POLITECNICO DI TORINO

Master Course in Nanotechnologies for ICTs

Politecnico
di Torino

Master Thesis

EQUILIBRIUM PROPAGATION FOR RECURRENT NEURAL NETWORKS BASED ON RESISTIVE SWITCHING DEVICES: FROM CIRCUIT IMPLEMENTATION TO SUPERVISED MACHINE LEARNING

Thesis advisors:

Prof. Fernando CORINTO

Prof. Carlo RICCIARDI

Scientific Supervisor:

Dr. Francesco MARRONE

Dr. Gianluca ZOPPO

Candidate

Andrea COSTAMAGNA

July 2021

Summary

The human brain is a biological machine capable of performing real time computing tasks with an extremely reduced power budget. For this reason, brain inspired computing aims at mimicking the brain functioning to implement novel computational paradigms. This requires to work both at the hardware and at the learning algorithm level, toward the optimization of their interplay.

On the algorithmic side, a well established neuronal model is the Additive Model, that treats the brain as a non-linear dynamical system evolving under the influence of external stimuli. This model is classifiable as a continuous time Recurrent Neural Network (RNN). Recently, Bengio and Scellier have designed a learning algorithm named Equilbrium Propagation (EP) [1] capable of efficiently performing learning on this model .

On the hardware side, the capability of ReRAM devices to store in a compact component multiple states of conductance has been explored to implement nanometric solid-state electronic synapses, i.e. the analogue of the biological units at the basis of learning. However, these devices present many challenges in terms of retention [2] and of linearity [3]. Additionally, RNNs can be mapped into continuous time electronic circuitry that, if implemented, would dramatically improve their performances in terms of computing time [4].

This work aims at exploring the interplay of learning algorithms and hardware toward the definition and simulation of a ReRAM-based network for the analogue implementation of the Additive Model.

In the first part, the versatility of Equilibrium Propagation is tested on the solution of two machine learning tasks: reconstruction of corrupted patterns and image classification. While discussing reconstruction, it is verified the possibility to use EP for the optimization of a wide range of networked structures. Additionally, this work proposes a modified version of the EP algorithm. The original formulation is remarkable since it was rigorously derived. Nevertheless, its generality is affected by the fact that the directional synaptic weights between any two neurons must be equal. To overcome this limitation, Scellier et al. extended the algorithm to non symmetric connections [5]. This second algorithm is derived by means of a heuristic approach. Hence, other solutions might be equally valid and should be taken into consideration. The proposed variant to EP is therefore to be interpreted as a possible alternative to the Scellier extension and its applicability is verified on the reconstruction problem.
For what concerns classification, the EP algorithm is applied to increasingly complex classification tasks. Finally, it is discussed a Feature Engineering approach inspired by the learning mechanisms of the Additive model. This allows to considerably reduce the number of synaptic connections needed while guaranteeing satisfactory performances. In all the problems considered

the non-linearity is introduced using two functions: the hard sigmoid or the hard hyperbolic tangent. Both of them are proved to be effective in the solution of all the investigated tasks while being compactly implementable in hardware.

The second part of this work aims at developing ngSpice models [6] to simulate the analogue circuit associated to the RNNs discussed in the algorithmic part. The starting point is a behavioral definition of the primitives to be connected in the network. Then, the description level is progressively lowered toward a full hardware description. In its final form, the proposed neuron is made of two operational amplifiers, three resistors, one capacitor and two Schottky diodes. This solution is particularly promising since this neuron is designed to limit the retention problem and the non-linearity of ReRAMs. In the last part of this work one additional non-ideality of the ReRAMs is considered: the cycle-to-cycle variability [7]. This feature prevents from achieving the performances that can obtained on digital platforms and forces to work with quantized states of conductance. When extending EP to quantized networks some precautions need to be taken and the classification task is used for discussing them.

Acknowledgements

A big thank you goes to my supervisors. I would like to thank Fernando Corinto for his supervision and for welcoming me into his group, whose prominent role in the memristors community has enabled me to learn a great deal from their dialogue with other groups of international importance. Furthermore, I would like to thank Carlo Ricciardi for his willingness to solve both scientific doubts and practical problems whenever I tackled him at the end of the lessons.

I am immensely grateful to my scientific supervisors. Together we have gotten our hands dirty on the topics discussed in the following pages. I would like to thank Gianluca Zoppo for the friendly way in which he made his mathematical knowledge available to me in the long discussions we had. Similarly, I would like to thank Francesco Marrone for the help he has given me with his rigour and with his extensive knowledge of electronics (and more).

Thanks also go to Jacopo Secco for his sparkling personality and entrepreneurship. The magnetism of these qualities allowed me to start a project that I greatly enjoyed.

I would like to thank my family for their constant support. First of all my parents for the house they built. This is for me both a place and a pulsating metaphor for all the values I share with them. In both meanings, this house was more than ever the basis of much of my inspiration during this project. Special thanks to Marta, my little sister, who teaches me so much with her wisdom and with whom I have spent days and nights working and studying to achieve our goals. Thanks also to Cinzia for being a living example of how happy you can be by following your heart in your daily and life choices.

I would also like to thank all the people who helped me build my future. I am grateful to Lucia, Matteo and Jack for their teaching and support in understanding what my next step should be. I am grateful to Professors Braunstein, Gonnelli and Marsili for helping me to get the PhD position of my dreams with their references. Finally, I am extremely grateful to the people who helped me shape my path, including Elena Ferro, Eleonora Testa, Ettore Merigioli and Professor Andriulli.

Finally, I would like to thank all those who know that they are responsible for my mental health. Thank you to everyone who has shared with me moments of light-heartedness, in the joyful dance that is this life.

Table of Contents

List of Tables IX

List of Figures X

Acronyms XV

1 Neuromorphic computing with resistive switching materials 1
1.1 The digital revolution . . . 1
1.1.1 The Von Neumann architecture . . . 1
1.1.2 Artificial Intelligence . . . 2
1.1.3 Need for efficient solutions . . . 4
1.1.4 Neuromorphic engineering . . . 5
1.2 Biological Neural Networks . . . 6
1.2.1 The Neuron . . . 6
1.3 Crossbars-based architectures . . . 10
1.3.1 Resistive Switching in ReRAM . . . 10
1.3.2 Crossbars . . . 10
1.3.3 Memristor . . . 12

I Learning Algorithms 14

2 Neuronal Dynamics 15
2.1 Grossberg theory of Recurrent Neural Networks . . . 15
2.1.1 Theory of embedded fields . . . 15
2.1.2 Additive Model . . . 18
2.1.3 Learning lists . . . 20
2.2 Dynamical systems . . . 21
2.2.1 Equilibrium Points . . . 22
2.2.2 The Cohen-Grossberg theorem . . . 23
2.2.3 Attractors and learning . . . 25

3 Learning: Equilibrium Propagation 28
3.1 Energy-based models . . . 28
3.1.1 Energy-based inference and learning . . . 29
3.2 Equilibrium Propagation . . . 30
3.2.1 The architecture . . . 30

3.2.2 The algorithm . . . 31
3.2.3 Hardware implementation . . . 36
3.3 Asymmetric Versions of Equilibrium Propagation . . . 38
3.3.1 Scellier Extension . . . 38
3.3.2 Alternative Extension . . . 39
3.4 Conclusions . . . 41

4 Reconstruction 42
4.1 Reconstruction . . . 42
4.2 Grossberg-Hopfield network . . . 43
4.2.1 One Pattern . . . 44
4.2.2 Solution with the three versions of the algorithm . . . 46
4.3 Multilayer with interconnections . . . 50
4.4 Recurrent Denoising AutoEncoder . . . 51
4.5 Conclusions . . . 52

5 Classification 55
5.1 Basics of classification with recurrent neural networks . . . 55
5.2 Logic functions . . . 56
5.2.1 Linearly separable classification problem . . . 56
5.2.2 Non-linearly separable classification problem . . . 58
5.3 Digit classification . . . 61
5.3.1 Multilayer network . . . 62
5.3.2 Feature Engineering . . . 65
5.4 Conclusions . . . 70

II Hardware **71**

6 Electronic Components 72
6.1 Diodes . . . 72
6.2 Operational Amplifier . . . 74
6.3 Resistive Switching Random Access Memory . . . 77
6.3.1 Physical mechanisms and programming . . . 77
6.3.2 The Stanford/ASU model . . . 80

7 Hardware Implementation 82
7.1 Standard Electrical Circuit Interpretation . . . 82
7.2 Behavioral circuit definition . . . 84
7.2.1 Integrator and activation function . . . 84
7.2.2 Building the network . . . 85
7.2.3 Verifying the circuit functioning . . . 88
7.3 Operational Amplifier and Hard Sigmoid . . . 89
7.3.1 Operational Amplifier Based Primitives . . . 90
7.3.2 Some dimensional considerations . . . 92
7.3.3 Operational Amplifier based network and non idealities . . . 94
7.3.4 The Retention Problem . . . 97

7.4 Conclusions 99

8 Diodes-based networks 101
8.1 Diode and Input of the Bengio-Scellier Model 101
8.1.1 State Neuron 103
8.1.2 Rate as the state variable in hard functions 104
8.1.3 Inference 105
8.2 Exploiting the low forward voltage of the Schottky diode 107
8.3 Conclusions 108

9 Stochasticity and Quantized networks 110
9.0.1 Random velocity field 110
9.0.2 Quantization of the weights 112
9.0.3 Motivation 113
9.0.4 Modified Equilibrium Propagation 115
9.1 MNIST classification 116
9.1.1 Multilayer network 117
9.2 Conclusions 118

10 Conclusions 120
10.1 Learning Algorithms 120
10.1.1 Update rules 120
10.1.2 Reconstruction 121
10.1.3 Classification 121
10.2 Hardware 122
10.2.1 Analogue Networks 122
10.2.2 Stochasticity and quantized networks 123
10.3 Future works 123

Bibliography 127

List of Tables

4.1 Parameters and performances of the three algorithms. EP is the symmetric version, a-EP the asymmetric version proposed by Scellier and a-EP* the modified asymmetric version we defined. All the activation functions are the hard version of the one indiated. 48

4.2 Distances in norm-2 of five matrices: the entry (i, j) of the table is the difference in norm 2 of the i-th matrix with the j-th matrix. The first matrix is a uniformly distributed random matrix while the others are the parameters optimized using equilibrium propagation. The second matrix is obtained with symmetric EP and using the hard sigmoid activation function while the third uses the hard hyperbolic tangent. The fourth matrix is obtained with the Scellier extension while the last one is the extension proposed in this work. 49

5.1 truth table of the AND function. 56

5.2 truth table of the XOR function. 59

5.3 truth table of the XOR function. 60

5.4 Figure of merits of the two networks optimized for the MNIST classification task. 64

5.5 Increase of the performances when adding the PCA-based engineered features. The first column is the number M of components considered, the second column is the dimensionality of the newly defined input vector z, i.e. $N_f + M^2$. The remaining columns are the remaining figures of merits. 70

7.1 Main features of the machine used for the simulations. 89

7.2 Performances of the hardware implementation of the hard sigmoid when used to do inference on the XOR problem. 97

8.1 Evaluation of the performances of the diodes based network for doing inference with the XOR problem. 106

8.2 Comparison numerical training of the MNIST classifier: Linear Additive model and non-Linear version inspired by the Stanford model. 108

9.1 Accuracy of the trained MNIST classificator when reducing the number of quantizaed states. 117

9.2 Accuracy of the trained MNIST classificator when increasing the learning rate at fixed number $N_{levels} = 11$ of quantized states. 118

9.3 Accuracy of the trained MNIST classificator when increasing the number of hidden nodes at fixed number $N_{levels} = 11$ of quantized states. 118

List of Figures

1.1 Schematic representation of the neurons functioning. Each neuron is described as an information processing unit made of three components: the dendrites, the soma and the axon. Each neuron receives the information via the dendrites and the soma integrates them. This results in a controlled alteration of the membrane permeability to ions, that changes the membrane potential. If the membrane potential overcomes a certain threshold, an action potential is sent through the axon to the surrounding neurons. 7
1.2 Resistive crossbar realized with normal resistors. 11
1.3 From left to right a resistor is substituted by a 1T1R memristive weight, that is then splitted into two memristive devices to allow the encoding of negative weights. 12

2.1 Abstract learning machine defined by Grossberg in the theory of embedded fields. To each behavioral unit (letter) it is associated a node in the machine. 16
2.2 Schematic representation of the travelling signal theorized by Grossberg for encoding the transfer of information. 17
2.3 Graphical representation of the conversion of spikes into rates. The spikes of a population of neurons are averaged in time and in neurons, converting the spiking information into a rate information. 19
2.4 Grossberg network: learning the list $A \rightarrow B$. 20
2.5 Schematic evolution of a dynamical system in the states space. 21
2.6 *Left:* A uniformly stable but not convergent system;*Right:* A not uniformly stable but convergent system; 23
2.7 Schematic representation of the N_s one dimensional dynamics experienced by the agents of the model. 26

3.1 Pictorial representation of the Energy function for classification purposes. . . . 29
3.2 Schematic representation of a recurrent neural network. Three classes of nodes can be recognized: the input nodes (x), the hidden nodes (h) and the output nodes (y). All of the units are connected by directional synaptic weights. The nodes labelled as d are the so called target nodes and are used to find the best combination for the synaptic weights. 31
3.3 Free Phase: The system is randomly initialized in the rates space and evolves freely minimizing the energy. The final point is the free phase equilibrium. . . . 33
3.4 Nudged Phase: The system starts from the free phase equilibrium and evolves in a modified energy, tilted toward the target region. 33

4.1 Schematic representation of the reconstruction problem. The correct image is corrupted by noise and is fed to a recurrent neural network. If the selected machine has learned the restoring operation it reconstructs in output the original version of the image. 43
4.2 Grossber-Hopfield architecture. The network is fully connected and each node acts both as an input and as an output neuron. 43
4.3 *Top-left*: Pattern and damaged version; *Top-right*: Evolution of the accuracy and of the objective function at the different epochs of the training; *Bottom-Left*: comparison of the weight matrix learned by EP with the Hebbian rule; *Bottom-Right*: comparison of the bias vector learned by EP with the target pattern. The network is defined with the hard-hyperbolic tangent as the activation function. . 45
4.4 *Top-left*: Pattern and damaged version; *Top-right*: Evolution of the accuracy and of the objective function at the different epochs of the training; *Bottom-Left*: comparison of the weight matrix learned by EP with the Hebbian rule; *Bottom-Right*: comparison of the bias vector learned by EP with the target pattern. The network is defined with the hard-sigmoid as the activation function. 46
4.5 Weight matrix of the digits in the data-set when training the network with the EP algorithm on the individual patterns. 47
4.6 Symmetric EP: Accuracy and Cost as a function of the training epochs. 48
4.7 Comparison of the weight matrices obtained with the three algorithms: *Top Left*: symmetric EP with the hard sigmoid; *Top Right*: symmetric EP with the hard hyperbolic tangent; *Bottom Left*: Scellier version of the a-symmetric EP; *Bottom Right*: Proposed a-symmetric EP. 49
4.8 Generalization of the Grossberg-Hopfield network to the Bengio-Scellier one. . 51
4.9 Bengio-Scellier network used as a denoising autoencoder. 51
4.10 Reconstruction using the input as nodes: *Top Left*: `MatLAB` table summarizing the architecture defined; *Top Right*: Weight matrix learned by EP; *Bottom*: history of the performances during training. 52
4.11 Schematic representation of the main blocks constituting a DAE. The input is encoded in a latent space and then decoded to the output space. 53
4.12 Reconstruction using the DAE architecture: *Top Left*: `MatLAB` table summarizing the architecture defined; *Top Right*: Weight matrix learned by EP; *Bottom*: history of the performances during training. 53

5.1 Graphical representation of the AND classification problem. 56
5.2 Structure of the AND architecture. 56
5.3 Data-set used for the AND problem. 57
5.4 AND history 57
5.5 Caption 58
5.6 Graphical representation of the XOR classification problem. 59
5.7 Minimally complex architecture for learning the XOR function 60
5.8 Performances of the RNN defined in the solution of the XOR classification problem. 61
5.9 When the inputs are both belonging to the false class the system converges to $y = -1$ for every starting point in the state space. 61
5.10 When the inputs are belonging to opposite classes the system converges to $y = 1$ for every starting point in the state space. 61

5.11 On the top there is the original 28×28 MNIST image while on the bottom there is the equivalent 14×14 version. 62
5.12 Scheme of the MNIST classification problem via a recurrent neural network . . 62
5.13 Accurcy and Cost of the network while training it on the MNIST data-set. . . . 63
5.14 Confusion matrix of the MNIST classification using the hard sigmoid. 63
5.15 Accuracy as a function of the mini batch size 65
5.16 2D example of PCA. The data-points are in light blue, the mean data-point is in red and the principal components are multiplied by a scaling factor proportional to the associated eigen-value. Taken with permission from [68]. 67
5.17 Caption 68

6.1 Symbol of the p-n diode and I-V characteristic of the `ngSpice` default model. . 73
6.2 Symbol of the Shottky diode and I-V characteristic of the `ngSpice` DFLS220L model. 73
6.3 Symbol of the Operational Amplifier. 74
6.4 Internal functioning of an operational amplifier: basic model as a voltage controlled source. 75
6.5 Internal functioning of an operational amplifier: parasitic currents and capacitors. 75
6.6 Open loop transcharacteristic for various gain values of the chosen operational amplifier model. 76
6.7 Schematic representation of the DC cycle in a valence change memory device. . 78
6.8 Programming the memristor with the stop voltage. 79
6.9 Programming the memristor with the compliance current. 79

7.1 Scheme of the Standard Electrical Circuit Interpretation of a recurrent neural network. The signals flowing through the synaptic connections together with a bias term are collected at the input of the neuron. Here, the Soma sums the synaptic contribution to the current effects occurring at the level of the leaky membrane, represented as an RC circuit. Finally, the rate model is completed by finding the rate equivalent of the state variable, corresponding to the membrane potential averaged over time and neurons. 84
7.2 Behavioral description of the neuron circuit. 85
7.3 Behavioral description of the activation function. 85
7.4 DC sweep simulation to verify the correct functioning of the activation function block. The input voltage has been varied in the interval $v_{in} = [-0.5v_o, 1.5v_o]$. . 86
7.5 Transient simulation of the neuron block connected in series to the activation function block. The simulation has been performed on the interval $t \in [0, 3\tau_o]$. 86
7.6 Behavioral description of a state node of the circuit. 86
7.7 Hardware implementation of the input in the Grossberg-Hopfield network. . . . 87
7.8 Hardware implementation of the input in the Bengio-Scellier network. 87
7.9 MNIST with the hard-sigmoid: Trajectories of the state neurons when performing inference using the behavioral description. 89
7.10 MNIST with the hard-sigmoid: Confusion Matrix of the 1000 data considered when performing inference using the behavioral description. 89
7.11 Reconstruction with the hard-sigmoid: Convergence of the trajectories for a given input image. 90
7.12 Analogue implementation of the neuron block. 90

7.13 Voltage follower Configuration 91
7.14 Activation function block. 91
7.15 Operational amplifier based implementation of the hard sigmoid. 92
7.16 Operational amplifier model including the offset terms. 93
7.17 Hard sigmoid based neuron implementation using operational amplifiers. . . . 95
7.18 Verification of the proper functioning of the neuron primitives defined using the operational amplifiers. 95
7.19 Negative output of the state neuron: $R_o = 0\Omega$ and varying K. 96
7.20 Negative output of the state neuron: $R_o = 100\Omega$ and varying K. 96
7.21 Negative output of the state neuron: $R_o = 200\Omega$ and varying K. 96
7.22 Positive output of the state neuron: $R_o = 0\Omega$ and varying K. 96
7.23 Positive output of the state neuron: $R_o = 100\Omega$ and varying K. 96
7.24 Positive output of the state neuron: $R_o = 200\Omega$ and varying K. 96
7.25 On the left the usual implementation of the synaptic weights, based on the assumption that the resistance value can be tuned at will. On the right the implementation of a multistate conductance block as the parallel of binary weights. 98
7.26 Schematic representation of an hardware accelerator for RNN systems using binary weights. 99

8.1 Diode: circuit symbol and I-V characteristic. 101
8.2 Kendall activation function on a simple network. 101
8.3 Graphical solution of the KCL for the Kendall activation function. 102
8.4 Diodes base implementation of the hyperbolic tangent. 102
8.5 Testing the functioning of the neuron block with the Kendall activation function at the output. 103
8.6 Testing the functioning of the neuron block with the Kendall activation function in the feedback loop. 103
8.7 Non-linear Leaky integrator: Neuron block built up embedding the Kendall-like activation function in the feedback loop of the operational amplifier. 104
8.8 MNIST: Confusion Matrix of the inference performed with the diode-based network. 106
8.9 MNIST: Normalized Confusion Matrix of the inference performed with the diode-based network. 106
8.10 Non-linear Leaky integrator: Neuron block built up embedding the Kendall-like activation function in the feedback loop of the operational amplifier with the use of Shottky diodes. 108

9.1 MNIST classificator: Trajectories of the neurons in the state space given a random initialization condition. 112
9.2 Random fields in the XOR classificator when increasing the noise level. The red curve corresponds to $\nu = 0$, the blue one to $\nu = 0.1$ and the black one to $\nu = 0.3$. 113
9.3 Quantization procedure: a continuous value is mapped into a quantized state. . 114
9.4 Quantization procedure: a random fluctuation is added to the quantization. . . . 114
9.5 Performances of the quantized version of the AND function. 115
9.6 Distribution of the weights in the trained hard sigmoid based MNIST network. . 117
9.7 Accuracy as a function of the quantization interval. 117

Acronyms

ML Machine Learning

ENIAC Electronic Numerical Integrator and Computer
EDVAC Electronic Discrete Variable Automatic Computer

TRADIC TRAnsistor DIgital Computer or TRansistorized Airborne DIgital Computer

LIM Logic In Memory

RNN Recurrent Neural Network

MSE Mean Squared Error

EP Equilibrium Propagation

CM Confusion Matrix

PCA Principal Component Analysis

CZM Corinto Zoppo and Marrone

BS Bengio and Scellier

KVL Kirchhoff's Voltage Law
KCL Kirchhoff's Current Law

LTM Long Term Memory

STM Short Term Memory

CAM Content-Addressable Memory

bNN biological Neural Networks

AI Artificial Intelligence
ANN Artificial Neural Networks

MAC Multiply and ACcumulate

ReRAM Resistive Random Access Memory

TE Top Electrode

BE Bottom Electrode

VCM Valence Change Memories

CNF Conductive Nano-Filament

HRS High Resistance State

LRS Low Resistance State

DPC Depression Potentiation Cycle
DAE Denoising AutoEncoder

AM Additive Model

Chapter 1

Neuromorphic computing with resistive switching materials

1.1 The digital revolution

In the course of the last century an unprecedented technological revolution has taken place, leading to astonishing advancements in the quality of life in industrialised countries. The fundamental driving forces of this revolution have been the foundation of information theory and the concurrent development of more and more convenient electronic hardware platforms and technology capable of processing data which nowadays permeate each aspect of our lives. Basically, large part in this revolution is to be attributed to our capability to extract data from the surrounding world and to perform actions based on the derived information. Hence, this revolution is ultimately based on our capability to build machines capable of performing increasingly complex computational tasks.

1.1.1 The Von Neumann architecture

It all started in 1936, when Alan Turing developed the first theoretical model of computation [8]. He proposed the *Turing machine*, an abstract system capable of solving problems following rules stored in a table. He theorised about the existence of a machine whose operating principle would be based on symbols written on a potentially infinite strip. His work also led to the concept of *Turing completeness*, a property attributable to a data-manipulation engine capable of simulating any Turing machine. In simplistic terms, any algorithm can be expressed as a Turing machine and the vast majority of modern programming languages are Turing complete. Therefore, a computer equipped with an operating system supporting those languages can be transitively considered to be Turing complete. The first practical realizations of computing systems were purpose oriented, like the decryption device *Bombe*. However, soon after World-War II the research aimed at creating general purpose computers, leading first to the Electronic Numerical Integrator and Computer (ENIAC) and right after to the Electronic Discrete Variable Automatic Computer (EDVAC) [9]. Thanks to their capability of solving numerical problems through programming they were both classified as Turing complete. The latter system was the first realization of the so called Von Neumann architecture, that is at the basis of almost any computing done to our days. It is based on the idea that every computational problem can be divided into three steps: first some data is to be pulled from a memory and sent to a processing unit, then the logic performs

the computation, and finally the results are sent back to the memory. While the first realization of such architecture was based on valve tubes technology, the work done at the Bell Labs, led by Schockley, allowed to create the first transistor-based computer in 1954: the TRAnsistor DIgital Computer or TRansistorized Airborne DIgital Computer (TRADIC)[10]. With this invention the Digital era officially started.

Moore's law and Dennard's law

What said so far allows to understand why the main structure of the Von Neumann architecture has remained central to our days. First of all, its Turing completeness makes it extremely versatile as a general purpose engine. The possibility of performing many tasks with the same machine is one of the biggest breakthrough the mankind has been capable of, and the power of this computing paradigm can be understood just thinking at what it is possible to do nowadays with a smartphone in terms of various forms of information transfer, including domotics. In addition to this, its implementation in transistor technology made it dimensionally scalable. To understand this statement, we can retrospectively follow the well known Moore's law. In 1965 Gordon Moore, co-founder of Intel, foresaw that the miniaturization of integrated circuits would have allowed to fabricate electronics with increasingly complex functionalities and at lower cost [11]. In particular, his analysis was based on the manufacturing cost per component, which fixes the optimal number of devices to be integrated in a single chip at a given technology. He noticed that this number was increasing through years at an exponential rate, and correctly assumed that this rate would have been qualitatively maintained. Therefore, as the semiconductors industry was assiduously committed to follow the Moore's trend, the computing power of the Von Neumann architecture naturally benefited in terms of performances and capabilities.
In the evolution of the transistor technology power consumption has always had a key role. Increasing the number of devices was necessary, but this could not induce an increase in power dissipation. In his paper of 1965 [11], Moore stated that *"shrinking dimensions on an integrated structure makes it possible to operate the structure at higher speed for the same power per unit area"*. This statement was formalized by a research led by Robert Dennard, who found the prescription for scaling the characteristic lengths of a MOSFET transistor, while maintaining constant the dissipated power per unit area [12]. More precisely, he estimated an expected halve in power dissipation per transistor from one generation to the next. If one combines this information with the doubling of transistors per unit area predicted by Moore, it is possible to understand that a constant power dissipation per unit area was maintained from one generation to the next, while chips became capable of more and more complex tasks.

1.1.2 Artificial Intelligence

While chips technology was evolving, computer science theory kept attracting people from various domains of research and to branch in variegated manifestations. Among the others, also Artificial Intelligence (AI) started to take form. As a matter of fact, AI is a field of research whose official date of birth is just two years after the creation of TARDIC, in 1956. Some of the most prominent minds of the last century met during the Dartmouth Summer workshop and spent almost two months discussing on how to mathematically describe every aspect of learning, so as to create a machine capable of doing it [13]. While research has been continuous on the topic ever since, the interest of people on the subject has fluctuated over the years.

Early years

The first twenty years are commonly considered the golden age of artificial intelligence. Many of the algorithms used today date back to the findings of the 50s and 60s. The problem, at the time, was that the computing power was not ready to sustain the creativity of researchers. This is to say that in the early days of computing the state of technology did not allow neither to store, nor to process any significant amount of data. Only toy models could be solved, contrarily to the expectation of the funding agents, who had trusted the exceedingly optimistic predictions of some researchers. In any case, the attempt to design artificial systems capable of human perception and learning continued through the years, in a steady collection of results with much academic interest despite the small echo in industry. A proper collection of all of the breakthroughs of this period is out of the scope of this work. The interested reader can refer to the extensive literature on the history of AI [14, 15]. It is just worth mentioning that in the 70s plenty of work was done at the Artificial Intelligence Lab at MIT on optimized hardware for AI application, that led to the `LISP` programming language [16]. This was the language of the Lisp machine, the first single user workstation, optimized for AI tasks and commercialized in 1979. Unfortunately for the developers, the commercialization occurred concurrently with the so called *AI winter*, during which, once again, AI found itself unfairly downgraded. To worsen the consequences of the diffused mistrust for AI, the technology was ready for starting the microcomputer revolution, led by Apple and IBM. In few years desktop computers could outperform Lisp machines even for Lisp programming tasks, killing at birth an extremely fascinating AI-oriented hardware solution. In any case Moore's law kept holding and computing power kept increasing. This allowed in the years from the 90s to the first decade of the new century to have sufficient computing power to use old algorithms for performing tasks at a significant rate. Machines started to play chess better than humans and as digital systems permeated every aspect of our life, AI algorithms were often silently behind the advancements our society experienced.

The Big Data Era

In the last ten years AI has re-emerged as a leading topic and, according to the predictions [17], it is here to stay. The main reasons for this renaissance, however, are not uniquely attributable to the Moore's law. Moore's law has certainly had large impact on the creation of more efficient hardware, but what really made the difference was the advent of the Internet and of Cloud computing, that pushed the mankind in the Big Data era. The Internet started to be present in the everyday life in the 90s and progressively diffused through the social fabric, permeating any aspect of our world. Soon enough, any kind of data was present on the Internet, carefully stored for later usage. In this way, billions of people contributed to the creation of huge data-sets.
A subset of AI is Deep Learning, studying various manifestations of the so called Artificial Neural Networks (ANN). Generally speaking, Deep Learning models are high dimensional parametric models that can act as functions approximators. These models benefit from large data-sets since their parameters can be optimized by presenting them many examples of something we want the model to learn. Due to the parametric nature of these models, at first it was believed that neural networks could not be made arbitrarily large. This expectation was based on the fact that complex parametric models are known to be subjected to overfitting. This is the machine equivalent of learning by heart, which is highly undesired when one wants to have a machine capable to correctly perform a task on data never seen during the learning phase. Nonetheless, as soon as it was possible to perform experiments with such networks, it was noted that, in some

cases, no over-fitting occurs in Deep Neural Networks [18]. In this way it was possible to make such models arbitrarily complex, pushing the limits of computing performances at will. This behavior is still under investigation, especially from physicists who are trying to understand it using tools from Statistical Mechanics [19]. However, even if unexplained, it led to brute force empirical Deep Learning approaches that have dominated the scene over the last years. This was made particularly easy by the seemingly unlimited computation resources available to the user thanks to the introduction of Cloud computing. In fact, this enabled the massive parallelization of submitted tasks, making it possible to train models with a huge number of parameters. Just to mention some famous architectures, the GPT-n family is a clear example for this trend. GPT-n constitute a family of AI based general purpose learners for text generation tasks. Up to now three evolutions of it have been developed and while GPT-2 had 1.5billions parameters, GPT-3 [20] has jumped to 175 billions.

1.1.3 Need for efficient solutions

Despite the uncountable benefits induced by the aforementioned Digital Era, we are now facing some fundamental issues. On the one hand the slowing down in the Moore's law and Dennard's scaling are forcing the community to start looking for alternatives to CMOS based Von Neumann architectures. On the other hand, modern computing trends in AI are no longer sustainable from an energetic point of view.

Post Moore era

The Moore's law and the Dennard's scaling continued to foster research through years. Nonetheless, in the last two decades the scientific world has become aware that this trend cannot proceed much longer [21]. The first signs came from the slow down in Dennard's scaling. As the dimensions are shrinked to nanometric scale, leakage currents gain progressively higher importance, consequently increasing the power dissipation per integrated circuit area. Since an increase in performances via technological advancements has become an increasingly hard challenge, the researchers started questioning the very structure of the architecture on which our computing systems are built on: the Von Neumann architecture. One of the major limitations in this computing system is also one of its greatest strengths, making it difficult to get rid of it. In fact, the general applicability of the Von Neumann architecture is largely guaranteed by the separation of memory from logic. Individually, memory and logic have been subjected to miniaturization thanks to the aforementioned trends. Nonetheless, while the logic becomes increasingly faster thanks to the growth in complexity that can be integrated in a single chip, the amount of information that can be transmitted from memory to logic has reached its maximum. This is to say that even if we can increase the speed of the actual computation performed by the central processing unit, the total speed of the device will always be the same, since it is dictated by the speed with which data is transmitted back and forward, in the unit time, from memory to logic. This is to say that, even if we can increase the speed of the actual computation performed by the central processing unit, the total speed of the processor will always be the same, since it is dictated by the rate at which data is transferred from memory to logic and back. Therefore, if one wants to increase performances, this is the first limiting factor to be dealt with and is generally referred to as the Von Neumann bottleneck. The most promising approach to solve this issue is to bring computation and memory together, in an approach named Logic In Memory (LIM). This has taken many different forms, that interestingly require one to reduce the generality of the model

and go back to purpose-oriented solutions for maximizing the performances. This thesis project can be identified as belonging to this class of approaches, as it will be better described later in the chapter.

The unsustainable trend in Deep Learning

Despite the fact that the stagnation in hardware performances could be compensated by performing it on computing clusters in the cloud, this would be blindly unsustainable in term of energy consumption. In an article published in Forbes [22] it was argued that training GPT-3, in terms of carbon footprint, is the Deep Learning equivalent of five cars through their whole lifetime. Not to mention the hazardous environmental impact of Von Neumann based data centers [23]. Given these premises and trends, the whole community started to explore alternatives to the way computing is currently performed. In a review paper on Non Von Neumann computing [24], the authors state that

The future of computing is at crossroads. The technological advances that have sustained the exponential growth of computing performance over the last several decades are slowing and the roadmap for future advances is uncertain.

Among the variegated approaches that are emerging on the scientific landscape, the field of Neuromorphic Engineering is gaining more and more credits as the future of AI electronic processors.

1.1.4 Neuromorphic engineering

Neuromorphic engineering is a branch of research sprouted from the mind of the "father of computation", Alan Turing. In his work on Intelligent machineries, he had envisioned the power of networked structures for computing tasks [25]. Nevertheless, since the issues mentioned above had not yet arisen, scientists and engineers in academia and in the corporate world did not push so hard towards a bio-inspired approach. The goals of this field have been collected in a on-the-run manifesto of this research area, published in 2015 [26]. Basically, Neuromorphic Engineering is a research branch of electronics engineering which tries to implement physical systems that, taking inspiration from the working principles of mammalian brains, try to solve AI related problems in a much more efficient way than conventional Von-Neumann digital systems are capable of. As already stressed, when talking about performances we mean both computing performances and energy consumption performances. The mammalian brain is capable of mesmerizing real time processing while consuming a very limited amount of resources. Therefore, the goal is to create brain-inspired hardware for reaching optimal performances in AI-related tasks. Brain inspired computing can naturally combine the eagerness of fast computing and the old desire of building intelligent machines. In fact, it is clear that advancements in high performance hardware for brain-inspired computation can only come as a result of a deeper understanding of the brain functioning and of its effective reproduction in some form of hardware. Neuromorphic engineering is therefore a broad domain of research that is being powered by the increasing collaboration of people coming from different research areas. Mathematicians, electrical engineers, physicists, neuroscientists, computer scientists and material scientists share on the same level the knowledge of their fields, in the attempt to build the next generation chips. Also the big names in Electronics are taking part to the change, including Intel and IBM. In fact, both of them have already proposed their own neuromorphic chips, Loihi [27] and TrueNorth

[28] respectively. Both of them are optimized for dealing with artificial neural networks and are based on the mature CMOS-technology. The two electronics firms managed to simulate the brain functioning by defining complex CMOS-based architectures. While these solutions already provide an example of major achievements in Neuromorphic Engineering, their dependency on transistors does not make them the definitive solution to our computing needs [29]. They allow to simulate neuronal behavior but the complexity of their architectures make them extremely power hungry, while remaining not promising in terms of future scalability, due to their dependency on an apparently saturated technology. Many questions are still open on how to optimally mimic our biological computing tool. One of the most promising directions, according to the International Roadmap for Devices and Systems [30], should come from the next generation of Memory devices.

1.2 Biological Neural Networks

As previously mentioned, the neuromorphic community has progressively gained more interest in emerging memory technologies. This is mainly due to their potential as electronics-based emulators of brain constituents, like synapses and neurons. This section aims at introducing the artificial neural networks starting from biological knowledge. The reference review paper for the discussion is the one of Tang et al. [29]. Among the different memory devices, this work focuses on the Resistive Random Access Memory technology. In fact, recent experimental results [31, 32] were extremely promising in terms of three figures of merit of memory devices: scalability, retention and endurance. In order to bridge biological and artificial neural networks, the brain mechanisms to be mimicked are introduced in the following section. This allows to highlight the properties that the electrical components must have to perform brain inspired computation.

1.2.1 The Neuron

The mammalian brain is a networked structure made of neurons [33]. The neurons are the computing units of the brain and are connected through synapses. From a biological point of view a neuron is a cell made of three components: a soma, an axon and dendrites. As every other living cell, they are delimited by a lipid double layer membrane which separates them from the extracellular environment. This membrane contains ion pumps, channels that selectively let ions enter in the cell or expel them. Their presence allows to have a controlled voltage drop across the membrane. In normal condition, the internal part of the neuron is at an electrical potential around $70mV$ lower than the extracellular environment. In the leaky integrate and fire model, the information processing of the neuron can be distributed among its three constituent blocks. First the information is received through the input units of the neuron, the dentrites. These are connected to the axons of the surrounding neurons and, in this simple model, they are assumed to just receive the incoming signal and to send it to the soma. The soma is the body of the cell and its functioning can be assimilated to the one of an integrator. Basically, it receives the stimuli transmitted through the dendrites and the ion channels on the membrane are opened to generate an in-flux or an out-flux of ions, depending on the kind of information transmitted. In this way, the ions concentrations on the two sides of the membrane potential are altered, consequently modifying the average voltage. If the membrane potential becomes larger than a certain threshold an action potential is generated, that propagates in the form of a transitory fast signal along the axon, toward the surrounding neurons.

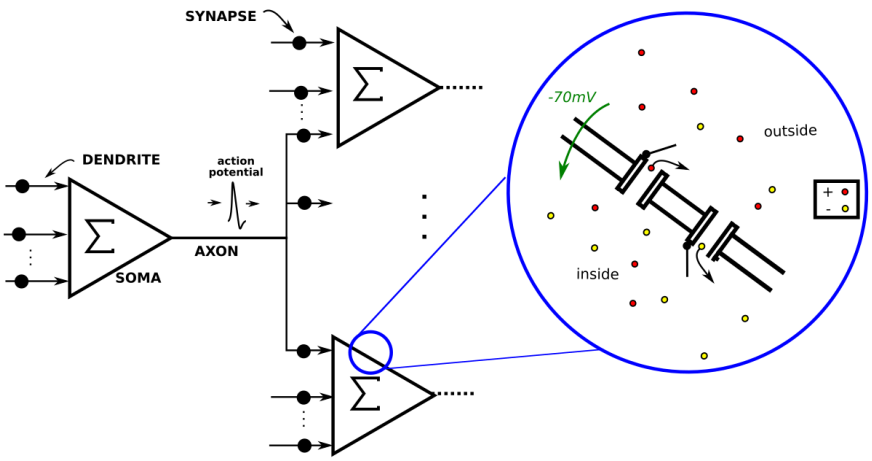


Figure 1.1: Schematic representation of the neurons functioning. Each neuron is described as an information processing unit made of three components: the dendrites, the soma and the axon. Each neuron receives the information via the dendrites and the soma integrates them. This results in a controlled alteration of the membrane permeability to ions, that changes the membrane potential. If the membrane potential overcomes a certain threshold, an action potential is sent through the axon to the surrounding neurons.

Leaky integrate and fire

From the early understanding of the electrical properties of neurons, people tried to define a compact model of the neuron activity. The membrane potential of the generic m neuron acts as a capacitor C_m, a component separating charge in such a way to build a dynamically varying potential. The potential evolves in time as soon as currents of ions flow through the membrane. These currents are assumed to be controlled in some way by the stimulus received at the dendrites. Finally, an input current is added as an external stimulus, that could be an experimental electrode or some sensory input. The Hodgkin-Huxley model for the m-th neuron thus becomes:

$$C_m \frac{dv_m}{dt} = \sum_k I_{mk}(t) + I_m(t) \tag{1.1}$$

A refinement of the model is provided by the so called leaky integrate and fire model:

$$C_m \frac{dv_m}{dt} = -\frac{v_m - e_m}{R_m} + \sum_k I_{mk}(t) + I_m(t) \tag{1.2}$$

where the resistive term is introduced to take into account possible leakages of the membrane. Additionally, a constant potential term e_m is added to take into account the equilibrium voltage difference dropping across the membrane.

Synapses and plasticity

To understand the nature of the ion currents, a deep comprehension of the way information arrives to the soma for integration is needed. The synapse is a structural link connecting the axon of a neuron, called pre-synaptic neuron, to the dendrites of the subsequent one, named post-synaptic neuron. Each time a spike traverses the axon of the pre-synaptic neuron, some chemicals

named neuro-transmitters are released at the level of the synapse, reaching the dendrites of the post-synaptic neuron. These substances locally modify the permeability of the membrane to ions, consequently inducing the appearance of a time dependent ion current, previously indicated as $I_{mk}(t)$. This stimulus can be either excitatory or inhibitory, respectively increasing or decreasing the voltage potential. Additionally, the synapses are known to vary in time, in a process called synaptic plasticity. This is to say that the capability of a signal arriving at a synapse will not influence the post-synaptic neuron in the same way at different moment in the life of the brain. Synaptic connections can be potentiated or de-potentiated in a process that is commonly believed to be at the basis of learning. In order to make this concept clear, it is then possible to model the ion currents in terms of the spike voltages that induced the release of the neurotransmitter at the level of the dendrite. This can be generally written as $I_{mk}(t) = G_{mk}(t)\zeta(t-\tau_{mk})$, where G_{mk} is a time varying conductance while $\zeta(t-\tau_{mk})$ is a signal indicative of the spiking activity in the pre-synaptic neuron, given the propagation delay τ_{mk}. Additionally, since a voltage potential is always a positive spike, the origin of the potentiation or de-potentiation must be encoded in the conductance. Since no intrinsically negative conductance can exist in nature, it is generally assumed that potentiation or de-potentiation can occur at different synaptic channels, leading to the so called adaptive integrate and fire model

$$C_m\frac{dv_m}{dt} = -\frac{v_m}{R_m} + \sum_k G^+_{mk}(t)\zeta^+(t-\tau_{mk}) - \sum_k G^-_{mk}(t)\zeta^-(t-\tau_{mk}) + I_m(t) + I^e_m \tag{1.3}$$

where the synaptic contribution has been splitted into the excitatory $(+)$ and inhibitory $(-)$ contributions and the rest potential has been introduced in the general term I^e_m. Furthermore, additional equations are required to describe the evolution of the synaptic weights, namely to model synaptic plasticity

$$\frac{d}{dt}G_{mk} = f(G_{mk}, v_m, I_m) \tag{1.4}$$

This corresponds to model the way the network is capable to adapt as a consequence of external information being introduced, and in the next chapters three closely related learning approaches are reported. The basis of them is Equilibrium Propagation, a learning algorithm ideated by Bengio and Scellier in 2017 [1]. This is a mathematically rigorous algorithm, that is however limited to networks in which the synaptic weights are symmetric. To overcome this potentially non-physical limitation, they proposed a follow up work to extend the algorithm to a-symmetric networks [5]. This corresponds to the second algorithm discussed. Finally, a third algorithmic solution is proposed, based on the logical connection of symmetric EP and a-symmetric EP. This is a new algorithm and might be biologically interpreted through the phenomenon of habituation.

Averaging in time and in neurons

The neuron model introduced so far is extremely simplistic but also general. The sum runs over all possible synaptic connections and the contribution at time t is governed by the presence of a spike originated in the soma of the pre-synaptic neuron at previous instants of time. Thanks to this non-locality in time, the model is extremely biologically plausible and it is generally called a *spiking model*. In the neuromorphic community, spiking models are having much success and a lot of research is oriented to create hardware in which the information transfer occurs through spikes [34].
An alternative point of view is to consider *rate models*. The basic idea is that the number of neurons in the brain is huge and there are experimentally proved neuronal structures in

mammalians in which neurons are arranged in columnar structures and having the same properties. In this context, let us consider a biologically layered structure in which a column of $N_{m'}$ identically functioning neurons are post-synaptic to a column of $N_{k'}$ identical pre-synaptic neurons. In this network it is possible to define the two populations as $P_{m'}$ and $P_{k'}$ and to define two coarse grained neurons m' and k', respectively associated to the pre-synaptic and to the post-synaptic groups of neurons. Additionally, for each pre-synaptic coarse grained unit it is possible to define a quantity called population activity.

$$\varphi_{k'}(t) \doteq \left\langle \sum_{k\in P_{k'}} \zeta(\tilde{t}-\tau_{km}) \right\rangle_{\Delta t, P_{m'}} \doteq \frac{1}{N_{m'}\Delta t}\int_{t-\Delta t}^{t} d\tilde{t} \sum_{m\in P_{m'}} \left(\sum_{k\in P_{k'}} \zeta(\tilde{t}-\tau_{km}) \right) \tag{1.5}$$

In words, given the pre-synaptic population $P_{k'}$, the population activity is defined as the average over a time interval Δt and over the post-synaptic population $P_{m'}$ of the spiking activity signals sent from $P_{k'}$ to $P_{m'}$ at an earlier time τ_{km}. The neurons labelled with m (k) and giving rise to the coarse grained neuron m' (k') are identical to each others. Hence, the synaptic weight connecting any two neurons of these populations must be the same, and it will be referred to as $G_{mk} = G_{m'k'}$. Similarly, also the capacitor and the resistor of each neuron must be shared by the individuals of each population, i.e. $C_k = C_{k'}$ and $R_k = R_{k'}$. Not much attention was given to the spiking signal on the communication line $k \to m$: $\zeta(t-\tau_{mk})$. However, it is clear that its definition should be taken to be the residual effect at time t of the individual spiking events occurred at earlier times on the communication line $k \to m$. These events are collected in the ensemble $S^{<t}_{k\to m}$ of spiking events and the spiking activity reads

$$\zeta(t-\tau_{mk}) = \sum_{s\in S^{<t}_{k\to m}} \kappa(t-\tau^s_{mk}). \tag{1.6}$$

This model assumes an exceeding homogeneity in large numbers of neurons. Nonetheless the idea of defining coarse grained versions of neurons by averaging out in time and in populations is a useful coding principle. In fact, this concept allows to substitute the biological spiking neurons with their coarse grained artificial counterpart, making the dynamic system of the neural network local in time. From a mathematical point of view this is obtained by applying the average defined in Eq.1.5 to both the sides of Eq.1.3. By further assuming that the evolution of the neurons is such that

$$\left\langle \frac{dv_m}{dt} \right\rangle_{\Delta t, P_{m'}} \simeq \frac{d}{dt}\langle v_m \rangle_{\Delta t, P_{m'}} = \frac{dv_{m'}}{dt} \tag{1.7}$$

one finally obtains

$$C_{m'}\frac{dv_{m'}}{dt} = -\frac{v_{m'}}{R_{m'}} + \sum_k G^+_{m'k'}(t)\varphi^+_{k'}(t) - \sum_k G^-_{m'k'}(t)\varphi^-_{k'}(t) + I_{m'}(t) + I^e_{m'} \tag{1.8}$$

where the current terms are the averaged version of the corresponding components appearing in Eq.1.3. This is almost equivalent to the Grossberg Additive Model for time continuous recurrent neural networks, that is discussed in more details in the next chapter.
At this point it has been discussed one of the simplest existing models of neural networks, present in the field for long time. The reason for this is that well established models can be safely used to test new learning algorithms and to discuss their hardware implementation.

1.3 Crossbars-based architectures

Once introduced the general functioning model for a neural network, it is possible to identify some circuital desiderata for implementing them. First of all, in Eq.1.8 is contained the most ubiquitous operation in the field of ANN: the multiply and accumulate operation (MAC). Expressions like $\sum_k G_{mk}(t)\varphi_k(t)$ are present in almost all the algorithms describing networked structures. This basically corresponds to a matrix vector multiplication and, if combined with the introduction of non-linearities, it is at the basis of the learning capability of these systems. In the current digital approach to neuromorphic hardware, accelerators are employed to parallelize large matrix vector multiplications. Nonetheless, the hardware that performs computation operates in a sequential way. Therefore, the first desiderata for new architectures to be developed is the capability of performing the MAC in parallel. Secondly, we pointed out that the conductances appearing in the integrate and fire models should be varying in time, according to a dynamics that corresponds to the learning process. In other terms the architecture should allow for a potentiation and de-potentiation of the synaptic connections between the different neurons. It turns out that both these requirements are met by crossbars of resistive switching devices, also referred to as memristive crossbars.

1.3.1 Resistive Switching in ReRAM

The key feature in biological synapses is their capability to change in time. As the previously described model showed that the strength of a synaptic connection can be associated to a conductance value, it would be desirable to have a compact component whose resistance can be tuned easily and in a controlled way. In the course of the last years many devices have been proven to show this feature and have been studied with the aim of implementing artificial synapses. Despite the ferment in the field, researchers are still struggling with the intrinsic stochasticity of the devices, and the difficulty to reach accuracies comparable with the one we can currently achieve with equally complex networks, simulated on digital hardware. One of the most promising emerging memory technologies is the Resistive-switching Random Access Memory (ReRAM)[35]. The main reason behind the scientific interest on it is that it appears to be more likely to achieve satisfactory incremental switching in the near future. Another interesting feature of these devices is that they are based on the well established metal oxide technology, involving materials such as HfO_x and TiO_x. From this technology the ReRAMs inherit the compatibility with CMOS circuitry, an essential requirement for a smooth transition to future hardware solutions. Additionally, ReRAM devices can be integrated very compactly and are extremely power efficient. In the following discussion, a Pt/HfO_2/Ti/TiN device [35] is used as a reference for modelling neural synapses.

