θ^{BN}, U_W, U_θ