Learning in Analog Spiking Neural Network with Floating Gate Synapses in Standard CMOS Technology

Tesi di Laurea Magistrale in
Electronics Engineering

Author: **Giovanni Camisa**

Student ID: 944734
Advisor: Prof. Giorgio Ferrari
Co-advisors:
Academic Year: 2021-22

Abstract

Spiking Neural Networks (SNNs) have become the most promising method to solve machine learning-based problems due to their biological and hardware plausibility and reduced complexity compared to Artificial Neural Networks (ANNs). In particular, the SNNs are the best candidate for real-time processing near the sensors, i.e., edge computing, because they can be implemented on extremely power-efficient dedicated hardware. However, the hardware implementable training algorithms for SNNs are still too immature to compete with ANN performance on real-world applications. After performing a comprehensive analysis of the learning techniques, this thesis work proposes an online training for SNNs specifically designed for floating gate CMOS technology. For the case study considered in this thesis, a simple analysis of a 3-axis accelerometer, the proposed method has proved to achieve a comparable accuracy with an ANN for edge computing. However, the SNN, when implemented on dedicated hardware, should be able to achieve power dissipation orders of magnitude less than the ANN one. The simulations and the analysis have been based on a neuromorphic chip in standard CMOS technology designed and implemented during previous thesis works.

Keywords: Machine Learning, Spiking Neural Network, Neuromorphic CMOS Technology, Accuracy, Edge Computing

Abstract in lingua italiana

Le Reti Neurali Spiking (SNNs) sono diventate il metodo più promettente per risolvere problemi basati sul machine learning grazie alla loro plausibilità biologica e hardware e alla loro ridotta complessità rispetto alle Reti Neurali Artificiali (ANNs). In particolare, gli SNN sono i migliori candidati per l'elaborazione in tempo reale vicino ai sensori, ovvero l'edge computing, perché possono essere implementati su hardware dedicati estremamente efficienti dal punto di vista energetico. Tuttavia, gli algoritmi di addestramento hardware implementabili per gli SNN sono ancora troppo immaturi per competere con le prestazioni delle ANNs sulle applicazioni del mondo reale. Dopo aver eseguito un'analisi completa delle tecniche di apprendimento, questo lavoro di tesi propone un allenamento online per SNN specificamente progettate per la tecnologia CMOS a gate flottante. Per il caso di studio considerato in questa tesi, una semplice analisi di un accelerometro a 3 assi, il metodo proposto ha dimostrato di ottenere una precisione comparabile a una ANN per edge computing. Tuttavia, l'SNN, se implementato su hardware dedicato, dovrebbe essere in grado di raggiungere ordini di dissipazione di potenza di grandezza inferiore a quello dell'ANN. Le simulazioni e l'analisi sono state basate su un chip neuromorfico in tecnologia CMOS standard progettato e implementato durante i precedenti lavori di tesi.

Parole chiave: Machine Learning, Reti Neurali Spiking, Technologia CMOS neuromorfica, Precisione, Edge Computing

Contents

Introduction

“I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think". The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous. If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words. The new form of the problem can be described in terms of a game which we call the "imitation game"”

Computing Machinery and Intelligence, Alan M. Turing, 1950

Turing was one of the first people to ever think about the real possibility of artificially recreating the human mind. The sole understanding of the human mind is one of the ancient questions that have always tormented philosophers and scientists. Nowadays, the incredibly fast development of new technologies in the integrated manufactory, artificial intelligence software and neuroscience may delude us that we are close to this milestone, but the truth is that there is still a long way to go to achieve a real understanding. We are still in the age of the "imitation game".

Artificial neural networks (ANN) were implemented due to the need for better performance on many tasks that are common to human beings, such as image recognition, video motion detection, and natural language processing. Even if, during the years, an impressive accuracy has been achieved, an enormous bottleneck in power efficiency and computational speed was found in Von Newmann’s architecture of the computer implementing the ANN. To solve this issue, today we are facing the rapid evolution of a new paradigm for information processing through artificial devices and circuits even more inspired by nature. However, even if these technologies are an attempt to emulate the brain, the lack of information in global interaction and not-spiking signals between neurons has lead to the implementation of algorithms with mixed approaches in between the mathematical optimization and the brain inspiration.

0.1. From Von Neumann bottleneck to neuromorphic computing

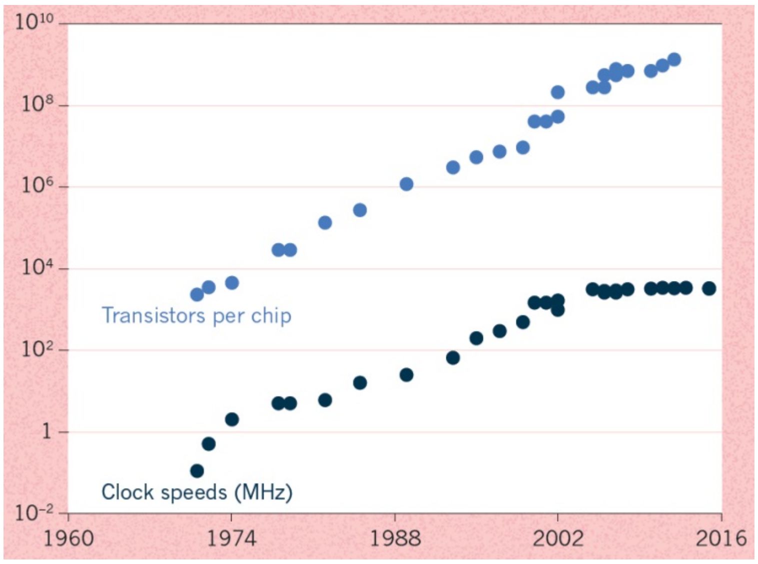


Figure 1: Transistors per chip and clock speed limited by overheating. The Moore's Law exponential trend is clearly visible. Figure from [57].

The rise in general usage of electronics is closely tied to an increase in the power and miniaturization of electronic components. It has been driven by "Moore's Law", which was stated by Gordon Moore, co-founder of Intel Corporation, in a famous 1965 paper [37] and then confirmed in 1975 [38]. This so-called "Moore's Law" states that the number of electronic components on an integrated circuit (IC) doubles every two years. This trend has proven to be true right up to our own time, and so semiconductor manufacturers have planned every technology node ever since. The "Law" has been revealed to be valid over the years, but it seems that it will not see the end of the decade 2020-2030 even with the current process innovations and design technology co-optimization (DTCO) [47]. The high number of devices per unit surface area leads to intolerable power dissipation, which is even worse considering the high operating frequency required for the top processors in the market today.

