POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Biomedica

Tesi di Laurea

Spiking neural networks from theory to practice: machine learning with biological plausible neuron model.

Relatore
prof. Elisa Ficarra

Luca Zanatta

Correlatori:
Francesco Barchi
Francesco Ponzio, Gianvito Urgese

Marzo 2019

Summary

Recent trends in deep learning have led to a proliferation of studies investigating deep Artificial Neural Networks (ANNs) in a supervised fashion. The most consolidated approach to train such ANNs relies on the back-propagation method, with the rigorous requirement of using a differentiable, continuous-valued activation function for the structuring element of the net, i.e. the neuron. Such function is fairly far away from the realistic biologically behaviour of the neuron, which relies on spikes to compute and transmit information. Spiking Neural Networks (SNNs), making use of spikes to manage information flows, seem thus the most feasible solution to fully explore how brain works and computes. Moreover SNNs are highly hardware friendly and more efficient than ANNs in general, opening the way to many applications, especially for portable devices. Besides this aspects in favor of SNNs, their implementation and use is still an open issue, due to the paucity of frameworks, simulators and practical case studies. The aim of the present Master degree thesis is therefore three- fold: 1)an exhaustive investigation of recent methods to develop and train SNNs in the field of image classification; 2)the testing of SNNs potentials in an extremely challenging task, i.e. the classification of histological samples, where the main challenge to be tackled is the extreme intra-class and inter-dataset variability, due to the architectural and colouring characteristics of the histological images. More specifically the experiments were focused on the classification of three categories of interest for Colorectal Cancer (CRC), namely adenocarcinoma, adenoma and hyper-plastic polyp; 3)the comparison with the state-of-the-art technique for image classification: deep Convolutional Neural Networks.

Acknowledgements

Vorrei ringraziare innanzitutto la mia relatrice, la professoressa Elisa Ficarra, per avermi dato l'opportunità di lavorare nel suo laboratorio ed in questo modo di approfondire gli argomenti di mio interesse sui quali spero di poter continuare a lavorare. Ringrazio anche Francesco Barchi, Francesco Ponzio e Gianvito Urgese per avermi seguito in questi mesi, aiutato ad acquisire nuove competenze ed a volte condiviso anche momenti di svago. Un grande ringraziamento va alla mia famiglia: dai miei genitori che mi hanno permesso di frequentare l'università qui a Torino e che ci sono stati sia dal punto di vista economico che morale, ai miei fratelli Fabio, Mattia e Mauro e a mia sorella Yvonne con cui di certo non sono mancati bei momenti insieme nonostante la distanza ed i miei nonni. Poter frequentare il Politecnico di Torino è stata una grande opportunità, a volte con momenti più difficili di altri, ma è stato anche per la mia famiglia che in questi momenti mi sono motivato per proseguire con costanza ed impegno i miei studi. In questi anni, il mio percorso è stato reso speciale anche dalle persone che ho incontrato qui e che adesso fanno parte della mia vita. Con ognuna di loro le esperienze fatte insieme sono un altro bel ricordo di questi anni e, ovviamente, spero di averne tanti altri con loro. Per questo motivo vorrei ringraziare innanzitutto Irene la sicula (chi più di lei mi ha sopportato?), ma anche: Alessandro, Alsona, Baran, Carlo, Chiara B, Chiara C, Chiara M, Dascia, Gabriel, Irene, Marco, Matteo, Mili, Michela, Momo, Nicolò, Pasqui, Postino, Roberto, Stefano, Thom, Yong.

Contents

Summary I

Acknowledgements II

List of Figures V

List of Tables VII

1 Introduction 1
- 1.1 Brief introduction to machine learning . . . 1
- 1.2 Simulators . . . 5
  - 1.2.1 Software simulators . . . 5
  - 1.2.2 Neuromorphic hardware . . . 6
  - 1.2.3 PyNN . . . 8

2 Spiking neural networks 10
- 2.1 Models . . . 10
  - 2.1.1 Hodgking-Huxley model (HH) . . . 11
  - 2.1.2 Leaky integrate and fire model (LIF) . . . 13
  - 2.1.3 Spike response model (SRM) . . . 13
  - 2.1.4 Izhikevich model (IZK) . . . 14
  - 2.1.5 Neurons comparison . . . 15
  - 2.1.6 Synapse . . . 16
- 2.2 Encoding . . . 19
  - 2.2.1 Input . . . 19
  - 2.2.2 Processing . . . 21
- 2.3 Neural Engineering Framework (NEF) . . . 23
- 2.4 Network topologies . . . 24
- 2.5 Learning . . . 26
  - 2.5.1 Hebbian rule . . . 27
  - 2.5.2 Spike-timing dependent plasticity (STDP) . . . 29
  - 2.5.3 Supervised learning . . . 31
- 2.6 Networks . . . 35

3 Methods and results 39
3.1 Input encoding 39
3.2 Datatset 39
3.3 Classifiers 41

4 Discussion 52

Bibliography 54

List of Figures

1.1 Neuron anatomy . . . 2
1.2 Neuron model of McCulloch-Pitts . . . 4
1.3 SpiNNaker multicore System-on-chip . . . 7
1.4 BrainScaleS and SpiNNaker systems . . . 8
1.5 TrueNorth and SyNAPSE project . . . 8
1.6 Loihi board . . . 9

2.1 Action potential . . . 10
2.2 Some biological spike trains . . . 11
2.3 Neuron of Hodgking-Huxley . . . 12
2.4 Electronic circuit of LIF neuron . . . 13
2.5 Membrane potential of LIF neuron . . . 14
2.6 Membrane potential of Izhikevich neuron . . . 15
2.7 Some spike trains . . . 16
2.8 Graph of computational cost-biological reliability . . . 17
2.9 Summary of the neuron behaviour . . . 17
2.10 Conductance of synapses . . . 18
2.11 Neural gas . . . 21
2.12 Time-to-first-spike coding . . . 22
2.13 Rank-order coding . . . 22
2.14 Latency coding . . . 23
2.15 NEF with two neurons . . . 24
2.16 NEF with 30 neurons . . . 24
2.17 Feedforward network . . . 26
2.18 Recurrent artificial neural network . . . 27
2.19 Synfire chain . . . 28
2.20 Reservoir computing . . . 28
2.21 Example of CNN . . . 29
2.22 CNN architecture and neuron . . . 29
2.23 Pooling layer . . . 30
2.24 Examples of learning window . . . 31
2.25 Examples of working STDP . . . 32

2.26 LIF vs soft LIF 34
2.27 Network of Schmuker 36
2.28 Network of Diamond 37
2.29 Network of Diehl 38
2.30 SCNN 38

3.1 Plot Poisson distribution 40
3.2 Firing rate with Poisson distribution 40
3.3 Firing rate with gamma distribution 41
3.4 Scatterplot of Fisher's iris dataset 42
3.5 Examples of Fisher's iris dataset 42
3.6 Example of digit in MINST 43
3.7 Examples of the three classes of CRC coloured. 43
3.8 Examples of the three classes of CRC in grayscale. 44
3.9 Graph of accuracies of Schmuker's network. 44
3.10 First try: training 45
3.11 First try: test 45
3.12 Second try 46
3.13 Third try 46
3.14 Accuracy graph 47
3.15 Example of divergence. 47
3.16 Accuarcy graph of the tries of Diamond's network. 48
3.17 Hyperparameters analysis of CNN. 48
3.18 Graph from MNIST 49
3.19 Graph from CRC 49
3.20 CRC misclassification 50
3.21 Hyperparameters analysis of CNN. 50
3.22 Confusion matrix of CNN and SCNN. 51

List of Tables

2.1 Table of Hebbian rules 30

Chapter 1

Introduction

This thesis was born from the idea of approaching to the interesting field of machine learning, using algorithms that mimic the behavior of mammalian brain, in particular using models of spiking neural networks (SNNs). In the last few years, a lot of companies (Intel and IBM) and universities have been investing in this new type of machine learning approach because it is showing a lot of positive sides: they are environmentally-friendly (in terms of power consumption) and hardware-friendly (a neuron model can be create with few transistor). In the current project, we are going to describe the third generation of artificial neural networks, then we will classify the images of MNIST and colorectal cancer (CRC) datasets and compare the spiking convolutional neural networks (SCNNs) with the convolutional neural networks (CNNs).

In this work, the reader can find a brief introduction to machine learning, followed by a summary of the state of the art of simulators and of the spiking neural networks models. The thesis continues with the analysis of four SNNs and in the end, the performed tests and their results are presented.

1.1 Brief introduction to machine learning

Studies of Machine Learning (ML) appeared in the first decades of the XX century. Machine learning uses statistical methods for improving the performance of patterns recognition [19]. The machine learning methods can be divided into two main categories: the unsupervised methods and the supervised methods. Artificial neural networks (ANNs) are in the subset of supervised methods.

Artificial neural network create mathematical models during the training phase. In this phase the classification error between the model response and the desired response is calculated and this error is used to adjust the model by modifying its internal parameters called hyperparameters. The training phase is performed by giving as input one stimulus (data that the network have to analyze as images and

signals) at a time: the training stimuli are contained in a training set. After every stimulus or every set of input, some hyperparameters are updated. This step allows the algorithm to predict the output of a given input without explicit instructions [23].

Artificial neural networks are inspired by the animal brains: the main goal of the ANN is to simulate the brain and solve the tasks with the same or better results compared to the biological brain [26].

The basic unit of the brain of the animals is the neuron. This type of cell responds to the electrical and chemical stimuli. Neurons are connected to each other by synapses, which are characterized by an efficiency (called weight). Interconnected neurons create neural pathways and neural circuits. A neural pathway connects a part of the brain to another one in order to send a message between two regions of the brain, while a neural circuit carries out a specific function [17]. Multiple neural circuits create large scale brain networks, that interact with each other allowing to perform complex functions [16].

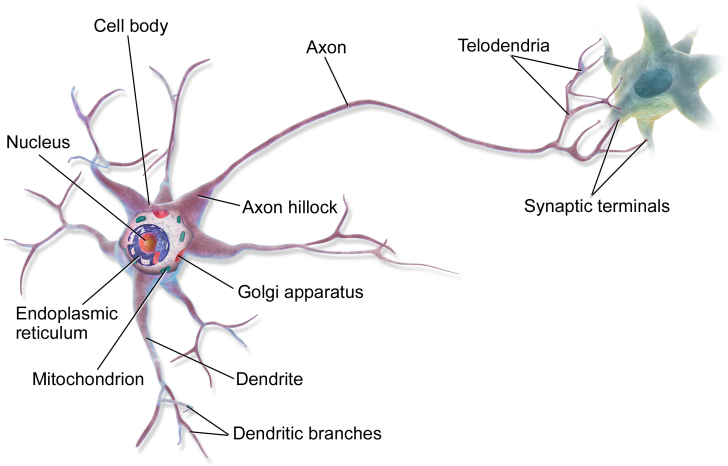

Figure 1.1: Neuron anatomy. We can see the junction between two neurons (called synapse), the cell body, the axon, where the signals are sent, and the dendrites, where the signals are received (from [17]).

The structure of the biological brain is mimicked by artificial neural networks: the same subunits interact with each other in different and complex ways in order to achieve a specific result. The fundamental elements of ANNs are the neurons, that are responsible for the basic operations. They behave in a different way depending on the neuron model, each model is characterized by an activation function, and they can be activated or inactivated from the input they receive. Group of neurons can be organized in a different way: a basic group of neuron can form a layer, different layers are generally present in a neural network. In general, each layer corresponds to a specific elaboration step. An elementary neural network is usually

composed of three layers:

- Input layer: it is composed of receive an input from the user;
- Hidden layer: it elaborates the data (stimuli) received from the input layer;
- Output layer: it gives the result of the computation.

This elementary structure can be expanded by adding complexity to the layers and by adding more layers to the structure.

The complexity of the model used to describe the behavior of the neuron allows us to identify three generations of ANNs [26].

First generation of ANNs

The first generation of ANNs (called also perceptrons or thresholdgates) is based on the neuron described by McCulloch–Pitts:

$$y = \begin{cases} 1, & \text{if } h = \sum_i x_i w_i \geq u \\ 0, & \text{otherwise} \end{cases} \tag{1.1}$$

where y is the output of the neuron, that can be 0 (stands for not activate) and 1 (stands for activate), h is the state of the neuron, w_i is a synaptic weight, x_i is an output of the previous neurons and u is the threshold, that at the beginning was set to 0, and then can be set by the user (introduction of a bias), giving more flexibility to the network [48].

The first generation of ANNs doesn't work well with continuous data.

Thanks to the binary nature of this ANN, the theory of Hebb[1] is used for training this type of NN: it claims that if a pre-synaptic neuron (that acts as input) causes the activation of the post-synaptic neuron (it receives the input stimulus), the synaptic weight is enhanced [48].

Second generation of ANNs

The second generation of ANN is the evolution of the first one and it is based on a neuron with a continuous activation function g. The neuron is defined as:

$$y = g(\sum_i x_i w_i - b) \tag{1.2}$$

[1]Donald Olding Hebb (July 22, 1904 – August 20, 1985) was a Canadian psychologist that works in the neurophysiology field. He is popular for the Hebbian rule, which it asserts that if a pre-synaptic cell helps to fire a post-synaptic cell, the synaptic is strengthened [20].

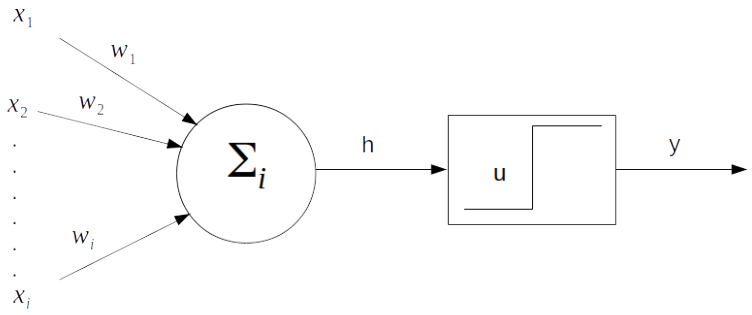

Figure 1.2: Representation of the neuron model of McCulloch–Pitts.

where g is often a sigmoid function and b is a function translation factor called bias depending on which the function is traslated.

With this type of ANNs the rule of Hebb is not used. New types of algorithms are employed, such as gradient descendent, that allows to modify the weights in the last layer using the error between the predicted output and the expected one. With the gradient descendent algorithm, there is backpropagation (BP) algorithm, that allows to propagate the error in the hidden layer [48]. This method allows the training of ANNs containing many hidden layers (deep learning) [48]. The second generation of ANNs is computationally more powerful: it can simulate all the boolean functions and some of them can be created with less neurons than the ones used in the first generation of ANNs [36]. Moreover, it has been demonstrated that an ANN of second generation can approximate, arbitrarily well, any continuous function with just one hidden layer [48].

Third generation of ANNs

The third generation neural networks have a new type of neural model called spiking neuron. This type of neurons is modelled with a system of Ordinary Differential Equation (ODE), which allow simulating the spike dynamics of the biological neuron. The presence of the spikes causes the information to be encoded over time: the codification is determined by the firing rate or by the period of time between two consequent spikes [36]. This type of ANN can create any function made from the first two generations using a single hidden layer and, in some cases, with fewer neurons [48].

This project focuses on the third generation ANNs: some tools (simulators) for running these ANNs are presented in the next section.

1.2 Simulators

Different simulators were implemented for running spiking neural networks, in particular, they can be divided into two classes: software simulators and hardware simulators. The hardware simulators can speed up the mathematical operations of the spiking neural models, being simple hardware accelerators. The neuromorphic hardware is an hardware accelerator inspired by human brain, it either implements the neural models through the hardware or builds a communication system that allows "real-time" communication between neurons.

Both software and hardware simulators will be described in detail in the following paragraphs.

1.2.1 Software simulators

There are a lot of software simulators, the most common ones are: NEST [30], Brian [31] and NengoDL [41] simulators. These three tools have a different philosophy.

NEST was developed by Dr. Marc-Oliver Gewaltig and Dr. Markus Diesmann and it is created for simulating huge spiking neural networks. This simulator includes many already implemented neuron models and it makes it easy to implement clusters of neurons and the connections between them. The connections (synapses) can be static or dynamic and they are characterized by two parameters: weight and delay. The weight is a value that will be multiplied by the value of the current corresponding to the input stimulus of the neuron, while the delay is a value that adds a delay between the output current of the pre-synaptic neuron and the input current of the post-synaptic neuron. As previously mentioned, there are static and dynamic synapses: the static synapses have a fixed weight and delay, while the dynamic ones have a weight or a delay that can change during the simulation.

Brian has a different philosophy from nest: while this is more focused on the networks, Brian gives more attention to the neurons. In fact, it allows the researcher to implement the neuron model by writing the mathematical equation in mathematical form, so that more time can be spent on the study of the model and not on the implementation.

NengoDL links these neuromorphic simulators (Nest and Brian) to the deep learning tools developed for the second generation ANNs (as TensorFlow [13]). These include a set of ML models and algorithms that help the researcher to build new ML architectures. Moreover, NengoDL wants to make the implementation of

hybrid neural networks (containing both continuous neurons and spiking neurons) easier.

1.2.2 Neuromorphic hardware

In the previous section SNN have been introduced. Companies are investing in these tools because they are hardware-friendly: in the last few years, many companies and universities have been developing different hardware accelerators (called neuromorphic hardware). These devices allow for the simulations of the neural networks to be faster and less power-consuming [46]. In the following section, we are going to explore some neuromorphic hardware made by different companies and universities.

