POLITECNICO DI TORINO

Faculty of Engineering

Master of Science in Electronic engineering

Master Thesis

ECG signals classification using Neuromorphic hardware

Advisor:
prof. Danilo Demarchi

Candidate:
Roberto Cattaneo

University tutor
Institute of Neuroinformatics, UZH and ETH Zurich
prof. Giacomo Indiveri

July 2018

Acknowledgements

"The single greatest cause of happiness is gratitude."

Auliq-Ice

If i have to write thank-you sentences to all the people that helped me in these years, this section would take a huge chunk of space within the thesis, and there would be none left to explain the work done.

As a son, i would like to thank my family for always being patiently at my side all the way through this long and arduous journey: during the endless days before the exams sessions, when even the air oozed nervous agitation, and after, whether i was celebrating a success or despairing for a failure. Thanks for having tolerated and advised me during all my endless "moments" thinking hard about the choices i had, drowning in indecision. Finally, thanks for having helped me, economically and logistically, to make the wish of an experience abroad come true.

As a student, i would like to thank all the professors, from primary school to university, who have played an important role during my scholastic career. Without them, i would't be the person i am now. Particular thanks go to Federico Corradi and Elisa Donati, for their invaluable help and support during the thesis work and, of course, i owe a big thank to Giacomo Indiveri and Danilo De Marchi, for giving me the opportunity of a fantastic experience abroad. Finally, i'm really grateful to the Polytechnic of Turin and to the Institute of Neuroinformatics of Zurich for giving me one of the best education that a student can have.

Last but not the least, i would like to thank all the people who have contributed to make these student years one of the most special and enjoyable period of my life: from my friends and school mates in Italy to my flat mates and master students in Switzerland.

Summary

Even if still far too little known, neuromorphic engineering is a new concept that provides for the adoption of modern analog and digital technologies in order to emulate neural networks models, as close as possible to how our brain works.

In this thesis we explore in deep an application of Dynap-se, the neuromorphic chip designed at the Institute of Neuroinformatics, part of the University of Zurich and ETH Zurich, studying its performances on a classification task based on real world signals, such as ECG ones.

As first step we give a significant introduction to the neuroinformatic world: we describe the properties of neurons, from their biological functioning to their computational models. We analyse what happens when bunch of them are connected together to form networks, and how they can learn to solve tasks. All these concepts lay the foundations for carrying forward the dialogue to the main topic: neuromorphic computation. We briefly summarize its history, immersed in the context of what we can call "the need for a revolution of digital computation". We describe what neuromorphic chips are and which is their current state of the art. Finally we explain what is the structure, characteristics and functionalities of Dynap-se, the chip which is behind all the performed analysis.

As second step we give basic knowledge on reservoir computing, the computational method used to solve the tasks that we have addresses, along with a basic description of the nature and characteristics of biological signals, in particular ECG ones. We describe in detail DYNAPSETools, freely available python library designed to simplify the interface with Dynap-se control software, and on which all the performed analysis rely. The description goes on with the details of all the experiments that have been carried on: from the solution of simple tasks, such as the classification of sinusoidal signals, to more complex ones, such as the classification of artificially created ECG signals.

As third step we show the performances of the reservoir network to solve the designed tasks: from 100% accuracy for the simple classification of sinusoidal signals, to the little drop at 98% for the more challenging task about the perturbation detection on sinusoids. We then switch the attention on the main characteristic of the reservoir network, which is the memory of the past stimuli, giving the results of its estimation using custom designed

analysis. The description goes on with the main task thesis task, the perturbations detection on ECG signals. We describe the performances of the network over three different types of perturbations: 94% for missing QRS ECG signals, 74% for missing T wave and 62% for missing P wave.

Finally we discuss the possible approaches that may improve the classification performances and speculate on which may be future application of neuromorphic devices.

Contents

Chapter 1

Introduction

> "Emotions don't seem like a very useful simulation for a robot... Hell, I don't want my toaster or my vacuum cleaner appearing emotional..."
>
> *Spooner in "I, Robot" movie*

The joke written in the quote above, although funny, it emphasises how humans have long held a fascination on the idea of emulating what our brain can do, even experience emotions. If you think about it, our brain is really a powerful and efficient tool. We can navigate in a complex world and interact intelligently with it. We can know at a first glance the humour of a person, wonder why he is sad or happy, and even guess what might have happened. All of this, and much more, using around an equivalent of 20 W, less than half the power used by an incandescent light bulb. Such performances are amazing and, although it seems a long way off to get something even similar to the brain, research on this field seems going in the right direction. Neuromorphic engineering is a promising step toward this goal and would probably have a major role in the future attempts to artificially replicate the functionalities of our brains.

Since, at the time of its writing, this is one of the first documents trying to demonstrate an application of neuromorphic chips, it is our duty to deeply introduce the argument, maybe unknown to most. For this reason the first part of the thesis contains a significant introduction to the topic. Neuromorphic engineering, however, is a multidisciplinary subject that, to be fully understood, requires knowledge of biology, physics, mathematics, computer science and electronic engineering. Although an introduction to each of these subjects is not physically possible, we tried to be as complete as possible, maintaining at the same time a clear and plain language, understandable independently from the reader background. Things are complex by itself, we don't have to make them more complex using difficult language.

In Chapter 2 we will explore the world of neurons and synapses, starting learning how

they work from a biological point of view. Such concepts will help us understand better the origin of the simple equations that model their behaviour, described in the same chapter. Then the focus will quickly steer into the description of the main features of neurons, which are the learning mechanism and the neural networks.

Chapter 3 valiantly tackles the main topic of the thesis: neuromorphic computing. The first lines are dedicated to a brief introduction of its history, immersed in the context of what we can call "The need for a computational revolution". Then the description carries on toward less philosophical topics, as what are neuromorphic chips and which is their current state of the art. At this point the concept of Dynap-se, the star of the thesis, shows up. We describe in detail the structure, characteristics and operation of such neuromorphic chip because it behind all the performed analysis, so it constitutes a topic of great importance.

With chapter 4 the general introduction to neuroinformatics and neuromorphic ends. It starts, instead, an overview of arguments that are strictly related to the thesis work. The chapter addresses the computational properties of the so called "Reservoir", focusing on his implementation with spiking neural networks, the Liquid State Machine (LSM). In Chapter 5, instead, is presented a brief description of biomedical signals, focusing on a particular type of them, the ECG signal. These two additional introductory chapters are meant to give some basic knowledge concerning the nature of the signals that has been addressed in the thesis work, and the computational models used to perform analysis on them.

With chapter 6 we get in the heart of the thesis. In a preliminary stage we briefly explain the structure and the characteristics of Dynap-se control software, in order to constitute the foundations on which is placed the subsequent description of DYNAPSETools. This last one is a high-level python library developed to simplify the use of the device. We inserted its characterization in parallel to Dynap-se operating instructions, highlighting in this way the advantages with respect to the previous approach. The chapter concludes with a detailed description of the methods and characteristics of the performed experiments, in such a way that they can be easily reproduced.

Chapter 7 tackles the description of the network performances to solve the designed tasks. In the first part of the chapter we describe how we can manipulate the time constant of Dynap-se neurons and synapses, and how their values influence the activity inertia of the reservoir network. Such analysis is really important because it is at the foundation of the reservoir network main feature: the memory of past stimuli. The description goes on with the analysis of reservoir performances on three tasks: classification of sinusoidal waves, perturbation detection on sinusoids and, finally, perturbation detection on ECG signals. We will describe the behaviour the network, analysing how the memory tuning influences the achieved performances on the solution of the proposed tasks.

The final Chapter 8 wrap up the thesis with a discussion about the key points and the

strategies that may lead to performance enhancing. The last words are dedicated to speculate about which might be the structure of a neumorphic powered device that can deal with real world application.

Chapter 2

Neurons, Synapses and neural networks

> "The human brain has 100 billion neurons, each neuron connected to 10 thousand other neurons. Sitting on your shoulders is the most complicated object in the known universe"
>
> *Michio Kaku*

We all know that our brains are composed by basic blocks called neurons. Maybe less people know what it is stated in the quote above, which highlights the magnitude of their complexity. But for sure not many know how neurons work, how they are connected and what are their characteristics. In this chapter we are going to discover that, and more, by trying to answer at very simple questions.

2.1 What are Neurons and Synapses?

2.1.1 Neurons

An example of Neuron structure is shown in fig. 2.1: it consists of a soma, where main cellular functions take place in order to maintain the cell alive, the dendrites, through which it receives inputs from other neurons, and the axon, long extension that carries its output signal. The Myelin sheath and the nodes of Ranvier present on this last part serves mainly to increase the speed of the signal travelling inside. Axon terminals connect to the dendrites of surrounding neurons, bringing the output of one cell to the input of another one.

What neurons do?
Neuron functioning is pretty simple: it combines inputs coming from its dendrites and, if some conditions are met, it produces an output pulse, called spike, all the way through its

axon. When such signal reaches its terminals, it may be completely or partially transmitted to the connected cells.

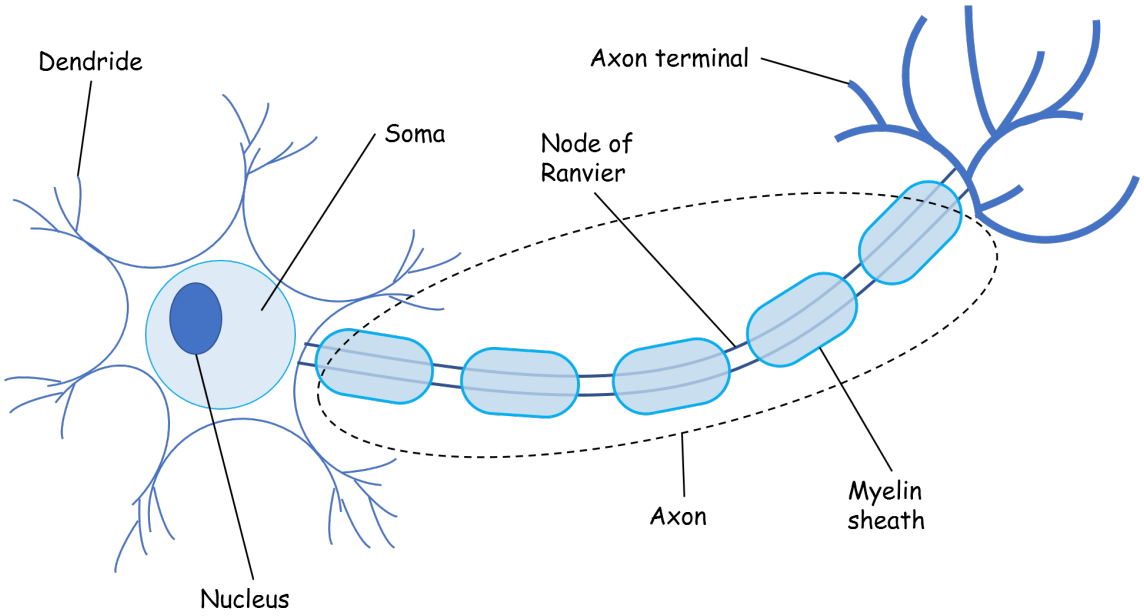

Figure 2.1: Schematic structure of a neuron

How neurons work?
A Neuron can be considered as, in a simple and funny description given by Rajesh P. N. Rao, a "leaky bag of charged liquid". This liquid, composed of Sodium Na+, Chlorine Cl- and Potassium K+ ions, is confined inside the neuron body (alias the bag), thanks to the presence of an impermeable lipid bilayer membrane (alias the bag plastic walls). Some "holes", called leaky channels, allow the passage of these ions through the lipid bilayer. In fig. 2.2 is present a description of the neuron membrane structure.

In steady state conditions, inside the neuron membrane there is a bigger concentration of potassium ions than outside, while for sodium ions and chlorine ones the situation is the opposite. This equilibrium is maintained thanks to two physical phenomenon taking place:

- diffusion: it produces movement of ions to places where their concentration is lower, as balls on a unstable pyramid tend to do. In the neuron the potassium would tends to go outside (where it is less concentrated) the membrane while the sodium and chlorine would tend to come inside (where they are less concentrated), passing through the leaky channels.
- electrostatic interaction: depends on the reciprocal attraction or repulsion of ions with different or equal charge. The potassium ions, going outside, create a negative

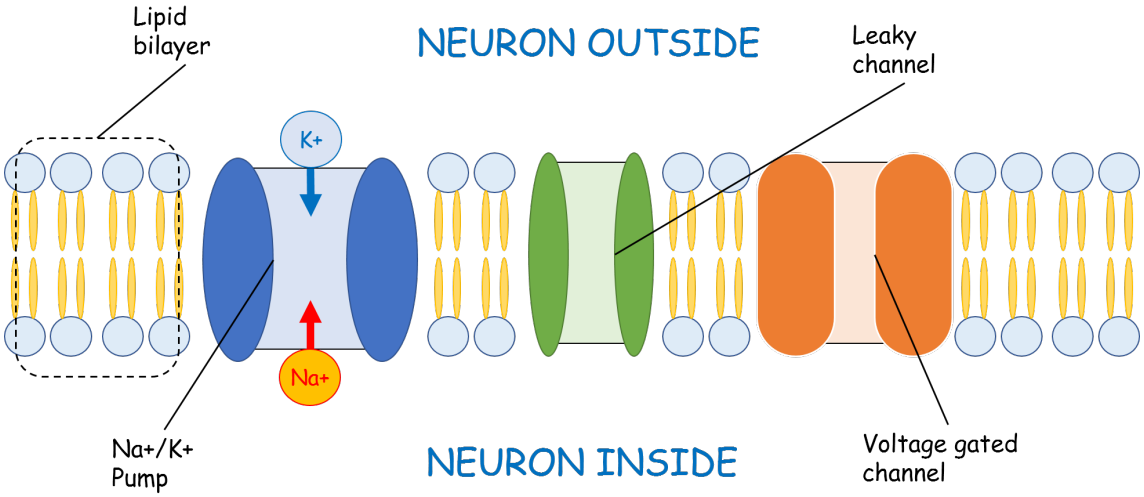

Figure 2.2: Schematic structure of the membrane of the neuron

charge inside the membrane, that would attract the positive ones present outside. The same happens for sodium.

Both these dynamics would tend to cancel the difference of potassium and sodium concentration between inside and outside membrane, but here come the Na+/K+ pumps. These channels, like real pumps, act against the leaky holes by bringing inside the potassium and kicking out the exceeding sodium, trying to restore the imbalance in concentration. The result of all these contributions is a disequilibrium in the final charge distribution: there are more positive charges outside than inside the membrane, resulting in a negative potential from inside to outside the membrane, which is the characteristics -70mV of a neuron.

Neuron potential, however, does not rest forever at -70mV, but produces the dynamic behaviour depicted in fig. 2.3, called spike. Such dynamics is characterized by a steep increasing of membrane potential, called depolarization, followed by a decreasing to the original resting potential, called hyperpolarization. This phenomenon is possible thanks to the action of a second type of "holes" present in neurons, the voltage gated channels (the orange structures in fig. 2.2). These last one, in contrast to the leaky ones, opens only for when the neuron membrane potential reach certain levels. Sodium channels opens around -55mV, while potassium ones opens around 30mV. When they open, ions flow in or out the membrane, changing the charge distribution and, hence, the membrane potential of the neuron, each one of them in a different way. Sodium tends to depolarize, so increase, the potential while potassium tends to hyperpolarize, so reduce, the membrane potential.

The effects of these two channels are enough to explain how a spike is generated. When the neuron potential raises, it passes through the -55mV level, triggering the opening of sodium channels. These will let inside the neuron a rush of positive sodium ions, with

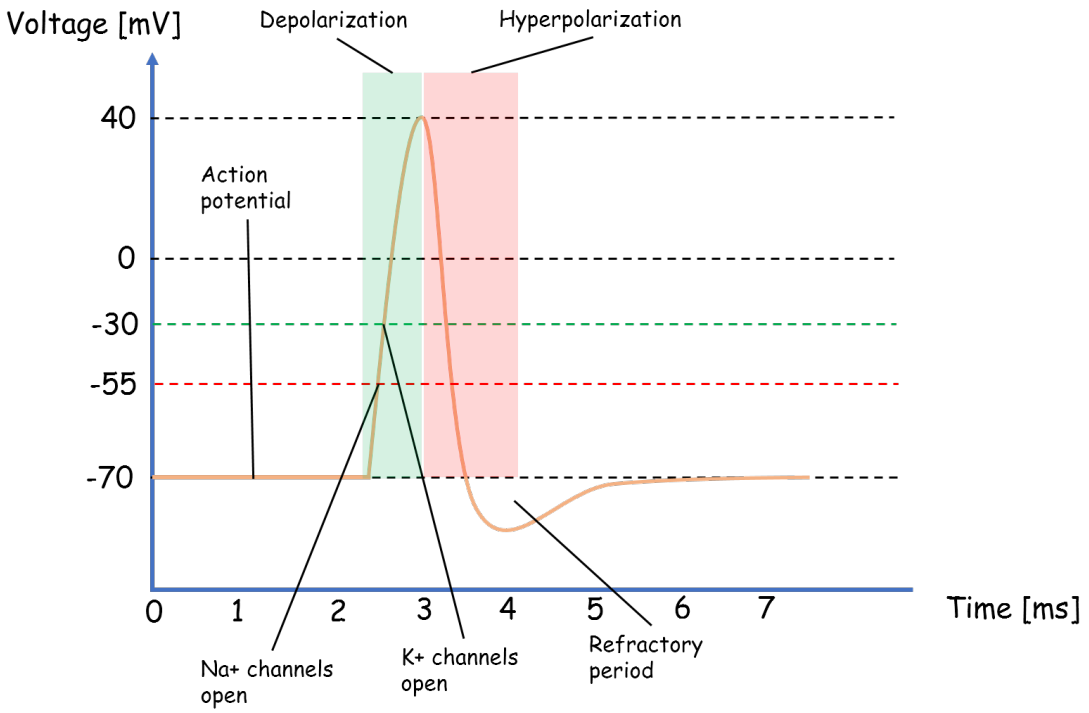

Figure 2.3: Graph explaining the action potential generated by a neuron.

consequent sharp depolarization of the membrane potential. The result of such mechanism is the action potential, also called spike. As voltage raises, sodium channels tends to close, while potassium ones begin to open, since we crossed the threshold of -30mV. Positive potassium ions starts flowing outside the membrane, leading to a decreasing of the membrane potential. At this point we have a configuration completely different from the resting one: a lot of sodium inside the membrane, and lot of potassium outside. It's the job of leaky channels and Na+/K+ pumps to bring back te neuron on the original conditions, allowing exchanging of ions until the resting potential is restored. During this "recovering" period, called refractory period, the neuron is insensible to inputs and unable to create a new action potential.

2.1.2 Synapses

In the previous paragraph we have described in detail all the passages that bring a neuron to create a spike. We have said that everything starts when the neuron potential goes above -55mV, triggering the opening of sodium channels. However, this statement raises a natural question: how can the membrane potential be moved away from the resting state? One of the ways this can happen is due to external influx of charges, which takes place by stimulating the neuron through structures called Synapses [1]. These last ones can be considered as "junctions" between cells that, even if there is no physical contact between them, connect the terminal part of the axon belonging the pre-synaptic neuron with the

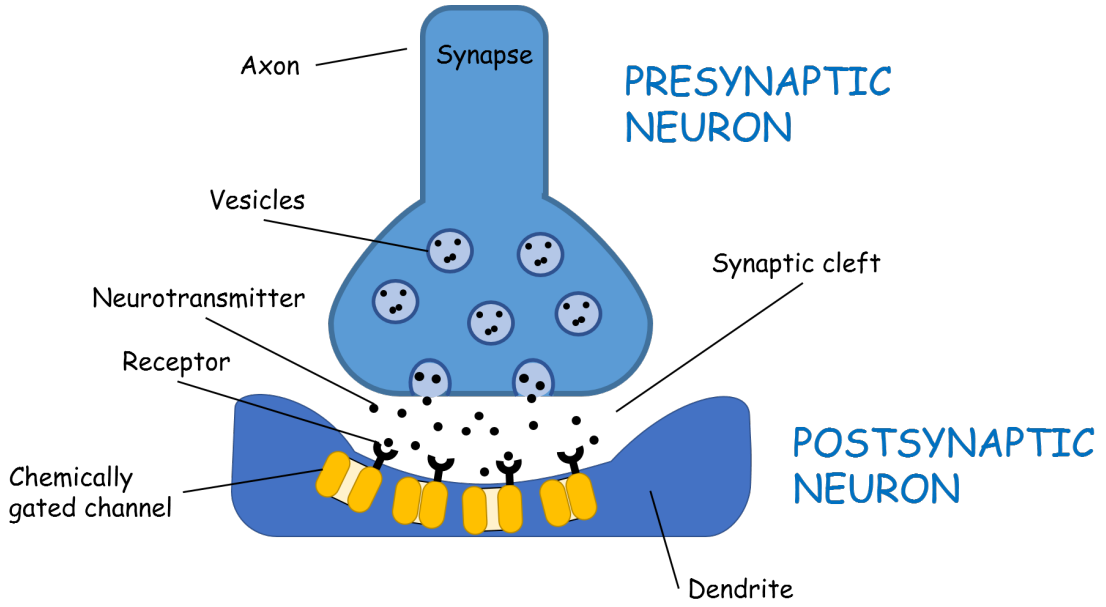

Figure 2.4: Schematic picture explaining the structure of a synapse

tip of a dendrite belonging to the post-synaptic neuron. An example is visible in fig. 2.4.

When an action potential, coming from the presynaptic neuron, reaches the extremity of the axon, it cannot directly propagate to the post-synaptic one because of the presence of a gap, called synaptic cleft. However, in order to perform the propagation, it triggers the release of chemical messengers, called neurotransmitters, inside the synaptic cleft. On the surface of the other side of the gap, in the post-synaptic neuron, there is a third type of ion channels, called chemically gated channels. Their name comes from the fact that they open only if certain chemicals bonds with them, for instance the neurotransmitters present in the synaptic cleft. When the bonding is performed, what happens next depends on the type of ion channel and the type of neurotransmitter. Some of them will create an influx of sodium, creating depolarization of the membrane potential; for this reason, they are called excitatory synapses. Others, instead, allow an out flux of potassium ions or influx of chlorine ones, creating a hyperpolarization of the membrane potential. Hence, they are called inhibitory synapses.

Synapses, in other words, are bridges that permit neurons to talk each other, and they are one of the main "ingredients" needed for the generation of neural networks.

2.2 How do we model neurons?

We have seen, in section 2.1, that neurons involve lots of complex biological and physical phenomena. A mathematical model developed at such level of abstraction would explain

very precisely the whole dynamic of a neuron but, at the same time, his complexity would make impossible any elaboration in the context of networks of neurons. For this reason today's computational models catch only the essential of neuron dynamics.

Many classifications rules can be adopted to separate neurons models in different groups. One of the most common is the separation between artificial and spiking neurons.

2.2.1 Artificial neurons

When analysing large networks of neurons, it can be very helpful to forget about their complex biological behaviour and focus only on their general functioning principle, explained at the beginning of section 2.1: a neuron makes a combination of its inputs, generating an output only if the result satisfies certain conditions. Perceptrons and Sigmoid neurons, which are described by artificial models, work exactly in that way. Their characteristics are shown in fig. 2.5.

Perceptrons make a linear combination between the input signals (I1, I2, I3) and a set of parameters, called weights (w1, w2 and w3). If the result of this combination is bigger than a certain threshold (thd), the output is one, otherwise is zero.

Sigmoid neuron works in a similar way, but applies a non linear function, called sig-

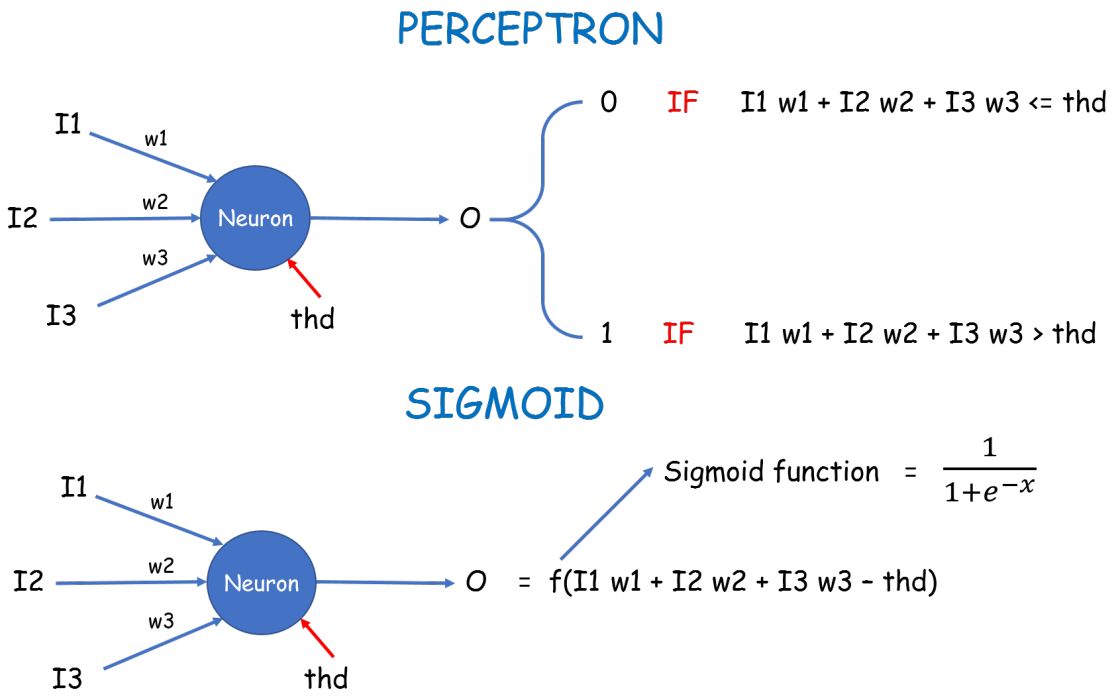

Figure 2.5: Schematic picture of Perceptron and Sigmoid artificial neurons models

moid, to a combination of input, weights and threshold. The result of this operation is that, in contrast with the Perceptron, its output can assume values in all the spectrum of real numbers from 0 to 1. Often such value is interpreted as the neuron normalized firing rate, which is the number of spikes produced in a certain amount of time. The normalization is such that an output value equal to 1 corresponds to the maximum firing rate that the neuron can realise.

Connecting multiple instances of this object together we create the so called artificial neural networks, or rate based networks, because they encode the information on the firing rate. It is evident that the simple model offered by artificial neurons is well suited for fast or large scale simulations, for this reason they are widely used for machine learning applications.

2.2.2 Spiking neurons

If we take into consideration the spiking activity of neurons, things become more complex. We need to find equations that relate the inputs to his membrane potential, define when a spike occurs, and what happens after. Since neuron biological behaviour, as explained in section 2.1, involves movements of ion charges and electrical potentials, in the past people started thinking about modelling their behaviour using electrical circuit theory. From this idea, different mathematical neuron models have been developed, which differ on how good they can predict the biological behaviour in different dynamics conditions.

Since a detailed description of spiking models is not in the scope of this thesis, we will focus our attention to the most used one: the I&F model, discovered by Feng and Brown in year 2000. A graphic summary of its composition is depicted in fig. 2.6.

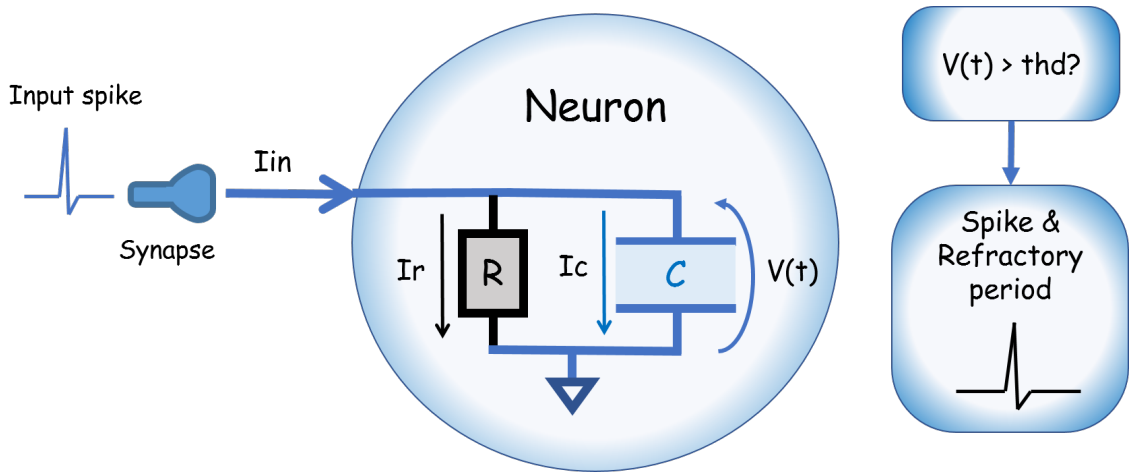

Figure 2.6: Schematic picture summarizing I&F model structure

The soma is modelled as a capacitor C in parallel with a resistor R, both driven by a current Iin. This last one is injected in the neuron "dendrite" from the axon of the presynaptic neuron, filtered by the synapse present between them. Such current can be expressed in the following way:

$$I_{in} = I_r + I_c$$

and is made of two components: the first current is the one flowing through the resistor R.

$$I_r = \frac{V(t)}{R}$$

the second current, instead, charges the capacitor C.

$$I_C = C \cdot \frac{dV}{dt}$$

Inserting the previous definitions inside the one for the total current, we can obtain a differential equation expressed in terms of the neuron membrane potential V(t):

$$I_{in} = \frac{V(t)}{R} + C \cdot \frac{dV}{dt} \tag{2.1}$$

Multiplying both members of Equation 1 by factor R, and introducing the time constant $\tau_m = RC$ of the leaky integrator, we obtain the canonical form:

$$\tau_m \frac{dV}{dt} = -V(t) + R \cdot I_{in} \tag{2.2}$$

With eq. (2.2) we have found a relation between neuron input and his membrane potential, expressing a current low pass filter. However, this representation is too simple to include the spiking behaviour of the neuron. To add this dynamics, we need to insert a condition:

$$if\ V(t) > thd\ \ -> \ \ spike \tag{2.3}$$

Such non linear condition establishes the behaviour of the neuron when the threshold is crossed, which corresponds to the generation of a spike. But what happens to the membrane potential after? Recalling from the biological description of section 2.1, neurons enter in a state of refractory condition, where the membrane potential is stable at a certain characteristic value. In order to model such dynamics with equations, we need to include another condition, expressed as:

$$when\ spike -> \ V(t) = V_{refr} \quad t_{spike} < t < t_{spike} + t_{refr} \tag{2.4}$$

In other words, for a period of time lasting t_{refr}, the membrane potential $V(t)$ is fixed to the value V_{refr}, which is the refractory period voltage.

The final neuron equation, including eq. (2.2), eq. (2.3) and eq. (2.4), can be easily implemented in software in order to simulate spiking neurons behaviour.

2.3 Synaptic plasticity and learning

So far we have explained how neurons behave from a biological point of view, and how they can be modelled with equations. However, we skipped one of their most important features, which is the true power of neurons: learning. Every day, in our daily life, we automatically experience its effects on our brain: for instance we learn new roads to go to work, or new concepts when studying at school, and so on. What happens in our brain in these situations?

Of course a detailed explanation of all the complex neural dynamics taking place in our brain, is out of the scope of this thesis. However, we can state that such high level learning comes from a low level phenomenon involving our neurons and synapses, called synaptic plasticity.

Recalling what we have said in section 2.2, neuron dynamics is influenced by spikes coming from other neurons, filtered by the action of the synapses. More precisely, a synapse scale the input in relation to a parameter called "synaptic weight". Such value express how strong is the influence of a certain input to the dynamics of the neuron. Synaptic plasticity is an automatic biological mechanism which modifies the weight values, changing the neuron response to the input stimuli. The final result of this operation is what triggers learning.

For instance, lets consider an example: we see a cat, but we don't know which breed belongs to. Our brain react to the visual stimuli with a response that doesn't give us any useful information. In other words, we experience the feeling of "i don't know which breed belongs to". Suppose that, later on, we get to know this information. From that moment on, we would certainly be able to recognize the breed of that cat with just a glance, even if the visual stimuli are the same as before learning it. In this situation synaptic plasticity phenomenon took place in our brain neurons. It influenced the weights of our internal connections such to transform the previous insignificant output in the learnt answer, which corresponds to the breed of the cat.

Besides how this automatic weight change take place, it is clear that the core of learning resides just in the weight changing. Tuning their values we can modify the activity of the neurons to accomplish certain tasks.

2.4 Neural networks

During the course of this review about neuroinformatic world, we focused mainly on the description and characterization of single neurons. In truth, as humans are capable of creating great things when they work together, the same happens for neurons. Their real power manifests itself only when they are placed together with other neurons, in order to form the so called neural network. A neural network is a collection of neurons interconnected together. Although there are many ways of classifying them, we will limit to discuss

only the most important distinctions: between artificial and spiking neural networks and between feed forward and recurrent neural networks.

Artificial neural networks (ANN) are, as the name suggests, networks of artificial neurons. These last ones, as described in section 2.2, can be modelled using the perceptron approach or the sigmoid one. Connections between neurons are pretty easy because there is no intermediate entity: the output value of a neurons can be directly used as input of other neurons. Such networks indeed does not aim to replicate the circuits present in our brain, but they are artificially made to accomplish practical tasks. For this reason their main area of expertise is, in conjunction with machine learning, to perform recognition tasks, such as speech, digit or human face recognition, etc.