Power dissipation is a key problem in modern CPUs (central processing units) and this heat wall limits the performance of the systems to such an extent that the clock frequency has almost plateaued in the last 15 years, moving away from the previous exponential trend (Fig. 1). The problems for the future of processors are not only at the individual device level, but also at the architecture level; current systems are almost all based on

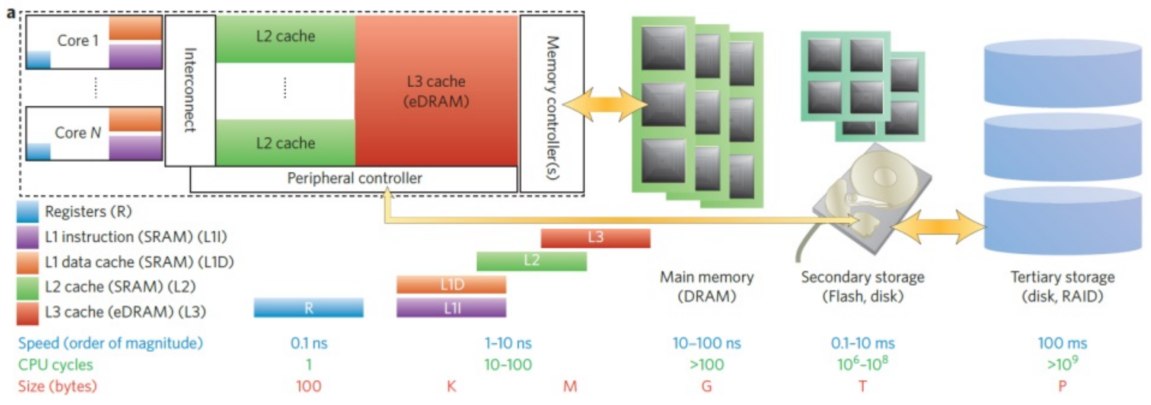


Figure 2: Representation of different physical memory. Their specifics are compared in terms of speed, CPU cycles and storage capability. Figure from [58].

the Von Neumann architecture, in which the computing unit and the memory unit are physically separated. This choice has proven successful as it provides a modular structure that allows different parts of the system to be worked on independently. Nevertheless, this approach is not suitable for the current market requirements, which have extremely high demands on speed and memory. The main problem lies in the different performance evolution of the CPU and memory units, with the former achieving far more speed than the latter.

This problem is even more crucial considering that modern systems are quite complex and based on a memory hierarchy (Fig. 2), with different levels. Near CPU is SRAM (static random access memory), a fast volatile memory that is accessed frequently and uses 6 FETs to store one bit, making it quite expensive and sparse. Below that is the DRAM (dynamic random access memory), a slower, denser, and cheaper volatile memory that can be considered a link between CPU and memory, as it is off-chip like the newest memory. At the very bottom is mass storage, whose most used type is flash memory, a very high density non-volatile memory that is also the slowest of the pyramid. The limit to the efficiency of this system (also called the Von Neumann bottleneck or memory wall) is that data must be shuttled back and forth between all these intermediate steps in order to be computed and stored. Since most of the memory units are outside the chip and the memory units are much slower than the CPU, this obviously has a major impact on the performance of the system [58]. This memory wall, combined with the heat wall and the end of Moore's law, raises serious concerns about the challenges facing the electronics industry in the near future, from AI to Big Data applications.

To solve these issues, new technologies able to process the data directly where they are

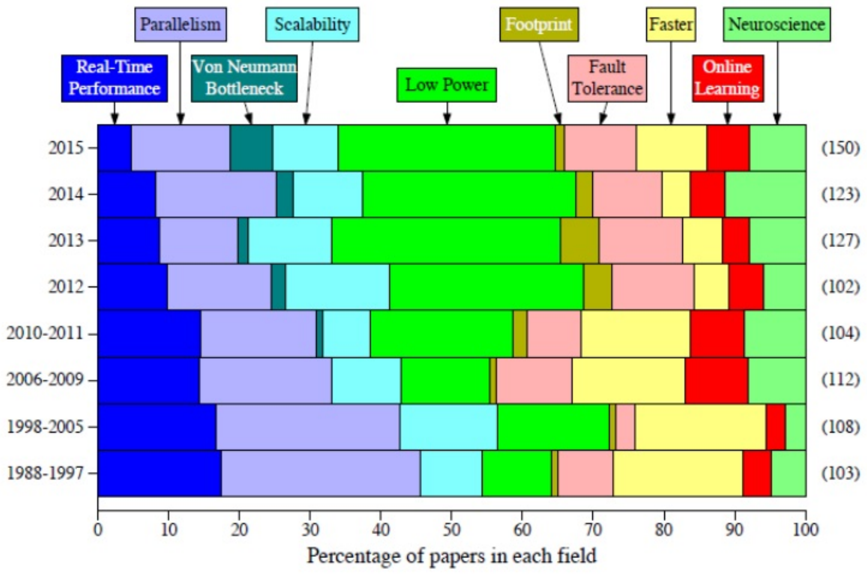


Figure 3: Different aims for neuromorphic computing research. Figure from [50].

stored have emerged during the last two decades. This new way of approaching hardware is the bio-inspired computing, which imitates the biological process happening in the brain and replicating the neuron-synapse interaction. The possibilities with these technologies are very promising and should provide higher density integration, lower power consumption and higher speed access [58] [50] (Fig. 3).

The simplest architecture is composed just by two elements: the neuron, the computational element and the synapse, the memory element. These two, connected together in different ways, generate several interesting properties. Its intrinsic parallelism makes it easier to compute operations with matrices of data, which are one of the slowest and most power hungry tasks a CPU could compute. The plasticity, i.e. the ability to update a stored information, is widely used in machine learning for learning and does not limit to the processing of input data, but also to the ability to recover from mismatches and process variability.