The University of Manchester, that is in the Human Brain Project (HBP) which is a European Union project that links over more than 100 infrastructures, including universities, research centres and teaching hospitals with the aim of advance in the fields of neuroscience, medicine and computing, designed a massively-parallel computer named SpiNNaker. The architecture of this device is inspired to the human brain which is formed by billions of units that communicate with each other through unreliable spikes. The first goal of the developers of this neuromorphic hardware is to create a huge computer capable to simulate massive neural networks in real time. This is so useful for roboticists, neuroscientists and computer scientists. The second aid is to create a new supercomputer architecture that is energy-efficient [9]. SpiNNaker has a hierarchic architecture and the whole structure is composed by 1,036,800 ARM9 cores and 7TB of RAM distributed in 57,000 nodes. The nodes communicate with each other through packages of 40 bits. The system has a communication speed of 5 billions of spikes/s. 48 nodes are zipped in one PCB in an hexagonal mesh. The whole system requires 1200 board and it can consume, at most, 90 kW of electrical power [10]. The base unit of SpiNNaker is a System-on-chip, this is Globally Asynchronous Locally Synchronous (GALS) composed by 18 ARM968 cores. These cores have a low energy consumption, but they have limited computational power. The SpiNNaker chip (shown in fig. 1.3) is composed of a router, at the centre, surrounded by 18 cores, of these 17 are used for the task, while one is used for checking the fault tolerance and manufacturing yield-enhancement purposes. Spinnaker makes use of an efficient brain-inspired network, so it uses package-switched network because it allows a high number of connections. Every chip uses his own clock for communicating, performing an asynchronous communication and, due to the fact that it is in a hexagonal mesh, every chip is linked to 6 other chips [11].

BrainScaleS is another neuromorphic hardware and it is formed by an uncut wafer and can emulate adaptative spiking neurons, that have low computational cost and no-fixed spike threshold, and dynamic synapses. This wafer is composed of 450 chips called HICANNs (High Input Count Analog Neural Network chips) [44].

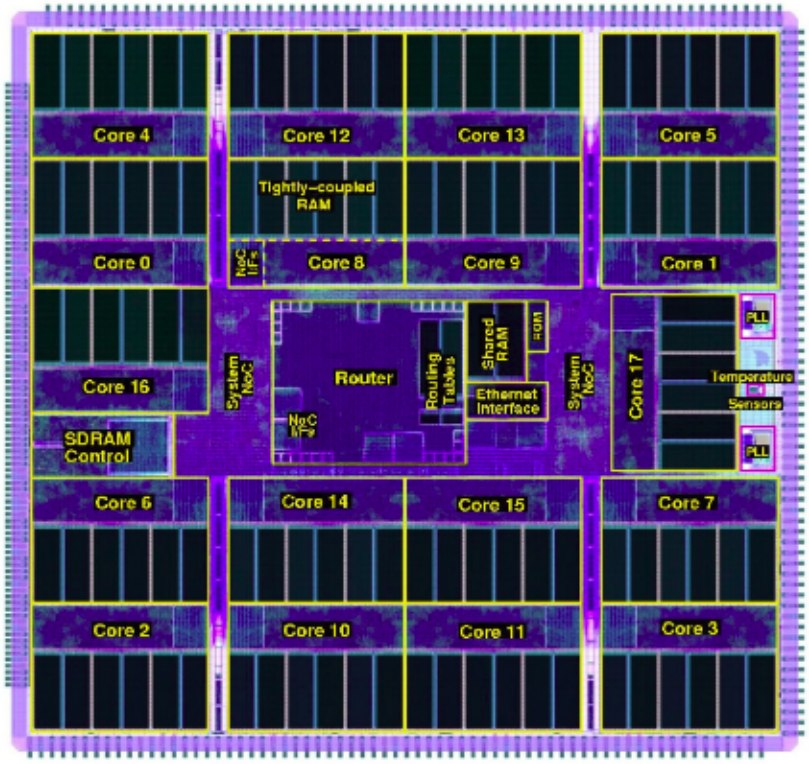

Figure 1.3: In the figure is represented the SpiNNaker multicore System-on-Chip (from [11])

From the simulation made on the wafer, it has emerged that this device operates at accelerated biological time with a factor of $10^3 \div 10^5$ compared to human brain [43] and it consumes less than a normal simulator, about $5.6\,W$ for wafer. Each wafer has $4 * 10^7$ synapses [43] and $260 * 10^3$ of neurons [44]. More wafer can be linked together: BrainScaleS is composed of 20 wafers [1].

TrueNorth is a chip developed by IBM and, as SpiNNaker, it is inspired to the mammalian brain but, while the second one is flexible, the first one is more dense and low-power [4, 12]. It is composed of 4096 cores and it can simulate 1 million neurons and 256 millions of synapses, all in just $28\,nm$. This chip has 5.4 billions of transistors and it consumes less than $100mW$. This board is so scalable in fact, in a DARPA program called SyNAPSE, they want to assemble up to 4096 TrueNorth chips so they can simulate 4 billions of neurons and 1 trillion synapses consuming less than $4kW$ [6].

Loihi is neuromorphic hardware created by Intel and launched in January of 2018 [7]. This chip is so scalable and it can have up to 4,096 cores per chip, while every core is composed of 128-neuromorphic cores, each of them can simulate up to 1,024 neurons so every core can simulate about 130,000 neurons and 130 millions of synapses [7, 24]. This architecture can link up to 16,384 chips that communicate

Figure 1.4: The figure on the left is the BrainScaleS system, composed by 20 wafer while, the figure on the right is SpiNNaker system. (from [2]).

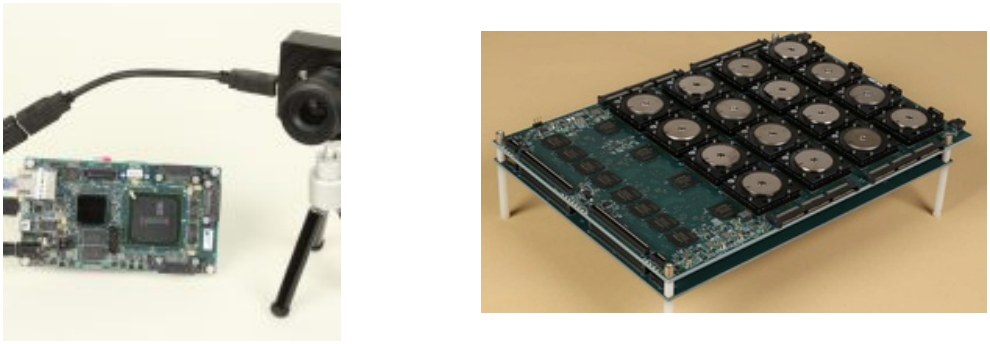

Figure 1.5: The left figure (from [5]) represents the TrueNorth chip, while the right one (from [12]) is a board of SyNAPSE project with 16 TrueNorth chips.

with each other [7].

1.2.3 PyNN

PyNN [25] is a common frontend for a lot of simulators (nest, neuron, brian, SpiNNaker, etc...). Every neuromorphic simulator uses its own programming language and/or syntax, so it is difficult porting the network between two different simulators, this operation can be useful for comparing the implementation difference or for checking if there are some bugs in the simulators. Moreover, every simulator has its positive sides and negative ones: often they are a tradeoff between scalability, flexibility and efficiency then, depending on what we are going to implement, important to use one simulator rather than another one. In this far west PyNN comes in to put some order, in fact his syntax is the same for a lot of neuromorphic simulators (making easy the porting between different simulator), it is scalable (it is easy to work with neurons populations and with the single cells) and it let the researcher implements the code in a hybrid way (it means that we can mix the

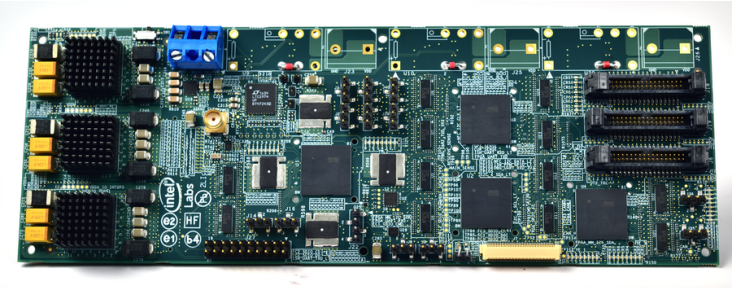

Figure 1.6: In the figure is represented the Loihi test board. It is composed of 512 neuromorphic cores, it means that it can simulate up to 524,288 neurons (from [7]).

PyNN syntax with native simulator syntax), the price of the last feature is lost of flexibility.

Chapter 2

Spiking neural networks

2.1 Models

The unit of the SNN is the neuron, this one is created to be as the biological one. It processes the information that arrives from the other neurons (pre-synaptic neurons) and it sends the spikes trains (sequence of spikes) to the other cells (post-synaptic neurons) through the axon. The probability that the neuron fires, increase with increasing the membrane potential: there will be a fire, or more then one, only if the membrane potential reaches a threshold (called spike threshold) [39].

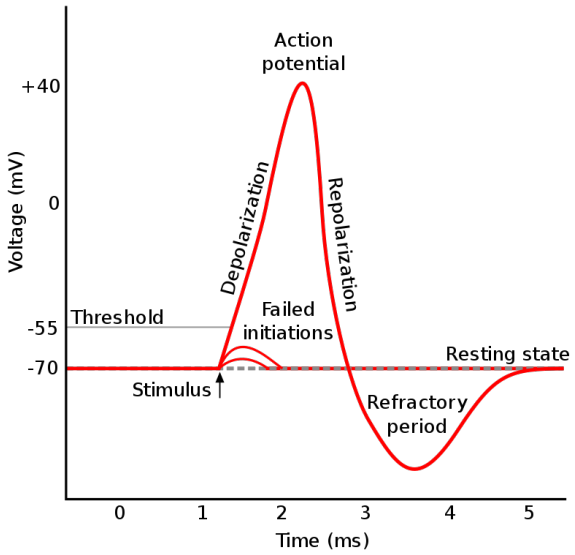

Figure 2.1: Representation of an action potential (from [18]).

In the figure 2.1 we can see an approximative plot of an action potential that is the spike of a neuron. The depolarization occurs when the input stimulus is high

enough to reach the spike threshold. When it happens the action potential rises and, until the ending of the repolarization, the neuron is in the absolute refractory period, in which it cannot fire again. After the repolarization, which ends with the crossing of the resting potential, the hyperpolarization starts in which is more difficulte that the neuron fire, but it is possible. This time period is called relative refractory period. In the end, the cell returns to resting potential.

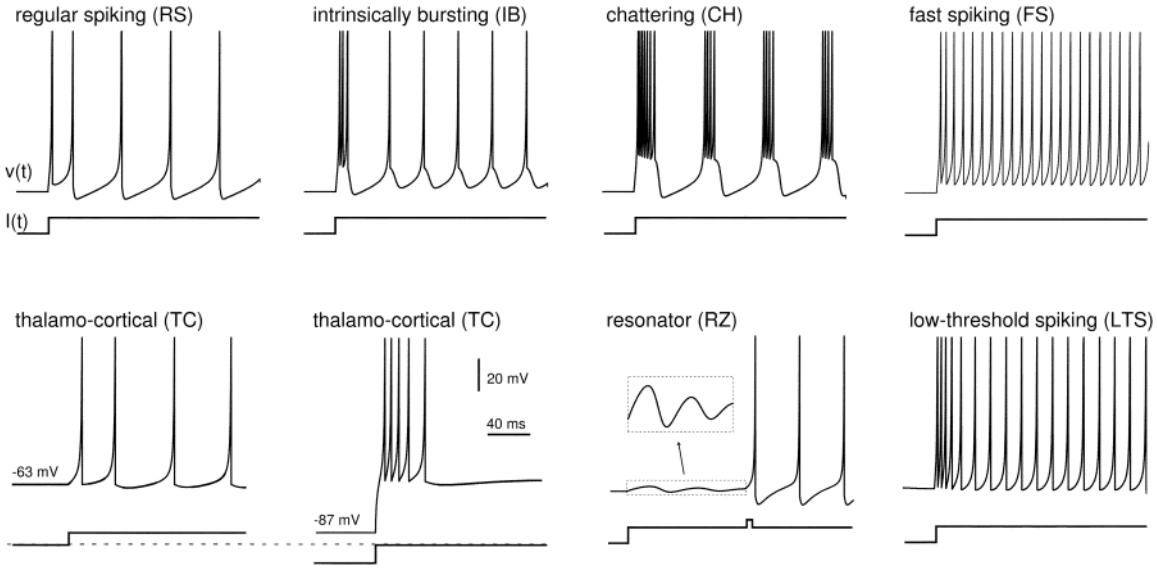

Figure 2.2: Representation of some biological spike trains, the main behaviour are from excitatory cell family (RS, IB, CH), inhibitory cell family (FS, LTS) and thalamocortical cell family (TC) (from [33]).

The first that models the biological neuron were Alan Lloyd Hodgkin and Andrew Huxley, who won the Nobel prize in 1963 [46]. They create an high accuracy biological model neuron, using a system of ODE. Due to it complexity, the neuron model has an high computational cost therefore many neurons models were created:

- Hodgking-Huxley model (HH) (section 2.1.1)
- leaky integrate-and-fire model (LIF) (section 2.1.2)
- spike response model (SRM) (section 2.1.3)
- Izhikevich neuron model (IZK) (section 2.1.4)

2.1.1 Hodgking-Huxley model (HH)

All the neurons models derive from the HH model, which is so complex because it models all physiological aspects.

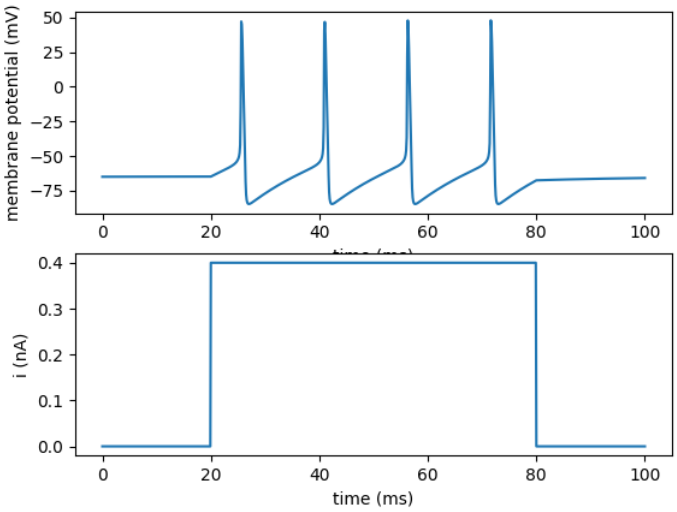

Figure 2.3: This is an example of the trend of the membrane potential of HH neuron (top) with a input current (bottom).

$$
\begin{cases}
\frac{a}{R_i}\frac{\partial^2 v(z,t)}{\partial z^2} = C_m \frac{\partial v(z,t)}{\partial t} + (v(z,t) + V_{Na}) g_{Na}(v) + (v(z,t) - V_K) g_K(v) + (v(z,t) - V_L) g_L \\
g_K = \bar{g_K} n(v,t)^4 \\
\frac{dn(v,t)}{dt} = \alpha_n [a - n(v,t)] - \beta_n n(v,t) \\
\alpha_n = f_{\alpha_n} \frac{v + V_{\alpha_n}}{e^{\frac{v + V_{\alpha_n}}{V_{\alpha_n}}} - 1} \\
\beta_n = f_{\beta_n} e^{\frac{v}{V_{\beta_n}}} \\
g_{Na} = \bar{g_{Na}} m(v,t)^3 h(v,t) \\
\frac{dm(v,t)}{dt} = \alpha_m [1 - m(v,t)] - \beta_m m(v,t) \\
\frac{dh(v,t)}{dt} = \alpha_h [1 - h(v,t)] - \beta_h h(v,t) \\
\alpha_m = f_{\alpha_n} \frac{v + V_{\alpha_{m1}}}{e^{\frac{v + V_{\alpha_{m1}}}{V_{\alpha_{m2}}}} - 1} \\
\beta_m = f_{\beta_m} e^{\frac{v}{V_{\beta_m}}} \\
\alpha_h = f_{\alpha_h} e^{\frac{v}{V_{\alpha_h}}} \\
\beta_h = f_{\beta_h} \frac{v + V_{\beta_{m1}}}{e^{\frac{v + V_{\beta_{m1}}}{V_{\beta_{m2}}}} - 1}
\end{cases}
\tag{2.1}
$$

Due to his complexity, it is impossible to create a large SNN with this neuron model.

2.1.2 Leaky integrate and fire model (LIF)

The LIF neuron is a semplification of HH neuron [37].

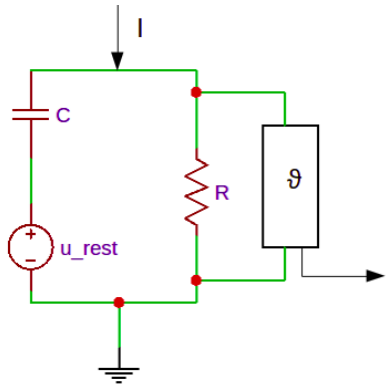

Figure 2.4: Electronic circuit of LIF neuron
Electronic circuit that represents an LIF neuron with a input current.

$$\tau_m \frac{du}{dt} = u_{rest} - u(t) + RI(t) \tag{2.2}$$

where $\tau_m = RC$ is the time constant and it sets the speed of charge and discharge of membrane potential $u(t)$. The neurons fires when $u(t)$ reaches the threshold (θ), at time $t^{(f)}$. After firing, the membrane potential is set to u_{rest} (often fixed to 0). The absolutely refractory period is forced to $u = u_{abs}$ for a certain period of time (d_{abs}). After that, the membrane potential will return to $u = u_{rest}$ [37].