Spiking neural networks (SNN) are, instead, networks of spiking neurons. Each one of them is modelled with biological realistic models and communicate with other neurons by the means of synapses. Such networks aim to model how the human brain works, like what has been done in Standford university with Spaun, the world's largest behaving brain model.

The main difference between these two types of networks is in the way they encode information. In ANN neurons "communicate" using floating point values going in a range from 0 to 1, which can be physically interpreted as normalized instantaneous firing rate. In SNN, at opposite, neurons encode information directly in every single spike. From a computational point of view, this choice gives SNN advantage over ANN. Spikes, in fact, encode information in a more powerful way than simple floating point values, because they inherently bring with them the timing knowledge. However, this advantage comes at a price: comparing the SNN and ANN models present in section 2.2, we clearly notice that the former ones are much more complex than the latter ones. This consideration leads to the fact that simulations of SNN are more computational expensive than the ANN.

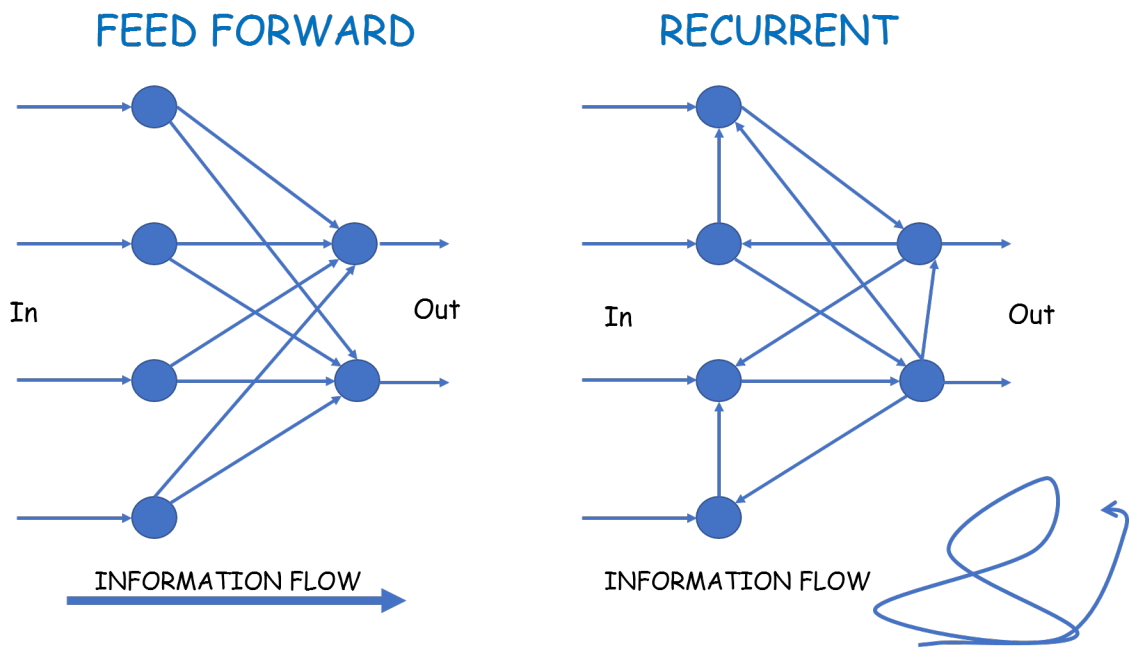

Figure 2.7: Example of a recurrent and recurrent neural network

Although SNN and ANN can be arranged in different ways, we will concentrate on the most general ones: feed forward and recurrent neural networks. Figure 2.7 shows a comparison about these two configurations.

Feed forward neural networks are organized neuronal structure where all the connections have the same direction, in the example from left to right. This fact imply that the activity of the network, and so the information carried by neurons, flows in one direction only, away from the source that is responsible for it. For this reason no memory behaviour can be induced in the network.

Recurrent neural networks, on the other hand, are characterized by high flexibility on the connectivity infrastructure: a neuron on the last column can be connected to a neuron in the first one, and so on. Such structure, for construction, collects input stimuli and react to them continuously, until its influence is too weak to trigger new activity in the network. In other word, a recurrent neural network is able to hold memory of the past stimuli.

Chapter 3

Neuromorphic computing

> "Many have suggested Moore's law is ending and that means we won't get 'more compute' cheaper using the same methods"
>
> *Chris Eliasmith*

In this chapter we will explore in deep the meaning of "neuromorphic computing", a word that back in the 80s was letting out his first cries, but is now increasingly assuming more importance since important companies, as IBM or Intel, started exploring this branch of neuroscience.

3.1 The need for a computational revolution

In the years after the 1960s, the dawn of IC fabrication, computation started be carried on by little building blocks, called transistors. Information, codified in the shape of 1's and 0's, was manipulated simply by switching their state. Those years were the beginning of the so called Digital Revolution.

In the mean time, an empirical law, described in a paper written in 1965 by Gordon Moore, started predicting the extraordinary phenomenon of transistor scaling. It's the well-known Moore's law: "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year". A law that nowadays is struggling to survive, but still valid. In simple words, he predicted that the amount of transistors present inside a chip would double every 18 months, and their speed would be double than before. Prediction that turned out to be valid and precise. Processors started getting more and more computational power, and could run algorithm with increasing complexity. Indeed, new technological features started appearing in digital devices. But such amazing technology scaling came at a price: the power consumption of each transistor increased dramatically, and along with it also the one required by the integrated circuits. Such progress went on for almost 50 years until about 2005, when the so called "power wall" was reached.

Microprocessors power consumption were so high, they reached the limit of commercial cooling technologies. That was a crucial point, where seemed difficult any further improvement in performances. A radical change of approach was definitely needed, and take place by developing a new way of designing computation, called power-aware computing. The meaning of this term is straightforward: put power consumption at the first place in the design of new computing devices, without losing performances. One of the consequences of this new computation approach was the introduction of multi core processors. Instead of using a single processing unit, work-load started to be shared between different "copies" of it, called cores. Increasing their number is possible to increase the performance of the device in an efficient way, although not with the same rate as before.

In parallel to this evolution we have the development of neuromorphic engineering [2]. Its origins date back to 1980s, with the work of Carver Mead, trying to find a way to mimic neural brain functioning on silicon hardware. Although human brain is surpassed by computer in many ways, it has many features that lack even in the most powerful pc. We actually analyse images, sounds, digits and classify them in many different ways, with high accuracy and no effort. The most impressive data, however, if the efficiency: human brains needs an amount of power equal to about 20W, much lower than most of newest processors present in the market. This unique features make them good candidates for a new way of making computation, that has been literally hiding in our mind. A bio-inspired hardware that can perform elaboration in a similar way as we do, with high parallelism, low power, self learning.

We arrive in this way at nowadays. Driven by the market necessity to have products with new functionalities, two topics started growing in popularity in last years: machine learning and artificial intelligence. Even if they are pretty old concepts, they became really important in the first two decades of this century. With machine learning computers can get knowledge from big amount of data and gain the ability to make predictions on new versions of them. Machine learning task are, for instance, image, speech or hand-written digit recognition. Artificial intelligence aim, on the other hand, is to develop machines that can perform task in an intelligent way. Even though these powerful methods provide solutions to problems that could't be solved before, high computational power is required to run their algorithms. This make impracticable their integration in slow or low power devices.

Machine learning and artificial intelligence brought man on the way leading to the realization of devices that can learn and behave intelligently. But the journey is hard and the digital computation is showing its limitation in the possibility of achieving this goal. On the other hand there is neuromorphic engineering, bringing the possibility to design and emulate neural networks, as well as complex learning algorithms, in an efficient low power way. These great features are the reason for which, in the last years, such branch of neuroscience started to get much more credit. Neuromorphic computing can be, in this way, the "computation revolution", as the one happened in 2005, that can move the first steps toward the realization of cognitive devices.

3.2 Neuromorphic chips

Neuromorphic chips are experimental hardware devices containing analog, digital, or mixed analog-digital circuits able to simulate or emulate in real time networks of interconnected neurons [3]. From a physical point of view, they resemble CPU chips present in all kind of electronic equipments, but there is a substantial difference. Instead of performing standard digital computation, hence working with 0's and 1's, meuromorphic chips are composed by basic blocks that communicate together as our brain neurons do.

While CPUs or GPUs are massively producted by giant companies such as Intel, AMD, NVIDIA, Qualcomm, etc., neuromorphic chips are in experimentation and development phase. Only few research groups in universities or companies are currently designing and producing these kind of devices. Figure 3.1 summarize all the known research groups working on neuromorphic chips, divided according to the way in which neurons (and synapses) are emulated. We can classify them in two types:

- Digital powered: Neurons and synapses are simulated by high speed digital hardware. In other words, the chip contains digital operators like adders, multipliers, etc. that, correctly driven, perform calculations of neuron characteristic variables, for instance the membrane potential.

- Analog powered: Neurons and synapses behaviour is completely emulated by custom analog electronic circuits. In other words, the chip contains basic blocks whose electronic dynamics exactly match the neuron behaviour. They do not calculate, for instance, the membrane potential of a biological neuron, but they directly reproduce it.

In fig. 3.1 are depicted the main groups working in neuromorphic hardware, divided according to the chip characteristics [4]:

- **IBM TrueNorth**: It is a neuromorphic ASIC chip produced by IBM. it contains 4096 custom asynchronous digital processor cores of spiking neural networks that can mimic a total of 1 million neurons and 256 million of programmable synapses.

- **Intel Loihi**: It is a mixed neuromorphic-digital chip produced by Intel. It contains 128 digital neuromorphic cores and 3 additional x86 processor cores, as well as logic for off chip communication. It can emulate up to 130000 neurons and 130 million synapses.

- **APT SpiNNaker project**: Product of the APT Manchester research group, that is part of the human brain project (HBP), used with the goal of speeding up neural network simulations. Each chip contains 18 ARM968 digital processors exchanging information by the means of asynchronous routers.

- **DYNAP-le**: Product of INI neuromorphic research group [5], it contains 256 fully analog neurons and three types of synapses: 64k of programmable analog synapses

with short-term plasticity and 64k with long term one, and 256 shared analog synapses.

- **DYNAP-se**: Product of INI neuromorphic research group [6], it contains 4 cores with 256 analog neurons each, for a total of 1024 neurons. Each of them can be stimulated by a maximum of 64 synapses, for a total of 64k analog synapses. Further details will be described in section 3.3.
- **NeuroGrid**: Product of Standford university neuromorphic research group, it contains 16 CMOS cores (a single instance is called "NeuroCore") which are linked in a tree network, each one containing mixed analog-digital 64k neurons, for a total of 1 million neurons; synapses, instead, are shared among neurons.

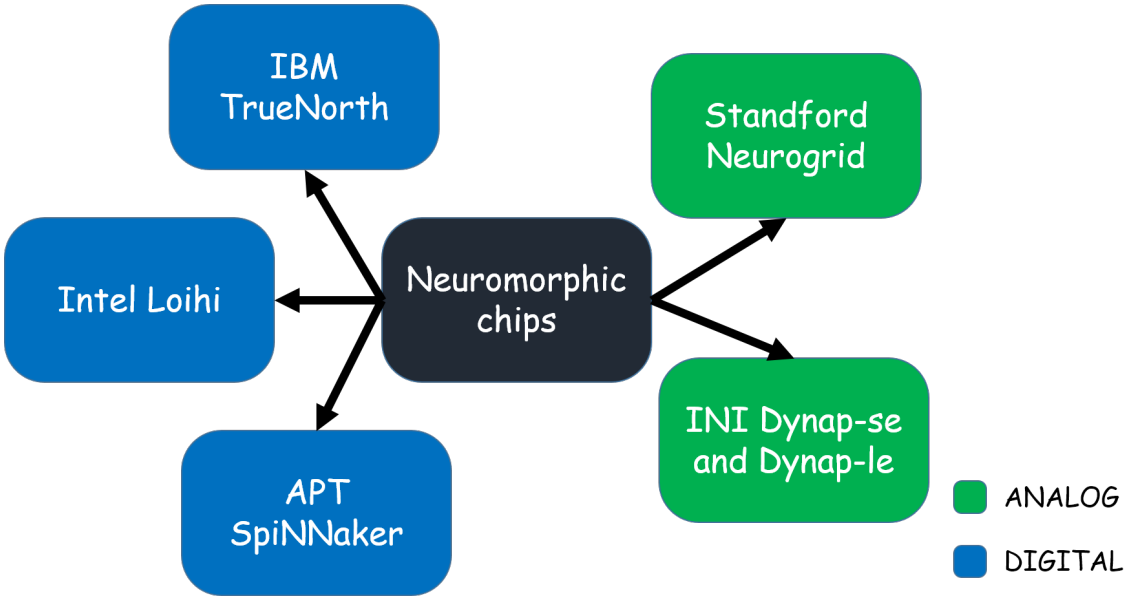

Figure 3.1: Schematic representation of the main neuromorphic chips research projects currently active

3.3 DYNAP-se neuromorphic chip

So far we have provided a general introduction to neuromorphic engineering, as well as an overview of the main chips present in the market. In this section, instead, we will focus on the description of the neuomorphic chip which has been used for the thesis project, called DYNAP-se.

3.3.1 General structure

The first part of the name, DYNAP, stays for Dynamic Neurormorphic Asynchronous Processor, which indicates an integrated hardware architecture able to perform elaborations

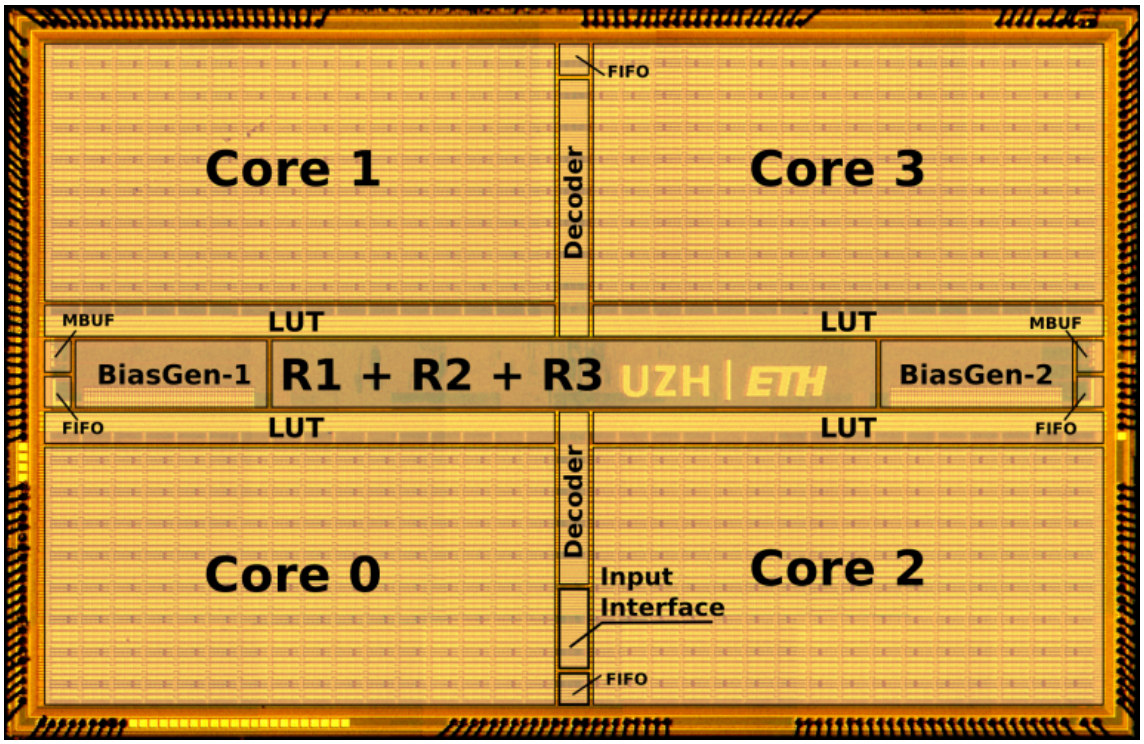

Figure 3.2: Internal schematic structure of a Dynap-se neuromorphic chip. Image courtesy of aiCTX company.

in a totally asynchronous way, following the rules of neuromorphic engineering. The "-se" at the end of the name stays for "scalable", meaning that it supports the connection with other chip instances, in order to scale up the complexity of the neural networks that can be designed.

Dynap-se is composed of 4 interconnected blocks called cores, each one of them containing 256 analog neurons. The other blocks shown in fig. 3.2 are mixed analog-digital architectures responsible for the configuration of neuron dynamics, connectivity between neurons and input output communication:

BiasGen-1 and BiasGen-2 are temperature compensated bias generators. Even if there are only two instances, they can supply 25 different high-precision voltages for each one of the 4 cores present inside the chip, for a total of 100. Changing their value it is possible to tune the electrical properties of the neurons and make them exploit a wide range of dynamical behaviours.

R1, R2 and R3, instead, are three level of asynchronous digital routers that, acting like semaphores, handle the connectivity between neurons and the communication with the external world.

The Dynap-se basic block

Looking at fig. 3.2, it is possible to notice that every core is made of a structure of 16 rows by 16 columns of yellow squares. These elements are the core of Dynap-se, the basic blocks that carry out the computation. To understand better what they are made of, we can look at the schematic description present in fig. 3.3.

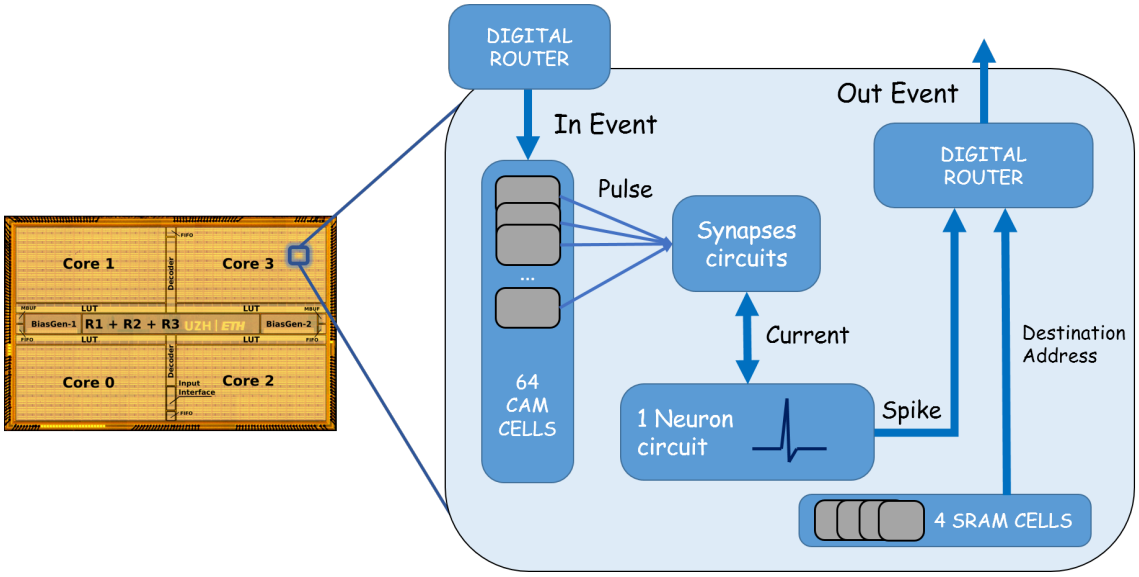

Figure 3.3: Schematic structure of Dynap-se basic block

Even if not visible in fig. 3.2, every basic block contained inside a core is made of different digital and analog components. Before entering into a detailed description of everyone of them, we want to give the big picture by introducing the general functioning of the block.

The core of the block is the neuron circuit, a network of electronic transistors that simulates the real dynamic of his biological counterpart. Whenever the neuron spikes, a digital data, called event, is generated and transmitted outside the block by the means of digital routers. The event contains information about the neuron that produced it, and which is his destination address or, in other words, where it should be routed to. This operation, referring again to the biological world, can be compared to the transmission of a output spike all the way thought the axon.

A similar process allows the neuron to receive inputs. Whenever a valid event presents itself at the input of the block, a digital pulse is generated and directed to the synapses circuits. These last ones are, as neurons, electronic transistors that simulates their real biological behaviour. The pulse is filtered and a proportional current is supplied or absorbed from the connected neuron, according if the synapse is excitatory or inhibitory type.

A more detailed explanation of the communication system, as well as a description of the role that CAM and SRAM cells have, is present in section 3.3.4.

3.3.2 Silicon Neurons

As introduced in the previous sections, Dynap-se has some characteristics that makes it very different with respect to the majority of other neuromorphic hardware. One of them is how neurons are implemented. Instead of using transistors to create digital hardware, and then use it to run algorithms simulating neuronal dynamics, Dynap-se adopt another approach. It uses transistors in a particular way such that the resulting silicon circuit directly emulates the behaviour of biological neurons.

This is not a trivial task. Neurons and transistors have completely different dynamics and, in addition, there an huge difference between neurons and transistors time constants: while the former one are on the order of millisecond (or even seconds), the latter one are much lower, in picoseconds scale. In other words, transistors are 1 billions times faster than neurons. A natural question raises: "How is possible to slow them down of such a big amount of times?".

The answer lies in the different ways transistors can operate. In normal operating conditions transistors behaves as fast switching devices, due to the fact that they are able to drain big amount of charges in no time, as a big tap discharge water very fast from a small tank. To increase the time needed to empty the tank, a natural idea would be to limit the flow of water going out through the tap. The same concept can be applied to transistors. Forcing them in a state where their capability to conduct charges is limited, it would be like "closing the tap a little". This correspond, in practice, to force them into sub threshold operation state. In this conditions, the current that a transistor can sink or supply is way smaller than the one in normal operating conditions (called strong inversion). The consequence is that their time constant becomes much higher than normal, comparable with the neuron one.

DPI circuit

Each neuron present in Dynap-se is composed of different blocks, performing each one a different functions. One of the most important part is the DPI circuit, visible in fig. 3.4.

From section 2.2 we have seen how spiking neurons can be modelled as current filters that, when particular conditions are reached, generate spikes. Indeed, DPI is a circuit of transistor that, operating in subthreshold conditions, is able to filter an input current in the same way as the biological neuron model does.

DPI complete equation is shown in eq. (3.1). Its derivation is outside the scope of this thesis, but the detailed explanation is reported in [7].

$$\tau(1 + \frac{I_{th}}{I_{out}})\frac{d}{dt}I_{out} + I_{out} = \frac{I_{th}I_{in}}{I_\tau} - I_{th} \tag{3.1}$$

where $\tau = CU_T/\kappa I_\tau$

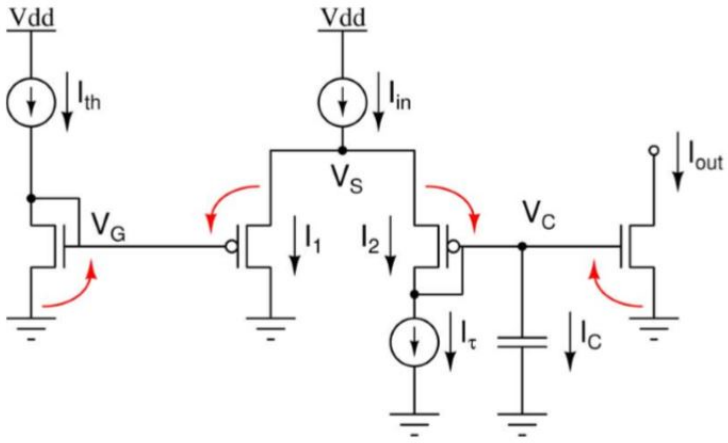

Figure 3.4: Schematic of a DPI circuit. Image taken from [7]

It can be demonstrated that, under the assumption of high input currents, and $I_{out} >> I_{th}$, eq. (3.1) can be simplified, obtaining eq. (3.2):

$$\tau \frac{d}{dt} I_{out} + I_{out} = \frac{I_{th}}{I_\tau} I_{th} \tag{3.2}$$

It is evident how eq. (3.2) implements a low pass filter of the input current. Moreover, the differential equation has a structure very similar to the one of equation in eq. (2.2) insection 2.2, representing the neuron I&F model. In other words, this circuit is perfect to emulate the reaction of neurons to input stimuli.

Neuron circuit

DPI circuit alone is not enough to create a complete neuron dynamics. It lacks, in fact, of spike and refractory period behaviour. To understand how the final neuron circuit can be obtained, we will proceed by steps. We will start from a simple DPI and, connecting from time to time new transistors, we will explain how the new version get closer to the real dynamic behaviour of the neuron, shown in fig. 3.8.

As initial step we want to introduce the capability to perform a spike. Taking as reference fig. 3.5, we add in series to the input DPI stage (highlighted in yellow), the red circuit, able to perform an action potential. Pay attention that the DPI circuit is a summarized version of the one present at fig. 3.4. The transistor connected to I_{th} is omitted, while the I_τ current generator is generated by transistor M_{L3}.
To understand the circuit behaviour we can look at figure Figure 3.8, representing the recorded membrane potential of a Dynap-se neuron, corresponding to the voltage V_{mem} in fig. 3.5. Each colored box represent in which time intervals the corresponding circuit went operational. The first part of the curve, indicated by the yellow square, is the initial filtering action done by the input DPI circuit. Then the SPIKE circuit get into action.

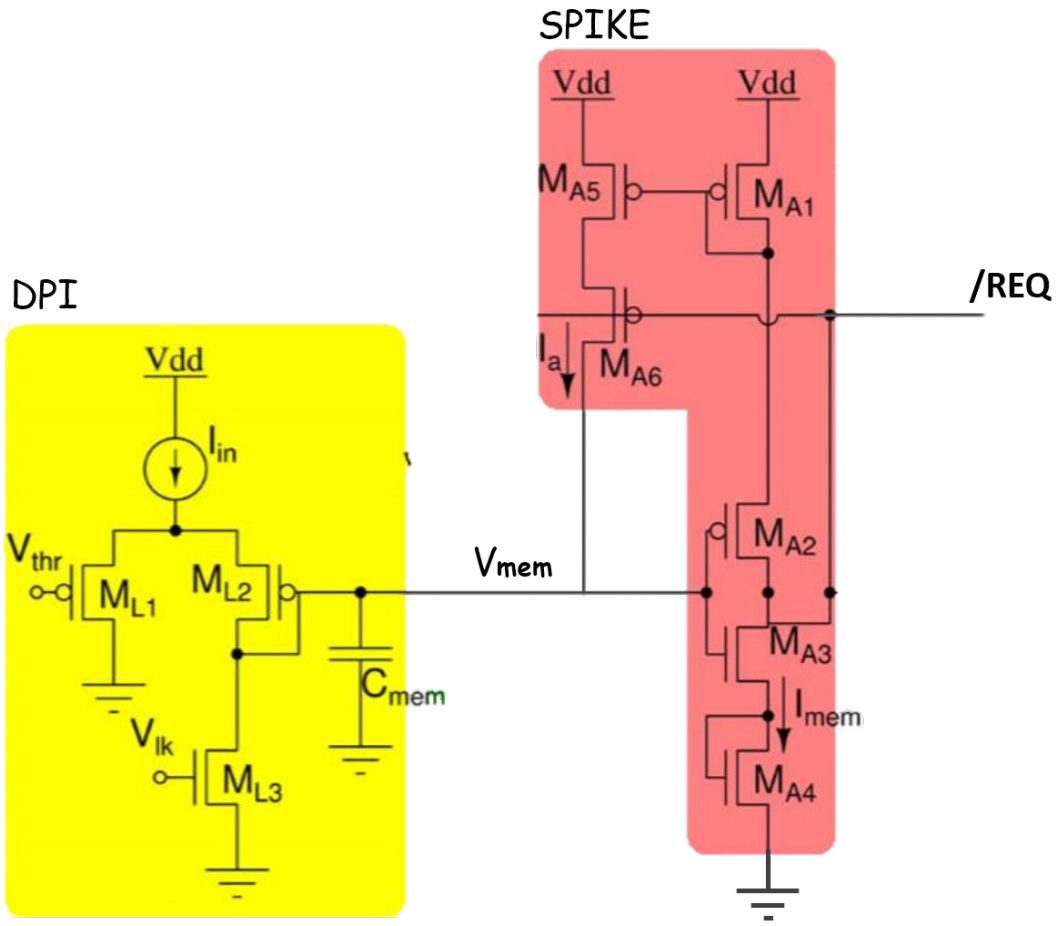

Figure 3.5: Partial schematic of the Dynap-se neuron circuit, containing input DPI and SPIKE dynamics circuits. Image from [7]

Looking at the schematic of fig. 3.5, by increasing V_{mem}, the same happens to the gate voltage M_{A3} and M_{A4}, leading to an increase of the current sunk from their drain. This will reflects to a decreasing of the potential of /REQ net. Since this last one is connected also to the gate of p-fet M_{A6}, it will increase its activity, pumping more current toward capacitor C_{mem}. This will lead to a further increasing of V_{mem}, and the cycle would repeat. Such mechanism, called positive feedback, will sharply raise the membrane voltage of the neuron and tide it to the maximum value, that we can call "high state". At the same exact time the digital signal /REQ is activated (going to zero logic state), sending a request toward the digital infrastructure present around the circuit. In simple words, the neuron is telling: "I have fired!" to the surrounding circuitry. This action is crucial, because it corresponds to the transformations from an analog spike to its digital counterpart.

Although this circuit, introducing the spiking dynamics, makes the neuron behaviour significantly closer to the biological one, is left to see how can be created a refractory period. Such dynamics can be obtained, as done before, by adding another block in series to the current neuron circuit, obtaining the structure present in fig. 3.6.

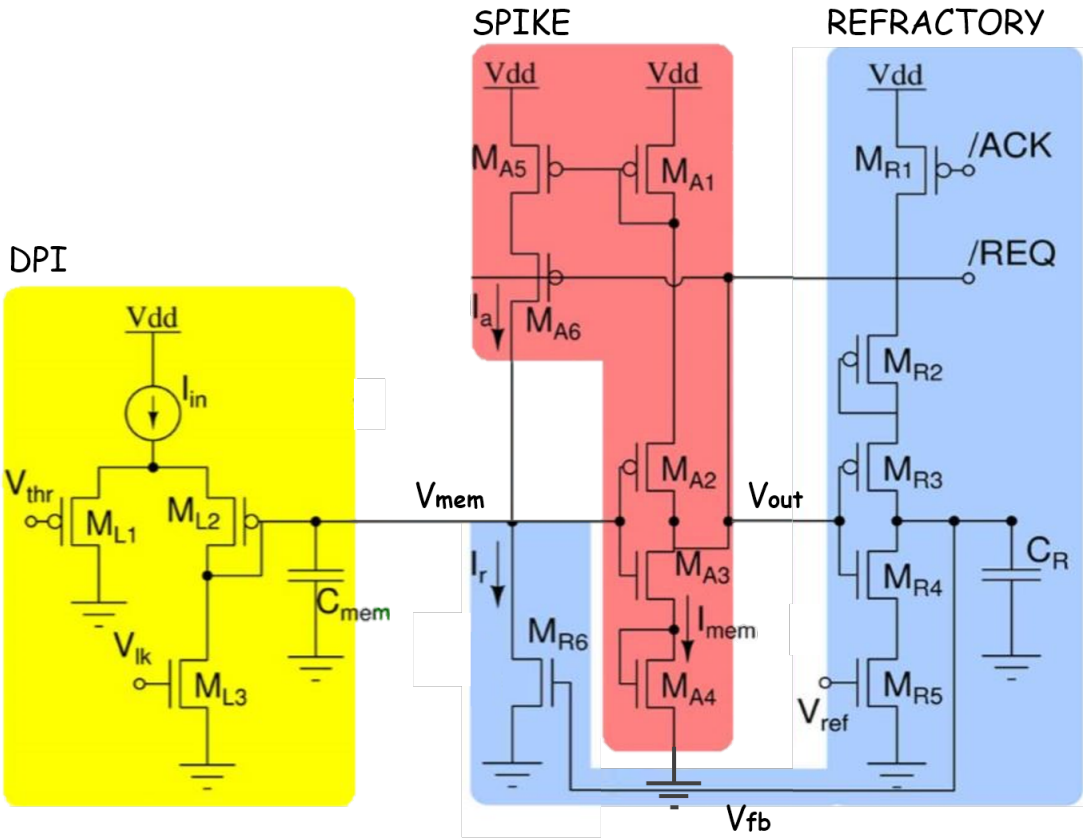

Figure 3.6: Partial schematic of Dynap-se neuron circuit, containing input DPI, SPIKE dynamics circuit and REFRACTORY period dynamics. Image from [7]

In the schematic, ACK is a digital signal controlled by the external digital logic. When not active, it at high logic state.

When the neuron spikes, voltage V_{mem} increases sharply while, at opposite, the voltage V_{out} at the gate of p-fet M_{R3} and M_{R2} decreases. These p-fets, in such conditions, would sink current from M_{R1} toward the capacitor C_R. However, since the ACK signal is normally not active (high logic state), this last transistor is closed and no current can pass through it. Such condition imply that, as long as ACK is not active, C_R capacitor cannot be charged, V_{fb} would remain at low logic state and, following the feedback loop, M_{R6} could not sink current from the membrane potential node. In other words, until the

external digital control does not acknowledge the neuron by setting ACK signal to low logic state, the neuron is in a stable "high state" condition. This behaviour ensures that no neuron spike can be missed by the external control.

When the event is acknowledged and ACK goes to low logic state, M_{R6} would sharply reset the neuron membrane potential V_{mem}, as shown on the blue section of fig. 3.8. In this period, every neuron input is cancelled by the action of M_{R6} transistor. The neuron is in refractory condition.

However, this condition does not last forever. With V_{mem} now at low level, and consequently V_{out} being in the opposite conditions, M_{R4} and M_{R5} transistors are now actively discharging capacitor C_R. This action would gradually inhibit transistor M_{R6}, making once again the neuron sensible to external inputs and putting an end to the refractory period.