0.2. From biology to Artificial Neural Network

0.2.1. The biological neuron

Morphologically speaking, the neuron was first described by Santiago Ram ´on [20], winner of the Nobel prize of 1906 with Camillo Golgi for their work on the structure of the nervous system. The simplest neuron (Fig. 4) known and closer to Ramón's description

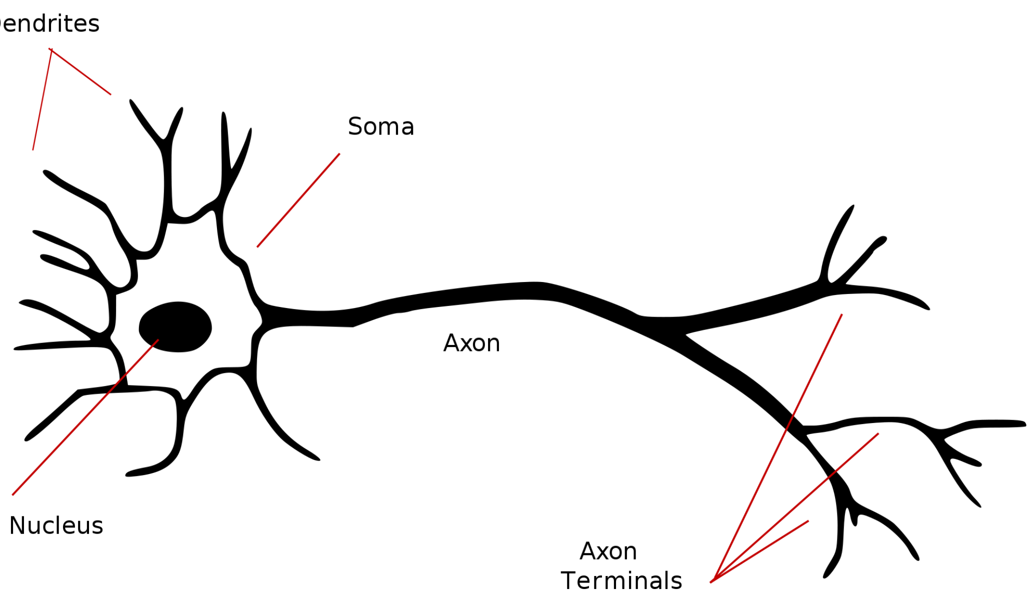


Figure 4: Simplified image of a neuron. Figure from [3].

is composed of three main components. The soma is the body of the neuron which contain the nucleus of the cell. Attached to the soma there are the dendrites that are the input branches of the neuron, where the signals are injected. The signals from the soma are then ejected from the terminals of a long connection called axon.

The first scientists to rigorously describe the neuron's working principles were Hodgkin and Huxley [23] who, by studying the squid's nervous system, created the first mathematical model in which they theorized that the information of the neuron is obtained by the voltage drop across the cell membrane. The result of such research led scientists to focus on the membrane voltage decomposing it into three different phases (Fig. 5). When the neuron does not receive any external signal the membrane voltage reaches the **resting potential**, a chemical equilibrium that stabilizes at -70mV in order to balance the forces created by ion gradient concentration. An injection of neurotransmitters to the dendrites can depolarize or hyper-polarize the cell creating respectively an excitatory or inhibitory behavior. The so-called **graded potential** returns to the resting potential if a threshold is not reached, due to the leaking channels along with the membrane and diffusion phenomena. Otherwise, if the potential reaches the threshold, the voltage-gated Na^+ ion channels open and activate a positive feedback that pushes the voltage drop across the membrane up to 40mV, the **action potential**. After a short amount of time the potential drops to -80mV due to the opening of voltage-gated K^+ ion channels. This happens regardless of the temporal shape of the graded potential. Furthermore, once the neuron reaches 40mV, the voltage gates become inactive for a brief period (refractory

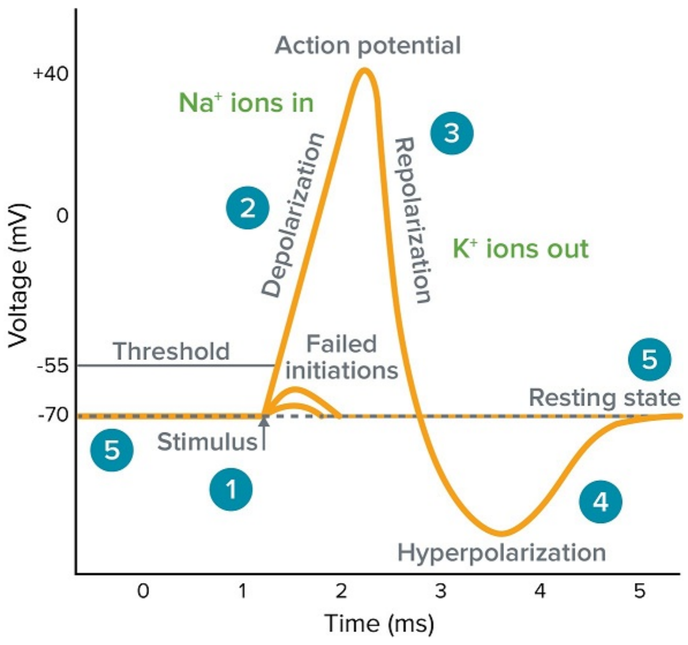


Figure 5: Phases of the membrane voltage. During the first millisecond the potential is resting at -70mV. Soon after 1ms signals are being received by the dendrites and the potential starts to grade. It is shown both the cases of threshold reaching and failing activation due to the leakage. After trespassing the threshold at 1,5ms the action phases starts with a depolarization and soon after polarization of the membrane. From 2,5ms to 5ms the neuron is refractory time and the voltage gates are inactive. Figure from [1].

time) where they cannot generate spiking.

The biological synapse

The connections between neurons became clear when the scientist Bernard Katz [31] modeled the neurotransmission mechanism through the introduction of the synapse. It connects the axon of a neuron to a dendrite of another and permits the neurotransmitters to diffuse across the cleft inside the synapse vesicle and bind to receptors on the postsynaptic neuron. The plastic memory of the neuron is expressed in the number of neurotransmitters released for a short or long time that depending on the different activity on the synapse itself. Unfortunately, the synapse itself could not explain the learning behavior and a more deep understanding was required.

There were some theories about learning during the years, mostly about local learning. but there is still no clear answer to this question. Donald O. Hebb [22] was the first to theorized that the the connections between neurons could increase or decrease their transmission capacity depending on the ongoing activity of the neurons they were linking.