2.1.3 Spike response model (SRM)

Spike response model (SRM) is a model of a neuron that doesn't depend on the voltage, but just on $(t - \hat{t})$, where $\hat{t}$ is the time of the last spike generated from it, moreover it is not described by a difference equation, but by an integral [38]:

$$u_m(t) = \eta(t - \hat{t}_k) + \sum_j w_{jk} \sum_f \epsilon_{jk}(t - \hat{t}_k, s) + \int_0^\infty \kappa(t - \hat{t}_k, s) i_{ext}(t - s) ds \tag{2.3}$$

where $u(t)$ is the membrane potential, the $\epsilon_{jk}(s)$ describes the trend of post-synaptic potential: j is the pre-synaptic neuron, while k is the neuron in which we are calculating the membrane potential. t_j^f is the spike train incoming from neuron j (f is the number of spikes) and $s = t - t_j^f$. $\kappa(t - \hat{t}_k, s)$ is the response to an inject current $i_{ext}(t)$. $\eta(t - \hat{t}_k)$ represents the trend of the action potential (when the membrane potential reaches the spike threshold θ) and the hyperpolarization after

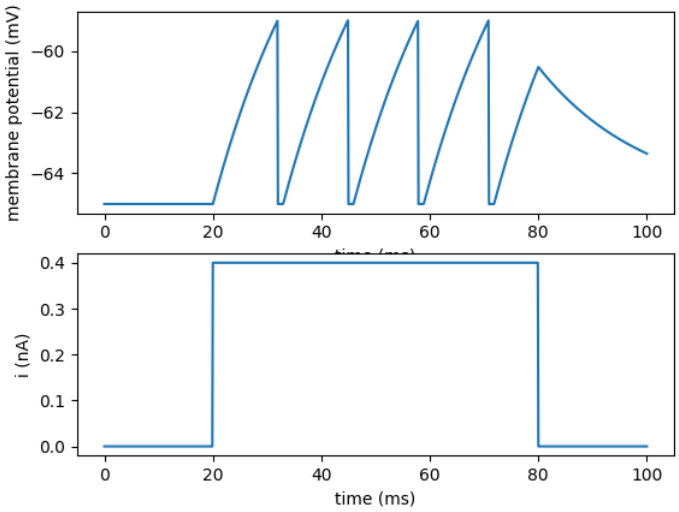

Figure 2.5: Trend of membrane potential of LIF neuron

the fire. Where no input occurrences, $u(t) = E_{rest}$ [38]. A feature of this model is that the spike threshold is not fixed, but it depends on time of the last spike: $\theta = \theta(t - \hat{t}_k)$ [38].

2.1.4 Izhikevich model (IZK)

Izhikevich model is a good tradeoff between biological reliability and computational cost[37]. This model is a simplified HH model, in fact it uses just two differential equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u \tag{2.4}$$

$$\frac{du}{dt} = a(bv - u) \tag{2.5}$$

$$\text{if } v \geq \theta \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.6}$$

where the equation 2.6 represents the post-spike behaviour of the neuron. v is the variable that represents membrane potential, while u is the recovery variable that describes the activation of K^+ ionic currents and the inactivation of Na^+ ionic currents. a, b, c and d are just dimensionless parameters [33] and θ is the spike threshold.

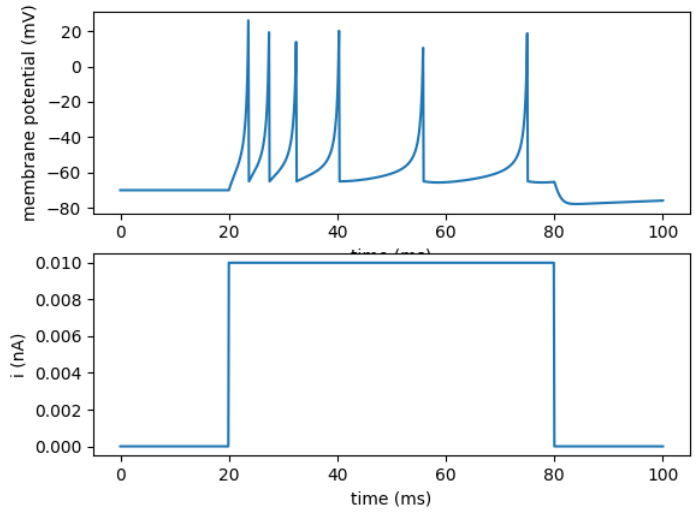

Figure 2.6: Trend of membrane potential of Izhikevich neuron (top) with a input current (bottom).

2.1.5 Neurons comparison

The HH neuron model is the most biological reliable model, while the LIF model is one of the easiest model to implement. The problem of the LIF model is that it is not able to reproduce all the biological spike trains that the first model can do.

In the figure 2.7 there are 20 different spikes trains. All of them can be created with the HH model, while just three of them can be reproduced by IF model. If we analyze the computational cost, HH model is the heaviest one, while IF is the lighter: for simulating 1ms of HH neuron model the CPU uses 1200 FLOPS[1], while for the second model just 5 FLOPS. Variants of LIF model need about ten FLOPS (see figure 2.8) [37]. In figure 2.9 we can see which kind of spikes trains the neurons can repeat and if the model has biological meaning. Therefore, depending on what we are going to do we can choose the appropriate model to simulate: if we are interested to a biological behaviour of a few neurons, we can use the HH model while, if we are going to simulate an SNN with many layers and neurons, we should use LIF model or IZK model.

[1]FLoating point Operations Per Second (FLOPS): it represents the operation number with the floating point performed by the CPU in one second.

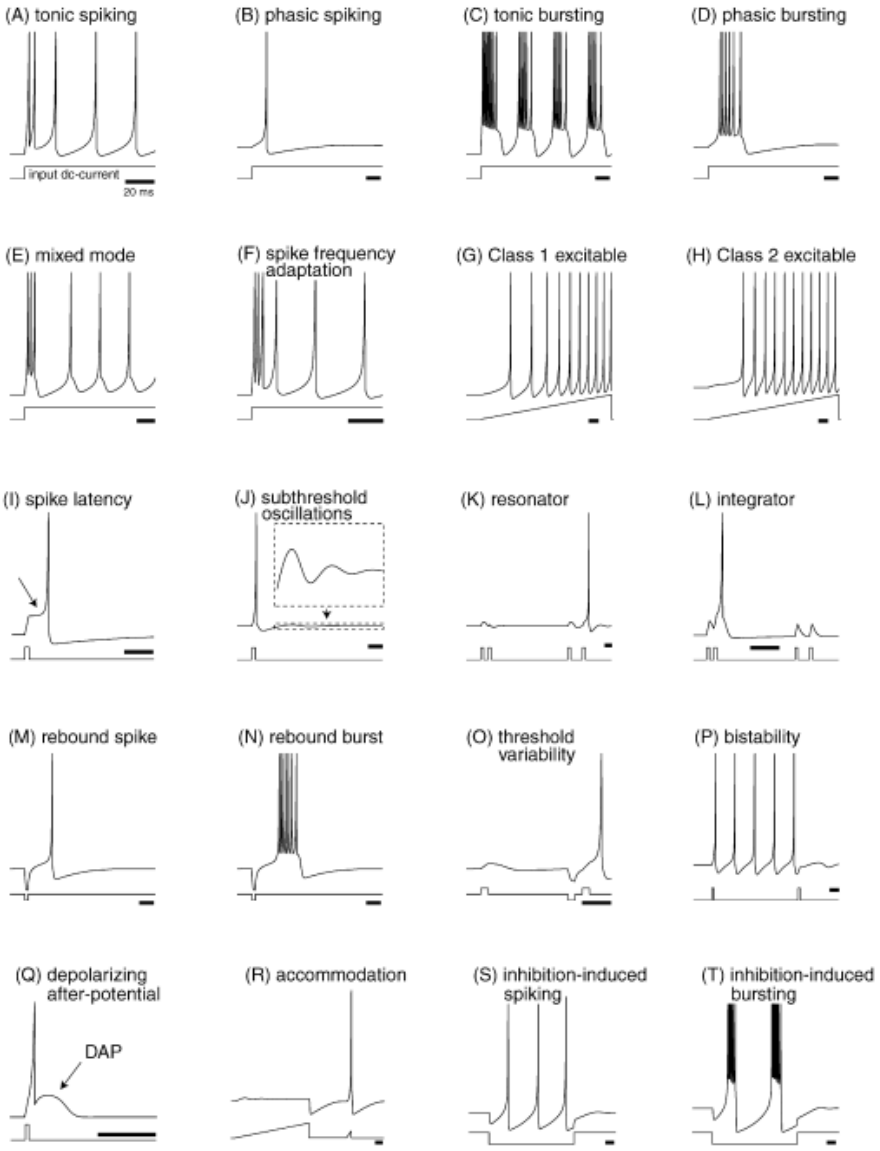

Figure 2.7: Some spike trains used for comparing some types of neurons (from [34]).

2.1.6 Synapse

The synapse is the junction between two neurons or between a neuron and a muscle/glandular cell. The message, when passes through a synapse, opens the ionic channels creating a postsynaptic current (PSC) and changing the membrane postsynaptic potential (PSP) [38]. In case of there is an excitatory signal, we have a depolarization of the membrane potential (EPSP) while, if there is an inhibitory signal, we have an hyperpolarization of the post-synaptic cell(IPSP) [38].

The PSC ($i_{syn}(t)$) is often modelling in function of the synapse conductance

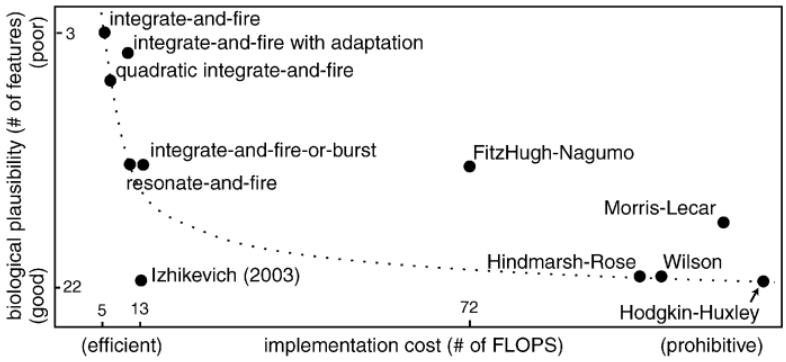

Figure 2.8: Graph of computational cost vs biological reliability (from [34]).

Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-induced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Figure 2.9: In this figure is shown which kind of behaviour the neurons model can have and if they have biological meaning. Blank cells mean that the author wasn't able to find the right parameters but in theory, they exist (from [34]).

($g_{syn}(t)$) and the PSP ($u_m(t)$)[38]:

$$i_{syn}(t) = g_{syn}(t)(u_m(t) - E_{syn}) \tag{2.7}$$

where E_{syn} is a synaptic parameter, which is -75V for inhibitory synapses and 0 for the excitatory one. In this model, g is only in function of the time, but this is not

the real behaviour of the biologic conductance because it also depends on the pre- and post-synaptic behaviour [38]. The simplification model (just in function of the time) is [38]:

$$g_{syn} = \sum_f w[e^{\frac{-t+t^f}{\tau_d}} - e^{\frac{-t+t^f}{\tau_r}}]H(t-t^f) \tag{2.8}$$

where w is the synaptic efficacy, τ_d is the decay time, τ_r is the rise time and t^f is the time of arrival of the pre-synaptic action potential. $H(t-t^f) = 1$ if $t > t^f$, 0 elsewhere. In the excitatory synapses, τ_r is really small in comparison to τ_d so we can rewrite the equation 2.8 as [38]:

$$g_{syn} = \sum_f we^{\frac{-t+t^f}{\tau_d}} H(t-t^f) \tag{2.9}$$

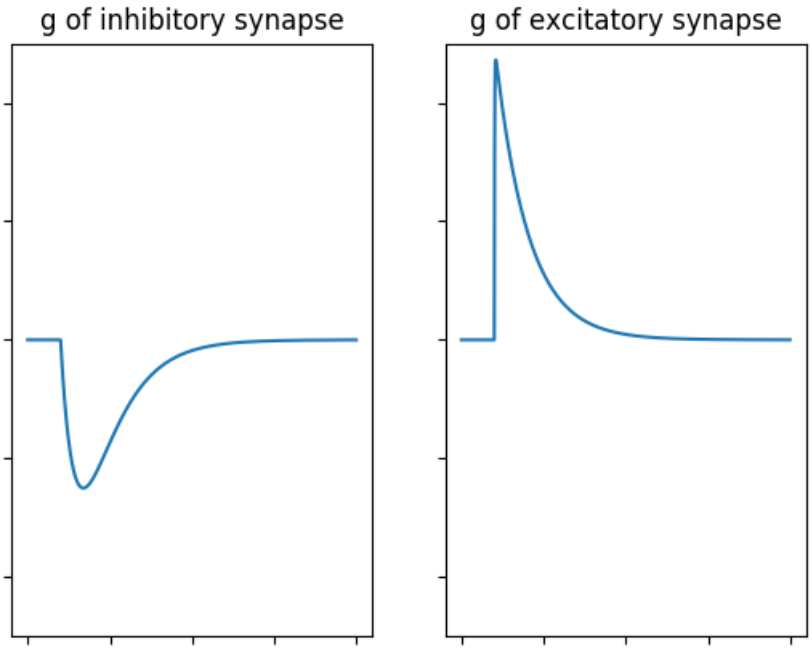

Figure 2.10: Trend of conductance of synapses.

2.2 Encoding

2.2.1 Input

There are many ways to transform a image in a spikes train, we are going to analize two of this methods: Poisson distribution and neural gas.

Poisson distribution

In this type of coding, the value of the pixel is seen as the instantaneous firing rate [29]. The bins are the maximum number of spikes that the neurons can fire in a single exposition time:

$$bins = \frac{t_{sim}}{dt} \tag{2.10}$$

where t_{sim} is the exposition time of an individual image and dt is the period of time where the neuron can spike at most once. The probability that the unit fires in dt is defined as rdt, where r is the intensity of the pixel. It has been shown that the probability of having n spikes in a time t, is the Poisson distribution [29].

Data: images to coding
Result: Poisson distribution of spike
$bins = \frac{t_{sim}}{dt}$;
x=random uniform distribution of image size;
for *all pixel* **do**
  if $rdt > x$ **then**
    1 spike
  else
    nothing
  end
end

Algorithm 1: Pseudocode of Poisson distribution of spike train

Neural gas

With this method the firing rate is no more proportional to the intensity of the pixel, but it depends on the distance between the virtual receptors (VRs), that are excitable neurons that have to traduce the information from distance VR-stimulus to time, and the image. With this strategy, the image is a point in a n-dimensional space (where n is the number of pixels for each image). The coordinates of the VRs are chosen to cover all the space of the stimulus and neural gas is chosen for this task.

In the neural gas algorithm, the VRs are initialized in a random way, then they are attracted by the images (these ones are presented to the VRs one-by-one): higher is the distance, lower will be the move (see fig: 2.11 and pseudocode: 2).

After train the VRs, the distance between the sensors and the stimulus should be calculated. This task is reached by using the Manhattan distance (that is calculated by adding the absolute value of the difference of the coordinates) normalized between 0 and 1:

$$r_{ij} = 1 - \frac{d_{ij} - d_{min}}{d_{max} - d_{min}} \tag{2.11}$$

where d_{ij} is the Manhattan distance between the j-th image and the i-th VR, d_{min} and d_{max} are the minimum and the maximum distance in the whole training set, respectively. The rate of each transducer will be:

$$\rho_{ij} = r_{ij}(\rho_{max} - \rho_{min}) + \rho_{min} \tag{2.12}$$

where ρ_{ij} is the firing rate of the i-th VR when the j-th image is shown as the input of the network. ρ_{max} is the maximum frequency and ρ_{min} is the minimum frequency of the VRs. The gamma distribution is created starting from this frequency.

Data: images: $D = \{\vec{x_1}, \vec{x_2}, \vec{x_3}, ...\vec{x_n}\}$ and VRs: $W = \{\vec{w_1}, \vec{w_2}, \vec{w_3}, ...\vec{w_p}\}$
Result: The coordinates of VRs
for *all* $\vec{x} \in D$ **do**
  for *all* $\vec{w} \in W$ **do**
    $d \leftarrow |\vec{x} - \vec{w}|^2$
  end
  sort all distances in ascending order;
  assign a rank r to all VRs;
  for *all* $\vec{w}$ **do**
    $\vec{w_j} \leftarrow \vec{w_j} + \alpha e^{\frac{r_j}{\lambda}}(\vec{x} - \vec{w})$
  end
end

Algorithm 2: Pseudocode of neural gas, where α is the learning rate and λ is the aka connectivity.

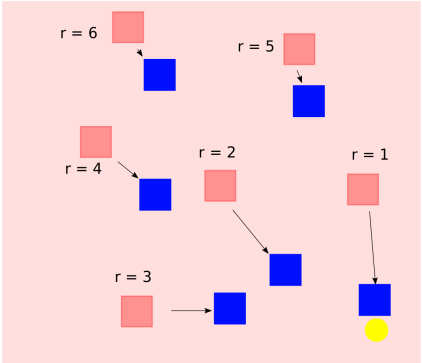

Figure 2.11: The red squares are the VRs before showing an image (yellow circle). When the image is displayed, the neural gas algorithm moves the VRs in a new position depending on the distance between the image and the VR: blue squares (modify from [14]).