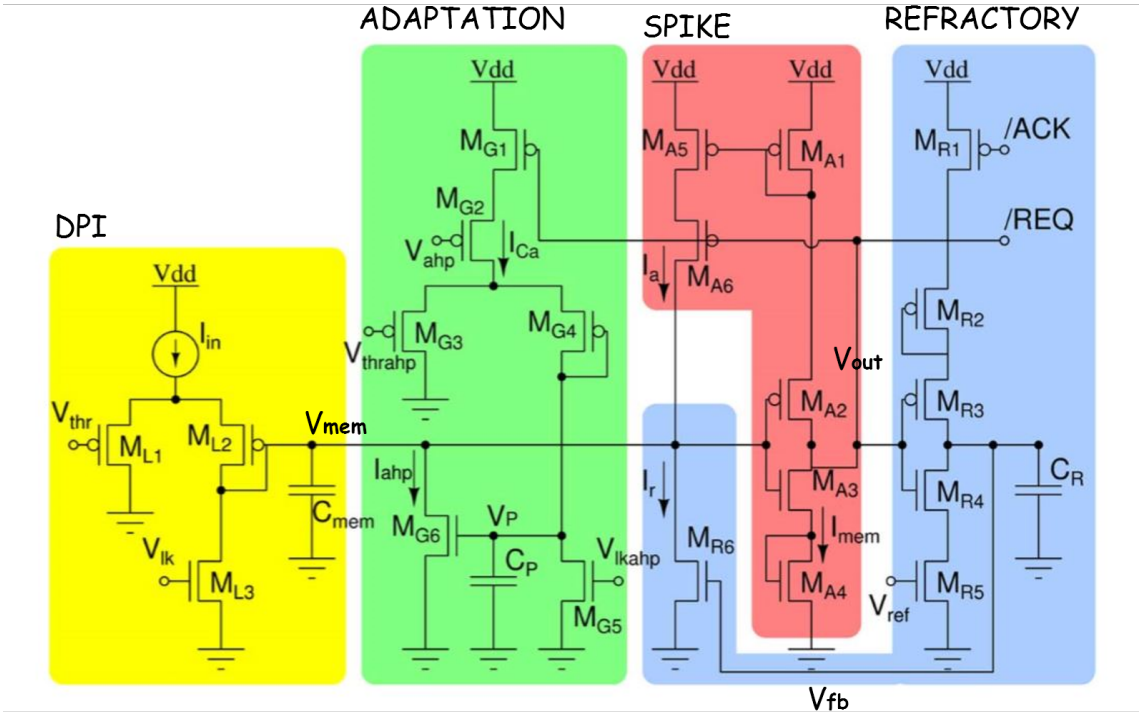

Figure 3.7: Complete schematic of the Dynap-se neuron circuit, containing input DPI, SPIKE dynamics circuit and REFRACTORY period dynamics and ADAPTATION block. Image from [7]

The complete Dynap-se neuron schematic, present in fig. 3.7, includes an additional circuit (the green one) modelling the spike frequency adaptation dynamics. This last one is a

phenomenon for which neurons reduce their spiking frequency, after an initial increase, when supplied with a constant input current. This complex dynamics can't be modelled if not with complicated circuits, that would require a long description. Since they are not fundamental in the normal usage of Dynap-se neurons, their analysis won't be covered in this chapter.

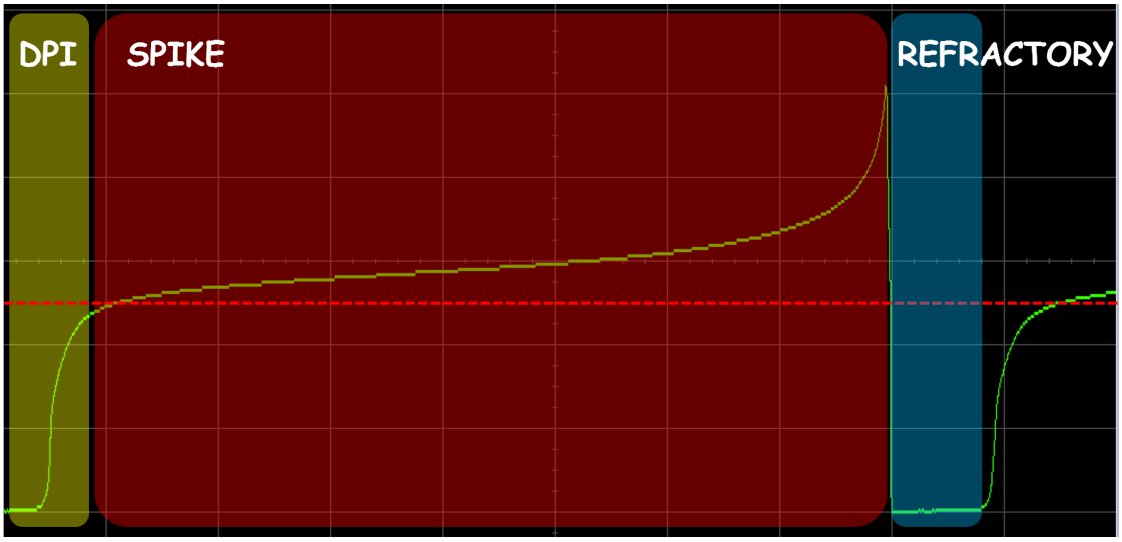

Figure 3.8: Oscilloscope recording of a Dynap-se neuron action potential in response to a input step current. The horizontal dotted red line represent the threshold of the silicon neuron

3.3.3 Silicon Synapses

In fig. 3.9 is depicted a schematic representation of the Dynap-se synapse circuit. Although the layout seems to be different from anything we have seen so far, its functioning is similar to the one expressed by the neuron DPI. Comparing the two circuits (fig. 3.9 for the synapse one with fig. 3.7 for the neuron one) we can notice that they are exactly complementary. Neuron input current I_{IN} is correspondent to synapse input current I_W, while M_{D1}, M_{D4} and M_{D5} are complementary to, respectively, M_{L1}, M_{L2} and M_{L3}. Capacitor C_{syn} works in the same way as C_{mem}, and output transistor M_{D6} is equivalent to M_{A3} and M_{A4}.

In other words, the synapse can be seen as a circuit that filters the input pulses, injecting or sinking the filtered current to/from the neuron. These input pulses, recalling the basic block description of fig. 3.3, are generated when a pre-synaptic event present itself at the input of the computational block. The voltage V_W regulates the gate potential of M_{D2}, making it behave as a "current tap" that regulates the influence of I_W on the rest of the circuit. In other words this parameter determines the synaptic weight value.
The general synapse equation, present in eq. (3.3), can be simply obtained using the same

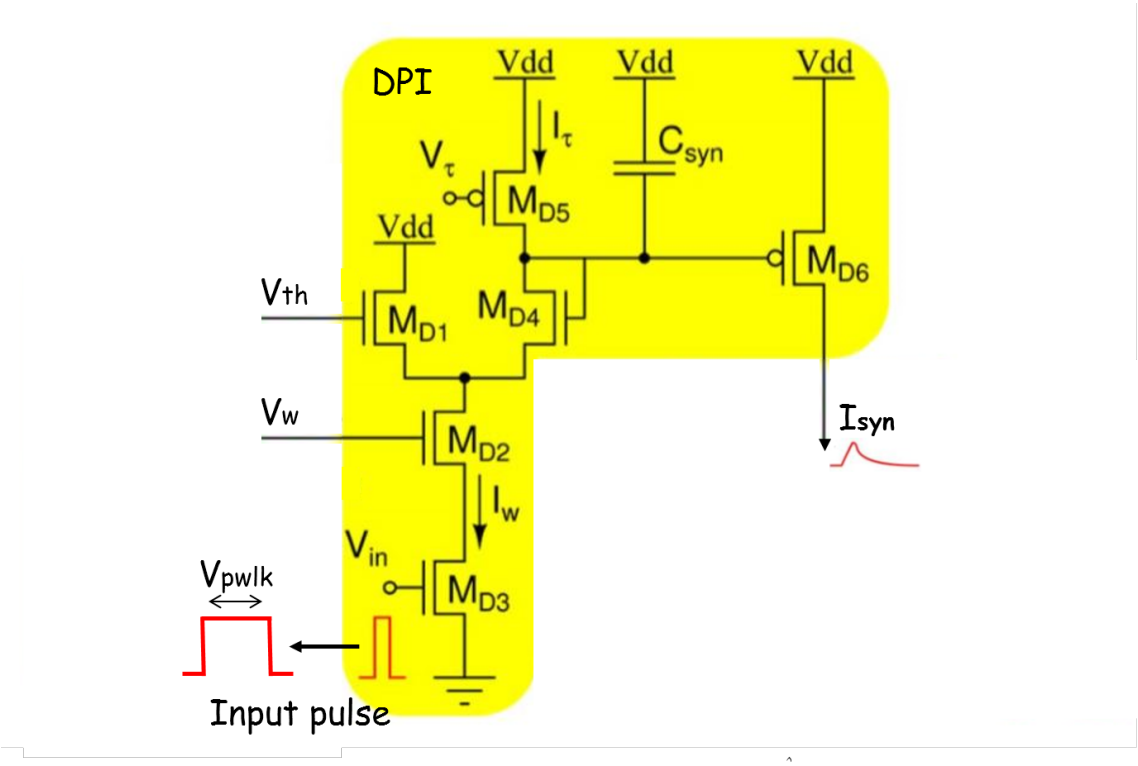

Figure 3.9: Simplified schematic of Dynap-se synapse circuit. Image from [7]

calculations done for the DPI circuit.

$$\tau_{syn}\frac{d}{dt}I_{syn} = I_{syn}(\frac{I_w}{I_\tau} + 1) \tag{3.3}$$

For each neuron there are 2 type of synapses: inhibitory and excitatory. The distinction is straightforward: the former ones sink current from the neuron, reducing (hyperpolarization) its membrane potential. The second ones, instead, inject current to the post-synaptic neuron, increasing its membrane voltage (depolarization).

Inhibitory and excitatory synapses are further classified as fast or slow. Although these terms would suggest a classification in function of synapse response speed, they lead to a misleading concept. In truth their difference is in the way they influence the membrane potential of the neuron:

- Fast excitatory and fast inhibitory synapses exploit the same dynamics, but with opposite results: the first ones inject current to the neuron while the second ones sink current from them.
- Slow excitatory synapses are similar to the fast ones, but includes the NMDA voltage gated channels dynamics. His explanation is outside the scope of this thesis.

- Slow inhibitory are similar to the fast ones, but stronger. In fact they drain current directly from the membrane potential of the neuron, avoiding the filtering action of the input neuron DPI.

3.3.4 Interconnections

In the first part of section 3.3 we have briefly described which is the structure and functioning of Dynap-se basic block. We haven't given a detailed explanation, but only information about the general context where neuron and synapses operates. In this section, instead, we will focus on two aspect of such system: the interconnection between neurons and which role CAM and SRAM cells play inside that mechanism.

In contrast to the standard digital systems, where memory is centralized, in neuromorphic chips it is spread uniformly across the neurons. In other words, every one of them owns a certain amount of memory which, in Dynap-se, is constituted by CAM and SRAM cells. Their purpose is to contribute to the creation of a flexible interconnecting system that would allow an efficient communication between silicon neurons.

We have seen, in section 2.1, that neurons communicate by producing an electrical output, called action potential. In the biological domain, the electrical spike travels inside the nervous system in order to bring information to the connected cells. In such network every connection between two different neurons is carried out by a private link and, although this approach would guarantee the maximum degree of flexibility, his implementation in hardware would not be feasible. In Dynap-se the interconnection problem has been overcome using a mix of hierarchical and mesh asynchronous routing strategies, with memory not concentrated in a single place but spread all over the computational blocks [6]. Since explaining such interconnections mechanism is not a trivial task, an analogy will be used, in order to make the concept as clear as possible.

In Dynap-se, connections work as the national post service. Lets suppose we want to send a letter to a friend that lives in another place of the country. We write our home address to let the receiver know where the letter arrives from and, of course, we write his home address, to let the post service know where the letter should be delivered. When the letter is sent, it passes through a hierarchical structure of shunting nodes: from sender post office to central storehouses, and then to destination post office. Neuron interconnections work approximately in the same way, but with few differences and constrains.

As every house has an address, the same happens inside Dynap-se. Every neuron has an unique 10 bit digital address that allows identification inside the whole chip population [8]. As every person has a list with friends house addresses, and a postbox to receive letters, neurons in Dynap-se have similar things: a "silicon address list" and "silicon postbox".

Neuron address list
It corresponds to the SRAM cells, 4 for every neuron. They contain routing informations

about the produced event destinations. In our analogy they could be, for instance, a sequence of post office or storehouses in which the letter must be passed through in order to reach the destination. Whenever the neuron generates a spike, if at least one SRAM cells is not empty, his corresponding digital event is routed toward the destinations written on the memory cells.

Neuron postbox
It corresponds to the CAM cells, 64 for each neuron. They contain information about at which event addresses the neuron should react to. In our analogy CAM cells could correspond to a "special" home mail box divided in 64 compartments, each one of them accepting only letters coming from a specific sender addresses. We can modify, for each compartment, which will be the valid one, or even assign none. A delivery would be inserted only if the sender address is present in one of the compartments.
For neurons the same happens. Whenever an event arrives, sender address is compared with the one contained in the neuron CAM cells. If a match occurs, an output pulse, directed toward the synapse, is produced. If multiple addresses matches, the pulse is scaled up accordingly.

3.4 DYNAP-se experimentation board

In section 3.3 we have described which are the structure, characteristics and working principle of Dynap-se. The reader, at this point, can easily realize that the chip contains all the elements needed to be independently functioning: it has a computation engine (neurons), an internal communication infrastructure and an input output interface. Besides the power supply, that is produced externally, it can operate and communicate with other chips without the need of any additional component.

However, when an user want to interact with the chip, it needs an infrastructure that can adapt the asynchronous analog world of Dynap-se to the synchronous digital world at which computers belongs to. The solution to this problem is Dynap-se experimentation board, an electronic device that allows the control and configuration of multiple Dynap-se chip.

Dynap-se experimentation board, visible in fig. 3.10, is an electronic device that has the resemblance of a blue box. It is powered by 4 Dynap-se chips wired together, for a total of 4096 neurons and 256k synapses. Its input/output ports, visible in fig. 3.11, allows the communication with other external devices:

- Personal Computers, by the means of a USB2.0 interface
- Event-based cameras, such as DVS (Dynamic Vision Sensor), through the 40 pin PAER/GPIO interface
- Up to 8 other Dynap-se experimentation boards, using the internal AER connectors.

Figure 3.10: Dynap-se neuromorphic processor version 1. Image courtesy of aiCTX company, Zurich

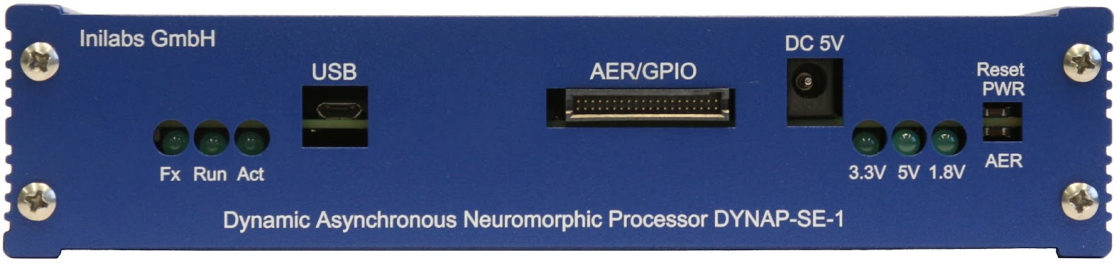

Figure 3.11: Dynap-se rear panel. It includes USB2.0 interface, 40 pin AER/GPIO, power supply input port, power and service leds. Image courtesy of aiCTX company, Zurich

> This can be used to scale up the complexity of the neural systems that can be implemented.

It is possible, in addition, to monitor the membrane potential of a maximum of 16 neurons at the same time, through the analog monitor connectors, depicted in fig. 3.12. An example of such visualization is present in fig. 3.8.

Connections between multiple boards can be performed with simple parallel cables, connecting together the AER Expansion slots present in the boards, visible in fig. 3.13. Such links do not need any additional hardware or software, because Dynap-se chips contain all the logic to handle the communication between each other.

Figure 3.12: Dynap-se front panel. The 16 yellow connectors are analog monitor outputs that, connected to an oscilloscope, allow the visualization up to 16 neuron membrane potential. Image courtesy of aiCTX company, Zurich

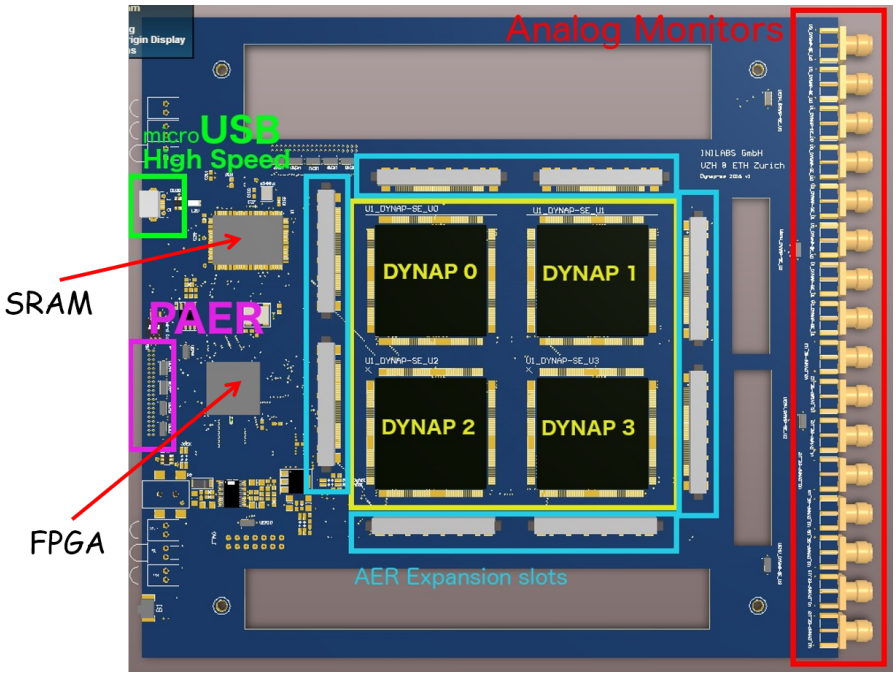

Figure 3.13: Dynap-se detailed internal structure (top view). The FPGA and SRAM create an interface between the chip and the micro USB, as well as with the PAER connector. Image courtesy of aiCTX company, Zurich

However, additional components are required to interface the chips with the input/output digital ports: a Xilinx Spartan 6 connected to an external SRAM memory, visible in fig. 3.13.

The first is an FPGA whose job is to convert the input data stream coming from the micro USB to the protocol understandable to Dynap-se, and vice versa. In this way the user can control and configure the chips directly from a PC terminal, viewing at the same time the spiking activity of the neurons.

The second one is an additional backup of memory that the user can freely use in order to perform on board analysis. For instance, it can store a customized event pattern that the FPGA can use to stimulate Dynap-se chips.

Chapter 4

Reservoir computing and LSM

> "The cognitive era requires novel architectures to accelerate and efficiently execute new learning algorithms"
>
> IBM scientist Stefan Abel

Recurrent neural networks have always been an important concept in neuroinformatics, mainly thanks to their outstanding computational properties. A stable recurrent activity could give rise to significant phenomena of dynamic memory, making possible the classification of signals in the temporal context. In this chapter we will explore Reservoir computing, an efficient training method that can be applied to recurrent neural networks in order to make them solve specific tasks. We will then describe the properties of its spiking neural network implementation, called Liquid State Machine (LSM).

4.1 Learning in neural networks and Reservoir computing method

In chapter 2 we have talked about the biological behaviour of neurons and synapses, as well as the meaning of neural networks, how they can be constructed and which are the main criterion for their classification. Although networks of neurons can actively react to stimuli and extract information from them, they can't solve problems by themselves. This can be accomplished only introducing a fundamental phenomenon in neuroscience: learning.

In the same chapter we have seen that, in the biological world, learning is accomplished by the so called synaptic plasticity: synapses modify their properties, in particular the so called synaptic weight, such to increase or decrease the sensibility of the post-synaptic neuron to the subjected input. In other words, every connection between neurons tune itself in order to solve a particular task.

When we shift from the biological world to the artificial one, for obvious reasons, we can't apply the same concept. Artificial learning is something that must be implemented as an algorithm. The biological mechanism is transformed to a sequence of updates of the synaptic weight, where an increment of its value for one specific input it makes the post synaptic neuron more sensible to that, and vice versa.

Since learning is so important, many studies have been carried on to determine the right sequence of steps that, according to the task, would return the optimal set of weights able to solve it. One of the most common ways to perform it is by using the so called gradient-descent based methods. Such algorithms are able to find the particular set of weights minimizing the error between the neural network response and the target one. However, although these methods are very powerful, they work well for feed forward neural networks, but many problems occur when trying to apply them to recurrent ones, for which we won't enter in detail. In other words, the complexity of the RNN training made them very difficult to use for real application.

In order to overcome these problems, a new algorithm has been developed, called Reservoir computing. The "version" developed for Artificial neural networks was discovered by Herbert jaegeris, and is called ESN (Echo state Network). The one for Spiking neural networks was discovered by Wolfgang Maas, and is called LSM (Liquid state machine).

4.1.1 Common RNN implementation

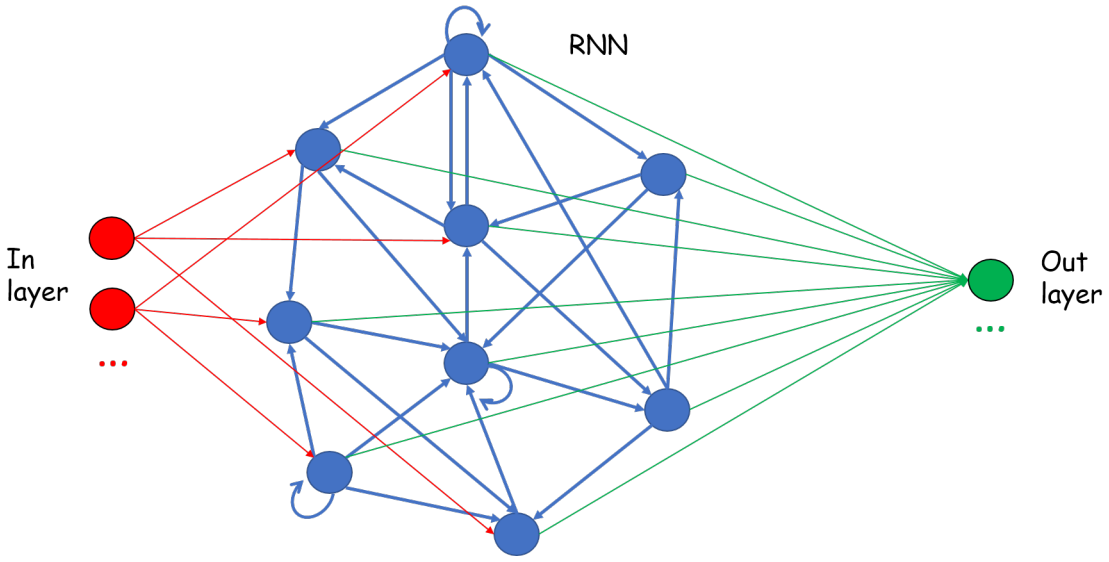

Figure 4.1: Schematic example of Recurrent Neural Network implementation

Figure 4.1 shows the common structure in which recurrent neural networks are organized.

An input layer of neurons, highlighted in red, stimulates a recurrent neural network, depicted in blue. In general, the connectivity between these two populations is very sparse, such that only a small fraction of all the possible connections are effectively done. Their main goal, in fact, is to inject inputs in few sparse point inside the pool of neurons, without being predominant over the dynamics of this last one.

The computation engine of the system resides in the recurrently connected neurons: information is received from the input layer and transmitted recursively forth and back between the different parts of the network. Such dynamics is also the one responsible of memory: the past stimuli, trapped inside the pool, influence present and future states of the network, until their memory does not fade.

Finally, pool activity is withdrawn and projected to the output layer. This last one is usually composed by few neurons, depending on the type of classification sought, and perform an all to all connection with neurons on the RNN layer. It will be up to the training algorithm, by setting the right weights, define which connections are important and which not, in order to correctly perform the task.

4.1.2 How Reservoir computing works?

The revolution of Reservoir computing resides on the simplicity of the training algorithm. Figure 4.2 explains clearly its difference with respect to the normal approach. In the left we can see a standard training applied on an RNN: the algorithm learn, for all the connections of the network, the set of weights that may accomplish a specified task. The main

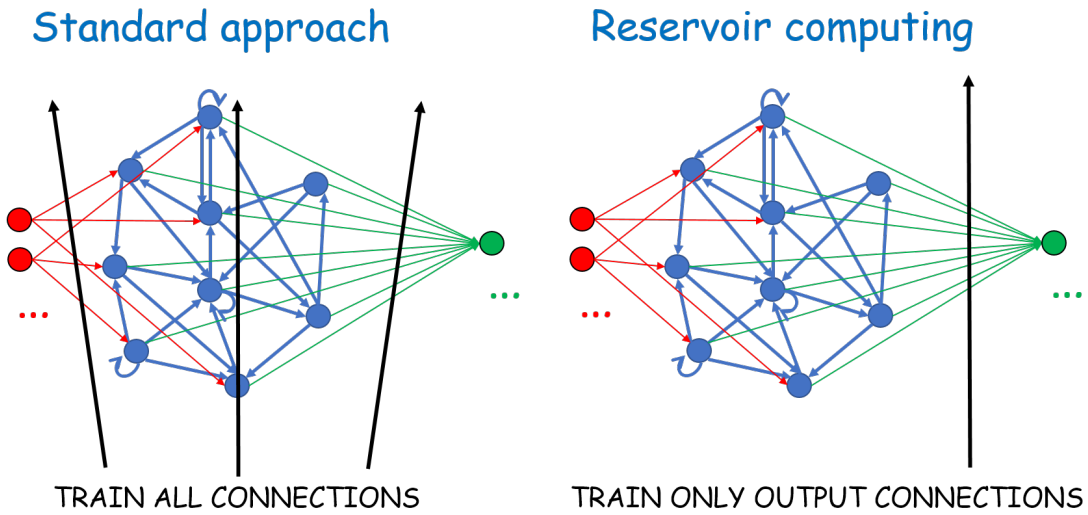

Figure 4.2: Comparison between RNN standard training approach and Reservoir computing one

problem related in such approach resides in the fact that no standard learning algorithm

can guarantee convergence, because all suffer of the so called vanishing gradient problem. In simple words, the weights updates become so small that they don't change at all, even if the training is not finished and the network is still unable to perform correctly the task. Such problem, very common in RNN, does not take place on feedforward neural networks.

Here, on this last statements, came the idea: researchers discovered that it was not necessary to train every single weight in the network in order to make it perform correctly the task. Instead of applying a global training it is sufficient, as shown in the right image of fig. 4.2, to train only the output layer connections that, being in a feed-forward configuration, do not suffer of the vanishing gradient problem.

4.2 LSM

The mathematical concept of LSM, acronym that stays for Liquid state machine, was proposed by Wolfgang Maas in 2002 [9]. Its definition is not valid only inside the world of neural networks, but is a general idea that can be extended to everything. Since its formal meaning is not easy to understand, we will explain the concept using an analogy.

Lets suppose it's a beautiful summer day and we go to the lake with some friends. There is no wind and the water is still. We decide to play a game: one guy throw a stone on the lake while the other one, initially looking the other way, has to turns toward the scene and evaluate the position where the stone fell. Such game is possible only because the water around the hitting point reacted to the initial stimulus for some time after the rock has sunk. In other words, the system "water in the lake" has, at every time, memory of the past inputs; such intrinsic reserve of information can be used to make predictions. In the example of the stone in the lake, we can guess the approximative stone hitting position or time, just by looking at the activity of the water.

Having clear this example, LSM should be easier to understand. A liquid state machine can be simply defined as a computation model for the analysis of dynamical behaviours, requiring the presence of two elements:

A liquid filter, which is an object that must be capable of reacting to present and past perturbations coming from the external environment. The response that it generates, reacting at such stimuli, is called liquid state.

A readout map, which is an object that must be capable to elaborate the dynamics of the liquid state, in order to give an output prediction.

Since the definition of LSM does not give any specific application context, it can be represented by anything that fits the definition. In our example, the liquid filter could be thought of as the water in the lake, the perturbation as the guy throwing stones, the readout map as the guy that has to make the guess only looking at the perturbations on

the water, which could represent the liquid state.

If we want to find an LSM correspondence in the neural network context, we should find which elements can satisfy its definition. The liquid filter can be implemented by a network of neurons, obtaining the so called "Reservoir of liquid neurons". However, not all the networks topologies match this definition correctly: feedforward ones, for example, are not able to maintain memory of the past, hence they could not represent a proper liquid filter. Recurrent neural networks, instead, are excellent candidates for that role. The liquid state, in such example, would correspond to the spiking activity of the Reservoir of neurons.
The perturbation, in this context, would correspond to the incoming spike trains showing themselves at the input of the Reservoir.
The readout layer, finally, should be made by a network of neurons producing instant elaborations of the liquid state. Such task can be implemented by a simple feedforward neural network.

A neural structure that can implement a liquid state machine is the one shown in fig. 4.3. It can be noticed that the depicted structure is very similar to the one shown in fig. 4.2,

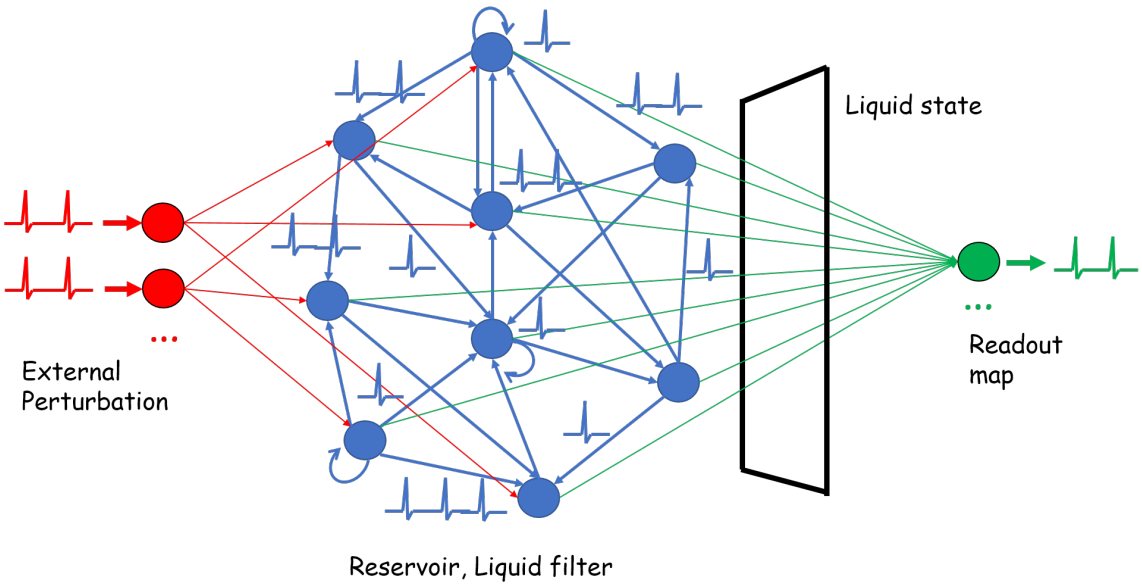

Figure 4.3: LSM implementation with spiking neural networks

when have talked about reservoir computing. In fact, LSM concept applied to spiking neural networks is a version of Reservoir computing. In other words, the working principle of the spiking neural network present in fig. 4.3 is equal to the one described for reservoir computing. All the connections between input and the reservoir, included the recurrent ones, does not need to be trained. The training must be only applied to the Readout map in order to force the output neuron activity be the desired one.

Chapter 5

Biomedical signals

5.1 What are biomedical signals?

Every day our hearth pumps blood throughout the body, our lungs enrich it with oxygen and our nervous system control all the voluntary and involuntary actions that we perform. Even if we can't really feel it, because most of what happens in our body is automatic and outside our control, these physiological processes have very complex dynamics. Maintaining the body alive and active is not an easy task. Taking in consideration the complexity of such processes, how can we diagnostic possible alterations on their activity, probably related by disorders in the body?

Luckily, most of these biological processes comes along with characteristics signals, whose dynamics is strictly related to the state of the process. We can classify them as:

- biochemical signals: hormones and neurotransmitters levels
- physical signals: the most common are temperature and pressure
- electrical signals: currents and potentials

Their dynamics encodes information about the body health state, therefore they can be used for disease diagnosis. In this thesis the attention is focused on electrical signals, in particular on ECG one, for which it will follow a brief characterization.

5.2 ECG signal

The reader may know from experience that, in order to get a sports medical certificate, people are requested to fulfil a physical examinations which determines if the patient conditions are suitable for sports or not. In such situation, one of the exams that people are expected to do, is the analysis of the heart signal. The procedure provides that the patient, laying still on a bed, should have specific areas of the body connected to a measurement machine, by the means of electrodes. Such device performs the so called ECG (Electrocardiography), the recording of the heart activity over a certain window of time. The resulting

signal is then analysed by doctors that, with experienced eyes, can tell whether there are issues or not. But where does ECG signal comes from?