1.3.2 Crossbars

One of the advantages of ultra-compact memory devices is the possibility to arrange them in crossbar structures. This fact allows a natural parallelization of the MAC operation. As previously mentioned, the ionic current induced by external stimuli to the m-th neuron of the network is:

$$I_m = \sum_k G_{mk} V_k \tag{1.9}$$

where the population activity has been substituted with a generic voltage value. In a more compact form, if there are N_u neurons in the system, it is possible to define the N_u dimensional vector of the ionic currents I_m, the N_u dimensional vector of the population activities and the $N_u \times N_u$ matrix of the conductances, expressing all of the synapses. This being said, the ionic currents are computable via the MAC operation

$$I = G \cdot V. \tag{1.10}$$

Figure 1.2 shows how it is possible to combine Ohm law with the Kirchhoff's Current Law (KCL) to compute a matrix vector multiplication in a naturally parallelized way. Of course one of

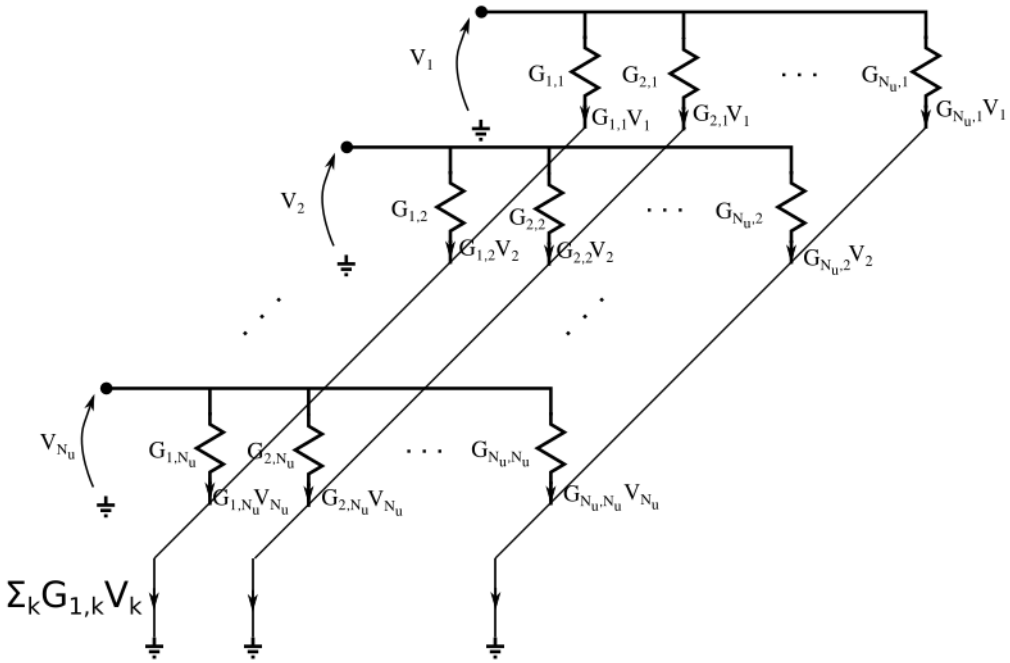


Figure 1.2: Resistive crossbar realized with normal resistors.

the nodes need to be grounded but, as later discussed, it is possible to achieve this by connecting it to the inverting input node of an operational amplifier, while connecting the non inverting input node to ground. In this way, ideally, the node is at virtual ground while the current can be used for defining the whole analogue brain. In Fig.1.2 normal resistors have been represented for the aim of clarity. However, they should be substituted with resistive switching devices. More precisely, the configuration typically adopted is the so called one transistor one resistor (1T-1R), that can be observed in Fig.1.3. First the resistor has been substituted with a resistive switching device in series with a transistor. The transistor allows to connect or disconnect the resistive device, depending on the configuration and on the need to re-program it. Moreover, to take into account the fact that no intrinsically negative resistance value can be achieved, the trick of inverting the voltage and using two resistors has been adopted. In this way both positive and negative weights can be achieved

$$I = G \cdot V \quad I = (G^+ - G^-) \cdot V \tag{1.11}$$

In any case, the interesting thing is that we pass from a sequential computation of the MAC operation, corresponding to a computational cost of $\mathcal{O}(N_u^2)$, to a constant computational time

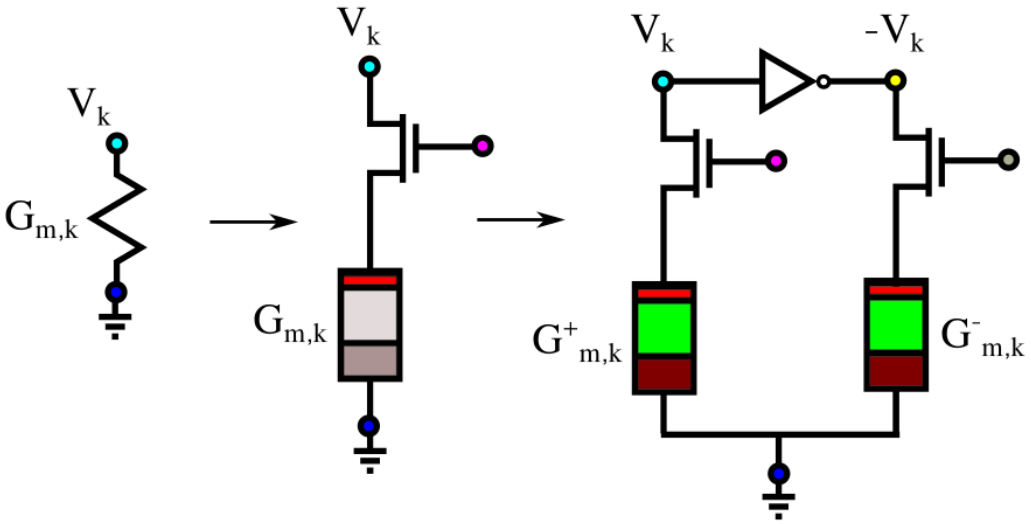


Figure 1.3: From left to right a resistor is substituted by a 1T1R memristive weight, that is then splitted into two memristive devices to allow the encoding of negative weights.

$\mathcal{O}(1)$.

1.3.3 Memristor

In 1971 Leon Chua theorized the existence of a fundamental circuital component, that should be added to the resistor, the inductor and the capacitor when putting into relation the fundamental electrical quantities. For an in depth theoretical characterization of the component the interested reader can refer to the work of Corinto et al. [36]. First of all, Chua expressed the constituent relationships of the fundamental ideal devices in terms of four quantities: the current i, the voltage v, the charge q and the flux linkage ϕ. The first two quantities can be connected to the second ones through a time derivative:

$$i = \frac{dq}{dt}$$
$$v = \frac{d\phi}{dt}$$

Then, the relationships between these variables expressed in terms of increments read

- $dv = Rdi$: The resistor connects a voltage variation to the corresponding current variation;
- $dq = Cdv$: The capacitor connects the charge variation to the corresponding voltage variation;
- $d\phi = Ldi$: The inductor connects the flux linkage variation to a the associated current variation;

Adducing symmetry arguments, he noticed that the existing circuital components fix a relationship between any two fundamental electric variables, except for the flux linkage and the charge. He therefore theorized the existence of a fourth non-linear component and he called it memristor.

In particular, this component can be expressed in two ways, depending on the variable considered to be independent

$$d\phi = M(q)dq \Leftrightarrow dq = G(\phi)\phi \tag{1.12}$$

Naturally, this component must be non-linear or it would simply be a resistor. Instead, if it is non-linear its conduction properties must depend on q or on ϕ and, in these cases, the functions M and G in Eq.1.12 are respectively called memristance and memductance. The interesting thing about this component is that it depends on the past history of the port variables (current and/or voltage)

$$\begin{cases} q(t) = \int_{-\infty}^{t} i(\tau)d\tau \\ \phi(t) = \int_{-\infty}^{t} v(\tau)d\tau \end{cases} \tag{1.13}$$

Therefore, its value must depend on the whole history of the device. In 2008 a group at the HP Labs led by Stanley Williams claimed that they had found Chua's memristor while they were trying to explain the pinched I-V curve of the TiO_2-based ReRAM they had designed [37]. This would generate a wave of enthusiasm that increased a lot the popularity of this component. This generated great interest in the research community of material scientists and electrical engineers which persists to these days and pushed the quest for brain-inspired analogue computers. Additionally, as said before, the neuroscience community struggled for years in the attempt to define rules well encoding learning. Furthermore, the goal in neuroscience is to express these rules in terms of dynamic equations for the synaptic weights, that are conductances. So lately, the theory behind the memristor seemed the natural way to go in the neuromorphic community. There has also been some criticism in subsequent works [38, 39] which adduced evidences of possible inconsistencies between the theoretical concept of memristor and the physical phenomena of resisting switching in emerging memory devices. Since this is not a theoretical work on memristive devices, the focus is put on the experimentally verified presence of resistive switching in ReRAM devices. This holds independently of the possibility to crystallize the neuronal dynamics by means of the theory of memristive devices.

Part I

Learning Algorithms

Chapter 2

Neuronal Dynamics

The Leaky integrate and fire model provides a well established characterization of many neuronal phenomena at the basis of the learning capabilities in the brain. Ultimately, this model describes the brain as a time-evolving dynamical system whose state variables are the membrane potentials of the neurons. This chapter brings together some results in the theory of dynamic systems, which are useful for the understanding of neuronal dynamics. The starting point is the Grossberg theory of recurrent neural networks, that complements the description of the brain contained in the previous chapter. In this way the Additive Model is introduced, corresponding to a description of the neuronal dynamics as a nonlinear dynamical system. In this model, time is not an explicit variable and the system is said to be autonomous. The reason why such a mathematical construct is capable of learning is mathematically understandable via the Cohen-Grossberg theorem. By stating the theorem, the conditions of validity identify the cases in which the model can be interpreted in a physically oriented way, by means of an energy function.

2.1 Grossberg theory of Recurrent Neural Networks

The neural network models considered in this work are known ever since the first formulation of the mathematical theory of neural networks. In 1957, Stephen Grossberg, as a college freshman, introduced a novel computational neuroscience paradigm that would constitute the foundation of most current biological neural network research. His ground-breaking idea was to use nonlinear systems of differential equations to model the dynamics of the brain in its learning and inference processes. These results started to appear in the scientific discourse only in 1964, when his monograph [40] was published. Grossberg wanted to create a learning model to describe the evolution of an adapting agent in a complex and changing environment. His research was primarily driven by open psychological questions and, for this reason, the behavioral model he ideated was not derived from the observation of experimental data, but rather deduced from postulates of psychology. In a paper from 1969 [41], he labelled the findings of the previous years as *theory of embedded fields*.

2.1.1 Theory of embedded fields

In its simplest form, this theory defines the agent as an abstract learning machine $\mathscr{M}$. An external actor in the learning process behaves as a teacher, presenting $\mathscr{M}$ with a list of “events” to be learned. Each event is identified with a "letter", corresponding to the abstract label in the language

of the machine. If the teacher wants $\mathscr{M}$ to learn the list of events $A \rightarrow B$, just as with a human being, (s)he should present several times the event A followed by the event B. If after some time the machine answers B when it is presented with the event A, then the machine has learned the list. In other terms, the learning machine has encoded an association. This is a *deterministic* learning theory, aiming at describing stimuli and response of $\mathscr{M}$ to a changing environment. To do so he assumed that each event, just as letters, corresponds to a single atomic unit than cannot be decomposed any further. Hence, each letter must be represented by a single state in $\mathscr{M}$. In order to do this he defined an abstract point p_i in $\mathscr{M}$ for each behavioral unit $r_i = 1,2,\ldots,n$, i.e. for each possible state of the machine (see Fig.2.1). Given this he observed that, after a stimulus

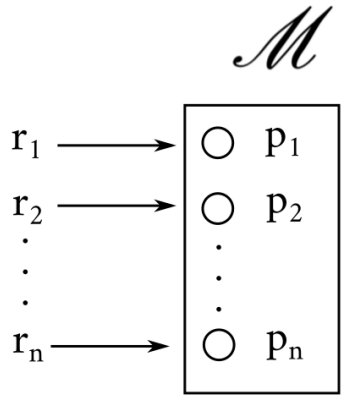


Figure 2.1: Abstract learning machine defined by Grossberg in the theory of embedded fields. To each behavioral unit (letter) it is associated a node in the machine.

is injected, $\mathscr{M}$ must react to it in some form to be capable of learning. He therefore defined a real valued function of time $u_i(t)$ for each point p_i. The value of $u_i(t)$ quantifies how recently r_i has been presented to $\mathscr{M}$. Hence, this quantity has the role of a memory of the sensory input r_i. Let us now consider the case in which we want to teach the list of events $A \rightarrow B \rightarrow C$. Just as before, each event is to be presented to $\mathscr{M}$, waiting some time Δt in between any two letters. He noticed that, as Δt is increased, the influence of A and B on the response of any learning machine gradually changes, ultimately becoming negligible. In other terms, a human being will learn the list if the three letters are presented by the teacher in succession. Instead, if the teacher says the letter A and presents the letter B after some hours or days the association cannot be created since the two events seem not correlated to the learning individual. In more technical terms the effects of prior presentations gradually wear off, ultimately becoming negligible. In addition to this, since the interest is on macroscopic phenomena, each $u_i(t)$ can be assumed to be continuous and differentiable. Another essential reasoning for the definition of the model is that if r_i is never presented to $\mathscr{M}$, then $u_i(t)$ remains at a fixed equilibrium value, which is (initially) set equal to zero. If r_i is presented to $\mathscr{M}$ at time t_i, then the memory variable should be expected to temporarily assume non-equilibrium values $u_i(t)$, that can be forced to be positive by convention. As already mentioned, the effect of an event must wear off and, for this reason, the final model must encode the decay of $u_i(t)$ after $t = t_i$. Coherently with this assumption, the quantities u_i can be associated to a specific type of memory called *short term memory* (STM).
Let us now suppose that the machine has learned the list of events $A \rightarrow B$. If the machine is then presented at time t with A and the teacher wants that after a time lapse τ_{AB} $\mathscr{M}$ rises the answer B, this must correspond to the presence of a signal travelling from u_A to u_B at finite velocity along a pathway e_{AB}. On the other hand, when $\mathscr{M}$ has not yet learned the list $A \rightarrow B$, other responses

than B must be possible to A, otherwise there would be nothing to be learnt. This implies that the machine must define a function $z_{AB}(t)$ that encodes the fact that B occurs in response to A. This function represents the correlation existing between the past value of $u_A(t)$ with $u_B(t)$ at all times. Since this is something that once learned must remain for a long time, he considered the terms z_{ij} to be associated to a form of long term memory (LTM). Additionally, since the list $A \to B$ is different from the list $B \to A$, then $e_{AB} \neq e_{BA}$ and $z_{AB}(t) \neq z_{BA}(t)$. In Fig.2.2 it is possible to see how e_{AB} is represented as an arrow whose head is labelled as N_{AB}. Finally, any

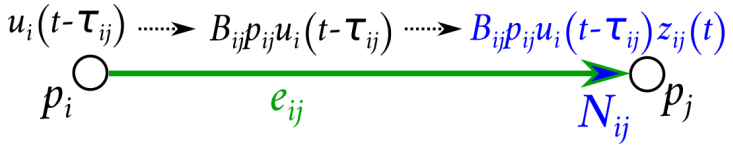


Figure 2.2: Schematic representation of the travelling signal theorized by Grossberg for encoding the transfer of information.

learning machine must be capable of learning both the lists $A \to B$ and $C \to B$. Hence, the state B is defined by combining the information coming from the surrounding units.
These postulates can be put in mathematical terms, yielding the first expression of the Grossberg model of adaptive behavior [42]

$$\begin{cases} \frac{du_i}{dt} = -A_i u_i(t) + \sum_{m=1}^{n} u_m(t-\tau_{mi}) B_{mi} z_{mi}(t) + I_i \\ \frac{dz_{jk}}{dt} = -Q_{jk} z_{jk}(t) + B_{jk} u_j(t-\tau_{jk}) u_k(t) \end{cases} \tag{2.1}$$

This corresponds to a system of coupled non-linear differential equations for the STM and LTM traces. Both dynamics are described by a passive decay term, forcing them to converge to a steady state in a sufficiently long time. Then, the STM trace is subject to a forcing external input I_i and perceives the information transmitted from the surrounding nodes proportionally to their past activity. On the other hand, both the units cooperate to determine how the correlation term z_{jk} has to be updated.
Subsequently [43], he further improved the model with the introduction of an anti-correlation term and by modifying the way the activity of the abstract points enters in the feedback terms. The improved model reads as

$$\begin{cases} \frac{du_i}{dt} = -A_i u_i + \sum_{j=1}^{n} [u_j(t-\tau_{ji}) - \Gamma_{ji})]^+ B_{ji} z_{ji}^+ - \sum_{j=1}^{n} [u_j(t-\tau_{ji}) - \Gamma_{ji})]^+ C_{ji} z_{ji}^- + I_i \\ \frac{dz_{ji}^{\pm}}{dt} = f_{ji}^{LTM}(\{z_{kl}^{\pm}\}_{k,l=1}^{n}, \{u_k\}_{k=1}^{n}) \end{cases} \tag{2.2}$$

where $[x]^+ = \max[x,0]$. Note that only the activity dynamics is here reported explicitly because, while modelling this term is necessary for this work, the dynamics later discussed for the LTM traces is of a different nature.
Given the psychology derived theory, it is then possible to recognize in each term of Eq.2.2 a neurophysiological counterpart. In particular let n be the number of neural cell bodies. The STM variable $\{u_i(t)\}_{i=1}^{n}$ must represent a relatively quickly evolving quantity and, for this reason, it can be identified with the average membrane potential. Going to the sums in the expression, they define the interconnection between two otherwise separated cell bodies. Hence, the term B_{ki} (C_{ki}) can be thought of as containing a structural information since when it is different

from zero there must be an excitatory (inhibitory) axon e^+_{kj} (e^-_{kj}) connecting u_k to u_j. In the previous derivation and in Fig.2.2 it is also possible to recognize in N^+_{kj} (N^-_{kj}) the synaptic end of e^+_{kj} (e^-_{kj}). Furthermore, z^+_{ki} (z^-_{ki}) are assumed to quantify the excitatory (inhibitory) chemical transmitter release in N^+_{kj} (N^-_{kj}). In Eq.2.2 the coefficients B_{ki} and C_{ki} do not only have a structural information. In fact, they also encode a proportionality factor for the description of the spiking frequency created by the activity of the neuron u_k in e^+_{kj} (e^-_{kj}) in the time interval $[t, t+dt]$. This is to say that, if this activity exceeds a threshold Γ_{ki}, then the neuron fires with a certain firing rate, proportional to $[u_k(t)-\Gamma_{ki})]^+B_{ki}$ ($[u_k(t)-\Gamma_{ki})]^+C_{ki}$). The time lag for the signal to flow across the axon e_{kj} is labelled as τ_{ki}. All together, if at time t the signal from u_k reaches N^+_{kj} (N^-_{kj}) it induces a release of the neurotransmitter in the extracellular environment separating the axon from u_j at a rate proportional to $[u_k(t-\tau_{kj}-\Gamma_{kj})]^+B_{kj}z^+_{kj}$ ($[u_k(t-\tau_{kj}-\Gamma_{kj})]^+C_{kj}z^-_{kj}$). All excitatory signals are then added while all the inhibitory signals are subtracted to u_j. Finally, $u_j(t)$ is also subject to a passive decay term $-A_iu_i$ and sensory inputs are introduced via the term $I_i(t)$. This last term can either correspond to a stimulus to the neuron from an experimentalist or to a signal coming from independent cells.

2.1.2 Additive Model

While in Eq.2.2 are already contained all the neurophysiological informations that are expected to be present in a neuron, the more commonly used model is the so called Additive STM equation [44]:

$$\frac{du_i}{dt} = -A_iu_i + \sum_{j=1}^{n} g_j(u_j)B_{ji}z^+_{ji} - \sum_{j=1}^{n} f_j(u_j)C_{ji}z^-_{ji} + I_i \tag{2.3}$$

If one considers the averaging operation in time and neurons discussed in the previous chapter, this is nothing else than the formulation in terms of rate functions of Eq.2.2, which is a spiking model. The non-linearities have been substituted with the generic functions $f_j(\cdot)$ and $g_j(\cdot)$, called *activation functions*. In Fig.2.3 it is possible to graphically observe the averaging operation. In fact, thanks to the Grossberg's theory, the average function introduced in the Leaky integrate and fire model must be a function of the average membrane potential in the pre-synaptic neuron, that was the main missing information in the Leaky integrate and fire model. Once fixed the interval of integration and the biological neurons to be averaged, it is possible to map the spiking intensity to a certain averaged membrane potential. The higher this value, the higher must be the number of spikes fired in the interval. Additionally, for low numbers and high numbers of spikes there is a saturation in the function since there is always a minimum and a maximum rate.
Going back to the Additive model, a unique feedback term is generally defined by lumping the LTM traces and the connection strengths in a unique contribution. When this is done the additive model for the STM traces is written in the simpler form

$$\frac{d}{dt}u_i = -A_iu_i + \sum_{j=1}^{n} g_j(u_j)z_{ji} + I_i \tag{2.4}$$

In the years following the derivation of this model, Grossberg extensively explored its applicability to various tasks, including vision, pattern recognition, reinforcement learning and many others [43–46]. The suitability of the Additive Model to mimic so many of the brain functionalities combined with its relative simplicity makes it a solid cornerstone of neural network research to our days.

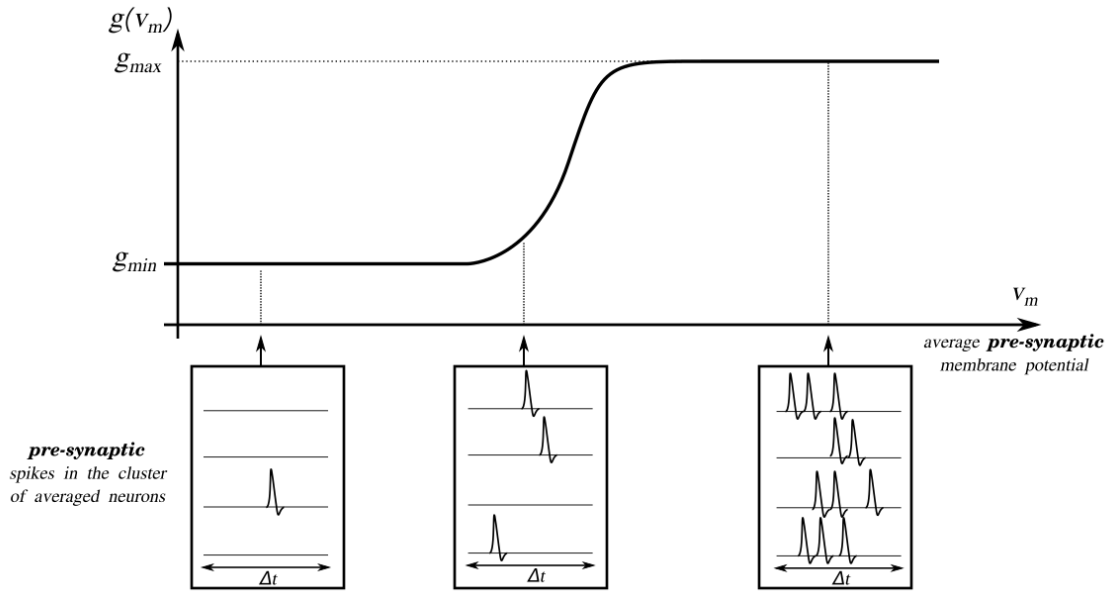


Figure 2.3: Graphical representation of the conversion of spikes into rates. The spikes of a population of neurons are averaged in time and in neurons, converting the spiking information into a rate information.

Despite this model started to appear in the neuroscience community in the 60s, it is usually known as the *Hopfield model* ever since Hopfield independently rediscovered it in the years going from 1982 [47] to 1984 [48]. In the review paper on nonlinear neural networks [44], Grossberg explained this as a consequence of the lack of mutual dialog between the psychology world and the physics and engineering one, induced by the progressive compartmentalisation experienced by the research world starting from the 20th century. As a tribute to both the two scientists, in this work this model is referred to as the Grossberg-Hopfield model.
While Eq.2.4 represents the dynamics followed by the STM traces, Grossberg also introduced in his learning framework the dynamics associated to the LTM traces, needed by the learning machine to improve its knowledge of the problem and to adapt:

$$\frac{d}{dt}z_{ji} = -F_{ji}z_{ji} + G_{ji}f_i(u_i)g_j(u_j) \tag{2.5}$$

Where the factor $G_{ji}f_i(u_i)g_j(u_j)$ is generally called the *Hebbian term*. The dynamical equations 2.4 and 2.5 describe two general types of nonlinear processes. Despite the fact that they are in principle coupled, they are expected to occur at considerably different time scales. This is to say that the cooperative-competitive nonlinear feedback processes described by Eq.2.4 operate on a relatively fast time scale. The network receives external information and elaborates it using as a computing media the STM values. On the other hand the LTM processes occur on a longer time scale. The terms defining the nature and strength of interconnections slowly vary, interacting with the voltage membranes via feedback loops. This dynamics is essential since it encodes the adaptability of the model to the external stimuli. This is the point where this historical path starts to diverge from the Grossberg theory. While Grossberg focused also on the dynamical sub-system in Eq.2.5, his approach is not deepened in the following. In fact, the goal here is to study the alternative learning approach described by Bengio and Scellier [1]. One should anyhow mention that both of them rely on an Hebbian learning mechanism and a proper mathematical

mapping could be identified in between them.
Before proceeding with the learning algorithms, there is still one essential result found by Grossberg worth being recalled as it will allow for a smoother introduction to the Bengio and Scellier approach to RNNs. An essential result derived by Grossberg is the formal proof of the conditions under which a very general dynamical system describing a learning machine can be used to do inference.

2.1.3 Learning lists

Before presenting the theorem, it is first necessary to introduce the concept of Associative memory. An associative memory, also known as Content Addressable Memory (CAM), is a memory in which it is possible to store some data, associated to a certain information at the input. Once the network has learned the association existing between the input and the data to be stored, if we introduce a partial information about the input, the CAM is capable to retrieve the missing information stored in it. This is an extremely general concept and many machine learning problems such as classification and reconstruction correspond to associative problems, solvable by these kinds of networks. Let us go back to analyze the neural network described by Grossberg in this perspective. In principle, the Grossberg machine is designed to respond to an input stimulus by converging to a certain state. Let us now assume that, in presence of a sustained input, the machine, following the dynamics of Eq.2.4, always converges to an equilibrium pattern. Then, the machine is capable of learning because, according to the Grossberg's analysis (Eq.2.5), it is able to adapt, namely to learn the proper response to a given excitation. The state to which the machine converges to is defined by the values of the STM traces, identifying a pattern in the network corresponding to the machine $\mathscr{M}$. Hence, an association of a response to a stimulus corresponds to identify a readable information to each pattern. Let us now assume that the

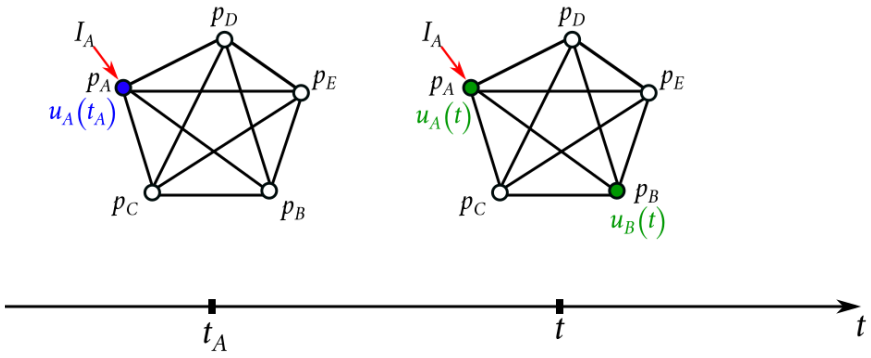


Figure 2.4: Grossberg network: learning the list $A \rightarrow B$.

machine has learned the list $A \rightarrow B$ but we do not know it. With respect to Fig.2.4, observing $\mathscr{M}$ allows to notice that it is a network that can be stimulated at its nodes. Moreover we notice that each node has a label, corresponding to an abstract event. If a stimulus I_A is switched on, the dynamics of node p_A is perturbed as described in Eq.2.4. Similarly, also the surrounding nodes are perturbed according to the values of the LTM traces. If the network manages to converge, the only node that is switched on must be node p_B and, since from the external this node is labelled as the abstract event B, an observer can discover that the answer to the stimulus A is stored at the node associated to the event B. Hence, the neural network defined by Grossberg is a general

form of CAM. Indeed, this is the simplest form of learning one can require to the network. This kind of CAMs structured as the Grossberg network were then proved to be applicable to many inference problems. In later chapters it is shown how the network associated to Eq.2.4 can be used in the reconstruction problem while a slightly modified version of it can be used to solve the classification problems. However, before proceeding it is essential to formalize the discussion and to determine under which conditions the system in Eq.2.4 manages to converge to an equilibrium pattern. In fact, this is a necessary condition for the network to be used for learning tasks.

2.2 Dynamical systems

In the previous section the characterization of the Additive Model was completed. This model assumes that the state of the system can be described in terms of the values of the STM traces, acting as state variables. Then, at each instant of time the system evolves dynamically, following rules that are uniquely dictated by the values of its state variables at that instant of time. For this reason the Additive Model is called *state space* model. Additionally, since the time dependence does not directly appear in the dynamics, the system is defined to be *autonomous* [49]. Therefore, the main result of the Grossberg's theory is that the time evolution of the membrane potentials can be described as an autonomous non-linear system of coupled differential equations. By defining the state vector $u(t)$ and by collecting the dynamic equations on the right hand side in a unique field $\mu(u(t))$, the neuronal dynamics can be written as

$$\frac{d}{dt}u(t) = \mu(u(t)) \tag{2.6}$$

Generally, the time derivative of the state vector is interpreted as an abstract velocity vector $\dot{u}$, so that the right hand side of the dynamical system represents a velocity field [50]. In any case, if the number of units is N_u, the time evolution of the system can be thought of as a well defined dynamics in a N_u-dimensional states space. In order to give a graphical interpretation of these concepts, one can imagine to initialize the system at some point in the state space $u(t'_0)$. The system then evolves according to the rules contained in the dynamical system. The

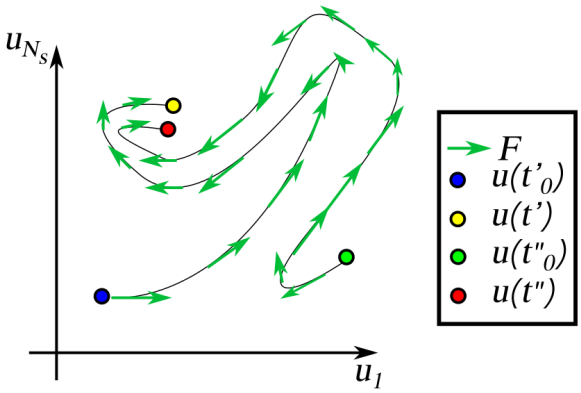


Figure 2.5: Schematic evolution of a dynamical system in the states space.

resulting trajectory will be travelled by with a time dependent speed. At each instant of time, corresponding to a different state vector, it is possible to represent the velocity vector field as

a collection of vectors tangent to the trajectory. Repeating this procedure starting from each point in the states space it is then possible to build up the velocity field in a graphical way (see Fig.2.5).

2.2.1 Equilibrium Points

Let us now assume that there is a point in the state space where the vector field is vanishing $\mu(\bar{u}) = 0$. This point is called a *fixed point* of the system. In fact, if the system ends up in $\bar{u}$ it will stop, since its dynamics is vanished. Nonetheless, in noisy systems like our brain [51], the fact that the dynamics vanishes in one point does not guarantee that the system will always remain in that condition. Depending on the nature of the velocity field around $\bar{u}$, in presence of a perturbation $u(t) = \bar{u} + \delta u(t)$ the system could either go back to the fixed point or escape from it, starting again its wandering through the state space. In order for a system to be amenable to do learning, it should be characterized by stable regions in the space, called *equilibrium regions*. Namely, the state of our brain should respond to a stimulus by converging to a stable configuration. In this way, this final configuration can be identified with the interpretation the learning agent is giving of the input it received.
It is therefore essential to define what it means for a system to be stable. A state $\bar{u}$ is said to be uniformly stable if, by choosing the starting point sufficiently close to it, its subsequent dynamics can be confined within a small neighborhood. In mathematical terms

$$\forall \varepsilon > 0 \exists \delta > 0 : \text{ if } \forall t > t_0 \;\; ||u(t_0) - \bar{u}|| < \delta \Rightarrow ||u(t) - \bar{u}|| < \varepsilon \tag{2.7}$$

Being uniformly stable does not guarantee that the system will end up exactly in the fixed point. For instance it could remain stuck in a cycle around the fixed point, satisfying stability but never reaching the desired point. The additional concept of convergence is to be introduced. A state is convergent if, when initializing the system sufficiently close to it and waiting a sufficiently long time, the system can be found arbitrarily close to the fixed point. Again, in mathematical terms this reads

$$\exists \delta > 0 : \;\; ||u(t_0) - \bar{u}|| < \delta \Rightarrow \lim_{t \to \infty} u(t) = \bar{u} \tag{2.8}$$

This being said, the two conditions are independent, as shown in Fig.2.6. However, if both the conditions are satisfied the system is said to be *asymptotically stable*. Additionally, a system is said to be *globally asymptotically stable* if it is stable and all the trajectories converge to a fixed point as times goes by. This is to say that it is the only equilibrium point of the system.
An extremely powerful approach for determining the stability of a system is the definition of a Lyapunov function, i.e. a scalar field defined on the state space:

Definition 2.1: Lyapunov's function

Given a dynamical system and the corresponding states space $\mathscr{S}$, the Lyapunov function is a scalar field E such that:

1. It has continuous partial derivatives with respect to all the state variables;
2. Given an equilibrium state $\bar{u}$, $E(\bar{u}) = 0$;
3. $E(u) > 0$ if $u \neq \bar{u}$

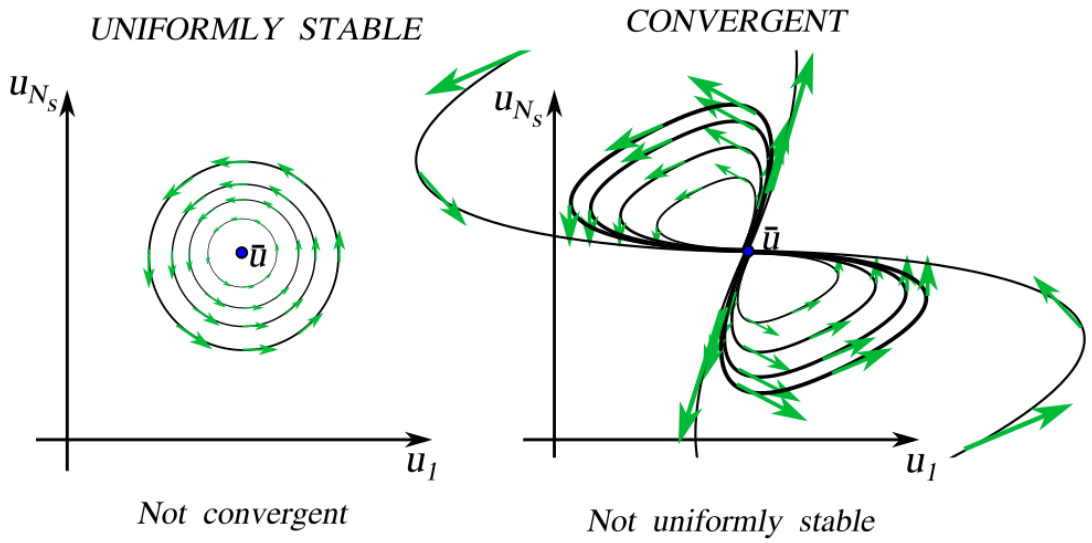


Figure 2.6: *Left:* A uniformly stable but not convergent system;*Right:* A not uniformly stable but convergent system;

Given this definition, Lyapunov proved that in order for an equilibrium point to be stable, this function must be non-positively varying in time in a small neighborhood of the equilibrium.

$$\frac{dE}{dt} \leq 0 \tag{2.9}$$

Moreover, in case of strict negativity the equilibrium point is asymptotically stable. In any case, while the presence of a Lyapunov function is a sufficient condition for having stability, it is not necessary. In fact, in many converging systems it is impossible to define a Lyapunov function.

2.2.2 The Cohen-Grossberg theorem

Grossberg and Cohen [52] proved the conditions under which the network described in 2.4 always approaches an equilibrium point in response to an arbitrary but clamped input pattern. As mentioned before, this corresponds to the study of the stability of the dynamical system but it ultimately results in determining whether the network can be used as a CAM or not. The final form of their theorem holds for a more general class of dynamical systems to which Eq.2.4 is just a particular case.

Theorem 2.1: Cohen-Grossberg theorem

Consider the dynamical system

$$\frac{d}{dt}u_i = a_i(u_i)\left[c_i(u_i) - \sum_{j=1}^{n} z_{ji} g_j(u_j)\right] \tag{2.10}$$

where $a_i(u_i)$ is called *amplification signal*, $c_i(u_i)$ is the *self-signal* function and $g_j(u_j)$ is the *other-signal* functions. If:

1. The coefficient matrix is symmetric: $z_{ij} = z_{ji}$;
2. the function $a_i(u_i)$ is non-negative: $a_i(u_i) \geq 0$;
3. the function $g_i(u_i)$ is **non-decreasing**: $g_i'(u_i) \geq 0$

Then, the dynamical system admits the global Lyapunov function

$$E = -\sum_{j=1}^{n} \int^{u_i} c_i(\xi_i) g_i'(u_i) d\xi_i + \frac{1}{2} \sum_{j,k=1}^{n} z_{jk} g_j(u_j) g_k(u_k) \tag{2.11}$$

Proof. By direct computation of the time derivative of E along a trajectory

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial u_i} \frac{du_i}{dt} = -\sum_{i=1}^{n} a_i(u_i) g_i'(u_i) \left[c_i(u_i) - \sum_{j=1}^{n} g_j(u_j) z_{ji} \right]^2 \tag{2.12}$$

Then, if the hypothesis are satisfied, along the trajectory the function E will necessarily be

$$\frac{d}{dt} E \leq 0 \tag{2.13}$$

Therefore it is a Lyapunov function of the system. □

The class of dynamical systems satisfying the Cohen-Grossberg theorem is extremely broad. As such, it became a fundamental tools in neurodynamics for studying different possible models. This theorem is essential since the existence of a global Lyapunov function guarantees that every trajectory approaches one of a possibly large number of equilibrium points. In the numerical part of this work the dynamical system used is the one in Eq.2.4. In particular, $a_i(u_i) = 1$ and the same non-linear function $g(u_i) = g_i(u_i)$ is used for all the neurons. Moreover, the LTM traces are collected in a symmetric matrix W, built in such a way that $z_{ji} = -W_{ij}$ and $W_{ii} = 0$, namely there is no self loop. In this way the STM additive model reads

$$\frac{du_i}{dt} = -u_i + \sum_{j=1}^{n} W_{ij} g(u_j) + b_i \tag{2.14}$$

With $c_i(u) = -u_i + b_i$. Furthermore, according to the Cohen-Grossberg theorem, the corresponding energy like function will be

$$E = \sum_{i=1}^{n} \int^{u_i} \xi_i g'(\xi_i)\, d\xi_i - \sum_{i=1}^{n} b_i g(u_i) - \frac{1}{2} \sum_{j,k=1}^{n} g(u_j) W_{jk} g(u_k) \tag{2.15}$$

The Cohen-Grossberg theorem proves the global stability of the additive model by defining a Lyapunov function, i.e. and energy-like function. If the state space is defined as the N_u-dimensional vector space populated by the possible values of the STM traces $u = \left[u_1, \ldots, u_{N_u}\right]^t$. Then, the dynamical system in Eq.2.14 will dictate the evolution of the point $u(t)$ over time, up to its convergence to an equilibrium point. The interesting thing about the introduction of an energy-like function is that it can be thought of as a potential landscape defined in the $N_u + 1$-dimensional world. Then, given any initial condition, the system $u(t)$ will evolve in the state space toward the minimization of the energy function. This evolution will stop only if the system manages to reach a local minimum.
As a last interesting remark, Cohen and Grossberg proved that the theorem holds not only for monotonously increasing functions, as required by Hopfield [48], but rather under the milder requirement that the *other-signal* functions $g_i(\cdot)$ are just not-decreasing.

2.2.3 Attractors and learning

An essential feature in the Additive model is that it is a dissipative system. Thanks to the damping effect in the dynamics, this feature generally guarantees the presence of points or of entire regions of vanishing velocity vector field, called attractors [49]. This property is desired since it is possible to exploit these attractors in order to do computation.

Model Learned

Suppose that some of the variables in the system introduce some external information and the network is designed with the aim of processing it. This might happen through the input currents defined before, but later in the work are discussed alternatives ways in which the input can be introduced in the network. The Grossberg-Hopfield network has the limit that the dimension of the network is fixed by the number of input nodes. On the contrary, a more general architectural choice admits many more state variables in addition to the input ones. This is the kind of architecture employed by Bengio and Scellier and deepened in the next chapter. In any case, the neurodynamics of the Additive model keeps holding and the most striking difference of the two approaches is the way the inputs are presented to the network. While in the Grossberg-Hopfield the inputs I_k are introduced as an additional bias terms ($b_k \to b_k + I_k$), in the Bengio-Scellier case they appear as clamped neurons ($\{u_k\}_{k=1}^{N_i} = \{I_k\}_{k=1}^{N_i}$). In both the approaches, some of the variables of these models are associated to a well defined meaning, that the machine is required to learn. These variables are usually called output nodes. In other terms, given an external stimulus, thanks to the damping nature of the dynamics the system will converge to an equilibrium point. This equilibrium point corresponds to a state in the states space in which the dynamics vanishes. In the additive model, this corresponds to the parametric formula in W and in the bias term I:

$$\bar{u}_m(I) = \sum_{k=1}^{N_u} W_{mk} g(u_k) + b_m \tag{2.16}$$

Generally, the quantity considered for a neuron is the associated firing rate. Therefore, if the N_o output nodes are identified as the quantities corresponding to the response of the machine to a stimulus, the recurrent neural network is ultimately a N_o-dimensional parametric function

$$\hat{f}_k^\theta(I) = g(\bar{u}_k(I)) \quad k = N_u - N_o + 1, \ldots, N_u \tag{2.17}$$

This equilibrium is strongly dependent on the parameters of the dynamical system and on the input stimulus, acting as an external force field in the additive model. Therefore, the learning procedure in terms of dynamical systems corresponds to modify the parameters of the differential equations so as to modify the exploration of the state space. More precisely, given the desired values of the state variables, the target region is identified. Hence, the goal is to define a system capable of converging to this region once the partial information is presented at its inputs. In mathematical terms, if we want to learn the response to a stimulus $H(I) \in \mathbb{R}^{N_o}$, the goal is to tune the parameters of the network so that the stationary point will fit the target response

$$\hat{f}_k^\theta(I) = \lim_{t\to\infty} g(u_k) \doteq H_k(I) \quad k = N_u - N_o + 1, \ldots, N_u \tag{2.18}$$

There are many ways in which this computing paradigm can be employed, and later in the thesis two typical ones are discussed: reconstruction and classification. However, before doing that, some concepts on energy-based models need to be introduced. In fact they are at the basis of the learning algorithm employed in this work.

Nature of the information learned

The nice thing about the Grossberg-Hopfield model, also including the more general Bengio-Scellier model, is that the nature of the information encoded is qualitatively easy to be understood. In its most general form, the models considered in the algorithmic investigation here presented is described by the equation

$$\frac{ds_m}{dt} = -s_m + \sum_{k=1}^{N_u} W_{N_i+m,k} g(u_k) + b_m \tag{2.19}$$

these are the equation that are later associated to the Bengio-Scellier model, that is more general because it considers the input as neurons not evolving in time. In the previous sections the

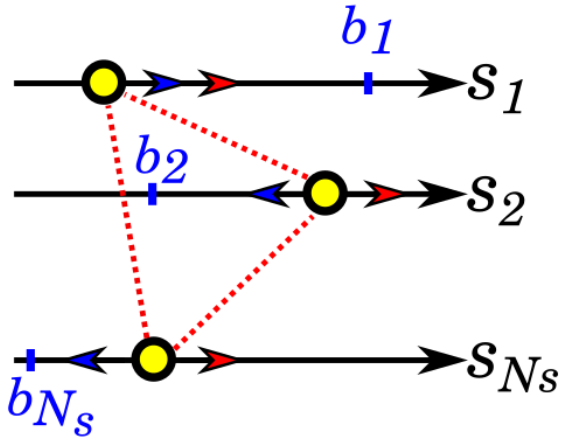


Figure 2.7: Schematic representation of the N_s one dimensional dynamics experienced by the agents of the model.

neuronal dynamics has been described as the time evolution in a N_u-dimensional state space (or N_s since the first N_i nodes do not evolve). Nonetheless, the motion can be described by considering each single neuron independently in its 1D evolution. In this perspective, the neuron will experience two kinds of forces. The first one is an attractive force toward the bias:

$$\mu_m^b = -(s_m - b_m) \tag{2.20}$$

while the second is a correlation term

$$\mu_{N_i+m,k}^w = W_{N_i+m,k} g(u_k) \tag{2.21}$$

While the first term represents an attracting field toward the point of space b_m, the second term represents the influence the unit u_k has on the agent s_m. Depending on the sign of the synaptic weight, three conditions can be identified:

- $W_{N_i+m,k} > 0$: Positive (Negative) values of the unit u_k will drive the state s_m to positive (negative) values. The nodes are correlated in their dynamics;
- $W_{N_i+m,k} < 0$: Positive (Negative) values of the unit u_k will drive the state s_m to negative (positive) values. The nodes are anti-correlated in their dynamics;

- $W_{N_i+m,k} = 0$: the state s_m is independent on the node u_k. The nodes are not correlated in their dynamics.

When dealing with symmetric weights, an energy function like the one in Eq.2.15 can be defined. Hence, in the energy picture the two fields will give rise to two energy terms, to be summed up to obtain the overall contribution. On the one hand there is the bias term

$$E_m^b = \int^{s_m} d\xi (\xi_m - b_m) g'(\xi) \tag{2.22}$$

This term is non-trivial and its interpretation is better to be done in term of the associated velocity field. On the other hand the weight matrix yields the term

$$E_{mk}^w = -g(u_m) W_{mk} g(u_k) \tag{2.23}$$

That allows to see that, in the perspective of minimizing the energy, the neurons tend to align or to dis-align based on their correlation, i.e. the sign of the weight connecting them.

Chapter 3

Learning: Equilibrium Propagation

This chapter is devoted to the discussion of the algorithms used in this thesis project. The main interest is on Equilibrium Propagation, a learning algorithm whose working principle is based on the definition of an energy function in the states space. For this reason, the discussion will start from the description of energy based models. Afterward, Equilibrium Propagation is introduced in the first version in which it was conceived by Bengio and Scellier. This algorithm is rigorously defined mathematically since it is based on the Bengio-Scellier theorem, stated in the process. Its mild conditions of validity make it extremely general. However, the need for a well defined energy function is by itself a limit of validity. This led Scellier et al.[5] to extend the algorithm to systems not admitting an energy function. Finally, the first result of the project is presented, corresponding to an alternative learning algorithm for training networks not admitting an energy function.

3.1 Energy-based models

Cohen and Grossberg [52], as well as Hopfield [47, 48], introduced the energy-like function mainly to investigate the existence of stable attractors in the state space of their network, necessary condition for encoding information. As previously mentioned, time continuous recurrent neural networks basically constitute nonlinear dynamical systems. Since the Lyapunov function is a fundamental quantity in the theory of dynamical systems, it enters naturally in the description. Nonetheless, as reviewed by LeCun [53], the idea of using energies to do inference and learning would later become a standalone paradigm in machine learning theory. This approach, called *energy-based learning*, is applicable to many machine learning models. These models are called *energy-based models* and the RNNs previously discussed, together with some learning rule for the LTM traces, are only a subset of them.
Many problems in machine learning are aimed at finding the relationship existing between two sets of variables. On the one hand there is a set $X = \{x^{(i)}\}_{i=1}^{N_{data}}$ of input data. In abstract terms, $x^{(i)}$ is the sensory input of the machine to be trained. On the other hand there is the set of the corresponding interpretations $Y = \{y^{(i)}\}_{i=1}^{N_{data}}$. For instance, X may be a set containing pictures of cats and dogs while Y the associated set of labels "cat" and "dog" or 1 and 0. In general $x^{(i)}$ will be a N_i-dimensional variable while $y^{(i)}$ a N_o-dimensional one. Between the two sets of variables

there are some dependencies and the goal of machine learning is to find a model capable of learning them. The goal is therefore to educate a machine in such a way that whenever it is presented with a certain data $x^{(i)}$ it gives the correct interpretation, namely it outputs the correct variable $y^{(i)}$. The idea at the basis of the energy based models is to postulate the existence of an energy function depending on both the variables x and y: $E(x,y)$. This function quantifies the degree of compatibility of the two variables and, by convention, it should be small when y is a good interpretation of x and high otherwise. A schematic representation of this concept is reported in Fig.3.1

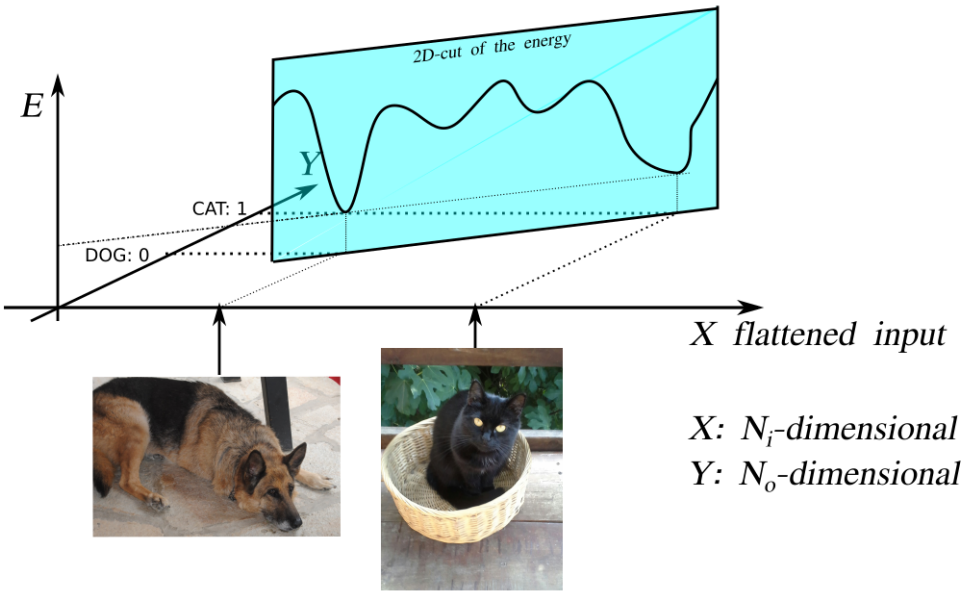


Figure 3.1: Pictorial representation of the Energy function for classification purposes.