2.2.2 Processing

At the beginning of the 1920s, the scientists thought that the neurons put the information in the firing rate. This type of communication was supposed and then demonstrated by Adrian and Zotterman who saw that the firing rate of the cutaneous receptors of frogs increases with the increasing of employed pressure [39]. At the beginning of the 2000s, the scientists notice that some parts of the brain do not zip the information in the firing rate but they put it in the precise timing of the action potential: the message is in the time, not in the frequency [39]. In support of this theory, there are two important facts:

- many behaviours are too fast for having a communication in the rate code (the timing window should be too large);
- the frequency doesn't hold all the information: the scientists discovered that in some parts of the primary auditive cortex, the neurons can coordinate the spike times of the neighbours. The result is that these clusters have some bursts in the signals, also if the mean frequency doesn't change[39].

Time to first spike

In this kind of encoding the information is in the delay between the start of the stimulus and the fire of the neuron. It has been shown that this latency carries double of the information compared to the shape of the spike and about the same info of the spike trains [39]. In this type of coding, the neuron can fire just once during the stimulus, and if it fires more then once, the other spikes are ignored [42]. This coding is so easy to implement: just one neuron with inhibitory feedback [42].

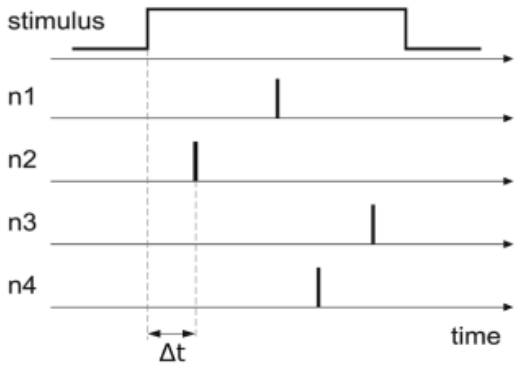

Figure 2.12: Example of time to first spike coding (from [39]).

Rank-order coding (ROC)

Another type of coding is rank-order coding (ROC) in which we check the firing rank: the stimulus is encoding in the order of the spikes [39]. In this kind of codify, every cell can fire just ones for every stimulus. ROC was found in the visual cortex of primates, and it allows the ultra-fast communication [39]. This encoding can allow $N!$ fire orders and it means that ROC can transmit $log_2 N!$ bits of information (where N is the number of the neurons) [47].

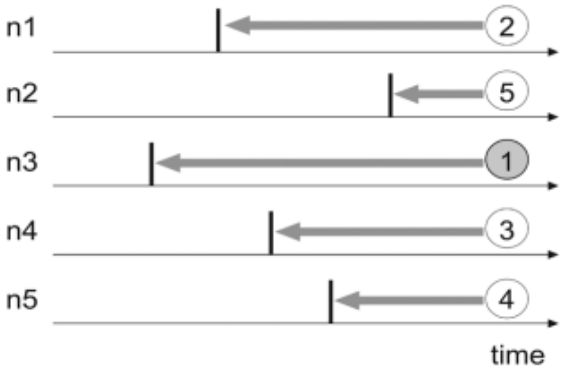

Figure 2.13: Example of rank-order coding (from [39]).

Latency code

In this coding the information is in the time between spikes of different neurons[2]. The amount of information that the latency code can transmit depends on the number of the neurons and on the timing windows: $N * log_2 t$ [47].

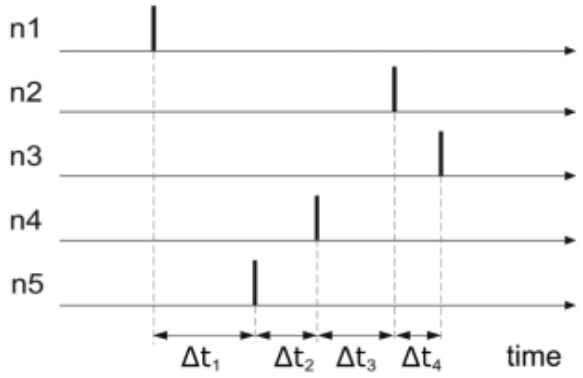

Figure 2.14: Example of latency coding (from [39]).

2.3 Neural Engineering Framework (NEF)

Neural Engineering Framework (NEF) is a platform that helps the researcher to implement a brain. We are going to briefly analyze it because it is the NengoDL substrate. First of all, it is important to talk about the tuning function. This is different from the response function (function that characterizes the neuron) because, while the second one depends on intrinsic parameters of the cell, the tuning function depends on intrinsic and extrinsic values as can be the position in the brain (depending on the position, a neuron can have a different stimulus to the same input, this happens because different neurons are located in various positions with different neighbours) [28]. The tuning curves can be represented as a function that binds input intensity to the firing rate (see fig. 2.15 and 2.16) and they are used for input encoding.

For reading the results, we have to decode the output. This phase is performed by a weighted sum of PSC of the neurons, in which the weights research is an optimization problem (the optimization algorithms decrease the error between the input and the output): the weights have to represent the input signal. Therefore, more important will be the neuron (for the representation of the stimulus) greater

[2]The relative time is so important in the communication between neurons because the "value" of the synapses is increased or decreased by the latency between pre- and post-synaptic spikes [39].

will be the weight and more neurons will be in the population and more precise will be the output [28].

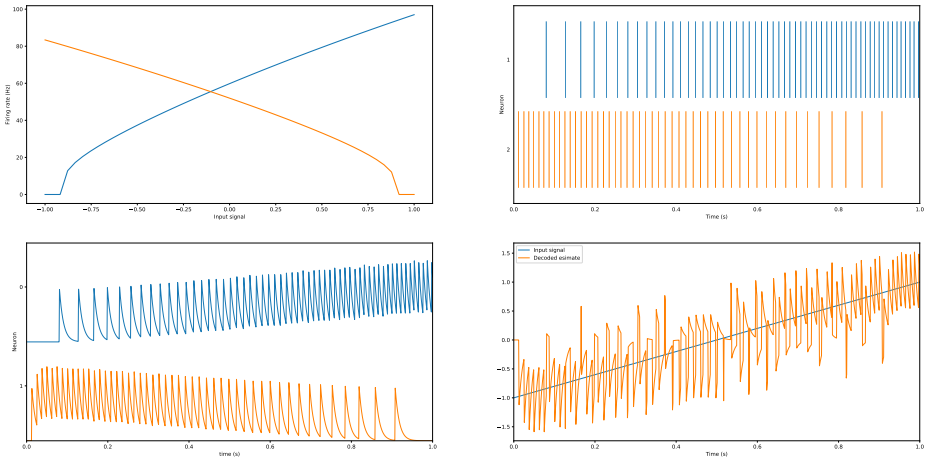

Figure 2.15: In clockwise order: A) Tuning curves of 2 neurons. B) The spikes trains generated from the input signal. Comparing A and B, we can see the blue one that increases the firing rate following the trends of its tuning curve. Idem for orange one. C) It is shown the input signal in blue and the output in orange. The input is just a line that increases from -1 to 1 in 1 second, while the output is a signal generated from a weighted sum of figure D. D) It is shown the PSC, this signal is obtained from applying the α-filter to figure B (modify from [8]).

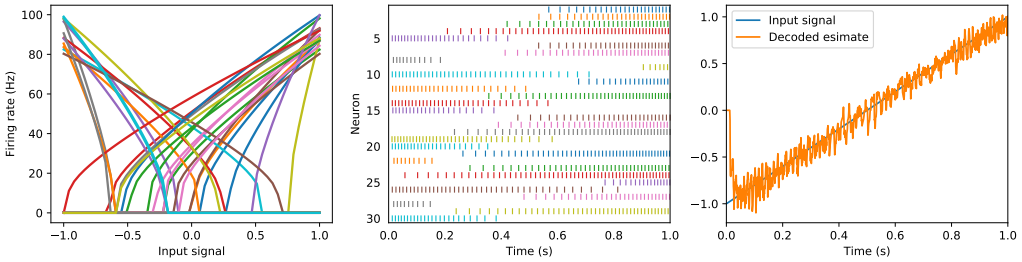

Figure 2.16: As we can see from this figure, increasing the neurons number, we can have a better representation of the output (from [8]).

2.4 Network topologies

There are different classes of neural architectures [39]:

1. feedforward networks: in which the connections are from the input layer to the output one, without feedback connections.;

2. recurrent networks: in this class, the connections are not strictly in one direction;

3. hybrid networks: these networks have some sub-recurrent population of neurons binds to the other with a feedforward connection;

4. convolutional neural networks: these networks are so famous, and they are a subset of feedforward networks but, due to their popularity, we are going to talk about them like if they are a different class.

In the biological brain, feedforward networks are in peripheral places. This type of neural circuits is easy to train [38] (see fig. 2.17). The second class of ANN can solve tasks more complex than the feedforward networks, but they are also more difficult to train [39] (see fig. 2.18). In this kind of circuits, lateral inhibition is important because it decorrelates the input signals [38]. There are different types of hybrid networks, but the most important are Synfire chain (see fig. 2.19) and Reservoir computing (see fig. 2.20). In the Synfire chain, the spikes flow from a layer to the next one in a synchronous way (feedforward connections), but in the layers, there are subpopulations connect to each other with recurrent connections [39]. The reservoir computing, called also Liquid State Machine (LST), has a fixed recurrent structure (reservoir), and a population of output neurons called readouts, usually connect in a feedforward way. With these types of network, the training of the recurrent network is bypassed because only the feedforward connections are trained [38]. The last family of neural networks are the convolutional neural networks (CNN). The popularity of this kind of networks is due to the high performance that they can reach and they can express better their potential in the image classification problems. The structure of the CNN is strictly hierarchical: the input layer is connected to the first hidden layer, that is connected to the second and so on until the output layer [3]. The neurons in the layer n are just locally connected to the layer n-1, the fully-connections are only in the last layers. The links between layers are managed by the 2D masks that act as a filter doing a 2D convolution. Actually, a layer is not just a 2D population of cells but is a 3D population: in the slices, called feature maps, is preserved the spatial organization of the image, while each feature map extracts a different feature from the input data [3]. Every feature map is calculated with the same mask, so the weights that should be calculated are less than the total number of connections [3]. The filter can be moved with different steps, called strides: the dimension of the slices of the next layer depending on these steps and, for reshaping them, is used the zero-padding (add zeros) on the edge of the feature map [3]. Between two convolutional layer, there is often a pooling layer. This layer reduces the dimension of the slices but not the depth, that means that

it acts only on the feature map, not in the whole volume [3]. This layer has not weights to set/train, and the most popular operations that it manages are: average and maximum. The learning of the CNNs is done through backpropagation, this method binds the use of rectified linear unit (ReLU) activation function for the neurons [3]. This function is 0 for negative values, while it is the bisector of the first quadrant of the Cartesian plane for positive values (see fig. 2.26.

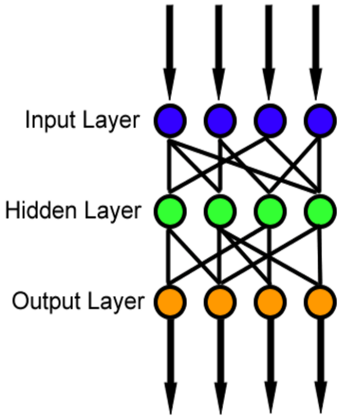

Figure 2.17: Example of feedforward network. As we can see, all connections link layer n with layer n+1 (from [21])

2.5 Learning

The learning is the process that allows the neural networks to learn for increasing the performances. In the biologic NN, this method is regulated by the synapse plasticity and it means that the synapses can modify (weights or delays), created or destroyed [39]. The changes due to the plasticity have a different duration:

- from 10 to 100 ms are called short term potentiation (STP) or depression (STD) [39];
- days, hours or more are called long term potentiation (LTP) or depression (LTD) [39].

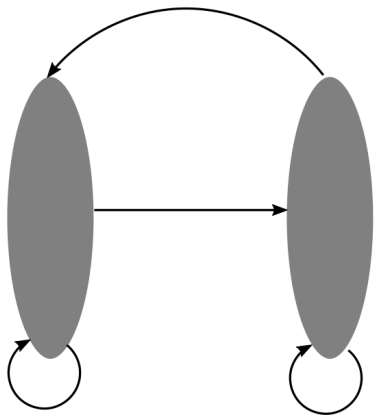

Figure 2.18: Example of RANN. The connections are not in just one direction, but they can be in all direction, in this image we can see cyclic connections.

2.5.1 Hebbian rule

Donald Hebb in 1949 was the first to wonder how and when the synapses update and keep their weight [37], so it theorized: if a pre-synaptic (that acts as input) neuron causes a post-synaptic neuron (it receives the input stimulus) to fire, the junction between them (called synapse) is enhanced and in mathematical terms:

$$\Delta w_{ij} \propto v_i v_j \tag{2.13}$$

where Δw is the weight change in the synapse between the pre-synaptic neuron (i) and the post-synaptic neuron (j) and v_i and v_j are the activities of the two neurons. In this equation, the synaptic efficacy changes only if both of neurons fire together. In the first theorized model, the junction between the neurons could just have a potentiation, not a depression, only later the depression was discovered. There are many rules that derive from the postulate of Hebb (see tab 2.1), but all of these ones have three common points [38]:

1. Time-dependence: the synaptic weight change depends on the exact fire time of pre- and post-synaptic neuron;

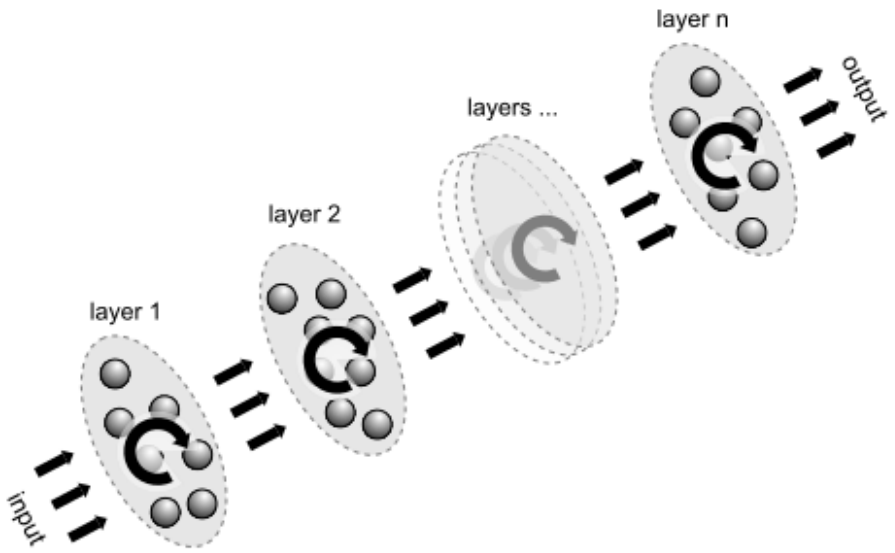

Figure 2.19: Example of synfire chain. Recurrent neural networks are linked together with a feedforward connections (from [38]).

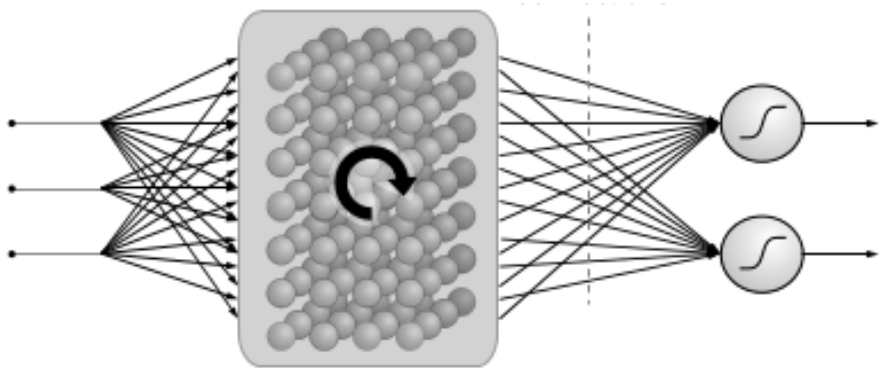

Figure 2.20: Example of reservoir computing. There is a huge recurrent neural network linked to readouts neurons. The plasticity is applied in the synapses between reservoir and output layer (from [38]).

2. Locality: the synaptic efficacy variation derives from some local variables (like pre- and post-synaptic activity and synaptic weight);

3. Interactivity: the magnitude of the change depends on the activity of the two cells.

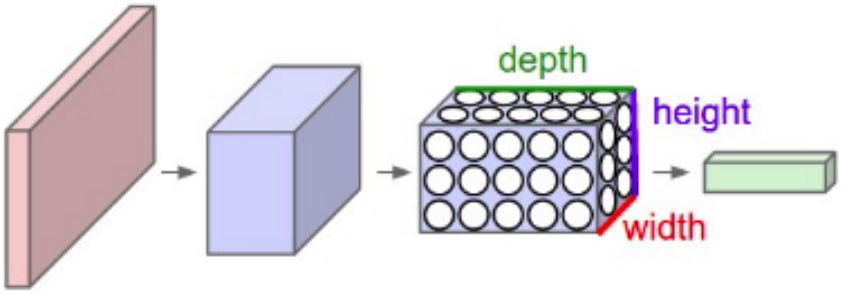

Figure 2.21: This is an example of CNN architecture, the layers can have a different depth. In the figure, the red one is the input layer and it has a number of neurons equal to the input image pixels (from [3]).

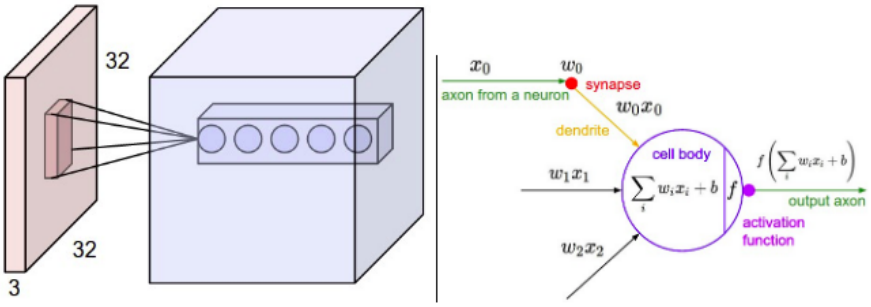

Figure 2.22: On the left is shown a CNN architecture: 3 depth input layer following by a 5 depth layer. On the right, the model neuron: the only one change between generations is the activation function (from [3]).