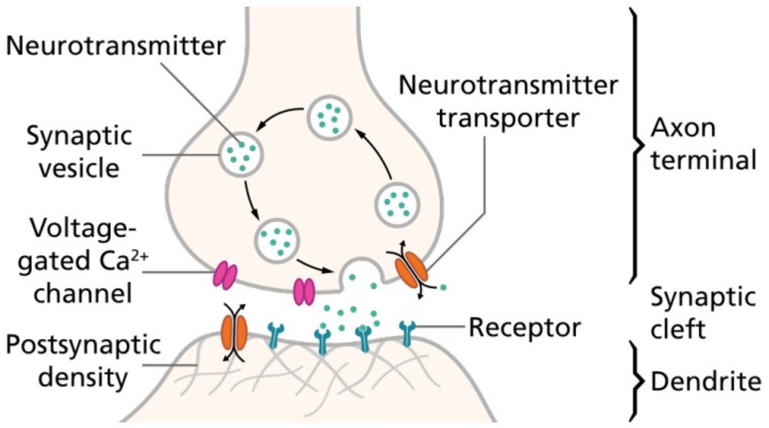


Figure 6: Simplified image of a synapse. Figure from [4].

The biology confirmed this theory, which works on local learning, but still there is no robust theory for the global learning of the brain and no certainty that this is the only local learning. A mathematical model that attempted to imitate the behaviour of the neuron to understand better the learning capability of the latter was made by F. Rosenblatt, the perceptron [46].

0.2.2. The evolution of perceptron

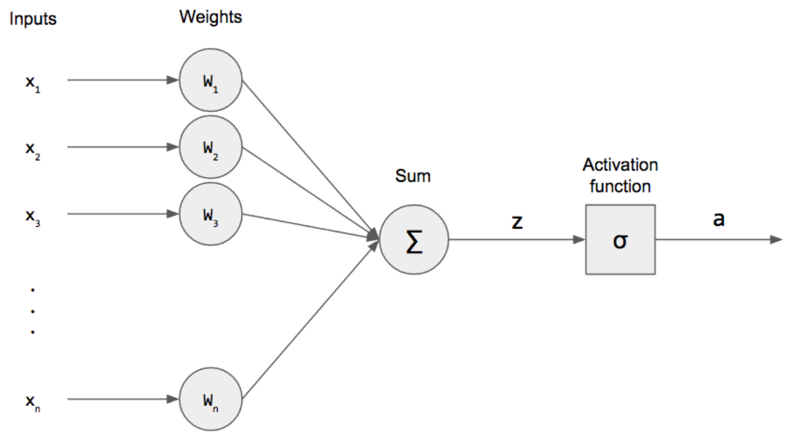


Figure 7: Single layer perceptron. Figure from [3].

The perceptron (Fig. 7) can be considered the main block of an ANN and if a biological neural network is composed just by synapses and neurons, the perceptron, as a mathematical model, needs a third element which is the activation function. The information is processed by multiplying each input number by the value of the weight and summed in a simplistic, but similar way to the neuron. The output signal, 'z' in Figure 7, is then

fed to a non-linearity that is called activation function. This is done both because the action potential of the neuron resembles a non-linearity and because, otherwise, a neural networks could not work as universal function approximators, but would be limited by linear functions. As it can easily be seen, the perceptron has some similarities with the neuron, but it is definitely not as accurate model of the latter. Nevertheless, most of the state of the art NNs are based on this model.
Over the years different structures of perceptron have been used to increase multilayer perceptron's capability of universal function approximators. The approaches have been of different kind, from simply developing new kind of activation function to add probabilistic activation and convolution between layers. The model that will be mostly taken into account in this thesis is the Spiking Neural Network (SNN), closer to a real neuron, that has a Heaviside step activation function and has the ability to retain memory of previous inference for a certain amount of time.

0.3. Artificial intelligence and neural networks

The term "artificial intelligence" is widely used in neuromorphic computing literature, but it has a much wither meaning. Even if this field was born inspired by the human mind, it includes any algorithm capable of enable problem-solving from sensible information. A portion of it is dedicated to machine learning which is the study of any algorithm that can improve automatically through experience and by the use of data.

A lot of machine learning algorithms have been developed and proved to be useful in tasks that are common for humans, but not for machines, such as classification and regression problems and data reduction. One of the first and most robust machine learning algorithm is the Support Vector Machine (SVM), widely used for classification tasks by supervised learning. RandomForest and XGboost are other algorithms based on regression trees that excels respectively in data classification and regression tasks.

These and others machine learning algorithms have been proved to have outstanding performances and are widely used, but there are some tasks that cannot achieve a sufficient accuracy with these methods. Tasks as temporal changing data/image classification and regression are too challenging for standard machine learning and a deep learning approach is needed.

Neural networks are the most promising deep learning architectures that have been proven able to solve these challenging tasks with Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). The advantages of using neural networks are not limited by the stationary ability of "understanding" data sometimes better than humans, but their

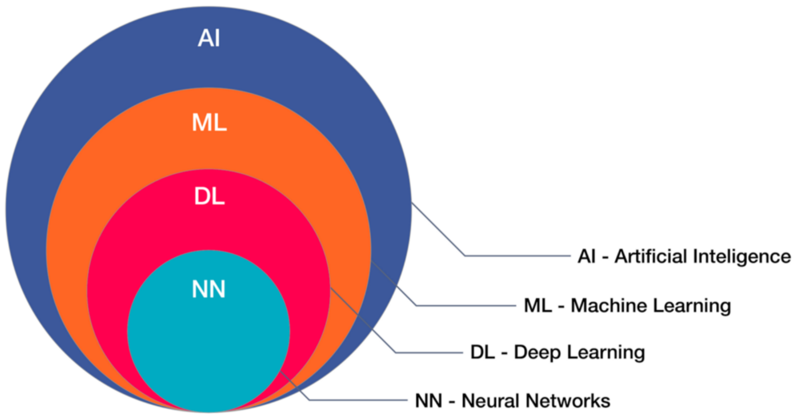


Figure 8: Artificial intelligence clustering. Figure from [2].

parallelism makes them suited for being implemented directly in hardware. Standard machine learning algorithms are made to perform optimally with sequential processing, and so their computation time and power dissipation is intrinsically linked to the computational time and power dissipation of standard computing technology. On the contrary, neural networks require huge time and power dissipation with standard sequential processing. Neuromorphic hardware could reduce the computational time of deep learning by orders of magnitude, as days or dozens of hours, to single hours or even minutes, with a proportional decrease of power dissipation. The possibility of improvement with these technologies is enormous considering what the brain can do consuming $10^{-10}J$ per action potential versus the $25nJ$ per operation of digital logic [49].