3.1.1 Energy-based inference and learning

Once postulated the existence of such an energy function, it is possible to describe the inference process. The energy function is the tool that the machine to be trained has at disposal to make sense of the inputs it receives. During inference, an input x is presented to the machine. Based on its knowledge of the world, the machine must give its interpretation of the input. The knowledge of the world is encoded in the energy function $E(\cdot,\cdot)$. Therefore, if all possible outputs are collected in the set $\mathscr{Y}$, the inference problem can be written as the minimization problem

$$y^* = \arg\min_{y\in\mathscr{Y}} E(x,y) \tag{3.1}$$

Therefore, each machine classifiable as an energy based model must be equipped with an energy function and with a minimization strategy for determining the best output to be associated to a given input.
While inference consists in minimizing an energy function, learning is the process by which the best energy function is selected. The idea is to define a parametric family $\mathscr{E}$, depending on the multidimensional parameter θ.

$$\mathscr{E} = \{E(x,y,\theta), \quad \theta \in \Theta\} \tag{3.2}$$

Then, one usually introduces a target function $J(X,Y,\theta)$ with which it is possible to quantify the quality of the selected model θ. This must be a function of the parameters and of all the input and output variables. In this way the function J provides a number expressing the goodness of the model θ in describing the given problem. Coherently with the convention taken for the energy function, it is reasonable to require this target function to take low values when the model is good and high values otherwise. Consequently, also learning can be expressed as a minimization problem, in the form

$$\theta^* = \arg\min_{\theta\in\Theta} J(X,Y,\theta) \tag{3.3}$$

Once solved this problem it is then possible to use the learned parameters θ^* for characterizing the energy function and, consequently, for doing inference.

3.2 Equilibrium Propagation

Section 2.1 introduced a family of time continuous recurrent neural networks that is generally known as time continuous Hopfield network. The idea at the basis of this model is to consider a system of agents evolving dynamically in presence of external fields, while mutually interacting. Previously it was discussed how the evolution of the STM traces can be mapped to a minimization of an energy-like function and this process was identified with what happens when the machine performs inference. Additionally, it was also introduced the dynamics for the LTM, namely for the parameters of the learning machine. In this section Equilibrium Propagation is introduced [1], corresponding to the framework introduced by Bengio and Scellier for training Hopfield-like models with equations different from the ones introduced by Grossberg. This introduces a learning algorithm that is particularly promising for hardware implementations.

3.2.1 The architecture

The architecture considered by Bengio and Scellier is more general than the one previously seen in two ways: first the network is enlarged through the introduction of additional nodes, increasing in this way the computational capacity; secondly inputs are not locally injected at one node but rather distributed to several of them as a weighted sum, where the weights are to be learned. In Fig.3.2 a schematic representation of such a network can be observed. The architecture is made of N_u units (or neurons) $u = \{u_i\}_{i=1}^{N_u}$. Each neuron can be either an input node, an hidden node or an output node. Any two units u_i and u_j mutually interact via directional synaptic connections, where W_{ij} expresses how much u_j influences u_i. In general $W_{ij} \neq W_{ji}$ but this preliminary discussion assumes $W_{ij} = W_{ji}$ and $W_{ii} = 0$. The nodes of the network can be divided into three sub-classes. First of all there are the inputs $\{x_j\}_{j=1}^{N_i} = \{u_j\}_{j=1}^{N_i}$. These are nodes that are always clamped and their role is to introduce the data to be interpreted inside the model. The inputs are then connected to two kinds of nodes. On one hand there are the output nodes $y = \{u_j\}_{j=N_i+N_h+1}^{N_u} = \{y_j\}_{j=1}^{N_o}$, corresponding to the units providing the final interpretation of the input. On the other hand there are the hidden nodes $h = \{u_j\}_{j=N_i+1}^{N_i+N_h} = \{h_j\}_{j=1}^{N_h}$, corresponding to additional agents whose presence increases the possibilities of interaction and, consequently, the possibility of learning inter-dependencies existing in between input and output variables. Hidden nodes and output nodes form a subclass called *state nodes* $s = \{u_i\}_{i=N_i+1}^{N_u}$. They differ from the input nodes in the fact that they evolve in time following a certain dynamics. For this reason these nodes correspond to the STM traces, while the input nodes are a generalization of

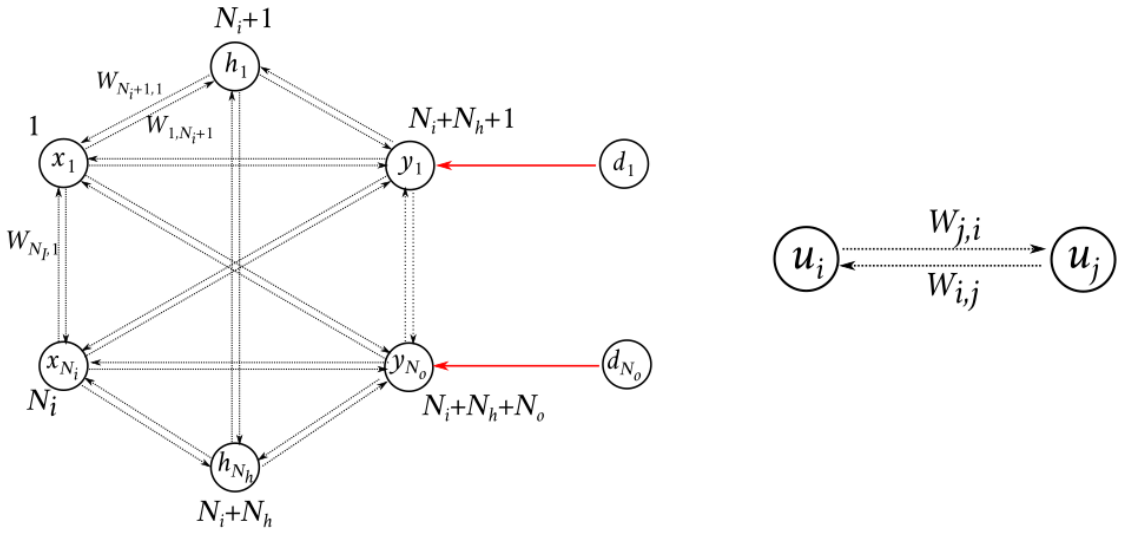


Figure 3.2: Schematic representation of a recurrent neural network. Three classes of nodes can be recognized: the input nodes (x), the hidden nodes (h) and the output nodes (y). All of the units are connected by directional synaptic weights. The nodes labelled as d are the so called target nodes and are used to find the best combination for the synaptic weights.

the biases in the Grossberg-Hopfield model. Following the notation used in [1], the neurons are here referred to using their associated state variable u_i. This can be interpreted as the average membrane potential of a coarse grained neuron, just as discussed when introducing the Grossberg theory. The model is characterized by the multidimensional parameter $\theta = (W, b)$, where W is the synapses matrix and b is the bias vector. The weight matrix can be non-zero only between nodes that are connected in the architecture. The biases are one per each state node. Once the sensory inputs are introduced via the input nodes, the synaptic weights distribute the input information to the other neurons and the whole network evolves under the influence of the sensory input. Similarly to before, while the value u_k can be interpreted as the activity of the k-th node, the information contained into it passes through a non linear function $g(\cdot)$ and is multiplied by a synaptic weight before to reach the subsequent neuron. The non-linearity maps the average voltage potential of the neuron in a proper firing rate. Bengio and Scellier proposed a supervised teaching strategy to train the model. By a two-stage learning setting, the network first relaxes to a preliminary equilibrium and then the correct target answer is provided to the output nodes by means of the external inputs $d = \{d_k\}_{k=1}^{N_o}$, shown in Fig.3.2.

3.2.2 The algorithm

Equilibrium propagation is an algorithm allowing to do energy based learning on the kind of architectures described in the previous sub-section. The cornerstone of this procedure is the Bengio-Scellier theorem, corresponding to a learning rule for an energy based model. This theorem is proved under very mild assumptions, so that it is valid for any energy function of the kind discussed in the Cohen-Grossberg theorem

$$E = -\sum_{j=1}^{N_u} \int^{u_i} (b_i - \xi_i) g_i'(\xi_i) d\xi_i - \frac{1}{2} \sum_{j,k=1}^{N_u} W_{kj} g_j(u_j) g_k(u_k) \tag{3.4}$$

As previously said, the parameters of the model are $\theta = (W, b)$, that identify the energy function as belonging to a family $\mathscr{E}$ of parametric functions. Each of them corresponds to a different possible model since, by changing W, one changes the way the network is recurrently connected. Similarly, by changing b one locally varies the value of an external field at each node, influencing the dynamics in the state space. Overall, u contains both inputs and outputs. Therefore Eq.3.4 is a function capable of quantifying the degree of compatibility between these nodes. Moreover, by varying the parameters θ it is possible to find the optimal energy function in this parametric family encoding the correct inter-dependencies between the different nodes. Before describing the learning procedure, let us discuss more concretely how a time dependent energy based model can be used to do inference.

Inference

Since we are using the Lyapunov function of the Cohen-Grossberg theorem, it is also possible to introduce the dynamic system used in this work for describing the evolution of the state nodes

$$\frac{dg(s_k)}{dt} = \frac{\partial g(s_k)}{\partial s_k}\frac{\partial s_k}{\partial t} = -\frac{\partial E}{\partial s_k} \qquad k = 1, \dots, N_s \tag{3.5}$$

An important difference with respect to the Grossberg-Hopfield network is that not all the nodes evolve. Moreover, one should notice that the dynamics considered is different with respect to the one introduced by Bengio and Scellier [1]. Thanks to the Cohen-Grossberg theorem some conditions under which this dynamic system converges to an equilibrium point are known exactly. In this way, after a sufficiently long time, the nodes converge to an equilibrium configuration, corresponding to a local minumum of the energy

$$s^0_\theta \in \arg\min_s E. \tag{3.6}$$

This is to say that, by defining the dynamic system associated to the energy function, the nodes evolve minimizing the Lyapunov function and this process is sketched in Fig.3.3. Once clamped the input, the only variables of the system are the state nodes. Therefore, in Fig.3.3 the state space is pictorially represented as the two sub-spaces of the hidden nodes and of the output nodes. More precisely, they have been represented after having distorted the average membrane potential into firing rates. In this space it is possible to define the energy profile, defining the cost of each configuration of the variables. The cost will depend both on the parameters of the model $\theta = (W, b)$ and on the value of x with respect to y, accordingly to the current model. After a random initialization of the state nodes, the system finds itself in the red point, corresponding to the initial condition. Subsequently, thanks to the neuron dynamics defined by the dynamical system it evolves toward the minimization of the energy, reaching the yellow point. That point is $g(s^0_\theta)$, and contains also the firing rate of the outputs $g(y^0_\theta)$. In this way, given a certain input the system proposes a certain output $g(y^0_\theta)$, namely it has performed inference. This process is called *free phase*. Figure 3.3 shows the target line, i.e. the correct output. It is worth pointing out that the target is independent on the hidden subspace. In fact, the hidden nodes introduce additional computing possibilities through their interaction with the inputs and outputs but they do not appear explicitly when it comes to do inference. Ideally, we would like the energy profile to be such that, after this evolution, the yellow points lays on the target line. To do so, a learning strategy needs to be defined for improving the energy landscape.

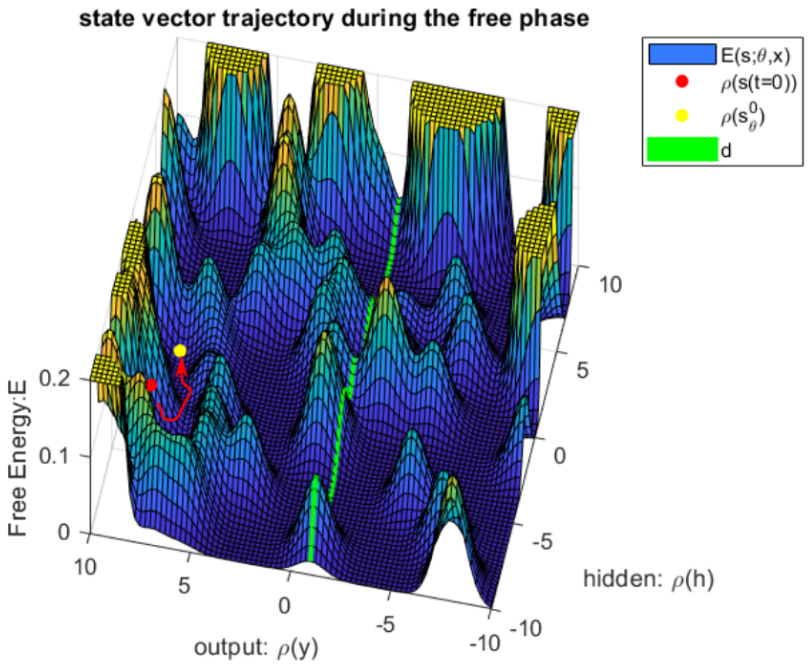


Figure 3.3: Free Phase: The system is randomly initialized in the rates space and evolves freely minimizing the energy. The final point is the free phase equilibrium.

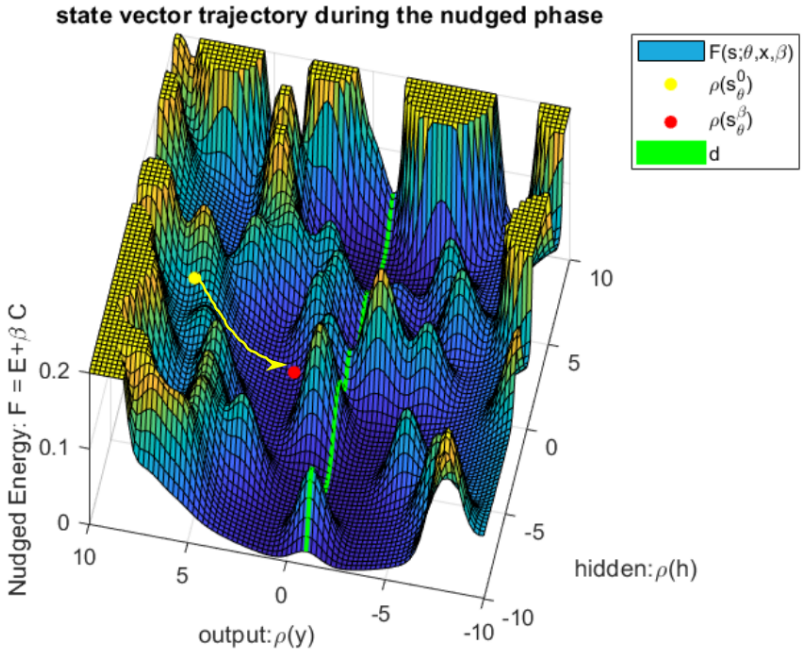


Figure 3.4: Nudged Phase: The system starts from the free phase equilibrium and evolves in a modified energy, tilted toward the target region.

Learning

Bengio and Scellier defined a function quantifying at each instant of time how far the current interpretation of the input is from the target region. This may be any distance function between y and d, and the one considered in the following is

$$C \doteq \frac{1}{2}\|g(y)-d\|^2 \tag{3.7}$$

An interesting feature of this choice is that, whatever point of the state space, the farther the interpretation is from the target, the higher the numerical value of C will be. Therefore, Bengio and Scellier proposed to use this kind of function as an external potential to drive the system toward the target. This is to say that, if one introduces the total energy

$$F = E + \beta C \tag{3.8}$$

where β is the *influence parameter* (or *clamping factor*), an additional information is introduced, suggesting to the system the direction to go to approach the correct region. If the local equilibrium reached in Fig.3.3 is taken as the new starting point, this augmented potential perturbs the equilibrium, forcing the system to evolve toward a close-by equilibrium. In this way a new prediction $g(y_\theta^\beta)$ is identified. This process can be observed in Fig.3.4 and the dynamics now becomes

$$\frac{\partial g(s_k)}{\partial s_k}\frac{\partial s_k}{\partial t} = -\frac{\partial F}{\partial s_k} \qquad k = 1,\dots,N_s \tag{3.9}$$

And the resulting minimization is named *nudged phase*.
When discussing the energy-based models, it was introduced the concept of target (objective) function. It was said that this function quantifies the goodness of the model in giving the right interpretation of the inputs. The idea of Bengio and Scellier was that the help given via the external potential is only provided during training, while during inference it must disappear. Once the model is trained, the energy function should be such that a good interpretation of the input is performed in the free phase. Therefore, if one considers a one-point data-set, for simplicity, a good candidate as a target function might be

$$J = C(\theta, v, s_\theta^0) \tag{3.10}$$

namely, the value of the cost function after the free phase has performed its interpretation. Using this idea, Bengio and Scellier argued that just requiring some reasonable assumptions on the energy function to be valid, the minimum s_θ^β is an implicit function of both θ and β. This observation led them to the Bengio-Scellier theorem:

Theorem 2.1: Bengio-Scellier theorem

The gradient of the objective function with respect to the network parameters θ is given by the formula:

$$\frac{\partial J}{\partial \theta}(\theta,x,d) = \lim_{\beta\to 0}\frac{1}{\beta}\left[\frac{\partial F}{\partial \theta}(s_\theta^\beta,\theta,\beta) - \frac{\partial F}{\partial \theta}(s_\theta^0,\theta,\beta=0)\right] \tag{3.11}$$

or, equivalently

$$\frac{\partial J}{\partial \theta}(\theta,x,d) = \frac{\partial C}{\partial \theta}(s^0_\theta,\theta,0) = \lim_{\beta\to 0}\frac{1}{\beta}\left[\frac{\partial E}{\partial \theta}(s^\beta_\theta,\theta,\beta) - \frac{\partial E}{\partial \theta}(s^0_\theta,\theta,0)\right] \tag{3.12}$$

The most interesting thing about this theorem is that it offers a direct way to compute the objective function gradient. This is important because it is possible to define a dynamics describing the weights update:

$$\frac{d\theta}{dt} = -\frac{\partial J}{\partial \theta}(\theta,x,d) \Rightarrow \Delta\theta = -\tilde{\eta}\frac{\partial J}{\partial \theta}(\theta,x,d) \tag{3.13}$$

where this could be interpreted as if the evolution of the θ values and of the s states occurs on two different time scales, the former much slower than the latter.
So far the energy function and all of the terms entering in the Bengio and Scellier theorem have been considered as referred to a single data. Nonetheless, when performing the update and while evaluating the performances of a network it is useful to have figures of merits evaluating the performances on the whole data-set. Hence, what said so far is directly generalized to a data-set containing N_d data by defining the energy and the objective function from the single data ones, in a linear way. Starting from the energy function, it evaluated at the points in the state space where the network converges

$$F(\{s^{(i)}\}_{i=1}^{N_d},\theta,\beta,X,\{d^{(i)}\}_{i=1}^{N_d}) = \frac{1}{N_d}\sum_{i=1}^{N_d} F(s^{(i)},\theta,\beta,x^{(i)}d^{(i)}) \tag{3.14}$$

where

$$F(s^{(i)};\theta,\beta,x^{(i)}) = \sum_{j=1}^{N_s}\int^{s_j^{(i)}} d\,\tau g'(\tau)\tau - \sum_{j,k=1}^{N_u} g(u_j^{(i)})W_{j,k}g(u_k^{(i)}) - \sum_{j=1}^{N_s} b_k g(s_k^{(i)}) + \\ + \frac{\beta}{2}\|g(y^{(i)}) - d^{(i)}\|_2^2$$

So that, the update of the synaptic weight W_{ij} performed on N_d data reads

$$\Delta W_{ij} = -\frac{\tilde{\eta}}{\beta N_d}\frac{\partial}{\partial W_{ij}}\sum_{k=1}^{N_d}(F(s^{(k)};\theta,\beta,x^{(k)}) - F(s^{(k)};\theta,0,x^{(k)})) = \\ = \eta\sum_{k=1}^{N_d}\left[g(u_i^{\beta,(k)})g(u_j^{\beta,(k)}) - g(u_i^{0,(k)})g(u_j^{0,(k)})\right]$$

Where the parameter $\eta \doteq \frac{\tilde{\eta}}{\beta N_d}$ has been introduced while the dependence of the equilibrium points $g(u_j^{\beta,(k)})$ on θ has been omitted for simplifying the notation. Similarly,

$$\Delta b_i = \eta\frac{\partial}{\partial b_i}\sum_{k=1}^{N_d}(F(s^{(k)};\theta,\beta,x^{(k)}) - F(s^{(k)};\theta,0,x^{(k)})) = \\ = \eta\sum_{k=1}^{N_d}\left[g(u_i^{\beta,(k)}) - g(u_i^{0,(k)})\right]$$

Finally yielding a three step algorithm called Equilibrium propagation. When performing the update after having considered each single data, the update would be:

1. Free phase: let the system evolve without introducing information externally $s_\theta^0 = \arg\min F(s, \theta, \beta = 0)$
2. Nudged phase: let the system evolve after having introduced a force attracting the system toward the target $s_\theta^\beta = \arg\min F(s, \theta, \beta)$
3. Update: $\theta \leftarrow \theta + \Delta\theta(s_\theta^0, s_\theta^\beta)$

This grasps the essential feature of the procedure while a more implementation-oriented description is provided in Algorithm 1 and in its training and test procedures can be found in Algorithm 2 and Algorithm 3. By referring to Algorithm 1, the data-set is first split into training and test sub-sets. Then, the whole data-set is cycled over for a number N_{epochs} of epochs. During each

Algorithm 1 Equilibrium Propagation

procedure EQPROP($X = \{x^{(i)}\}_{i=1}^{N_d}, Y = \{d^{(i)}\}_{i=1}^{N_d}$)
  $X_{train} = \{x^{(i)}\}_{i=1}^{N_{train}}, X_{test} = \{x^{(i)}\}_{i=1}^{N_{test}} \leftarrow X$
  $Y_{train} = \{d^{(i)}\}_{i=1}^{N_{train}}, Y_{test} = \{d^{(i)}\}_{i=1}^{N_{test}} \leftarrow Y$
  $\theta^{(0)} = (W^{(0)}, b^{(0)}) \leftarrow$ initialize
  $\tilde{t} \leftarrow 1$
  while $\tilde{t} \leq N_{epochs}$ **do** ▷ Repeat for N_{epochs} epochs
    $\theta^{(e)} \leftarrow \text{TRAIN}(X_{train}, Y_{train}, \theta^{(e-1)})$
    $[\text{acc}_{test}^{(e)}\ \text{acc}_{train}^{(e)}, \text{cost}_{test}^{(e)}\ \text{cost}_{train}^{(e)}] \leftarrow TEST(X_{train}, X_{test}, Y_{train}, Y_{test}, \theta^{(e)})$
    $\tilde{t} \leftarrow \tilde{t} + 1$

of them the model θ is updated using the training data and its targets. Afterwards, the test data are used to check the accuracy of the model on data not used during the training phase, so as to ensure the generality of the learned model.
Focusing on the training procedure, after an initialization of the parameters and of the state from which the evolution should start, the training data are analyzed one by one. For each of them the network evolves both freely and nudged and the update terms are modified. The model θ is not updated after having observed the whole data-set but rather after small sub-sets of it called mini batches, having a dimension called *mini batch size*. In this way, the N_d number of data on which the energy function and cost functions are minimized is associated to the minibatch. For what concerns the testing procedure of Algorithm 3 its main purpose is to determine the accuracy and the objective function. Ideally, the accuracy should increase while the cost decreases. Additionally, the result should be general, that is the reason why it is always done a comparison of the performances when presenting to the network data seen during the training phase and data never seen before.

3.2.3 Hardware implementation

One of the most promising features of this algorithm is its apparent high compatibility with an analog hardware implementation. The big success of Artificial Neural Networks comes from the Backpropagation algorithm, which is the workhorse of most Deep Learning algorithms. In particular, for what concerns Recurrent Neural Networks the most extensively used algorithm is Recurrent Backpropagation, developed by Almeida–Pineda [54]. The community has tried to

Algorithm 2 Training

procedure TRAIN($X_{train}, Y_{train}, \theta$)
  $\Delta W, \Delta b \leftarrow 0,0$
  $s \leftarrow$ random initialization
  while $k \leq N_{train}$ **do** ▷ Explore all the data in the training set
    $s_\theta^{0,(k)} \leftarrow argmin_s E(s;\theta,x^{(k)})$ ▷ free phase
    $s \leftarrow s_\theta^{0,(k)}$
    $s_\theta^{\beta,(k)} \leftarrow argmin_s F(s;\theta,\beta,x^{(k)},d^{(k)})$ ▷ nudged phase
    $s \leftarrow s_\theta^{\beta,(k)}$
    $\Delta W \leftarrow \Delta W - \eta[g(u_\theta^{\beta,(k)})g(u_\theta^{\beta,(k)})^T - g(u_\theta^{0,(k)})g(u_\theta^{0,(k)})^T]$
    $\Delta b \leftarrow \Delta b - \eta[g(u_\theta^{\beta,(k)}) - g(u_\theta^{0,(k)})]$
    if $mod(k, \text{batchsize}) = 0$ **then** ▷ Update
      $W \leftarrow W + \Delta W$
      $b \leftarrow b + \Delta b$
      $\Delta W, \Delta b \leftarrow 0,0$
    $k \leftarrow k+1$
  return $\theta = (W,b)$ ▷ trained network

Algorithm 3 Testing

procedure TEST($X_{train}, X_{test}, Y_{train}, Y_{test}, \theta$)
  $J_{test}, J_{train} \leftarrow 0,0$
  $acc_{test}, acc_{train} \leftarrow 0,0$
  while $k \leq N_{test}$ **do** ▷ Explore all the data in the test set
    $s \leftarrow$ random initialization
    $s_{\theta,train}^{0,(k)} \leftarrow argmin_s E(s;\theta,x_{train}^{(k)})$
    $s_{\theta,test}^{0,(k)} \leftarrow argmin_s E(s;\theta,x_{test}^{(k)})$
    $\text{acc}_{test} \leftarrow \text{acc}_{test} + (g(y_{\theta,test}^{0,(i)}) == d_{test}^{(i)})/N_{test}$
    $\text{acc}_{train} \leftarrow \text{acc}_{train} + (g(y_{\theta,train}^{0,(i)}) == d_{train}^{(i)})/N_{test}$
    $J_{test} \leftarrow J_{test} + \frac{1}{2}||g(s_{\theta,test}^{0,(k)}) - d_{test}^{(k)}||^2/N_{test}$
    $J_{train} \leftarrow J_{train} + \frac{1}{2}||g(s_{\theta,train}^{0,(k)}) - d_{train}^{(k)}||^2/N_{test}$
    $k \leftarrow k+1$
  return $\text{acc}_{test}, \text{acc}_{train}, J_{test}, J_{train}$ ▷ performances on the training and test data

design hardware models for training these networks from the early days of the theory. Starting from Hopfield and Tank [47, 48, 55], also Chua himself dedicated his knowledge on non-linear circuits to investigate the computational capabilities of brain-like networked circuits [56, 57]. This is mainly because if one managed to generate an analog network in which the membrane potential of each neuron is modelled with an RC parallel circuit and all the neurons are connected via proper circuital components, given an initial condition it could be possible to find the fixed point of the network just letting evolve the circuit toward its equilibrium. This is particularly evident if one considers the work done on the Co-content function and the Content function of

a network, which led the electronics community to reach the same conclusions of the neuro-dynamics people. Unfortunately, learning through Backpropagation automatically requires an increased amount of resources, since it is based on the explicit calculation of the gradients. To overcome this issue, Hertz et al. [58] started investigating approaches to estimate the gradient rather than exactly computing them. Bengio and Scellier presented their learning framework by explicitly referring to this approach. The learning rule they came up with uniquely requires the knowledge of the firing rate of the neurons, which could be learned after the nonlinear RC network has reached convergence. In addition to the seminal papers, many researchers who worked on EP highlighted that the local update rules defined by this algorithm could drastically reduce the complexity of the hardware required with the current alternative algorithms. It is particularly worth mentioning the work done on continual weights updates by Ernoult et al. [59], as well as some Spice analysis on memristive-based networks by Kendall et al. [60].

3.3 Asymmetric Versions of Equilibrium Propagation

The previously described algorithm is based on rigorous proofs of validity under mild conditions on the energy function. Nevertheless, the very same requirement that an energy function can be defined poses considerable concerns on the biological plausibility of the algorithm. As a matter of fact, the asymmetry in the synaptic connections is believed to be at the basis of many learning capabilities of the brain [61]. For this reason, in the follow up work made by Scellier et al. [5], the authors proposed an extension of the framework of Equilibrium Propagation to general dynamics.

3.3.1 Scellier Extension

The idea behind their proposal is that, when defining the evolution in the parameters space, it might be sufficient to approximately identify the correct direction, without the need to have the exact update at each evolution step. In addition to this, Bengio et al. [62] have verified that a simplified Hebbian update rule can properly reproduce experimental observations of STDP. If j is the pre-synaptic neuron and i is the post-synaptic neuron, the proposed simplified Hebbian update, in differential form is

$$dW_{ij} \propto g(s_j)ds_i \tag{3.15}$$

Furthermore, they noticed that the Additive model can be characterized by a non symmetric weight matrix. The evolution of the network can still be defined as the time evolution in a vector field, with the only difference that, the a-symmetry of the weight matrix makes it non trivial to define and energy function.

$$\frac{ds_i}{dt} = \mu(W,u) \tag{3.16}$$

Focusing on the Additive model, they noticed that the linear appearance of the firing rate in the vector field allows to say that $g(s_i) = \frac{\partial \mu_i}{\partial W_{ij}}$. And this led them to postulate the update rule

$$dW_{ij} \propto \frac{\partial \mu_j}{\partial W_{ij}} ds_i \tag{3.17}$$

that, in the Additive Model, can be written as

$$\begin{cases} \Delta W_{ij} = \eta g(s_j^0)(s_i^\beta - s_i^0) \\ \Delta b_i = \eta (s_i^\beta - s_i^0) \end{cases} \tag{3.18}$$

And they proved that this expression minimizes the objective function.

3.3.2 Alternative Extension

The Scellier extension of Equilibrium Propagation is potentially more general than the symmetric counterpart. However, the update rule was derived by means of a heuristic approach, thus leaving open an interesting branch of research devoted to formally prove the mechanisms behind its effectiveness or identifying new algorithmic solutions. This section addresses this open question by proposing a new asymmetric update rule.
The basic assumption is that there is a more general algorithm than EP, named a-EP, which in principle can admit asymmetrical updates of the weight matrix W. This algorithm is supposed to be EP-like, in the sense that the update is supposed to be calculated from the knowledge of the states reached at the convergence of the nudged phase (s_θ^β) and the free phase (s_θ^0).
As a starting hypothesis, let us initialize the parameters to a point in the parameter space in which the weight matrix is symmetric. In this way EP is rigorously defined and, in principle, we could choose to compute the update either using EP or using a-EP. Let us start from the update using EP. First, it is possible to expand the rate in the nudged equilibrium point

$$\begin{aligned} g(s_i^\beta) &= g(s_i^0) + \frac{\partial g(s_i^\beta)}{\partial s_i^\beta}\bigg|_{s_i^\beta = s_i^0} \frac{\partial s_i^\beta}{\partial \beta}\bigg|_{\beta=0} \beta + o(\beta) = \\ &= g(s_i^0) + g'(s_i^0) \frac{\partial s_i^\beta}{\partial \beta}\bigg|_{\beta=0} \beta + o(\beta) \end{aligned}$$

If this term is substituted into the EP update rule, the update becomes

$$\begin{aligned} \Delta W_{ij}^s &= \eta [g(s_i^\beta) g(s_j^\beta) - g(s_i^0) g(s_j^0)] = \\ &= \eta \left\{ \left[g(s_i^0) + g'(s_i^0) \frac{\partial s_i^\beta}{\partial \beta}\bigg|_{\beta=0} \beta \right] \left[g(s_j^0) + g'(s_j^0) \frac{\partial s_j^\beta}{\partial \beta}\bigg|_{\beta=0} \beta \right] + o(\beta) - g(s_i^0) g(s_j^0) \right\} = \\ &= \eta \left[g'(s_i^0) g(s_j^0) \frac{\partial s_i^\beta}{\partial \beta}\bigg|_{\beta=0} \beta + g'(s_j^0) g(s_i^0) \frac{\partial s_j^\beta}{\partial \beta}\bigg|_{\beta=0} \beta + o(\beta) \right] \end{aligned}$$

Since β is taken to be small in EP, the derivative of the s_j^β function can be approximated as an incremental ratio in β and the update becomes

$$\boxed{\Delta W_{ij}^s = \eta [g'(s_i^0) g(s_j^0)(s_i^\beta - s_i^0) + g'(s_j^0) g(s_i^0)(s_j^\beta - s_j^0)] + o(\beta)} \tag{3.19}$$

On the other hand, the symmetric update can also be written as

$$\Delta W_{ij}^s = \frac{W_{ij}^a + W_{ji}^a}{2} + \left(\Delta W_{ij}^s - \frac{W_{ij}^a + W_{ji}^a}{2} \right) \tag{3.20}$$

where the symmetrization of the a-symmetric update rule has been introduced. The update ΔW_{ij}^s is the optimal one when the matrices are constrained to be symmetric. On the other hand, the update ΔW_{ij}^a is the optimal one in general. Since both of them should be computed locally, given s_θ^β and s_θ^0, it is possible to assume that the distance of the symmetric update from the symmetrization of the exact update is a $o(\beta)$, i.e.

$$\boxed{\Delta W_{ij}^s = \frac{W_{ij}^a + W_{ji}^a}{2} + o(\beta)} \tag{3.21}$$

For all the cases in which this is true, it is possible to compare Eq.3.19 with Eq.3.21 and, by neglecting terms of the order of $o(\beta)$, one finally obtains

$$\Delta W_{ij}^a + \Delta W_{ji}^a = 2\eta[g'(s_i^0)g(s_j^0)(s_i^\beta - s_i^0) + g'(s_j^0)g(s_i^0)(s_j^\beta - s_j^0)] \tag{3.22}$$

In principle it would not be possible to associate term by term, however, if we consider Bengio's experimental finding on the simplified Hebbian update rule $dW_{ij} \propto g(s_j)ds_i$, it is possible to identify the asymmetric update with

$$\boxed{\Delta W_{ij}^a = \tilde{\eta} g'(s_i^0)g(s_j^0)(s_i^\beta - s_i^0)} \tag{3.23}$$

This update rule is extremely analogous to the one obtained by Scellier et al., the only difference being the first derivative of the post-synaptic rate. Basically, if the post-synaptic neuron saturates in the free phase, no update is to be done. Similarly, also the bias term can be reproduced with the same corrective factor as before

$$\Delta b_i = \eta[g(s_i^\beta) - g(s_i^0)] \simeq \eta g'(s_i^0)(s_i^\beta - s_i^0) \tag{3.24}$$

One might observe that, in this way, the advantage of EP over Backpropagation would be lost since, also in this case, the derivative of the activation function needs to be explicitly computed. Despite the truth of this statement, this also introduces a first valid reason for using the hard functions. In fact, in this work the activation functions employed are the hard sigmoid and the hard hyperbolic function, that are both defined as

$$g(s) \doteq \min(\sigma_\uparrow, \max(\sigma_\downarrow, s)) \tag{3.25}$$

with $\sigma_\uparrow = 1$ while $\sigma_\downarrow = -1$ for the hyperbolic tangent and $\sigma_\downarrow = 0$ for the hard sigmoid. This choice is primarily dictated by the fact that these functions can be implemented in a very compact way. Naturally, when implementing in analogue hardware it will not be possible to exactly implement the activation function, but rather some smoother version of it. Hence, the derivative is defined on the whole domain and is approximately equal to

$$g'(s) \simeq H(s - \sigma_\downarrow) - H(s - \sigma_\uparrow) \tag{3.26}$$

with H(s) the Heavyside step function

$$H(s) = \begin{cases} 0 & \text{if } s < 0 \\ 1 & \text{if } s > 0 \end{cases} \tag{3.27}$$

Consequently, the effect of the derivative term can be easily achieved by verifying whether the corresponding rate has saturated or not.
As a final remark, it is possible to observe that the newly introduced term $g'(s^0)$ might be interesting from a biological point of view, as it could be interpreted as encoding the habituation capability of the network. Habituation is the process by which the reaction to a stimulus decreases if that stimulus occurs for sufficiently long time. In our case, the response to the stimulus would be the weight update. If the stimulus at the input of neuron i is maintained at a level such that the rate saturates, an organism capable of learning will understand that no new information needs to be encoded and will stop modifying the synaptic connection. In this perspective, the term $g'(s_i^0)$ will allow to avoid to modify the weight whenever habituation is experienced.

3.4 Conclusions

This chapter introduces the three learning algorithms used during the numerical part. The starting point is Equilibrium Propagation, that is an energy based framework for training time continuous recurrent neural networks. Thanks to its local update rules, it seems to be extremely promising for analog implementations. In fact, this feature is expected to allow for an hardware implementation requiring a smaller number of components with respect to the ones required by other learning algorithms, such as Backpropagation. This chapter also introduces the asymmetric version proposed by Scellier et al. [5], highlighting the STDP plausible update rule. Finally, a modified version of the asymmetric update rule is proposed, connecting the STDP plausible update rule with the optimal learning performed by symmetric equilibrium propagation. This new algorithm allows to introduce in the description the concept of habituation, that is generally considered one of the main phenomena occurring in a neuron based system.

Chapter 4

Reconstruction

In this chapter, the use of the EP algorithm for the reconstruction problem is discussed. Reconstruction corresponds to restoring an image from a damaged version of it and many different recurrent neural networks can solve this problem. The first network considered is the Grossberg-Hopfield one, in which a single pixel is associated to each node. This network is particularly interesting since in some oversimplified cases the information learned by the algorithm can be directly interpreted. Afterward, the morphology of the network is progressively modified, so as to pass from the Grossberg-Hopfield network to the Bengio-Scellier one. For each architecture considered it is verified the correct functioning of the algorithm in optimizing the network for the solution of the Reconstruction problem.

4.1 Reconstruction

In the previous chapter it was discussed how Grossberg, while trying to model the brain functioning, defined a fully connected layer of neurons evolving according to the dynamics of the Additive Model. He started from psychological postulates and he wanted a network capable of learning a list. Hopfield described this very same behavior while rediscovering the emerging computational capability of this network. As said before, this property is called associative memory and in this section it is analyzed a practical example through the reconstruction problem. Following Zoppo et al. [63], let us consider ten images, or patterns, each one representing a different digit going from 0 to 9. The digits are represented as 8×8 binary matrices, i.e. each pixel is either a 0 or a 1 or also either a -1 or a 1, depending on the activation function used. In both cases, the saturation of the activation function is indicated via the notation $\sigma_i \doteq g(s_i)$. A schematic representation of the problem can be seen in Fig.4.1. During the training phase the network receives a corrupted image, where the corruption is achieved by flipping the pixels with a flip probability $p = 0.1$. Then, different networks are trained using EP, so as to create machines capable of restoring the correct image after a certain transitory. Three different network possibilities are here considered, introducing the input in different ways and structuring the information in a growing order of complexity.

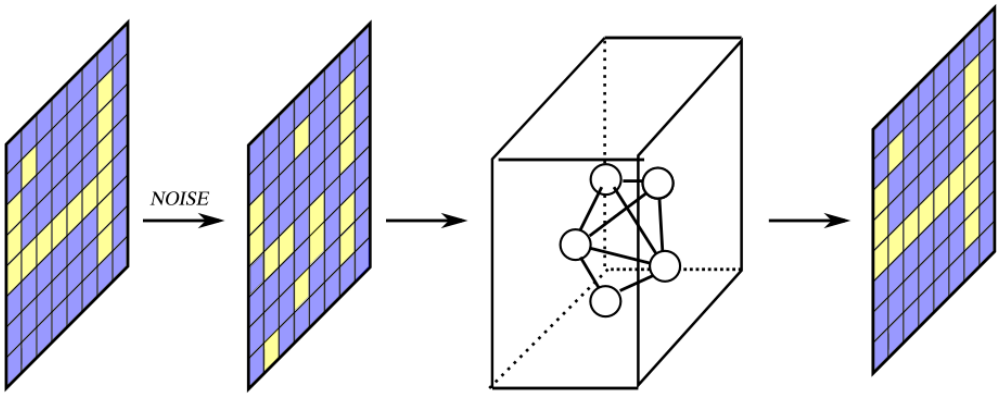


Figure 4.1: Schematic representation of the reconstruction problem. The correct image is corrupted by noise and is fed to a recurrent neural network. If the selected machine has learned the restoring operation it reconstructs in output the original version of the image.

4.2 Grossberg-Hopfield network

The first architecture considered consists of a single layer of $N_s = 64$ neurons fully connected to each others (see Fig.4.2). Each neuron evolves according to the following dynamics:

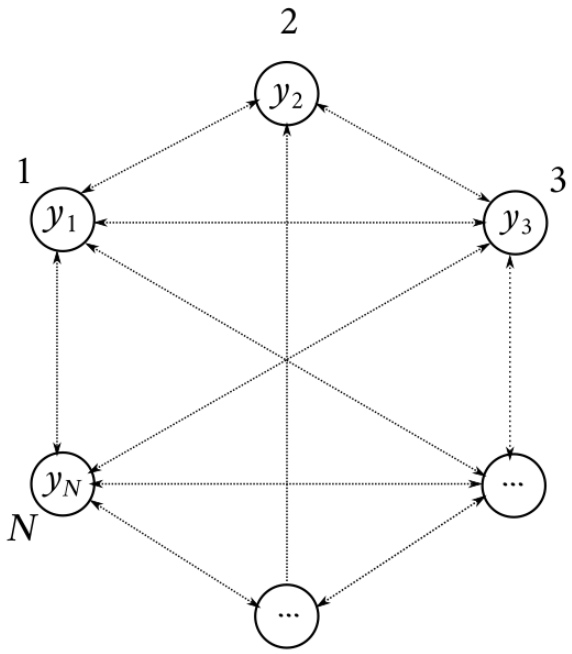


Figure 4.2: Grossber-Hopfield architecture. The network is fully connected and each node acts both as an input and as an output neuron.

$$\frac{d}{dt}s_i = -s_i + \sum_{j=1}^{N_s} W_{ij} g_j(s_j) + b_i + I_i \tag{4.1}$$

corresponding to the Cohen-Grossberg generalized Lyapunov function

$$E = \sum_{j=1}^{N_s} \int^{s_j} \tau g'(\tau)\, d\tau - \frac{1}{2} \sum_{i,j=1}^{N_s} g(s_i) W_{ij} g(s_j) - \sum_{i=1}^{N_s} b_i g(s_i) - \sum_{i=1}^{N_s} I_i g(s_i) \tag{4.2}$$

This is almost exactly the Cohen-Grossberg network, with the slight difference that an additional degree of freedom was added through the bias term. The input I_i corresponds to a partial information on the input pattern. Focusing on the inference phase, the system will start in a random position of the state space. During all the evolution, each variable will be subject to the force $I_i - s_i$, driving the system toward the input I_i. Nonetheless, this input does not contain the perfect information. Therefore, the remaining terms in Eq.4.1 can be learned so as to correct potential mistakes in the input. The bias b_i is an additional uniform external field while the sum over all the other neurons is needed to take into account the inter-dependency existing between two any variables throughout the dynamics. In the next section this statement is developed for the oversimplified case of learning one pattern.

4.2.1 One Pattern

Let us start from creating a machine capable of reconstructing one single image. The energy function quantifies the likelihood for a configuration to exist. According to what discussed while introducing energy-based learning, it is possible to build the network just from intuitive arguments, without the need of using an algorithm for the task. In fact, if the pixels i and j are always switched on together, then W_{ij} should be positive, so that $g(s_i)W_{ij}g(s_j)$ lowers the energy when the two pixels have the same value. Similarly, if the two pixels are never to be switched on together the synaptic weight connecting them should be negative, so as to raise the energy every time the two satisfy the correct relationship. In this sense the synaptic weights express the correlation or anti-correlation with their sign, and the importance of the effect on the dynamics with their modulus. Since the dynamics brings the system toward the minimum of the energy, if the external field $I_i - s_i$ would lead to recreate the damaged image at the output, the network corrects the trajectory in two ways. First of all the bias term can introduce a constant force, strengthening the attraction of the constant term $I_i + b_i - s_i$ toward the correct point in space. Therefore, if the pattern to be learned is the binary image $\sigma = [\sigma_1, \dots \sigma_{N_s}]^t$, with $\sigma_i \in \{+1, -1\}$, we should expect that $b_i \propto \sigma_i$. In addition to this, the synaptic weights inform about the possible presence of non-optimal mutual values of the pixels. Therefore, in some sense their contribution to the energy minimization is ultimately corresponding to a correlation. In the case of only one pattern, only one configuration is desired to be the correct one. This is to say that the goal is to have a unique attracting region M in the state space, corresponding to

$$s \in \mathscr{S} : \;\; g(s_i) = \sigma_i \;\; i = 1, \dots, N_s \tag{4.3}$$

Indeed, M is an entire region in the states space. Since σ_i corresponds to the maximum or minimum value the activation function can reach, any s_i above or below the saturation will correspond to a valid state variable. Alternatively, if the system is considered as evolving in the rate space [1], M must be a single limiting point. The region M must be a minimum of the energy function and the sign should satisfy the previously mentioned dependencies on the mutual

[1] A space in which the space variables are the states filtered by the rates.

values of the target state variables. Therefore, a good choice is expected to be $W_{ij} \propto \sigma_i \sigma_j$. In fact, for $\sigma_i \pm 1$ this expression exactly reproduces the sign in the correlation information discussed before. The proportionality factor must be a positive term, allowing to introduce a proper balance between the various forces in the dynamical system. The discussion naturally extends to the case in which the lower limit of the activation function is $\sigma_{\downarrow} = 0$. This corresponds to the Hebbian rule and the Hopfield paper [47] presents a more in depth discussion of its applicability together with its generalization to more than one pattern. The goal of this analysis is to give an intuition on the nature of the information learned by EP, as this will be useful in the following chapters. Therefore, as a proof of concept, a Grossberg-Hopfield network was trained for the reconstruction of the digit 1. This can be observed in Fig.4.3. On the top left it is presented the pattern. In the 8×8 matrix only the pixels $4 + k8$ with $k = 0, \ldots, 7$ are 1 while all of the others are -1. The network was trained for 20 epochs, with the learning rate fixed to $\eta = 0.001$

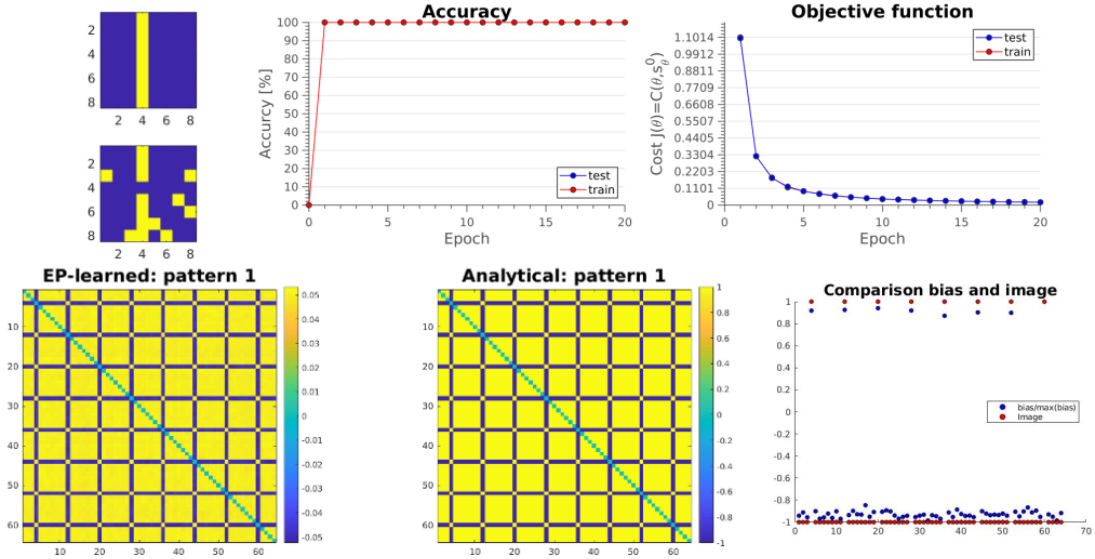


Figure 4.3: *Top-left*: Pattern and damaged version; *Top-right*: Evolution of the accuracy and of the objective function at the different epochs of the training; *Bottom-Left*: comparison of the weight matrix learned by EP with the Hebbian rule; *Bottom-Right*: comparison of the bias vector learned by EP with the target pattern. The network is defined with the hard-hyperbolic tangent as the activation function.

and the clamping factor to $\beta = 0.1$. In Fig.4.3 the bias learned by EP was flattened and it was normalized so as to force its maximum value to be 1. From a direct comparison with the target pattern it is clear that, in this oversimplified problem, the network has learned that the optimal constant field to be encoded is the constant one driving the system toward the target M-region. Finally, on the bottom left the weight matrix learned by EP is compared with the matrix defined as $W_{ij} = \sigma_i \sigma_j$. More precisely the diagonal of this matrix was removed since self-correlation are not taken into consideration. From the picture, it is clear that the difference is only a scaling factor. One would expect the same parameters also when considering an activation function saturating to 0 and 1, the only difference being that the weight matrix is now expected to satisfy $W_{ij} \propto (2\sigma_i - 1) \cdot (2\sigma_j - 1)$. The training was repeated using the hard-sigmoid and the results can be observed in Fig.4.4. The features learned are clearly related to the ones learned for the previous activation function. The dark pixel are the negative ones, corresponding to the synaptic

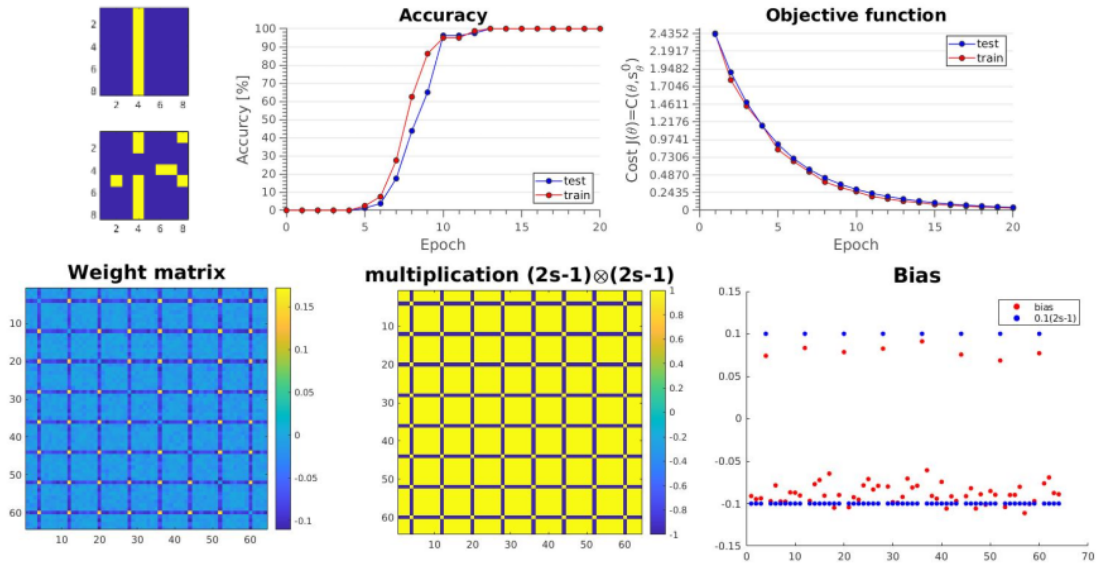


Figure 4.4: *Top-left*: Pattern and damaged version; *Top-right*: Evolution of the accuracy and of the objective function at the different epochs of the training; *Bottom-Left*: comparison of the weight matrix learned by EP with the Hebbian rule; *Bottom-Right*: comparison of the bias vector learned by EP with the target pattern. The network is defined with the hard-sigmoid as the activation function.

weights connecting switched ON pixels with switched OFF pixels. Moreover, the synaptic weights connecting pixels that should be switched ON are always positive thanks to the positive correlation between them. The only difference with respect to the previous case is that the weights connecting pixels that should be OFF together are set to zero. The reason why this might happen is that, in the parameters space, the energy function defined as $W_{ij} = 0$ for $\sigma_i = \sigma_j = 0$ has the same value as the Hebbian one when evaluated in the target region M. Apparently, the stronger information is in the anti-correlation of the ON pixels with the OFF pixels and in the correlation of the ON pixels. Since this information is sufficient for the network to learn, the weights that should correlate OFF target pixels can be set to zero. For the aim of completeness in Fig.4.5 are reported the weight matrices obtained for the other patterns when using the hard sigmoid. All of them present the well defined information content previously highlighted, as well as the relaxation in the information encoding related to the pixels that are OFF in the target.