Our heart is a biological pump made of four alcoves, "holes" were blood collects and then is pumped out. It works with the same principle as filling and emptying a balloon of water. An heart beat is not a single instantaneous phenomenon, but involves a pattern of contractions and relaxations of the different chambers, with precise locations and time intervals. All heart muscle activity is controlled by electrical stimuli coming from the autonomous nervous system, which produces a potential difference strong enough to be measured from different parts of the body. Without getting into detail about the different techniques of measuring such potential, ECG signal has the characteristic shape depicted in fig. 5.1.

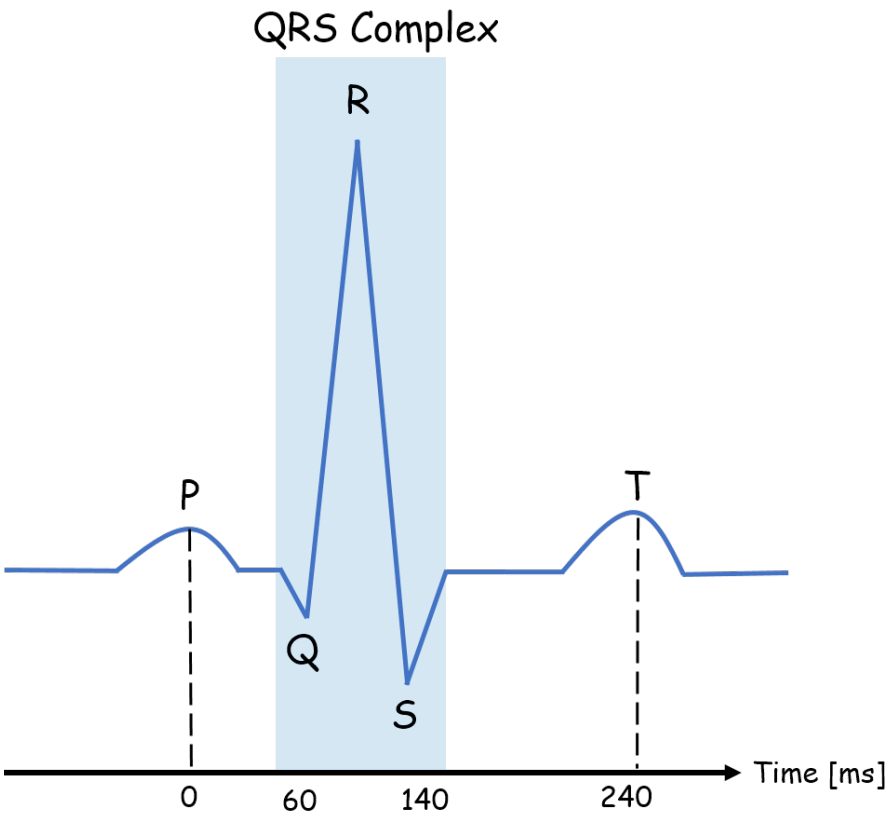

Figure 5.1: Visual description of ECG signals shape

Some sections are clearly distinguishable. Starting from the left, we have:

- The P wave, the first small bump of 0.1, 0.2 mV voltage and duration from 60 to 80 ms
- The P-Q segment, where the signal stays stable for other 60 to 80 ms.

- The QRS complex, where the signal has a sharp raise and fall behaviour, of 80 ms duration and 1 mV amplitude.
- The ST segment, following the QRS complex, lasting from 100 to 120 ms.
- The T wave, a bump similar to the P wave, but with an amplitude that can go from 0.1 to 0.3 mV and a longer duration, from 120 to 160 ms.

Every one of these sections can be linked to a particular operation that occurs inside the hearth as, for instance, the blood transfer from one alcove to another one. A perturbation acting on that normal sequence of events it would reflect immediately on the ECG signal dynamics. Possible variation could be the temporal broadening/narrowing of section intervals, the amplification/attenuation of the different ECG signal bumps or their complete distortion.

Since perturbations on the normal heart activity are usually associated with diseases, here comes the real importance of being able to classify between normal or abnormal ECG signals.

Chapter 6

Methods

6.1 DYNAP-se control software cAER

In section 3.4 we have described the structure of Dynap-se development board, explaining how it can be interfaced with a PC terminal thanks to the operation of the internal FPGA. However, while we previously focused more on the board side of the communication, in this section we will give a brief introduction of how it is handled on computer side.

Dynap-se board can be controlled from PC terminal using a stack up of two software drivers, called respectively libcAER and cAER. A visual overview of the whole control system is depicted in fig. 6.1

libcAER is the kernel of Dynap-se control software. Written in C language, is the low-level software library that handles the communication between Dynap-se board FPGA and the PC side. Although it guarantees high control and flexibility over the device, the complexity of the code required to execute even basic operations is significant. Besides this fact, the user needs good knowledge of Dynap-se device structure and characteristics in order to design an analysis using such software. For such reasons another layer of abstraction has been interposed between the user and the hardware, giving rise at cAER software.

cAER is the high-level software build over libcAER, written in C and C++ languages. The core of the program is constituted by modules, basic blocks that exploit specific operations over Dynap-se board, as event visualization, network implementation, stimuli generation, etc. The user, from command line or using a graphical interface, can manually configure the parameters of every module, whose changes reflect on the functioning of the device, as change neuron and synapse biases, select neuron time constant, write SRAM and CAM cells, etc. In other words, the code required to perform operations in Dynap-se is already written in side modules, therefore the user can perform analysis only by configuring and combining them together.

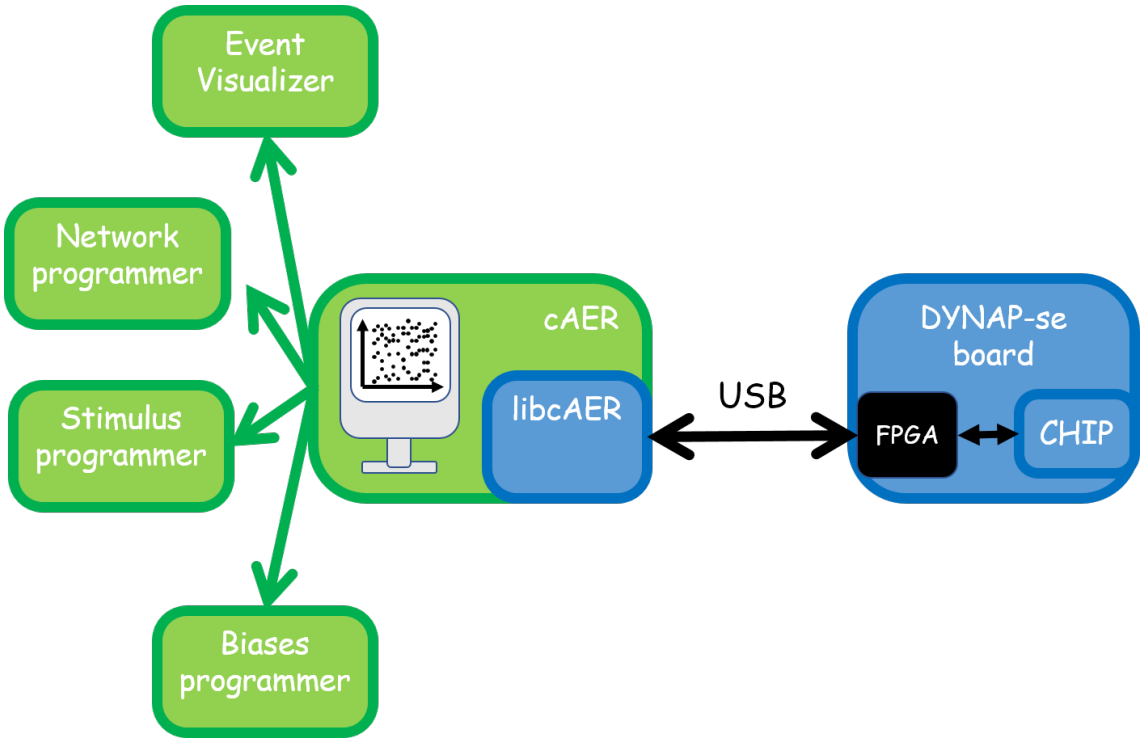

Figure 6.1: Schematic view of how is performed Dynap-se chip control on software and hardware

6.2 Development of software to simplify elaboration with Dynap-se board

A great chunk of the thesis work relates to the development of a software structure able to simplify Dynapse-se board programming effort. In fact, we can compare writing software for Dynap-se as coding in low level language. Such approach clearly allows high control over the device, but at the same time imposes huge limitations on flexibility and increases the complexity of the code, even for simple operations. As low level language is the foundations holding up a tower of software abstraction layers, the first goal of the thesis work has been directed toward doing something similar for Dynap-se software. In other words, trying to hide as much as possible the complexity of the device structure from the final user. The resulting work consisted on the deveopment of DYNAPSETools, an open source python library that interfaces with Dynapse-se control software and simplify the use of the device.

In this section we will describe the characteristics of the work that has been carried on. First, we will explain what is the meaning of "operating with Dynap-se board" and which type of analysis can be done. We will then focus the attention on how these operations were performed with the previous approach, and discuss how they has been simplified and

improved in DYNAPSETools library.

6.2.1 Standard processing flow with Dynap-se board

Described in detail in section 3.4, Dynap-se is an experimental board whose main purpose is for testing the behaviour of neural networks at hardware level. In other words, it makes possible move from theoretical ideas of computational neuroscience directly to physical implementations, in order to build a working proof of concept. The main process flow, when working with Dynap-se, can be divided in the following steps: network generation, input stimuli definition, and network activity analysis. A summary of such process is present in fig. 6.2.

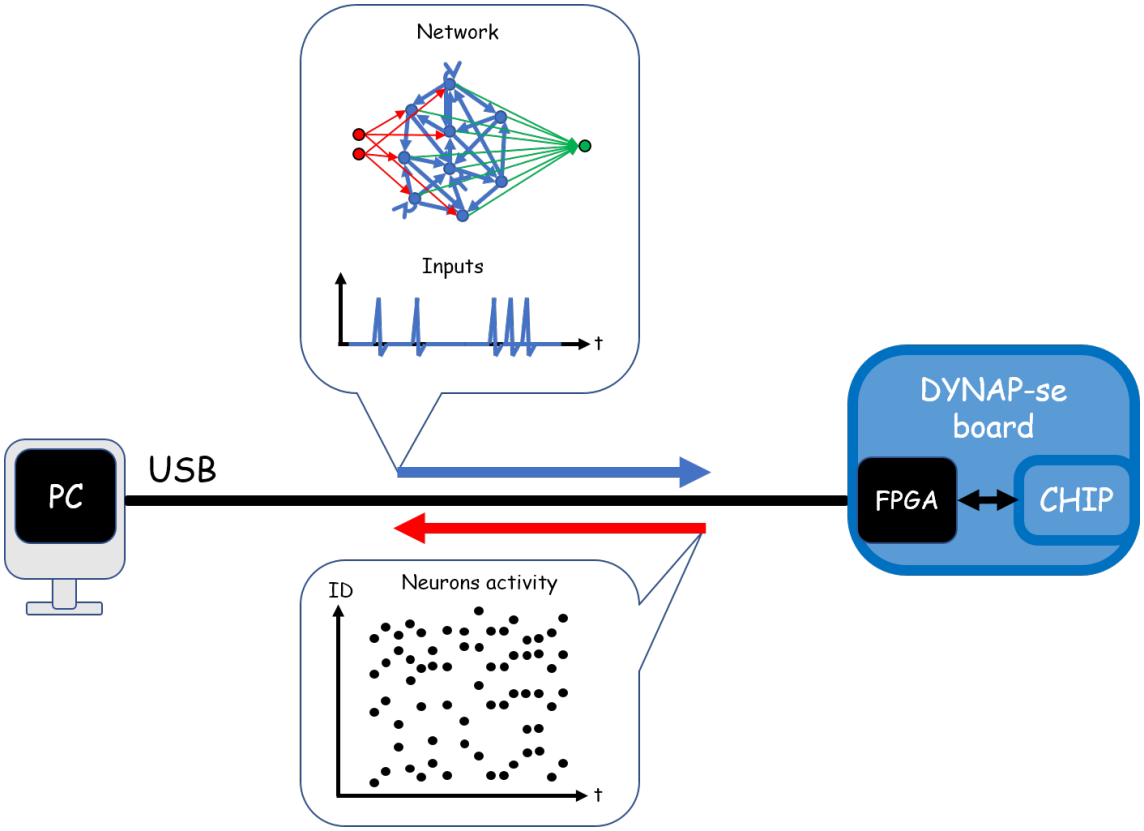

Figure 6.2: Schematic explanation of standard processing flow with Dynap-se board

Network generation

The first step consists on the creation of the neural network that need to be analysed. More in detail, a network is defined by specifying its topology, the characteristics of its populations, and the properties of the interconnections between them. In Dynap-se there is an additional factor to take into account: neurons are real entity that have a certain

location in the device. Therefore a population is not completely defined only by his properties, but needs to be bounded to a specific position in the neuron grid. The final goal of this step is to obtain a sequence of virtual connections that, downloaded to the device, translates into physical ones between silicon neurons.

Input stimuli definition
Once the networks has been completely defined, it is necessary to specify, according to the type of analysis we want to accomplish, its input stimuli. For instance, if our aim is to study the behaviour of specific interconnection properties, we must define all the input patterns that will stimulate what we are interested to analyse. Such patterns will be then loaded into Dynap-se board and transformed into physical stimuli for our network, as an input current or input spikes.

Network activity analysis
This is the most complex step. Once the device is programmed with the network that we want to study and its input stimuli, the real analysis begins. Dynap-se board provides a set of features that guarantee a strong interaction with it: the user can visualise in real time the activity of all internal neurons and access to the membrane potential voltage of a limited amount of them. Populations activity can be recorded inside a compressed file and played back at a later stage. From these informations the user can have feedbacks on the network dynamics and has the opportunity to correct his behaviour by changing manually or automatically neurons and synapses dynamics, tuning their 25 characteristics parameters.

Analysis classification
What has been addressed in the previous paragraphs are the general guide lines on how analysis with Dynap-se are performed. However, the presented work flow can be realised in two different ways, what we call Offline mode" and "Online mode".

In the first one, the adjective offline indicates that the analysis is more focused on the pre processing and post processing of Dynap-se data. In other words, the time spent designing the analysis performed by Dynap-se control software is much less than the pre and post processing of the data. The first phase consist on the preparation and configuration of input stimuli, network architecture and neuron parameters, as stated in the previous section. In the subsequent step, Dynap-se control software is used to program and stimulate the board, recording the internal neuron activity in outputs files. Finally, on the post processing phase such recordings are imported, and the required elaborations are applied.

Online analysis, instead, gives to the user more functionalities. Network connectivity and input/output elaboration can be done, with certain limitations, in real time. In other words, the pre and post processing time give way to the analysis performed by the control software that, for this reason, increase dramatically his complexity. Although online mode

gives high flexibility, for example allowing parameter driven analysis or real time elaborations of Dynap-se outputs, requires in-depth knowledge of the device and Dynap-se low level coding paradigms.

6.2.2 Creating networks for Dynap-se

As described in section 6.2.1, the first step of Dynap-se analysis flow consists on the design and implementation of the neural network that we want to emulate. Such task can be achieved by writing, for each of the connections in the network, the proper addresses inside SRAM and CAM cells of source and destination neurons. The standard approach provides for the memory programming using the low level functions included in Dynap-se software libraries.

For small networks such approach may still be reasonable, but it becomes easily unusable when their size start growing. The main disadvantages lie in the poor flexibility and the high difficulty of debugging possible errors. In addition, there is another fact. Taking as reference section 3.3, we know that SRAM and CAM cells are programmed with a different purpose: while the first ones contains destination addresses, the second contains the ones at which the neuron is sensible. Such consideration imply that, for every connection, at least one CAM cell must always be written, while the story is different for SRAM cells. For 1, 10 or even connections, if some requirements are satisfied, 1 SRAM cell is enough to encode all the connections.

Firstly, this fact explains why the amount of available SRAM cells per neuron is much lower than the one for CAM cells. Secondly, highlights the necessity of a real connection manager able to abstract the user from the interconnecting problem. It should take care of translating high level connection formats to low level CAM/SRAM implementation, maintaining consistency on the two representations.

Working in collaboration with Rodrigo Ceballos Lentini, a master student of INI department, we developed a software module for cAER control software, called Netparser.

Netparser module
Netparser module is a component that can be added to cAER control software in order to improve his functionalities. This module is a neural network manager. It mainly help the user in two aspects:

- It abstracts away all the complexity of Dynap-se connectivity, avoiding manually setting the SRAM and CAM registers to connect neurons together.
- It keeps a map of all connections that has been done on the device, checking for conflicts when new connections are inserted.

Connections are listed in a .txt file, one after the other, with a user friendly format that requests the specification of source, destination and few connections parameters: for each

one of them, in fact, is possible to choose the type of synapses that must be used and the strength of the link. The complexity of the interconnection system, in this way, is totally hidden to the user.

The module, in addition, maintains an updated map of all the connections that has been made inside the board. Such feature gives the module the ability to perform controls on the new connections and raise errors if conflicting conditions occurs, for instance CAM and SRAM overflow or CAM clash situations.

DynapseNetGenerator

Although Netparser module brings the user away from the low level Dynap-se board programming, he is still requested to write instructions that map the network into a list of connections between neurons. The resulting code would be, other than poorly readable, not flexible, because tied to the physical implementation of the network.

For this reasons we have developed DynapseNetGenerator, a python module of DYNAPSE-Tools libray able to further increase the abstraction level between the hardware and the user, reducing the complexity needed to program the board.

The way the network is programmed changes: its is not seen anymore as a sequence of connections, but as a collection of interconnected populations. A population is a cluster of neurons that belongs logically to the same group. Following this new approach, a network can be defined in three steps:

In the first one the user splits the network in different populations, specifying for each one of them the physical location in which they will be placed inside Dynap-se chip.

In the second step the populations neurons are virtually connected together using high level python instructions, without caring about how they will be physically implemented.

In the third one the connections between the virtual populations are automatically converted into the equivalent physical ones, and written on a .txt document. Such file can be imported by Netparser module and finally loaded inside Dynap-se board.

Even if the procedure seems long an complex, it solves the problems we have introduced in Netparser module section. The population approach obliges the user to a logic division of the network structure, improving code readability. In addition, the automatic conversion of virtual connections into physical ones completely move away the user from their real implementation. Every change in the network structure would be easily reflected in the physical implementation, requiring no code changes and dramatically improving the flexibility.

6.2.3 Creating input stimuli for Dynap-se

Once the network has been correctly programmed, the next step consists on the design of the input stimuli. These last ones are similar to the digital events produced by neurons spiking activity, but they are completely programmable in time and destination. In simple words, handling the input stimuli is like controlling the dynamics of a "virtual"neuron, deciding when he has to spike and at which destination. In truth, the key of these stimulation process is the FPGA present on Dynap-se board. Correctly programmed, it can send to the chip customly designed digital events that, accepted by the internal interconnecting system, are routed toward the neurons we want to stimulate. FPGA configuration can be done in two ways:

A first approach consists on sending the stimuli directly from PC to FPGA, through the USB interface. The FPGA would then take care of their redirection toward Dynap-se chip. The main disadvantage of such method is the low temporal resolution given by the limitations of USB link. With the second approach, we separate the method in two parts: at first we send the stimuli to the FPGA and save them into the on board SRAM memory. The stored events contains not only informations about the source and destination addresses, but also a timestamp. When correctly triggered, the FPGA would read the content of the SRAM and route every stimulus toward Dynap-se chip, with a delay indicated by the event timestamp. The advantage of this two-step approach is the high temporal resolution, since inputs are taken directly from board memory and don't need to pass through the USB port.

This second approach is the one used for the analysis carried on in this thesis, and for which DYNAPSETools library has been designed to operate with.

FPGA Spike generator module
DYNAP-se control software cAER contains an useful module called FPGASpikeGenerator. This module allows the user to program the FPGA in order to send custom events patterns to DYNAP-se chip. Such patterns can be realised by preparing a .txt file were each line encodes the information needed for one stimulus. Each one of them is composed by 2 arguments, separated by a comma:

- Source Neuron Address: Encodes informations about the source and destination of the event. It is used by the internal Dynap-se logic to correctly route the stimulus.
- Interspike Interval : Value representing the time between an event and his previous one, expressed in multiples of a parameter called ISI base. Changing this last one we can change the temporal resolution of the generated stimuli.

An example of such stimulus encoding is depicted in the code reported below. In this example the resolution is 1 us time step (ISI Base = 90).

```
79, 20000
79, 20000
79, 10000
79, 10000
142, 20000
142, 20000
```

The code file will generate 2 events with 20 ms of interspike interval (corresponding to 50Hz) with encoded destination address equal to 79, then 2 events with 10 ms delay between them, finally 2 events on encoded destination 142 with 20 ms delay.

DynapseSpikesGenerator
It is evident from the previous code that the manual stimuli generation is not an easy task. For this reason we have developed DynapseSpikesGenerator, a python module that allows the user to create customised event patterns. Instead of manually writing the .txt file, the user has the possibility to use high level objects, abstracting from the complexity of the whole process.

The main key of DynapseSpikesGenerator module stays on the InputPattern class. Every InputPattern can be programmed in order to contains a sequence of events, from one to many. Moreover, they can be combined together in order to create more complex stimuli, in a way similar to the use of LEGO blocks.

The library allows the user high flexibility, providing a set of different functions to create a customize stimuli pattern:

- Create patterns from user defined lists containing customized event times and addresses
- Create constant frequency events, as well as linear frequency modulation
- Encode an input signal in spikes with threshold encoding methodology
- Plot generated spike patterns
- Write output .txt file containing coded events

6.2.4 How analyse output results

The arguments seen so far, neural networks and input stimuli design, are part of the pre-processing phase that should be completed before working with Dynap-se board. As described in section 6.2.1, all the analysis performed in the thesys work uses the Offline approach, which involves saving Dynap-se activity on an output file, and elaborate the results in a second time, doing what we call post-processing. In this section we will describe how the developed software provides the user with functions that simplify the design of post-processing data scripts.

Output module
cAER control software contains a module called OutputFile, which collect events from Dynap-se board and store them in a compressed file format, named .AEDAT. In detail, an .AEDAT file is a collection of packets, each one divided in two parts:

- Header: Contains informations about the number of events present in the packet and their type.
- Body: Contains the events present inside the packet.

The extraction of events from an .AEDAT file gives temporal and spatial information. In particular, the following parameters can be decoded:

- chip id: The id of the chip where the events took place
- core id: The id of the core where the events took place
- neuron id: The id of the neurons responsible of the event
- ts: the time stamp of the events, expressed in us scale and taking as reference the time when cAER control software has been started.

DynapseOutDecoder
Built on top of the code gently provided by Federico Corradi, DynapseOutDecoder is a python module of DYNAPSETools library that take care of the decoding and processing of events from the .AEDAT format. The core of the module is the high level object called EventsSet. An EventsSet is, as the name suggests, a collection of events that are spatially and temporally defined, i.e. can be attribuited to a neuron address inside Dynap-se and inserted a timeline of events.

From the EventsSet object is possible to build custom analysis or, as alternative, benefit from different functions offered by the library:

- **Events filtering:** Output recordings from Dynap-se contain the activity of all the neurons present inside. The device, however, is affected by a background activity problem: mismatch between neuron circuits could lead to unwanted activity, even when no input is present. Since such undesired events will appear in the output recording, the library provides a simple way to design filters able to isolate only of the activity of neurons we are interested o, and removing the unwanted one.
- **Raster plot:** A raster plot is a representation of a network temporal activity in function of the neuron index. In simple words, every time a neuron creates an event, the correspondent neuron index is plotted at the time step in which he has fired. An example of raster plot is visible at fig. 6.3
- **Analog wave matrix evaluation:** An Analog wave associated to a neuron activity is an algorithmic way to obtain a continuous analog function from the discrete time

events generated by the neuron. An Analog wave matrix is a collection of Analog waves extended to an entire network., an example is depicted in fig. 6.3. Such objects represent a detailed representation of the spiking activity of a network, because every spike deeply influence their shape. For this reason they are very useful for training or testing purposes.

- **Extraction of temporal activities:** The library allows the user to extract recorded events temporally located between the activity of two neurons. Such feature is really useful in order to synchronize the offline analysis with the recorded network activity only on the time window of interest.

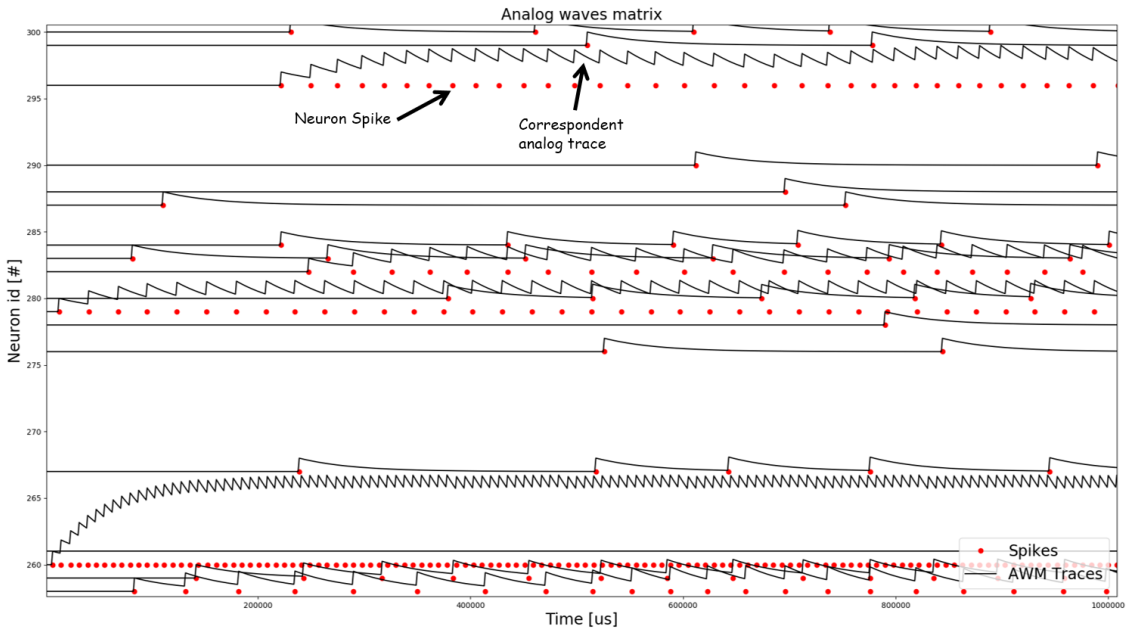

Figure 6.3: Snippet of an analog wave matrix (black lines) constructed on a raster plot (red dots)

6.3 Experiments design

In section 6.2 we talked about the software tools developed to simplify the programming of DYNAP-se. In this section, instead, we will focus our attention on the techniques that we have used to carry on the tests on the neuromorphic board.

Initially, we provide a description of the general structure on which every experiment is based on, while detailed information are reported in section 6.4. Later we will get in deep into the description of the problems faced during the experiments design, and how the developed libraries has been used to overcome them.

6.3.1 General structure of an experiment

With the term experiment we indicate the sequence of steps that has been designed to carry on elaborations with Dynap-se neuromorphic board. Although each experiment has his own characteristics, because created in order to solve a certain type of task, they all have a common structure, which is based on the work flow described in section 6.2.1. All the features we have implemented over this base methodology derives from the main goal of the project, which is the classification of ECG signals using reservoir computing. Such definition gave us all the information over which the experiment structure should be based on:

- The term "classification" highlight the need to implement training and testing procedures, which will be applied over datasets of analog signals. Such inputs, which are characterized by a continuous shape, must be adapted to the spiking neural networks world.
- The term "reservoir computing" defines two things: first, it establishes which network structure must be used to solve the tasks. Second, it specifies the computational method that will be used to train the network.

Wiring all these information together, we came up with the definition of the elaboration structure described in fig. 6.4 and fig. 6.5.

In a preliminary phase, we program Dynap-se board with a reservoir of recurrently inter-

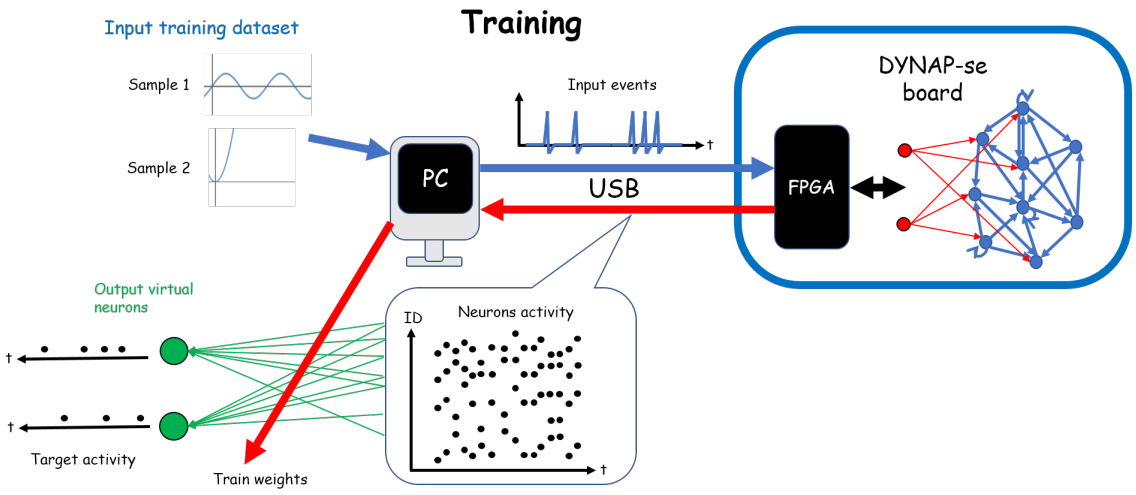

Figure 6.4: Schematic description of designed experiment training procedure

connected neurons, with the same structure as the one explained in section 4.2. In addition, we construct a training and testing set of input stimuli by converting the analog signals that we want to classify into a sequence of events. This last step is performed in order to bring them into a format compatible with the neurons chip communication system,

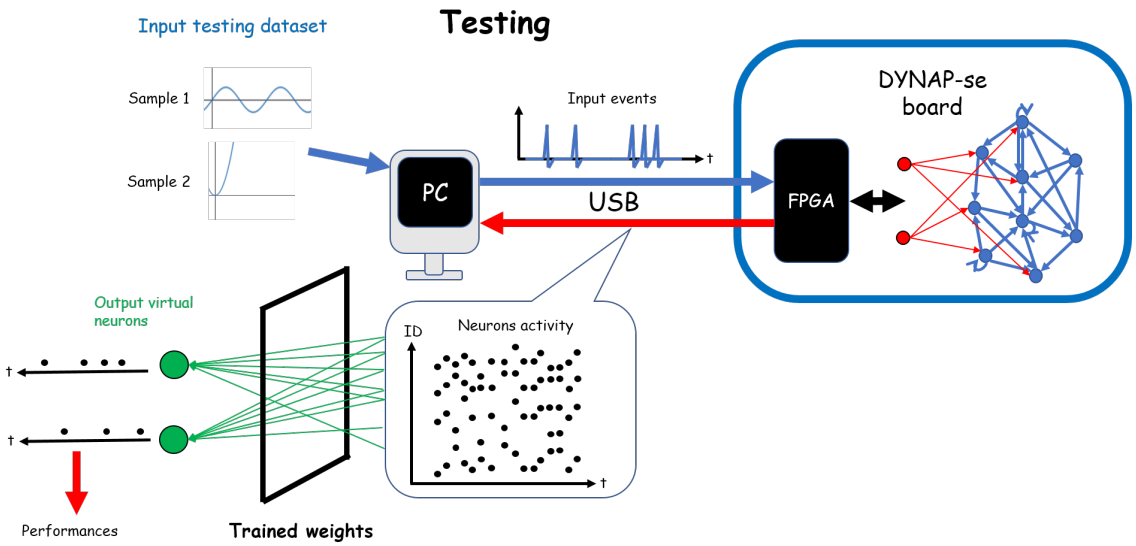

Figure 6.5: Schematic description of designed experiment testing procedure

explained in section 3.3.

We then provide the resulting training trains of spikes as input to the previously programmed reservoir, and record its firing activity. Performing an off-line analysis, we apply a regression analysis between the activity of the network and the required activity of the output stage. The result of such operation is a set of weights that should make the network achieve the classification task.

To evaluate the performances of the trained networks, we perform another recording with Dynap-se, this time providing the testing set as input. We record the output activity and, by applying the weight found in the previous training phase, evaluate how different is the output stage activity from the correct one.

From a technical point of view this flow of operations seems straightforward, if performed by CPU based computational systems. The same cannot be said when trying to execute the same operations with Dynap-se neuromorphic hardware. Different issues, not present in software, emerges. In the next section we will describe the problems we had to face, as well as the solutions that we have implemented to overcome them.

6.3.2 Inputs encoding algorithm

In section 6.2 we have described one of the way in which DYNAP-se chip can be stimulated, which imply sending to the chip patterns of artificially created events. These last ones will be then accepted by the internal interconnecting system and routed toward the neurons we want to stimulate. In our case, however, we need a conversion system that transforms the

datasets of analog signals into patterns of events, related in some way to their initial shape.

In order to solve the problem, we decided to accomplish the signal-to-event transformation using a threshold algorithm. In fig. 6.6 is depicted an application example, where an half period sinusoidal signal is converted into events. Such algorithm, as the name suggests, define a fixed threshold value and, whenever the signal steps up or step down of an amount bigger than it, an event is generated. According to the direction of the step, two different streams are generated: for positive steps an UP event is generated, for negative ones a DOWN event is generated. The two resulting sequences can be used as two different input channels for network stimulation. Such algorithm does not fix the maximum and minimum firing frequency, but they depend only on threshold amplitude and signal dynamics.