0.4. Aim of the Thesis

In this Thesis work, an asynchronous CMOS-based spiking neural network [34] is taken as a reference. A prototype of the chip with 3 neurons have been implemented by Polidori E. [44] and Polidori C. with the scope of testing the limits of this implementation. Even if a synaptic weight adaptation based on the Spike Time Dependent Plasticity principle was already implemented, this unsupervised training does not exploit the full possibility of this technology. Starting from a detailed analysis, the aim is to further expand the network into a more complex neuronal connection and develop a supervised or reinforcement learning for the synaptic weights.

During this thesis work the chip has been measured and analyzed. A suitable supervised online-learning that could be implemented has been developed and simulated. The choice of the algorithm will be explained and motivated with a review of the current the state-of-the-art learning for Spiking Neural Networks.

1 | Learning in spiking neuromorphic hardware

The usefulness of a neural network is mainly due to its intrinsic ability to incorporate learning by updating the synaptic weights between pairs of neurons. Different learning are going to be exploited since different training are required for different tasks. The chapter develops the basic concepts to understand learning in SNN from the definition of learning in standard ANN to the state-of-the-art training for SNN.

1.1. Inference

Mathematically speaking standard multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy regardless of the activation function. This features was demostrated by K. Hornik in 1989 and stated in the famous theorem [30]:

Theorem 1.1. *For every squashing function Ψ, every r and every probability measure μ on (R^r, B^r), both $\Sigma\Pi^r(\Psi)$ and $\Sigma(\Psi)$ are uniformly dense on compacta in C^r and ρ_μ-dense in M^r.*

$$\Sigma_{j=1}^{q}\beta_j G(A_j(x)), x\epsilon R^r, \beta_j\epsilon R, A_j\epsilon A^r, q\epsilon N$$

$$\Sigma_{j=1}^{q}\beta_j \Pi_{k=1}^{l_j} G(A_{jk}(x)), x\epsilon R^r, \beta_j\epsilon R, A_{jk}\epsilon A^r, l_j\epsilon N$$

Where $\Sigma\Pi$ is the hidden layer of a feedforward network, r the dimension of the input space and μ the input space environment and Ψ is the activation function. A feedforward network is a neural network that has no recurrent connection between the neurons and where the neurons are divided in layers (Fig.1.1 **a**)).

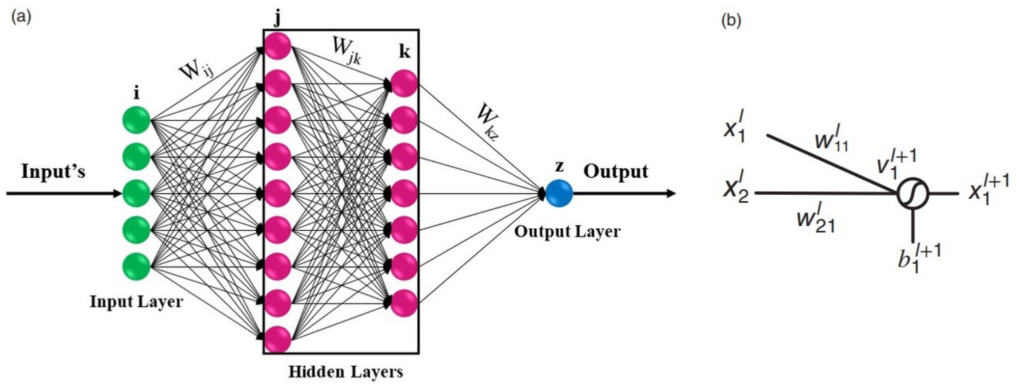


Figure 1.1: **a**) Multilayer perceptron, a feedforward ANN. Figure from [52]. **b**) Connection between presynaptic neuron and postsynaptic neuron in ANN. Figure from [39].

The output computation of an ANN is usually appealed as inference and can be calculated as (Reference Fig. 1.1 (b)):

$$x_j^{l+1} = f(\Sigma_{i=1}^{n} w_{ij}^l x_i^l + b_j^{l+1}) \tag{1.1}$$

Where l is the layer of the presynaptic neurons, l+1 the layer of the postsynaptic neurons, j is the number of the postsynaptic neuron in the l+1, n is the number of the neuron in the presynaptic layer l. In the equation the output of the presynaptic neurons x_i^l (i = 1,2,...,n) is multiplied by the synaptic weights w_{ij}^l that connect presynaptic neurons with the postsynaptic neuron number j. These values are then summed together with a bias b_j^{l+1}, included to provide more design flexibility, and fed to a non-linear activation function $f()$ (Ψ in Theorem 1).

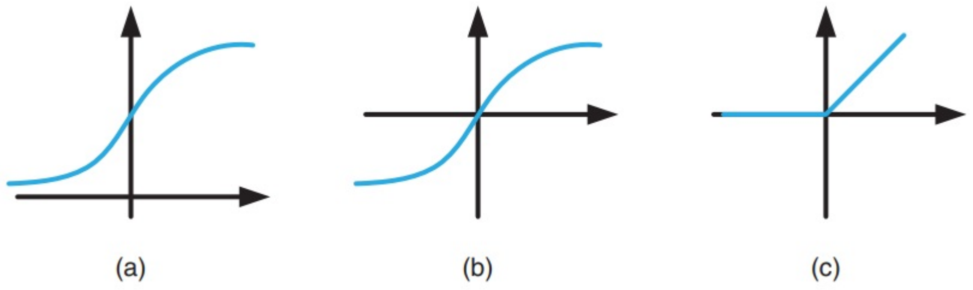


Figure 1.2: Popular activation functions.**a**) A sigmoid function. **b**) A hyperbolic tangent function. **c**) A ReLu function. Figure from [39].

Popular activation functions are hyperbolic tangent function, logistic function, and rectified linear unit (ReLU) (Fig.1.2).The real power of the neural networks is given by the non-linearity of the activation function that is the key factor to enable neural networks to approximate any function with a moderate number of parameters.

1.2. Neural Network's typical architectures

ANN is a versatile and adaptive machine-learning tool and as such, it has been adopted to solve multiple and different kinds of tasks. Experimentally, different structures have been proved to be better at solving specific tasks than others, depending on the spatial or temporal information of the dataset or on its discrete or continuous nature.