4.2.2 Solution with the three versions of the algorithm

In this section the Grossberg-Hopfield architecture is used to solve the Reconstruction problem on the whole data-set of ten images. This complicates the interpretation, but a qualitative savor of the information learned by the machine is still possible. This time there should not be a unique equilibrium region but rather one for each pattern p. As an extension of the previous discussion, the region in the state space where the network converges when it is presented with a corrupted version of the corresponding input $I^{(p)}$ is named M_p. For what concerns the bias, also in this case the network can use it for encoding a reference constant field in the states space. For this reason, since more than one pattern must be learned and they all have equal importance, the network should learn as an optimal bias term the vector related to the baricenter of the M_p

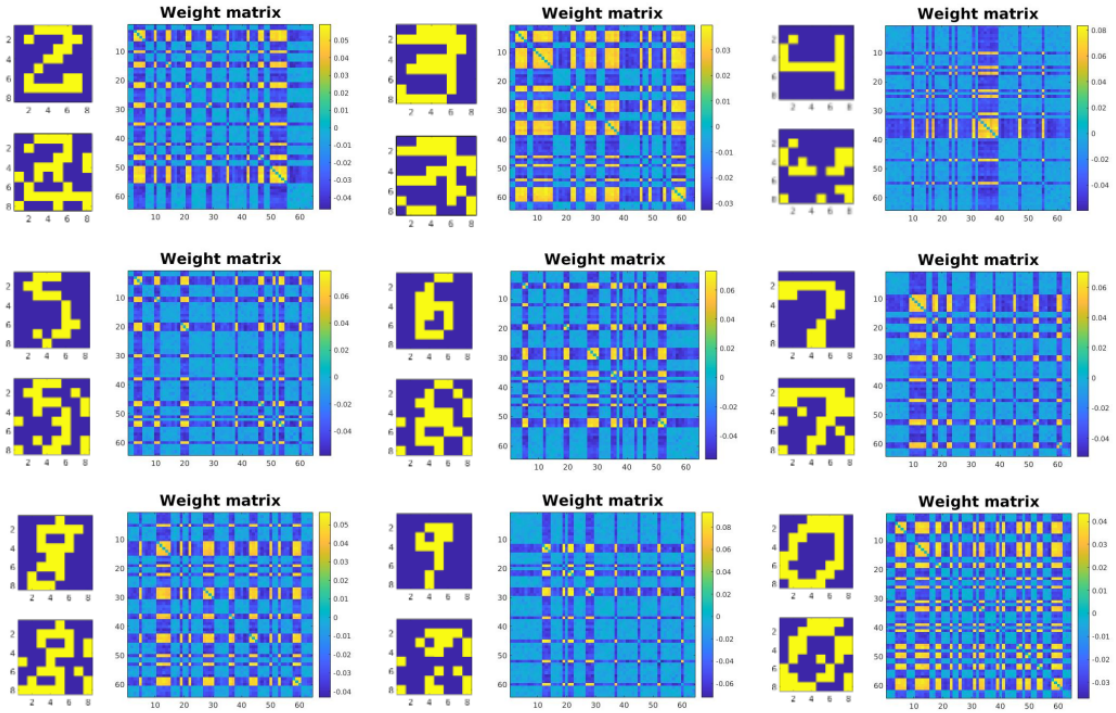


Figure 4.5: Weight matrix of the digits in the data-set when training the network with the EP algorithm on the individual patterns.

regions. Therefore, this time $b_i \propto \sum_{p=1}^{10} \sigma_i^{(p)}$. Moving to the weight matrix, it will also be shared in between the patterns. As a logic extension of the one pattern case, it must therefore encode a weighted information on the correlation existing among the pixels. In his work Hopfield [47] proposed as a prescription the weighted sum of the weight matrices for the single patterns, i.e. the generalized Hebbian rule. This is however not exact, particularly in the case here considered, that differs from the one he analysed in the fact that the Additive Model introduces an additional bias term and part of the information can be encoded in it. The weight matrix is therefore a nontrivial expression of the correlation existing among the pixels and an algorithm is needed for finding it. Also in this case the role of each input $I^{(p)}$ is to perturb the velocity field shared by the patterns so as to direct it toward the M_p-region. The information in $I^{(p)}$ is corrupted and the network will use the information it has learned about the general features of the data-set to adjust any pixel not likely to be correct.
Throughout the work the main focus is on the symmetric version of the EP algorithm. Therefore, the starting point was to train the network with EP, fixing the learning rate to $\eta = 0.005$ and the clamping factor to $\beta = 0.25$. The training was performed using $N_{train} = 1600$ corrupted images and tested against $N_{test} = 400$ newly generated images. The number of epochs chosen was 20 and the mini-batch size was set to 1. The activation function considered is the hard-sigmoid. With this choice of parameters, the achievable train accuracy is of 99.5% and the test accuracy is of 99.75%. The objective function of the final model evaluated on the training data is 0.012943, to be compared with 0.025247 on the test data. This shows that EP allows to solve the reconstruction problem when the whole process is performed in purely numerical way. Additionally, the network was trained with the hard hyperbolic tangent. Since only slight modifications of the hyper-parameters were found to be necessary, their value and the resulting

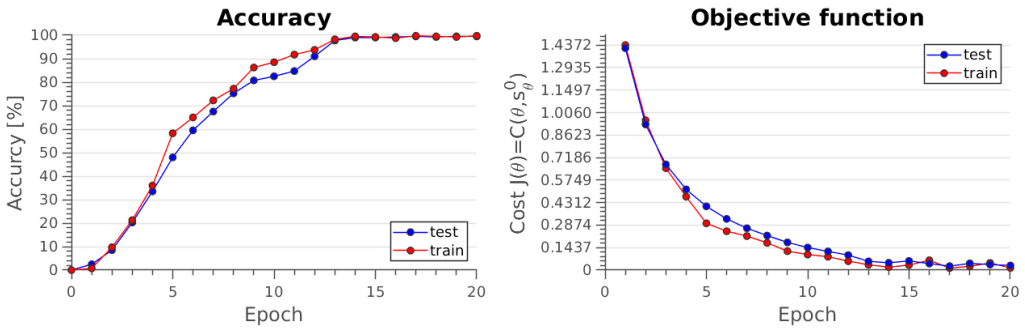


Figure 4.6: Symmetric EP: Accuracy and Cost as a function of the training epochs.

performances are collected in Tab.4.1. The plots of the accuracy and the cost function are not reported because they are qualitatively equivalent to the ones obtained for the symmetric version of the algorithm.
Imposing the symmetry on the weight matrix is extremely advantageous from a theoretical point of view, since it allows to define an energy function and to rigorously derive the update of the EP algorithm. However, this symmetry constraint imposes a considerable reduction of the parameters space and might also be not physically reasonable, considering the intrinsic asymmetry of the brain processes. Therefore, the asymmetric versions of EP was tested on the problem considered. The hyper-parameters chosen and the main figures of merit for the performances are reported in Tab.4.1 for both the Scelier version and for the one proposed in this work. This table should not be considered as a comparison since no fine hyper-parameters optimization was performed. It is just to be taken as a witness that all of the three algorithms can be equally used for the solution of the reconstruction problem. It is also interesting to observe

algorithm	function	η	β	ACC_{train}	ACC_{test}	J_{train}	J_{test}
EP	sigmoid	0.005	0.25	0.9950	0.9975	0.033977	0.02916
EP	tanh	0.001	0.4	0.9975	0.995	0.022184	0.038746
a-EP	tanh	0.002	0.4	0.995	0.9925	0.10254	0.099604
a-EP*	tanh	0.002	0.4	0.9975	1	0.080942	0.073572

Table 4.1: Parameters and performances of the three algorithms. EP is the symmetric version, a-EP the asymmetric version proposed by Scellier and a-EP* the modified asymmetric version we defined. All the activation functions are the hard version of the one indiated.

the weight matrices learned by the different algorithms, that have been reported in Fig.4.7. From a qualitative standpoint, also in this case strong similarities can be observed in between the information learned with the symmetric EP for different activation functions. Moreover, the shades of the hard sigmoid weight matrix presents an higher quantity of near zero values, in accordance with the information relaxation discussed for the one-pattern case. Additionally, a strong similarity of the weight matrices obtained with the asymmetric versions of the algorithm as well as with the ones obtained with the symmetric version. The distances in norm 2 of these four matrices were computed and are reported in Tab.4.2. More precisely, the distances considered were primarily the ones between the matrices obtained with the four algorithms and with a uniformly generated matrix with weights in the interval [-1,1]. The random matrix is needed to give a reference, i.e. a completely non-informative object. In order to make a

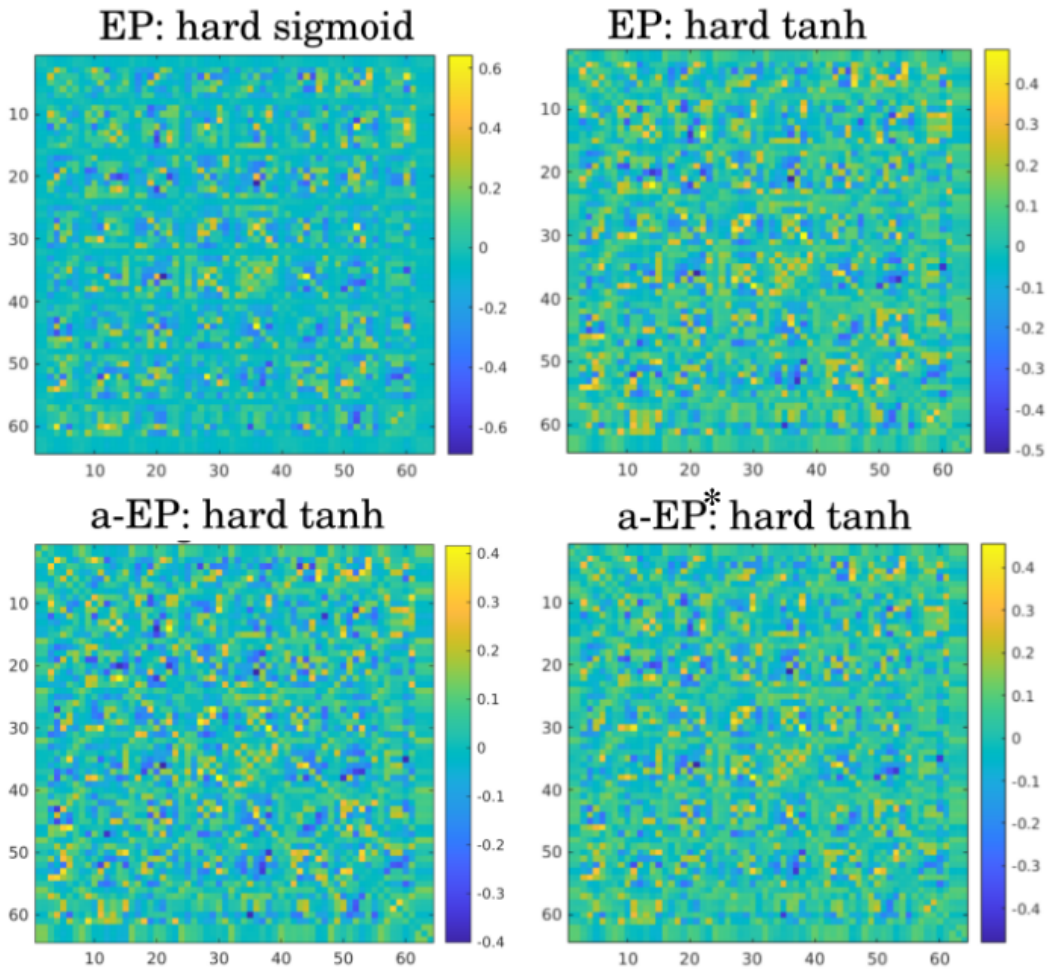


Figure 4.7: Comparison of the weight matrices obtained with the three algorithms: *Top Left*: symmetric EP with the hard sigmoid; *Top Right*: symmetric EP with the hard hyperbolic tangent; *Bottom Left*: Scellier version of the a-symmetric EP; *Bottom Right*: Proposed a-symmetric EP.

proper comparison all of the matrices were normalized dividing them by their maximum entry in absolute value. This is interesting because it suggests that there might be an absolute minimum

	random	EP sigmoid	EP tanh	a-EP	a-EP*
random	0	9.599	9.875	10.056	9.786
EP sigmoid	9.599	0	7.217	7.896	7.122
EP tanh	9.875	7.217	0	1.689	0.904
a-EP	10.056	7.896	1.689	0	1.559
a-EP*	9.786	7.122	0.904	1.559	0

Table 4.2: Distances in norm-2 of five matrices: the entry (i, j) of the table is the difference in norm 2 of the i-th matrix with the j-th matrix. The first matrix is a uniformly distributed random matrix while the others are the parameters optimized using equilibrium propagation. The second matrix is obtained with symmetric EP and using the hard sigmoid activation function while the third uses the hard hyperbolic tangent. The fourth matrix is obtained with the Scellier extension while the last one is the extension proposed in this work.

in the parameters-space, corresponding to the optimal weight matrix. Moreover, this optimum should be symmetric since even when we use a-symmetric updates the parameters converge to a symmetric matrix. The symmetry of the weights when training the network with a-EP has already been observed in other networks [5]. In this case the reason for which this happen can be understood from the discussion on the information content of the parametric model. In fact, any two pixels enter symmetrically in the mutual interaction. The correlation of the node i with the node j must be equal to the correlation of the node j with the node i. Therefore, due to the symmetric nature of the Reconstruction problem it is reasonable that the exact weight matrix should be symmetric and that an a-symmetric algorithm, despite the higher freedom of its dynamics in the parameter space will converge to a symmetric configuration.

4.3 Multilayer with interconnections

The network described in the previous section can be considered as a special case of another architecture, corresponding to the one originally proposed by Bengio and Scellier. In order to see this, let us notice that, for both the hard sigmoid and the hard tanh, it is safe to rename the inputs as $I_i = x_i$ and to substitute it with $x_i \mapsto g(x_i)$. As a matter of fact, in the hard functions there is absolutely no difference. If this is done, it is possible to associate to each input a node in the network, that does not vary with the state nodes since it is clamped. In this way the nodes of the architecture read $u = [x; s]^t$, with $x \in \mathbb{R}^{N_i}$ and $s \in \mathbb{R}^{N_s}$. In these terms, the Grossberg-Hopfield architecture is such that $N_s = N_i$ and $N_u = N_i + N_s = 2N_s$. In order to go further the weight matrix must be re-defined so as to account for the newly defined neurons. To do so, the old matrix $W^{(s)} \in \mathbb{R}^{N_s \times N_s}$ is replaced by a new matrix $W \in \mathbb{R}^{2N_s \times 2N_s}$, defined in such a way that the only interconnection existing is the one from the input node $u_k = x_k$ to the associated state node $s_k = u_{N_i+k}$. In mathematical terms, this corresponds to define the sub-matrices connecting the state nodes with the input nodes as

$$W_{km} = W_{mk} = \delta_{m,N_i+k} + \delta_{N_i+k,m} \tag{4.4}$$

where only one delta function can actually appear in the expression. Moreover, the sub-matrix connecting the input nodes is the zero matrix. Overall, for $i = 1, \dots, N_s$:

$$\begin{aligned}
\frac{d}{dt} s_i &= -s_i + \sum_{j=1}^{N_s} W_{ij}^{(s)} g(s_j) + b_i + I_i \\
\frac{d}{dt} u_{N_i+i} &= -u_{N_i+i} + \sum_{j=N_i+1}^{N_u=2N_s} W_{N_i+i,j} g(u_j) + b_{N_i+i} + g(x_i) \\
\frac{d}{dt} u_k &= -u_k + \sum_{j=N_i+1}^{N_u=2N_s} W_{kj} g(u_j) + b_k + \sum_{j=1}^{N_i} g(x_j) \delta_{k,N_i+j} \\
\frac{d}{dt} u_k &= -u_k + \sum_{j=N_i+1}^{N_u=2N_s} W_{kj} g(u_j) + b_k + \sum_{j=1}^{N_i} W_{kj} g(x_j) \\
\frac{d}{dt} u_k &= -u_k + \sum_{j=1}^{N_u} W_{kj} g(u_j) + b_k
\end{aligned}$$

Therefore, the inputs can be equivalently thought of as a layer of neurons, each of which is connected only with one other node in the output layer. In order to increase the number of degree

of freedoms, the input nodes can be allowed to connect also to the other output neurons, as shown in Fig.4.8. Using EP, the weights connecting the clamped input nodes to the output neurons can be learned authomatically, in order to find the optimal redistribution of the information to the next layer.

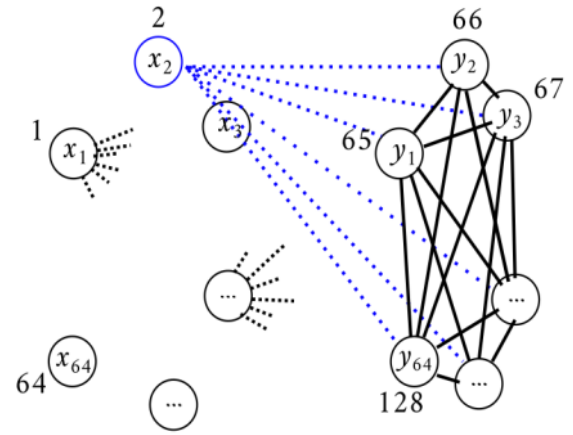


Figure 4.8: Generalization of the Grossberg-Hopfield network to the Bengio-Scellier one.

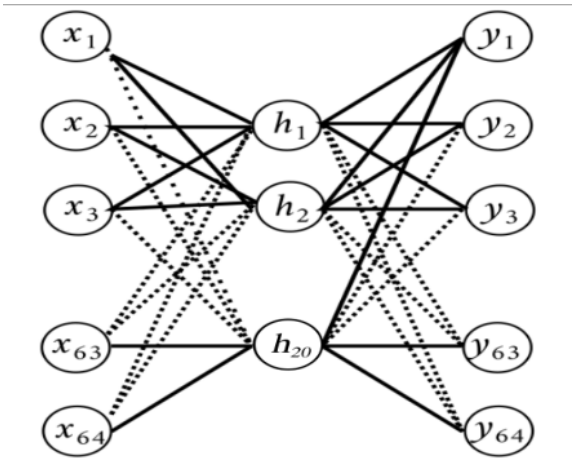


Figure 4.9: Bengio-Scellier network used as a denoising autoencoder.

The network was trained on 1600 data and the learned model was tested on 200 data. From the hand-tuning optimization the selected hyper-parameters were $\eta = 0.01$ and $\beta = 0.5$. A batch size of 20 was selected and the network was trained for 10 epochs. In Fig.4.10 it is possible to observe the training history through the epochs. The objective function on the train data reached $J_{train} = 0.00080329$ and on the test data $J_{test} = 0.0032227$. For what concerns the accuracy on both the data the network reached 100% accuracy. This comes at the cost of introducing more resources but it could be thought of as a natural way of extending the memory in case the Grossberg-Hopfield alone had not a storage capacity sufficiently high for the desired number of patterns.

4.4 Recurrent Denoising AutoEncoder

Once generalized the Grossberg-Hopfield network with the Bengio-Scellier one, it is now possible to do one step further in terms of complexity. While discussing Equilibrium Propagation it was pointed out that the most general network may be further enlarged by introducing hidden neurons. If properly defined, the new networks that can be designed would naturally increase the computational potential. In the most trivial way this is achieved by projecting the input nodes in a new space with higher dimensions. However, also a reduction of the dimension could be useful, as discussed in this section. The addition of a layer in between the input layer and the output layer defined before generates a network that can be identified with a recurrent denoising autoencoder (DAE).
A DAE is a low pass filter capable of removing high frequency noise from an image. Neural networks have been proved to be highly effective for the implementation of such machines thanks to their capability of naturally creating bottlenecks of information. In Fig.4.11 it is reported the basic idea behind this concept. The architecture is made of two operational blocks: an encoder and a decoder. The Encoder maps a vector field into a vector field of reduced dimension, called

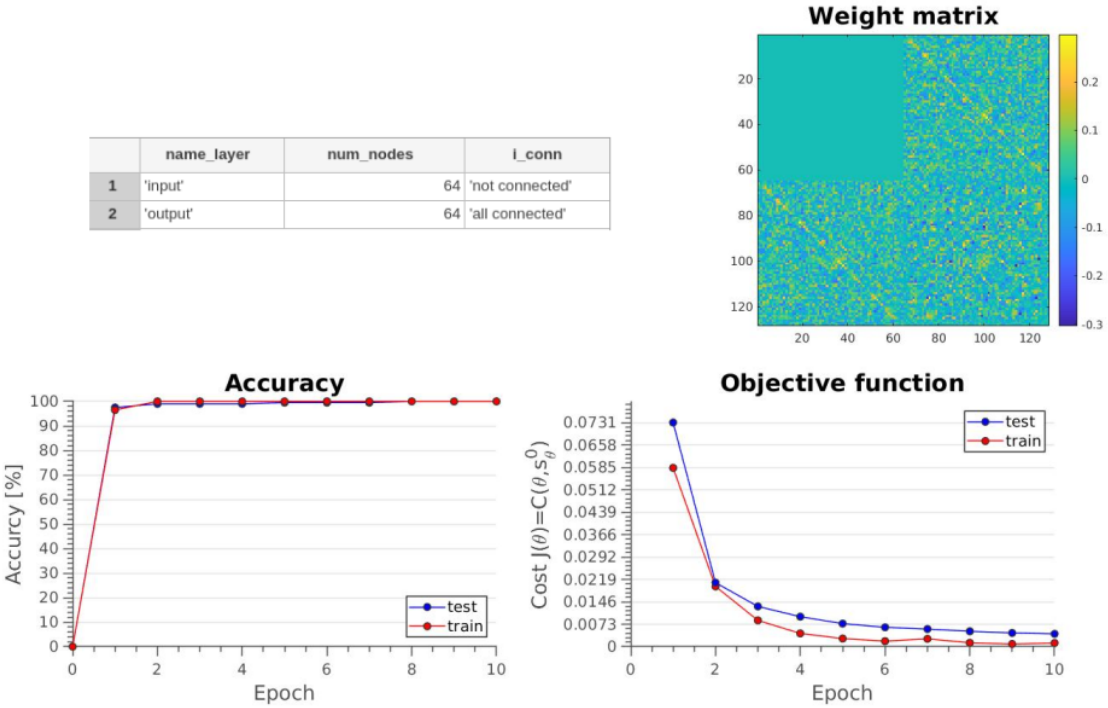


Figure 4.10: Reconstruction using the input as nodes: *Top Left*: `MatLAB` table summarizing the architecture defined; *Top Right*: Weight matrix learned by EP; *Bottom*: history of the performances during training.

latent space. Then, a decoder takes the compressed information and restores the original data. This is a natural low pass filter since the latent space constitutes a bottleneck of information. Not all the information presented in the input can be transferred integrally to the output and the structure needs to encode some structural constraints present in the data it has to handle, so as to overcome the architectural problem. Therefore, forced by the impossibility to take into account all possible shades, only the core information of the data is learned, and the noise is ignored. A layered structure was defined, having a 20 dimensional latent space represented in Fig.4.9. The network was trained on 1600 training data and 400 test data for 10 epochs. This time the batch size was set to 1 while the parameters optimized on the previous network ($\eta = 0.01$, $\beta = 0.5$) turned out to yield satisfactory performances: the accuracy of the best model was 99.25% while the objective function reached $J_{train} = 0.045468$ and $J_{test} = 0.053961$.

4.5 Conclusions

This chapter is devoted to discussing the applicability of Equilibrium Propagation for the solution of the Reconstruction problem. The starting point is the Grossberg-Hopfield network, corresponding to a single layer fully connected RNN. The nature of the information learned by the algorithm is discussed in the simple case in which the network has to learn only one pattern. In particular, the bias vector can be put in one-to-one correspondence with the target vector and the weight matrix is shown to store the information on the correlation existing in between the pixels. Then, the analysis focuses on the ten patterns case, testing the three algorithms introduced in the previous chapter and arguing the possibility of the existence of an optimal weight value in

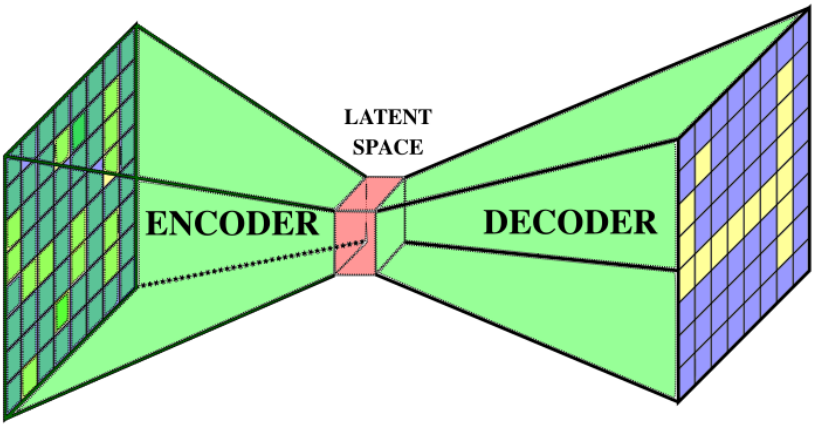
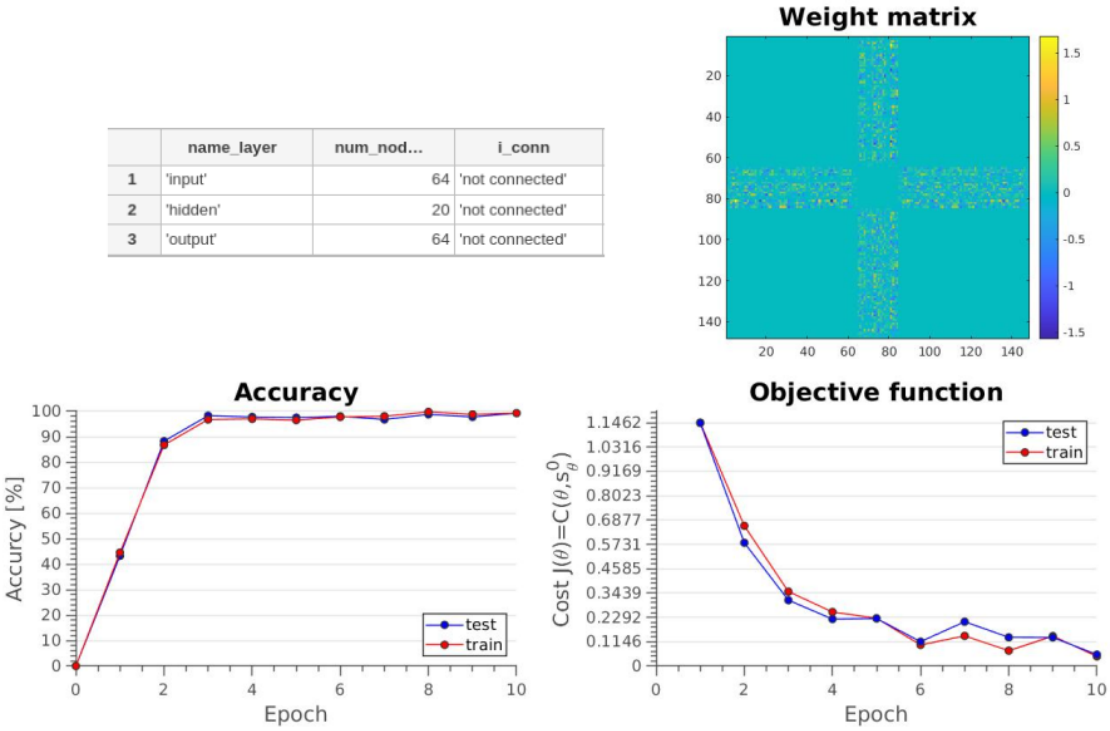


Figure 4.11: Schematic representation of the main blocks constituting a DAE. The input is encoded in a latent space and then decoded to the output space.



	name_layer	num_nod...	i_conn
1	'input'	64	'not connected'
2	'hidden'	20	'not connected'
3	'output'	64	'not connected'

Figure 4.12: Reconstruction using the DAE architecture: *Top Left*: `MatLAB` table summarizing the architecture defined; *Top Right*: Weight matrix learned by EP; *Bottom*: history of the performances during training.

the symmetric sub-space of the parameters space. Afterwards, the Grossberg-Hopfield network is proved to be a particular case of the Bengio-Scellier one. As a first example, it is considered a network in which each input is shared by all the state nodes. Finally, a recurrent Denoising autoencoder is proposed and the Reconstruction problem is solved using it. This last architecture is particularly interesting since it corresponds to a layered structure. Layered structures are

naturally modular and this is particularly practical in the perspective of designing configurable AI chip.

Chapter 5

Classification

The classification task is one of the most common problems in machine learning. Given a data-set containing objects from a certain number of classes, the model is trained to recognize the class to which an input data belongs to. In this chapter, classification problems of increasing complexity are considered. The starting point is the regression of the AND function, which is essentially a linear binary classification problem. Later, it is discussed the XOR regression, a non-linear binary classification problem. The last problem considered is digit classification. In this way it is possible to verify the success of the recurrent neural network models when combined with the Equilibrium Propagation algorithm for solving the classification problem. Finally, a Feature Engineering approach is discussed. This is inspired by the nature of the information encoded by the Additive model and allows to drastically reduce the number of parameters needed.

5.1 Basics of classification with recurrent neural networks

Neural networks are extremely successful for solving classification tasks. When considering recurrent neural networks, this must be understood in terms of dynamical systems. Given a dataset X containing images belonging to N_o classes, a neural network used for classification will generally have N_o output nodes, connected to the N_i output nodes through an arbitrarily complex networked parametric model. Once trained, each output node n corresponds to a parametric function $\hat{f}_n^\theta$ of the input. Since the model used is the Grossberg's Additive Model, this corresponds to the value of the corresponding state variable at convergence, after the application of the non-linearity.

$$\hat{f}_n^\theta(x) = g(y_n(t \to \infty)) = g\left(\sum_{j=1}^{N_u} W_{\tilde{n}j} g(u_j) + b_{\tilde{n}}\right) \quad \tilde{n} = N_i + N_h + n \tag{5.1}$$

Due to the fact that the AM is a rate model, the activation function will saturate at both limits:

$$\begin{cases} \lim_{s\to\infty} g(s) = \sigma_\uparrow \\ \lim_{s\to-\infty} g(s) = \sigma_\downarrow \end{cases} \tag{5.2}$$

Where $\sigma_\uparrow = 1$ and $\sigma_\downarrow = -1$ for the hyperbolic tangent while $\sigma_\downarrow = 0$ for the sigmoid and the hard-sigmoid. Therefore, training the network for solving the classification task corresponds to identify each one of the N_o classes with an output node and to find the parameters θ such that given an input belonging to the k-th class, $\hat{f}_k^\theta(u)$ will be the highest value at convergence.

5.2 Logic functions

Let us start from the problem of learning logic functions, that is a binary classification. The interesting thing of these networks is that their simplicity makes it possible not only to theoretically determine the minimally complex architecture capable of solving them, but also to visualize the dynamics in the state space. First of all it is necessary to associate $\sigma_\downarrow$ to the false boolean variable and $\sigma_\uparrow$ to the true one. In the following, two of the main logic functions are considered: the AND and the XOR.

5.2.1 Linearly separable classification problem

Given two logic values x_1 and x_2, the AND function evaluated on them is defined as follows:

$$x_1 \wedge x_2 = \begin{cases} \sigma_\uparrow \text{ iff } x_1 = \sigma_\uparrow \text{ and } x_2 = \sigma_\uparrow \\ \sigma_\downarrow \text{ otherwise} \end{cases} \tag{5.3}$$

This can be equivalently expressed with the truth table Tab.5.1. Figure 5.1 shows that, in the

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Table 5.1: truth table of the AND function.

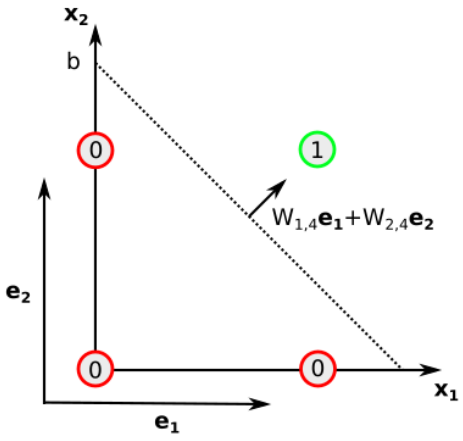


Figure 5.1: Graphical representation of the AND classification problem.

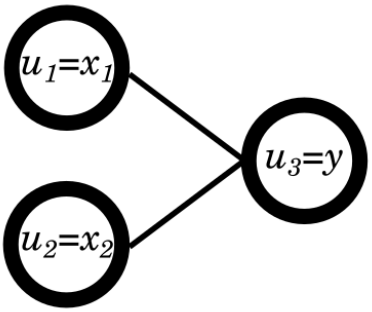


Figure 5.2: Structure of the AND architecture.

(x_1, x_2) space, the two truth classes can be separated by means of a line. Due to this peculiar behavior, this is called a *linearly separable* problem and a sigle neuron, namely the *perceptron* parametric function, can be fit to solve this task

$$\hat{f}^\theta = g\Big(W_{31}x_1 + W_{32}x_2 + b\Big) \tag{5.4}$$

In fact, if one defines the vector $\hat{u} = [W_{31}, W_{32}]^t$, it can be chosen as the vector normal to the line needed to separate the two classes. If the scalar product of the weight matrix with an input vector is performed (Wx), the result is a scalar that will be positive only if x is a point on the right of the line parallel to the separation line in Fig.5.1, and intercepting the origin. Instead, the additional bias term b allows to find the correct separating line. In this way, the term $Wx + b$ will be grater than a threshold value in one class and lower than the threshold in the other. If this threshold value corresponds to the mean image of the activation function, it is easy to see how the value at the output identifies the correct class. In Fig.5.3 it is reported the data used for training the

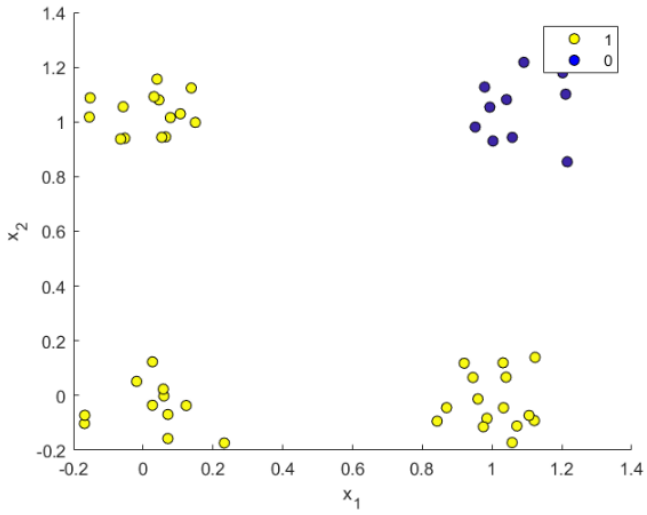


Figure 5.3: Data-set used for the AND problem.

network. As it is commonly done, some random noise has been added to make the problem closer to something that would happen in a real logic gate. The selected activation function was the *hard sigmoid* and the network was trained using $N_{train} = 100$ data for the training and $N_{test} = 80$ data for the test. The hyper-parameters were optimized to $\eta = 0.5$ and $\beta = 0.1$ and the network was trained for $N_{epochs} = 10$ epochs, checking the accuracy at the end of each epoch. The batch-size was fixed to 1 and the weights were initialized using the standard initialization technique [64]. The behavior of the network through the training process can be observed in Fig.5.4.

Let us now observe some figures of merit of the dynamical system. As expected, the parameters

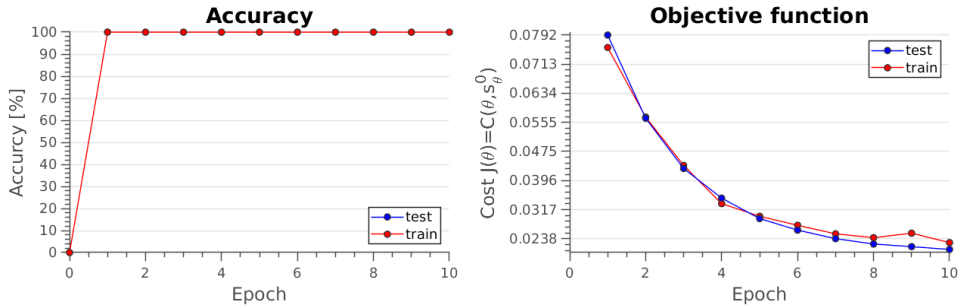


Figure 5.4: AND history

learned were $W_{13} = W_{31} = 0.4804$ and $W_{23} = W_{32} = 0.4686$ while the bias learned is $b =$

-0.2528. This result can be interpreted in two ways. The first one is in terms of the final parametric model. As expected, the vector normal to the separating line is parallel to the diagonal to the first quadrant. For what concerns the bias, it identifies a separating line crossing the diagonal of the first quadrant at a point whose distance from the origin is 0.25. Since the average value of the hard sigmoid is 0.5, the hard-sigmoid will reach this value at another point of the diagonal, whose distance from the origin is 0.5. In this way, the points $[0,0]$,$[0,1]$ and $[1,0]$ will certainly be on the lower half of the image of the sigmoid while only $[1,1]$ will be in the higher half. By identifying the two halves as false and true respectively, the theoretical expectation is reproduced. This reasoning is only done at convergence. Instead, in Fig.5.5 the complete information about the associated dynamical system is presented. In the AND function there is only one dynamical variable, i.e. the output. Once given the input, the velocity field is defined in the 1D state space, and it can be explicitly represented for all the inputs. This was done in Fig.5.5: In blue are reported the numerical value of the velocity field while in red are sketched the corresponding arrows, driving the dynamics at each value of y. As it is possible to evince, in all the cases there is only one equilibrium point and it resides in the correct half of the state space.

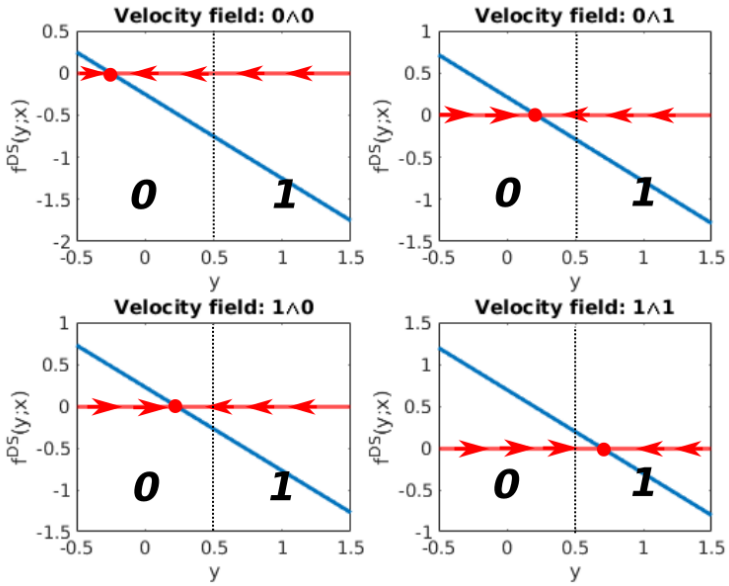


Figure 5.5: Caption

5.2.2 Non-linearly separable classification problem

Once clarified what is meant with using recurrent neural networks for the classification task, it is possible to move to considering the exclusive OR (XOR) problem. The function can be defined as

$$x_1 \otimes x_2 = \begin{cases} \sigma_\uparrow \text{ iff } x_1 \neq x_2, x_i \in \{\sigma_\uparrow, \sigma_\downarrow\} \\ \sigma_\downarrow \text{ otherwise} \end{cases} \tag{5.5}$$

This can be equivalently expressed with the truth table Tab.5.2 or, alternatively, with the graphical representation in Fig.5.6. As it is possible to evince, this problem is not linearly separable. This automatically implies that the architecture in Fig.5.2 is no longer sufficient for performing a

x_1	x_2	$x_1 \otimes x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Table 5.2: truth table of the XOR function.

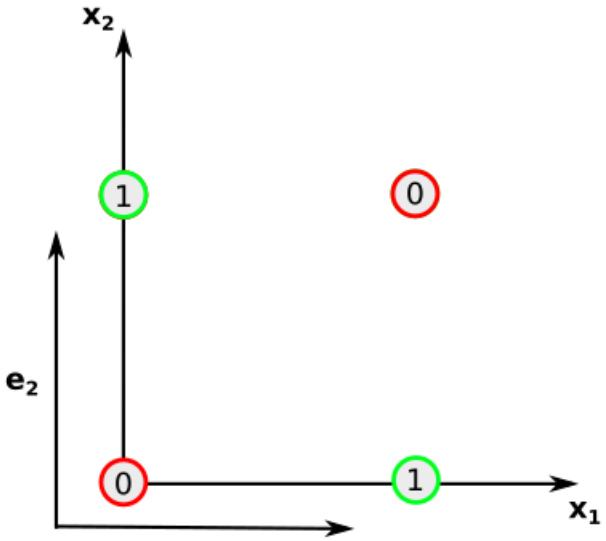


Figure 5.6: Graphical representation of the XOR classification problem.

similar classification. Nonetheless, the complexity is still sufficiently low to guess the minimally complex architecture needed to solve the problem. As a matter of fact the XOR problem can be written as two nested linearly separable problems

$$x_1 \otimes x_2 = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \tag{5.6}$$

as it is shown in Tab.5.3. Just as for the AND function, the composite non-linear function corresponding to this formula can be written. When considering activation functions bounded between 0 and 1, one should keep in mind that the result naturally extends to the hyperbolic tangent. Let us start from the negation

$$\neg x = 1 - 1x \tag{5.7}$$

Going to the composition of the negation with the AND

$$\neg x_1 \wedge x_2 = g\left(\frac{1}{2}(1 - 1 \cdot x_1) + \frac{1}{2}x_2 - 0.25\right) = \tag{5.8}$$

$$= g\left(W_{31}x_1 + W_{32}x_2 + b_3\right) \tag{5.9}$$

That is a linearly separable problem. Furthermore, also $x_1 \wedge \neg x_2$ can be solved with a similar function. Just as before, x can be substituted with $g(x)$ without compromising the separation,

x_1	x_2	$\neg x_1$	$\neg x_2$	$\neg x_1 \wedge x_2$	$x_1 \wedge \neg x_2$	$x_1 \otimes x_2$
0	0	1	1	0	0	0
0	1	1	1	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

Table 5.3: truth table of the XOR function.

finally obtaining a formula corresponding to the parametric model that can be learned by the Bengio-Scellier RNN. This implies that the minimally complex architecture that could solve the problem is the one reported in Fig.5.7. In fact, each layer must solve one of the nested

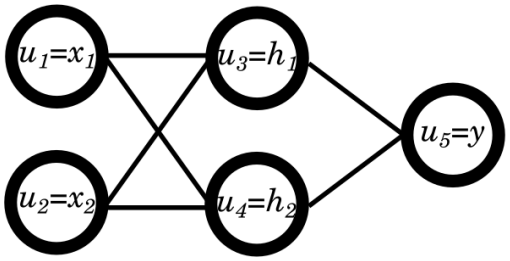


Figure 5.7: Minimally complex architecture for learning the XOR function

linearly separable problems. The training has been performed using $N_{train} = 400$ training data and $N_{test} = 320$ test data, sampled with the same approach used for the AND function(Fig.5.3). The hyper-parameters have been optimized to $\eta = 0.1$ and $\beta = 0.5$. The training was perrformed for $N_{epochs} = 10$ epochs and the performances were checked at each epoch, with mini batch size equal to 1, i.e. updating the weights after each data. The model reached an accuracy of 100% with $J_{test} = 6.8596 \cdot 10^{-7}$ and $J_{train} = 6.5668 \cdot 10^{-7}$.

The XOR problem is of particular interest because it allows a clear graphical representation of the state space. The network has two hidden nodes and one output node, so that the state space is a 3D euclidean space. For each input the system was initialized at ten random starting points:

$$s_i(t=0) = \zeta_i \quad i = 1,\ldots,3 \quad \zeta_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \tag{5.10}$$

The solution can be observed in Fig.5.9 and in Fig.5.10. As it is possible to evince, when the two inputs are equal the system converges toward an equilibrium point in the subspace $y = -1$, corresponding to $\hat{f}^{\theta} = -1$. On the contrary, when the inputs are one the opposite of the other the system converges to an equilibrium point at $y = 1$, corresponding to $\hat{f}^{\theta} = 1$.

Similar performances were obtained using a hard sigmoid. In the last chapter of this thesis it is described a 2x2x1 network trained for solving the XOR problem with the hard-sigmoid activation function. Nevertheless some difficulties were encountered in properly training the network under these conditions. Hence, in this case the problem was circumvented by introducing a hidden space with a slightly higher dimensionality, so as to increase the strength of the model in learning the logic function. The number of hidden nodes was increased to 5, thus creating the RNN $2 \times 5 \times 1$. The network was trained on 400 data and tested on 320, fixing the learning rate to 0.1 and the clamping factor to 0.5. The process was iterated for 10 epochs, using 12 data in each mini-batch. In the course of 10 epochs the network managed to achieve a 100% accuracy on the

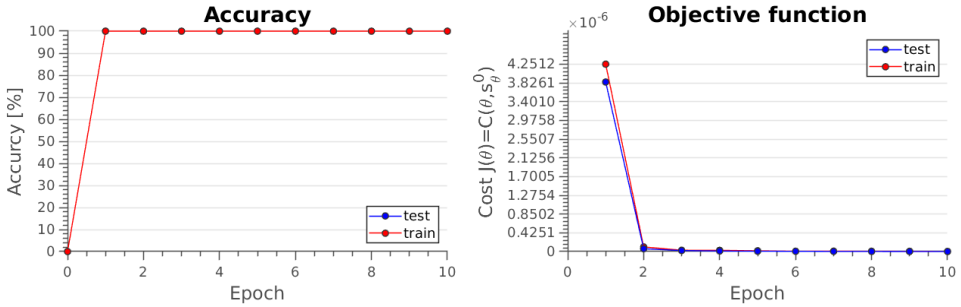


Figure 5.8: Performances of the RNN defined in the solution of the XOR classification problem.

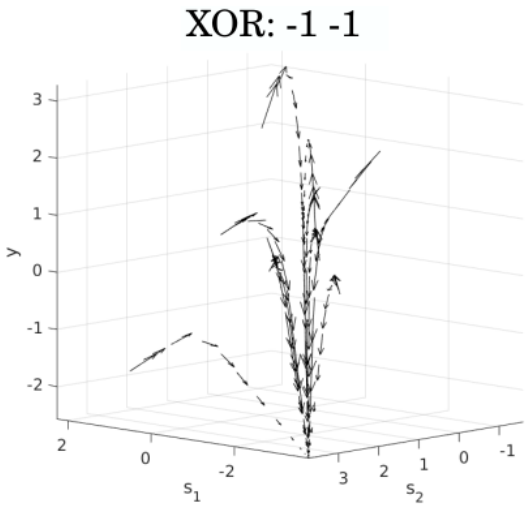


Figure 5.9: When the inputs are both belonging to the false class the system converges to $y = -1$ for every starting point in the state space.