2.5.2 Spike-timing dependent plasticity (STDP)

In the third generation of ANNs the time is explicit, then the Hebbian rule was changed, this new rule is called spike-timing dependent plasticity (STDP) or temporal Hebbian rule [37] and it is defined as:

$$\frac{dw_{ij}(t)}{dt} = a_0 + S_i(t)\left[a_1^{pre} + \int_0^\infty a_2^{pre,post}(s)S_i(t-s)ds\right] + \tag{2.14}$$

$$+ S_j(t)\left[a_1^{post} + \int_0^\infty a_2^{post,pre}(s)S_j(t-s)ds\right] \tag{2.15}$$

where $\frac{dw_{ij}(t)}{dt}$ is the weight update value, $S_i(t)$ and $S_j(t)$ are the pre- and post-synaptic spike trains respectively, defined as sums of Dirac deltas. a_0 is a term of linear decay, while the two integral terms are a low pass filter. The synaptic efficacy changes also if just one of the two neurons spikes because of the non-Hebbian terms

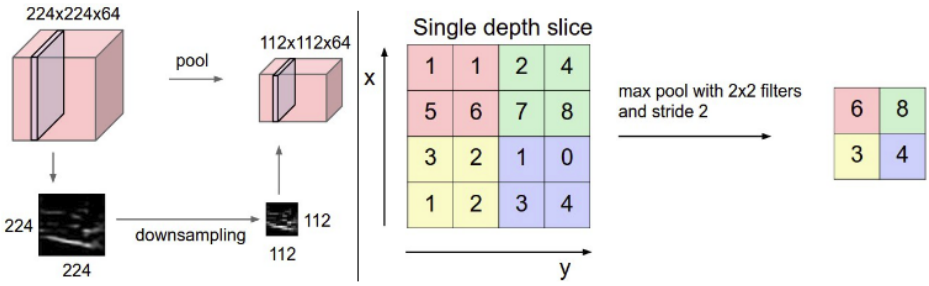

Figure 2.23: In the figure is shown how maximum pooling layer act: it reduces the dimension of the feature maps, but not the number of them (from [3]).

Rule	**Equation**
Standard Hebbian rule	$\Delta w_{ij} \propto v_i v_j$
Hebbian rule with decay	$\Delta w_{ij} \propto v_i v_j - c_0$
Hebb with pre-synaptic gating	$\Delta w_{ij} \propto (v_i - v_\theta) v_j$
Hebb with post-synaptic gating	$\Delta w_{ij} \propto (v_j - v_\theta) v_i$
Covariance rule	$\Delta w_{ij} \propto (v_i - \bar{v_i})(v_j - \bar{v_j})$

Table 2.1: Some Hebbian rules. Δw_{ij} is the change of the weight, v_i and v_j are the pre- and post-synaptic activities respectively, c_0 is the decay constant, v_θ is the gating threshold and $\bar{v_i}$ and $\bar{v_j}$ are the mean pre- and post-synaptic activities [38].

a_1. The integral terms give the mutual contribution beacuse $s = t_j^f - t_i^f$. a_2 are called kernels, and they define the learning window. There are many learning windows, but the most important are shown in the fig. 2.24 [38]. The main differences between them are the symmetry with respect to the origin of the axis and respect to y-axis and the continuity in the neighbourhood of $t = 0$ [37]. The most important window is the exponent one:

$$W(s)^{STDP} = \begin{cases} a_2^{post,pre}(s) = A_+ e^{-\frac{s}{\tau_+}} & \text{if } s \geq 0, \\ a_2^{pre,post}(-s) = -A_- e^{\frac{s}{\tau_-}} & \text{if } s < 0 \end{cases} \tag{2.16}$$

$$W(s)^{aSTDP} = \begin{cases} a_2^{post,pre}(s) = -A_+ e^{-\frac{s}{\tau_+}} & \text{if } s \geq 0, \\ a_2^{pre,post}(-s) = A_- e^{\frac{s}{\tau_-}} & \text{if } s < 0 \end{cases} \tag{2.17}$$

The eq. 2.16 is the learning window for STDP (normally $a_0 < 0$, $a_1^{pre} > 0$ and $a_1^{post} < 0$), while eq. 2.17 is the one for anti-STDP (normally $a_0 < 0$, $a_1^{pre} < 0$ and $a_1^{post} > 0$). The terms $A_+, A_- > 0$ and they represent the magnitude, while $\tau_+, \tau_- > 0$ and are the time constants of the process [38]. Normally the synaptic efficacy is updated by an additive rule ($w = w + \frac{dw}{dt}$) or by a multiplicative one ($w = w(1 + \frac{dw}{dt})$)[37].

It has been shown that the STDP process has two important proprieties:

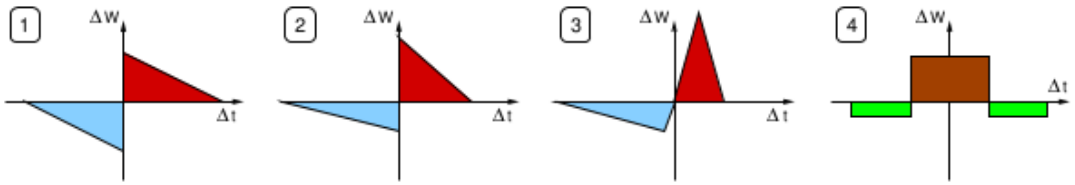

Figure 2.24: Some examples of learning window (from [37]).

1. the first spike of the train is the most important one (it holds more information);

2. an asymmetric window strengthens the synapses in which the pre-synaptic neuron has a precise spike time, while depress the connections where the spike time has a jitter [38].

This learning method still has a problem: there is a maximum and a minimum weight.

In the SNN the STDP process is the base of almost all the unsupervised learning methods [37]. Unsupervised methods used an unlabled input, and the network have to classify them on the base of some features.

2.5.3 Supervised learning

The aim of supervised learning is: given an input spiking train $S_i(t)$ and a teaching signal $S_j(t)$, find the vector of weights that minimize the error between $S_i(t)$ and $S_j(t)$ [38]. For reaching this goal there are a lot of methods, some of them do not have biological meaning while some others have it.

Methods based on the gradient

For the methods based on the gradient, the use of the derivative is envisaged, but the output of spiking neurons is not continuous. SpikeProp is a method developed by Bohte and his team that bypass the problem of continuity. In this technique, every neuron can fire just ones for every simulation cycle and after it, the membrane potential is ignored. With this trick, the membrane output becomes continuous [38]. In the beginning, this method was used only for the synaptic efficacy, but later was implemented for the synaptic delays, for the time constants and for the spiking thresholds too. All these new features make the SpikeProp converges faster and they allow to have smaller neural networks for the same task [38]. This technique was tested for a lot of standard datasets (Iris dataset, Wisconsin breast-cancer dataset and the Statlog Landsat dataset) and for all of them it had the same accuracy of sigmoidal neural networks (networks of the second generation in which the neurons have a sigmoidal activation function). Moreover, this method was used

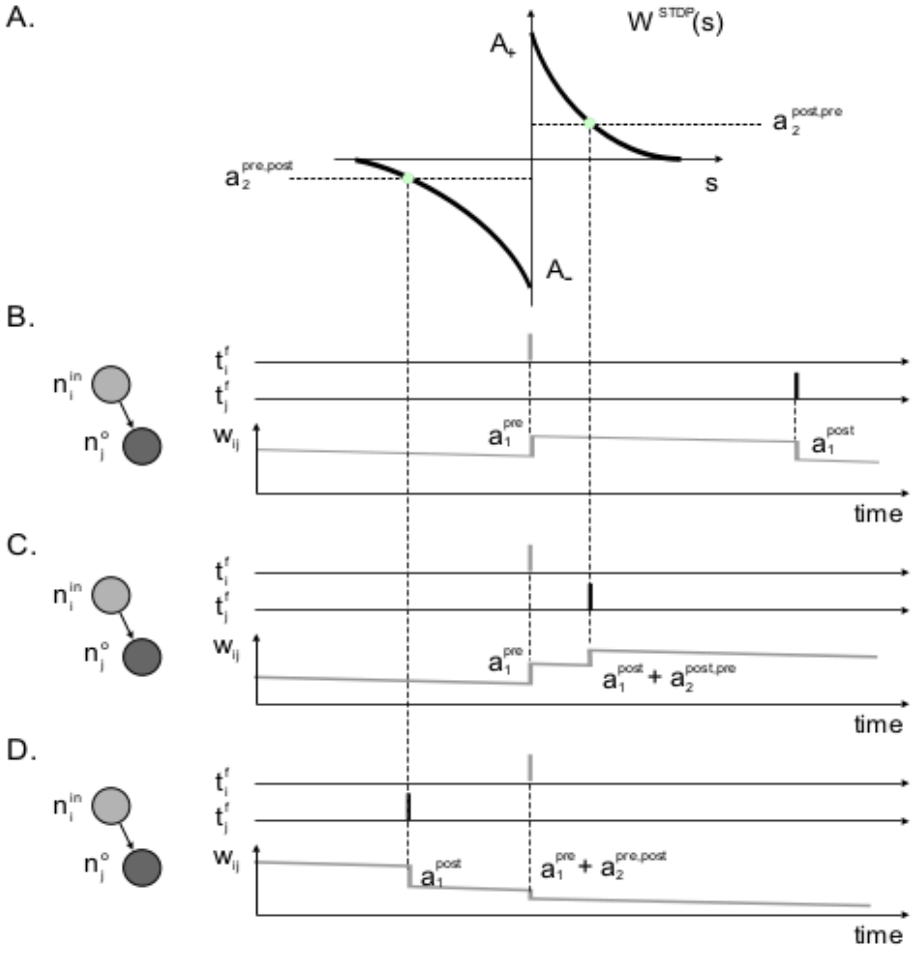

Figure 2.25: A is an exponential learning window: in the x-axis, there is the delay between the spike of pre-synaptic neuron and the post-synaptic neuron while in the y-axis, there is the magnitude of the weight update. B is an example when the post-synaptic neuron fires out of the learning window, in this case, the synaptic weight has no change. C represents the case in which the post-synaptic neuron spikes after pre-synaptic neuron, but in the learning window. D shows how junction weight change if the post-synaptic cell fires before (but in the learning window) the pre-synaptic cell. In all cases, the weight is decreasing, this behaviour is due to the a_0 coefficient, which is a linear decay (from [38])

for real-world datasets, and for all of them it converged, while it didn't always happen for the ANNs of the second generation [38].

SpikeProp has a few downsides too:

1. the information can be only in the first spike, while some message can only be decoded in the spikes trains [38];

2. due to the first point, the only method of coding is time to first spike (see 2.2.2);

3. this technique is only for SRM neurons (see 2.1.3) [38];

4. the synaptic weight is updated only if the post-synaptic neuron spikes, so if this cell never fires the junction weight (also called efficacy) never change [38];

5. due to the fourth point, the accuracy of the networks train with this algorithm has a strong dependency to initial weights [38].

This technique is the father of some other methods, like QuickProp and Rprop, which are computational less heavy, and MultiSpikeProp for multilayer SNN [46].

With SpikeProp, it has been bypassed the differential problem just cutting the signal. This cannot happen if we are training a recurrent spiking neural network (RSNN) because we need to propagate the error in the previous epoches. In the recurrent ANNs of the second generation, it is used backpropagation through time (BPTT) that uses derivates. In the case of RSNN, [15] proposed the use of a pseudo-derivate:

$$\frac{dz_j(t)}{dv_j(t)} := \gamma max\{0{,}1 - |v_j(t)|\} \tag{2.18}$$

$$v_j(t) = \frac{V_j(t) - B_j(t)}{B_j(t)} \tag{2.19}$$

where $z_j(t)$ is the spike train (a sum of Dirac delta function), $B_j(t)$ is the spike threshold and $V_j(t)$ is the membrane potential, so $v_j(t)$ is called normalized membrane potential. This pseudo-derivate is not stable if RSNN is unrolled for a lot of epoches, so $\gamma < 1$ has been introduced usually it is set to 0.3 [15].

For bypassing the problem of non-differential activation function of the SNN, [32] proposed another method for training the SNN with backpropagation: train a second generation ANN and then change the neurons with spiking neurons. This approach can be done with all types of neurons, here we are going to see it with LIF model that is defined by:

$$\tau_{RC}\frac{dv(t)}{dt} = -v(t) + J(t) \tag{2.20}$$

where τ_{RC} is the membrane time constant, $J(t)$ is the input current and $v(t)$ is the membrane potential. If we fix the input current to a value (j), we can solve the equation for the time:

$$r(j) = \begin{cases} [\tau_{ref} - \tau_{RC}\log(1 - \frac{V_{th}}{j})]^{-1} & \text{if } j > V_{th}, \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

where $r(j)$ is the firing rate of the LIF neuron, τ_{ref} is the refractory period and V_{th} is the spike threshold that we set to 1. With this firing rate, we cannot calculate the derivate due to the presence of a non-differential point for $j = 1$ (see fig. 2.26). So the challenge becomes to smooth that point and bound the value of it. To perform it, [32] introduces a new equation:

$$r(j) = \left[\tau_{ref} + \tau_{RC} \log(1 + \frac{V_{th}}{\rho(j - V_{th})})\right]^{-1} \tag{2.22}$$

$$\rho(x) = \gamma \log(1 + e^{\frac{x}{\gamma}}) \tag{2.23}$$

where gamma is a parameter that controls the smoothing of the firing rate function. This equation creates the soft LIF neuron model.

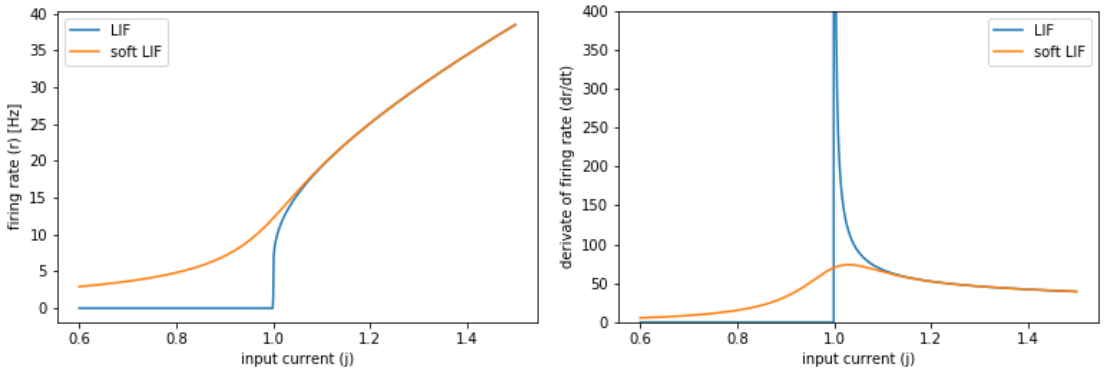

Figure 2.26: In the left figure, we can see the firing rate in function of the input current, while in the right one, we can see the derivate of the firing rate. In j=1 there is an unbounded and non-differential point for the LIF neuron, while there is a differential and bounded point for soft LIF neuron. The plot is created with $V_{th} = 1$, $\tau_{ref} = 0.004$, $\tau_{rc} = 0.02$ and $\gamma = 0.03$ (modify from [32]).

Until here, we saw how to model the neurons, but the messages that the they send to each other pass through the synapses and these junctions add noise to the spikes trains. The noise is modelling with a Gaussian distribution with 0-mean and standard deviation sets to 10 [32]:

$$r(j) = \left[\tau_{ref} + \tau_{RC} \log(1 + \frac{V_{th}}{\rho(j - V_{th})})\right]^{-1} + \eta(j) \tag{2.24}$$

$$\eta(j) = \begin{cases} G(0, \sigma) & \text{if } j > V_{th}, \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

The conversion in SNN is performed after training. In this process, the weights, the delays, the connections and all other hyperparameters are fixed to the values reach during the train, the only things that will be changed are the neurons (from soft LIF to LIF), the noise will be removed and the α-filter[3] will be added for modelling the synapses.

Evolutionary strategy

The evolutionary algorithms are good strategies for optimization problems. One problem with this method is: which kind of mutation strategies is good for us? Gaussian distribution finds the best solution in local, while the Cauchy distribution explores a bigger solution space due to a bigger number of mutation [38]. This algorithm can be used both for the synaptic-weights and for the synaptic-delays. This type of learning was tested with Iris dataset and with XOR problem and in both cases, it gave results similar to SpikeProp [38].

2.6 Networks

In this section, we are going to analyze four different spiking neural networks.

Network of Schmuker

This network is created by Schmuker and his team [45] and it is inspired by the blueprint of the insect olfactory system. For encoding the input in the spike domain, it has been used the neural gas algorithm (see 2.2.1).