The three main types of network topologies used in this thesis are:

- **Fully Connected Neural Network**: it is the simplest form of NN. Every neuron of one layer is connected to every neuron of the forward layer. The operations involved in computing inference are usually matrix multiplications followed by non-linearity. Examples of networks with this structure are Multilayer Perceptron (MLP, Fig. 1.1 a), introduced in the previous section and usually trained with backpropagation (see next section) or Radial-Basis Function (RBF). RBF networks are composed of three layers and use radial basis functions as non-linearity of the hidden layer. Radial basis functions are functions whose output is determined by the distance between the input and the origin. Differently from MLP, learning in RBF networks is interpreted as a curve-fitting (approximation) problem in a high-dimensional space.

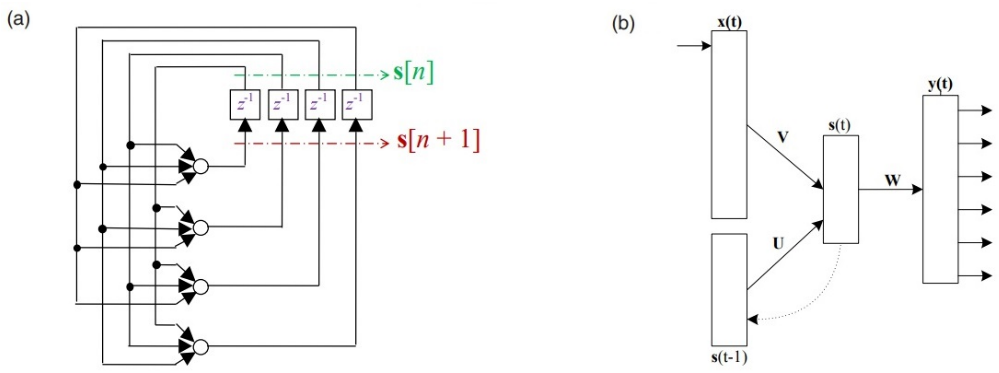


Figure 1.3: Recurrent neural networks. **a**) Four neurons Hopfield neural networks.**b**) Three layer neural network with recurrent hidden layer 's'. Figure **b** from [21]

- **Recurrent Neural Network**: RNN does not only have feedforward connections but also connections that form a directed cycle. It must be noted that the directed cycle includes some form of time delay, i.e. it is a feedback structure and not an algebraic loop. This kind of feedback enables the network to acquire memory

that makes the output dependent on multiple inputs over time. Such memory is helpful in dealing with applications where the input data have temporal or sequential correlations.
A peculiar example of this network structure is the Hopfield network. Hopfield networks are RNNs with binary neurons that can serve as a content-addressable memory. Each memorized sample is stored as a local minimum of the network's energy function and works as an attractor of the energy function. If an input similar to that memory sample is presented to the network, it will collapse in a state linked to that specific attractor. Instead, temporally correlated input data are usually classified using MLP with recurrent connections. These networks are trained with backpropagation through time that consists of backpropagating the error through the network unfolded over time.

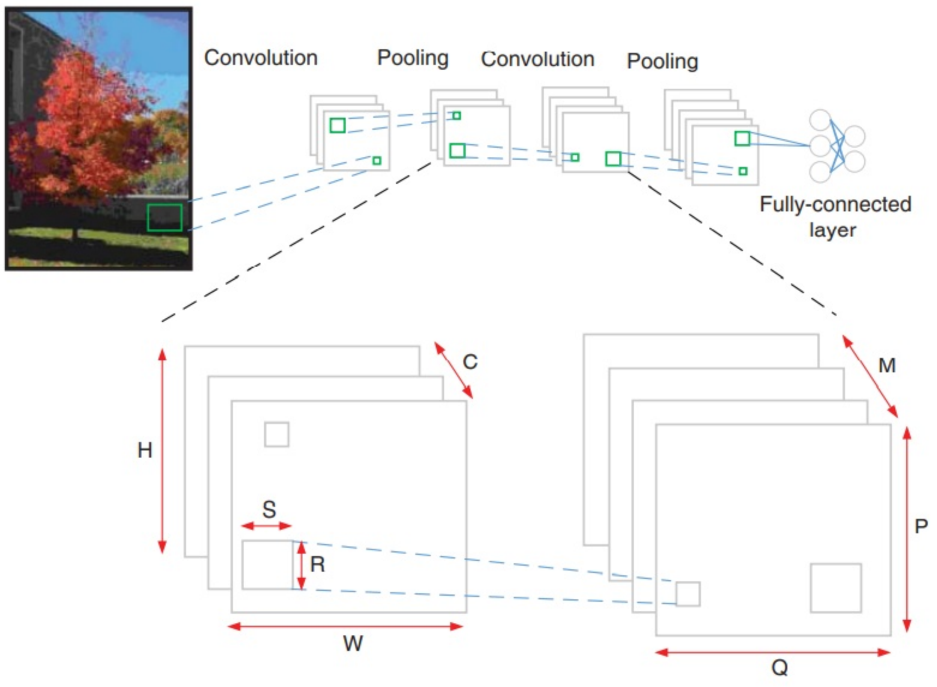


Figure 1.4: Convolutional neural network scheme. Figure from [39]

.

- **Convolutional Neural Network**: CNN is another variation of MLP based on filters (convolutions) that result in feature maps. Each filter is applied to all feature maps of the previous layer, a procedure that allows finding features no matter which map contains them. Such a method also allows the network to gather the necessary information regarding the relative spatial arrangement of features, since the location of features itself is not important. Furthermore, between each convolutional layer a pooling layer, i.e. a two-dimensional filter, is needed and can consist of max pooling, averaging, stochastic sampling or a layer of trained neurons. Such a layer is necessary not only to reduce the size of the feature map, but also to reduce the impact of feature displacement and other distortions. The network's last layers can be a MLP or another kind of machine learning algorithms to have a more accurate regression or classification based on the extracted features of the previous layers. This kind of architecture has been inspired by the cat's visual cortex and it is suitable for image analysis, especially for the detection of objects. Such a process works because images are stationary and features that appear in one part of the image are as likely to appear in other parts of the image. Furthermore, CNNs are robust to the translation of the images and can be trained with backpropagation to be robust also to rotations.

1.3. Fundamentals of Artificial Neural Network's Learning

Learning is a process where unknown ANN parameters are adapted through continuous process of stimulation from the environment. The way how the change of parameters takes place determines the learning itself. The more common parameter to be taken into account is the synaptic weight. We can express the learning as:

$$w^l_{kj}(n+1) = w^l_{kj}(n) + \Delta w^l_{kj}(n) \tag{1.2}$$

Where in the learning step $n+1$ the synaptic weight $w_{kj}(n)$, which is the old weight between neuron k and j, is changed by $\Delta w_{kj}(n)$ amount resulting in the new value of the weight $w_{kj}(n+1)$. Sometimes also bias parameters are trained during the learning, but they can be treated as a synaptic weight parameter with its input always being one.