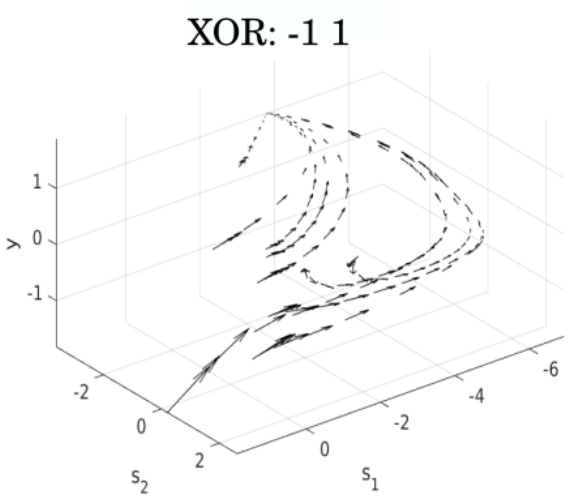


Figure 5.10: When the inputs are belonging to opposite classes the system converges to $y = 1$ for every starting point in the state space.

test data and a 99.375% on the training data, corresponding to a $1.65 \cdot 10^{-8}$ error on the training data and $5.59 \cdot 10^{-9}$ on the test data.

5.3 Digit classification

The classification of the MNIST data-set [65] is a universal standard for checking the correct functioning of a ML classifier. The content of this data-set are the ten handwritten digits going from 0 to 9, each one written as a 28×28 images. A nearest-neighbor interpolation was performed to obtain its 14×14 equivalent, with the aim of reducing the computational burden and to allow for a full data-set training within a reasonable time. The comparison between the two data-sets can be observed in Fig.5.11 The ML problem to be performed corresponds to determine the class number using handwritten images. The way this is done is sketched in Fig.5.12. The entries of the matrix representing an image are inserted in a RNN as the input vector x. Many architectures are possible to be used for the network but all of them must be characterized by one input layer of $N_i = 14^2 = 196$ input nodes and $N_o = 10$ output nodes.

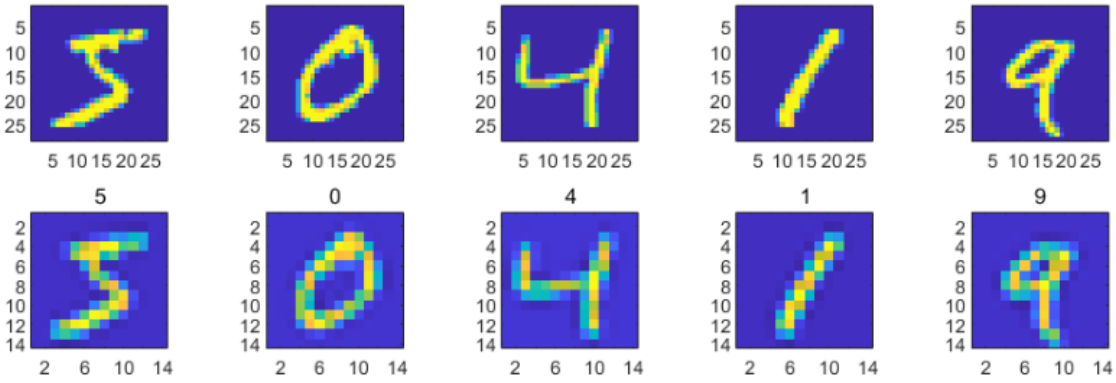


Figure 5.11: On the top there is the original 28×28 MNIST image while on the bottom there is the equivalent 14×14 version.

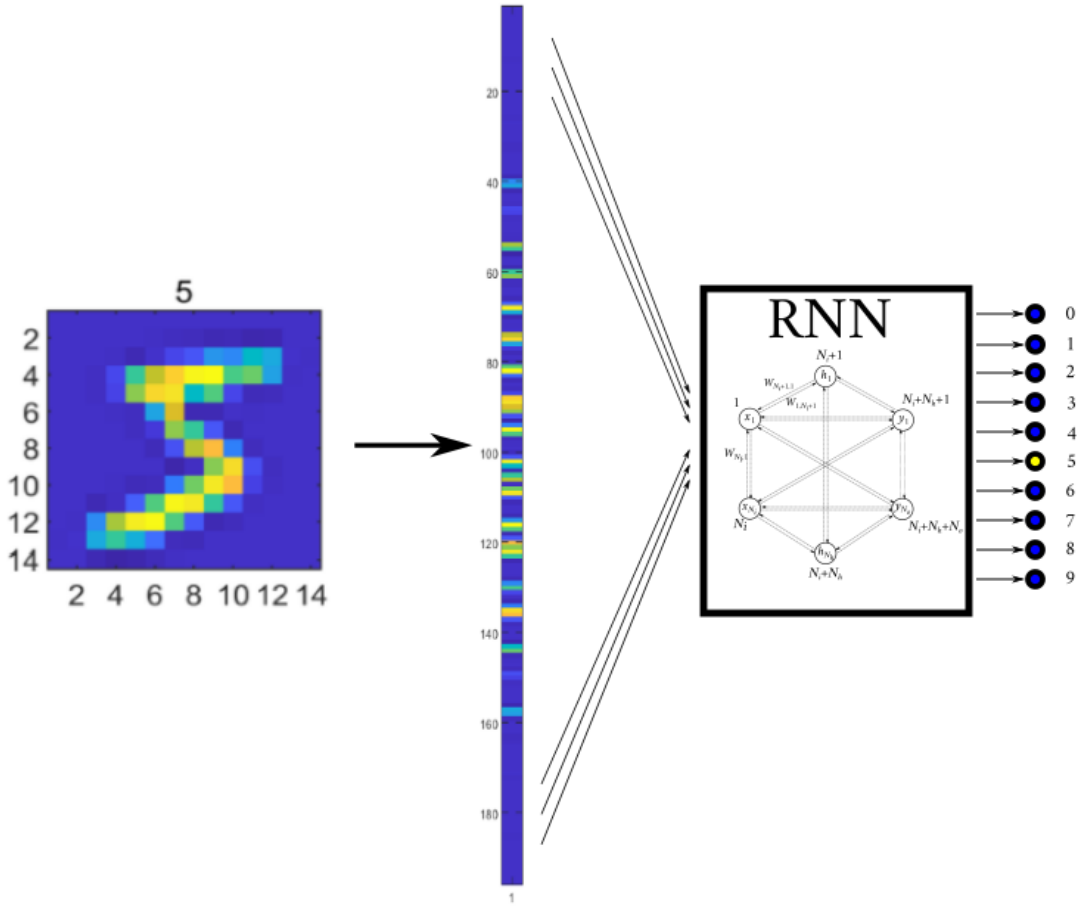


Figure 5.12: Scheme of the MNIST classification problem via a recurrent neural network

5.3.1 Multilayer network

The starting point of this analysis follows closely the architecture chosen by Bengio and Scellier [1]. The network considered is a three layers network, in which the 196-dimensional input is fully connected to 128 hidden neurons, that in turns are fully connected to the output. The activation

function used was the hard sigmoid and the hyperparameters were optimized to $\eta = 0.01$ and $\beta = 0.2$. Once initialized the network using the standard initialization, the model was trained on the whole data-set, reaching an accuracy of 96.98% in 10 epochs. In Fig.5.13 is reported the history of the training while in Fig.5.14 it can be observed the final confusion matrix of the

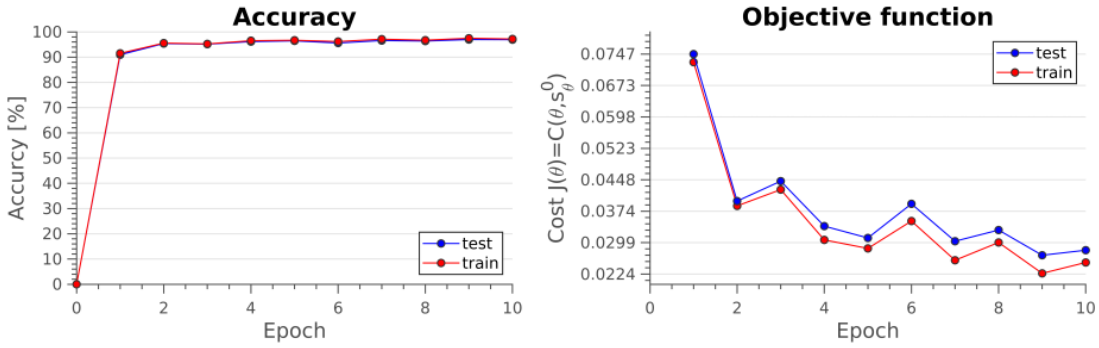


Figure 5.13: Accurcy and Cost of the network while training it on the MNIST data-set.

model. Bengio and Scellier [1] achieved 100% accuracy with an analogous network. However,

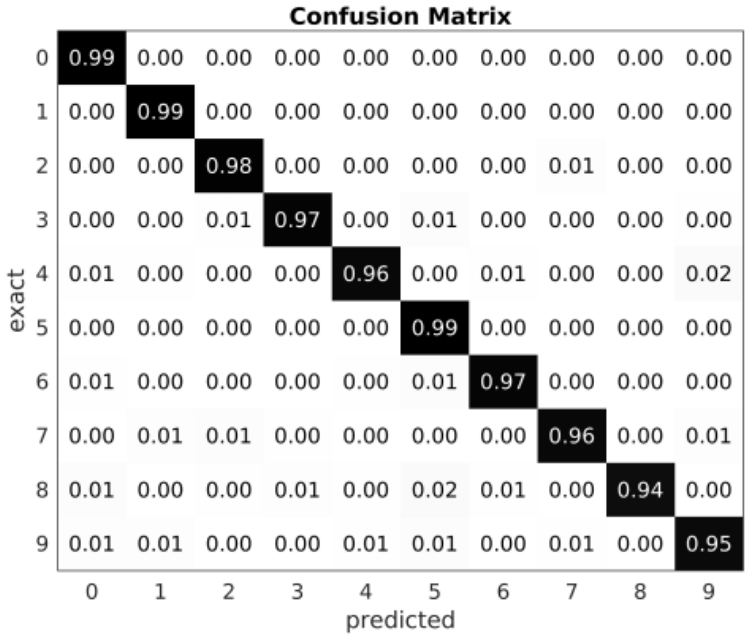


Figure 5.14: Confusion matrix of the MNIST classification using the hard sigmoid.

some important differences are present and might be the cause of the reduced performances of this implementation:

1. The dynamical evolution of the state nodes is different. This was done because the interest of this thesis aims at the circuital implementation and the Additive Model has already been studied from an hardware point of view;

2. They used the 28×28 images while here, for the aim of speeding up the calculations, 14×14 images have been used;

3. They introduced a different η parameter per each layer. This trick allowed them to increase the performances, even if it is not completely justified by their theory. In this work the algorithms strictly follow the original formulation of their work;

4. They used the Glorot-Bengio initialization while the standard one [64] was observed to yield better performances for the selected parameters;

5. They trained the network for 25 epochs while only 10 are considered in this analysis;

6. They integrated the dynamics equation manually while, in this work, the built-in `MatLAB` routine `ode45` has been used.

Due to the importance of the MNIST classification to benchmark a ML model, also the hyper-parmeters needed for solving the problem using the hard hyperbolic tangent were optimized. In Tab.5.4 are reported the main figure of merits of the MNIST classification using the two activation functions of interest. In both cases the network was trained on the whole data-set, corresponding to $N_{train} = 60000$ and $N_{train} = 10000$. The architectures selected were equivalent, with 196 input nodes, 128 hidden nodes and 10 output nodes. In both cases the training was performed for 10 epochs, with mini-batches of 20 data.

activation	η	β	ACC_{train}	ACC_{test}	J_{train}	J_{test}
sigmoid	0.01	0.2	0.975	0.9698	0.022596	0.026901
tanh	0.005	0.1	0.9642	0.9614	0.1182	0.12819

Table 5.4: Figure of merits of the two networks optimized for the MNIST classification task.

Mini batch size

The main purpose of this thesis is to work on the circuital implementation of the algorithms here discussed. To this aim, it is later discussed how the synaptic weights can be substituted with resistive switching devices and how it is possible to program them using external digital hardware. The important thing is that every time the weights are to be updated, a computation must be done and a programming procedure must be effected on the crossbar. Therefore, this process is both time and power demanding, implying the natural advantage of reducing the number of circuital updates to the lowest possible number. In particular, considering the previously introduced concept of mini batch, such an update is to be performed after the network has experienced a number of inputs equal to the mini batch size. So far this parameter was kept fixed to 20, in analogy to what done in [1]. However, in order to reduce the number of updates to be done on the weights, it is necessary to verify whether it is possible or not to increase the mini batch size. In order to do so, an architecture of the kind previously described was trained with the hard sigmoid activation function. The dimension of the data-sets used were respectively $N_{train} = 6000$ and $N_{test} = 1000$, rather than the full data-set. In practice, the only important thing is the variation in accuracy with respect to the working point of miny batch size equal to 20. Therefore, reducing the size of the data-sets considered induces a reduction in the accuracy but it preserves the possibility to perform a proper comparison. The hyper-parameters were kept fixed at $\eta = 0.05$ and $\beta = 0.1$ and the training was performed over 5 epochs. The results can be observed in Fig.5.15. The plot shows that the mini batch size could be increased with no significant loss of accuracy. Nonetheless, if it is increased too much it can lead to a drastic

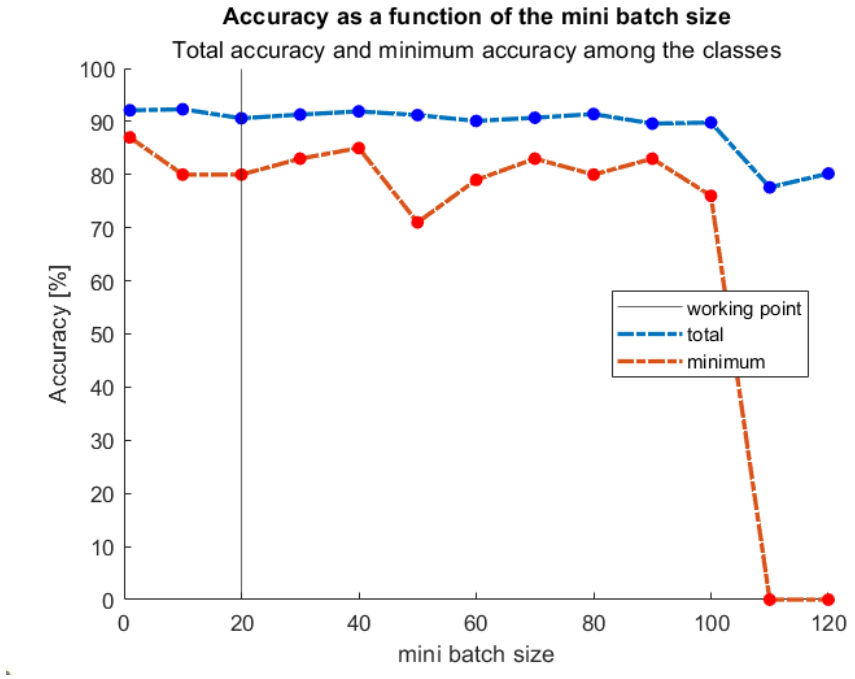


Figure 5.15: Accuracy as a function of the mini batch size

degradation of the performances. This is particularly evident if one considers the accuracy of the class whose classification is subject to the lowest accuracy. This is particularly dramatic when the mini-batch size is higher than 100. Nonetheless, from this analysis it is possible to see that the number of updates necessary could be halved, yielding a considerable reduction of the power required for the update.

5.3.2 Feature Engineering

Later in this work quantized network are discussed. Recently, Laydevant et al.[66] proposed an algorithm to train network with binarized weights. They managed to reach state of the art accuracies on the MNIST data-set but at the cost of introducing an hidden layer with a demanding number of neurons. The need to go in high dimension is easy to be understood when one passes from a continuous weights update to a discrete one. In the continuum much more information can be encoded in the weights, that can store particularly detailed informations about the correlations in between the neurons. When the information that can be stored in the weights is limited, a workaround is needed to achieve the same accuracy. Rather than limiting to increase the number of nodes, an alternative is to project the data-set into an higher dimensional space using a process called Feature Engineering. Features Engineering consists in pre-processing the input data to a network in order to limit the amount of information the network needs to extract. For this purpose it can be noticed that, despite the MNIST classification problem is not linearly separable, trying to perform a linear separation can lead to an accuracy of around 80%. The approach here proposed is based on the definition of the new features starting from the Principal Component Analysis and combining it to the previously gained insight on the nature of the information learned by the Additive model.

Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised ML technique capable to find the most informative linear representation of an object belonging to a given data-set. In particular, given a N_{train} dimensional data-set, each object of which is characterized by a number of features N_f, PCA determines a hierarchically-based new set of features [67]. In the following an overview on PCA done in a previous work [68] is adapted to the context under investigation.
Let us consider a generic object $x^{(i)} \in \mathbb{R}^{N_f}$. Firstly, an Euclidean orthonormal basis is introduced via the column vectors $\mathscr{B}_f = \{e_s\}_{s=1}^{N_f}$ and the corresponding identity operator reads

$$\mathbb{I} \doteq \sum_{s=1}^{N_f} e_s e_s^t$$

The object can then be represented as a linear combination of these basis vectors, weighted by the associated projection

$$x^{(i)} = \sum_{s=1}^{N_f} (e_s^t x^{(i)}) e_s = \sum_{s=1}^{N_f} x_s^{(i)} e_s \tag{5.11}$$

The aim of Machine Learning is to identify and exploit informative patterns in a data-set. Therefore, disposing of a large number of such objects is desirable. As already mentioned, these objects are generally stored in a matrix, that is named X. Having at hand this data-set allows to define a new basis in which to describe the input vectors, and this is done through a procedure named Principal Component Analysis. Overall, performing a PCA on the dataset X_{train} can be identified with the definition of two important quantities:

- The average vector x_o. The knowledge of this quantity allows to describe each object as a variation from it $x = x_o + \tilde{x}$. $\tilde{x}$ is termed the mean-adjusted vector and the mean vector in the previously defined basis reads

$$(x_0)_k = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} x_k^{(i)}.$$

- A new orthonormal basis: $\mathscr{B}_{pca} = \{p^{(l)}\}_{l=1}^{N_f}$. This set of vectors is generally referred to with the name *principal components* (or principal directions). They correspond to the normalized eigenvectors of the covariance matrix

$$C = \sum_{s,s'=1}^{N_f} c_{ss'} e_s e_{s'}^t$$

$$c_{ss'} = \frac{1}{N_{train} - 1} \sum_{i=1}^{N_{train}} (x_s^{(i)} - (x_0)_s)(x_{s'}^{(i)} - (x_o)_{s'}).$$

  and the eigenvalues are termed the *variances* $\{\sigma_i^2\}_{i=1}^{N_f}$. The principal components are the directions along which, in the data-points, there are the most informative variations with respect to the average vector. They are hierarchically sorted depending on the value of the associated eigenvalue.

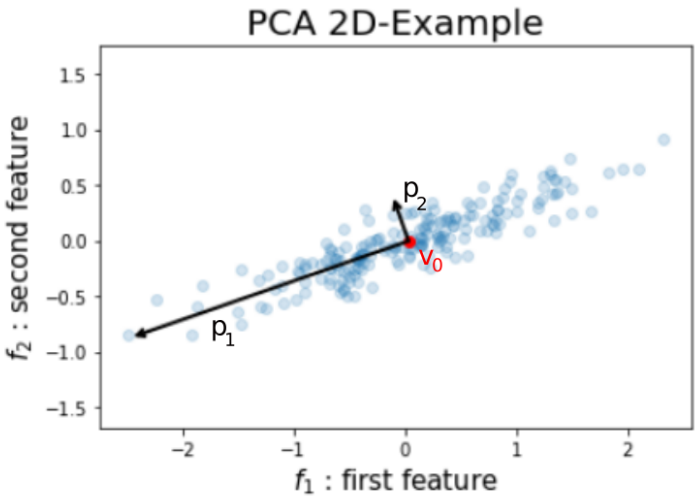


Figure 5.16: 2D example of PCA. The data-points are in light blue, the mean data-point is in red and the principal components are multiplied by a scaling factor proportional to the associated eigen-value. Taken with permission from [68].

Generally, the meaning of this analysis is explained by referring to a 2D example (see [69]). In Fig. 5.16 it is possible to observe a data-set of points randomly scattered along a line. PCA has been performed on this data-set with the aim of graphically representing the technique described so far. The mean vector x_0 is represented in red and any point in the data-set can be represented starting from this point. The most informative direction is the eigen-vector of the correlation matrix associated to the biggest eigenvalue, namely $p^{(1)}$. The principal components have been scaled proportionally to the associated eigen-values to represent their different importance in describing the data. PCA has been able to understand that the most striking feature of the objects to be described is their distribution along a line. Therefore, the first principal component ($p^{(1)}$) corresponds to that line. Then, the second eigen-vector, orthogonal to the first one, takes into considerations the remaining information in the 2D space.
Once identified the previous quantities, any object x belonging to the same class of the data-set can finally be expressed in this new basis in an expansion called *principal components decomposition*:

$$x - x_o = \mathbb{I}\tilde{x} = \sum_{l=1}^{N_f} [(p^{(l)})^t \cdot \tilde{x}] p^{(l)} = \sum_{l=1}^{N_f} \gamma_l p^{(l)} \tag{5.12}$$

Linear separation of the MNIST data-set

With the aim of generating new features, a PCA was performed on the training data-set, obtaining in this way the matrix of the principal directions

$$P = \begin{bmatrix} | & | & & | \\ p^{(1)} & p^{(2)} & \dots & p^{(N_f)} \\ | & | & & | \end{bmatrix}$$

As said before, the column vectors in this matrix identify the most informative directions in the features space. They are sorted by importance, so that $p^{(1)}$ will be the direction along which the

data most strikingly vary. The optimal features in which to represent the data should be the most informative possible. Since the principal components allow to separate the features so that they are maximally informative, it is reasonable to define the variable

$$\gamma \doteq P^t \tilde{x} = \begin{bmatrix} \gamma_1 & \gamma_2 & \dots & \gamma_{N_f} \end{bmatrix}^t \quad x, \gamma \in \mathbb{R}^{N_f} \tag{5.13}$$

The new variable will have the same number of features. However, the component γ_i will have a precise higher hierarchical importance with respect to any γ_j, whenever $j > i$. In order to use this, first it was tried to solve the problem linearly. To each input vector it was subtracted the average one, computed on the training set

$$\tilde{x} = x - x_o \tag{5.14}$$

Then, the architecture was defined to be a one layer network, where each input was directly connected to the 10 output nodes, the only evolving variables. The network was trained with 3000 training data and tested against 500 images. The learning rate was set to $\eta = 0.05$, the clamping factor to $\beta = 0.5$ and the hard sigmoid was used as the activation function. After 10 epochs of training, the network reached a 82.8% accuracy on the test set and a 86.6% accuracy on the training one. The plots reporting the history of the training can be observed in Fig.5.17. This is a remarkable result, considering that the number of trained weights is 1960 synaptic

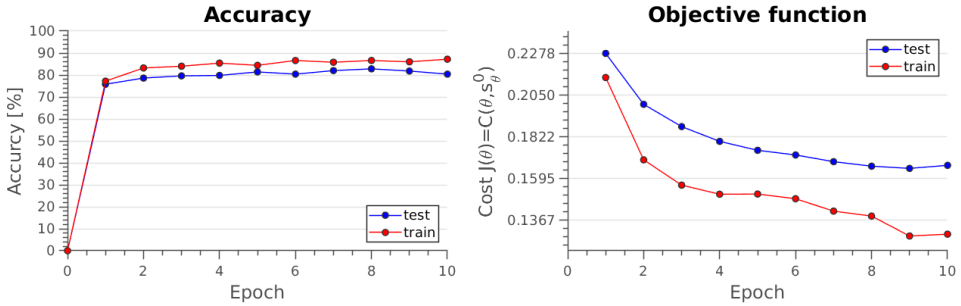


Figure 5.17: Caption

weights and 10 biases, for a total of 1970 parameters, considerably lower than the 250880 of the previous model. However, so far nothing new has been added since the new data-set is a linear transformation of the old one. This is to say that the learning process is just a linear separation of the original data in an alternative basis. Considering this high linearity in the data, it would be useful to exploit this for designing an efficient network, with the smallest number of parameters possible. When adding an hidden layer this linear information is lost. Therefore, in the next section we will propose some features with which to augment the original features. The goal is to introduce information that the RNN should learn by itself, simplifying in this way the task for the network and allowing to reach higher accuracies with a lower number of parameters.

Correlation based non-linear features

The goal of this section is to define non-linear features with which to augment the input. Let us consider once more the generic vector x and its projection γ in the space of the principal components. For the previous discussion about the representation in the PCA basis, the information contained in each entry of the projected vector γ is associated to a different principal component.

Therefore, the first entry will be much more informative than the last ones. If one wants to design new features while minimizing the number of weights used, the most natural thing to do is to try to use the first few M entries of γ. This is possible only thanks to the hierarchical way in which PCA redefines the variables. In order to increase the accuracy, the new features to be created must introduce non-linear information. The simplest way in which we can try to do this is by computing the Kronecker product of the first M values of γ with themselves. In this way, the non linear set of features for the variable x reads

$$\gamma^{nl} = \begin{bmatrix} \gamma_1^2 & \gamma_1\gamma_2 & \cdots & \gamma_1\gamma_M & \gamma_2\gamma_1 & \gamma_2^2 & \cdots & \gamma_M^2 \end{bmatrix}^t. \tag{5.15}$$

Alternatively it is possible to define the associated matrix

$$\Gamma^{nl} = \begin{bmatrix} \gamma_1^2 & \gamma_1\gamma_2 & \cdots & \gamma_1\gamma_M \\ \gamma_2\gamma_1 & \gamma_2^2 & \cdots & \\ \vdots & & \ddots & \vdots \\ \gamma_M\gamma_1 & & \cdots & \gamma_M^2 \end{bmatrix}. \tag{5.16}$$

If one considers the entry Γ^{nl}_{mk}, this reads

$$\Gamma^{nl}_{mk} = \gamma_m\gamma_k = \gamma_m\gamma_k^t = (p^{(m)})^t \cdot \tilde{x}\tilde{x}^t p^{(k)} \tag{5.17}$$

Where it has been used the fact that the transpose of a scalar is the scalar itself. Finally, defining P_M as the matrix whose columns are the first M principal components, the Γ^{nl} matrix defining the new feature vector γ^{nl} reads

$$\Gamma^{nl} = (P_M)^t(x-x_o)(x-x_o)^t P_M \tag{5.18}$$

By direct comparison of the Γ^{nl} matrix with the correlation matrix it is possible to evince that the introduced non-linear features are the contribution of the single data to the correlation matrix, represented in the sub-space of the first M principal components. In the end, given an input vector $x \in \mathbb{R}^{N_f}$, the new set of features is defined as the vector $z \in \mathbb{R}^{N_f+M^2}$

$$z = \begin{bmatrix} \gamma \\ \gamma^{nl} \end{bmatrix}. \tag{5.19}$$

The Kronecker product of a vector with itself generates repeated values. Nonetheless they were not removed, since an increase in performances was observed when proposing also the repeated values. Therefore, an additional number of M^2 entries were inserted. The single layer network was trained for a progressively increasing number of components. The reason why this choice might be well suited for helping the network to learn seems to be related to the relationship existing between PCA and the nature of the information encoded in the Additive model. In Chapter 4 the information encoded in the weights is argued to be closely related to the correlation existing in between the variables. Hence, using as new features the most relevant correlations with a given number of features is a good way to obtain a tunable accuracy with the number of parameters.
The important result is that by reducing the number of parameters by one order of magnitude it was still possible to achieve performances comparable with the ones obtained in the multi-layered network. To understand why this might be useful, one should consider that at least one resistive

M	features	parameters	accuracy test	accuracy train
0	196	1970	82%	87%
10	296	2970	92.2%	95%
15	421	4220	92.6%	96.8%
20	596	5970	93.6%	98%
25	821	8220	94%	98.8%
30	1096	10970	95%	98.6%

Table 5.5: Increase of the performances when adding the PCA-based engineered features. The first column is the number M of components considered, the second column is the dimensionality of the newly defined input vector z, i.e. $N_f + M^2$. The remaining columns are the remaining figures of merits.

switching component should be associated to each one of these parameters. Working prototypes of similar networks have already been discussed. However, researchers are often forced to limit the experimental measurements to simple one layer networks with a limited number of resistive components. Since this often results in dramatic reductions in the accuracy [70], the proposed feature engineering might be a simple tool for increasing the compactness of the networks while guaranteeing a satisfactory accuracy with single layer networks.

5.4 Conclusions

In this chapter Equilibrium Propagation is used for solving classification problems of progressively higher degree of complexity. The starting point is the AND function, the analysis of which allows us to discuss a 1D dynamical system and the way the velocity field can be learned. Then, a 3D dynamical states space is analysed while solving the XOR problem. This problem is of historical interest in the Machine Learning world and allows to verify the correct implementation on a non-linearly separable problem. Finally, the MNIST classification problem is addressed. The main advantage of these numerical simulations is that they allow to verify the capability of the Additive Model to solve also classification tasks in multilayered networks. In this way, new valid architectures are optimized, obtaining in this way optimal parameters, useful when going to the hardware implementation of the network. In addition, two hardware oriented analysis are performed. Firstly it is verified the possibility to reduce the number of updates needed in the training phase. Secondly it is defined an heuristic approach to engineer informative features from the principal components of the training data-set. As mentioned in the main text, this might be particularly useful to increase the accuracy of the implemented networks at the research level.

Part II

Hardware

Chapter 6

Electronic Components

This chapter summarizes the basic features of the main circuital components used for the implementation. Taking for granted the resistor, the capacitor and the sources, the first component considered is the diode, of which its basic functioning and the Schottky version are briefly discussed. The second component is the Operational Amplifier, that is essential both for the implementation of the neuron and for the activation function. Finally, the resistive switching phenomenon in bipolar valence change memory devices is briefly described since it is at the basis of the implementation of the synspses.

6.1 Diodes

The diode is a non-linear component physically corresponding to a semiconductor junction. On one side of the junction the semiconductor is p-doped and is named *anode* while on the other side the semiconductor is n-doped and is called *cathode*. The reader can refer to the Sze book [71] for an in depth discussion on the physical mechanisms at the basis of the electronic properties of this device. In simplistic terms, at room temperature the dopants are ionized, so that on the n-side electrons are promoted to the conduction band while in the p-side holes are released in the valence band. If the materials are separated, both of them are globally and locally neutral in each region since the mobile charge balances the fixed charges of the dopants. On the contrary, when the junction is created there will be a high difference in concentration of the mobile carriers and a diffusion process will drive the free carriers in the material in which they are minoritary. As this process goes on, while the global neutrality is always satisfied, the local neutrality is violated in the junction region, where the carriers are injected to the other side. This charge will generate an electric field competing with the diffusion of the carriers, i.e. introducing a competing drift current. In terms of energies, the generation of a localized charge locally alters the electrostatic potential of the free carriers, progressively creating a barrier aimed at confining the free carriers in the region in which they are majoritary. Once equilibrium is reached, the current components are balanced and no current flows through the device. If a positive voltage is applied at the p-side, this will result in a lowering of the barriers and in a consequent injection of minority carriers that will diffuse. In the meanwhile, in the region where they are the majoritary carriers they experience an electric field and move by a drift effect. Overall, this will result in an exponentially increasing current as a function of the applied voltage. On the other hand, if a negative voltage is applied at the anode, the barriers are further raised. The only available conduction mechanism is an injection of the minority carriers populating the top parts of the respective energy barriers.

This results in an almost constant reverse saturation current, generally termed I_o. Putting these things together, it is possible to mathematically prove that this device can be considered as a non-linear resistor and its characteristic equation reads

$$i_D = I_o\left(e^{\frac{v_D}{V_T}} - 1\right) \tag{6.1}$$

In Fig.6.1 it is shown the circuital symbol together with the I-V plot of the *Default* diode model implemented in `ngSpice`. The red dashed line corresponds to the piece-wise approximation of the diode in which the current is assumed to be zero up to the voltage V_γ, starting from which the diode behaves as a voltage generator at the voltage $V_\gamma \simeq 0.7V$. These devices can be characterized by the so called *forward voltage*, corresponding to the positive voltage at which the current has a selected value

$$i_{D;fwd} \simeq I_o e^{v_{D;fwd}/V_T} \rightarrow v_{D;fwd} = V_T \log \frac{i_{D;fwd}}{I_o} \tag{6.2}$$

The substitution of one of the two semiconductors with a metal corresponds to the definition of

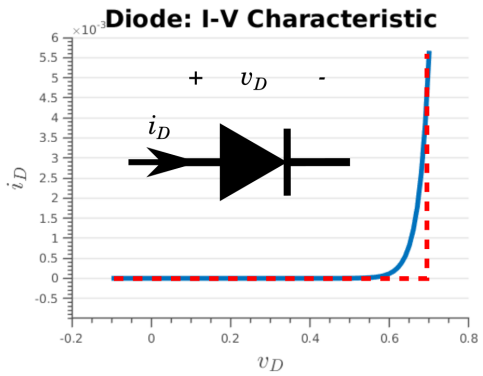


Figure 6.1: Symbol of the p-n diode and I-V characteristic of the `ngSpice` default model.

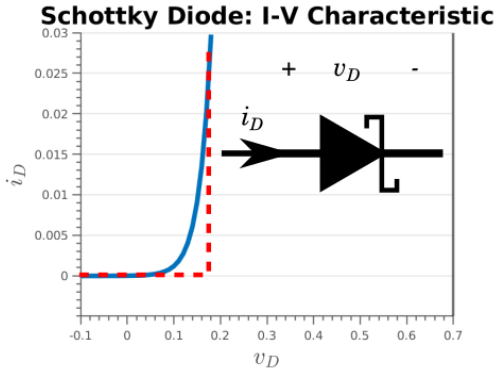


Figure 6.2: Symbol of the Shottky diode and I-V characteristic of the `ngSpice` DFLS220L model.

the Schottky diode. This device qualitatively behaves as the diode since its characteristic curve can be found to be

$$i_D = I_s\left(e^{\frac{v_D}{V_T}} - 1\right) \tag{6.3}$$

Nonetheless, the inversion charge might now be dominated by a new phenomenon. In fact, in inversion condition, the inversion current will still have a contribution from the injection in the metal of the minority carriers of the semiconductor. However, from the metal side the carriers will flow to the semiconductor due to thermoionic emission. This change in the physical phenomenon results in a reverse current that can be considerably higher than the the one in the diode $I_s \gg I_o$. Therefore, when considering the forward voltage of the Schottky diode at the same reference current, the two are related by

$$v^{s-s}_{D;fwd} - v^{m-s}_{D;fwd} = V_T \log \frac{I_s}{I_o} \tag{6.4}$$

Since the ratio of the two terms can be of the order of 10^6 and at room temperature $V_T = 26mV$ the forward voltage can be hundreds of millivolts lower than the corresponding forward voltage of the p-n diode. If the threshold voltage V_γ of the p-n diode is of the order of $0.6V$, the corresponding quantity for the Shottky diode can be as low as $V_\gamma = 0.15V$. In Fig.6.2 it can be observed the case of the DFLS220L Shottky diode [72], that is used in the final implementation of the neuron unit.

6.2 Operational Amplifier

The next electronic component to be introduced is the operational amplifier.

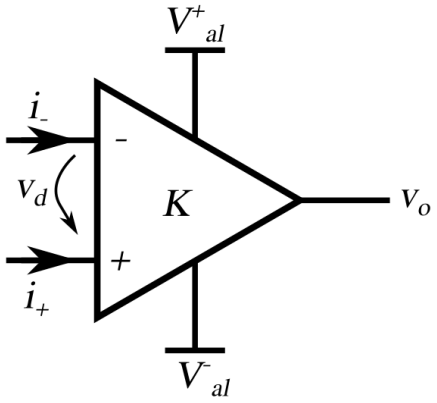


Figure 6.3: Symbol of the Operational Amplifier.

This device is represented in Fig.6.3 and, ideally, the current entering at its input nodes and the differential voltage between its input nodes can be neglected.

$$\begin{cases} i_+ \simeq 0A \\ i_- \simeq 0A \\ v_d \simeq 0V \end{cases} \tag{6.5}$$

The interesting thing about these features is that, if the differential voltage v_d is close enough to $0V$ and the positive node is connected to ground, then also the negative node is constrained to a low voltage, without being directly connected to ground. This node is said to be at *virtual ground*. Each operational amplifier presents at the output an amplified version of the input differential voltage $V_o = f_K(v_d)$, where the meaning of the K parameters can be observed by performing a first order Taylor expansion of the output

$$\begin{aligned} f_K(v_d) &\simeq f_K(0V) + f'_K(0V)v_d + o(v_d) = \\ &= f_K(0V) + Kv_d + o(v_d) \end{aligned}$$

While $f_K(0V)$ corresponds to the bias term, the K factor is the gain of the amplifier. At the output of the component the voltage must vary. Therefore, the ideal operational amplifier must have a diverging gain $K \to \infty$. Another interesting property is the dependence of the device on the power supply. The amplifier cannot present an output voltage higher than V^+_{al} or lower than V^-_{al}. Therefore, in case the circuit tried to drive the output outside from its natural limits, the

device would tend to saturate at the supply values. The properties of this component make it extremely versatile and ubiquitous in analogue electronics. In the following chapters it will be shown to be the key component for the implementation of an analogue neuron. In fact, it will appear in all of its constituent blocks: from the integrating block to the activation function.
In order to go further, let us discuss a more realistic model for the component. In Fig.6.4 it is reported a model of the operational amplifier, allowing to take into account some non-idealities.

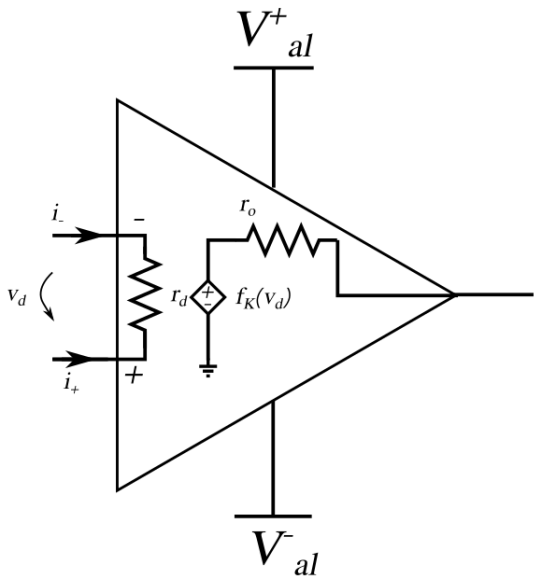


Figure 6.4: Internal functioning of an operational amplifier: basic model as a voltage controlled source.

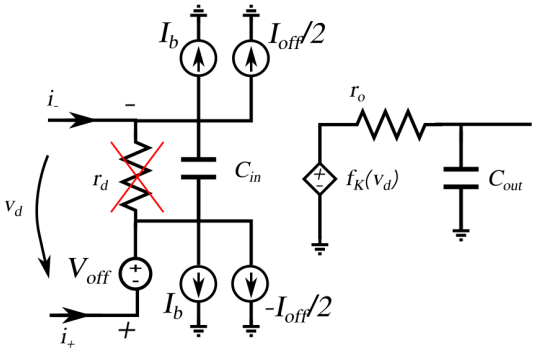


Figure 6.5: Internal functioning of an operational amplifier: parasitic currents and capacitors.

First of all, the input nodes are assumed to be connected by a resistance. This implies that the input currents will in general be different from zero, as well as the differential input voltage will be not rigorously vanishing. In general, the input resistance is considerably high, generally in the range $r_d \in [10^6, 10^{12}]\Omega$, so that it can safely be assumed to satisfy the ideal limit $r_d \to \infty$. Moving to the output port, the operational amplifier takes a vanishing input voltage and it transforms it into a finite output voltage. Therefore, considering the gain factor K, in the ideal component it must tend to infinite. As a further constraints on the gain, the output voltage can never exceed the values of the power supply in both direction. To properly model this function, the voltage controlled source representing the voltage amplification was modelled as follows:

$$\begin{cases} f_K(v_d) = V_{al}^- + \Delta V_{al} f^\sigma\left(\frac{K}{\Delta V_{al}} v_d\right) \\ \Delta V_{al} = V_{al}^+ - V_{al}^- \end{cases} \qquad \lim_{x\to\infty} f^\sigma(x) = 1 \qquad \lim_{x\to-\infty} f^\sigma(x) = 0$$

This expression is reasonable because it allows for a continuous amplification saturating at the low alimentation value and at the high alimentation value. Moreover, if one linearizes the function around the ideal differential voltage:

$$\begin{aligned} f_K(v_d) &= f_K(0V) + \left.\frac{df^\sigma}{dv_d}\right|_{v_d=0V} v_d + o(v_d) \\ &= f_K(0V) + K v_d \end{aligned}$$

corresponding to the usual linear amplification of the operational amplifier. The hyperbolic tangent was chosen to model this behavior:

$$f^{\sigma}(v_d) = \frac{1}{2}\left[1 + \tanh\left(\frac{2K}{\Delta V_{al}} v_d\right)\right] \tag{6.6}$$

With this choice the saturation of the amplification was correctly accounted for and the correct behavior for small values of the differential voltage

$$\begin{cases} \lim_{v_d \to \infty} f_K(v_d) = V_{al}^{+} \\ \lim_{v_d \to -\infty} f_K(v_d) = V_{al}^{-} \\ \left.\frac{df^{\sigma}}{dv_d}\right|_{v_d = 0V} = K \end{cases}$$

Additionally, at zero input voltage the operational amplifier is biased in such a way that the output voltage is half the alimentation value $f_K(0V) = \frac{1}{2}(V_{al}^{+} + V_{al}^{-})$. In the ideal case the gain factor should diverge $K \to \infty$ to guarantee an appreciable voltage at the output of the device. Instead, in real applications such a gain factor ranges between 10^5 to 10^6. The worst case value for an operational amplifier is of the order of 10^4. Since this quantity is just a number, it is not influenced by the adimensionalization procedure and it is directly mapped into the corresponding circuit quantity. Finally, while in the ideal device the output resistance is zero, in the real device an output resistance r_o needs to be considered. The open loop transcharacteristic for various gain

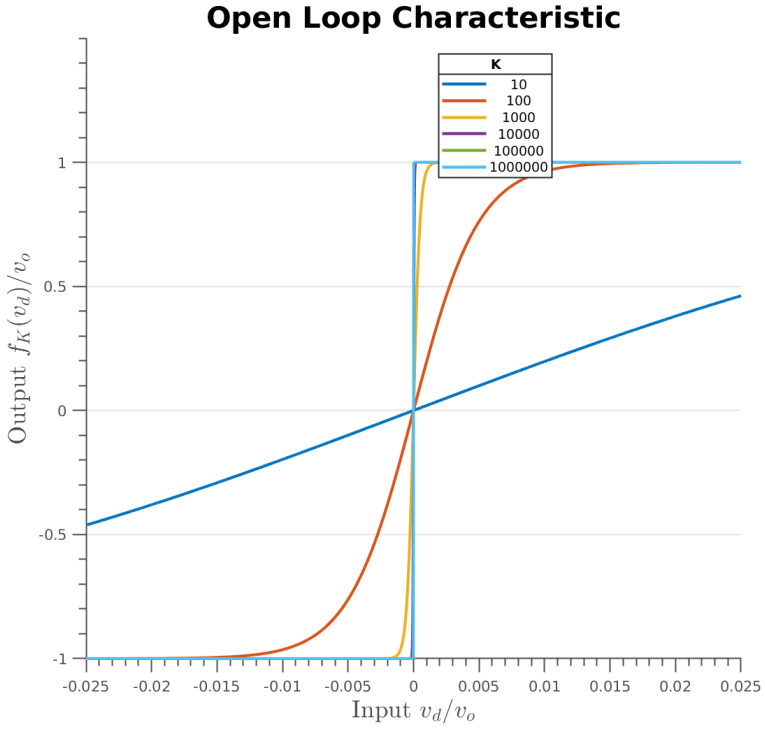


Figure 6.6: Open loop transcharacteristic for various gain values of the chosen operational amplifier model.

values of this device can be observed in Fig.6.6. It is worth remarking that, even if the worst case values for the non idealities are $r_o = 150\Omega$ and $K = 10^4$, in the simulations even worse

values were considered. This was done because some models are found to be extremely stable to these non-idealities while the proposed model for the amplifier is extremely general. Therefore, showing the stability of the implementation to extremely bad amplification units is not limited to operational amplifiers. On the contrary, this would suggest the possibility to use less complex and area demanding amplification blocks.

6.3 Resistive Switching Random Access Memory

In this section it is qualitatively discussed the working principle of an extremely promising resistive memory device: the Resistive Random Access Memory (ReRAM). This memory is one of the most popular to date due to its large number of interesting features. First of all, it can be used as a memory since its resistance value can be programmed by means of a proper electrical stimulus. Moreover, as later discussed, the physical process at the basis of the resistance variability can be harnessed to store more than one bit in a single component. In an era in which increasing the density of integration of the CMOS technology has become a serious challenge, this property is extremely interesting, making the ReRAM a possible competitor with the CMOS-based NAND Flash memory technology. The structure of a ReRAM is extremely simple. It consists of a two terminal metal-insulator-metal, where the insulator is often taken to be Hafnium Dioxide HfO_2. This is appealing because the HfO_2-technology is compatible with the CMOS implementation, and this is an essential requirement for allowing a smooth technological transition. In addition to this, the simplicity of the structure makes it easily scalable and allows for a high density integration. As mentioned before, one of the main interests in the Neuromorphic community is the reduction of the power consumption. Also in this perspective the ReRAM seems a good choice due to its low power consumption ($\leq pJ$). Additionally, the device presents a good capability to maintain the programmed state, i.e. it exhibits good *retention* capabilities. In order for the ReRAM to be competitive as a non-volatile memory, this device should be characterized by a fast switching from one state to another, but its resistance value should also remain unperturbed when applying voltages of few hundreds of millivolts. For what concerns the switching speed, this seems to be satisfied by the current ReRAM experimental realizations. On the other hand, the low voltages allowed to avoid an alteration of the states is something to be considered very carefully, and will be at the basis of the proposed implementation.

6.3.1 Physical mechanisms and programming

The physical mechanism at the basis of the resistive switching properties of a ReRAM is widely accepted to be the formation and rupture of a conductive nanofilament (CNF). In the following the work done by Giovinazzo et al.[35] is closely followed as it constitutes a good reference for understanding how it is possible to experimentally perform the multistate programming. For this reason, their implementation of the ReRAM is used as a reference. In their implementation, the voltage is applied to the Titanium Top Electrode (TE) while the Platinum Bottom Electron (BE) is grounded. The oxide of their choice was HfO_2. The starting point of this overview is a DC characterization, that will shed light on the concept of Conductive NanoFilment CNF. Afterwards, the focus is put on pulse-based programming in the HRS, that is extremely promising in terms of fine control of several multi-states. Finally, the Stanford model [73] is outlined, due to its versatility for `Spice` simulations while being highly effective in fitting experimental data.

DC characterization

Let us start from applying a voltage sweep to the TE with the BE grounded, in order to study the I-V curve (Fig.6.7). The first step is to create the filament, in a process called *forming* (red line). After the filament is formed the subsequent processes just require a local formation and destruction of the filament. On the contrary, at the beginning the filament needs to be built up to connect the conductive electrodes. A positive bias attracts the O^{2-} ions present in the HfO_2 to the TE. Titanium is easily oxidized, so that a layer of TiO_x appears at the interface.

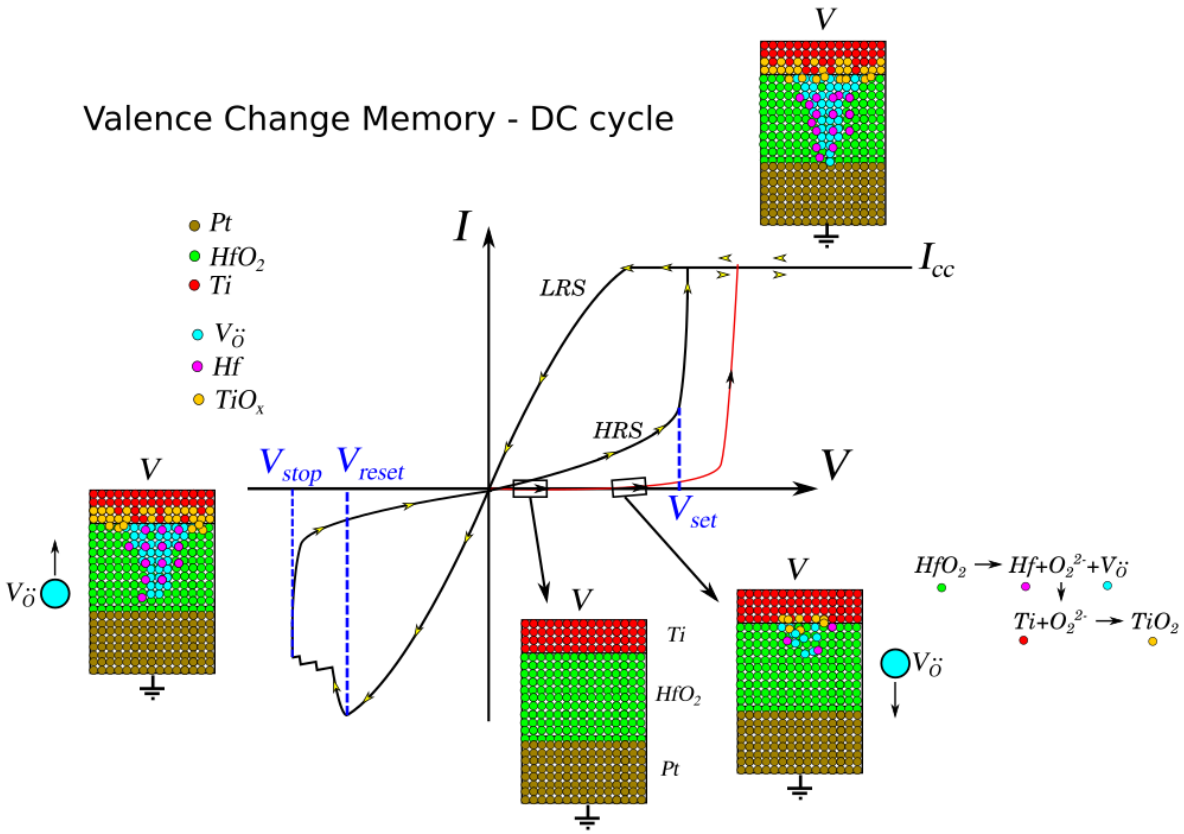


Figure 6.7: Schematic representation of the DC cycle in a valence change memory device.