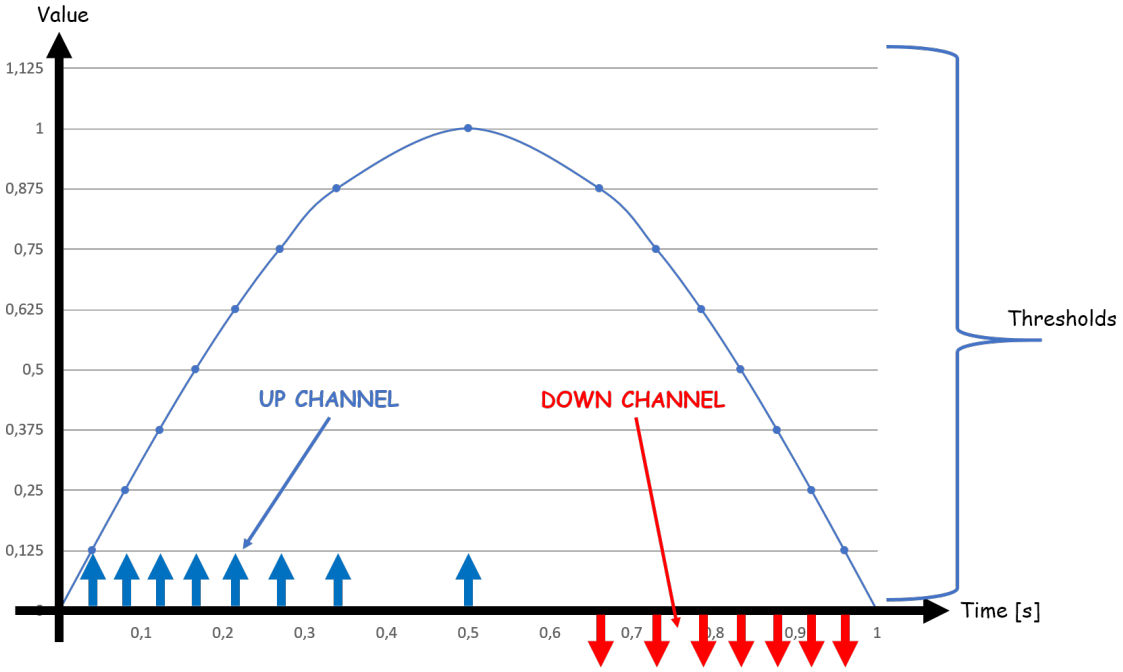

Figure 6.6: Example of conversion from analog signal to spikes. In the example, half a period of sinusoid is converted in events, with a threshold equal to 0,125

In the figure is possible to see how every time the signal cross an horizontal line, representing an instance threshold, an event is generated. When the signal is rising the spikes have blue color, corresponding to the UP channel event stream. On the other hand, when the signal is falling down, the events change color to red, indicating the DOWN channel events stream.

6.3.3 How exploit automatic sample extraction

When performing testing or training, neural networks are provided with streams of input samples and produce an output activity response, that will be used for learning algorithms or performances evaluations. In Dynap-se we can apply the same procedure, but it's important to bear in mind that some problems raises. In first place, how can we temporally isolate, from the output recordings of the network activity, the events associated only to training or testing samples? In other words, we need a procedure that allows us to filter away all Dynap-se activity that is not related to the input stimulus, but to the natural background activity of its neurons. We called it automatic sample extraction

Automatic sample extraction is an operation for which, starting from the output recording of the network activity, is possible to automatically extract the temporal events that are related to a certain input sample. Such functionality is essential for the proper working of the experiment, because it is at the base of the training and testing operations.

The idea we came to involves an upgrade of the network structure, as well as of the input patterns. Besides the stimulation of the network we want to analyse, we stimulate two other neurons, called start trigger and stop trigger. Their activity will have a purely functional goal. The first neuron will be activated just before the beginning of a dataset sample stimuli. In the recorded board activity, we associate the start trigger event as the time where the neuron activity is related to the input sample. In the same way as the start trigger, the stop trigger neuron would encode the end of the activity related to the sample. In this way, isolating only the events present between the ones produced by the trigger neurons, we can automatically extract the activity of each of the dataset samples. Start and stop trigger works as synchronizers for every sample, determining the timing borders in which it happened.

6.3.4 How to exploit automatic target evaluation

Once extracted the events associated to training or testing samples, how could they be automatically labelled? This operation consist in the association of every sample with the target value of the output classification layer, depending on the input that has been applied. Such job can't done manually for every performed analysis, but an automatic approach, that we can call Automatic target evaluation, is needed.

Automatic target evaluation imply the ability to retrieve, from the recordings of the network output activity, the input sample type that has been applied as input. When doing training, in fact, two elements are essential: the activity of the neurons whose connections are involved in the training, and the desired target values of the classification neurons. Retrieve the first ones is an easy task in Dynap-se, because already present in the output recordings. The same cannot be said for the target values. In fact, they are strictly related with the input stimulus type that, unfortunately, cannot be easily obtained only by analysing the output activity of the network.

This problem can be solved in a similar way done for the automatic sample extraction: use some particular neuron events to encode informations useful for the analysis. In the current case we don't need only a start and stop trigger, but one neuron for each input signal type, everyone of them encoding for a different target activity we want to train for. The input spike pattern must be designed such that, just before an input sample is sent, one of these neurons is activated, encoding in his activity the number of the sample that is being sent.

In this way it becomes easy to retrieve the input sample type from the output recording, because it is only necessary to discriminate which of the stimulated neurons has fired.

6.3.5 How visualize input stimuli in recordings

When working with network spike recordings, the user can look at the raster plot of the activity and make some first-aid comments about its behaviour. This type of analysis is useful to find a correlation between the performances on the problem-solving task and the activity of the network. In this occasions, it is highly recommended, and extreme useful, to make a comparison between the input events pattern and the output behaviour of the network.

In order to implement this functionality, we can connect the input channels, other than to the network that must be analysed, also to a specific set of neurons, tuned in such a way to spike one time for each input event. These neurons don't have any other connection besides the one at input, and don't influence the network behaviour, but only work as a representation of the input events pattern. Its activity is intrinsecally included in the output recordings, so it is easier for the user to make numerical or visual analysis of the network behaviour according to the input stimulus. This approach brings with itself another advantage. Reversing the steps performed to encode the input signal into discrete information (explained in section 6.3.2), it is possible to reconstruct the original shape of the signal from the recorded events. An example is depicted in fig. 6.7.

Such feature is really important because it makes possible to graphically associate the input signal dynamics to the network activity and the classification results, making easier the evaluation of the network properties and the search for solutions to enhance the performances.

6.3.6 Summary of input controls

Taking into account all the solutions that has been described in the previous sections, every input pattern designed for the experiments done with Dynap-se has the structure depicted in fig. 6.8. In the figure are represented the different parts composing the pattern:

- Input start trigger event: excites a neuron that triggers the start of the pattern
- Target encoding event: excites a neuron that encodes for the input sample type

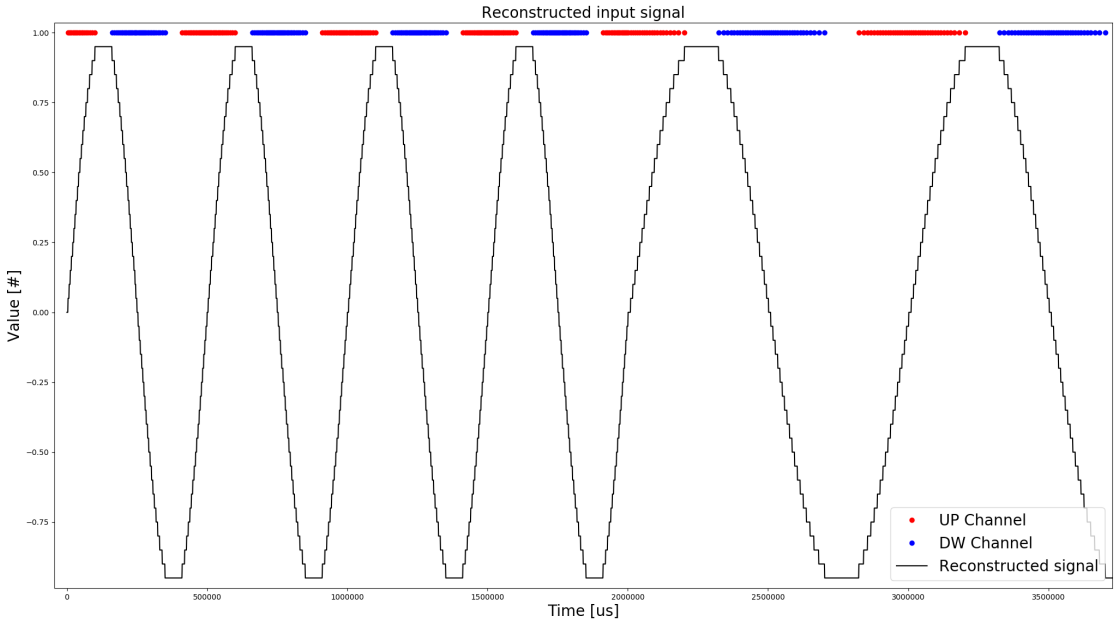

Figure 6.7: Reconstructed signal from input spikes pattern

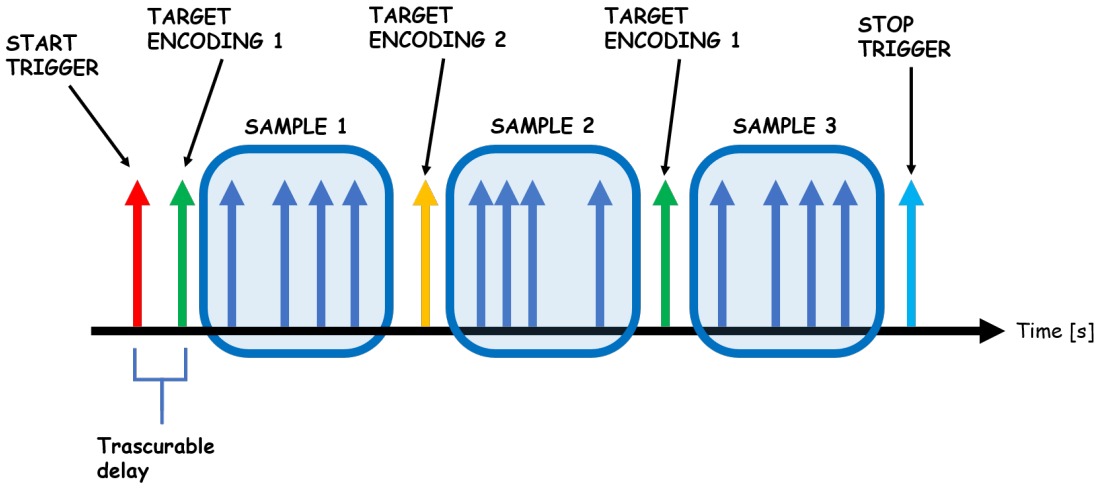

Figure 6.8: General structure of input stimulus pattern, complete of all the encodings

- Sample 1, 2, 3: subpatterns making up the input sent to the device network
- Stop trigger neuron: excites a neuron that triggers the stop of the pattern

With such input structure is possible to design sophisticated analysis where most of the operations can be performed automatically by the software, using the informations contained directly in the input stimuli. With the input structure in fig. 6.8, for example, we can automatically extract the resulting network activity related to the input by taking

only the events present from start and stop trigger; moreover, we can tune the training or testing procedures taking the wanted samples following the indications of the target encoding event.

6.4 Experiments description

In section 6.3 we have described the fundamental approach followed to conduct experiment on DYNAP-se board, carefully designed to guarantee flexibility and reduce the amount of manual processing that must be done. The only parameters that are subjected to changes are the programmed network and the input stimuli, as well as the neuron settings. In the current section, on the other hand, we will focus the description on each performed experiment, providing all the characteristics and details that ensure their reproducibility.

6.4.1 Network memory analysis

Memory analysis is an experiment that involves the impact of neuron and synapses time constants on the global behaviour of a reservoir. In a more specific way, we want to estimate how much the activity of the reservoir is influenced by the past stimuli, factor very important for this kind of networks, as we saw in section 4.2. We will describe in the following the methodology that has been developed in order to perform the estimation. In the first part we focus the attention on the way neurons and synapses time constant has been evaluated. In a second step, instead, we will describe the designed experiment to test the network memory response.

Neuron and synapses time constant estimation
The membrane time constant defines the temporal characteristics of a neuronal response to a input synaptic stimulus. From a biological point of view, tt is defined as the time requested by the membrane potential to decade below the 37% subsequently to the depolarization caused by the stimulus. Synaptic time constant, instead, is directly related to the rate of decay of the post synaptic current produced, and is related to chemical and electrical properties of the channels where is located. If we switch from the biological world to the neurmorphic one, we can relate simply relate the time constants of both neuron and synapses to their DPI circuits representations. We can rewrite the formula written in chapter 3 in function of the biases parameters of which we can access, in the following way:

$$\tau_{out}\frac{d}{dt}I_{out} + I_{out} = \frac{I_{th}}{I_\tau} \tag{6.1}$$

Where $\tau = \frac{CU_T}{\kappa I_\tau}$

Such equation is what defines the dynamics, for the neuron, of the filtered current coming from an input spikes, as well as the current output of a the synapse circuit. It tells us something import about the dynamics: if we decrease synapse or neuron I_τ current,

we can increase the *tau* values, which correspond to a decrease of the current filter time constant. Changing this value we can can tune how slowly the synapse or neuron inject current toward the membrane capacitors, so we can influence the membrane time constant. In order to demonstrate so, we create a low frequency input stimuli at 2Hz as input to a whole DYNAP-se neuron core, depicted in fig. 6.9.

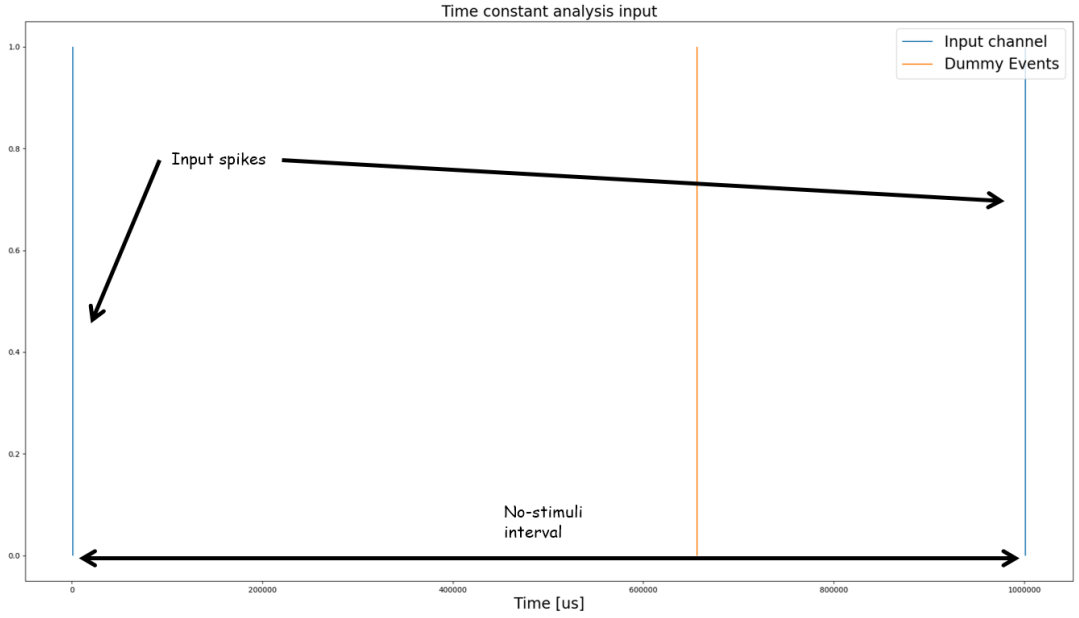

Figure 6.9: Input signals for reservoir memory evaluation testing

We tuned the synapses weights such to give the neuron a pulse able to raise the level of the membrane potential, but not let it fire. In this way we make sure that the only influence to the neuron is its input DPI circuit, and the synapse one. Then, by changing the I_τ and I_{th} parameters of both neuron and synapse, we can estimate the change of time constant by visualizing with the oscilloscope the neuron membrane potential.

Since only the slowest between neuron and synapse time constants influence the membrane potential dynamics, we can decide if to estimate the former or the latter by making the other one the longest can be achieved in DYNAP-se.

Reservoir time constant estimation

We have shown in the previous section a method to estimate the neuron and synapse time constant. We want to but how can we get the same result with a recurrent network of neurons, like the reservoir? The approach we have decided to follow is similar to the one used for neuron and synapses time constant evaluation. It requires to stimulate the network with a custom spike pattern and analyse the network activity after the last spike

has been sent. An example of the designed input is present in fig. 6.10. Since we wanted to make an estimation of the time constant in a situation as close as possible to the future experiments one, we have used the same inputs that we will use in the first experiment. They are spaced by a no-stimulus intervals long enough in order to ensure that the activity would fade.

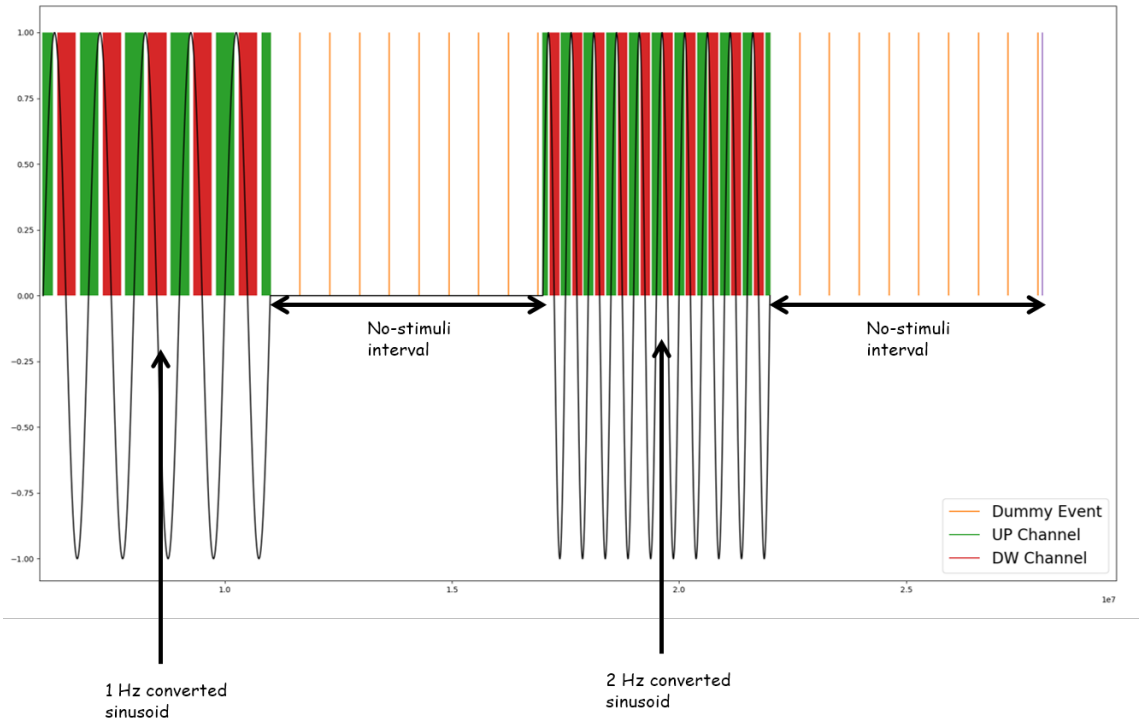

Figure 6.10: Input signals for reservoir memory evaluation testing

We estimated the reservoir time constant as the amount of time that it takes from the network total analog wave activity, calculated as explained in section 6.2.4, to return at the base level it was before the input started. That amount of time defines how long the input was still influencing the network activity even if it was not anymore present, therefore can be correlated to the memory of the network.

6.4.2 First task: classify sinewaves from their frequency

The first task that the network was requested to accomplish it consisted in the classification of analog sinewaves signals with different frequencies. In the following we will not only give informations about the task characteristics, but we will broadening the description providing the details for all the followed passages: how the network has been implemented, how the chip biases has been tuned, and how output training and testing has been performed. Such informations are detailed at this stage, but are valid for all the following tasks. When significant changes are applied, a new description will be inserted.

Task summary

The network input stimuli are two sinusoidal signals at two different frequencies: signal A at 1 Hz, while signal B at 2 Hz. The goal of the reservoir is to classify between jittering versions of them, providing different output activities according to the frequency of the input signal. In the following lines we will focus in detail on the characteristics of the designed network, as well as the tuning procedure adopted to find optimal biases parameters.

Input signals

Network inputs are constructed from custom datasets of jittering sinusoidal signals. The training dataset is constituted by a sequence of 50 versions of the two types of sinusoidal signals, injected with a 4ms jitter derived from a Gaussian distribution. Their conversion into spikes is achieve by using the threshold algorithm, explained in section 6.3. In fig. 6.11 is depicted a graphical representation of the conversion results, for two random jitter applyed. Note that, to improve variability, the jitter value is not shared among all the spikes, but retrieved randomly from the distribution for each one of them. The resulting UP and DOWN channels constitutes the input streams of events that are sent as input to Dynap-se chip for training pourposes.

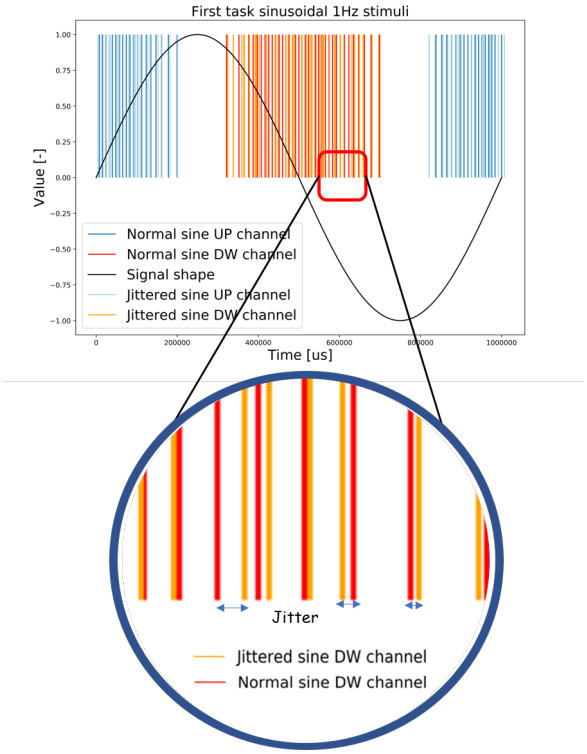

Figure 6.11: Example of first task 1 Hz sinusoidal signal with and withouth temporal jitter

The testing dataset has been created in the same way as the training dataset, and is composed by other 50 versions of sinusoidal signals jittering versions not included in the training one.

Network Architecture

The designed network can be logically divided into two different parts, both located in the Chip 0 of Dynap-se board, but belonging to different physical cores. The division is depicted in fig. 6.12.

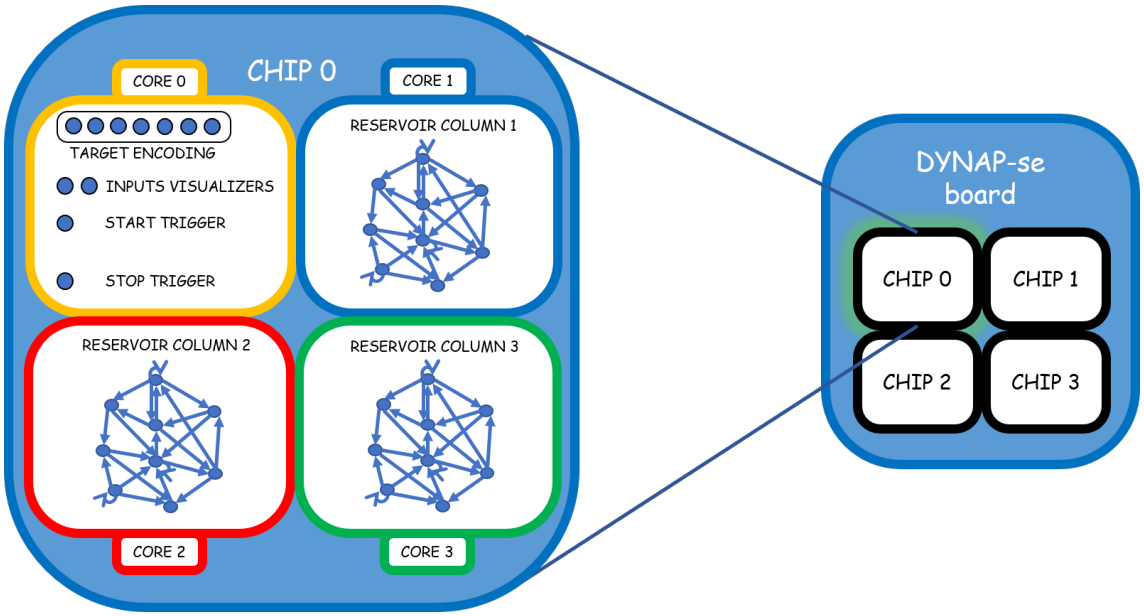

Figure 6.12: Disposition of the network inside Dynap-se board

The control core is located in Core 0 and includes all the neurons whose activities encodes informations for experiment analysis and control. Among them we have the input start and stop trigger neurons, the target evaluation ones and input stimuli visualizers, for which we have talked in detail in section 6.3.

The computational cores are located starting from Core 1, where is placed the RNN reservoir network. Its structure and characteristics are inspired from the one described in Maas paper [9], firstly because he is the inventor of the LSM approach, and secondly because the tasks for which his network has been used are computationally similar to the ones addressed by the thesis work. We will describe in the following lines the characteristics of the network, while the detailed parameters and probability distributions used to design its structure are reported in appendix A.

The reservoir structure is a recurrent neural network composed by 3D columns of neurons, each one of them consists of 4x4x16 neurons, for a total of 256 neuron/column. These

structures are inspired from the one described in Maas paper, but they have been adapted to the neuron disposition inside Dynap-se chips. Putting more column together, in our case along the y direction, we can change the general shape of the network and get different behaviours. 80% of column neurons exploit a fast excitatory dynamics, while the remaining 20% a fast inhibitory one. The types are randomly assigned with an uniform distribution. An example of column neural structure is present in fig. 6.13, while at fig. 6.14 is depicted an example of multiple instances packet together on the y direction.

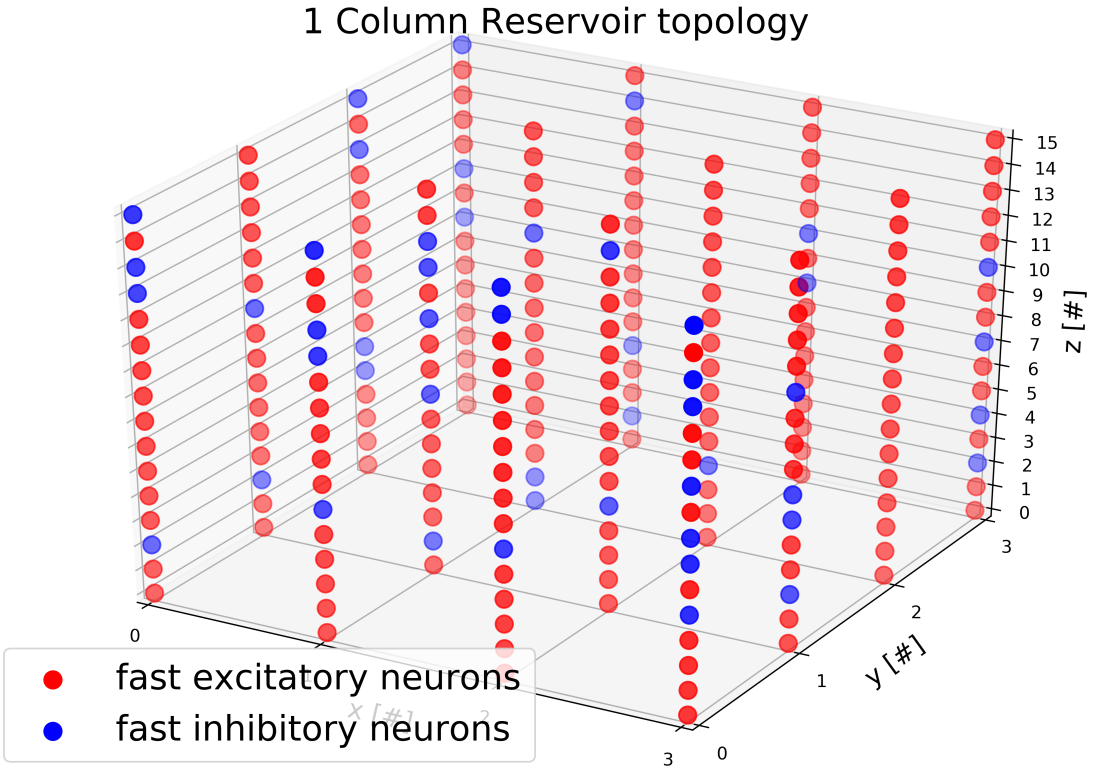

Figure 6.13: Reservoir network topology with 1 column

For what concern connectivity, each column receives inputs from the UP and DOWN channels of the virtual neurons, emulated by the board FPGA, at which they are connectd with a probability taken from a uniform distribution. A visual demonstration is depicted in fig. 6.15. All the connections have been configured as fast excitatory (see section 3.3 for more informations), with a synaptic strength that is derived from a Gaussian distribution, in order to deliver a slightly different input to the network areas.

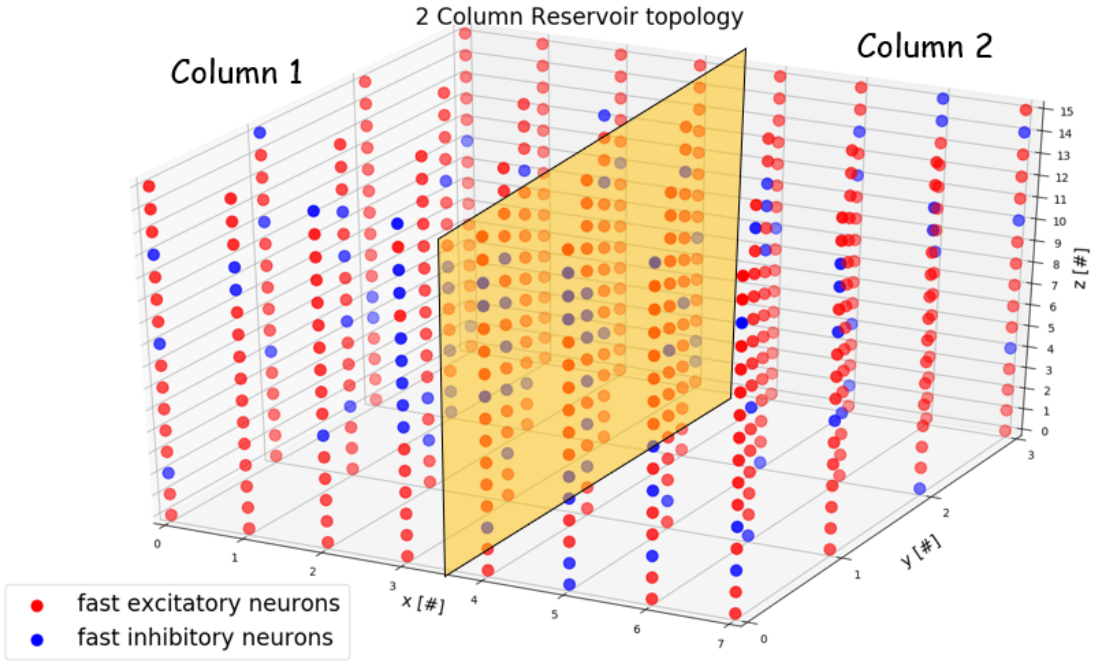

Figure 6.14: Reservoir network topology with 2 column sided on the y axis (separated by the yellow rectangle)

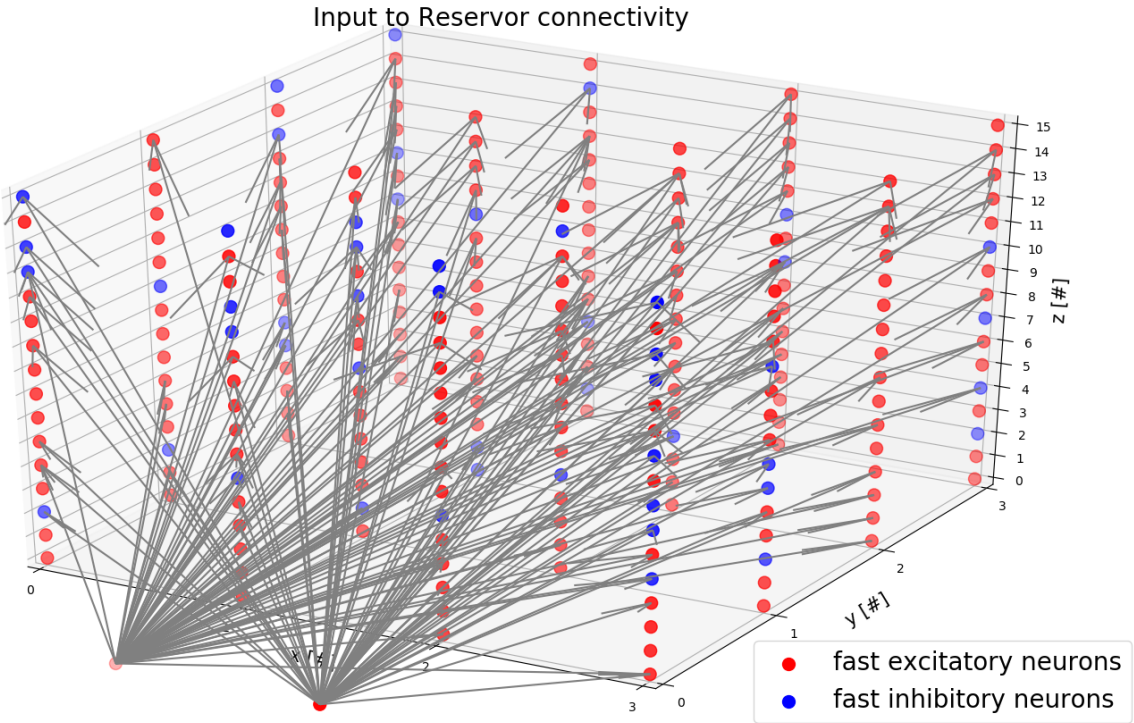

Figure 6.15: Connectivity map between the input neurons and the Reservoir ones

Recurrent connections are allowed between all the different types of neurons, excitatory or inhibitory. The connectivity probability is derived from a Gaussian distribution inversely proportional to the Euclidean distance of the connection. In other words, the more the neurons are further apart, the less will be the chances of a connection, thereby promoting short distance ones. The coefficients that regulate this probability distribution are tuned to allow stronger and broader inhibitory connections with respect to the excitatory neurons one. As a title of example, in fig. 6.16 are reported the connectivity of the inhibitory neurons toward the excitatory ones.