This network was created for running on neuromorphic hardware. It is composed of: virtual receptors (VRs), a number of subunits called glomerulus same as VRs and an association layer. The aim of VRs is to convert the images into spike trains. The glomeruli are made of a layer of receptor neurons (RNs), another layer of projection neurons (PNs) and a layer of local inhibitory neurons (LNs). The RNs are just an input for the network: they are used for transducing the exact spike time, created by VRs, in spikes. The PNs are linked only to LNs of the same glomerulus, while LNs inhibit the PNs of all glomeruli (but no self inhibition is performed), this action is called moderate lateral inhibition and it is so important because it decorrelates the information, without changing them [45]. It is important that is moderate otherwise the information will be altered. The decision layer is made of a number of populations equal to the number of classes, every population in this layer is composed of association neurons (ANs) and inhibitory neurons. All PNs are projected to all ANs while the ANs of each population are linked to their

[3]This filter is biological reliable [32] and it is defined as: $\alpha(t) = \frac{t}{\tau_s} e^{\frac{-t}{\tau_s}}$

inhibitory neurons and these ones are connected to all other populations of ANs. This inhibitory, that is strong, performs a soft winner take all (sWTA). Learning is performed by a gradient descendent method and it is applied just in the projections between PNs and ANs. If the output is right, all the connections between the winner population and PNs that fire more than a fixed threshold are enhanced of a certain value while, if the output is wrong, the synapses that help to fire are depressed of a fixed value.

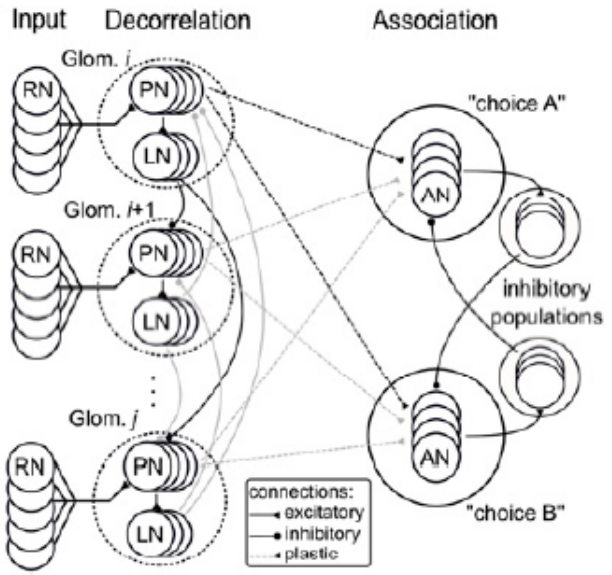

Figure 2.27: In the figure is represented the network of Schmuker (from [45]).

Network of Diamond

This network is taken by [26] and it was created for SpiNNaker.It is composed of a VRs layer, which coding the input signal with neural gas technique (see 2.2.1), a layer of spike source neurons, which coding from time to spikes and a layer of RNs that sends the spikes to a PNs layer. In this layer, there is the inhibition between different populations. The last layer is an association layer (the neurons inside it are called ANs) with a strong lateral inhibition that creates a winner take all circuit (WTA). The learning is made through STDP (see 2.5.2) in the projections between PNs and ANs. To the RNs is attached a population of Poisson neurons, that makes the firing rate more biological and stabilizes the network. The STDP in this network is used for supervised learning, so we have to use a teaching signal that is provided by another population of spike source neurons linked to ANs.

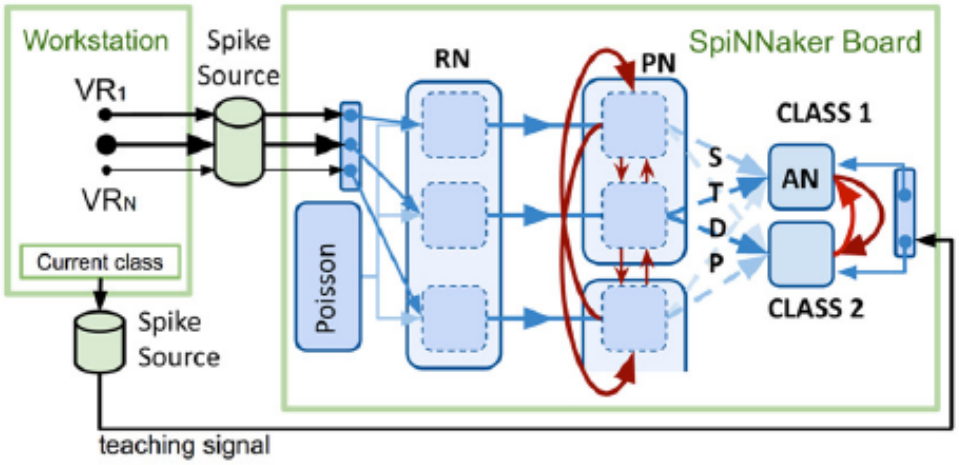

Figure 2.28: In the figure is represented the network of Diamond (from [26]).

Network of Diehl

This network is implemented by Peter U. Diehl and Matthew Cook [27] with Python and Brian simulator. This network is an unsupervised network that uses the STDP rule for learning and it is composed of three layers: the input layer that consists of 28x28 neurons (equal to the dimension of the input image), the excitatory layer and an inhibitory layer, both of them have, in the training phase, LIF neurons with adaptative threshold and in the testing phase the normal LIF model. The adaptative neurons were used for preventing the "epileptic neurons" as this adaptative spike threshold increases the value of the spike threshold as the number of spikes increases. The first layer is connected to the second one in an all-to-all fashion (in these links there is the synapse plasticity), the second layer is connected in one-to-one way to the third one, while the last one is connected in an all-to-all fashion to the second one, but without self-inhibition (see fig 2.29). The stimulus is presented for 350 ms and after that, the network has a sleeping time of 150 ms, this time is used for let all the parameters return to the initial value [27]. The coding of the inputs is performed by Poisson distribution neurons (see sec 2.2.1). The firing rate of the input neurons is set to the pixel value quarter; the maximum value of the pixels can be 255, so the maximum frequency can be 63.75 Hz. If the second layer fires less then 5 times, the image is shown to the network again, but the frequency of the input neurons is increased by 32 Hz. After the training, the LIF adaptative neurons were switched to a normal LIF, with the spike threshold fix to the value reached during the previous phase and the STDP synapses were changed with the static ones, with the value calculated during the training phase. The labels are used only now, all the dataset is shown again, and the excitatory neurons are labelled with the label that allows them to fire more. Different tests are made with this network and the only hyperparameter that change is the number of neurons

in the second and third layers: 100, 400, 1600, 6400. The times that the dataset was shown to the network changes with the number of neurons: 1, 3, 7, 15. The accuracies that the network reaches are 82.9%, 87%, 91.9%, 95%.

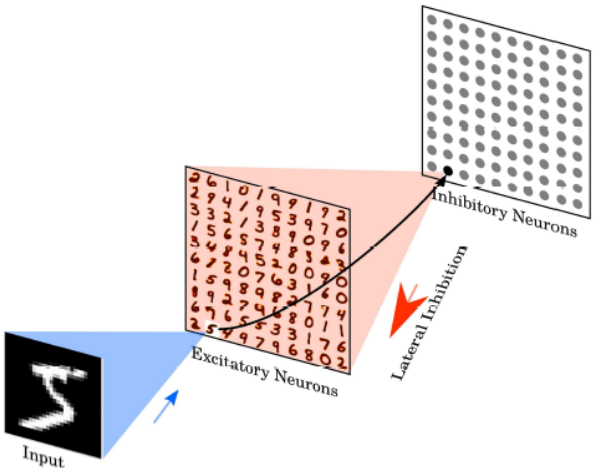

Figure 2.29: In the figure is represented the network of Diehl(from [27]).

SCNN

This network is a classic CNN (see 2.4) and it is composed of an input layer that has the number of neurons equal to the number of input image pixels. The next layer is a convolutional layer composed of 32 filters, followed by another convulational layer with 64 filters. The next three layers are, in order, an average pooling layer, a convulational layer with 128 filters and another average pooling layer. The output layer is a dense layer with as many neurons as there are classes. All filters have a dimension of 3x3, and they are 2 strides, it means that every convolutional layer decreases the dimension of the problem by 2. The polling layers halved the size of the problem.

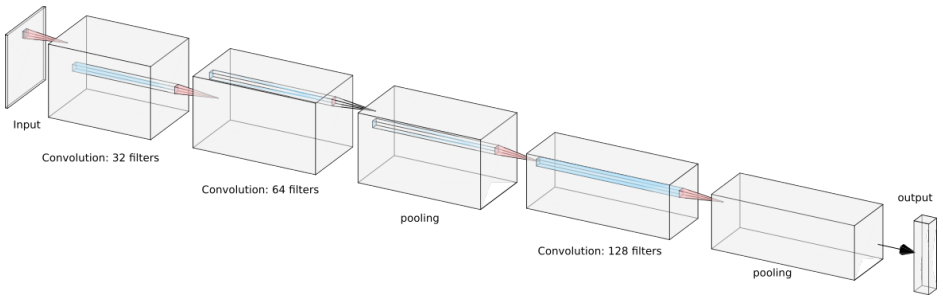

Figure 2.30: Sepresentation of SCNN.

Chapter 3

Methods and results

3.1 Input encoding

The first essential step a fruitful use of a spiking neural network is the input encoding, which will affect all the subsequent phases. For this reason the first test that I did, was for encoding the input stimulus in such a way that the networks were able to process the dataset. Going into further details, as input encoding techniques, I have implemented neural gas and Poisson methods (both described in section 2.2.1) with Python 3.6. In the neural gas encoding, the Modular toolkit for Data Processing (MDP) [49] toolkit helped me to position the VRs in the space of images. From these tests, I expected to see a different firing rate due to the different input of the VRs (see fig. 3.3 and 3.2) and, in the case of Poisson distribution, I was expected that I can recreate the input stimulus starting from the firing rate (see fig. 3.1).

3.2 Datatset

In my tests, I used different datasets. My two main goals were to classify the images of colorectal cancer (CRC) dataset and compare the performances between the second generation ANNs and the third generation ones. The CRC dataset [40] is made up of images depicting three categories of human colon tissue, namely healthy tissues (H), tissues with tubulovillous adenoma (AD) and tissues with adenocarcinoma (AC). Such dataset is extremly challenging for classification due to the extreme intra-class variability attributable on one hand to the intrinsic complexity in the biological context, on the other hand to imaging techniques, affecting colors and quality of the images. The images size is 218x218 pixels; the training set is made up of 10,500 samples while the test set is composed of 3,000 images. For reducing the dimensionality of the problem, I downsampled from 218x218 to 64x64 and then I converted them into grayscale. Due to complexity, this dataset was used only with neural networks that have got good performances with MNIST

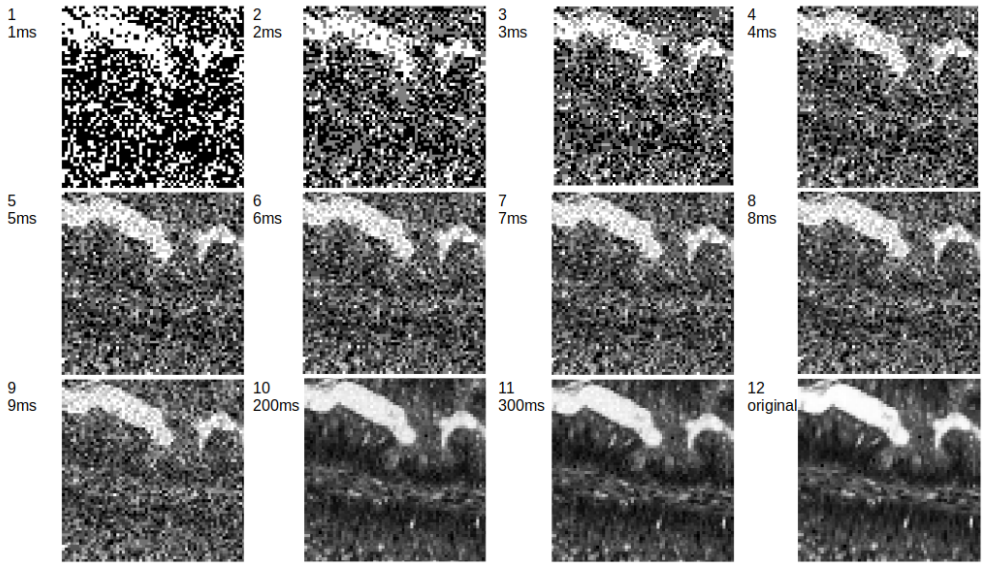

Figure 3.1: In this figure, we can see the evolution of one image with Poisson distribution: the first nine images are the mean of the spikes of the first nine milliseconds of image exposition, while the tenth and the eleventh are the mean of the spike trains at $200ms$ and $300ms$ respectively. The last one is the original image.

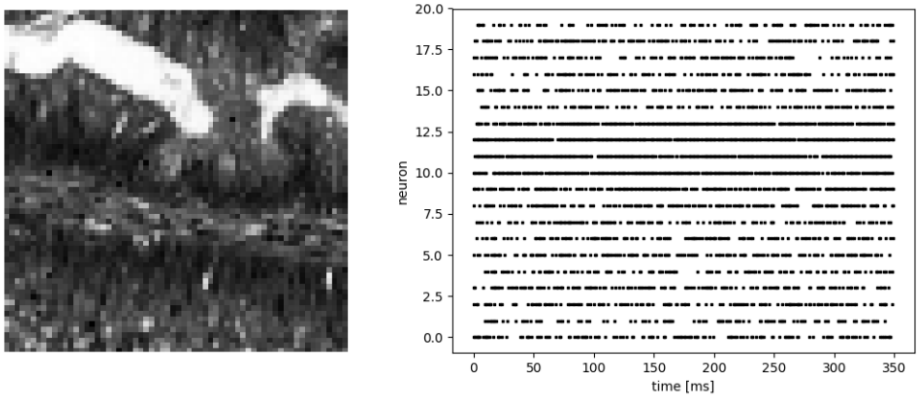

Figure 3.2: On the left, there is the original image, while on the right there are the spike trains of the first 20 neurons generated with Poisson distribution with time of exposition of $350ms$.

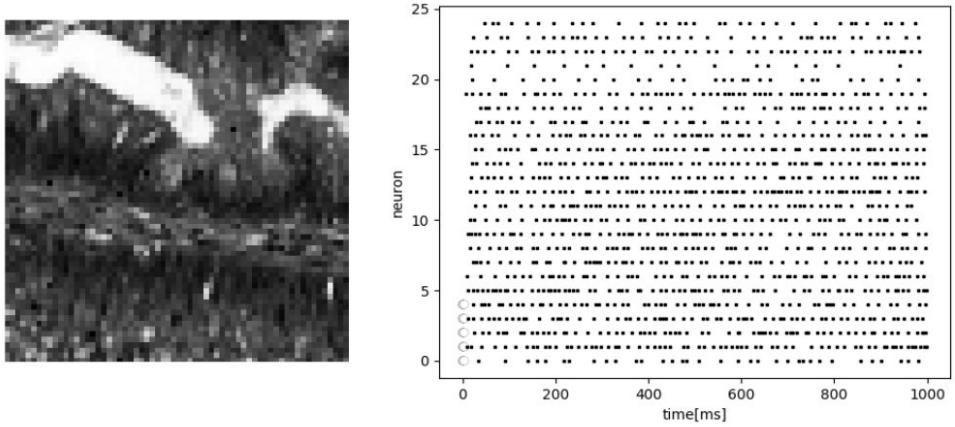

Figure 3.3: On the left, there is the original image, while on the right there are train spikes of 25 VRs generated from a gamma distribution with an exposition time of 1000 ms.

dataset [35]. Such dataset is composed of the ten hand-writing digits in grayscale (i.e. numbers from 0 to 9). Images are 28x28 pixels, that means they are composed of 764 pixels. The training set is composed of 60,000 samples, while the test set is composed of 10,000 images. These data were been created enlisting about 250 writers, taking care that the people that wrote for the training set, didn't write for test set and vice-versa.

For completeness, I want briefly introduce Fisher's iris dataset because Schmuker and his team used it too for testing the accuracy of their network [45]. This dataset is composed of 150 samples divided into three classes: Iris setosa, Iris virginica and Iris versicolor. In the space of features, the first type of flower is well separated by the others, while last two types are overlapped [22].

3.3 Classifiers

Network of Schmuker

I implemented this network (see section 2.6) with python 3.6 and PyNN 0.9.3, using as simulator nest 2.16. First of all, I tuned the hyperparameters (just the synapse weights and the parameters of the learning) of the network for gain the same accuracy that Schmuker and his team reached. This result was obtained using just 200 samples in training and 200 in test. After this operation I used the same hyperparamters, but I changed the classes, this time I worked with 4 and 9 because

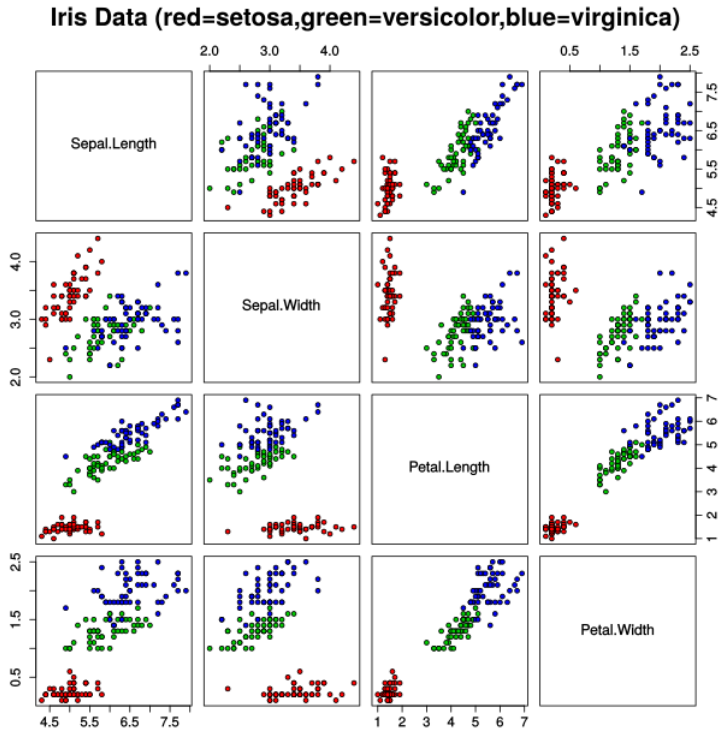

Figure 3.4: In the figure is represented the scatter plot of Fisher iris dataset (from [22])

Figure 3.5: From left to right we have iris setosa, iris versicolor and iris virginica (from [22]).