As the process goes on, the absence of oxigen negative ions progressively forms a filament of positive vacancies $V_{\ddot{O}}$, attracted toward the BE. The filament will have a conic shape, wide at the TE and narrow at the BE. Once the filament is formed, the device is in the LRS thanks to a conductive filament between the electrodes. From now on, in normal conditions, the device will mainly experience local changes of the filament at the BE. In fact, if one now applies a negative voltage, the oxigen vacancies in the intermediate oxide start to be attracted by the TE and start to experience a force that makes them migrate toward the TE, far from the thin tip of the cone. The electric field gets stronger the closer the vacancies are to the BE, therefore, as soon as the electric field is sufficiently high, vacancies start to migrate toward the TE. In this process, an insulating gap is formed at the narrow side of the filament, reducing the capability of conducting electrons from one metal plate to the other. The first appreciable effect of migration can be observed at a voltage value named V_{reset}. After this first abrupt change, a step-like behavior is then observed in the resistance states, witnessing the progressive rupture of the filament. This is usually attributed to the presence of a preferred atomic-scale rearrangement. In Fig.6.7 is also

reported the V_{stop} value, corresponding to the minimum voltage value considered in the DC cycle. Below this value, the system is in a HRS depending on the value of V_{stop} and the voltage can be raised to explore this new configuration. In any case, what results from the DC analysis is that after the forming process a cyclic behavior can be attained. Starting from the HRS one can apply a positively increasing voltage. When the applied voltage reaches a V_{set} value, in a relatively quick time interval the system jumps from the HRS to the LRS, thanks to the CNF creation. The scheme shows also the compliance current. This is done because, when the device switches to the high conductance state, the current might raise to unbearable values. The compliance current (I_{cc}) removes this possibility by limiting the amount of current that can flow in the device.

Programming: Compliance current and reset voltage

In Fig.6.7 it is possible to observe that two parameters are externally fixed: the stop voltage and the compliance current. Both of them can be exploited to tune the resistance states of the devices. In Fig.6.8 are shown three I-V curves, parameterized on different V_{stop} values, in order to show the effect that changing this parameter has on the cycle characteristic. When following the IV

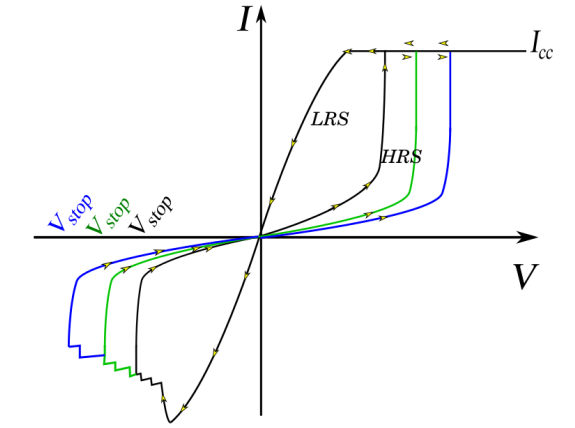


Figure 6.8: Programming the memristor with the stop voltage.

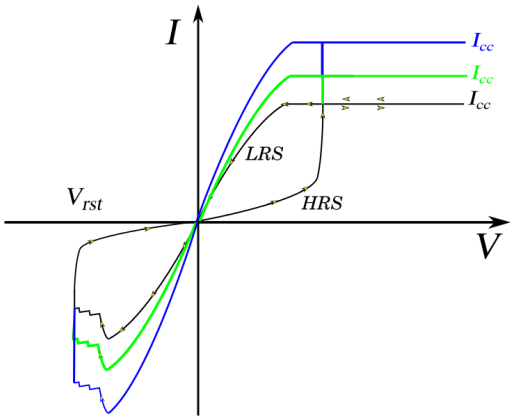


Figure 6.9: Programming the memristor with the compliance current.

plot in the LRS, independently of the stop voltage, the first rupture of the filament will always occur at the same V_{reset} voltage. What changes is that, as the negative voltage is further increased, more oxigen vacancies will migrate, lowering the conductance of the HRS. Therefore, lowering the V_{stop} will induce a tuning of the HRS. On the other hand, let us change the compliance current. In Fig.6.9 the effect of this change was sketched. The compliance current blocks the process of vacancies formation to a threshold value. By increasing (lowering) its value, it is possible to increase (reduce) the thickness of the CNF or, more likely, the number of CNF. This will induce an increase (lowering) of the conductance in the LRS.

HRS modulation by pulses

In addition to the previously discussed possibility to tune the resistance states, Giovinazzo et al. also discussed a third possibility. From the DC analysis it was remarked the presence of step-like resistance states in the device while moving from the V_{reset} to V_{stop}. These states can also be exploited and are particularly promising because a voltage variation as small as $\Delta V \leq 50mV$ is

sufficient to determine a resistance change. Therefore, this seems likely to enable a fine grain control of several multistates of conductance. As shown by the authors, the most reliable way of doing it is by means of pulses. By applying negative (positive) pulses it is possible to have depression (potentiation) of the synaptic connection. In this way, they proved the existence of a linear variation of the conductance in the range $[1\mu S, 32\mu S]$, that they called the *Subthreshold region*. A linear variation with the number of pulses is a relation which is extremely appealing in the perspective of developing fast programming approaches and in terms of having a sufficient number of conductance states to exploit for mimicking the synapses. These findings are at the basis of the discussion done in the following chapters. Nonetheless, it should be pointed out from the beginning that the first two programming approaches are more likely to be used for our system. The absence of state retention when the stimulus is around $50mV$, despite being an advantage, is not compatible with the retention requirements of our system.

6.3.2 The Stanford/ASU model

The promising features of the ReRAM devices led to the creation of various models for their simulation. The technological study of the device is out of the scope of this work and they cannot be discussed extensively. However, the interested reader can refer to the unifying work done by Panda et al.[74] for an extensive description of the whole spectrum of possibilities. Among the various models, this work focuses on the Stanford/ASU model [73]. This is a physics based parametric model in which the filament gap previously discussed is treated as an internal state variable. The groundbreaking idea of Guan et al. was that, when it comes to electron conduction in the ReRAM environment, the conduction mechanisms most commonly associated with ReRAM systems share some common features: both the tunneling distance and the field strength have an exponential relationship. For this reason, they defined a generalized conduction mechanism capable of encoding these properties. The electron conduction mechanism they proposed reads:

$$i(v;g) = i_{SA} e^{-g/g_{SA}} \sinh \frac{v}{V_{SA}} \tag{6.7}$$

Thanks to the generality of the model, it is possible to find the effective parameters i_{SA}, g_{SA} and V_{SA} by means of a fit of the experimental data. As reference values, the fit performed in the seminal paper are considered: $g \sim 0.3nm$ and $V_o \sim 0.6V$. Given this information, this model can be associated to the programming method reported in Fig.6.8. In fact, when programming by means of a change in the stop voltage, the effect is exactly to vary the gap in the high resistance state.

An extremely important thing to be remarked is the intrinsic stochasticity of the ReRAM devices. From an experimental point of view, it is in principle possible to identify a relationship in between the stop voltage and the corresponding gap. However, this dependency will not be entirely deterministic. More in general, what it is possible to obtain is an average relation over the programming cycles: $\langle g \rangle_{cycles}(V_{stop})$. This is the so called cycle-to-cycle variability, which is one of the most important stochastic behaviors in the device and the `Spice` implementation of the Stanford model takes this into account. In this perspective, one should immediately understand that the hardware version of the synaptic weights is considerably different with respect to the one used in the *Learning Algorithms* part. First of all the weight should not vary in the continuum but rather in a quantized way. This is primarily motivated by the physics of the device. Each time a vacancy recombines, removing a contact, the conductance is lowered by a quantum of conductance. Moreover, due to the stochasticity of the filament rupture the states must be defined

sufficiently far apart to be identified as distinguishable, and fluctuations should be introduced in the description. In the Stanford model, this was done by adding to the gap a time and temperature dependent gaussian random fluctuation, with the temperature varying as the device is electrically stimulated. Another striking difference with respect to the weights previously considered is the linearity. As deepened in the next chapter, the Additive model can be mapped to a linear RC circuit, in which the current on the synaptic weight G_{kj} reads $i_{kj} = G_{kj}\varphi(v_j)$, where the function is bounded by a characteristic voltage value $|\varphi(v_j)| \leq V_o$. Since $\varphi(v_j)$ corresponds to the voltage drop at the terminals of the ReRAM, the corresponding Stanford expression would be

$$\begin{cases} i_{kj} = G_{kj} V_{SA} \sinh \frac{\varphi(v_j)}{V_{SA}} \\ G_{kj} \doteq i_{SA} e^{-g_{kj}/g_{SA}} / V_{SA} \end{cases} \tag{6.8}$$

From this it is clear that, in order for the previous analysis to be valid, the characteristic voltage of the circuit needs to be fixed so as to work in linearity. This is to say that, if $V_o \ll V_{SA}$, the system is approximately in linearity and the current reads $i_{kj} \simeq G_{kj}\varphi(v_j)$.

Chapter 7

Hardware Implementation

In this chapter, the hardware implementation of the Additive Model is discussed. The starting point is to map it into the Standard Electrical Circuit Interpretation of a recurrent neural network. In this way the characteristic quantities of the circuit are introduced, that are necessary for a proper dimensioning of the electrical components. The result of this process are N_s Kirchhoff's current laws, one per each state variable. In this way, the solution of a dynamical system can be obtained by letting evolve a dynamical circuit. The simulations are performed using the `ngSpice` circuital simulator. In the following, two circuital solutions are presented. The first one is a Behavioral description, in which ideal components are considered that exactly reproduce the relationship defining the dynamical system. Subsequently, the Operational Amplifier is introduced, allowing to lower the description level of the network. In the process some importance is to be given to the components dimensionalization, taking into account the ReRAM devices features. This allows to shed light onto some compatibility problems existing when interfacing operational amplifiers with memristive devices.

7.1 Standard Electrical Circuit Interpretation

The learning capability of recurrent neural networks has aroused the interest of the hardware community thanks to the work done by Hopfield. Ever since his early research [47], he explicitly mapped the dynamical system previously discussed in a high-level description of an analogue circuit. This was then extensively explored through the years in the field of nonlinear circuit theory, especially by Chua and collaborators [56, 57, 75]. In this section the mapping is performed explicitly, in order to introduce the characteristic quantities needed to map the dimensionless equations described so far into well defined circuits. Let us start from the generic formulation of the Additive Model:

$$\frac{du_k}{dt_a} = -u_k + \sum_{j=1}^{N_u} W_{kj} g(u_j) + b_k \quad k = N_u - N_s + 1, \ldots, N_u \tag{7.1}$$

By construction, every term of this equation is dimensionless. The introduction of a characteristic voltage value v_o allows to map the state variable u_k into an equivalent voltage $v_k = v_o u_k$ of an electrical circuit. Therefore, Eq.7.1 can be thought of as the time evolution of the voltage at some nodes of a non-linear resistive network. This corresponds to the introduction of the following

characteristic quantities:

$$\begin{cases} v_k = v_o u_k \\ t = \tau_o t_a \end{cases} \quad \text{where} \quad \begin{cases} [v_o] = [v_k] = V \\ [\tau_o] = [t] = s \end{cases} \tag{7.2}$$

The time constant τ_o can be written as the product of a characteristic capacitance and of a characteristic resistance $\tau_o = R_o C_o$. The substitution of these dimensionless quantities in Eq.7.1 yields the system of equations

$$C_o \frac{dv_k}{dt} = -\frac{v_k}{R_o} + \sum_{j=1}^{N_u} \frac{W_{kj}}{R_o} v_o g(u_j) + \frac{v_o b_k}{R_o} \tag{7.3}$$

For the aim of simplifying the notation, it is better to define the following quantities

$$\begin{cases} G_{kj} = \frac{W_{kj}}{R_o} \\ I_k = \frac{v_o b_k}{R_o} \\ \varphi(v_k) = v_o g(v_k / v_o) \end{cases} \quad \text{where} \quad \begin{cases} [G_{kj}] = [R_k^{-1}] = S \\ [I_k] = [v_o / R_o] = A \\ [\varphi(v_k)] = [v_o] = V \end{cases} \tag{7.4}$$

The Additive Model can then be mapped into the standard electrical circuit interpretation of neural networks, first introduced by Plonsey and Fleming [76]:

$$C_o \frac{dv_k}{dt} = -\frac{v_k}{R_o} + \sum_{j=1}^{N_u} G_{kj} \varphi(v_j) + I_k \tag{7.5}$$

With respect to the schematics proposed in chapter 1, we can now try to understand what is needed for an actual hardware implementation. Figure 7.1 shows a more accurate model of the neuron. First of all, the neuron is connected to all the neighboring ones through synaptic connections, i.e. resistive switching components. These signals are cumulated at a summing node, corresponding to the Soma of the biological neuron. In addition to this, also the bias term has been introduced in the form of a variable resistor connected to a fixed voltage value. This is because all the models considered are equipped with a bias term as a parameter to be learned. This parameter is characteristic of the model and, for this reason, it must be stored in a non-volatile component as well as the synaptic connections. Once summed these signals, the current is fed to an RC circuit, corresponding to the leaky membrane potential. Finally, the voltage drop at the ends of the RC is distorted by a non linear activation function, transforming the average membrane voltage into the corresponding spiking rate. It is worth pointing out that the convention adopted is the one used by `NgSpice`, in which once fixed a direction in a branch, both the voltage and the current are referred to it. This is a high level description of the core of the neuron to be implemented. Nonetheless, it does not accounts for the inputs, that are architecture dependent. While in the Grossberg-Hopfield model the inputs are injected as currents in the input summing node, in the Bengio-Scellier model the input are introduced as a voltage source.
In the following the numerical simulations discussed so far are mapped into hardware simulations. The basic idea here is to design a circuit to implement the kind of dynamical systems described so far. This is to say that, rather than using a generic numerical solver of differential equations as done before, the dynamic system to be solved is mapped in a corresponding netlist. Eventually, a transient simulation is performed in `ngSpice`, finding in this way the steady state of the dynamical system on hardware.

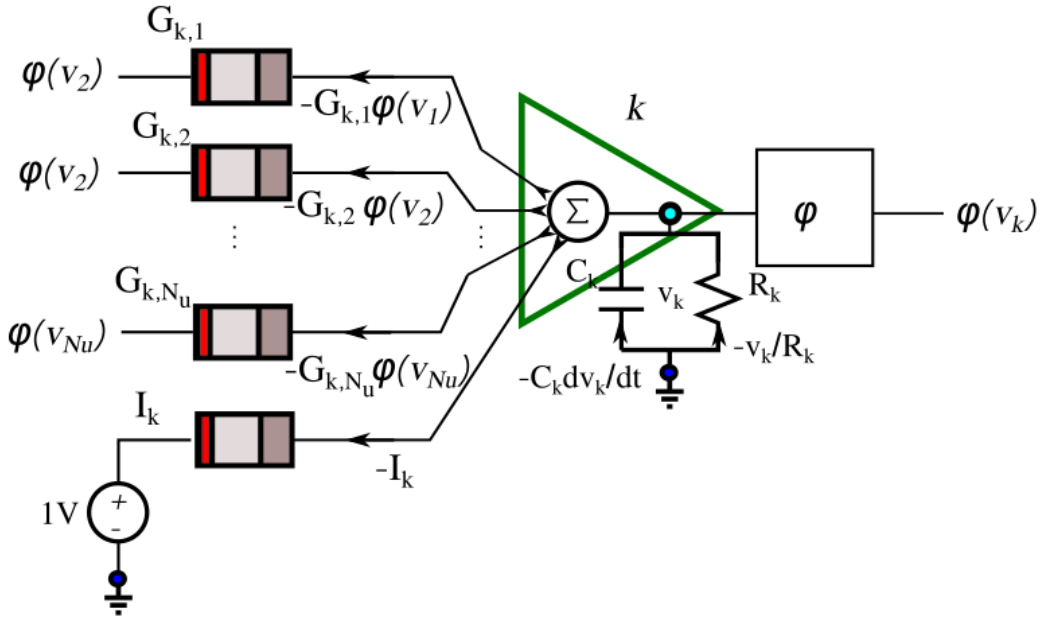


Figure 7.1: Scheme of the Standard Electrical Circuit Interpretation of a recurrent neural network. The signals flowing through the synaptic connections together with a bias term are collected at the input of the neuron. Here, the Soma sums the synaptic contribution to the current effects occurring at the level of the leaky membrane, represented as an RC circuit. Finally, the rate model is completed by finding the rate equivalent of the state variable, corresponding to the membrane potential averaged over time and neurons.

7.2 Behavioral circuit definition

The first thing done was the definition of the network using a behavioral description. This is general and any analogue implementation has to be a non-ideal version of what presented in this section. The selected process amounts to split the building blocks of the network into behavioral primitives, that were connected on the basis of the architecture considered. The goal of the behavioral description is to create the ideal structure of the network. Afterwards, once obtained a working circuit, the ideal building blocks is substituted with non-ideal ones, toward the simulation of a progressively more realistic circuit. In this way, the non-idealities can be introduced by substituting the primitives with a lower level representation of the same building blocks.

7.2.1 Integrator and activation function

Considering Fig.7.1 it is possible to see that the elements to be modelled are two: the integrator and the activation function. The behavioral circuit for the neuron is represented in Fig.7.2. The input is a reference voltage node. This is necessary because, in this way, the outputs of the neighboring neurons can be connected to it and the incoming signal depends uniquely on the state of the previous neuron. The direction of the input current is taken positive outward, so that

its value will be

$$I_{m\leftarrow\partial m} = -\sum_{k=1}^{N_u} G_{mk}\varphi(v_k) - I_m \tag{7.6}$$

The last term is the bias reported in Fig.7.1 while ∂m identifies the neighboring neurons. Once defined the incoming signal, it must be coupled with the dynamics of the state variable. To do so it is sufficient to recognize Eq.7.5 as the KCL of a RC circuit and a current source controlled by all the neighboring neurons. Therefore, a RC circuit is connected to ground and the state variable v_m is chosen to be the voltage drop from ground to the other end of the RC. The voltage controlled source is connected to the same nodes, so as to obtain Eq.7.5.

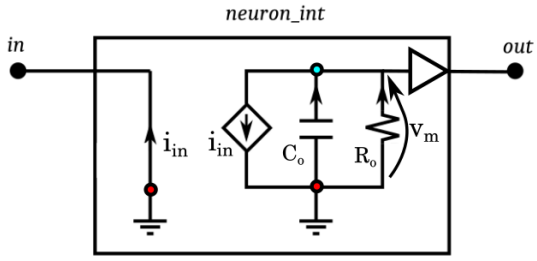


Figure 7.2: Behavioral description of the neuron circuit.

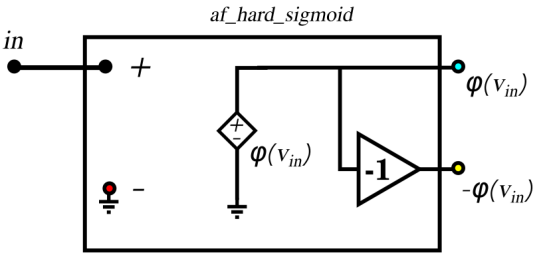


Figure 7.3: Behavioral description of the activation function.

Moving to the non-linearity, the behavioral circuit can be observed in Fig.7.3. The input voltage is used to control a voltage source, defined as the non-linearity $\varphi(\cdot)$. Together with the nonlinear version of the state variable, also the negated version of it is given at the output. The motivation of this will be clearer in the following. In Fig.7.4 it has been reported the simulation done of the output voltage as under an input voltage going from $-0.5v_o$ to $1.5v_o$ The block for the body of the neuron (Fig.7.2) was introduced and a transient analysis of $3\tau_o$ was performed. The time interval was set to $10^{-2}\tau_o$ and the voltage across the capacitor was initialized to $1.5v_o$. The result of this simulation can be observed in Fig.7.5. Figure 7.4 allows to verify the correct functioning of the activation function block. On the other hand, Fig.7.5 allows to verify that connecting the integrator block with the activation function block it is possible to obtain at the outputs of the activation function the correct distortion of the state variable throughout its evolution.

7.2.2 Building the network

Once defined the building blocks, it is now possible to integrate them in a networked structure. Let us start from the state neuron, which is present in both the Bengio-Scellier and the Grossberg-Hopfield networks. The structure implemented has been reported in Fig.7.6. Starting from the left, the memristive crossbar transmits the signal from the output nodes of the surrounding neurons to the state unit considered. As previously described, two resistive switching devices are used, one if the synaptic weight is positive and one if it is negative. When defining the network there are two possibilities: either two neurons should be independent or they need to be connected. In the first case both the switches have to be open. On the contrary, in the second case only one among the two memristors will be connected, depending on the sign of the weight to be programmed. In principle, this is not the only possibility. If one memristor is

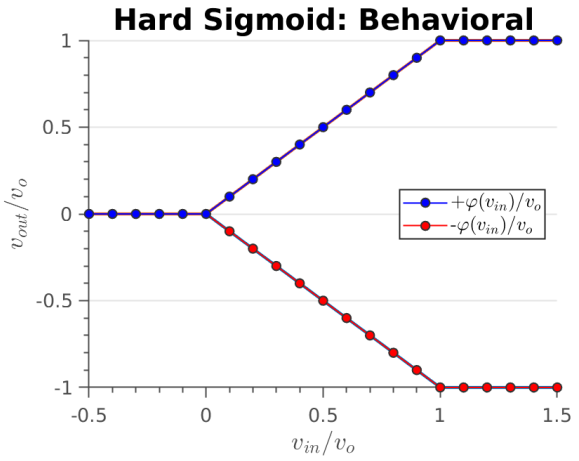


Figure 7.4: DC sweep simulation to verify the correct functioning of the activation function block. The input voltage has been varied in the interval $v_{in} = [-0.5v_o, 1.5v_o]$.

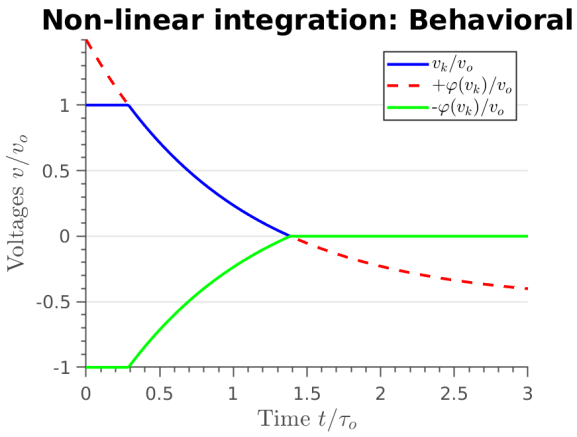


Figure 7.5: Transient simulation of the neuron block connected in series to the activation function block. The simulation has been performed on the interval $t \in [0, 3\tau_o]$

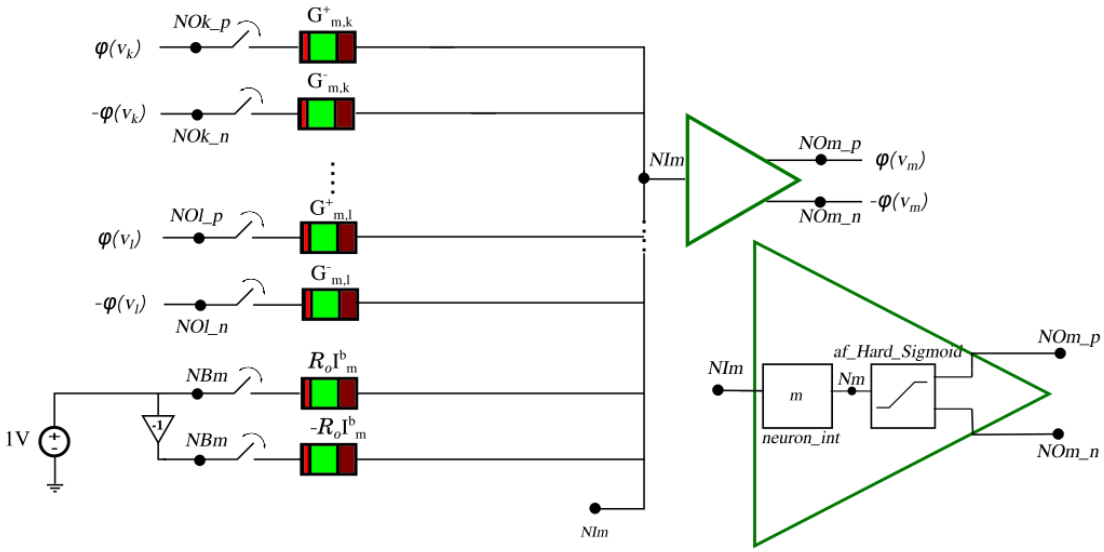


Figure 7.6: Behavioral description of a state node of the circuit.

associated to the positive weight and one to the negative weight there are more possible choices of their conductances to reproduce a desired value of the final parameter. Nonetheless, the choice of using only one memristor is the less ambiguous one, hence easier to implement. A further development would require to substitute the ideal switch with a MOS transistor, corresponding to the 1T1R crossbar described before. Since the model must be non-volatile, the information about the sign should be encoded in an additional binary memory element, in a configuration that could be called the 1R1T1R configuration. This would introduce additional non-idealities, that are not considered in this preliminary analysis. All the synaptic connections enter in the neuron input, which is referred to ground. In this way the current coming from the surrounding

neurons and entering in the node *NIm* (see Fig.7.6) will be

$$I_{NIm\leftarrow\partial NIm} = -I_{NIm\rightarrow\partial NIm} = \sum_{k=1}^{N_u} G_{mk}\varphi(v_k) + I_m^b$$
$$= \sum_{k=1}^{N_u} \delta_{SW^+,ON} G_{mk}^+ \varphi(v_k) - \sum_{k=1}^{N_u} \delta_{SW^-,ON} G_{mk}^- \varphi(v_k) + I_m^b$$

Where the delta functions are used to indicate whether or not one of the two switches is conducting. Since only one of the switches must be conducting, they will be related as $\delta_{SW^+,ON} = 1 - \delta_{SW^-,ON}$.
In all the models discussed so far there is a bias term to be introduced. This is some non-volatile information, to be stored in a resistive switching device as well as any synaptic weight. In the circuit, this has been achieved by connecting to the neuron $1v_o$ or -$1v_o$, accordingly to the sign of the weight stored. This voltage reaches the input of the neuron through a resistive switching device, storing the absolute value of the weight. The current is then injected in the RC parallel of the *neuron_int* block, making the voltage evolve according to

$$C_o \frac{dv_m}{dt} = -\frac{v_m}{R_o} + \sum_{k=1}^{N_u} G_{mk}\varphi(v_k) + I_m^b \tag{7.7}$$

As a final step, the voltage across the RC parallel is taken by the activation function block, which first applies the non-linearity and then inverts the resulting signal. In this way the signals are transmitted to the surrounding neurons, in the process of mutual interaction to which the nodes are subjected.
What is missing in the description is the introduction of the information from the external world, that depends on the architecture chosen. When considering the Grossberg-Hopfield model, the

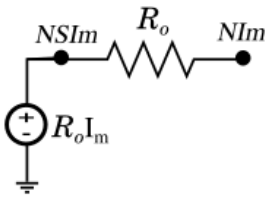


Figure 7.7: Hardware implementation of the input in the Grossberg-Hopfield network.

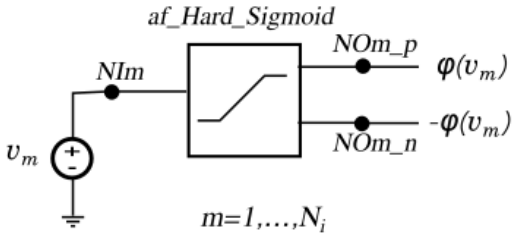


Figure 7.8: Hardware implementation of the input in the Bengio-Scellier network.

input is introduced as an additional bias. To do so, since all the quantities have been controlled in voltage, the clamped input value can be obtained as voltage values at the input. This input must be converted into current by the unit conductance connecting the input of the circuit to the input of the neuron, floating on the bottom left of Fig.7.6. In this way the full Grossberg-Hopfield model is defined

$$C_o \frac{dv_m}{dt} = -\frac{v_m}{R_o} + \sum_{k=1}^{N_u} G_{mk}\varphi(v_k) + I_m^b + I_m \tag{7.8}$$

Instead, for what concerns the Bengio-Scellier model the input is introduced as an additional neuron. However, in this case the membrane voltage is clamped, and one should only apply the

non-linearity to it. This can be observed in Fig.7.8.
Before performing the simulation it is necessary to define the network. Basically, this corresponds to connecting the building blocks described so far in order to produce the correct architecture. Given the weight matrix, the bias vector and the numbers of state, hidden and input neurons the implementation can be easily automated.

7.2.3 Verifying the circuit functioning

Before considering more realistic models for the different components, the correct functioning of these two networks was verified on the previously defined models. As mentioned before, once mapped the dynamic system into equations, it is possible to use it for solving the system of non-linear differential equations. Initially, the circuital solver must give the very same performances as the purely numerical one. Basically, the `MatLAB` solver ode45 was substituted with a `ngSpice` netlist defining an architecture evolving according to the same equations. It is worth noticing that all the amplitudes were expressed in terms of the characteristic quantities. In order to bridge in a simplified way the hardware and the numerical description, the network was first implemented in these units, i.e. imposing $R_o = 1\Omega$, $C_o = 1S$, $\tau_o = 1s$ and $v_o = 1V$.

Classification

The first thing to be done is to define the differential equation solver for the classification problem, that is named odeCIR_CLA. Since the interest is mainly in verifying that the network can actually be used to solve the dynamical system, this solver was used for doing inference. First a network was trained in `MATLAB` for the solution of the MNIST dataset. The training was performed on the whole dataset, using an architecture 196x125x10. The learning rate was fixed to $\eta = 0.05$ and the clamping factor to 0.2. Ten epochs were considered and, as usual, mini batches of 20 data were used. The activation function selected was the hard sigmoid. The resulting accuracy was 96.78% on the test set and 97.5% on the training set. The weight matrix and the bias vector were stored and were used to define the spice netlist. Then, the defined network was tested on 1000 images, reaching an accuracy of 97.8%. The main problem with this network is that each time the odeCIR solver is called, the network needs to be defined and than the integration can be performed. The combination of transferring information, defining the circuit, solving the differential equation and then returning the results makes the whole process extremely slow. For instance, the training just described took around 43 minutes, despite the parallelization on the four cores available on the machine (see Tab.7.1). This approximately corresponds to an equivalent time per data of $\Delta t \simeq 3s$. Given a network of medium dimensions, N_{train} data are needed for the training and N_{test} for the test. Each time both a free phase and a nudged phase need to be performed, bringing to $2N_{train}$ integrations. The test should be performed on a number of N_{test} data both for the training set and the test set. Therefore, for each epoch $2(N_{train} + N_{test})$ simulations need to be performed, bringing to a total simulation time which is $2(N_{train} + N_{test})N_{epc}\Delta t$. This makes it clear the huge computational burden of using circuital simulators for obtaining the evolution of these hardware constructs.

Reconstruction

Similarly, for the reconstruction, each network considered was trained in the `MATLAB` framework up to achieving 99.75% of accuracy both on the training and test data-set. The weights and biases

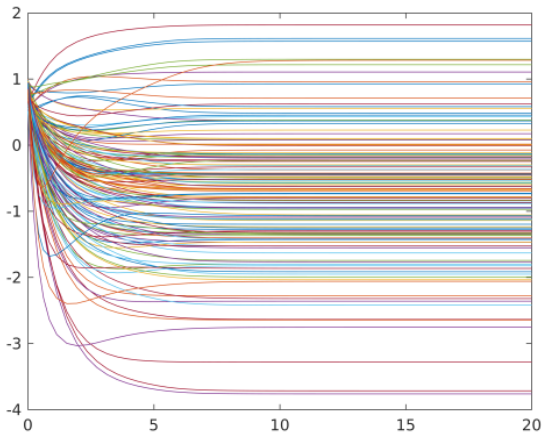


Figure 7.9: MNIST with the hard-sigmoid: Trajectories of the state neurons when performing inference using the behavioral description.

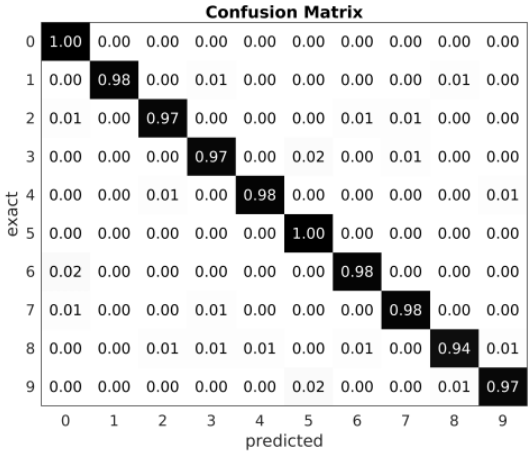


Figure 7.10: MNIST with the hard-sigmoid: Confusion Matrix of the 1000 data considered when performing inference using the behavioral description.

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
Address sizes:	39 bits physical, 48 bits virtual
CPU(s):	8
Thread(s) per core:	2
Core(s) per socket:	4
Socket(s):	1
Vendor ID:	GenuineIntel
Model name:	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
CPU MHz:	800.129
CPU max MHz:	4000,0000
CPU min MHz:	400,0000

Table 7.1: Main features of the machine used for the simulations.

found were then used to define the netlist and from this the correct integration was performed on 100 data, reaching an accuracy of 99% in 100 seconds. One trajectory associated to this simulation can be observed in Fig.7.11.

7.3 Operational Amplifier and Hard Sigmoid

One critical detail for moving to a lower level description of the network is the summing node. It must be at the reference voltage, to guarantee that the voltage drop across the conductance is dictated uniquely by the output of the previous neuron. However, the current cannot be exactly at ground since it must reach the node k, where it sums to the contribution of the RC component modelling the membrane. One possible solution is to use the operational amplifier. As discussed in the previous chapter, both the current entering at its input nodes and the differential voltage at

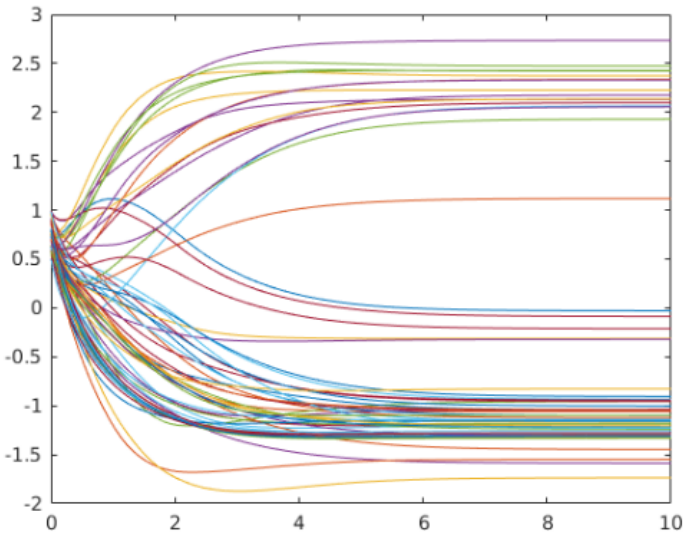


Figure 7.11: Reconstruction with the hard-sigmoid: Convergence of the trajectories for a given input image.

its input nodes can be neglected. In the next sections this property is exploited for building the analogue neuron.

7.3.1 Operational Amplifier Based Primitives

Using this component in a configuration called *integrator*, it is possible to completely represent the body of the neuron, as reported in Fig.7.12. In this way all of the memristors have a node

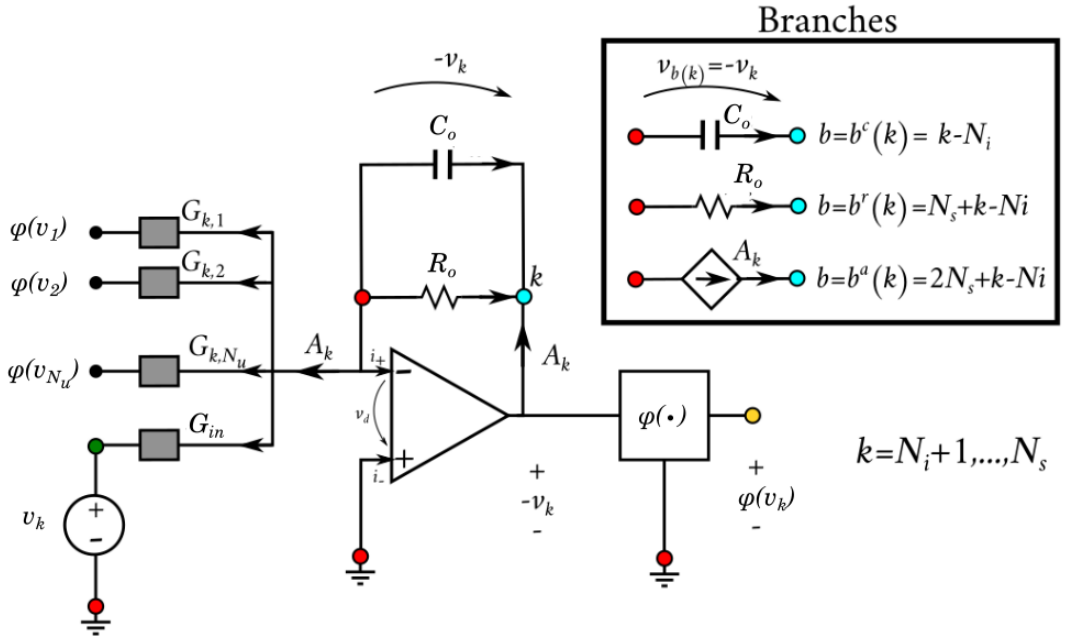


Figure 7.12: Analogue implementation of the neuron block.

at the virtual ground of the operational amplifier and this results into a summation of each contribution to obtain the current A_k.

$$A_k = -\left(\sum_{j=1}^{N_u} G_{kj}\varphi(v_j) + I_k\right) \tag{7.9}$$

Since $i_+ \simeq 0A$, almost all of the current converges at the node shared with the capacitor C_o and the resistor R_o. One essential feature of this circuit is that the reference voltage is not v_k, but $-v_k$. This must be taken into account by the activation function, that was chosen to be the hard sigmoid. A direct application of the non linearity would yield a completely wrong picture. This is because the hard sigmoid $\min(1,\max(0,x))$ is not anti-symmetric, i.e. it is different from $-\min(1,\max(0,-x))$. Therefore, in the following it is used the fact that

$$\min(1,\max(0,x)) = -\max\left(-1,\min\left(0,-x\right)\right) \tag{7.10}$$

Another interesting thing regarding the activation function is that it is now possible to understand why it was given priority to the hard-sigmoid and to the hard-tanh as activation functions. Not only they allow to have a proper saturation of the dynamics, as pointed out by Bengio and Scellier, but it also apparently allows for a simple circuital implementation when using operational amplifiers. In fact, there is a particular configuration of the operational amplifier that exactly reproduces the hard functions and it is called the *voltage follower* configuration (see Fig.7.13). In this section it is discussed the hard-sigmoid but also the hard-tanh could be implemented with this component. The features of the device allow to fix the output voltage

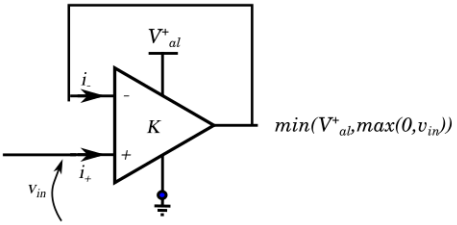


Figure 7.13: Voltage follower Configuration

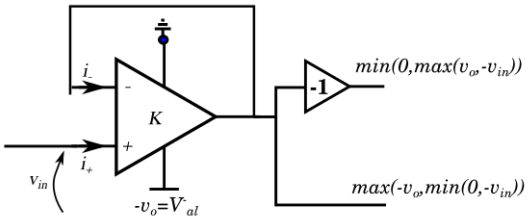


Figure 7.14: Activation function block.

equal to the input one, independently of the current. This is an amplifier with gain 1, having an extremely high input resistance and a relatively low output resistance. In input it absorbs an extremely low amount of current while at the output it is an ideal voltage generator. The feature to be exploited is the saturation of the device to the supply voltages. The output voltage cannot exceed the power supply values. Therefore, it yields a straightforward implementation of the hard functions.

$$V_{out} = min(V_{al}^{+}, max(V_{al}^{-}, v_{in})) \tag{7.11}$$

Nevertheless this block cannot be directly placed after the operational amplifier. As mentioned before the input voltage at the activation function block will be the negated form of the reference voltage. Since both the non-linear version of the voltage and its negation are needed, it is reasonable to first implement the negation and, afterwards, to invert it. This can be theoretically achieved by connecting the positive supply to ground and the negative supply to a properly chosen V_{al}^{-}. Moreover, in order to properly perform the mapping of this function with the adimensional equations it is clear that the supply corresponds to the characteristic voltage: $|V_{al}^{-}| = v_o$. Finally, also the inverting block can be implemented using operational amplifiers. The complete operational amplifier based hard sigmoid can be observed in Fig.7.15.

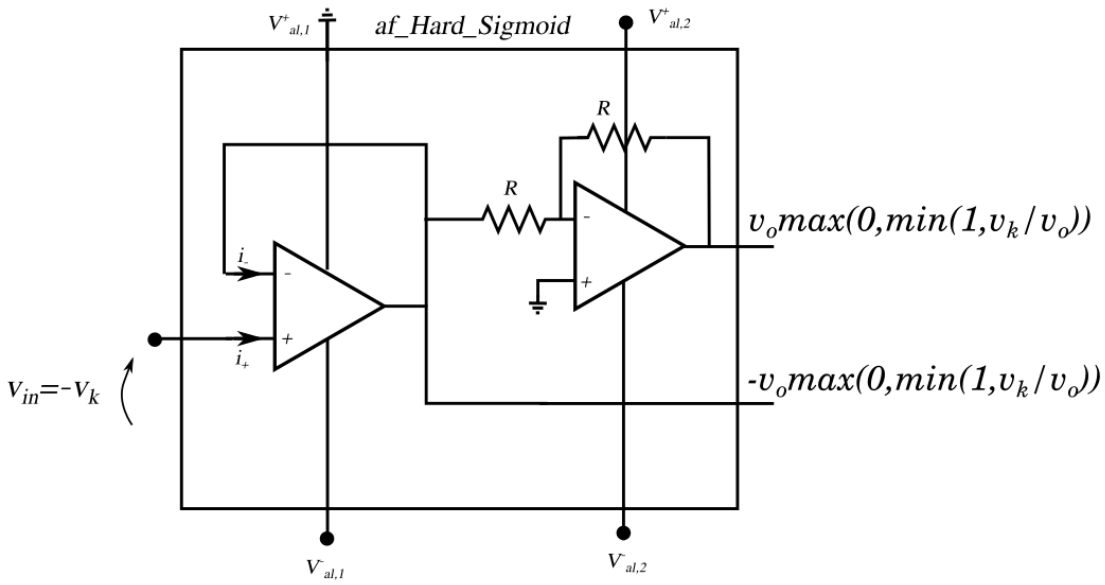


Figure 7.15: Operational amplifier based implementation of the hard sigmoid.

7.3.2 Some dimensional considerations

Let us consider the VCM device characterized by Giovinazzo et al. [35]. When discussing the programming of the memristive device, they showed that using a number of pulses lower than 30, it is possible to achieve a linear variation of the conductance with the number of pulses, which yields a particularly simple way of achieving well spaced and defined conductance states. These states belong to the interval $G \in [1,32]\mu S$ and they called this interval the *subthreshold region*. This allows to identify the order of magnitude of other quantities, such as the membrane resistance. In fact, if a numerical simulation finds the dimensionless weights to be $|W| \leq \omega_{abs}$, the characteristic resistance must be chosen to be

$$R_o = \frac{\omega_{abs}}{G_{max}} \tag{7.12}$$

to put some numbers, in the case of subthreshold region of the HfO_x valence change memory, $G_{max} = 32\mu S$. Previously, it was noticed that the weight parameters of the MNIST architecture 196x125x10 could be pruned to $\omega_{abs} \simeq 0.3$. Therefore, in this specific case $R_o \simeq \frac{0.3}{30\mu S} = 10k\Omega$ could be a good choice for the resistance. This being fixed, it is then possible to tune the simulation time by modifying the capacitor. For instance, in order to achieve a characteristic time $\tau_o = 1ms$, compatible with the proper functioning of many operational amplifiers, the capacitance should be set to $C_o = 100nF$. This value seems reasonable if one compares this characteristic time with the typical slew rate of the operational amplifiers, which is generally around $10V/\mu s$. In order to have a faster circuit one could further reduce this quantity. Nonetheless, additional considerations should be done on the effect of parasitic capacitors. In Fig.6.5 these capacitors have been introduced. These appear mainly because the internal structure of the operational amplifier is a network of transistors, that present capacitive effect as their working principle. These capacitors will interact with other capacitive terms, such as the one in the retroaction of the neuron. Considering the fact that these capacitors are generally of the order of few pF, the capacitor of the neuron should not be reduced too much, so that the capacitive effect dominating the dynamics, i.e. the slowest one, is the one associated to the integration of the neuron.

Reducing the Parasitic Currents and voltages

Another reason why one should not reduce too much the voltages of the system comes from the domain of the parasitic currents and voltages present in operational amplifier based circuits. The reason why reducing the maximum voltage to the order of tens of millivolts is a bad choice can be understood by considering a more precise static model of the operational amplifier. In

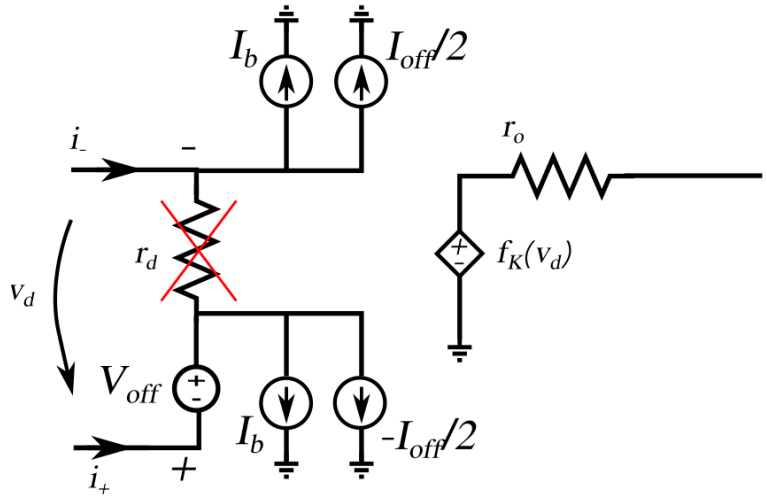


Figure 7.16: Operational amplifier model including the offset terms.

Fig.7.16 and in Fig.6.5 are respectively reported the static and dynamic model for the operational amplifier. First of all the currents entering in the device cannot be neglected because the input resistance is not rigorously infinite. To take this into account, the resistance and any other current leakage can be substituted with current generators, in which the input currents are decoupled into a bias term $I_{bias} = (i_+ + i_-)/2$ and an offset term $I_{off} = |i_+ - i_-|$. Additionally, an offset voltage is added. This takes into account the voltage value v_d must overcome before a change in it is appreciable at the output. Let us now use this information to deepen the quantities that are expected to appear in the device. Previously it was highlighted that the current associated to the bias term is $I_k = \frac{b_k v_o}{R}$. Considering the choices done so far and the fact that $|b| \leq \omega_{abs}$, it is possible to obtain

$$|I_k| \leq 10^{-4} S \omega_{abs} v_o \tag{7.13}$$

by selecting $v_o = 100mV$, this becomes $|I_k| \leq 10^{-5} \omega_{abs} A$ and in the aforementioned MNIST case, $|I_k| \leq 3\mu A$. From this, it is possible to understand a reason why the characteristic voltage should not be taken too low. Let us start from considering the offset voltage. This is an unpredictable quantity, the value of which is determined by the internal properties of the device. Therefore, it would be desirable to have the effects of the voltages in the system to be dominating over this one. To make an example, let us consider the operational amplifier family TL081. The worst case offset voltage is $|V^{wc}_{offset}| = 5mV$, imposing to take the highest possible reference voltage for the system. In addition to this, the offset current and the bias currents have a node in common with the bias term. In the worst case $|I^{wc}_{bias}| = |I^{wc}_{offset}| = 5nA$, but for some operational amplifiers they could raise up to $500nA$. Naturally, the best operational amplifier must be selected so as to minimize these quantities, but to remain general in the discussion, it is clear that the characteristic voltage should be selected as the highest voltage not changing the memristive state, in order to avoid the bias and offset quantities to become detrimental. In the following the

focus is put on the model presented in Fig.6.4. In particular, the input resistance is chosen to be almost ideal. Since its value generally varies from $10^7\Omega$ to $10^{12}\Omega$, it was set to the highest possible value, that in dimensionless units reads $R_d^{(a)} = 10^{12}/R_o$. In this way the discussion can be limited to the output non-idealities. For what concerns the gain, the worst case is generally around $K = 10V/mV = 10^4$ while good values can reach 10^6. The last quantity to be considered is the output resistance. In the ideal behavior it is considered to be zero. However, in some operational amplifier it can be around 150Ω, that despite being relatively small with respect to other resistances, may be not negligible in terms of performances. Therefore, the adimensional resistance $R_{out}^{(a)} = \frac{R_{out}}{R_o}$ is considered, ranging from zero to $R_o^{(a)} = 0.015$ or higher.
The last quantities to be commented are the resistances in the inverting amplifier. The circuit was investigated under the strong assumption that the electrical quantities overcome the offsets in the circuit. In order to guarantee this, let us try to optimize the resistances to obtain this behavior. With respect to the model of the operational amplifier in Fig.6.4, let us impose the input voltage of the inverting amplifier to vanish, in order to determine the output contributions due to the offsets. The output voltage due to the offsets are partially induced by the currents flowing in the parallel of two equal resistors $V_u^{(i)}|_{off} = \frac{R}{2}\left(I_b + \frac{I_{offset}}{2}\right)$ and partially due to the offset voltage $V_u^{(v)}|_{off} = V_{off}\left(1 + \frac{R}{R}\right) = 2V_{off}$. This implies that the offset voltage cannot be limited. In fact, the two resistances must be equal to have an inverting amplifier. Hence, this also fixes the gain of the amplifier at the non-inverting node, where the offset voltage is present. For this reason, the only thing that can be done is to limit the contribution of the offset current, and this yields the requirement $V_u^{(i)}|_{off} \ll V_u^{(v)}|_{off}$, yielding the condition

$$R \ll \frac{4V_{off}}{I_b + \frac{I_{offset}}{2}} \tag{7.14}$$

Using the worst case values $I_{off} \sim I_b \sim 500nA$ and the typical offset voltage of around $1mV$, one obtains $R < 5.3k\Omega$. Considering the fact that extreme conditions were considered, which are almost impossible to be obtained, $R = 1k\Omega$ is considered to be a good value but also $R < 5.3k\Omega$ could be a safe choice, considering the highly pessimistic worst case values selected. In terms of dimensionless quantities, $R = 1k\Omega$ corresponds to $R^{(a)} = 0.1$.