The variation of the weights can be determined by different learning algorithms that are a solution to the learning problem. A problem can be solved by different algorithms, each with its advantages and drawbacks. These algorithms can be clustered in three learning paradigms that determine the relation of the ANN to the environment: supervised learning, unsupervised learning and reinforcement learning.

1.3.1. Supervised learning

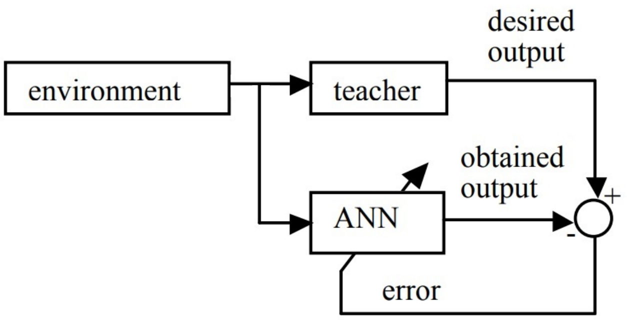


Figure 1.5: Block scheme of supervised learning paradigm.

Supervised learning is the most developed type of machine-learning task using neural networks. This paradigm is characterized by the presence of a teacher. The teacher has knowledge of the problem in the form of precise input-output pairs. The input is fed to the network for inference and the output of the network is compared to the respective output of the teacher. From this comparison an error is calculated and the parameters of the ANN are changed under the influence of the input vectors and error values. This learning process is repeated until the network learns to imitate the teacher. After learning is completed, the teacher is no longer required and the ANN can work without supervision. A function $L(w)$, which is usually referred to as loss (cost), is required to calculate the errors. The function can be represented as a multidimensional error surface which depends on the free parameters of the network. A point on the error surface is defined by weights and corresponds to any configuration of the ANN. The minimization of the loss function is mostly done through a gradient descent-based optimization or its derivatives, even though some other standard optimization methods, such as the genetic algorithm, can be used as well [39]. The method approximates the true loss function locally and updates the network parameters in the opposite direction of its gradient. Equation 1.2 can be written as:

$$w^l_{kj}(n+1) = w^l_{kj}(n) - \eta \frac{\partial L(w(n))}{\partial w^l_{kj}(n)} \tag{1.3}$$

Where the weight update consists of the linearized gradient multiplied by a factor η, called learning rate. This factor is fundamental for a proper change of the weights. When the update is too small, the linearized model matches well with the true loss function, but it can remain stuck in local minima. On the contrary, when the changes in the parameters are too large the true loss function behaves significantly differently from the assumed linearized model, which results in divergence. Furthermore, even if the learning rate is

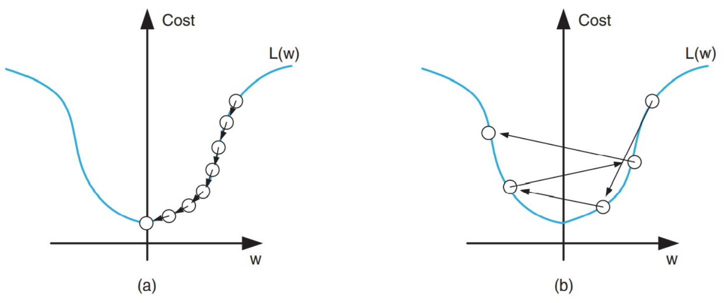


Figure 1.6: Illustration of a one dimensional gradient descent process. The learning process can be viewed as movement of the point down the error surface towards the global minimum. Figure from [39].

not large enough to diverge, the weights can oscillate around a global or local minima. Clearly, the choice of this factor is crucial for a proper learning of the network, but there is not a precise method for this. The most common strategy is to go through an empiric process of trial and error.

Even if there is no theoretical limit to the potentiality of learning, the gradient descent method cannot guarantee the convergence to the global minimum. The neural network is expected to figure out the correct function mapping by minimizing the difference between its output and the target. However, if the network does not have enough free parameters it cannot properly approximate the function. This case corresponds to an underfitting of the data, since the network cannot properly fit the inputs. The opposite happens when the network has too many degrees of freedom and fits the data, but approximates a different function from the desired one. In both case the network performs poorly on a test dataset. A good way to distinguish them is to look at the performance of the training set. If the network is unable to correctly fit, also the training set it means that it has not enough free parameters for the task and it is underfitted. If the opposite happens, we are in the overfitting case. To avoid underfitting the simplest and more efficient solution is to increase the size of the network, while for overfitting process can be more complex. To solve overfitting more constraints to the learning are needed and can be done both by acting on the dataset or by adding a regularization term to the loss function.
The process of finding the gradient with respect to each parameter of the network is called backpropagation. This method was successfully implemented in 1986 [17] and since then it has been widely used as a definitive method for training neural networks. The process utilizes the chain rule used to find the derivative of a composite function in order to obtain the gradient associated with each synaptic weight that is not directly connected to the

output neurons. The gradient of the loss function can be expressed as:

$$\frac{\partial L(w)}{\partial w_{kj}^{l}} = \frac{\partial v_j^{l+1}}{\partial w_{kj}^{l}} \frac{\partial x_j^{l+1}}{\partial v_j^{l+1}} \frac{\partial L(w)}{\partial x_j^{l+1}} \tag{1.4}$$

where:

$$v_j^{l+1} = \textstyle\sum_{i=1}^{n} w_{ij}^{l} x_i^{l} + b_j^{l+1}$$
$$x_j^{l+1} = f(v_j^{l+1})$$

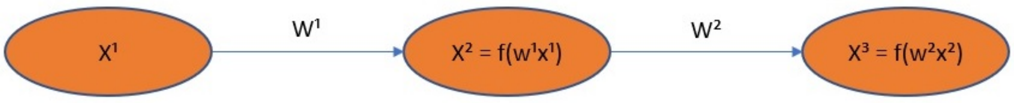


Figure 1.7: Scheme of a simplified network with 3 layers: input layer, hidden layer and output layer. Each layer with only one neuron