I wanted to check if the network can classify two classes that are more overlapped. The result was that the accuracy has dropped a lot. The next step was to run the network with all clasess and 30,000 samples in training and 5,000 in test. The result, as expected, was that the network was not able to classify the dataset, so it

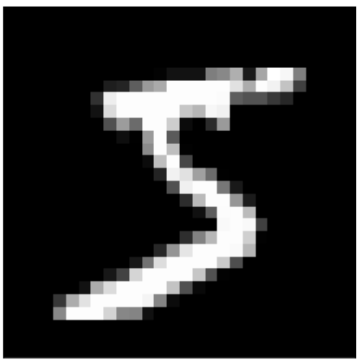

Figure 3.6: It is represented an example of hand-writing digit from MNIST dataset.

Figure 3.7: CRC dataset. Images are RGB with 218x218 pixels.

was useless to continue the test on the CRC dataset.

Network of Diamond

First of all, I implemented this network for SpiNNaker, using python 2.7 and PyNN 0.7.5. After the same results of Diamond, with his dataset (train and test were the same and the spikes trains were availble in [26]), I re-implemented the network for nest 2.16 using python 3.6 and PyNN 0.9.3. This step was necessary because the SpiNNaker board that we have, can simulate a limited number of synapses and neurons. Before starting the simulation with the entire MNIST dataset, I have pursued some preliminary experiments; in particular the network was trained:

i) with the same boudary conditions described in the correspondig paper (see fig. 3.10 and 3.11)[26];

ii) running the network without Poisson neurons (see fig. 3.12);

iii) running the network without Poisson neurons and RNs (see fig. 3.13).

The rationale behind these experiments was to test if such a deep network, as described in [26], was indispensable to a sufficient high accuracy, or if same results

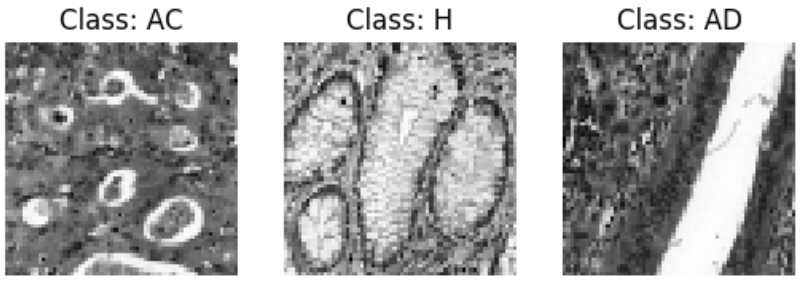

Figure 3.8: CRC dataset. Images are grayscale with 64x64 pixels.

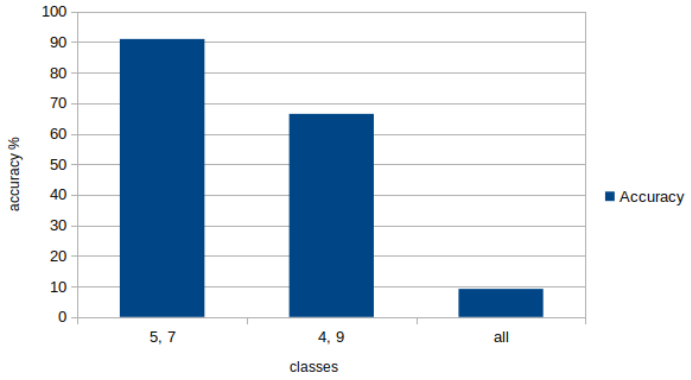

Figure 3.9: In this figure is shown the accuracy of the Schmuker's network in three trials: with classes 5 and 7, with classes 4 and 9 and with all classes.

may be reached by shallower (or simpilier) networks just with an investigation among network hyperparameters, then I expected that the network with these changes continues to well classify the data of Diamond, with just a change in the hyperparameters: the maximum value that the connections between PNs and ANs can reach.

Between test i) and ii), I just removed the Poisson neurons and I increased the maximum value for the synapse (from 0.3 to 0.9), while between try ii) and iii) I only removed the RNs: no change was made to the hyperparameters. We can see from figures 3.11, 3.10, 3.12 and 3.13 that these three networks are equivalent.

After these few tests, I came back to Diamond's network and, without changing the hyperparameters of the network, I tuned the number of VRs and the number of images in the train and in the test, the results are shown in fig. 3.14: the network cannot classify. As it is possible to see from figure 3.15, the main reason for the bad accuracy is that the network, during the simulation, diverges.

Starting from the hypotesis that the noise is one of the most probable causes of divergence, I removed the Poisson layer and I made an exploration table in which only the number of RNs, PNs and ANs change (they can assume these values: 15,

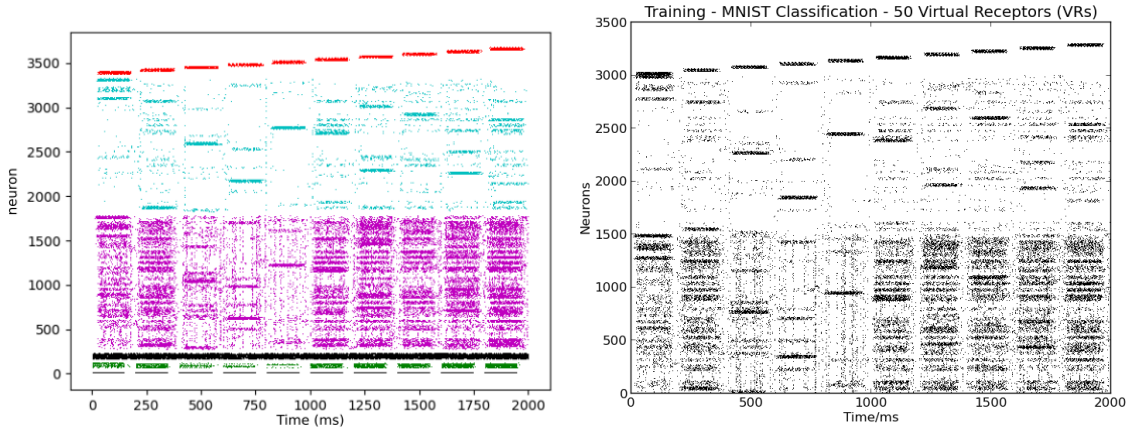

Figure 3.10: The figure on the left is what I obtained from the first try during the training phase, while on the right there is the Diamond's one, as we can see the patterns are similar.

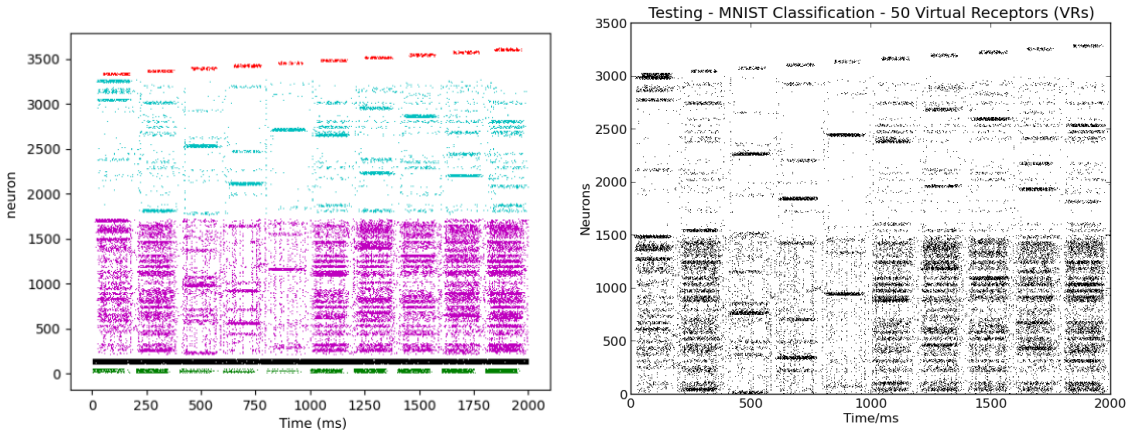

Figure 3.11: The figure on the left is what I obtained from the first try during the test phase, while on the right there is the Diamond's one, as we can see the patterns are similar.

30, 60). As dataset I used the entire MNIST (60,000 samples for training and 10,000 for testing). The results are shown in figure 3.16. As it can easily gathered from this figure, the network has a small increase in accuracy, but it is still not able to classify.

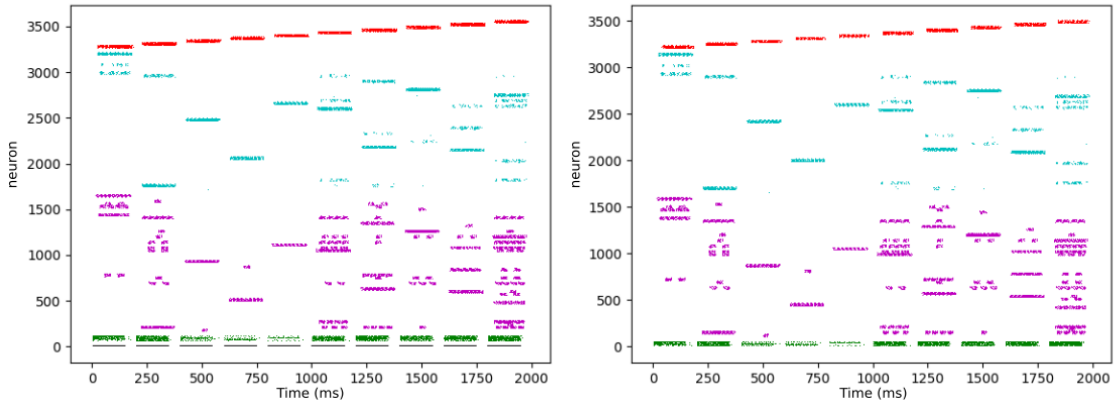

Figure 3.12: The figures represent the second test: Diamond's network without the Poisson neurons. On the left, there is what I obtained from the training phase, while on the right there is the test. As we can see, the ANs spike in a similar way of the previous trial.

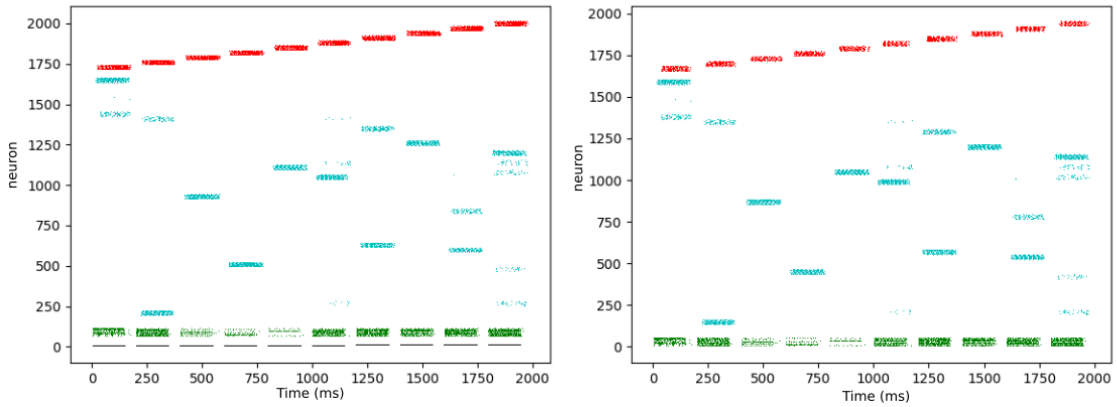

Figure 3.13: The figures represent the third test: Diamond's network without the Poisson neurons and the RNs. On the left, there is what I obtained from the training phase, while on the right there is the test. As we can see, the ANs spike in a similar way of the first trial.

SCNN

For this network, I used NengoDL 2.1.1 and python 3.6. I run it with entire MNIST, obtaining an accuracy of 98%. Before running such model on the whole CRC dataset, a hyperparameters exploration was made with the aim of finding best working condition for the model. The tuning of the hyperparameters has to be done just with the training set, while the test set is used only for checking the

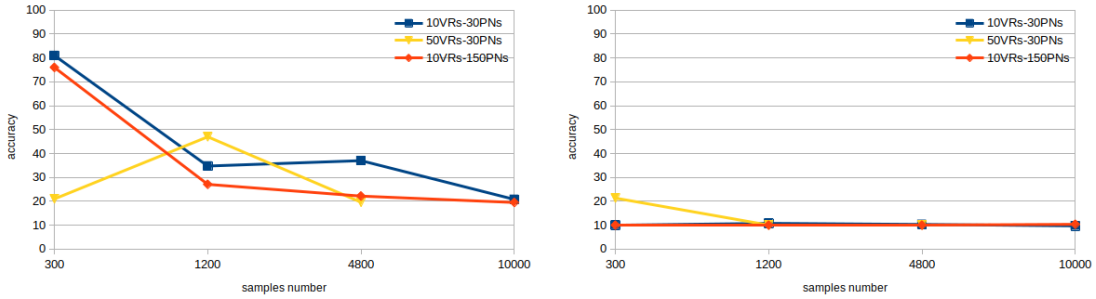

Figure 3.14: In these two figures is shown the trend of accuracies for the training (right) and the testing (left).

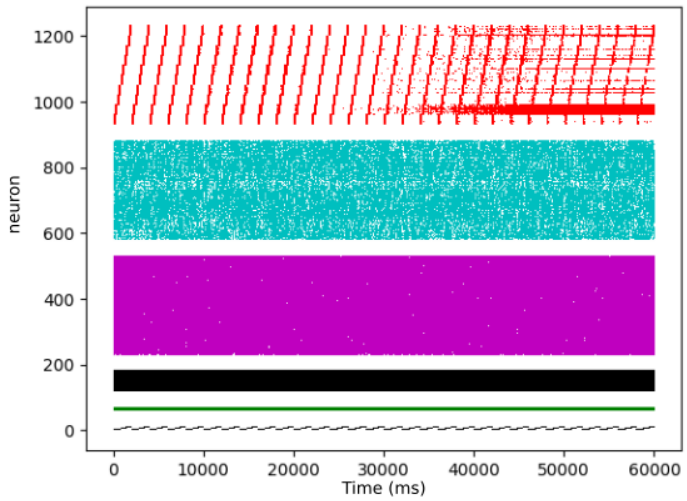

Figure 3.15: In this figure is shown the behaviour of the network: it diverges after few epochs.

performance of a network after the choice of parameters. To perform this task, I split the training set into two subsets called training set and validation set. The training set is 90% of the original one, and it is used for training the network, while the validation set is the 10% of the original one, and it is used for checking the accuracy of the SCNN after every hyperparameter change. The parameters that I changed are the learning rate and the optimizer (see fig. 3.17). As we can see from figure 3.17 the best accuracy was obtained from the couple: leanrning rate 10^{-4} and RMSProp. After setting these two parameters, I trained the network again the network with the whole training set and then I run the SCNN the test set, obtaining an accuracies of 95%.

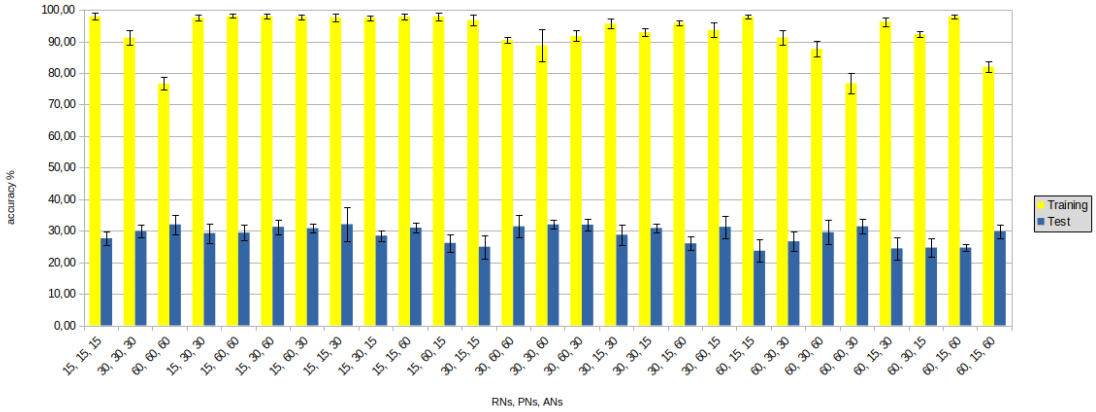

Figure 3.16: In this graph is shown the accuracies of the 27 tries made changing the number of RNs, PNs, ANs. The x-axis represents, in order, the number of RNs, PNs and ANs.