7.3.3 Operational Amplifier based network and non idealities

As mentioned before, the input resistance was imposed to be $R_d = 10^{12}\Omega$, i.e. $R_d^{(a)} = 10^8$. The state neuron was defined as the block represented in Fig.7.17 and its integration capability was tested when initializing the voltage drop across the capacitor First, the ideal conditions were imposed, choosing $K = 10^6$ and $R_o = 0\Omega$. The capacitor was initialized so that the voltage drop from the output of the integrator to the virtual ground was fixed to $1.5v_o$. This is different with respect to what done in the behavioral circuit. In fact, in that case the voltage at the node not connected to ground was the state variable while here the state variable is the negation of the same quantity. In Fig.7.18 the essential variables are reported: they are the state variable and its filtered version, both when passing through the saturating voltage follower and when inverted.

For each pair of values a DC sweep analysis was performed, varying the input voltage from $-1.5v_o$ to $0.5v_o$. Then, the positive output was plotted as a function of $-V_{in}/v_o$ since,

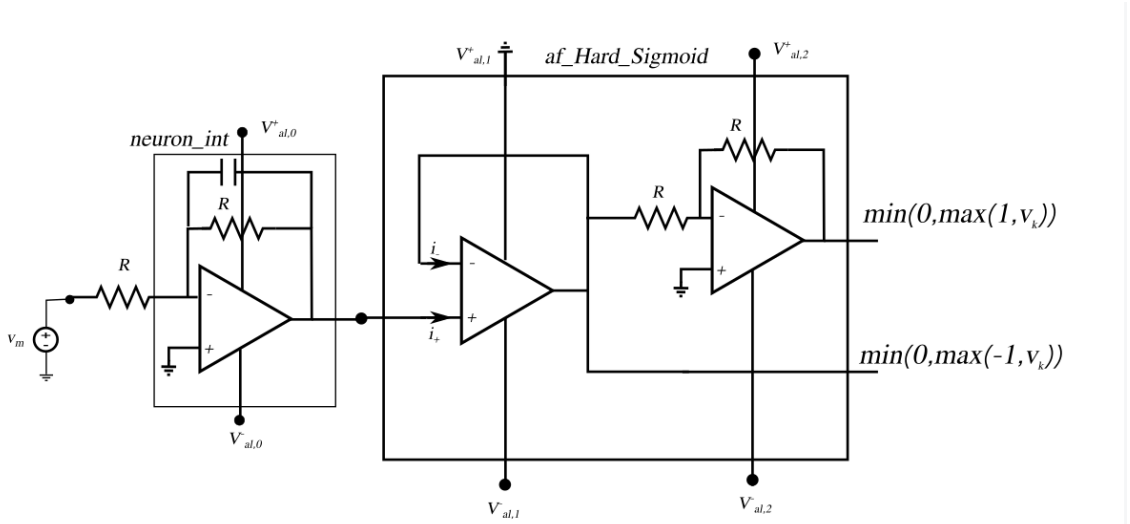


Figure 7.17: Hard sigmoid based neuron implementation using operational amplifiers.

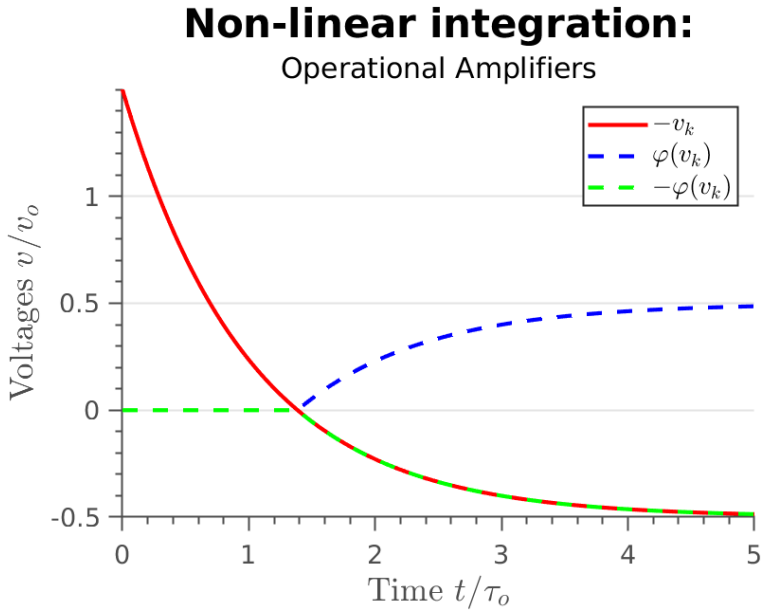


Figure 7.18: Verification of the proper functioning of the neuron primitives defined using the operational amplifiers.

as mentioned before, this is the actual state variable of the system. As it is possible to evince, the variation of the gain factor introduces a variation of the slope around the bias value and a smoothing of the saturation. On the other hand, the increase of the output resistance has an evident consequence on the saturation value, since what saturates is the $f_K(v_d)$ voltage controlled source. The presence of the output resistance imposes a non negligible drop across it, inducing a lowering of the output voltage of the operational amplifier. The need for the inverting block introduces an additional non-ideality. This results in a further drop of the voltage, so that the saturation value will be further decreased.

Interestingly, in order to see considerable effects at the output of the activation function one

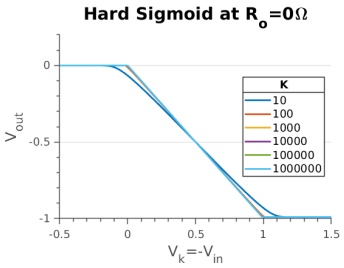


Figure 7.19: Negative output of the state neuron: $R_o = 0\Omega$ and varying K.

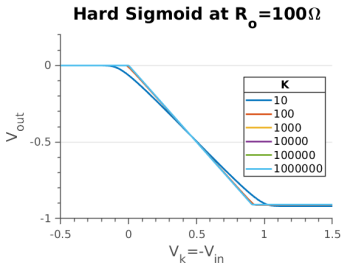


Figure 7.20: Negative output of the state neuron: $R_o = 100\Omega$ and varying K.

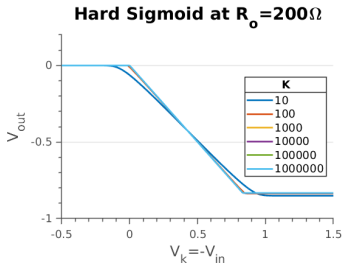


Figure 7.21: Negative output of the state neuron: $R_o = 200\Omega$ and varying K.

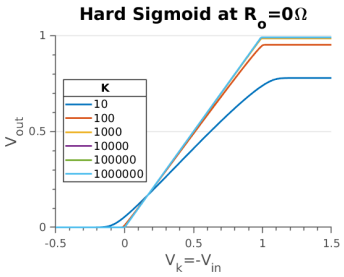


Figure 7.22: Positive output of the state neuron: $R_o = 0\Omega$ and varying K.

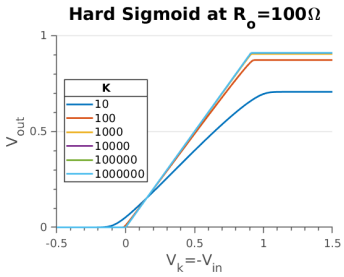


Figure 7.23: Positive output of the state neuron: $R_o = 100\Omega$ and varying K.

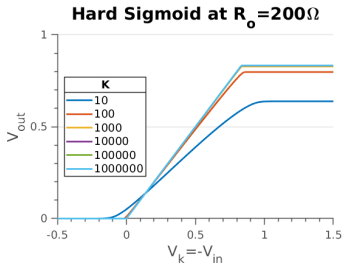


Figure 7.24: Positive output of the state neuron: $R_o = 200\Omega$ and varying K.

must use ridiculously bad amplifiers and high resistance values. Naturally one should pay attention to the possibly detrimental consequences when considering deep networks, but intuitively the non idealities are just slightly modifying the activation function. Since a multiplicative factor can be easily embedded in the v_o value, it is reasonable to assume that this non-ideality could be easily be accounted for.

XOR problem

In order to verify the applicability of this device, the here defined network was used to do inference. The chosen problem was the XOR classification, solved using the $2 \times 5 \times 1$ network. The network was used directly to do inference, varying progressively the degree of non ideality. In particular, the gain factor was ranged from $K = 1$ to $K = 10^6$ while the output resistance was varied from 0Ω to 600Ω, collecting each time the accuracy of the network. The interesting thing about the XOR function is that the prediction is binary. Therefore, once the output node has converged to the value y^∞, the prediction is encoded in the quantity $g(y)$. The truth class can be obtained by computing the inverse $1 - g(y)$ and by comparing the two, in order to determine if the output is closer to a zero or to a 1. For this reason, if the network fails while doing inference it must be due to a detrimental alteration of the vector field in which the system evolves. The influence of the non idealities on the activation function are propagated via the mutual interaction of the nodes, and can lead to wrong predictions of the network. However, one should notice that the errors start to appear only for extremely non ideal cases. Considering that in a normal operational amplifier the output resistance is generally equal to 150Ω at worst and that a gain factor equal to 10^4 is generally considered the lowest achievable value, the network seems to be

$R_o[\Omega]$	$K = 10^6$	$K = 10^5$	$K = 10^4$	$K = 10^3$	$K = 10^2$	$K = 10^1$	$K = 10^0$
0	1	1	1	1	1	0.525	0.525
50	1	1	1	1	1	0.525	0.525
150	1	1	1	1	1	0.525	0.525
300	-	-	1	1	1	0.525	0.525
450	-	1	1	1	0.865	0.525	0.525
600	1	1	1	0.995	0.78	0.525	0.495

Table 7.2: Performances of the hardware implementation of the hard sigmoid when used to do inference on the XOR problem.

stable to these non idealities. More precisely, a network trained on the XOR problem with an ideal hard sigmoid is not affected neither by the output resistance nor by the gain factor. The problem of this conclusion, however, is that it relies on the assumption that the power supply of the operational amplifier is such that $V_{al}^{+} - V_{al}^{-} = 1V$. In particular, the idea is to exploit the saturation capability of this device to implement the non linearity, which is the main quantity affected by the non idealities. This allows to understand the importance of finding an alternative to the operational amplifier for the implementation of the activation function. In fact, this would allow to decouple the circuit from the aforementioned non-idealities. This is to say that the detrimental effects primarily originated from the fact that they directly enter in the alteration of the activation function. It becomes therefore clear the need for alternatives for removing the effect of the non-idealities.

7.3.4 The Retention Problem

One major issue when it comes to the choice of dimensional parameters is the amplitude of the voltage drops across the memristors. Luckily, the voltage drop across a memristor is always bounded by the non-linear function, that was imposed to be the hard sigmoid. As previously mentioned, the dimensional version of the non-linear function is $\varphi(v_k) = v_o g(v_k/v_o)$, so that $\varphi(v_k) \in [0, v_o]$. This brings us to the first compatibility constraint introduced by the hard sigmoid activation function on the architecture. The parameters of the RNN to be defined should not vary in time, since the model should be non-volatile. It would therefore be desirable if the voltage drop at their end did not introduce substantial variations. The problem of this requirement is that the vast majority of the resistive switching devices is capable of keeping memory of a memconductance value only if not stimulated with voltages higher than around $100mV$. This suggests that the characteristic voltage should be imposed to be $v_o = 100mV$, so that the voltage drop can be limited by exploiting the saturation capability of the operational amplifier. Unfortunately, this is in contrast with the physical realization of a real operational amplifier, that requires power supplies of at least $1V$ to be properly working. This is clearly not compatible with the multistate retention of practically any memristive device currently available. The easiest way to circumvent the problem is to get rid of the multistate feature and use the device as a binarized memory, as already done in the Spiking neural network community [77]. In Fig.7.25 it is reported the new way in which the resistive switching materials should be used for storing the weights. Each memristor is either in the High Resistance State (HRS) or in the Low Resistance State (LRS). In this way, N_m devices will allow for $N_m + 1$ conductance states. When using the classical crossbars and the multistate memristors, the simplest thing to do is to

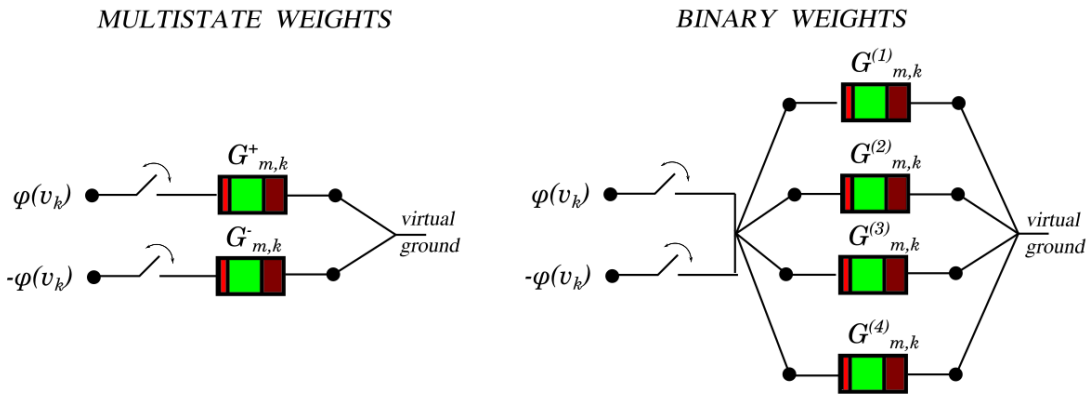


Figure 7.25: On the left the usual implementation of the synaptic weights, based on the assumption that the resistance value can be tuned at will. On the right the implementation of a multistate conductance block as the parallel of binary weights.

assign a positive value to one memristor and negative one to the other. In this new approach, this would lead to an overwhelming number of devices required. Therefore, the same memristors are used both in case of positive and negative weight and their sign is memorized as the switch that should be conducting. The true advantage of creating the multistate by putting in parallel more than one binary weight is the fact that the retention of the HRS and LRS in the ReRAMs is considerably higher than the one of the multistates [35]. Moreover, the memristors remain in the corresponding state up to voltage values of the order of $1V$. Therefore, this is the only way to go when dealing with saturating operational amplifiers for the definition of the activation function. One might criticize the fact that this would introduce a too drastic increase of space with respect to the double crossbar solution. Despite the correctness of this observation, one should also note that the numerical simulations that currently simulate the RNN work exploiting binary weights. Implementing these networks on a crossbar in which each node can store more than one weight would introduce two advances. First of all a considerable increase in compactness. Secondly the possibility to reduce a matrix vector multiplication, which is $\mathcal{O}(N^2)$ to a $\mathcal{O}(1)$ parallel in memory operation. Despite their power, recurrent neural networks are not particularly popular also due to the long simulation times required by them for a single simulation, which is of the order of $\mathcal{O}(N^2T)$, with T the simulation time. If the use of operational amplifier based neurons does not allow to store multiple bits in the same memristive device, only one of the two advantages is lost. It could still be possible to use an amount of memory comparable with the one currently used by a Von Neumann architecture with the great advantage in terms of computational speed, that is the major limitation to the simulation of RNNs. An hardware implementation, however area demanding, would still reduce the computational time from $\mathcal{O}(N^2T)$ to $\mathcal{O}(T)$, considerably increasing the applicability of RNNs. The way in which a single crossbar can be converted to this new version is reported in Fig.7.26, where each dot corresponds to a binary ReRAM. This should not be considered as the definitive solution to the multistate retention. Any advancement on this property would be dramatically beneficial in terms of area integration. Nonetheless, this solution is compatible with this future update and should be preferable in an early implementation, considering the fact that it would still work effectively as an AI hardware accelerator.

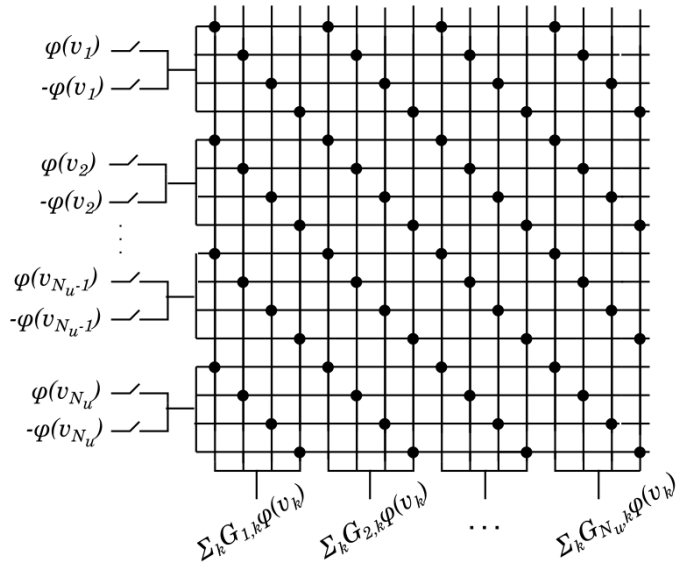


Figure 7.26: Schematic representation of an hardware accelerator for RNN systems using binary weights.

7.4 Conclusions

This chapter aims at discussing the first details related to the hardware implementation. First, the Additive model is mapped to a well defined system of Kirchhoff's current laws, introducing in this way the characteristic quantities of the network. Then, two preliminary versions of the implementation are discussed. The first one is purely behavioral and its aim is mainly to introduce the desired architecture with ideal primitives. The functioning of the network is verified both on the classification and on the reconstruction task. Later, a possible implementation for the hard sigmoid and for the neuron are discussed. In both the blocks the implementation is based on operational amplifiers and the choice of the electronic components is discussed. In particular, if the maximum weight learned in the numerical simulation is ω_{max} and the maximum programmable conductance is G_{lrs}, the characteristic resistance reads

$$R_o \doteq \frac{\omega_{max}}{G_{lrs}} \tag{7.15}$$

Additionally, in order to guarantee to work with time constants compatible with the working mechanisms of the operational amplifier, it is argued that the time constant should be fixed once and for all to $1ms$, from which the characteristic capacitance reads

$$C_o \doteq \frac{\tau_o}{R_o} \tag{7.16}$$

In all the cases it is required to use operational amplifiers in their inverting configuration and it is shown that the corresponding resistances should be chosen to be

$$R \leq 5.3k\Omega \tag{7.17}$$

Finally, for what concerns the characteristic voltage its choice must be dictated by the properties of the circuit. In the case of operational amplifier based system, the choice is naturally the

absolute value of the power supply. The use of operational amplifier automatically induce the presence of non-idealities and, according to the proposed analysis, the main ones are argued to be related to a variation in the saturation value of the amplifier. This could be accounted for by properly re-defining the characteristic voltage. Nonetheless, this implementation is ultimately not compatible with the resistive switching devices considered due to the too high voltage values required by the operational amplifier to properly work, with respect to the ones that would avoid a re-programming of the memristive devices. Moreover, it is noticed that an alternative way to avoid the reprogramming could be to go back to binary weights, momentarily abandoning the multi-state potentiality of the memristors. this is inspired by the work done in the Spiking neural network community. The reason why this could be interesting is that the RNN community needs efficient hardware for speeding up the calculation and a binary-based implementation of the multi-states would still guarantee a considerable acceleration of the simulations done on RNN.

Chapter 8

Diodes-based networks

In this chapter, a network implementation based on the hard tanh activation function is proposed. This is based on two anti-parallel diodes, analogous to the ones described by Kendall et al. [60]. The starting point is a brief description of the activation function, using the input node of the Bengio-Scellier network as a working table. Next, attention is drawn to the state neuron, showing how the encoding of the diode in the previously described leaky integrator can be beneficial in terms of time. Finally, the results obtained in the inference phase on the XOR and on the MNIST problem are discussed.

8.1 Diode and Input of the Bengio-Scellier Model

The diode is a non-linear component whose I-V characteristic reads

$$i_D = I_o\left(e^{\frac{v_D}{\eta V_T}} - 1\right) \tag{8.1}$$

Fig.8.1 shows the circuital symbol and the I-V plot of the *Default* diode model implemented in `ngSpice`. The red dashed line corresponds to the piecewise approximation of the diode: the current is assumed to be zero up to the voltage V_γ, starting from which the diode behaves as a voltage generator at the voltage $V_\gamma \simeq 0.7V$. Following what done by Kendall et al. [60],

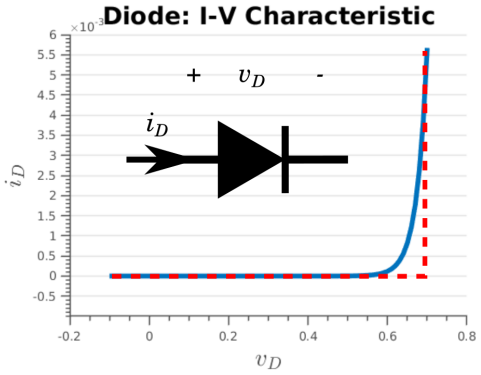


Figure 8.1: Diode: circuit symbol and I-V characteristic.

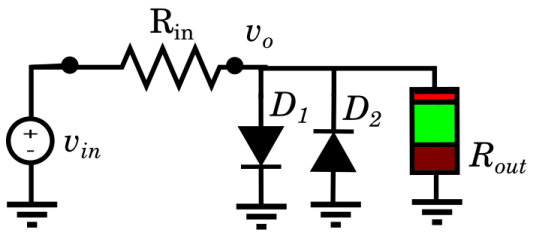


Figure 8.2: Kendall activation function on a simple network.

two anti-parallel diodes are used for the activation function (see Fig.8.2). As later shown, this corresponds to an implementation of the hard hyperbolic tangent. In fact, the KCL at the node shared by the two diodes reads

$$\frac{v_{in}-v_o}{R_{in}} = 2I_o \sinh\frac{v_o}{\eta V_T} + \frac{v_o}{R_{out}} \tag{8.2}$$

where R_{in} is the resistance of the real voltage source present at the input and R_{out} corresponds to the parallel of the memristive connections with the post-synaptic neurons. For what concerns R_{out}, this resistor is generally higher than $1k\Omega$ and, consequently, much higher than a reasonable value for R_{in}, that was taken to be equal to $R_{in} = 50\Omega$. Therefore, the Kirchhoff's current law becomes

$$v_{in} - v_o \simeq 2R_{in}I_o \sinh\frac{v_o}{\eta V_T} \tag{8.3}$$

This is a trascendental equation and an exact analytical solution cannot be found. Nonetheless, it is ameanable for a graphical solution, that is sketched in Fig.8.3. Once again, the piecewise

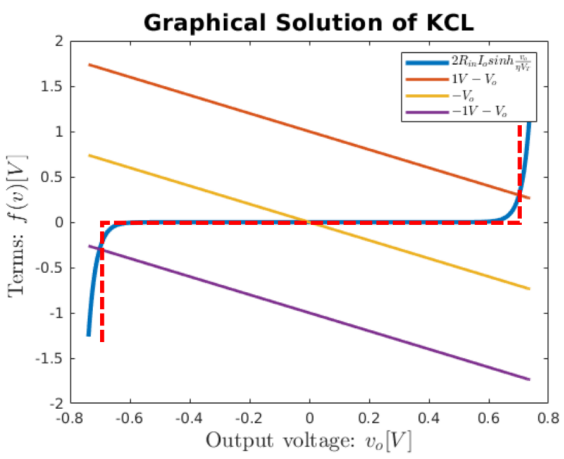


Figure 8.3: Graphical solution of the KCL for the Kendall activation function.

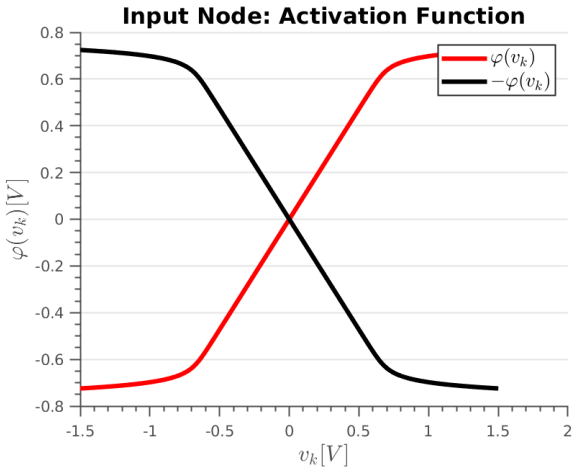


Figure 8.4: Diodes base implementation of the hyperbolic tangent.

approximation of the diode-related quantities is represented in dashed red. In this way, it is clear that a good approximation for the solution is

$$\begin{aligned} v_o = \varphi(v_{in}) &= \min(V_\gamma, \max(-V_\gamma, v_{in})) = \\ &= V_\gamma \min(1, \max(-1, v_{in}/V_\gamma)) = \\ &= V_\gamma g(v_{in}/V_\gamma) = V_\gamma g(u_{in}) \end{aligned}$$

Where $g(\cdot)$ is the hard hyperbolic tangent considered in the previous chapters while the natural choice for the characteristic voltage now becomes V_γ. Just as it was previously done for the hard sigmoid, not only $\varphi(v_k)$ is to be provided but also $-\varphi(v_k)$. For the inversion of the signal the solution adopted was the inverting configuration of the operational amplifier, just as presented in Fig.7.15.

Before proceeding, it is worth to make a remark on the adopted terminology, in light of the simulation results of the previous chapters. Rigorously, the hard functions are defined as

$$g(x;\sigma_\uparrow,\sigma_\downarrow) \doteq \min(\sigma_\uparrow, \max(x,\sigma_\downarrow)) \tag{8.4}$$

In the previous chapters it was shown that, if a rigorously hard function can do regression, also its continuous and differentiable equivalent can do the same. Additionally, any analogue implementation of an activation function, however close to the desired one, must be continuous and differentiable. For this reason, when in the following hard functions are considered, the reader should know that we are referring to a function whose first derivative is well defined on the whole domain.

8.1.1 State Neuron

This section is aimed at discussing the implementation of the state neuron characterized by the hard hyperbolic tangent. One possibility would be to do the same thing done for the input node on the network proposed in Fig.7.12. This would imply to introduce at the output of the integrator the block of the hard-tanh, composed of a 50Ω resistance followed by two anti-parallel diodes, connected to ground. In this way, the node shared by the diodes and the resistance would be forced to saturate any time its voltage value tried to exit from the interval $[-V_\gamma, V_\gamma]$. The proof of this is exactly equal to the one done for the input node, with the only difference that the voltage source is now time dependent. It is worth noting that the voltage at the output of the integrator is always the state voltage with inverted sign. Therefore, thanks to the odd parity of the hard-tanh, the node of the diode corresponds to $-\varphi(v_k)$. The positive version of the same quantity can be obtained by means of the same inverting amplifier defined for the hard-sigmoid. As usual, the capacitor was charged in order to guarantee a dynamics exploring both the linear

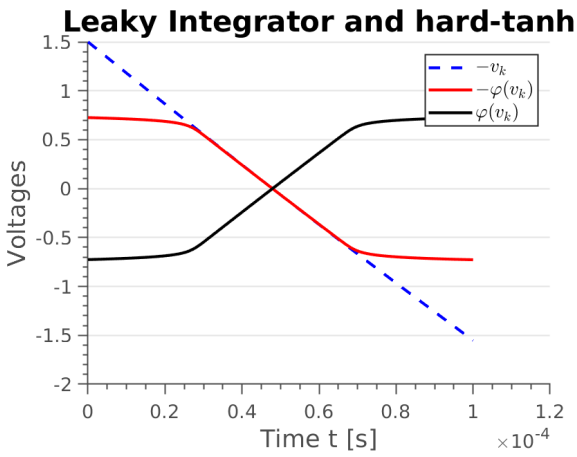


Figure 8.5: Testing the functioning of the neuron block with the Kendall activation function at the output.

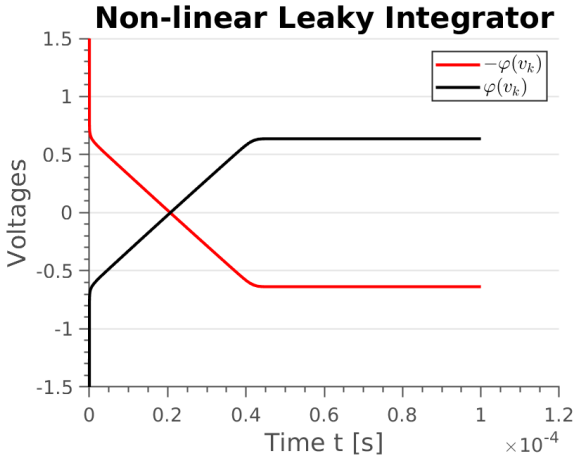


Figure 8.6: Testing the functioning of the neuron block with the Kendall activation function in the feedback loop.

region and the saturation ones. The correct functioning of the activation function block can be observed in Fig.8.5. At this point, one possibility to go further would be to modify the code and to test the correct functioning of the network. Nonetheless, this network, as well as all of the networks considered so far, has one striking feature: the evolution of the state variable proceeds even after the saturation of the corresponding rate. This is unnecessary because the way the rest of the network perceives that voltage is only as its rate equivalent. For this reason, one might point out that the real state variable of the system is $\varphi(v_k)$, ant it would be desirable to remove the unnecessary evolution of the v_k outside the saturation region. In order to achieve this the

anti-parallel diodes were embedded in the integrator, as shown in Fig.8.7, that from now on is named the *Non-Linear Leaky Integrator*. The associated characteristic trajectory is reported in Fig.8.5. In the next section the validity of this circuital solution is mathematically motivated, in light of the working principles of the parametric model.

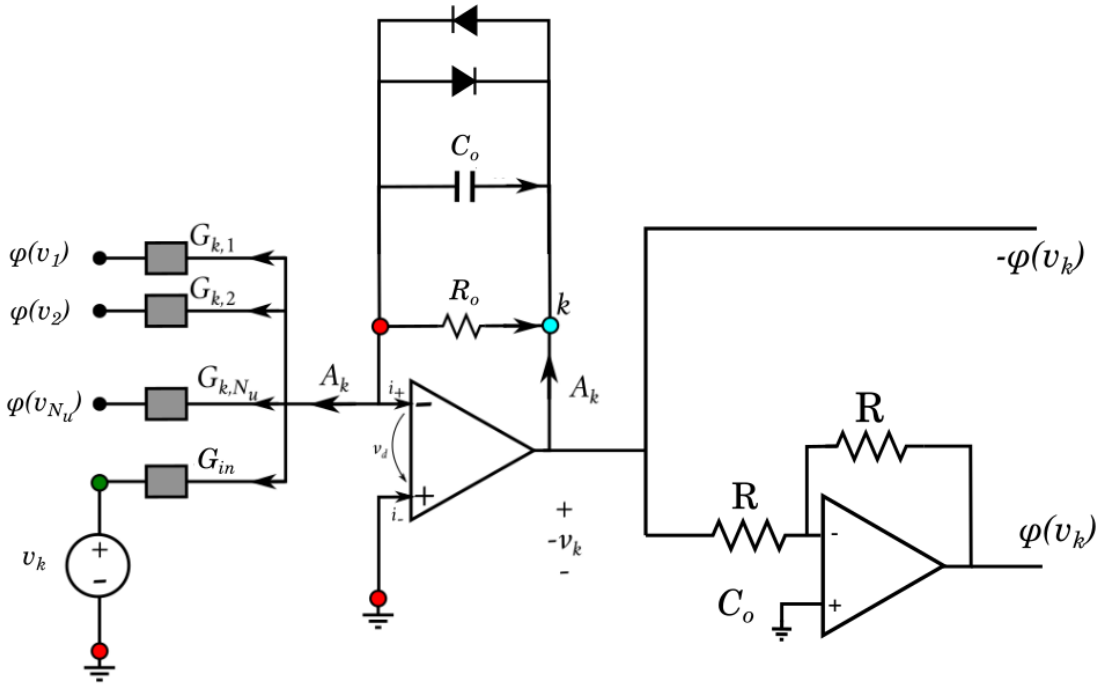


Figure 8.7: Non-linear Leaky integrator: Neuron block built up embedding the Kendall-like activation function in the feedback loop of the operational amplifier.

8.1.2 Rate as the state variable in hard functions

The advantage of the circuital solution reported in Fig.8.7 is both in terms of area and of simulation time. The gain in terms of area is due to the fact that, in the alternative, the 50Ω resistor cannot be removed. Doing this would be the equivalent of introducing two voltage sources in parallel, that is not feasible. The 50Ω resistor allows both the operational amplifier and the diodes to fix their value at the corresponding node, hosting the remaining voltage difference. On the contrary, the anti parallel diodes in the loop of the amplifier are connected to virtual ground and contribute to the definition of the output node. For this reason, no resistance is needed. In addition to this an operational amplifier has been substituted with two diodes, implying a dramatic reduction of the needed area. Another considerable advantage of this solution is in terms of simulation time. Forcing the voltage to be directly bounded in the only interval in which its change can be appreciable allows to speed up the convergence time, as reported in Fig.8.6. To understand this, let us write the Kirchhoff's current law for the generic node k of this system:

$$C_o \frac{dv_k}{dt} = -\frac{v_k}{R_o} - A_k - 2I_o \sinh \frac{v_k}{\eta V_T} \tag{8.5}$$

this corresponds to the introduction of an attractive exponential field in the region outside the hyper-cube $[-V_\gamma, V_\gamma]^{\otimes N_s}$. In fact, if the voltage triest to exit from the boundaries, the dynamical system becomes

$$\begin{cases} v_k > V_\gamma \rightarrow C_o \frac{dv_k}{dt} \simeq -I_o e^{\frac{v_k}{\eta V_T}} \\ v_k < -V_\gamma \rightarrow C_o \frac{dv_k}{dt} \simeq I_o e^{-\frac{v_k}{\eta V_T}} \end{cases} \tag{8.6}$$

In words, whenever the capacitor stores a charge that corresponds to a voltage outside the allowed range, one of the two diodes switch on, exponentially depleting the capacitor of the excess charge. In terms of dynamical systems this corresponds to the introduction of an exponentially strong attracting velocity vector field. This allows to substitute the dynamical system in Eq.8.5 with the following one

$$C_o \frac{d\varphi(v_k)}{dt} = -\frac{\varphi(v_k)}{R_o} - A_k \tag{8.7}$$

with the only caution of restating that, in analogue electronics, almost any function is a continuous and differentiable version of the ideal one. The hard functions have the extremely useful property that they are linear in within the boundaries of the saturation region. Therefore, the value at each edge corresponds to value of the edge itself. In formulas:

$$g(u) = \begin{cases} \sigma_\downarrow & u \leq \sigma_\downarrow \\ u & u \in [\sigma_\downarrow, \sigma_\uparrow] \\ \sigma_\uparrow & u \geq \sigma_\uparrow \end{cases} \tag{8.8}$$

The first reason why this is useful is that the voltage at the capacitor reads

$$C_o \frac{d\varphi(v_k)}{dt} = C_o \varphi'(v_k) \frac{dv_k}{dt} = \begin{cases} 0 & \text{if } v_k = v_o\sigma_\downarrow \vee v_k = v_o\sigma_\uparrow \\ C_o \frac{dv_k}{dt} & \text{if } v_k \in [v_o\sigma_\downarrow; v_o\sigma_\uparrow] \end{cases} \tag{8.9}$$

Therefore, in the region where the rate is not saturated, the dynamical system reads

$$C_o \frac{dv_k}{dt} = -\frac{v_k}{R_o} + \sum_{j=1}^{N_u} G_{kj}\varphi(v_j) + I_k \tag{8.10}$$

Since the evolution in the linear region is the same and the real state variable of the system in $\varphi(v_k)$, this system has the same learning capability as the Additive model, and its hardware implementation comes at the advantage of removing not informative dynamics.

8.1.3 Inference

This subsection presents the results of the most relevant simulations performed in `ngSpice` on the diode-based network. Differently from the previous networks, the dimensionless description was abandoned in favor of a more realistic choice of the electronic quantities.
As a starting point the network was tested on the XOR binary classification task. The previously trained hard-tanh based network was used to define the architecture. In order to perform the dimensioning of the components, the maximum absolute value of the weights was first considered $\omega_{abs} = 1.9561$. From this, using the dynamic range of the reference ReRAM, the characteristic resistance was fixed to $R_o = \omega_{abs}/32\mu S = 61.127k\Omega$. With the aim of fixing the time constant

to $\tau_o = 1ms$ the capacitor was then fixed to $C_o = 16.359nF$. Finally, the characteristic voltage was taken as $v_o = V_\gamma \simeq 0.7V$ while the resistor in the inverting amplifier was imposed to be $R_{inv} = 6.1127k\Omega$. The network was tested on 200 data for different values of the gain factor of the amplifier and of the output resistance. Each simulation was performed on a simulation time of $\Delta t = 2ms$. The network was tested on 200 data for different values of the gain factor of the amplifier and of the output resistance. The results of the simulations can be observed in Tab.8.1. Interestingly, output resistances of the order of the $k\Omega$ were needed to observe an appreciable

$R_o[\Omega]$	$K = 10^6$	$K = 10^5$	$K = 10^4$	$K = 10^3$	$K = 10^2$	$K = 10^1$	$K = 10^0$
0	1	1	1	1	1	1	1
150	1	1	1	1	1	1	1

Table 8.1: Evaluation of the performances of the diodes based network for doing inference with the XOR problem.

worsening of the performances and, therefore, the network can be safely considered to be robust to non idealities and it is possible to focus on testing the MNIST architecture.
The dimensioning procedure was repeated for the hard-tanh based network trained in Chapter 5. In this case the maximum absolute value of the weights was found to be $\omega_{abs} = 0.485$, yielding the characteristic resistance $R_o = \omega_{abs}/32\mu S = 15.158k\Omega$. Once again the time constant was fixed to $\tau_o = 1ms$, corresponding to the capacitor $C_o = 65.973nF$. Due to the presence of the diodes, the characteristic voltage was set to $v_o = V_\gamma \simeq 0.7V$ and the resistor in the inverting amplifier to $R_{inv} = 1.516k\Omega$. Each simulation was performed on a simulation time of $\Delta t = 2ms$. Performing inference on 100 random data, it was possible to achieve an accuracy of 98%. The integer and normalized versions of the confusion matrix can be found in Fig.8.8 and in Fig.8.9, respectively.

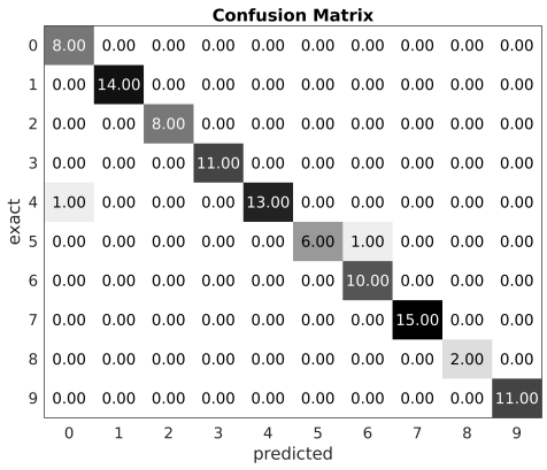


Figure 8.8: MNIST: Confusion Matrix of the inference performed with the diode-based network.

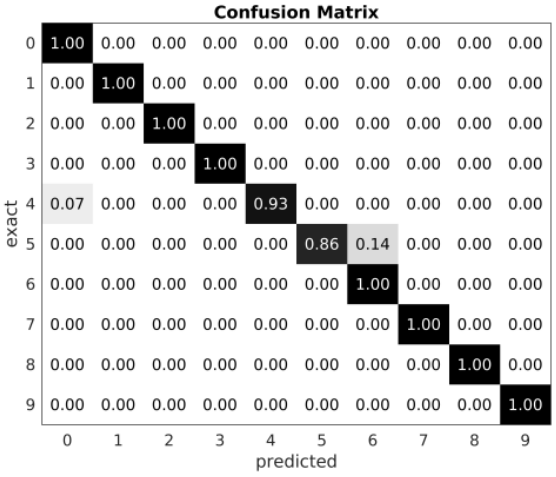


Figure 8.9: MNIST: Normalized Confusion Matrix of the inference performed with the diode-based network.

8.2 Exploiting the low forward voltage of the Schottky diode

The introduction of the diodes in place of the saturating operational amplifiers has several advantages with respect to the hard sigmoid function. Nonetheless, also in this case, the characteristic voltage is sufficiently high to modify the value of the mem-conductance or, in any case, to enter in the non-linear regime of the memristor. It would therefore be desirable to further reduce the characteristic voltage, in order to minimize the source of these non-linearities. In the previous sections it was shown that, when working with diodes, the characteristic voltage is to be chosen as $v_o = V_\gamma$, i.e. the forward voltage of the diode. For the preliminary analysis a normal p-n diode was used, characterized by the typical value $V_\gamma \simeq 0.7V$. Since this parameter is the one dictating the saturation voltage dropping at the extremes of the memristor, it would be desirable to reduce the saturation value to a quantity that not only does not re-program the memristor, but also that is approximately linear. To develop on this point let us consider the Stanford model

$$i_{kj} = G_{kj} V_{SA} \sinh \frac{\varphi(v_j)}{V_{SA}} \tag{8.11}$$

Where a reasonable value for V_{SA} could be $V_{SA} = 0.6V$, accordingly to fits done on experimental data [73]. In the previous section the diodes-based network was shown to be robust to non-idealities when simulated in `ngSpice`. In other terms the behavior of the purely numerical network is almost equivalent to the one of the simulated circuit. On the other hand, this simulation is unbearably time demanding. Therefore, in order to take into account the Stanford model, it is reasonably safe to go back to the `MATLAB` implementation and to synthetically introduce its dimensionless equivalent. To do so, the a-dimensional synaptic current must be chosen to be

$$\begin{aligned} A_{kj} &= (R_o G_{kj}) \frac{V_{SA}}{V_\gamma} \sinh \frac{\varphi(v_j)/V_\gamma}{V_{SA}/V_\gamma} = \\ &= W_{kj} u_{SA} \sinh \frac{g(u_j)}{u_{SA}} \end{aligned}$$

In order to have the linearity, it would be desirable to have the term $u_{SA} = V_{SA}/V_\gamma$ as large as possible, to guarantee $A_{kj} \simeq W_{kj} g(u_j)$. This would correspond to choose a diode with the smallest possible V_γ. In these terms, the Schottky diode has the extremely appealing property that the typical forward voltage for these devices is around $150 - 450mV$. Accordingly to what seen in the introductory chapter, a reasonable assumption is $V_\gamma \simeq 0.2V$. Therefore, the a-dimensional u_{SA} becomes $u_{SA} = 3$. Considering what said before about the voltage the memristor can bear to remain in linearity, the substitution of the semiconductor diode, characterized by $V_\gamma = 0.6 - 0.7V$ with a Schottky diode could allow, in principle, the implementation of the network in hardware while avoiding a re-programming of the memristors and keeping them in linearity throughout the evolution of the dynamical system. The schematic of this circuit is reported in Fig.8.10. In order to numerically sustain these reasonings, two numericale simulations were compared. In the first one the network was trained with the usual Additive model, while using as activation function the hard hyperbolic tangent. The hyperparameters used were the same of Chapter 5 but, this time, the simulation was performed for 6000 training data in order to create a target accuracy. Afterward, the Additive Model was altered by introducing the non-linearity of the Stanford model. This was achieved by performing the following substitutions

$$\begin{cases} W_{mk} g(u_k) \mapsto W_{mk} u_{SA} \sinh \frac{g(u_k)}{u_{SA}} \\ b_k \mapsto b_k u_{SA} \sinh \frac{1}{u_{SA}} \end{cases} \tag{8.12}$$

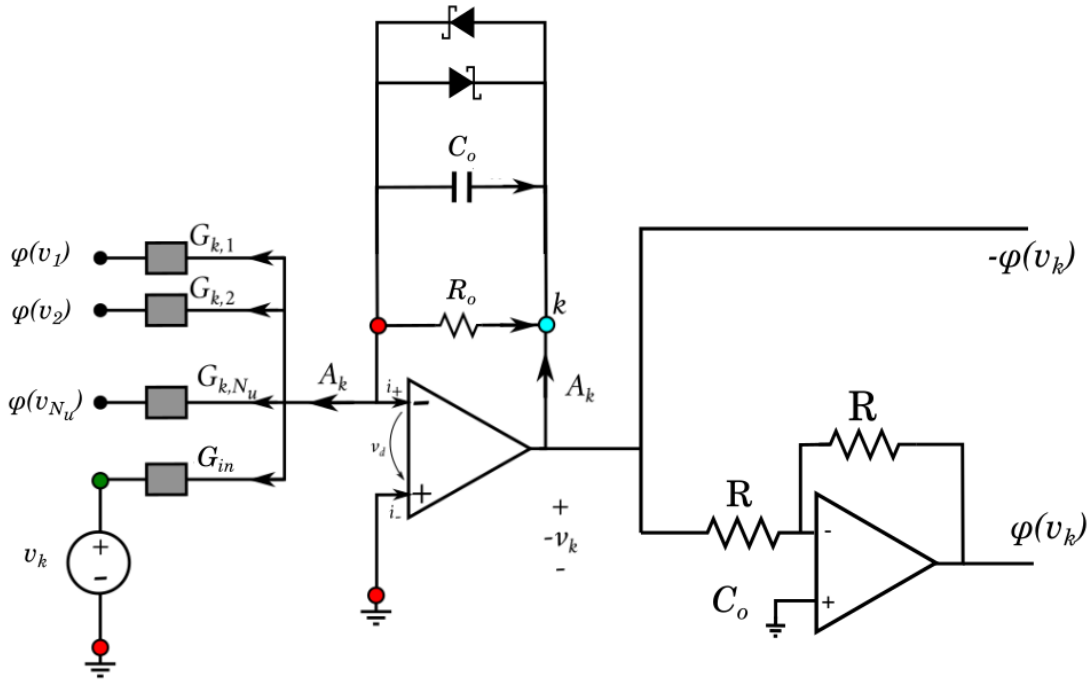


Figure 8.10: Non-linear Leaky integrator: Neuron block built up embedding the Kendall-like activation function in the feedback loop of the operational amplifier with the use of Shottky diodes.

corresponding to a proper introduction of the Stanford-ASU non-linearity in the a-dimensional simulation. Both the networks were trained for 10 epochs on 6000 training data, testing them on 1000 test data and updating the result each 20 data. The learning rate was fixed to 0.005 and the clamping factor to 0.1. In Tab.8.2 it is possible to compare the results of the two simulations. Apparently, despite the distortion in the velocity field EP can learn the classification with the

case	ACC_{train}	ACC_{test}	J_{train}	J_{test}
linear	0.926	0.886	0.23572	0.36953
Stanford	0.925	0.891	0.25131	0.36264

Table 8.2: Comparison numerical training of the MNIST classifier: Linear Additive model and non-Linear version inspired by the Stanford model.

same accuracy as the exacly linear case, provided that the characteristic voltage of the circuit and the V_{SA} of the Stanford model are in the ratio previously discussed

8.3 Conclusions

In this chapter the network is modified so as to introduce in it the activation function defined by Kendall et al. [60]. The first step is to show that it can be used to estimate the hard hyperbolic tangent and the network primitives are based on it. Then, it is proposed an implementation in which the activation function is embedded in the feedback loop of the integrator configuration, obtaining

in this way a non-linear integrator. This technological solution would imply a considerable reduction of area required for the integration, due to the substitution of an operational amplifier, in the voltage follower configuration, with two anti-parallel diodes. Finally, the network is tested via its capability to do inference. In this way it is possible to verify that this technological implementation is well suited for the analogue integration of the dynamical system. Finally, the use of Schottky diodes is proposed to achieve the desired compatibility of the network with the existing resistive-switching devices in terms of linearity and of stability to the electrical stress induced by the dynamical evolution. The diode-based network is also interesting because it allows to remove non informative dynamics, consequently speeding up the dynamical evolution.

Chapter 9

Stochasticity and Quantized networks

So far, all of the systems considered worked at machine precision. This is to say that it was implicitly assumed that the decimal digits can be precisely controlled up to the sixteenth decimal digit. Nonetheless, when one wants to test an algorithm on a real device, this is certainly not the case. As seen in the introductory chapter on the hardware part, among the other possibilities the conductance can be imposed by means of pulses. Alternatively, it might be set working on the current compliance or with the stop voltage. In any case, to a given programming parameter α a mean conductance $\bar{G}(\alpha)$ can be associated, but the real value observed is a random variable $G(\alpha) = \bar{G}(\alpha) + \zeta(t, T, \alpha)$ where, in the most general case, the random fluctuation is a non trivial function of temperature, time and of the programming. Since the update presents this limitation, one generally identifies the stable conductance states in such a way that the stochastic fluctuation falls in within a given interval of conductance. This solution allows to have a better control on the value programmed and the larger is the interval of each quantized state the more controllable will be the programming. In this chapter this concept is discussed using the `MATLAB` codes developed, by adapting Equilibrium propagation to noisy and quantized networks.