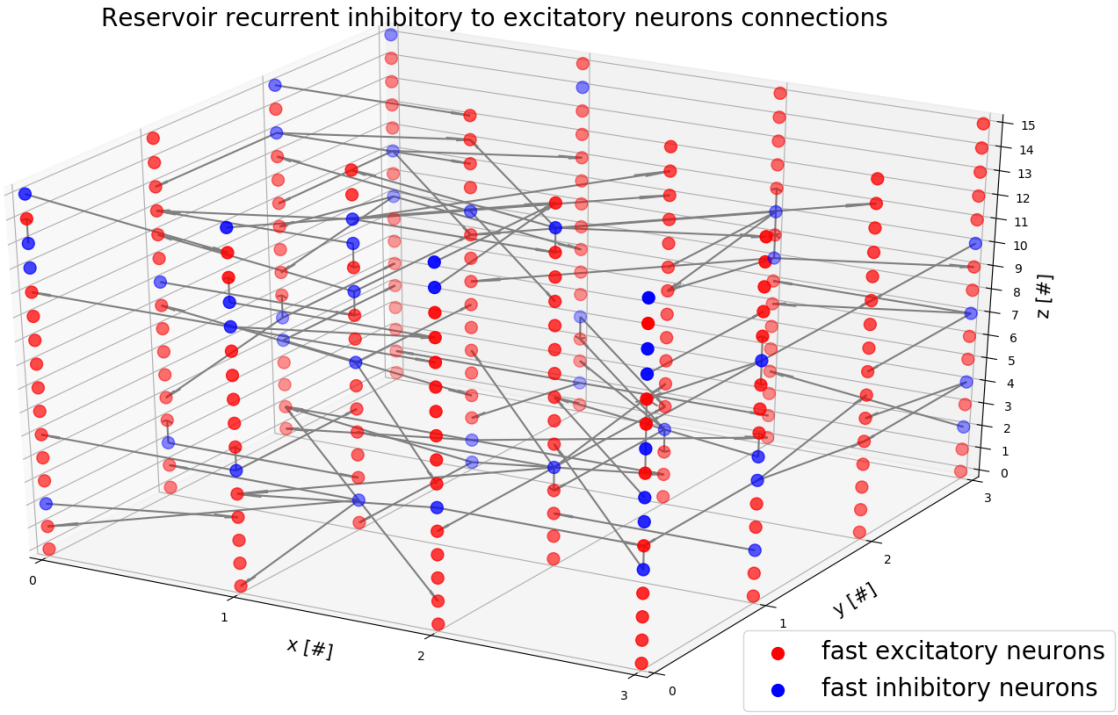

Figure 6.16: Connectivity map between the input neurons and the Reservoir ones

Such network structure is inspired from Wolfgang Maass paper regarding LSM computation [9]. In appendix A are reported more detailed informations about the network characteristics, as well as all the parameters values used to design the network structure.

Network Biases tuning

In the following lines we will provide the guidelines designed in order to obtain a correct behaviour of the network. With "correct behaviour" we mainly mean three things: only the neurons that are involved in the analysis must be active, the control core must be biased in order to exploit a deterministic behaviour, required in order to act as a experiment controller, and the computational cores must be tuned such to exploit a memory behaviour required for a LSM to work properly (as explained in section 4.2).

The elimination of the unnecessary neurons is done by increasing their leakage current at the maximum allowed. Such condition is obtained by increasing the parameter I_TAU of the neuron that, by looking at fig. 3.5, corresponds to an increase of voltage V_{lk} across the transistor M_{L3}. In this way every input to the neuron is drained toward the ground node, preventing the charging of capacitor C_{mem}. In our design, this condition has been applied to all Dynap-se Chips besides the one where was located the network object of analysis, which is Chip 0.

Besides the elimination of unnecessary neuron, a secondary step would see the elimination of unnecessary neuron or synapses features. Even if not used, leakage in the transistor creating this features could lead to influence of neuron and synapses dynamics. In our network, for example, we used only fast excitatory and fast inhibitory synapses. Following the same approach applied for neuron elimination, we removed the unwanted contribution of slow excitatory and slow inhibitory synapses by increasing the parameter I_TAU, combined with the decreasing of the synaptic weight. In a similar way we canceled the influence of the neuron adaptation feature, not used in the first analysis.

The control core tuning is a more delicate topic. The dynamics we want to obtain, in this case, is not a normal way of operating with neurons. For our purpose we require, in fact, that every neuron must fires one time for every input spike, independently from its frequency. The reason for this requirement resides in the need of having the complete control of a chip population in order to exploit the features we have discussed about in section 6.3. The problem, however, is that we have only full control over the virtual neurons emulated by the FPGA, but not the physical one. The solution that we have found, hence, consists on connecting selected virtual neurons to some physical ones, making sure that these last one fires only one time for every input spike. Such approach would inheritely extend the control of the virtual neurons to the physical ones.

To obtain this behaviour we had to run a small experiment. We connected the target Core to a single virtual neuron, sending a constant frequency input stream of events. We have then increased to the maximum allowable value the I_TH parameter. Such parameter corresponds, taking as reference fig. 3.5, to a decrease of voltage V_{thr} across the transistor M_{L1}. This operation thus let the DPI filter amplify considerably the input current sent to the neuron and, combining it with the increasing of the synaptic weight, we would certainly obtain a post-synaptic neuron spike for each input event. In order to prevent bursting behaviour, the leakage current I_TAU must be increased progressively until the condition of one spike for each input is achieved. Varying the input stream frequency it is possible to test the range where such dynamics held.

The computational cores tuning goal is to give the network the ability to process the current inputs in a way that takes into account the influence of the previous stimuli. Such behaviour, as explained in section 4.2), is what creates an LSM. Note that only the presence of a recurrent neural network is not enough to obtain a liquid state machine. For example, if the internal recurrent connections weights are weak enough, the RNN doesn't

achieve recurrent activity, becoming in this way a feed forward neural network. As we have seen, this type of networks does not exploit memory behaviour, hence cannot be considered an LSM. Since such topic is of great importance for the tasks that has been addresses, a detailed description is given in section 7.1.

Output training and testing
The output layer is composed of 2 virtual neurons, which we can will Neuron 1 and Neuron 2. Each one of them is fully "interconnected" with all the neurons of the reservoir. In truth there is no physical connection between them, because the virtual neurons are are modelled in software in a perpeptron-like approach. Their outputs status is derived by mathematically combining the reservoir activity with the connections floating point weights. According to the input signal frequency, their expected behaviour of the output neurons is the following:

- With 1 Hz input sinusoid, output neuron 1 is firing (status equal to 1), while output neuron 2 is not firing (status equal to 0)
- With 2 Hz input sinusoid, output neuron 1 is not firing (status equal to 0), while output neuron 2 is firing (output value equal to 1)

In other words, output Neuron 1 must react only when Signal 1Hz is stimulating the network while output Neuron 2 must react only when Signal 2Hz is present.

After having converted to spikes using the threshold algorithm we have described in section 6.3.2, it is sent as input to the network, while its activity is recorded in an output file. The same approach is used to stimulate the reservoir with the training dataset. The network activity is then retrieved and the Analog Wave Matrix is calculated by using the developed python library. The output weight training is then performed by using simple ridge regression and the trained weights are applied to the testing dataset activity, obtaining a classification prediction. Such prediction, since produced by just mathematical calculations, would be very noise and unstable and, for this reason, it will be filtered by a sliding window performing the average over the sample time interval.

A threshold is applied to the resulting prediction: output neuron is 1 if it is above the threshold or 0 in the other case. The final output, so constructed, is compared to the target one, automatically evaluated from the recording. The final value for the sample prediction is its last one before the beginning of the next input, because it contains all the informations that the network has integrated during the sample. The performance of the network is evaluated by comparing these last values with the equivalent target ones, expressed in percentage over the total amount of inputs.

6.4.3 Second task: detection of perturbations in sinewaves

Task summary
While in the first task we have studied the reservoir behaviour with noisy sinusoidal signals,

in the current one we want to test the network capabilities of maintaining in memory the previous stimuli that has been presented. To do so we will partly use the same samples adopted for the previous task, but combined in order to request a bigger effort from the network in order to obtain reasonable performances. In the following we will describe the characteristic of the task, focusing on the features that make it more difficult with respect to the previous one. The network structure and biases applied are the same as the first task, so no description about the network architecture, bias tuning or output training will be discussed, but we reference to section 6.4.2

Input signals
Starting from the datasets created for the first task, including 50 noisy istances of 1 Hz and 2 Hz sinusoidal signals, the construction of the datasets is based upon them, using the following guidelines:

Signal 1 is constructed taking one sample for each of the two signals, joining the 1Hz one directly after the 2Hz. The resulting signal is depicted in fig. 6.17.

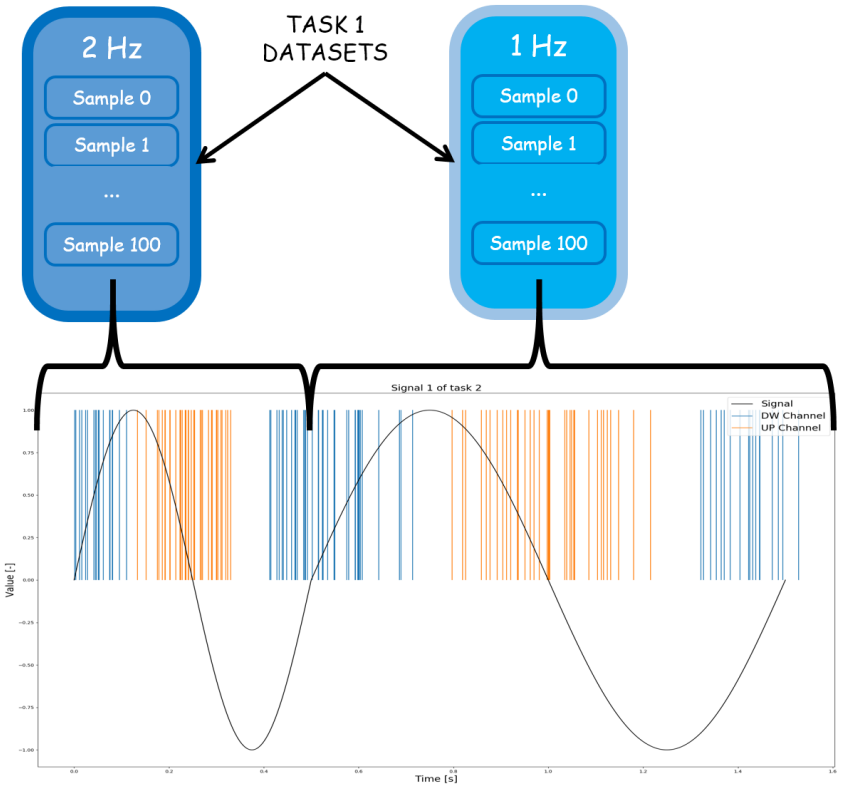

Figure 6.17: Example of second task Signal 1 composition from the previous datasets

Signal 2 is constructed by creating a new equivalent 1 Hz signal with a duration equivalent

to the one of Signal 1. The created signals are different from any of the ones that has been used in the previous datasets. An example is depicted in fig. 6.18.

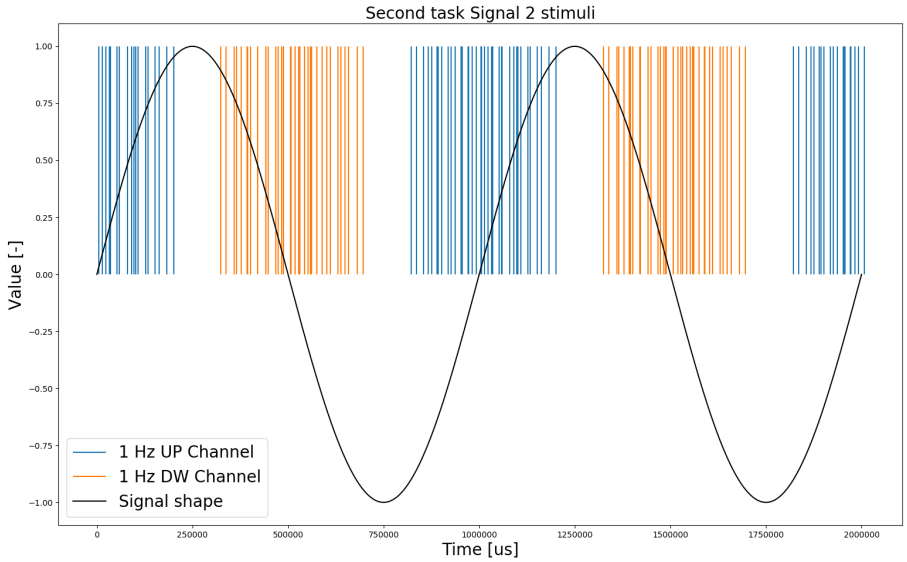

Figure 6.18: Example of second task Signal 2 generation

The same procedure has been followed to generate the testing dataset, obtaining in this way 50 training and testing samples, the same amount that has been addressed by the first task.

From the shape of the input signals is possible to understand where the difficulty of the task resides. In the first half part of the input, the network will perform the same classification as in the first task, while in the second part things become more complex. The two signals, besides the different jitter distribution, become equal. The only way the network can still be able to give a correct classification at the end of the sample is by remembering the first part of the input stimulus, and using this memory to maintain the output prediction stable at the correct value.

6.4.4 Third task: detection of perturbations in artifically created ECG signals

Task summary
While with the first task we have studied the behaviour of the network in response to sinusoidal waves, and with the second one we tested its memory with a "perturbation" constituted by one period of 2 Hz sinusoidal signal, in this task we want to address the classification of ECG signals.

Input signals
To simplify the analysis, and permit to explore with flexibility the properties and abilities

of the designed network dealing with a task of such complexity, we decided to simulate the signals in software. Based on the code gently offered by Felix Bauer, we developed over it a framework able to define a training set and a testing set of jittered ECG signals, alternating normal signals to one where are applied customly selected perturbations. The code, moreover, allows us to modify the shape of the signal according to our need and apply custom disturbances of different intensity and duration and study the behaviour of the network according to them.In the next lines we will show example of perturbations that can be created with the obtained code.

The ecg signal we are using for the test has a frequency of 60 bpm, equivalent to 1 s of period, and can be seen in fig. 6.19. It is constituted by three waves, from left to right: the P-wave, the QRS complex, and the T-wave, separated and shaped in order to simulate the dynamics of a normal ECG signal. There is no implementation of frequency noise or modulation caused by the breathing. The first goal, in fact, is to try to classify the most simple version of them and, then, if possible, gradually introduce more complexity.

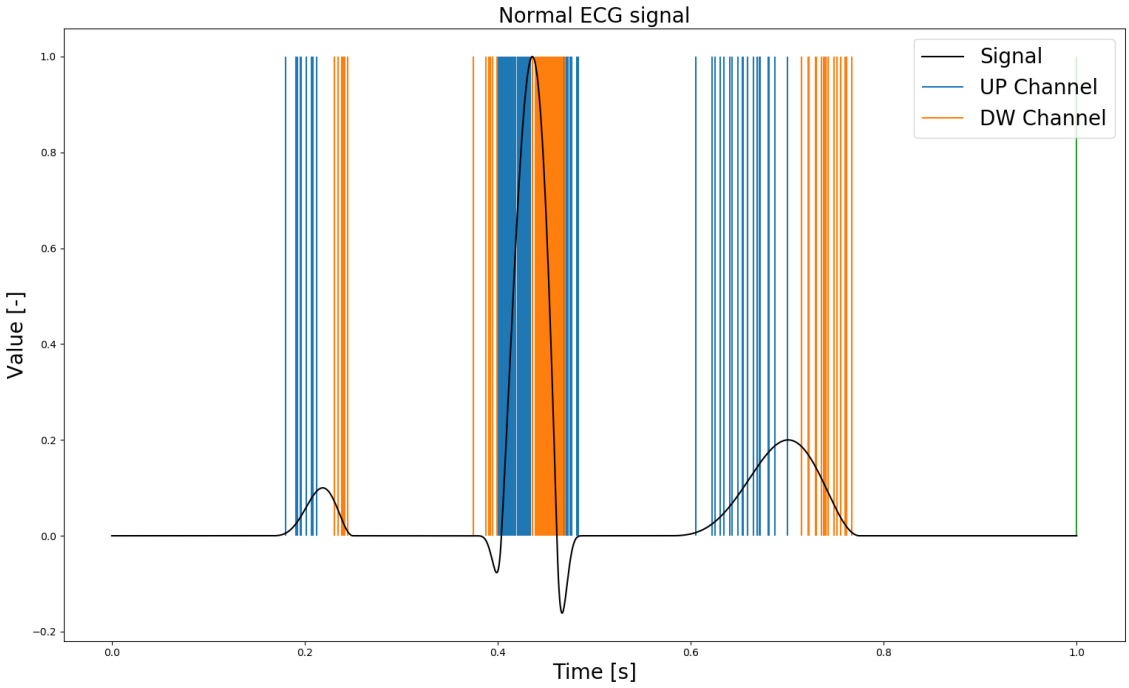

Figure 6.19: Example of normal jittered ECG signal

It can be seen in the figure, from the irregular distribution of the event, that the signal is affected by a 4ms jitter taken from an uniform probability. In this way we always know the maximum shift that the events can be subjected to. In fig. 6.20, instead, is depicted a trace where the T wave, the last of the two present in a normal ECG, is missing.

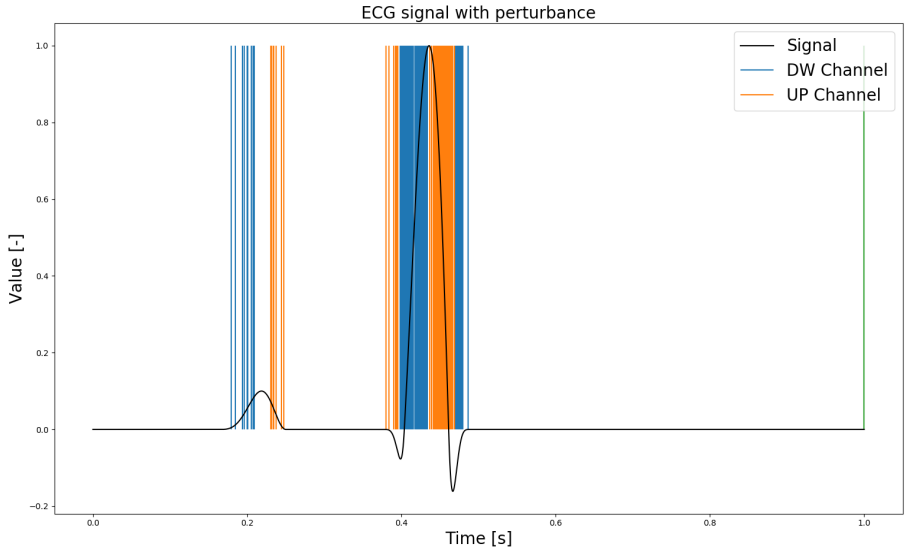

Figure 6.20: Example of a jittered ECG signals affected by the lack of T wave

The second perturbation that we will consider is the lack of the whole QRS complex, as depicted in fig. 6.21. We can see how the only contribute to the final pattern is given only by the P and T waves.

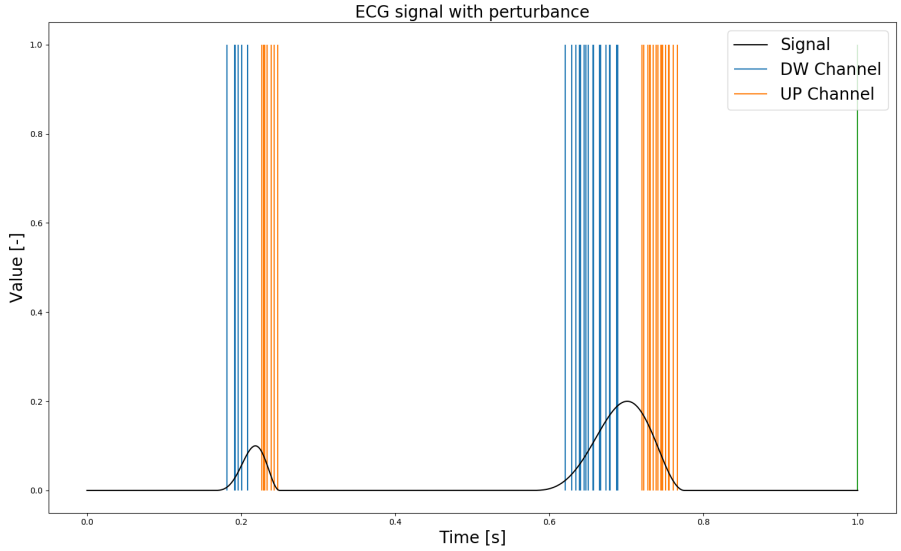

Figure 6.21: Example of a jittered ECG signals affected by the lack of the QRS complex

The last perturbation that we will deal with is, instead, the lack of the P-wave, at the beginning of the ECG signal.

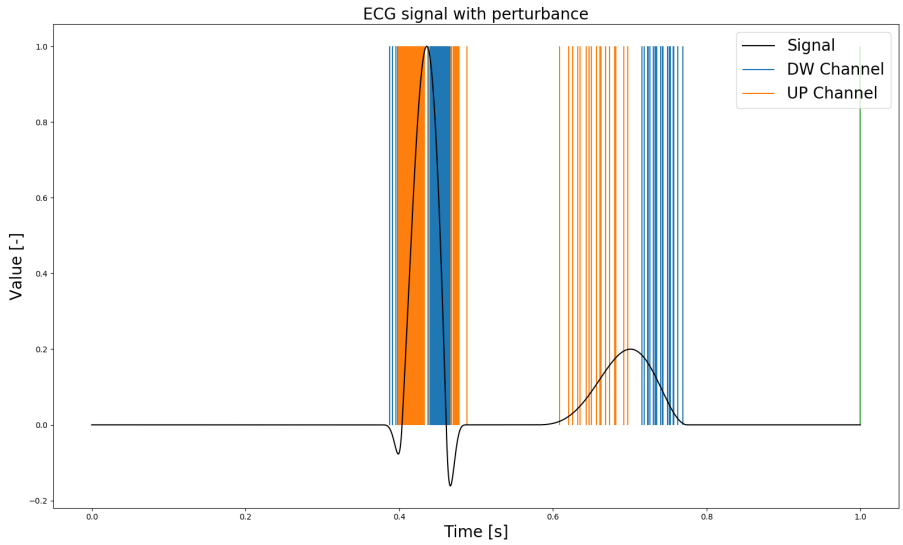

Figure 6.22: Example of a jittered ECG signals affected by the lack of the P wave

Chapter 7

Results

In chapter 6 we have described the guidelines that has been followed to develop the software used to work with DYNAP-se, the framework used to carry on the experiments, and lastly we have talked in detail about the characteristics of each task that has been deal with. The description has been held in a sequential way, giving no particular motivation about choices or logical connections between the experiments. It is the duty of this section to link everything together, having on his part the results of all the experiments that has been done. Often, in fact, some choices has been done according to the outcome of previous analysis, because there was a connection between them. As every work done in the research world, also this one is rich of failures, for instance analysis that didn't bring any expected result or choices that were wrong. Clearly no detailed informations will be given regarding these analysis, but often we will refer to some of them in order to explain choices that, otherwise, would not be understandable.

7.1 Evaluation of network memory

Introduction and motivation
The following analysis, in the thesis timeline, has been carried on subsequently the first attempt to solve the tasks we aimed for. The final performances described in the next section have been achieved in retrospect of different analysis we have performed in order to improve the network properties.

In our first attempt to solve the first task, the classification of sinusoidal signals in function to frequency (explained in **??**), the obtained network performances were even too satisfying, with a maximum value of 100% accuracy. Such good results demonstrate the need of a new task, harder than before, in order to test the effective abilities of the network. When trying to solve it, however, the performances dropped down to a small 55 percent, showing how the results obtained in the previous stage needed to be analysed in a more detailed manner, as well as the network structure and characteristics.

To understand in deep why the analysis explained in this section is needed, we can start by looking at the activity of out first version of reservoir trying to solve the first task that has been proposed, depicted in fig. 7.1

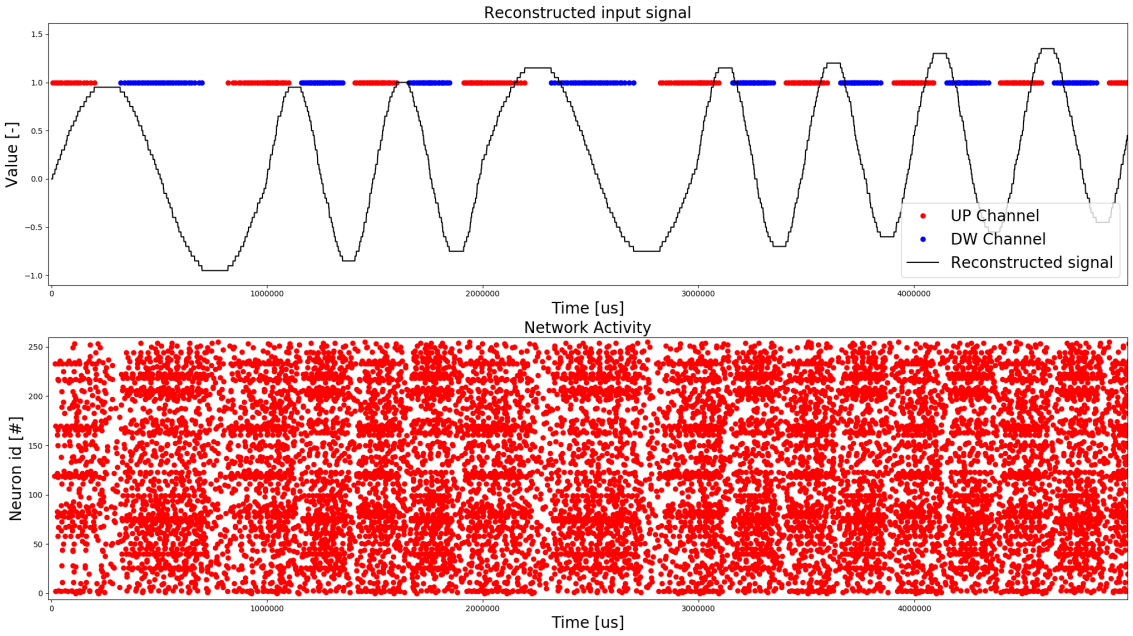

Figure 7.1: Snippet of first task activity with first designed reservoir network

As is possible to notice from the picture, the network activity faithfully follows the input signal dynamics and, for every change of frequency, it adapts pretty quickly to the variations. Such behaviour, although it seems apparently positive because makes easier a classification between the two input signals, in truth hides a characteristics that made it not fit for the second proposed task: the lack of past stimulus memory.

In section 4.2, when we gave an introduction to reservoir computing, we had a detailed explanation of the nature and characteristics of the LSM, an object able to process the current inputs in a way that takes into account of the previous stimuli influence. It has to be pointed out that such requirement is not automatically satisfied by implementing a recurrent neural network structure. For example, if the internal recurrent connections weights are not strong enough, the RNN doesn't achieve recurrent activity, behaving in this way as a feed forward neural network. These types of networks, as we have seen, does not exploit memory behaviour, hence cannot be considered as an LSM.

From fig. 7.1 it can be clearly seen that the network is not influenced by the past stimuli, but the activity is "overwritten" whenever a different input dynamics comes as input. In order to solve the problem, different approaches has been thought, and carried on: from the analysis of the network structures to the research of better connectivity parameters,

to the biases tuning. Without getting too much into detail of the different solutions, we will focus on the way that gave us the wanted results, which is the research for correct biases.

The analysis of reservoir activity memory
The hypothesis over which we built our analysis is based on the fact that the biases we have configured the network restrain it to have a fast dynamics, which prevent any memory of the past stimuli to form. To undo this behaviour, we had to find which parameters we to modify in order to increase the general time constant of the network.

Using the analysis described in section 6.4.1, we studied the correlation between the neurons and synapses time constants and the biases parameters I_TAU and I_THD. As it is possible to see from fig. 7.2 and fig. 7.3, by decreasing their values we obtain waves that are broader in time, because there is more integration of the input activity.

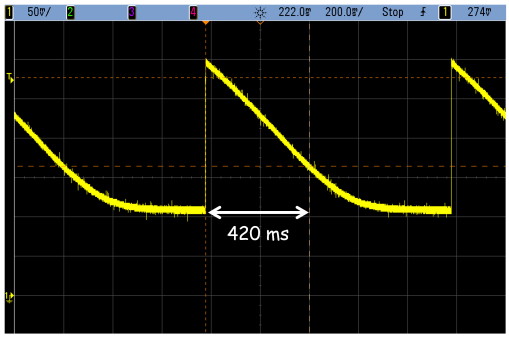

Figure 7.2: Neurons membrane potential response to input pulse with high τ constant

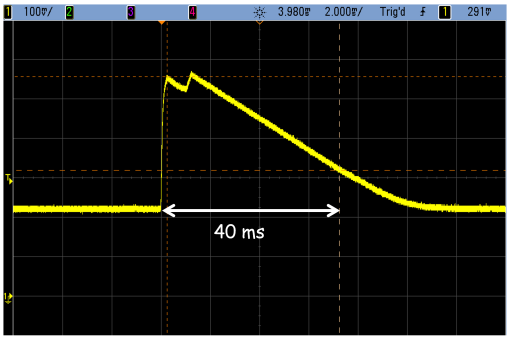

Figure 7.3: Neurons membrane potential response to input pulse with low τ constant

In fig. 7.4 we can appreciate the amplitude of the spectrum of possible time constants that can be achieved with DYNAP-se neurons: from hundreds to few ms time constants.

Such graph is an estimation that does not want to be a formal method to evaluate the neuron and synapses time constant, but gives a indicative behaviour of them. In fact many uncertainties affects the results and should be taken into account for a proper evaluation: from the measurements uncertainties, performed using oscilloscope, to the high variability of these curves between different neurons and synapses, caused by the mismatch of the electronic transistors with whom they are built.
In the light of this fact, we decided to tune the reservoir bias parameters in order to stretch the neuron and synapses time constants to the maximum that can be achieved in DYNAP-se board. To test if the the change had an effect on the network dynamics, we performed the second part the memory analysis explained in section 6.4.1, obtaining the results depicted in fig. 7.5. We evaluated the analog wave response of all reservoir

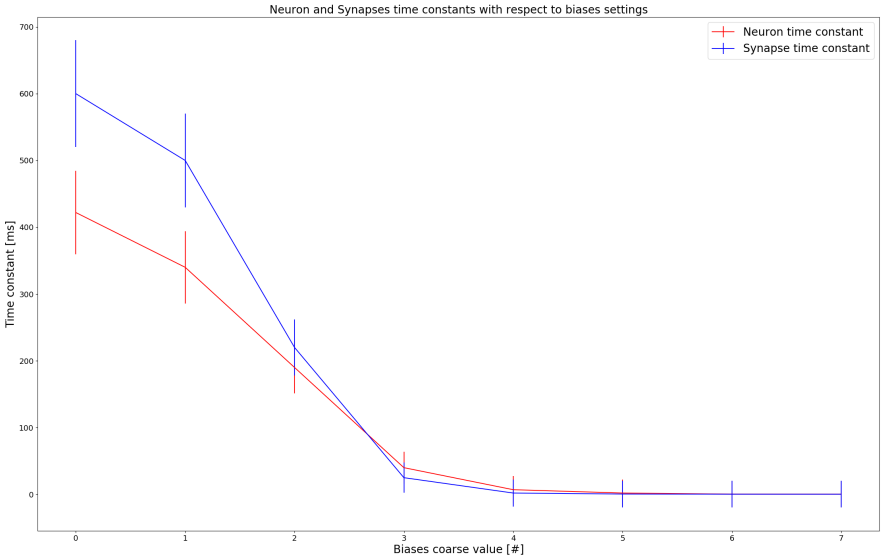

Figure 7.4: Graphical representation of the neuron and synapse time constant in function to the I_{TH} and I_{τ} biases coarse value parameters

neurons responding to the same input stimuli used for task 1, for three different neurons and synapses time constants settings.