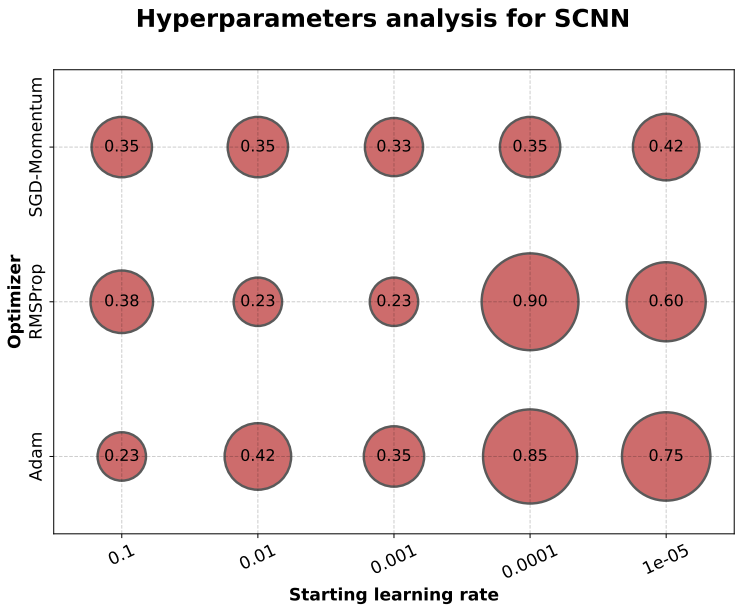

Figure 3.17: In this figure is shown the accuracies of each subset of hyperparameters.

In figure 3.20, it is shown two examples of misclassification. Seems reasonable to attribute such misclassification errors to the absence of the colonic glands, characterizing the normal, health coloinc tissue.

With CNN (see section 2.4), using the same dataset and the same architecture 2.6. The result of the hyperparameters tuning (as the SCNN, only the optimizer and the learning rate were changed) is shown in figure 3.21. I took the best couple of parameters and then the network has been retrained, and the confusion matrix

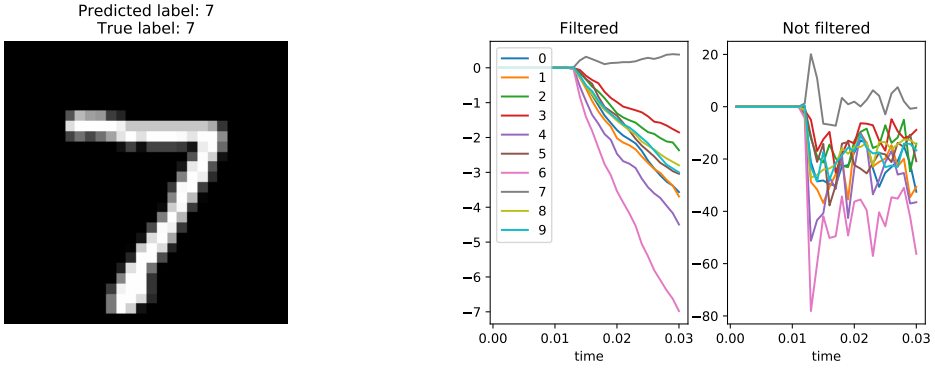

Figure 3.18: On the left is shown the input figure, on the right the activation of the output neurons (with and without α-filter).

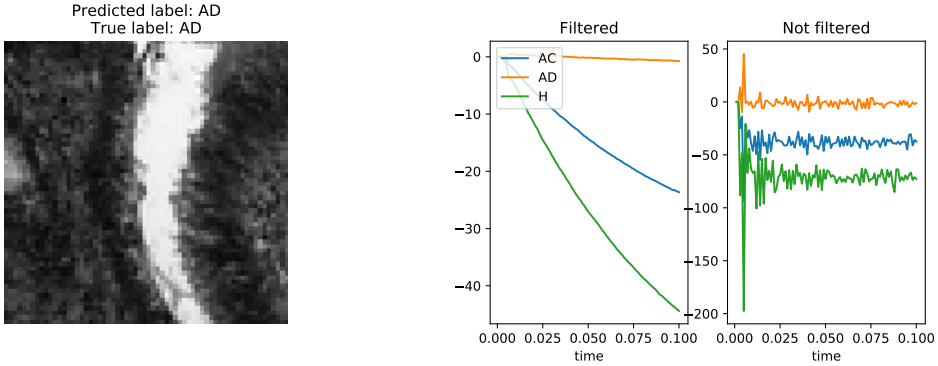

Figure 3.19: On the left is shown the input figure, on the right the activation of the output neurons (with and without α-filter).

in figure 3.22 has been calculated. As we can see, CNN has almost the same classification for all the classes, while SCNN classifies better the AD class and worse the H one.

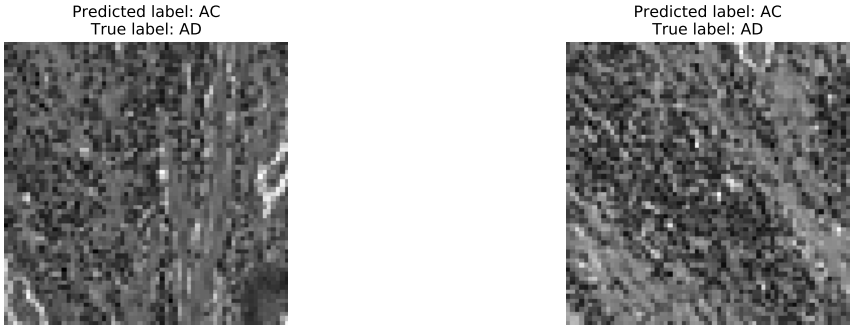

Figure 3.20: An example of misclassification. In this two images the misclassification is due to the fact that no glands are inside, but just noise.

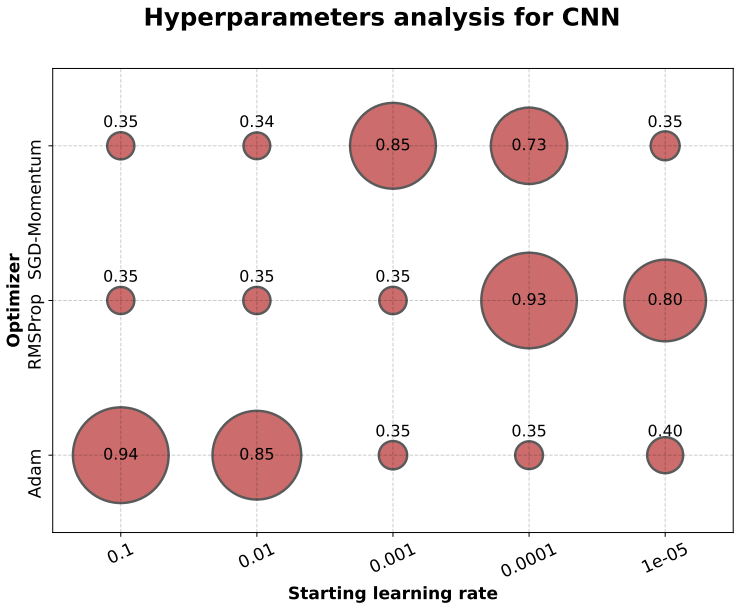

Figure 3.21: In this figure is shown the accuracies of each subset of hyperparameters.

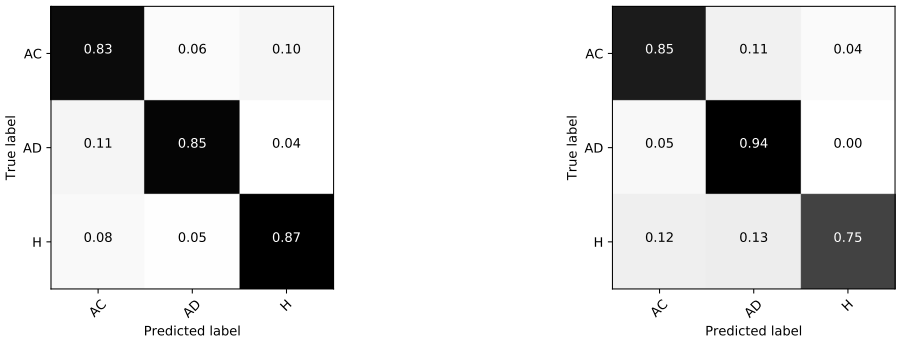

Figure 3.22: On the left is shown CNN confusion matrix, on the right the SCNN one.

Chapter 4

Discussion

The main aim of this master thesis was to fully investigate the state of the art regarding SNNs in the field of image processing. SNNs are promising under different points of view. First, they are highly biological plausible, this means that their investigation and improvement could produce interesting insights into the brain behaviour. Secondly, the so-called neuromorifc hardware, which is the natural substrate where to implement SNNs, can open the way to tremendously accelerated computation for both training and test, core issues when speaking about machine learning. The other face of the coin of such considerations, is the current paucity regarding frameworks and benchmarks to implement and test SNNs, with the outcome of discussing interesting theoretically correct results without any application to prove the effective use of these techniques. In the light of these speculation, I faced the study and implementation of three different state-of-the-art SNNs architecture, namely the network of Schmuker, the network of Diamond and a SCNN, obtaining the results discussed in the following. The network of Schmuker is biologically inspired and it has a complicated architecture. I reached the same performance as the authors with two-classes MNIST dataset (90% of accuracy with classes 5 and 7), but when I used two MNIST classes more overlapped (4 and 9) or the whole ten-classes dataset, the network cannot reach good accuracies as it was expected: in the first test I reached 65% of accuracy, while in the last test the accuracy fell to 10%. This network has different problems. First the learning algorithm: it adds or removes a fixed value to the synapses that help to fire. This approach is not the best, as it does not consider the value of the error. Second, the architecture. It is made up of many layers and a high number of neurons, which makes the model prone to overfit and in general difficult to tailor. The network of Diamond with its easier architecture and with a biologically inspired learning rule, is a simplification of the network of Schmuker. The implementation of this network for SpiNNaker was quite straightforward, while the implementation for nest was more troublesome, due to the different values that the STDP can have. Despite having achieved the initial results as described in the reference paper, which means

100% of accuracy with an extremely reduced training and test sets, I was not able to reach comparable results with the MNIST classification task. In their work, the authors reached 60% of accuracy with the whole MNIST dataset while my network diverged. Probably my low accuracy is partially due to the not-complete inspection of hyperparameters that I was able to perform, due to time and computational constraints. Anyway, the intrinsic complexity in tailoring this net must be kept into account, as it will limit any future application or case study. The entire network is again very complex: the Poisson layer and the RN layer may be removed, and, with this condition, the network was more stable: it did not diverge, and it reached a better accuracy: 30% (just without Poisson layer). This finding is anyway not satisfying yet. An approach to reach better results can be adding more synapses between PNs and ANs with different delays. Since the synapses in the learning layer hold the information of the network with this operation, we are increasing the information storing capacity. Anyway, the results were frustrating again. The SCNN is the network that reached the highest results: 98% of accuracy with MNIST dataset and 95% of accuracy with CRC dataset. With this kind of network, a second-generation ANN is used for training while a proper SNN is used for inference. Hence, despite no computational improvements were obtained for the training phase, at least the inference latency may be accelerated with the employment of neuromorphic hardware. The comparison between CNN and SCNN, showed us that CNNs are more robust, having almost the same accuracy in all the classes of CRC dataset, while SCNN can reach only 74% of accuracy in the classification of healthy tissues. On the other hand, the SNN classified sick tissue as healthy only in the 4% of the cases, while the CNN classified cancer and adenomas as healthy in the 14% of the cases. This is a key feature when speaking about expert systems for medical classification, where the cost of misclassifying healthy is neglectable compared to misclassify pathological cases.

To summarize, SNNs seem nowadays quite immature to be proficiently used in ML tasks, due to several implementational problems, mainly related to the training phase. Nonetheless, the promising aspects in terms of less required computational time, faster inference, biologically plausible structure and performance that are comparable to the state-of-the-art techniques, are encouraging a lot many research teams to pursue SNNs study and improvement.

Bibliography

[1] About the brainscales hardware. https://electronicvisions.github.io/hbp-sp9-guidebook/pm/pm_hardware_configuration.html#f1, 2019. [Online; accessed 4-March-2019].

[2] About the brainscales hardware. https://www.humanbrainproject.eu/en/silicon-brains/neuromorphic-computing-platform/, 2019. [Online; accessed 7-March-2019].

[3] Convolutional neural networks (cnns / convnets). http://cs231n.github.io/convolutional-networks/, 2019. [Online; accessed 15-March-2019].

[4] How ibm got brainlike efficiency from the truenorth chip. https://spectrum.ieee.org/computing/hardware/how-ibm-got-brainlike-efficiency-from-the-truenorth-chip, 2019. [Online; accessed 5-March-2019].

[5] Ibm demos event-based gesture recognition using a brain-inspired chip at cvpr 2017. https://www.ibm.com/blogs/research/2017/07/brain-inspired-cvpr-2017, 2019. [Online; accessed 5-March-2019].

[6] Introducing a brain-inspired computer. http://www.research.ibm.com/articles/brain-chip.shtml, 2019. [Online; accessed 5-March-2019].

[7] Loihi - intel. https://en.wikichip.org/wiki/intel/loihi, 2019. [Online; accessed 13-March-2019].

[8] Nef summary. https://www.nengo.ai/nengo/examples/advanced/nef_summary.html#Principle-1:-Representation, 2019. [Online; accessed 18-March-2019].

[9] Spinnaker project. http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project, 2019. [Online; accessed 4-March-2019].

[10] Spinnaker project - architectural overview. http://apt.cs.manchester.ac.uk/projects/SpiNNaker/architecture, 2019. [Online; accessed 4-March-2019].

[11] Spinnaker project - the spinnaker chip. http://apt.cs.manchester.ac.uk/projects/SpiNNaker/SpiNNchip/, 2019. [Online; accessed 4-March-2019].

[12] Synapse program develops advanced brain-inspired chip. https://www.darpa.mil/news-events/2014-08-07, 2019. [Online; accessed 5-March-2019].

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

[14] Barchi, Peluso, Ponzio, and Rizzo. Neural gas in action, placing virtual receptors for a spiking neural network classifier. 2018.

[15] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, pages 795–805, 2018.

[16] Wikipedia contributors. Neural circuit — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Neural_circuit`, 2018. [Online; accessed 27-December-2018].

[17] Wikipedia contributors. Neuron — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Neuron`, 2018. [Online; accessed 27-December-2018].

[18] Wikipedia contributors. Action potential — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Action_potential`, 2019. [Online; accessed 28-January-2019].

[19] Wikipedia contributors. Apprendimento automaticico — wikipedia, the free encyclopedia. `https://it.wikipedia.org/wiki/Apprendimento_automatico`, 2019. [Online; accessed 12-March-2019].

[20] Wikipedia contributors. Donald o. hebb — wikipedia, the free encyclopedia. `https://wiki2.org/en/Donald_Hebb`, 2019. [Online; accessed 19-March-2019].

[21] Wikipedia contributors. Feedforward neural network — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Feedforward_neural_network`, 2019. [Online; accessed 19-February-2019].

[22] Wikipedia contributors. Iris flower data set — wikipedia, the free encyclopedia. `https://wiki2.org/en/Iris_flower_data_set`, 2019. [Online; accessed 7-March-2019].

[23] Wikipedia contributors. Machine learning — wikipedia, the free encyclopedia. `https://wiki2.org/en/Machine_learning`, 2019. [Online; accessed 12-March-2019].

[24] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta

Jain, et al. Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[25] Andrew P Davison, Daniel Brüderle, Jochen M Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, 2:11, 2009.

[26] Alan Diamond, Thomas Nowotny, and Michael Schmuker. Comparing neuromorphic solutions in action: implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms. *Frontiers in neuroscience*, 9:491, 2016.

[27] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[28] Chris Eliasmith. *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.

[29] Mazdak Fatahi, Mahmood Ahmadi, Mahyar Shahsavari, Arash Ahmadi, and Philippe Devienne. evt_mnist: A spike based version of traditional mnist. *arXiv preprint arXiv:1604.06751*, 2016.

[30] Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[31] Dan FM Goodman and Romain Brette. The brian simulator. *Frontiers in neuroscience*, 3:26, 2009.

[32] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

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

[34] Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

[35] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[36] Boudjelal Meftah, Olivier Lézoray, Soni Chaturvedi, Aleefia A Khurshid, and Abdelkader Benyettou. Image processing with spiking neuron networks. In *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, pages 525–544. Springer, 2013.

[37] Hélene Paugam-Moisy and Sander Bohte. Computing with spiking neuron networks. In *Handbook of natural computing*, pages 335–376. Springer, 2012.

[38] Filip Ponulak. Supervised learning in spiking neural networks with resume method. *Phd, Poznan University of Technology*, 46:47, 2006.

[39] Filip Ponulak and Andrzej Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71(4):409–433, 2011.

[40] Francesco Ponzio, Enrico Macii, Elisa Ficarra, and Santa Di Cataldo. Colorectal cancer classification using deep convolutional networks-an experimental study. In *BIOIMAGING*, pages 58–66, 2018.

[41] Daniel Rasmussen. NengoDL: Combining deep learning and neuromorphic modelling methods. *arXiv*, 1805.11144:1–22, 2018.

[42] Hannes P Saal, Sethu Vijayakumar, and Roland S Johansson. Information about complex fingertip parameters in individual human tactile afferent neurons. *Journal of Neuroscience*, 29(25):8022–8031, 2009.

[43] Johannes Schemmel, Daniel Briiderle, Andreas Griibl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1947–1950. IEEE, 2010.

[44] Johannes Schemmel, Johannes Fieres, and Karlheinz Meier. Wafer-scale integration of analog neural networks. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 431–438. IEEE, 2008.

[45] Michael Schmuker, Thomas Pfeil, and Martin Paul Nawrot. A neuromorphic network for generic multivariate data classification. *Proceedings of the National Academy of Sciences*, 111(6):2081–2086, 2014.

[46] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 2018.

[47] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural networks*, 14(6-7):715–725, 2001.

[48] Julius von Kügelgen. On artificial spiking neural networks: Principles, limitations and potential.

[49] Tiziano Zito, Niko Wilbert, Laurenz Wiskott, and Pietro Berkes. Modular toolkit for data processing (mdp): a python data processing framework. *Frontiers in neuroinformatics*, 2:8, 2009.