9.0.1 Random velocity field

The presence of random fluctuations in the value of the programmed memristor was already discussed in chapter 6 . In this section it is investigated the effect of a random component on the stability of the dynamical system. In particular, random fluctuations are simplistically modelled as gaussian random numbers. Despite the fact that this assumption has been done in the past [78], it might not always be optimal for modelling the actual behavior.
Let us consider a-dimensional variables and let us express the variability as a function of a normal gaussian variable $\xi \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$. If we define $\zeta^w = \nu\xi$, this will have zero mean and standard deviation ν: $\zeta^w \sim \mathcal{N}(\mu = 0, \sigma^2 = \nu^2)$. Then, considering the Additive model, each single synaptic weight W_{ij} will have a random component ζ^w_{ij}, as well as each bias term: $b_i = \bar{b}_i + \zeta^b_i$. Additionally, each of them will be related to a normal random variable as

$\zeta_i = \nu\xi$. In this way

$$\begin{aligned}\frac{du_k}{dt} &= -u_k + \sum_{n=1}^{N_u} W_{kn} g(u_n) + b_k = \\ &= -u_k + \sum_{n=1}^{N_u} \bar{W}_{kn} g(u_n) + \bar{b}_k + \sum_{n=1}^{N_u} \zeta_{kn}^w g(u_n) + \zeta_k^b \\ &= -u_k + \sum_{n=1}^{N_u} \bar{W}_{kn} g(u_n) + \bar{b}_k + \zeta_k(u)\end{aligned}$$

Therefore, the additional term does not only introduce a new bias term, competing with b_k in defining the average point of the state space toward which the system should converge. But also completely unjustified correlations and anti-correlation between the different units appear. To understand the gravity of the term let us perform a worst case analysis of the noise term. Gaussian random variables have the property that a linear combination of normally distributed random variables

$$\zeta = \sum_{i=1}^{N} a_i \zeta_i \quad \zeta_i \sim \mathcal{N}(0,1) \tag{9.1}$$

is a gaussian random varible with zero mean and variance which is the sum of the coefficients: $\zeta \sim \mathcal{N}(0, \sum_{i=1}^{N} a_i)$. Therefore, each point of the space is perturbed by a term which is

$$\begin{aligned}\zeta_k(u) &= \sum_{n=1}^{N_u} \zeta_{kn}^w g(u_n) + \zeta_k^b = \\ &\leq \sum_{n=1}^{N_u} \zeta_{kn}^w + \zeta_k^b \sim \mathcal{N}(0, \sigma^2 = (N_u+1)\nu^2)\end{aligned}$$

So in the worst case, corresponding to an entire region of the state space where the activation functions saturate, the dynamics occur under the influence of a field having an intensity that cannot be predicted and whose variance scales linearly with the number of nodes. The potential gravity of this can be understood by considering that, when dealing with the MNIST data-set, the total number of nodes was $N_u = 196 \cdot 128 \cdot 10 = 250880$. By considering a generic inference trajectory with the trained model (see Fig.9.1), it is possible to see that in this case only a few state values enter in the worst case saturation region but the vast majority remains at a nonzero value. Therefore, the variance might be lower but in any case almost all the nodes will give a non negligible random contribution. In order to appreciate the importance of this problem let us consider the XOR problem. The system was let free to evolve by fixing the noise level ν to 0, 0.1 and 0.3. The trajectories were initialized at four points in space: [0,0,0], [0,1,0], [1,0,1] and [1,1,1]. In all the cases the system was allowed to evolve and the effect of the random component was observed. In all the cases, a low noise level $\nu = 0.1$ introduces some distortion with respect to the case with no noise, but the equilibrium points are not drastically changed. On the other hand, when considering a noise level $\nu = 0.3$, the distortion starts to induce detrimental effects in the dynamics. In both the cases $(1,0)$ and $(0,1)$ only some starting point guarantees the convergence to the desired equilibrium, while in the $(1,1)$ case all the trajectory are completely governed by the stochastic term. This behavior is most likely due to the fact that the more input terms are equal to zero the smaller will be the number of random contributions of the synaptic weights appearing as a random force field. Therefore, while in the $(0,0)$ case no qualitative

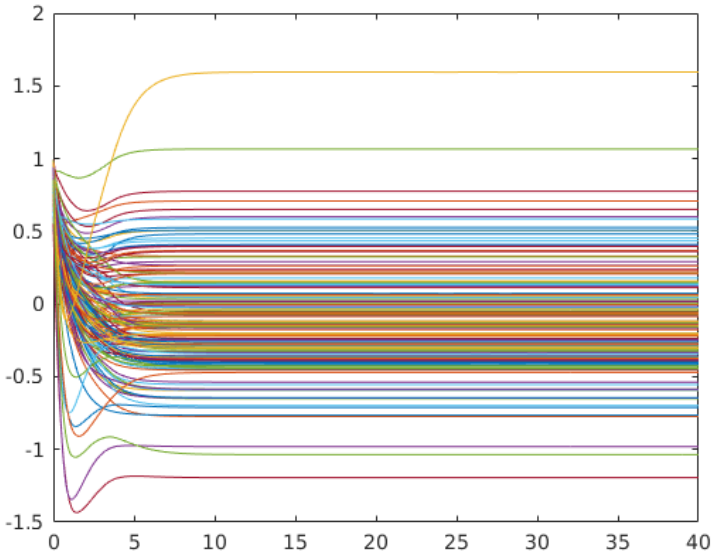


Figure 9.1: MNIST classificator: Trajectories of the neurons in the state space given a random initialization condition.

change in the dynamics is observed, when both the inputs are set to 1 the random contribution will be maximally varying and the likelihood that this effect will induce detrimental degradation is extremely high.

9.0.2 Quantization of the weights

The impossibility of precisely programming in the continuum imposes some restrictions on the amount of information that can be stored in a single weight. In fact, in order to guarantee an external control on the value programmed, it is necessary to identify states of quantized conductance. The idea behind this concept is schematically represented in Fig.9.3 and in Fig.9.4. The red dotted line corresponds to a purely numerical approach, in which given a conductance value in the continuum it is exactly the value programmed, up the the machine precision. As soon as one introduces quantized intervals, the situations becomes the one in the black stair function. The $Q(x)$ values are the values that it is safe to program within a reasonable accuracy. In total there are N_{levels} levels. The nature of the conductances imposes a further constraint. Even if the updates could in principle span the whole real axis, the conductances will have a maximum G_{max} and a minimum G_{min} possible value. Therefore, any update above and below that value will be forced to saturate. Considering these constraints, from one state to the other there will be an interval of $\Delta = \frac{G_{max}-G_{min}}{N_{levels}-1}$ conductance values. In Fig.9.4 it is reported the real programming process expected for the hardware implementation. Once computed the update, the new conductance value to be programmed G^{new} is found. One then identifies to which quantization interval this value belongs to, and the corresponding quantized value $\bar{G}$. This value is associated to a programming parameter α, such as the number of pulses n_{pulses} to be sent to the device. Therefore, after reset this programming is performed. This process corresponds to a sampling of a conductance value from the probability distribution defined by the properties of the memristive device. In this perspective, Fig.9.3 corresponds to the case in which the states are so much far apart that the variance of the memconductance is negligible with respect to

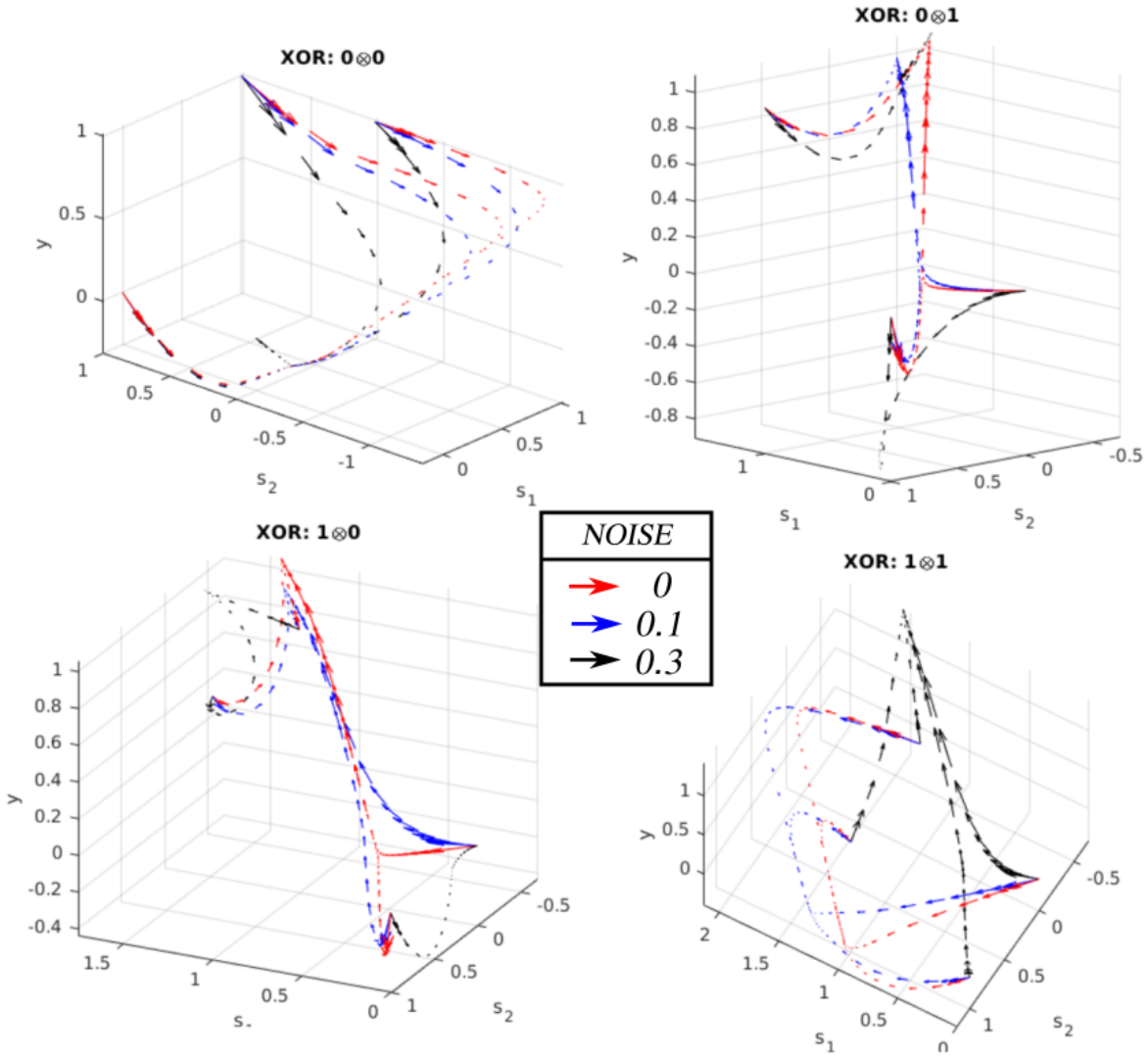


Figure 9.2: Random fields in the XOR classificator when increasing the noise level. The red curve corresponds to $\nu = 0$, the blue one to $\nu = 0.1$ and the black one to $\nu = 0.3$.

the quantization interval. The number of states that can be programmed strongly depends on the nature of the device. Given a resistive switching device and the accuracy desired for the programming it is clear that the number of quantization values will be fixed accordingly. the following section discusses the effect of quantization on previously defined networks.

9.0.3 Motivation

This quantization might seem artificial and poorly supported by the Bengio and Scellier theory. Nonetheless, it can be qualitatively motivated in a fashion similar to the one they used for defining the asymmetric Equilibrium propagation algorithm. In the formulation considered so far, Equilibrium Propagation has been used to evaluate the left hand side of the dynamical system in the parameters space and to estimate the weight update

$$\frac{d\theta}{dt} = -\frac{\partial J}{\partial \theta} \rightarrow \Delta\theta = -\eta \frac{\partial J}{\partial \theta} \tag{9.2}$$

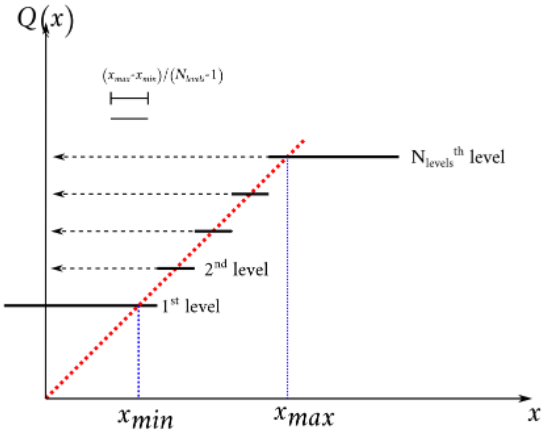


Figure 9.3: Quantization procedure: a continuous value is mapped into a quantized state.

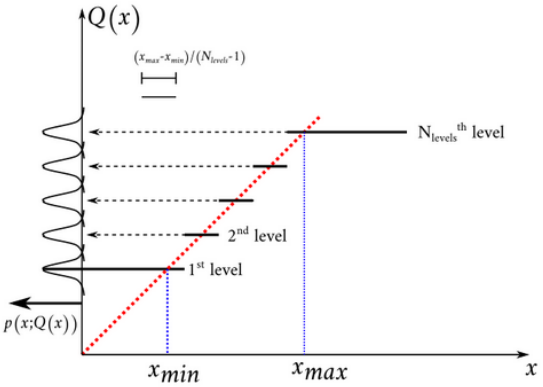


Figure 9.4: Quantization procedure: a random fluctuation is added to the quantization.

When imposing a quantization, all the quantities defined so far will remain the same, the only difference being that only a limited portion of the parameters space can now be explored. In fact it is not possible anymore to explore this space in the continuum, but only at discrete values of it. This is not a problem if the discrete values are the one relevant for encoding the information. In fact, the evolution of the network will still allow to evaluate the ideal weight update. In the quantized update this quantity identifies the direction in which to move, even if the exact value is not allowed. To prove this concept it is better to use networks on which an overall analytical understanding is available, i.e. the AND function. As previously discussed, only two values are needed for the weights: 0.5 and -0.25. Therefore, the network was trained for two epochs, imposing $W_{min} = -0.25$, $W_{max} = 0.5$ and $N_{levels} = 2$. This time, the learning rate and the clamping factor require some attention. Since the mini batch size is set to 1 in this simple problem, the updates of the weights reads

$$\begin{cases} \Delta W_{ij} = \eta\left[g(s^{\beta}_{\theta,i})g(s^{\beta}_{\theta,j}) - g(s^{0}_{\theta,i})g(s^{0}_{\theta,j})\right] \\ \Delta b_i = \eta\left[g(s^{\beta}_{\theta,i}) - g(s^{0}_{\theta,i})\right] \end{cases} \tag{9.3}$$

Let us reason in terms of extremes. If the states were binarized, we would have $g(s) = \sigma \in \{\sigma_{\uparrow} = 1, \sigma_{\downarrow} = 0\}$. Considering the bias at node i, an update must be done every time the clamped solution and the free solution differs, i.e. when one of the variables is 0 and the other one is a 1. However, this automatically implies that Δb_i can only take three values: $\pm\eta$ and 0. In turns, this implies that it is no longer possible to vary the learning rate arbitrarily, since a too small value would imply the impossibility to learn anything. Therefore, when the mini batch size is fixed to 1 the learning rate must be chosen satisfying

$$\eta \geq \Delta = \frac{W_{max} - W_{min}}{N_{levels} - 1} \tag{9.4}$$

Where it must be greater or equal since, in general, it is not guaranteed that the only case in which a correction is worth to be done is the case in which the two equilibriums have oppositely converged. On the contrary there might be a nearby quantized parameter increasing the performances of the network. The same reasoning holds for the weights update. In any case, the lower the learning rate the more the free and clamped solution must be different from

each other in order to introduce a variation. It is therefore to be expected that, when choosing $\eta = \Delta$ the clamping factor should be high, so as to drive the variables to the correct value, consequently inducing a big adjustment where needed. When this happens, the identification of the algorithm with EP might be debatable. Considering the comparison done by Bengio and Scellier themselves with other learning theories, a high value of the β might be interpreted as a transition from Bengio-Scellier learning to Contrastive Hebbian learning, in which the updates read

$$\begin{cases} \Delta W_{ij} = \eta\,[g(s^{\infty}_{\theta,i})g(s^{\infty}_{\theta,j}) - g(s^{0}_{\theta,i})g(s^{0}_{\theta,j})] \\ \Delta b_i = \eta\,[g(s^{\infty}_{\theta,i}) - g(s^{0}_{\theta,i})] \end{cases} \tag{9.5}$$

In the case of the AND function it was imposed $\eta = \Delta = 0.75$ and a good β value was found to be 0.75. In Fig.9.5 it is possible to see how 100% accuracy can be achieved after just one epoch and the parameters were correctly found to be $W_{13} = W_{23} = W_{31} = W_{32} = 0.5$ and $b = -0.25$. Before to proceed, it is worth introducing some similar reasonings for the case of batch size

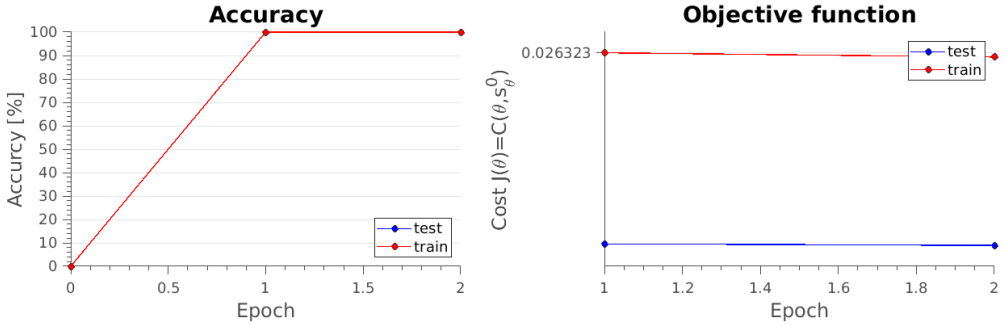


Figure 9.5: Performances of the quantized version of the AND function.

higher than one. In that case the update reads

$$\begin{cases} \Delta W_{ij} = \eta \sum_{k=1}^{N_{batch}} [g(s^{\beta,k}_{\theta,i})g(s^{\beta,k}_{\theta,j}) - g(s^{0,k}_{\theta,i})g(s^{0,k}_{\theta,j})] \\ \Delta b_i = \eta \sum_{k=1}^{N_{batch}} [g(s^{\beta,k}_{\theta,i}) - g(s^{0,k}_{\theta,i})] \end{cases} \tag{9.6}$$

also in this case the worst case is introduced as

$$\eta \geq \frac{\Delta}{N_{batch}} = \frac{W_{max} - W_{min}}{N_{batch}(N_{levels} - 1)} \tag{9.7}$$

However, it is essential to observe that in this case choosing the learning rate to be the minimum one would correspond to update the weights only if the model was totally wrong on all the batch data, and the error was exactly the same. This is a very selective condition and imposing this would probably imply a considerable difficulty in learning. Therefore, it could be safer to consider $\eta \geq \Delta/(2N_{levels})$, corresponding to the limit condition in which in half of the iterations the two phases converge to the same value and half of the time they converge to opposite values, but always in the same direction. However it still remains quite restrictive and one needs to be careful in the tuning of this parameter.

9.0.4 Modified Equilibrium Propagation

Equilibrium Propagation has a strict constraint on its parameters, i.e. the weight matrix must be symmetric. This information is essential to be remarked because a not careful update of the

parameters would induce a fast divergence of the matrix from a symmetric one to an asymmetric one. To understand this statement, let us follow the algorithm in a mini-batch. Firstly, both the EP phases are performed for each data in the minibatch, summing up the different contributions. This is done under the assumption that some sort of digital circuitry can be used to store these quantities

$$\begin{cases} \Delta W_{ij} = \eta \sum_{k=1}^{N_{batch}} [g(s_{\theta,i}^{\beta,k}) g(s_{\theta,j}^{\beta,k}) - g(s_{\theta,i}^{0,k}) g(s_{\theta,j}^{0,k})] \\ \Delta b_i = \eta \sum_{k=1}^{N_{batch}} [g(s_{\theta,i}^{\beta,k}) - g(s_{\theta,i}^{0,k})] \end{cases} \tag{9.8}$$

At this point the matrix of the updates is perfectly symmetric, while the W matrix will be not, due to the presence of a random noise. Since a digital processing of the quantities will be necessary in any case, before to do the update it is worth map the matrix into a symmetric quantity. To this aim the matrix is substituted with its symmetric part

$$W \mapsto W^s = \frac{1}{2}(W^t + W) \tag{9.9}$$

Then, having at disposal a fixed number of quantized levels N_{levels} in between a minimum value ω_{min} and a maximum value ω_{max}, the quantization operation described in Fig.9.3 can be written as

$$\bar{W} = \mathscr{Q}(W^s + \Delta W, \omega_{min}, \omega_{max}, N_{levels}) \tag{9.10}$$

This matrix will be characterized by only N_{levels} values and will be symmetric. This corresponds to the quantity that is desired to be programmed in the memristor. Given the Look Up Table it is possible to determine the α-parameter needed for programming each entry of the matrix and, once performed the programming, the new weight matrix is $W = \bar{W} + \zeta_w$ This matrix will be used for performing the evolution steps during the next mini batch. It is worth stressing the importance of the symmetrization procedure on the weight matrix. In case of an asymmetric weight matrix, it would be impossible to define an energy function and the Bengio-Scellier theorem would not be valid. Performing the update in this way seems the only possible way to limit the divergence of the model from unexpected behaviors.
Finally, since also the bias is a non-volatile quantity defining the model, also this quantity was modelled with a memristor, and also in this case there will be a random update. The difference in this case is that no symmetrization procedure is required on this quantity. The average value will therefore be

$$\bar{b} = \mathscr{Q}(b + \Delta b, \omega_{min}, \omega_{max}, N_{levels}) \tag{9.11}$$

and the actually programmed value is $b = \bar{b} + \zeta_b$. In terms of the gaussian noise, this implies that if the standard deviation is lower than one third of the width of the intervall, then the solution will be extremely stable, since it will only induce a contained fluctuation around the optimal value programmable.

9.1 MNIST classification

This section is a collection of the main results obtained in the context of the MNIST classification with quantized and noisy networks. The first network investigated is a layered structure and its performances are investigated while varying the quantization interval. In the final part it is also discussed the possibility to perform the quantization on the one layer Feature Engineered network.

9.1.1 Multilayer network

The first structure analyzed was a layered one, in which the 196 nodes are fully connected to 125 hidden node, in turn connected to the 10 output nodes. The network was first trained for 10 epochs, without quantization or random terms. The number of training data considered has been reduced to 6000. Both the learning rate and the clamping factors have been fixed to 0.1. The accuracy achieved in this continuum case is 93% and, in Fig.9.6, it is possible to observe the distribution of their values in the trained network. From this information the minimum value of

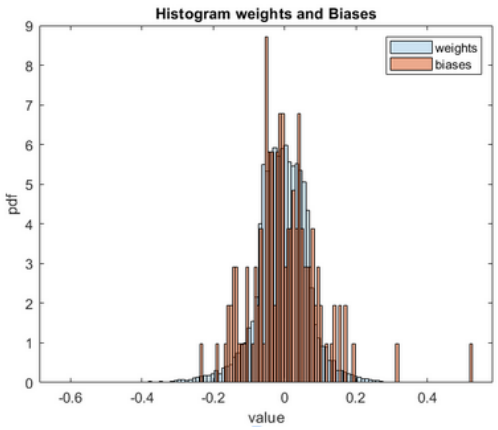


Figure 9.6: Distribution of the weights in the trained hard sigmoid based MNIST network.

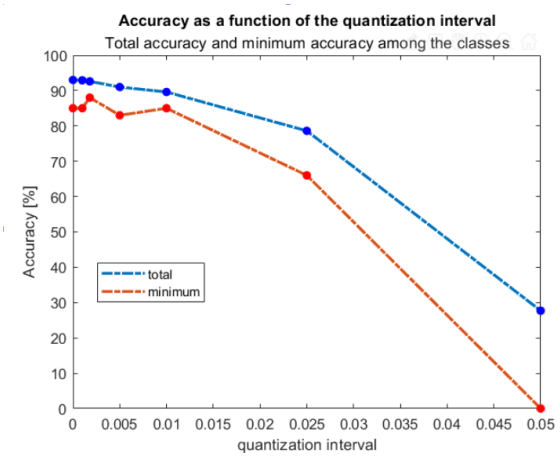


Figure 9.7: Accuracy as a function of the quantization interval.

the weights is approximately $W_{min} = -0.3$ while the maximum value is $W_{max} = 0.3$. Afterward, the number of quantized states was varied and in Fig.9.7 it can be observed the decrease in accuracy induced by the lower amount of information that can be encoded in each weight. The continuum case was labelled as $\Delta = 0$ and it was used to determine a reasonable interval for the weights. It is worth pointing out that both the learning rate and the clamping factor were

Δ	0	0.001	0.0018	0.005	0.01	0.025	0.05
N_{levels}	∞	601	331	121	61	25	13
acc	93%	92.9%	92.6%	91%	89.6%	78.6%	27.7%

Table 9.1: Accuracy of the trained MNIST classificator when reducing the number of quantizaed states.

blindly kept fixed, while these quantities must necessarily vary according to what previously said. For this reason, when the levels are reduced, the system becomes progressively more rigid to changes and it will be progressively harder to perform an update. As a proof of concept of this the number of levels was fixed to 11, corresponding to requiring the memristor to have 5 non-zero states. Considering the small amount of weight values having modulus higher than 0.2, the pruning was further increased by fixing $\omega_{min} = -0.2$ and $\omega_{max} = 0.2$. The training phase was performed on 3000 data, raising the clamping factor to $\beta = 1$ and forcing the learning rate to vary. For each value a simulation was run for 10 epochs, This results in an improvement of the performances as it can be observed in Tab.9.2. Nonetheless it is still not possible to reach performances comparable with the ones achieved in the continuum. This might be due to the

η	acc_{test}	acc_{train}
0.01	11%	13.4%
0.05	69.2%	76%
0.1	77%	84%
0.5	47%	42%

Table 9.2: Accuracy of the trained MNIST classificator when increasing the learning rate at fixed number $N_{levels} = 11$ of quantized states.

previously mentioned reduced capacity of the network in terms of information content. In order to improve the accuracy of the model while keeping fixed the number of quantized levels to 11, the only remaining degree of freedom is the number of nodes in the hidden layer. Therefore, this hyper-parameter was increased, achieving the improvement in the performances reported in Tab.9.3 This shows that, as expected, in quantized networks it becomes necessary to increase

N_h	acc_{test}	acc_{train}
125	77%	84%
150	78.4%	85%
200	81.6%	88.6%
300	82%	85%

Table 9.3: Accuracy of the trained MNIST classificator when increasing the number of hidden nodes at fixed number $N_{levels} = 11$ of quantized states.

the number of parameters to cope with the reduction in the information storage capability of the network. This brings us back to the analysis done on the feature engineering process. In Chapter 5 it was discussed a Feature Engineering approach to project the data-set in a higher dimensional space containing information of the correlation of the pixels and it was suggested that this approach might be particularly suitable for preliminary tests with real hardware realizations. It is thus reasonable to verify the effect of training these networks when the weights are quantized. The selected network was the one built up from the first 15 principal components, yielding 421 input nodes and 4220 parameters. As a consequence of the complete change in the architecture, the hyperparameters are naturally to be changed and the optimized ones were $\beta = 0.75$ and $\eta = 0.05$. With the aim of comparing the performances with the previous architecture, also in this case 11 levels were used for the weights and the interval was taken to be $[-0.2,0.2]$. Indeed, the "continuum" training verified this interval to be reasonable also for the features engineered network. The accuracy on the test set was 84.2% while the accuracy on the training set reached 87.8%. This accuracy is higher than the one obtained when considering 300 hidden nodes, reaffirming the interest in the proposed Feature Engineering approach.

9.2 Conclusions

This chapter presents some preliminary considerations and simulations on the quantized version of Equilibrium Propagation. The need for quantization is primarily dictated by the non-idealities of the resistive-switching devices, that force to take into account the possible presence of variations in the programmed value of the model parameters. First, the random variations are

shown to enter in the dynamical system as position dependent random velocity fields of the state space. Then, it is described the quantized version of Equilibrium Propagation, stressing the importance of the symmetrization procedure so as to ensure a reasonable symmetry in the weight matrix. Additionally, the increase in the quantization is noticed to induce a progressive reduction of the update rate, due to the need for stronger update signals to modify the state. As a consequence, the quantization is argued to require an additional tuning of the hyperparameters, so as to compensate for the higher resistance of the model to learning.

Chapter 10

Conclusions

This work is aimed at investigating a Recurrent Neural Network model, named Additive Model, from an implementation-oriented point of view. Recurrent neural networks are extremely powerful supervised machine learning models. However, they have the disadvantage that both training and testing are extremely demanding in terms of computation times. In fact, given a network of N_u neurons, the processing of a piece of data requires the integration of a dynamic system whose number of variables is of the order of N_u. In the absence of other computational costs, the system would require a time T to reach convergence. However, a numerical simulation requires the calculation of matrix-vector multiplications, that bring the total computational cost to be of the order of $\mathcal{O}(N_u^2 T)$. The use of crossbars made of resistive switching components to model synapses, i.e. the parameters of the model, could theoretically bring the computational cost to $\mathcal{O}(T)$. In addition, it would be extremely beneficial to find a way to reduce the integration time to a minimum. Having said that, two things are needed to create a recurrent neural network in hardware: an efficient algorithm to optimise the parameters and a compact realisation of the network building blocks. The first part of this work is devoted to algorithmic aspects and the supervised learning approaches investigated are based on the Equilibrium Propagation algorithm, devised by Bengio and Scellier. Instead, in the second part some circuit solutions are proposed together with their simulation in ngSpice.

10.1 Learning Algorithms

The main objective of the algorithmic analysis is to develop a framework for training RNNs with different types of Equilibrium Propagation. In concrete terms, the main objectives were to verify the applicability of EP to various supervised machine learning problems and to obtain networks to be simulated in hardware at a later stage. However, additional results were obtained in the process, which are commented on below.

10.1.1 Update rules

In the literature there are two types of EP, classified according to the symmetry of the synaptic weight matrix: the symmetrical version and the Scellier extension to asymmetrical matrices. The symmetrical version [1] is rigorously proved, but symmetry is a constraint that could detrimentally limit the potential of the model. On the other hand, the asymmetric version [5] is potentially more general, but the update rule was derived by means of a heuristic approach, thus

leaving open an interesting branch of research devoted to formally prove the mechanisms behind its effectiveness or identifying new algorithmic solutions. The contribution of this study to this open question is to propose a new asymmetric update rule from the asymmetric version and Bengio's experimental observations. Again, the proposed update rule is obtained by combining logical deductions with Bengio's experimental observations. However, the proposed update rule has an additional term, compatible with Scellier's deduction, which corresponds to the first derivative of the rate of the post-synaptic neuron. In this context, an initial benefit of using so-called hard functions for introducing non-linearity into the network is also seen: in first approximation this derivative term can be evaluated in a simple way with a threshold function that is either 0 or 1. This is the first reason why either the hard hyperbolic tangent or the hard sigmoid has been used throughout the work.

10.1.2 Reconstruction

The first supervised machine learning problem considered corresponds to reconstructing damaged patterns and is termed Reconstruction for short. The additive model is shown to be effective with different types of networks, which differ from each other in their architecture and the way in which the input is inserted. First, the Grossberg-Hopfield Model is described. In this case a neuron is defined for each pixel, the network is fully connected and the input is entered as a bias term. This also allows to shed light on the type of information that the Additive Model learns, which is based on the correlation between different nodes in the network. In addition, the applicability of all three EP-like update rules is verified. It is subsequently shown that the Grossberg-Hopfield network can be regarded as a special case of an architecture named Bengio-Scellier network. In the latter, the inputs correspond to clamped neurons, the output neurons are connected to the input neurons and/or, if necessary, to a network of hidden nodes that increase the computational capabilities of the model. As an example of the latter network, a recurrent denoising autoencoder is defined and it is used to solve the reconstruction problem.

10.1.3 Classification

The second supervised machine learning problem considered corresponds to classifying images. The Bengio-Scellier network is the most naturally suited to solving this problem. In fact, it is sufficient to introduce an output node for each class and define a hidden network complex enough to "raise" the correct output given a certain input. Classification problems of increasing complexity are considered: a binary classification of a linearly separable data-set, a binary classification of a non-linearly separable data-set and a classification into ten classes of handwritten digits. The usefulness of the first two is mainly to verify the correctness of the framework developed and to gain insight into the mechanisms of inference and learning in simple networks. On the other hand, the last is the most common benchmark in AI and is needed to test the applicability of the Additive Model to problems closer to real-world applications. In addition to verifying the possibility of solving this problem using both hard hyperbolic tangent and hard sigmoid, some hardware implementation-oriented considerations are made. Firstly, it is discussed that using the analogue network in the learning phase would require a side circuitry to perform the weight update during the learning phase. It is reasonable to assume that it is preferable to limit the number of updates to the essential minimum. Therefore, it is shown to what extent the performances of the network remain acceptable by increasing the number of data processed before performing the update, in jargon the mini batch size. In order to achieve satisfactory

accuracy, it is necessary to introduce a hidden layer. However, if one tried to solve this problem with the Additive Model by means of linear separation, i.e. using a one layer network, one would obtain an accuracy of the order of 80%. The introduction of a hidden layer looses this linear information and forces the use of an extremely large number of parameters. Therefore, this work proposes as an alternative approach to augment the starting features with non-linear features, engineered in line with the type of information that the network is expected to learn. Since the information learned by the model has been shown to be related to the correlation between variables, non-linear features containing this information may be ideal, allowing the network to focus on higher moments. For this reason, a general approach is proposed to introduce correlation information in a controlled way, i.e. choosing the amount of non-linear information to be introduced and consequently improving accuracy. This approach is based on the hierarchy in the data that can be obtained by means of PCA, which is based on the solution of the eigenvalue problem of the correlation matrix. With the new features, accuracies comparable to the state of the art can be achieved, while drastically reducing the number of parameters in the model. Such approaches could also be useful in academia, since large-scale integration of memristive crossbars is a subject of research and the number of synaptic weights available to many research groups is rather limited.

10.2 Hardware

One extremely advantageous thing about the Additive Model is that its circuit implementation is relatively simple and has been studied in the past under the name of standard circuital interpretation.

10.2.1 Analogue Networks

The differential equations used in the *Learning Algorithms* part are mapped into the corresponding circuit equations, explicitly introducing characteristic dimensional quantities to be defined by the choice of circuit components. Ultimately, processing a data with the Additive Model corresponds to solving a nonlinear dynamical system. Therefore, the objective is to define circuit quantities equivalent to those appearing in the numerical simulations and, having done so, processing a data corresponds to letting the circuit evolve to convergence. Three circuit solutions are introduced and for each of them the network is defined using the parameters optimised in the *Learning Algorithms* part. Afterwards, the circuit simulation is used to verify the correct hardware integration, possibly even in the presence of non-idealities to test the robustness of the solution. The first circuit considered is purely behavioral in the sense that the architecture is defined using ideal primitives. This makes it possible to automate the definition of the circuit from the sole knowledge of the weight matrix and the bias vector. To proceed, it is necessary to introduce an implementation of the neuron and, following what done by Zoppo, Marrone and Corinto in [63], the choice done in this work is to use an operational amplifier in the leaky integrator configuration. The operational amplifier is an active component whose output should saturate outside the range defined by the supply voltage. Therefore, the second proposed circuit solution uses this property to model the hard sigmoid activation function. The model used for the amplifier is an extremely general amplifier model and, interestingly, the networks considered is shown to be quite stable to variations in parameters such as output resistance and voltage gain. However, this solution makes it necessary to use the supply voltage of the circuit as the characteristic

voltage, and this identifies the order of magnitude of the voltages which will fall on the resistive switching devices. Since a voltage at the ends of resistive switching devices exceeding a few hundred millivolts induces undesired changes in the conductance state, this presents concerns in terms of compatibility with a properly functioning circuit. To solve this problem, a final circuit is introduced, which compactly implements the hard tanh activation function. It is mathematically justified that the introduction of two anti-parallel diodes in the leaky integrator feedback allows the hard tanh to be implemented in a very compact manner. In addition, this solution has the advantage of limiting the dynamics within a range defined by the diode's *forward voltage*, which is in the order of hundreds of millivolts. Moreover, the presence of the diodes in parallel with the capacitor of the integrator allows to remove unnecessary dynamics, consequently speeding up the integration time. The functioning of the network is tested for the inference task on the handwritten digits classification problem. In addition, the use of Schottky diodes makes it possible to reduce the reference voltage to around $200mV$, which is widely compatible with the retention of many ReRAMs in the literature. Once verified that the hardware simulation perfectly reproduced the numerical result, a more realistic model is introduced for resistive switching devices. Since no appreciable difference is seen between the circuit simulation and the corresponding `MATLAB` simulation, the simulations were done with the dimensionless models in the `MATLAB` code. This choice is mainly related to computational cost reasons. Using the ASU-Stanford model it is discussed that for typical parameters of ReRAMs the Additive Model in the presence of the non-idealities of ReRAMs achieves performances comparable to those obtained with ideal synapses, further confirming the possibility of implementing this network in hardware. As part of this process, a number of precautions are identified as necessary in order to choose the right circuit components.

10.2.2 Stochasticity and quantized networks

Up to this point, the networks were modelled with weights obtained in the `MATLAB` framework. However, when switching to analogue networks, the accuracy that the weights can be programmed to is heavily penalised. For this reason, if one wants to numerically model the programming that takes place experimentally, one can only speak of a desired programming value and a fluctuation around it. The effect of detrimental uncontrolled random fluctuations is discussed and it is highlighted the importance of introducing a quantization in the weights, because a network already quantized in the numerical code is more likely to be reproducible in hardware. While describing the extension of Equilibrium Propagation to quantized networks the main precautions to be taken are discussed. Additionally, the main challenges related to quantized networks are highlighted using as a working table the digits classification problem. Finally, once verified that the quantization implies the need for increasing the number of nodes in the network, it is verified that the Feature Engineered one layer network yields promising accuracies even when quantized. Once again its reduced number of parameters makes this solution extremely appealing from an experimental realization point of view.

10.3 Future works

Due to the limited time available for the experience many results need for a more in depth analysis. First of all the proposed update rule needs to be further analysed, with the aim of formalizing the logical deductions and assumptions into rigorous proofs. Additionally, also the

feature engineering approach proposed would require some additional mathematical analysis and extensions. The features that can be designed with PCA are limited and, working with higher statistical quantities rather than the correlation it might be possible to further increase the benefits of this approach. Finally, on the hardware side an actual experimental realization needs to be done, in order to verify the correct functioning of the diodes based network.

Bibliography

[1] Benjamin Scellier and Yoshua Bengio. “Equilibrium propagation: Bridging the gap between energy-based models and backpropagation”. In: *Frontiers in computational neuroscience* 11 (2017), p. 24.

[2] M Azzaz et al. “Endurance/retention trade off in HfOx and TaOx based RRAM”. In: *2016 IEEE 8th international memory workshop (IMW)*. IEEE. 2016, pp. 1–4.

[3] Wei Feng, Hisashi Shima, Kenji Ohmori, and Hiroyuki Akinaga. “Investigation of switching mechanism in HfO x-ReRAM under low power and conventional operation modes”. In: *Scientific reports* 6.1 (2016), pp. 1–8.

[4] Andre Xian Ming Chang, Berin Martini, and Eugenio Culurciello. “Recurrent neural networks hardware implementation on FPGA”. In: *arXiv preprint arXiv:1511.05552* (2015).

[5] Benjamin Scellier et al. “Extending the framework of equilibrium propagation to general dynamics”. In: (2018).

[6] Paolo Nenzi and Holger Vogt. *Ngspice Users Manual Version 23*. 2011.

[7] Giuseppe Piccolboni et al. “Investigation of cycle-to-cycle variability in HfO 2-based oxram”. In: *IEEE Electron Device Letters* 37.6 (2016), pp. 721–723.

[8] Alan Mathison Turing. “On computable numbers, with an application to the Entscheidungsproblem”. In: *Proceedings of the London mathematical society* 2.1 (1937), pp. 230–265.

[9] John Von Neumann. “First Draft of a Report on the EDVAC”. In: *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75.

[10] MM Irvine. “Early digital computers at bell telephone laboratories”. In: *IEEE Annals of the History of Computing* 23.3 (2001), pp. 22–42.

[11] Gordon E Moore et al. *Cramming more components onto integrated circuits*. 1965.

[12] Robert H Dennard et al. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268.

[13] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. “A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955”. In: *AI magazine* 27.4 (2006), pp. 12–12.

[14] James Moor. “The Dartmouth College artificial intelligence conference: The next fifty years”. In: *Ai Magazine* 27.4 (2006), pp. 87–87.

[15] Michael Haenlein and Andreas Kaplan. “A brief history of artificial intelligence: On the past, present, and future of artificial intelligence”. In: *California management review* 61.4 (2019), pp. 5–14.

[16] L Guy Jr. *Common LISP: the language*. Digital, 1984.

[17] Ray Kurzweil. *The singularity is near: When humans transcend biology*. Penguin, 2005.

[18] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[19] Lenka Zdeborová. “Understanding deep learning is also a job for physicists”. In: *Nature Physics* 16.6 (2020), pp. 602–604.

[20] Muminov Ibrokhim Botir Ugli. “Will Human Beings Be Superseded by Generative Pre-trained Transformer 3 (GPT-3) in Programming?” In: *International Journal on Orange Technologies* 2.10 (), pp. 141–143.

[21] Greg Yeric. “Moore’s law at 50: Are we planning for retirement?” In: *2015 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2015, pp. 1–1.

[22] Rob Toews. *Deep Learning’s Carbon Emissions Problem*. `https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/?sh=122fa5a76b43`. 2020.

[23] Kashif Bilal, Saif Ur Rehman Malik, Samee U Khan, and Albert Y Zomaya. “Trends and challenges in cloud datacenters”. In: *IEEE cloud computing* 1.1 (2014), pp. 10–20.

[24] Sankar Basu et al. “Nonsilicon, Non-von Neumann Computing—Part I [Scanning the Issue]”. In: *Proceedings of the IEEE* 107.1 (2018), pp. 11–18.

[25] Alan Mathison Turing. *Intelligent machinery*. 1948.

[26] National Nanotechnology Initiative et al. “A nanotechnology-inspired grand challenge for future computing”. In: (2015).

[27] Mike Davies et al. “Loihi: A neuromorphic manycore processor with on-chip learning”. In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[28] Clint Witchalls. “A computer that thinks”. In: *New Scientist* 224.2994 (2014), pp. 28–29.

[29] Jianshi Tang et al. “Bridging biological and artificial neural networks with emerging neuromorphic devices: fundamentals, progress, and challenges”. In: *Advanced Materials* 31.49 (2019), p. 1902761.

[30] IRDS™ Technical Community. “IRDS™ 2020: Beyond CMOS”. In: (2020).

[31] B Govoreanu et al. “10× 10nm 2 Hf/HfO x crossbar resistive RAM with excellent performance, reliability and low-energy operation”. In: *2011 International Electron Devices Meeting*. IEEE. 2011, pp. 31–6.

[32] Zhiqiang Wei et al. “Highly reliable TaOx ReRAM and direct evidence of redox reaction mechanism”. In: *2008 IEEE International Electron Devices Meeting*. IEEE. 2008, pp. 1–4.

[33] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[34] Giacomo Indiveri, Elisabetta Chicca, and Rodney Douglas. "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity". In: *IEEE transactions on neural networks* 17.1 (2006), pp. 211–221.

[35] Cecilia Giovinazzo et al. "Analog Control of Retainable Resistance Multistates in HfO2 Resistive-Switching Random Access Memories (ReRAMs)". In: *ACS Applied Electronic Materials* 1.6 (2019), pp. 900–909.

[36] Fernando Corinto, Pier Paolo Civalleri, and Leon O Chua. "A theoretical approach to memristor devices". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 5.2 (2015), pp. 123–132.

[37] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. "The missing memristor found". In: *nature* 453.7191 (2008), pp. 80–83.

[38] Sascha Vongehr and Xiangkang Meng. "The missing memristor has not been found". In: *Scientific reports* 5.1 (2015), pp. 1–7.

[39] Isaac Abraham. "The case for rejecting the memristor as a fundamental circuit element". In: *Scientific reports* 8.1 (2018), pp. 1–9.

[40] S Grossberg. "The theory of embedding fields with applications to psychology and neurophysiology". In: *Rockefeller Institute for Medical Research, New York* (1964).

[41] Stephen Grossberg. "Embedding fields: A theory of learning with physiological implications". In: *Journal of Mathematical Psychology* 6.2 (1969), pp. 209–239.

[42] Stephen Grossberg. "A prediction theory for some nonlinear functional-differential equations i. learning of lists". In: *Journal of Mathematical Analysis and Applications* 21.3 (1968), pp. 643–694.

[43] Stephen Grossberg et al. "Learning and energy-entropy dependence in some nonlinear functional-differential systems". In: *Bulletin of the American Mathematical Society* 75.6 (1969), pp. 1238–1242.

[44] Stephen Grossberg. "Nonlinear neural networks: Principles, mechanisms, and architectures". In: *Neural networks* 1.1 (1988), pp. 17–61.

[45] Stephen Grossberg. "Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity". In: *Proceedings of the National Academy of Sciences of the United States of America* 59.2 (1968), p. 368.

[46] Stephen Grossberg. "Some physiological and biochemical consequences of psychological postulates." In: *Proceedings of the National Academy of Sciences of the United States of America* 60.3 (1968), p. 758.

[47] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[48] John J Hopfield. "Neurons with graded response have collective computational properties like those of two-state neurons". In: *Proceedings of the national academy of sciences* 81.10 (1984), pp. 3088–3092.

[49] Simon Haykin and N Network. "A comprehensive foundation". In: *Neural networks* 2.2004 (2004), p. 41.

[50] Steven H Strogatz. *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[51] Edmund T Rolls and Gustavo Deco. *The noisy brain: stochastic dynamics as a principle of brain function*. Vol. 34. Oxford university press Oxford, 2010.

[52] Michael A Cohen and Stephen Grossberg. "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks". In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 815–826.

[53] Yann LeCun et al. "A tutorial on energy-based learning". In: *Predicting structured data* 1.0 (2006).

[54] Fernando Pineda. "Generalization of back propagation to recurrent and higher order neural networks". In: *Neural information processing systems*. Citeseer. 1987, pp. 602–611.

[55] Df Tank and J Hopfield. "Simple'neural'optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit". In: *IEEE transactions on circuits and systems* 33.5 (1986), pp. 533–541.

[56] M Kennedy and La Chua. "Unifying the Tank and Hopfield linear programming circuit and the canonical nonlinear programming circuit of Chua and Lin". In: *IEEE Transactions on Circuits and Systems* 34.2 (1987), pp. 210–214.

[57] Michael Peter Kennedy and Leon O Chua. "Neural networks for nonlinear programming". In: *IEEE Transactions on Circuits and Systems* 35.5 (1988), pp. 554–562.

[58] John Hertz, Anders Krogh, Benny Lautrup, and Torsten Lehmann. "Nonlinear backpropagation: doing backpropagation without derivatives of the activation function". In: *IEEE Transactions on neural networks* 8.6 (1997), pp. 1321–1327.

[59] Maxence Ernoult et al. "Equilibrium propagation with continual weight updates". In: *arXiv preprint arXiv:2005.04168* (2020).

[60] Jack Kendall et al. "Training End-to-End Analog Neural Networks with Equilibrium Propagation". In: *arXiv preprint arXiv:2006.01981* (2020).

[61] Umberto Esposito, Michele Giugliano, Mark Van Rossum, and Eleni Vasilaki. "Measuring symmetry, asymmetry and randomness in neural network connectivity". In: *PloS one* 9.7 (2014), e100805.

[62] Yoshua Bengio et al. "STDP-compatible approximation of backpropagation in an energy-based model". In: *Neural computation* 29.3 (2017), pp. 555–577.

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

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

[65] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: `http://yann.lecun.com/exdb/mnist/`.

[66] Jérémie Laydevant, Maxence Ernoult, Damien Querlioz, and Julie Grollier. "Training Dynamical Binary Neural Networks with Equilibrium Propagation". In: *arXiv preprint arXiv:2103.08953* (2021).

[67] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[68] Andrea Costamagna. "Designing models using machine learning: one-body reduced density matrices and spectra". PhD thesis. Politecnico di Torino, 2020.

[69] Jake VanderPlas. *Python data science handbook: Essential tools for working with data*. " O'Reilly Media, Inc.", 2016.

[70] Hooman Farkhani et al. "Lao-NCS: laser assisted spin torque nano oscillator-based neuromorphic computing system". In: *Frontiers in neuroscience* 13 (2020), p. 1429.

[71] Simon M Sze, Yiming Li, and Kwok K Ng. *Physics of semiconductor devices*. John wiley & sons, 2021.

[72] Diodes incorporated. *Datasheet Schottky Diode DFLS220L*. `https://www.diodes.com/assets/Datasheets/ds30517.pdf`. 2008.

[73] Ximeng Guan, Shimeng Yu, and H-S Philip Wong. "A SPICE compact model of metal oxide resistive switching memory with variations". In: *IEEE electron device letters* 33.10 (2012), pp. 1405–1407.

[74] Debashis Panda, Paritosh Piyush Sahu, and Tseung Yuen Tseng. "A collective study on modeling and simulation of resistive random access memory". In: *Nanoscale research letters* 13.1 (2018), pp. 1–48.

[75] Leon Chua and Gui-Nian Lin. "Nonlinear programming without computation". In: *IEEE Transactions on Circuits and Systems* 31.2 (1984), pp. 182–188.

[76] Robert Plonsey. "Bioelectric phenomena". In: *Wiley Encyclopedia of Electrical and Electronics Engineering* (2001).

[77] Johannes Bill and Robert Legenstein. "A compound memristive synapse model for statistical learning through STDP in spiking neural networks". In: *Frontiers in Neuroscience* 8 (2014), p. 412. ISSN: 1662-453X. DOI: `10.3389/fnins.2014.00412`. URL: `https://www.frontiersin.org/article/10.3389/fnins.2014.00412`.

[78] Thomas Dalgaty et al. "In situ learning using intrinsic memristor variability via Markov chain Monte Carlo sampling". In: *Nature Electronics* 4.2 (2021), pp. 151–161.