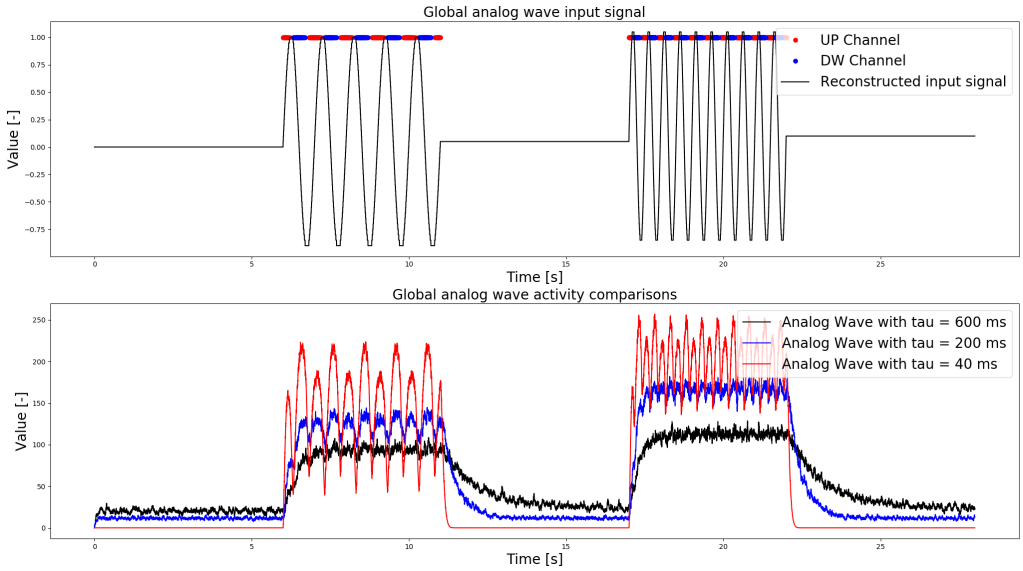

Figure 7.5: Reservoir activity inertia in function of neurons and synapses dynamical properties

On the upper part of the figure we can see the input signal automatically reconstructed

from the spikes of DYNAP-se control neurons, which are represented by the red and blue dots. On the bottom part, instead, is depicted the sum of the reservoir neurons analog waves, a trace that follows the general activity of the network. From th examples shown in the picture is possible to notice how the reservoir behaviour dramatically changes in function to neurons and synapses time constants. We span from no temporal activity inertia with the red curve, whose shape follows pretty easily the input signal dynamics, to the black one, which is the least sensible to inputs dynamics, but its activity lasts for a much higher time. Getting a precise value of the time inertia is a very hard job, because the activity is noisy. An estimation of the values are:

- Analog wave (red one) with tau = 600 ms : 4 s
- Analog wave (blue one) with tau = 200 ms : 2 s
- Analog wave (black one) with tau = 40 ms : 0.4 s

Such result gives us hints about how must be carried on the delicate neurons and synapses time constant tuning, and what are its consequences on the network activity. If we decrease it we can have high temporal resolution, catching very fast dynamics happenings on the input signal, but we reduce the amount of memory we can get from the network. On the other hand, increasing its value we are more insensitive to the local dynamics of the signal, but the activity can get much more inertial behaviour, hence be influenced by the past neural activity.

Comparison with no recurrent connections network

The previous analysis addressed the study of reservoir dynamic memory variations according to the neurons and synapses timing properties. We have decided to extend the analysis by looking at the impact of the connections on the exploited memory dynamics. To achieve so, we performed the same steps as before, but with a reservoir where the recurrent connections has been removed. The resulting waves, compared with the previous ones, are depicted in fig. 7.6.
From the figure we can notice that the green traces, corresponding to the dynamical behaviour of the reservoir without connections, decay faster than the one coming from the recurrently connected reservoir. Since the only difference between the latter and the former is the presence or not of the connections, such results imply that they are the key element responsible for the difference. The corresponding new time inertias are:

- Analog wave with tau = 600 ms : 3 s
- Analog wave with tau = 600 ms : 1 s
- Analog wave with tau = 600 ms : 0.25 s

These values denote how, even if the connections has a strong impact on the dynamics of the network, we can still get good influence even in absence of them. Even if such result seems creating doubt about the real benefit of adding this complex pattern of connections,

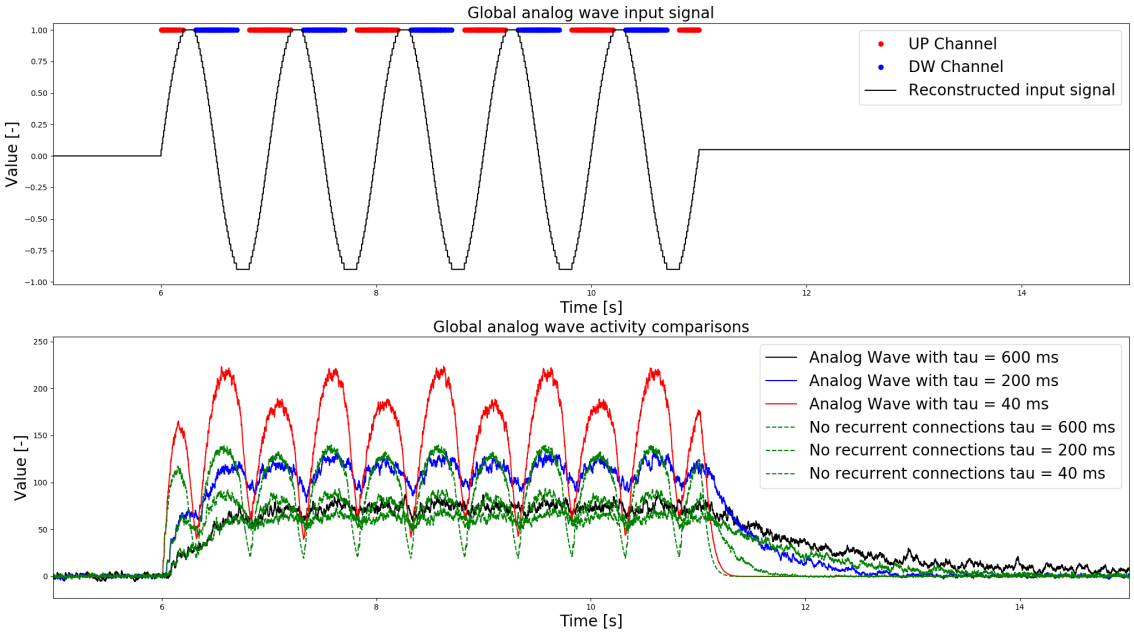

Figure 7.6: Detailed comparison of reservoir inertia with recurrent connections (black, blue and red) traces and without connections (green traces)

we can say that their usefulness does not lay only on the network memory increasing, but in the ability of creating complex non linear transformations on the inputs, that allows an easier recognition. Such concept is not only reserved on the topic of RNN, but can be extended to every neural network structure: when we connect a neuron to others, we are adding computation and transformations on the input signal dynamics.

7.2 First task results

Introduction and motivation
Such task is based on the one addressed by Xiaying Wang in his work [10], where the author tried to solve it using a software simulation of spiking neurons reservoir. Since such research is at the foundations of the thesis work, it was a reasonable starting point trying to achieve with DYNAP-se the same results obtained in software level.

The task, whose general characteristics are explained in deep in section 6.4.2, it consists on the classification of jittered analog sine waves at two different frequencies.

The reason for adopting jitter, instead of inject signals with simple white noise, resides in the idea that the latter one can almost always be filtered away. On the other hand, according to the way we construct spikes (explained in section 6.3.2), jitter can be obtained only from an alteration of the "normal" shape of the signal. Such variations cannot

be filtered away, because are inherently part of the signal itself. They must be accepted as they are and we have to designing systems and architecture insensible to their influence.

In the case of ECG signals, we can't expect a clean and stable wave at every period, but it will be certainly subjected to slope changes, that will directly affect the frequency patterns of the spikes. Noise, on the other hand, does not directly influence the spike pattern, but must pass through the conversion threshold algorithm, leading to a difficult prediction of its impact.

Results description

For what concern the classification results, without much effort we could get a accuracy of 100% over different iterations of the generated datasets. In fig. 7.7 is depicted a snippet of the chart used to analyse the network performances and, in case of errors, understand the reasons. In this picture is possible to see how all the effort that we have put to realise the complex automatic analysis structure explained in section 6.3 has been repaid. At the top we have the input signal shape reconstructed from the input spikes (the red and blue dots), they too retrieved from the output activity recording. In the middle we have the reservoir raster plot, while at the bottom we have the prediction of the network and the target values.

The prediction is sampled at the end of the sample time slot of validity, and its value is determined according to the threshold: if higher or equal, it is considered to be 1, and vice versa.

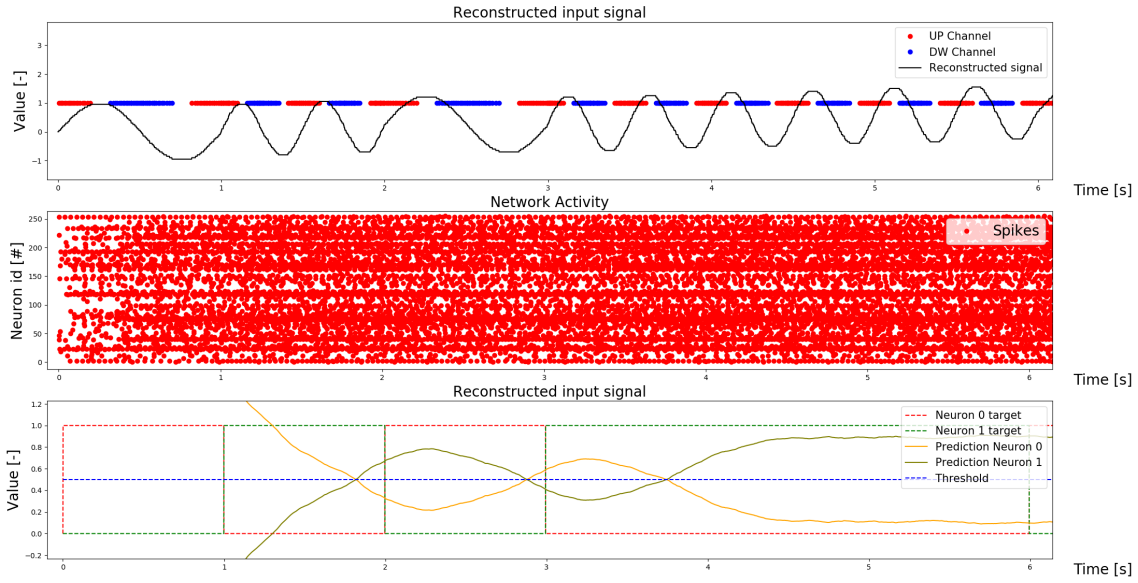

Figure 7.7: 6 seconds snippet of a prediction evaluation graph for the high time constant network. It can be seen the input reconstructed signal (top), the network activity (middle), and the prediction (bottom)

The input signal shape is a the sequence of the test samples which, for the first task, are sinusoids at two different frequencies, 1 Hz and 2Hz. Even if it seems, from the input signal shape, having a variable offset, because moving on the vertical direction, that is the effect of the jitter on the reconstruction algorithm. When the input signal changes type, we can notice that the reservoir activity slowly adapts to its dynamics, confirming the slow time constant. A further confirm of this fact can be noticed by the output prediction , which slowly moves toward the correct value, and reach it at the end of the sample time. It can be noticed that the prediction of the first sample is diverging from the boundaries. Such phenomena takes place due to the influence of the initial conditions: since the time constant of the network is slow, it need time to adapt at the signal dynamics, producing no valid results for the first sample.

Since good performances could be achieved without particular difficulties, we supposed that the task for which the reservoir has been used was not hard enough to test its real computational power. To confirm this theory, and understand if the obtained performances were a result coming from the high temporal constant of the network, we ran an equal experiment but changing its properties. The time constant value, using the guidelines developed in section 7.1, was reduced of one order of magnitude with respect to the previous one, at about 40ms. Even with this change, the resulting performances of the network still reached a maximum of 100%. In fig. 7.8 is present a snippet of its activity while performing the task.

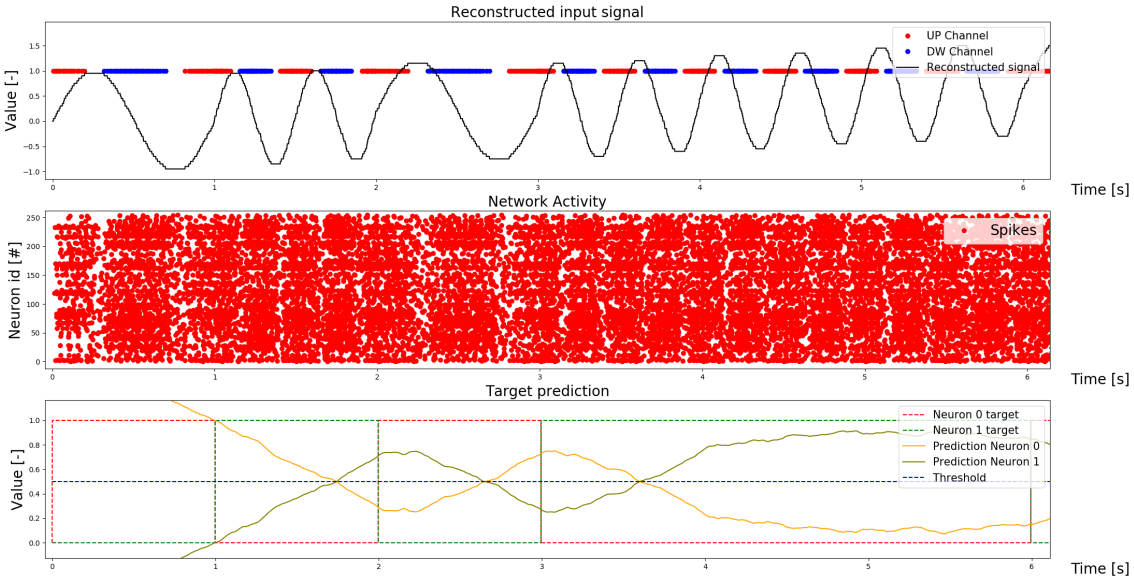

Figure 7.8: 6 seconds snippet of the prediction evaluation graph for the low time constant network. It can be seen the input reconstructed signal (top), the network activity (middle), and the prediction (bottom)

Comparing the activity of this last network with the one retrieved from the previous one,

we can clearly see the effect of the time constant different, that we described in section 7.1. It is clear how the former reservoir faithfully follows the input signal dynamics, while the latter one it doesn't. However, even with the lack of memory given by the low time constant, the ability to solve the task didn't change.

From this analysis it was clear that the task of jittered sinusoidal signals, designed in the way we have done, does not require memory to be solved. By looking at fig. 7.9, we can see that there is a clear division between the activities with respect to the two types of inputs. Hence, for a learning algorithm is easy to find a set of weights able to map the two different activities into the final required targets.

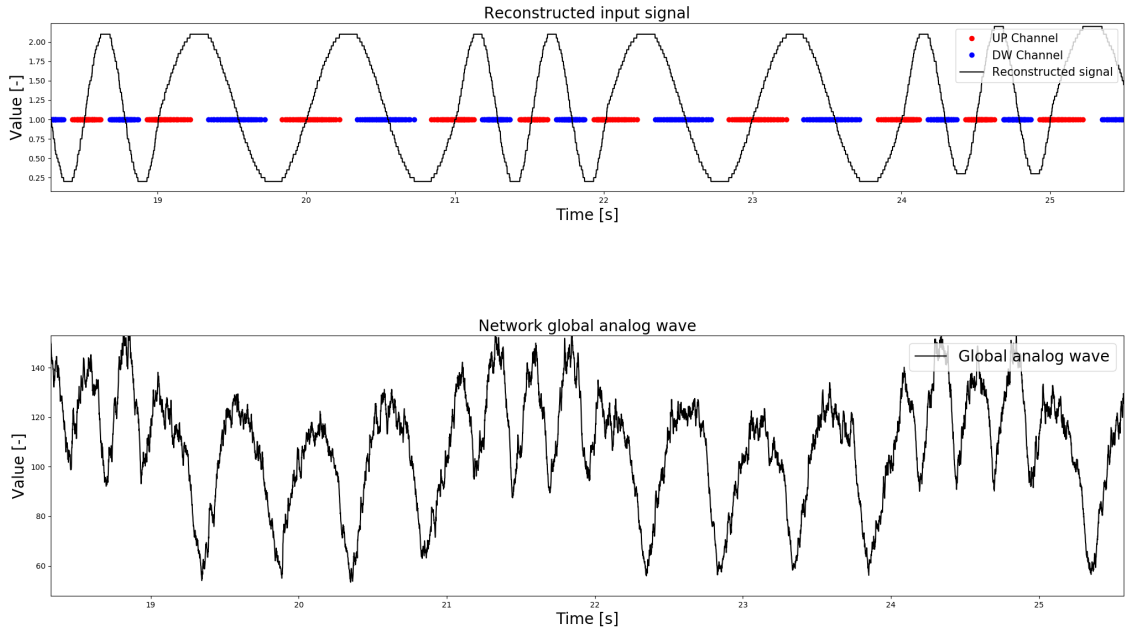

Figure 7.9: Global analog wave activity for the low time constant reservoir network

7.3 Second task results

Introduction and motivation

In the previous section we have demonstrated the proper functioning of the framework created to provide offline training and testing capability to neural networks in DYNAP-se. We applied such framework to the reservoir network, solving the classification of sinusoidal signals and achieving good performances. However, on the other hand, we also discovered how the difficulty of the designed task was not proper adjusted to test the reservoir properties. From that point we started asking ourselves how could we change the input signals in order to increase the difficulty of the task. Usually, in the neural network world (but not only), memory is needed when trying to distinguish between two entity that are equal in a certain time interval, but were different in the past. Following this philosophy, we build

a new task, explained in section 6.4.3, which is based on the framework developed with the first task, but with changes that would make it more complex, giving us a way to test the network memory.

Results description

The complexity of the resulting design was high and it was not easy to overcome. The most complicated part of the task is located at the second part of the disrupted signal (Signal 1). That is the time when the memory of the reservoir has to remember the previous perturbation and act in order to push the classification result toward Signal 1, avoiding the natural decaying in favour of signal 2.

In fig. 7.10 we start showing a snippet of the behaviour of the low time constant reservoir. We can notice how, as happened with the previous task, the network follows faithfully the dynamics of the input signal. Such behaviour create advantages when it has to quickly discover the 2Hz section between 2 pair of 1Hz sinusoidal inputs but, as long as the perturbation is gone, the network activity rapidly decays, and so does the prediction toward the wrong classification result.

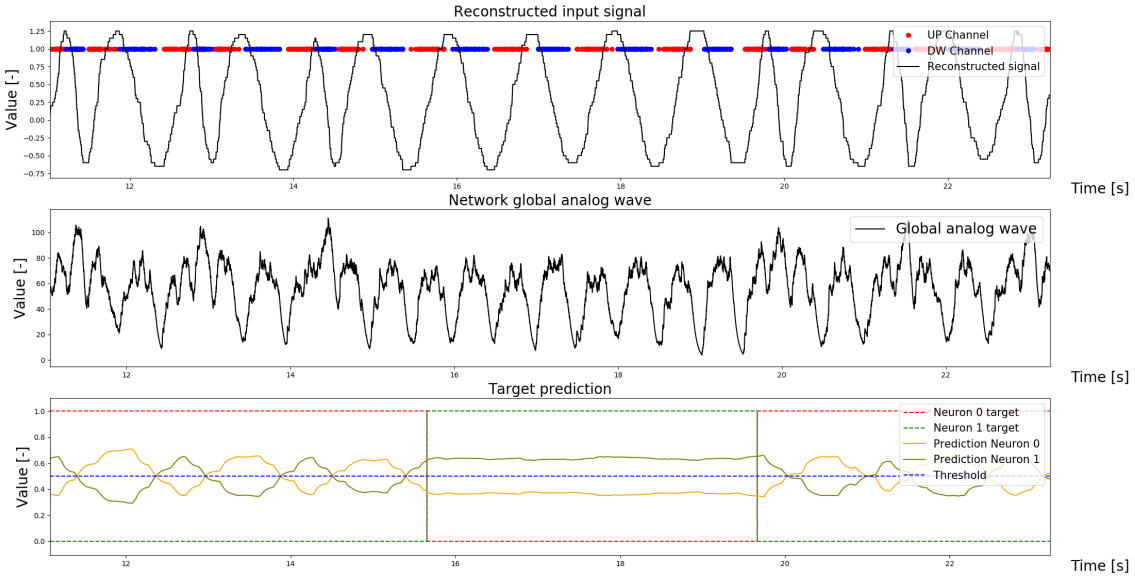

Figure 7.10: 12 seconds snippet of the second task prediction evaluation graph for the low time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

We can even notice, looking at fig. 7.11, that the network has less certainty even for the classifications that were pretty easily performed in the previous task, because the prediction traces are closer to the threshold value. The reason for such drop must researched on the training phase. During this phase, the regression retrieve the network activity at different steps and associate it with the correspondent output target values, in the attempt

to find the weights that match best. It can be noticed, by looking at fig. 7.10, that there are time intervals (for instance from 15 to 17 seconds) where the 1Hz sinusoidal signal must be trained to obtain two different outputs. This is the most crucial moment for the reservoir, where memory is required. If the reservoir activity it's the same for both input signals, the training algorithm would get confused, because asked to find weights able to map the same activity to different results, which is not possible. This results on an higher uncertainty on the prediction, that will get closer and closer to the central threshold, increasing the possibility of errors. With this network we got a maximum of 49% accuracy over the created datasets.

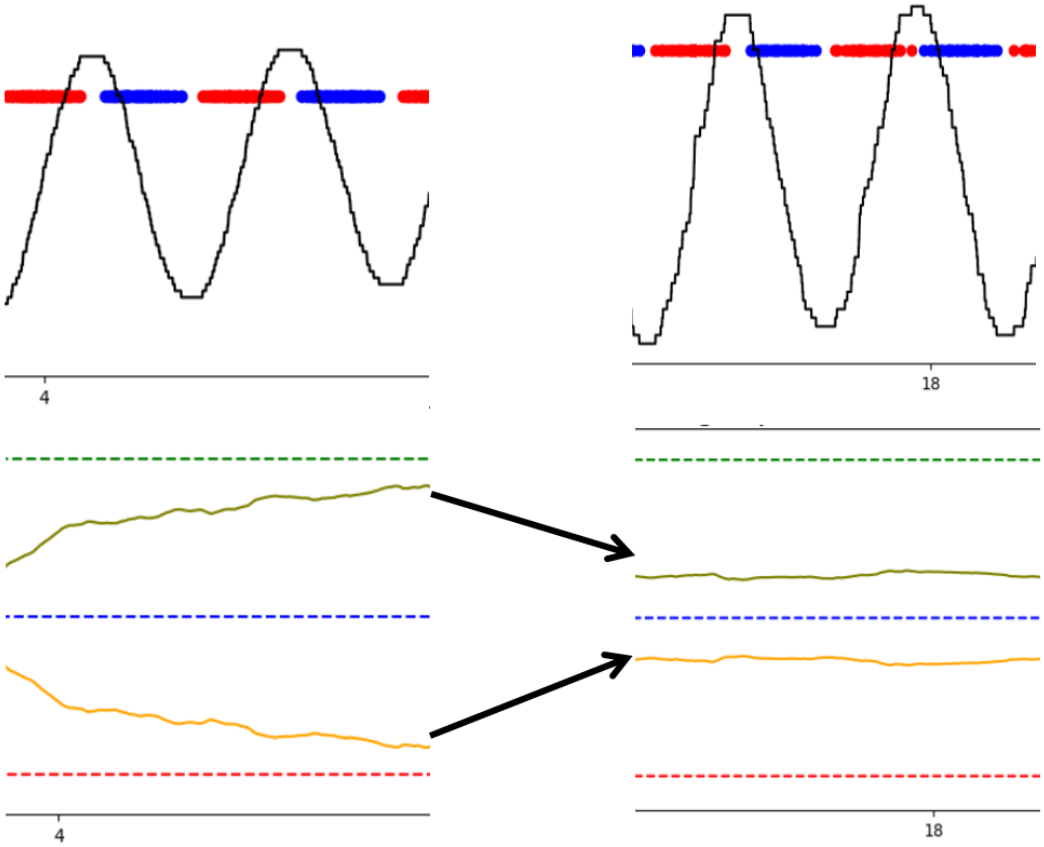

Figure 7.11: Comparison between prediction for different tasks but similar conditions

In fig. 7.12, instead, is depicted the behaviour of the long time constant reservoir. As can be seen from the picture, the prediction is clearly defined for the parts where the signal has a different shape in both samples, while tends to fade where they become similar. With respect to the previous network behaviour, in this case the memory of the reservoir tends to have a residual activity for a certain period after the 2Hz perturbation ended. That difference of activity is exactly what the training algorithm needs in order to properly distinguish between the two samples, even if the input signals have the same shape for

that particular period. With this type of network we got a maximum of 94% accuracy over the created datasets.

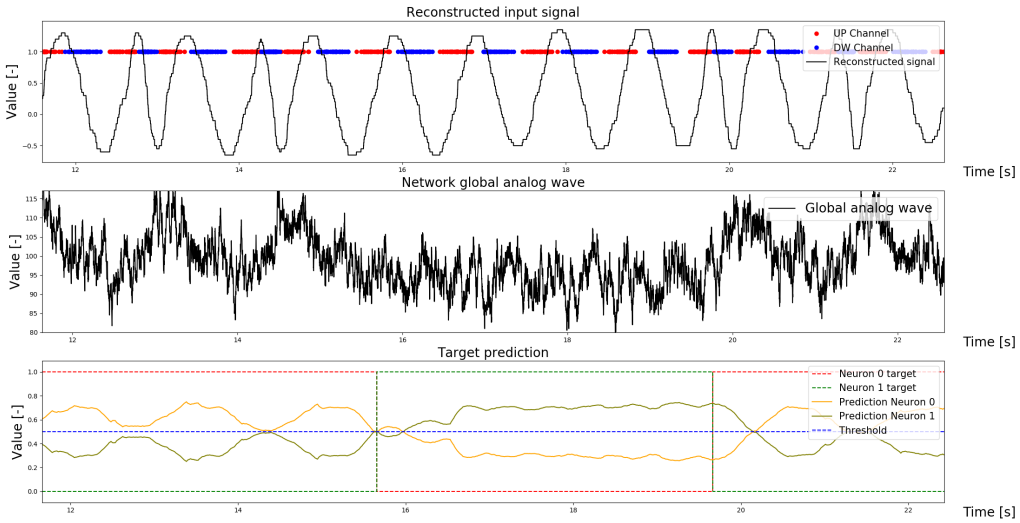

Figure 7.12: 12 seconds snippet of the second task prediction evaluation graph for the high time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

A further demonstration of what we have said before can be seen by looking at fig. 7.13, were we put the analog wave traces for the two different network time constants in comparison, but averaged in order to exhibit the general behaviour of the curve. We can notice from the figure, in a more neatly way, which is the difference in activity memory after the 2Hz perturbation.

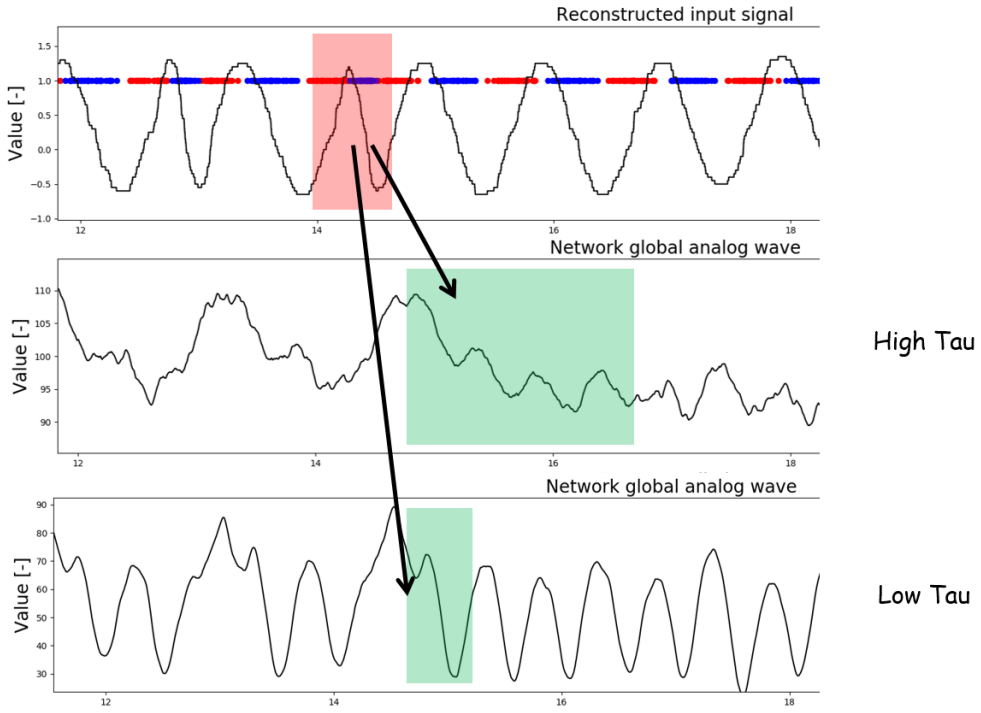

Figure 7.13: Comparison between smoothed analog wave activity

7.4 Third task results

Introduction
In the previous sections we have shown reservoir computational characteristics dealing with two tasks: classification of signals at different frequencies and disrupted signals, in the world of sinusoidal waves. With the following task we partially leave that world, getting the analysis toward the ECG signals classification.

Results description
ECG signals have a dynamics that resemble the one of sinusoidal waves: both of them have a oscillating behaviour but, while the latter one spread such oscillation on the whole period, the formed one concentrates it in a small part. The result is that ECG signals have periods in the order of seconds as the sinusoids we have been studying, but dynamics that lasts few hundreds of ms. This fact has strong consequences on the behaviours of the input stimuli supplied to the network.

ECG signals, in addition to the concentrated central oscillation, is characterized by two other smaller "bumps", called P and T waves. Their slower and smaller dynamics, with respect to the central QRS complex, complicate the threshold value selection for the signal to spike conversion. Keeping the same value we used for the first two tasks, as visible in

fig. 7.14, would considerably limit the influence of the P and T wave over the network, making it difficult to detect possible perturbations.

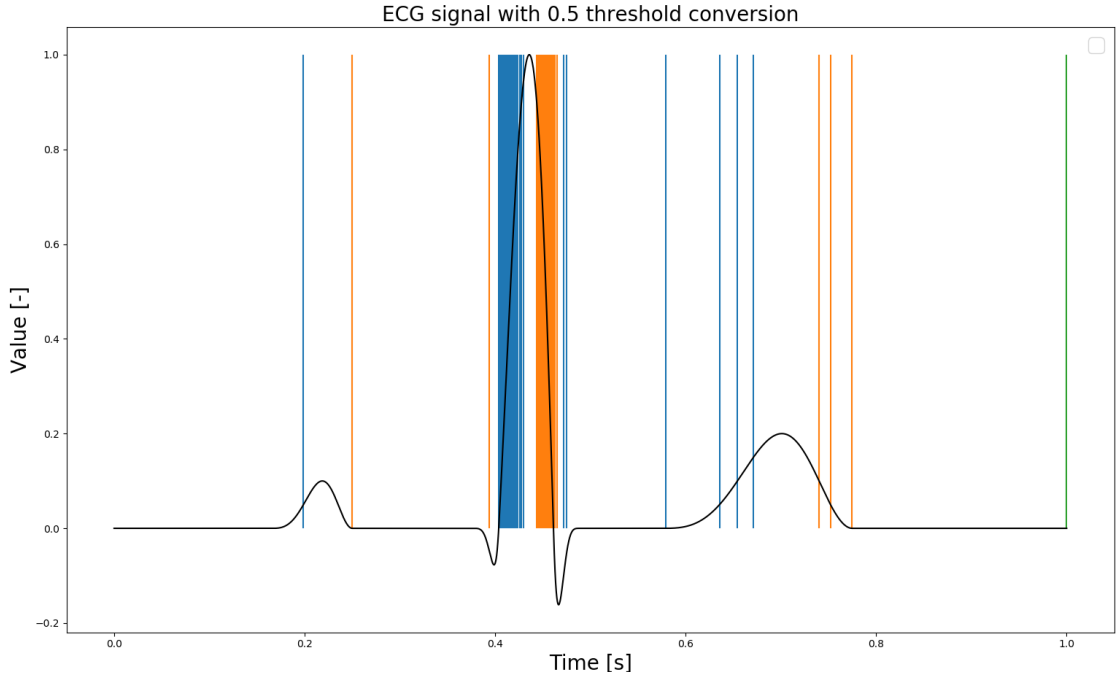

Figure 7.14: Normal ECG signal conversion to spikes with threshold = 0.5

On the other hand, the same threshold would be enough for the QRS complex, because it leads to an input signal dynamics close to the one used for the previous tasks. Since we will try to perform classifications not only on the QRS complex, we decided to lower the threshold in order to guarantee for P and T wave a dynamical behaviour similar to the one adopted in the previous tasks. The consequence of this choice is the consistent increase of the QRS activation, that could arrive at frequencies of kHz, as can be seen in fig. 7.15
As we have described in section 6.4.4, ECG classification can be done with respect to a various number of possible perturbations: Missing T wave, missing QRS complex and missing P waves. Everyone of such perturbations acts in a different point on the ECG signals and is produced by different problematic in the hearth activity. In the following we will describe, for each input, which impact he had on the network activity and the performances that we could get for different time constants of the network. With the knowledge we have gained by the previous analysis, we would make hypotheses on the expected behaviour of the network, demonstrating or confuting them by describing the analysis that we have performed.

QRS complex missing

The QRS complex is the wave present in the middle of the ECG signal, and it's the one usually detected in order to find out which is the heart bit of a person. It's the sharpest and highest peak, and exploit the fastest dynamics inside the ECG signal. For this reason,

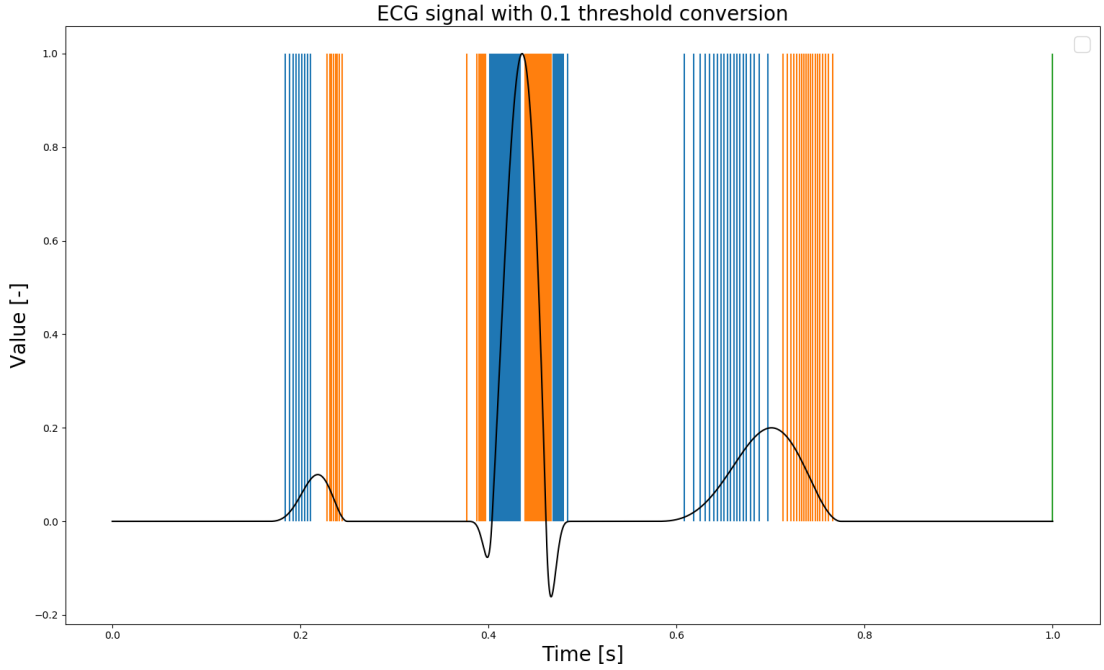

Figure 7.15: Normal ECG signal conversion to spikes with threshold = 0.1

according to the algorithm we use to convert signals into spikes, it is also the part of the signal that would have the maximum generated spiking frequency, by looking at the conversion of fig. 6.19.

The task consists on the classification of two types of signals: a normal ECG trace from one where the QRS complex has been completely removed. The graph in fig. 6.21 show an example of the perturbation effect on the signal.

Taking into account the high input activity brought by the presence of QRS complex, we supposed that the task would have been easily solved by the network. The high difference of the input dynamical activity in the samples would have been enough for the training algorithm to be able to distinguish between them.

In fig. 7.16 we show a snippet of the performance evaluation for the QRS missing task. In the first row we can that some of the ECG signals presented in input to the network have the QRS part (samples from 4 to 8 s) while some others do not, as the ones from 8 to 10 s. The offset at which the reconstructed signal is subjected derives from the jitter applied to the input signal, that makes the reconstruction harder. In addition, the dynamics of the QRS complex is so fast that can't be correctly followed by the input visualiser neuron. These last ones collect multiple input spikes before firing, leading to a decreasing of the reconstructed input signal range. The maximum obtained performances over the created datasets reach a value of 98%.

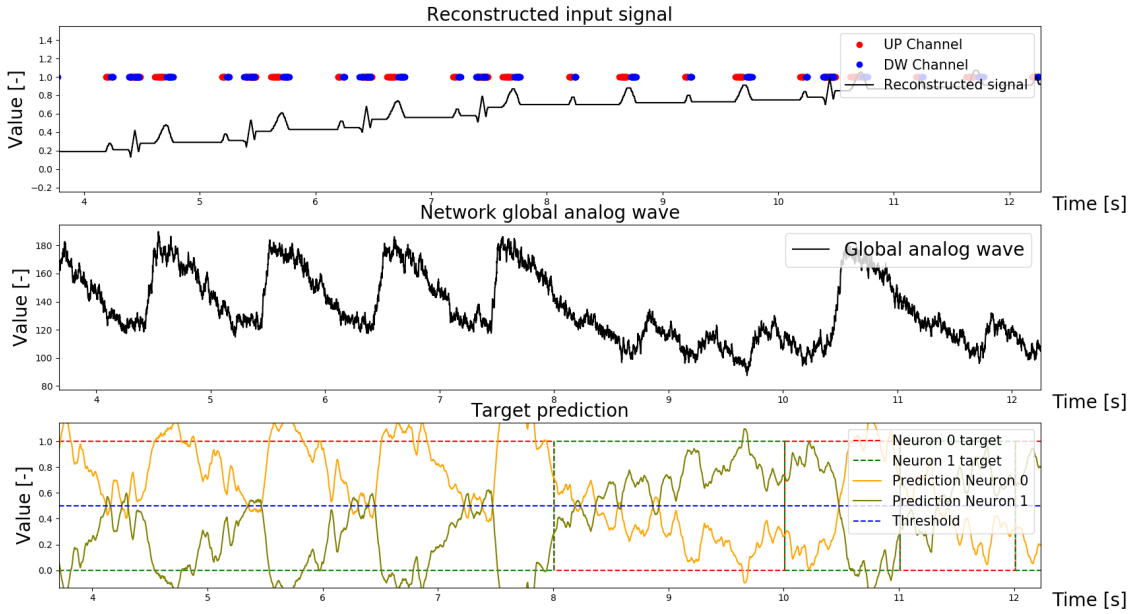

Figure 7.16: 12 seconds snippet of the missing T wave task using the high time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

It is evident how, every time the QRS complex is presented to the network input, it corresponds to a sharp increasing of the network activity (middle picture), which then is associated to a correct high value prediction for neuron 0 (bottom picture). Such behaviour testifies how the training algorithm exploits the high activity introduces by the QRS complex in order to make the correct prediction. The consequence is that this last one follows pretty faithfully the same shape of the total network activity, during the rising or falling time. We have to remember, in fact, that the low activity condition, as happens for example from time 8 to 12 s, is associated to the absence of QRS complex, which corresponds to a predicted value equal to 0 for neuron 0 and 1 for neuron 1. This means that every time the activity tends to slowly drop down after the sharp increasing caused by the QRS complex, the output prediction layer start associating it as a similar condition to the one with absence of QRS complex, leading to a decreasing of the prediction. However, the residual memory of the network make sure that the prediction does not drop below the threshold, before that a new sample arrives. Such behaviour, for example, does not happen when lowering the time constant of the network, as visible in fig. 7.17. While the noisy prediction it becomes well defined in the time intervals overlapping with the QRS complex, just after it directs toward the opposite and wrong value. The maximum obtained performances over the created datasets reach a value of 52%.

Missing T wave

The T wave is the last wave taking place on the ECG signal, just before a new period

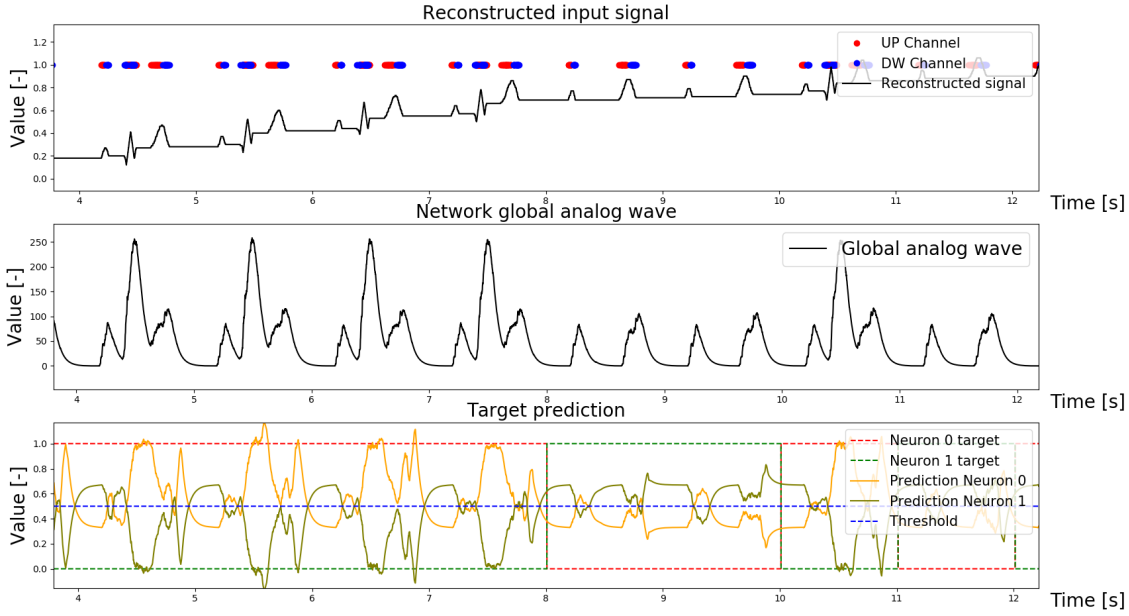

Figure 7.17: 12 seconds snippet of the missing QRS wave task using the low time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

starts. The task we studied is the detection of absence of such part on the complete ECG wave, as shown in fig. 6.20. We supposed that, since the signal is at the end of the whole sample, it has a big advantage for the classification performances, because it is close to the point where we perform a decision from the output layer behaviour. The activity that derives from the prediction is not influenced by any other dynamics of the input signal. In other words, we expected that the missing T wave task would have had the best chances to have good prediction performances. In fig. 7.18 we can see a snippet of network activity and the prediction performed over the task.

Comparing the activity of the network with respect to the previous task in fig. 7.16, we can see that the general shape of the analog wave is very similar to the snippets were the QRS complex was present, while there is a huge difference when this last one is absent. Such fact shows how the presence or absence of the QRS complex is much more influent than the one of the T wave. In fact, by looking at fig. 7.18, is possible to appreciate the T wave influence by looking at the decaying of the analog wave after the QRS complex. When the T wave is present, for example for samples 1, 2 and 3, the dynamics tends to have a slightly more roundish shape with respect to the samples when T wave is not present, like the 4th one. The change is really small, testifying that the P wave have some influence, but surely much lower than the one given by QRS complex.

Such fact is the reason for which the prediction is noisy and shows high uncertainty.

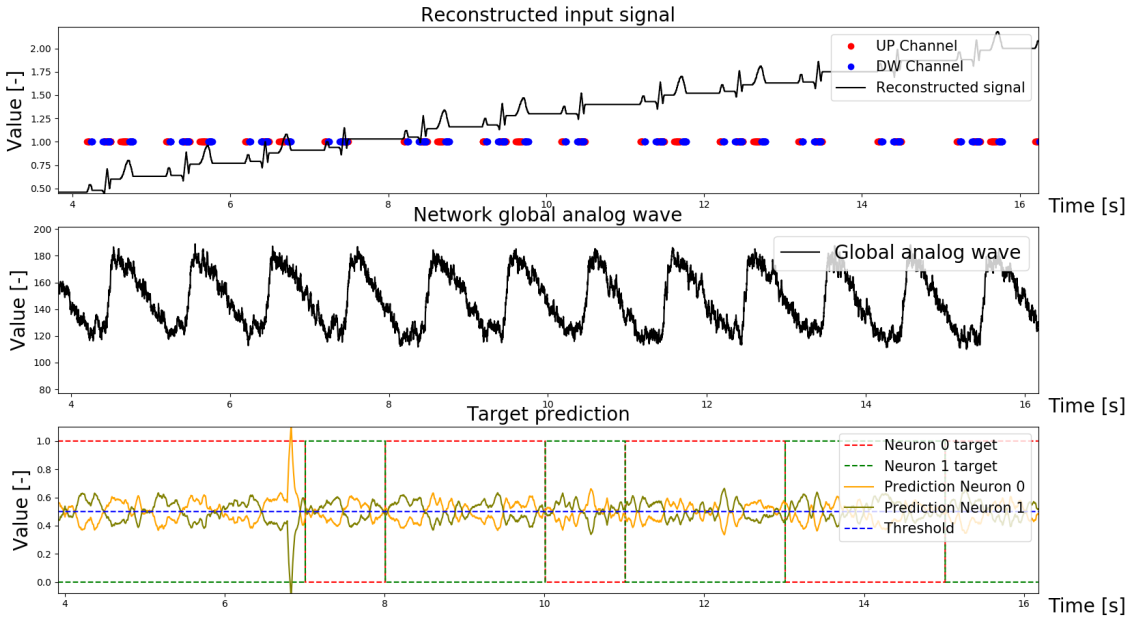

Figure 7.18: 12 seconds snippet of the missing T wave task using the high time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

As we have described in section 7.3, when we train the weights for different targets values by using similar base network activities, it is very difficult for the training algorithm to find a way in order to classify correctly the two inputs. The same is happening here: the low influence that the T wave absence gives to the network activity is such that the final resulting prediction is not certain about which input signal has been provided. The maximum obtained performances over the created datasets reaches a value of 74%.
In fig. 7.19 we show, as done for the previous tasks, the activity of the low tau network. Comparing his behaviour with respect to the high tau one, we can see that the prediction amplitude is, in some parts, far from the threshold value, showing high certainty on the classification for that period of time. This fact derives from the low memory exploited by such network. Its activity reduces pretty quickly after the QRS complex, making the influence of the P-wave more evident with respect to the high tau network. This is demonstrated by the fact that the T-wave absence, for instance in the 4th sample, is correctly detected. Despite this fact, however, the low memory of the network makes the prediction fall immediately on the wrong direction, bringing the network to have a lower performance than the more stable high tau one, with a maximum of 50%.

Missing P wave

With the previous two tasks we have discovered how the reservoir is able to correctly detect disturbances on the QRS complex, but fails when have to deal with weaker signal, as with the T-wave. A natural continuation of the previous analysis is to try to show what

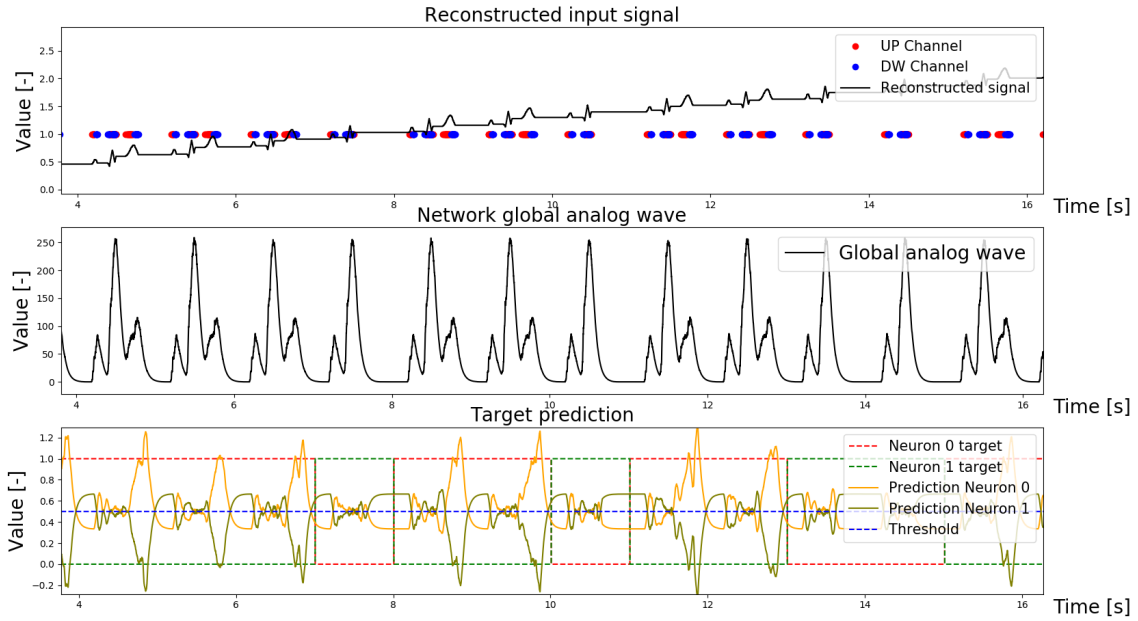

Figure 7.19: 12 seconds snippet of the missing T wave task using the low time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

happens when bringing to the limit the request for high sensibility to the inputs and long memory. To do so, we tried to make the classification of the P-wave, which is the very first part of the ECG signal dynamics, preceding the QRS complex.

An example of the P-wave shape is present in the fig. 6.19, showing the relative conversion in spikes. In fig. 6.22, instead, is depicted the perturbed signal object of classification. It can be noticed how the P-wave shape is even smaller than the T-wave one, hence the activity associated with it is weaker. This is the first motivation over which is based the hypotheses that such task is more complex than the ones we have faced before. The second reason, instead, is related to its position. The P-wave is, as we said before, at the beginning of the ECG signal, meaning that an eventual classification should have a memory lasting for the whole duration of the signal, which is a feature very difficult to achieve.

In fig. 7.20 we show, as for the previous tasks, the activity of the high tau network. It can be noticed, by looking at the global analog wave, that the P-wave has a small influence on the overall network activity, slightly increasing the lowest value reached by the analog waves. Such influence, by looking at the target prediction graph, proves to be not enough to allow the training algorithm correctly identify the perturbation. The maximum obtained performances over the created datasets reach a value of 62%.

In fig. 7.21 we show, as for the previous tasks, the activity of the low tau network. Comparing his behaviour with respect to the high tau, we can see that the prediction

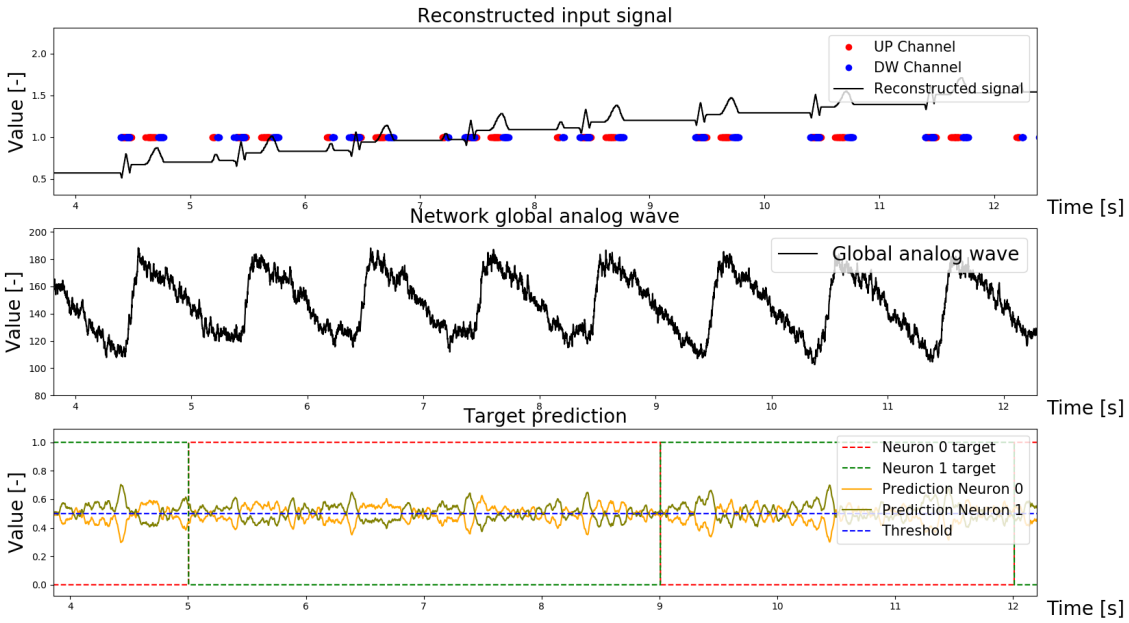

Figure 7.20: 8 seconds snippet of the missing P wave task using the high time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

amplitude is closer to the target ones, far from the threshold value. The low memory exploited by such network, again, makes the activity react much widely over the P-wave input influence, making it more evident with respect to the high tau network. This is demonstrated by the fact that the P-wave absence, for instance in the 1st sample, is correctly detected. However, the low memory of the network makes the prediction fall immediately toward the wrong values, leading the network to have worse performance than the high tau one, equal to 50%.

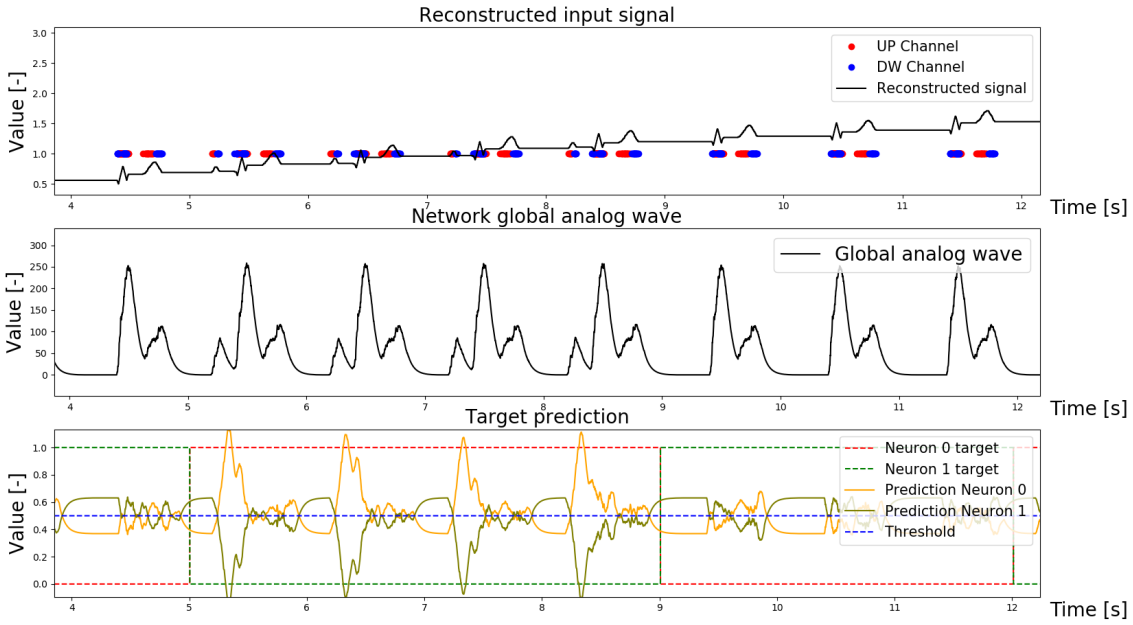

Figure 7.21: 8 seconds snippet of the missing P wave task using the low time constant reservoir network. It can be seen the input reconstructed signal (top), the network global analog wave (middle), and the prediction (bottom)

Chapter 8

Discussion

The presented thesis work has shown the results of a first approach to realise an application with an analogic neuromorphic hardware. Although the performances achieved are strongly influenced by software environment, because obtained giving artificially created ECG signals, and training artificial neurons, it constitutes a good basis for future analysis aimed to the creation of hardware neuromorphic working classifiers. The achieved performances can't be compared with similar architecture because, at the time when the thesis work has been done, there were no public reports claiming similar applications on neuromorphic hardware. Hence in this section i will focus the discussion on the most important points over which new analysis should concentrate such to developed future architectures able to accomplish the task in a better way.

The thesis work has been focused on the implementation of recurrent neural network following the rules described by Wolgang Maass. For sure this has proved to be a good starting point, because some important considerations and results were already achieved in software. On the other hand, such approach still has a lot to be discovered and explore, because there are no standard frameworks that are able to guarantee the correct functioning and optimization of the performances over the required task, as it happens for artificial neural networks. A good step on this direction, and for a future implementation in real world applications, would be to develop a scientific and mathematical formalisation able to move from task definition to the network design and implementation in a reasonably efficient way. A possible step in this direction would be the exploration of new reservoir architectures different from the adopted one, whose structure and connectivity would involve less probabilistic parameters. An example would be a modular structure that can be composed in order to solve specific tasks, where the probability parameters are substituted by equivalent deterministic ones. Such solution would give more predictability to the outcome of the network, allowing parametric analysis aiming to study their impact over the network dynamics and final performances, if present. Such work should be accompanied by the development of reliable testing framework able to evaluate the main characteristics and performances of the implemented networks outside the close application target, but in a more general context. In this way it is possible to know if the approaches followed to

improve the network architectures are valid or not.

Leaving the more general area of reservoir computing, and focusing more on the specific work that has been carried on, a promising continuation would be the realisation of the whole network completely in hardware. The part currently missing on the chip is the most important, but also most difficult to realise, which is the output layer, currently implemented in software. In fact such process is intrinsically really complex to realise on DYNAP-se because it does not provide any learning features directly implemented on chip. All the possible learnings that can be applied over the device must be emulated in software by writing low level code for cAER control program. Since the synaptic bias weights present in DYNAP-se are limited in number (one for each core, for a total of 16), they lead to the impossibility of training networks with enough complexity to overfill them. The training process so must take another approach. It can be realised by dynamically change the number of CAM for the same connection, hence strengthening or weakening it. Such approach would still be hampered by the connectivity limitations of the chip, which correspond to a maximum of 64 inputs or each neuron, but at the moment is the only way that can be followed in order to realise supervised training on DYNAP-se chip.

Focusing more in detail on the software approach used to perform the training and testing, we can demonstrate that, from a certain point of view, is not far away from a possible hardware representation. The trainig and testing, in fact, is based upon the analog wave created from the network activity, by convolving it with a kernel simulating the low pass filter action of a synapse. The analog waves, in other words, can be imagined as the output current coming from a synapse, in our case the ones connected to the output neurons. When we apply regression algorithms we are trying to find the weights list such that, combining them with the different input analog waves, it creates an output that should match the task target one. This action is similar to the one performed by a neuron which combines the input coming from the different synapses and produces an output that, filtered, can be associated to the prediction result. Modifying the impact of the different synapses by programming the CAM cells, is possible to guide the neuron output activity toward the desidered one. In summary, we demonstrate that the created software model is not far from a possible hardware representation, but still need effort to find the correct way to move between the two representations.

We will conclude the discussion suggesting a possible architecture where a neuromorphic chip classifier, like the one developed, can be successfully applied. The big advantage of neuromorphic chips is the ability to use the event-based nature of neurons in order to solve complex problem with limited power consumption, feature that makes them fit for real time applications where power is the main concern. On the other hand, neuromorphic chips are recently developed devices and yet not able to guarantee high reliability or performances comparable to state of the art digital system. Hence they can't be used as the only source of elaborations, but must be paired up with digital hardware in order to build a low power digital system powered up by neuromorphic computation, as the one depicted in fig. 8.1. The latter one would work as a low power always on intelligent

sensor, performing real time classification of patterns over the signals of interest, in our case ECG signals. The digital chip, instead, would hangle the communication with the external digital infrastructure but, in normal operating conditions, it would stay in a low power sleep state. In this way the elaboration is carried on almost entirely the time by the neuromorphic system that, considered the low power consumption, does not need to be turned off. When a condition for which it has been trained to respond it presents, a specific event can be used to wake up the digital processor and let him handle the elaboration of such situation, which can include saving it in memory or performing standard digital elaboration. In other words, even if the digital processor is a digital synchronous and deterministic device, in this system is working in an event-based manner, because realise computation only when important conditions raises. A system build in this way could surely bring to high power saving and, consequently, can be applied in applications where long lasting battery powered system is a main issue.

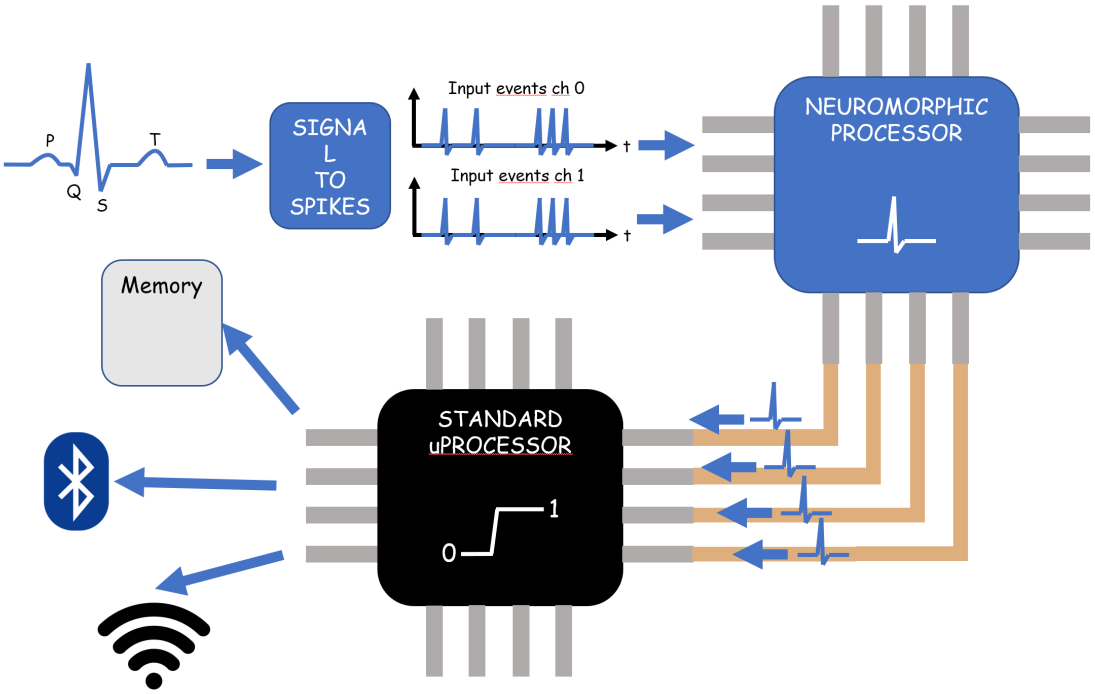

Figure 8.1: Schematic representation of a possible application for a digital system powered by neuromorphic capabilities

Appendix A

Reservoir network parameters

The designed reservoir network follows the structure described by Maass in [9]. Here it will follow the collection of the most important parameters over which the used network has been built.

The reservoir is made of 3D neurons columns, each one counting 4x4x16 elements, for a total of 256 neurons, which is the dimension of a whole core in Dynap-se chip.

The network is divided in 80% excitatory neurons and 20% inhibitory ones.

Connectivity probability is dependent on the distance between two neurons, and is defined in the following way ():

$$P = C \cdot \exp(-(\frac{D(N1, N2)}{\lambda})^2) \tag{A.1}$$

Where λ tune the mean number of connections and mean distance between connected neurons, and has been set to value 3.

C parameter is another degree of freedom allowing to have different probability distribution for different types of connections:

- Connections excitatory to excitatory $C_{EE} = 0.3$
- Connections excitatory to inhibitory $C_{ET} = 0.2$
- Connections inhibitory to excitatory $C_{IE} = 0.4$
- Connections inhibitory to inhibitory $C_{II} = 0.1$

The distance between two neurons is calculated with the Euclidean formula:

$$D(N1, N2) = \sqrt{(X_{N1} - X_{N2})^2 + (Y_{N1} - Y_{N2})^2 + (Z_{N1} - Z_{N2})^2} \tag{A.2}$$

Where X, Y and Z are the coordinated of two neurons N1 and N2.

The input to reservoir connectivity probability is equal to 0.3, meaning that only 30% of reservoir neurons are connected to input.

The connectivity weights, intended as multiple programmed CAM for a single connection, are obtained from a Gaussian distribution with mean equal to 2 and mean equal to 0, however only the positive values are taken as valid.

Bibliography

[1] T. C. Südhof, R. C. Malenka, "Understanding Synapses: Past, Present, and Future" in *Neuron*, v. 60, n. 3, p. 469–476, July 2018.

[2] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, J. S. Plank, "A Survey of Neuromorphic Computing and Neural Networks in Hardware" in *CoRR*, v. abs/1705.06963, p. 2531–2560, May 2017.

[3] R. Douglas, M. Mahowald, C. Mead, "Neuromorphic Analogue VLSI" in *Annual Review of Neuroscience*, v. 18, 1995.

[4] G. Indiveri, S.-C. Liu, "Memory and Information Processing in Neuromorphic Systems" in *Proceedings of the IEEE*, v. 103, n. 8, pp. 1379–1397, August 2015.

[5] Q. Ning, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses" in *Frontiers in Neuroscience*.

[6] S. Moradi, N. Qiao, F. Stefanini, G. Indiveri, in "A scalable multi-core architecture with heterogeneous memory structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)" August 2017.

[7] E. Chicca, F. Stefanini, C. Bartolozzi, G. Indiveri, in "Neuromorphic Electronic Circuits for Building Autonomous Cognitive Systems" September 2014.

[8] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events" in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, v. 47, n. 5, pp. 416–434, 2000.

[9] W. Maas, Thomas, T. Natschlager, H. Markram, "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations" in *Neural Computation*, v. 14, p. 2531–2560, 2002.

[10] X. Wang, in "Noisy signals classification using Liquid State Machine" pp. 1–22, July 2017.