For the sake of simplicity, a trivial monodimensional example of a three layer multilayer perceptron with a simple loss function will be considered (Fig. 1.7), in the time lapse of a single cycle of inference and backpropagation (n will be omitted):

$$L(w) = \frac{1}{2n}(y-t)^2$$

Where $y = x^3$ is the output of the network and t is the target (desired output).

$$\frac{\partial L(w)}{\partial w_{11}^2} = \frac{\partial v_1^3}{\partial w_{11}^2} \frac{\partial x_1^3}{\partial v_1^3} \frac{\partial L(w)}{\partial x_1^3}$$

$$\frac{\partial v_1^3}{\partial w_{11}^2} = \frac{\partial (w_{11}^2 x_1^2)}{\partial w_{11}^2} = x_1^2$$

$$\frac{\partial x_1^3}{\partial v_1^3} = \frac{\partial (f(v_1^3))}{\partial v_1^3} = f'(v_1^3)$$

$$\frac{\partial L(w)}{\partial x_1^3} = \frac{\partial (\frac{1}{2}(x_1^3 - t)^2)}{\partial x_1^3} = x_1^3 - t = e$$

So for the weight connecting the hidden layer to the output layer the weight update would be:

$$\Delta w_{11}^2 = -\eta \frac{\partial L(w)}{\partial w_{11}^2} = -\eta \frac{\partial v_1^3}{\partial w_{11}^2} \frac{\partial x_1^3}{\partial v_1^3} \frac{\partial L(w)}{\partial x_1^3} = -\eta x_1^2 f'(v_1^3)(x_1^3 - t) = -\eta x_1^2 \delta^3$$

Where $\frac{\partial L(w)}{\partial v_1^3} = \frac{\partial x_1^3}{\partial v_1^3} \frac{\partial L(w)}{\partial x_1^3} = f'(v_1^3)(x_1^3 - t) = \delta^3$ is denoted in literature as the error (vector), which, in this case, is the error of the last layer. The calculation for the previous weight is similar, but partially depends on the weight update of the deeper layer:

$$\frac{\partial L(w)}{\partial w_{11}^1} = \frac{\partial v_1^2}{\partial w_{11}^1}\frac{\partial x_1^2}{\partial v_1^2}\frac{\partial L(w)}{\partial x_1^2}$$

$$\frac{\partial v_1^2}{\partial w_{11}^1} = \frac{\partial (w_{11}^1 x_1^1)}{\partial w_{11}^1} = x_1^1$$

$$\frac{\partial x_1^2}{\partial v_1^2} = \frac{\partial (f(v_1^2))}{\partial v_1^2} = f'(v_1^2)$$

$$\frac{\partial L(w)}{\partial x_1^2} = \frac{\partial v_1^3}{\partial x_1^2}\frac{\partial x_1^3}{\partial v_1^3}\frac{\partial L(w)}{\partial x_1^3} = w^3 f'(v_1^3)(x_1^3 - t) = w_{11}^2 f'(v_1^3)(x_1^3 - t) = w_{11}^2\delta^3$$

$$\Delta w_{11}^1 = -\eta\frac{\partial L(w)}{\partial w_{11}^1} = -\eta x_1^1 f'(v_1^2) w_{11}^2 \delta^3 = -\eta x_1^1 \delta^2$$

With these brief calculations, it has been shown that the gradient of the loss function with respect to each synaptic weight matrix can be expressed, in a more compact way, as the output of the previous layer of neurons times the error vector of the forward layer. The error of each layer is calculated as follows:

$$\delta_j^l = f'(v_j^l)\sum_{k=1}^{n}\delta_k^{l+1} w_j^l \qquad n \text{ number of neurons in layer } l+1$$

Except for the last layer that is:

$$\delta_j^z = f'(v_j^z)e_j \qquad z \text{ output layer}$$

Historically, the sigmoid function and hyperbolic tangent function were used extensively in neural networks. Especially, the sigmoid function was widely used because of its derivative, that can be expressed as a function of the output of the reference neuron, speeding up the computational process. Recently in deep learning, the ReLu function and its derivatives are predominantly used. The reasons are the further increase of computational speed, since the ReLu function can be computed as a conditional pass operation, and because it prevents the gradient to vanish in deeper layers during backpropagation. In fact, the sigmoid function and tangent function derivatives tend to zero at the edges, causing the gradient to vanish in case a synaptic weight is too high or to low. It would take a lot of iterations to further modify that specific weight. Using a ReLu function prevents this situation to happen.

1.3.2. Unsupervised learning

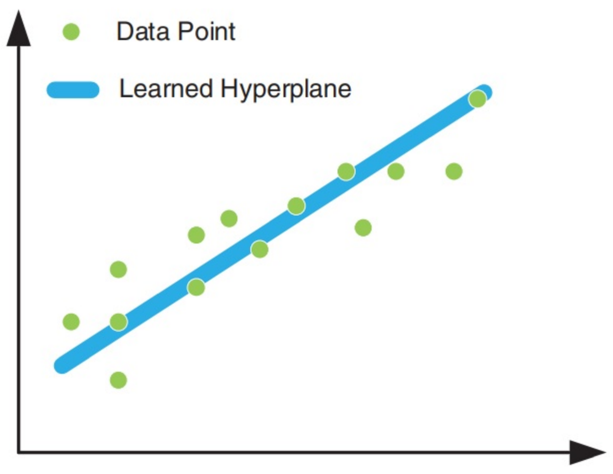


Figure 1.8: Illustration of dimensionality reduction in unsupervised training. The data can be classified using just the dimension of the blue line. Figure from [39]

No labels for the data nor a direct reward for positive behavior are needed for the unsupervised learning paradigm. If data exhibits redundancy, this method of training enables the network to discover underlying data structures such as patterns, correlations and categories by itself. Without redundancy the input data resembles random noise, making unsupervised learning impossible. The learning normally appears in the form of dimensionality reduction (Fig. 1.8), as most natural signals have certain built-in structures. This can be accomplished by self-organizing networks that, depending on their architecture, can infer different underlying structures and be used for different purposes. In general, the network must learn to represent the input and must produce some output based on the learned representation. The output can represent:

- Similarity: The network converges in a single output showing a similarity measure between the input sample and all previous (averaged) samples.
- Principal Component Analysis (PCA): Extension of similarity measure to multiple vectors, i.e. the network outputs similarities to the eigenvectors of the data correlation matrix.
- Clustering: The network discovers the inherent groupings in the data with an indicator output function.
- Prototyping: Similar to clustering except the output is a representative sample of a cluster instead of an indicator function.

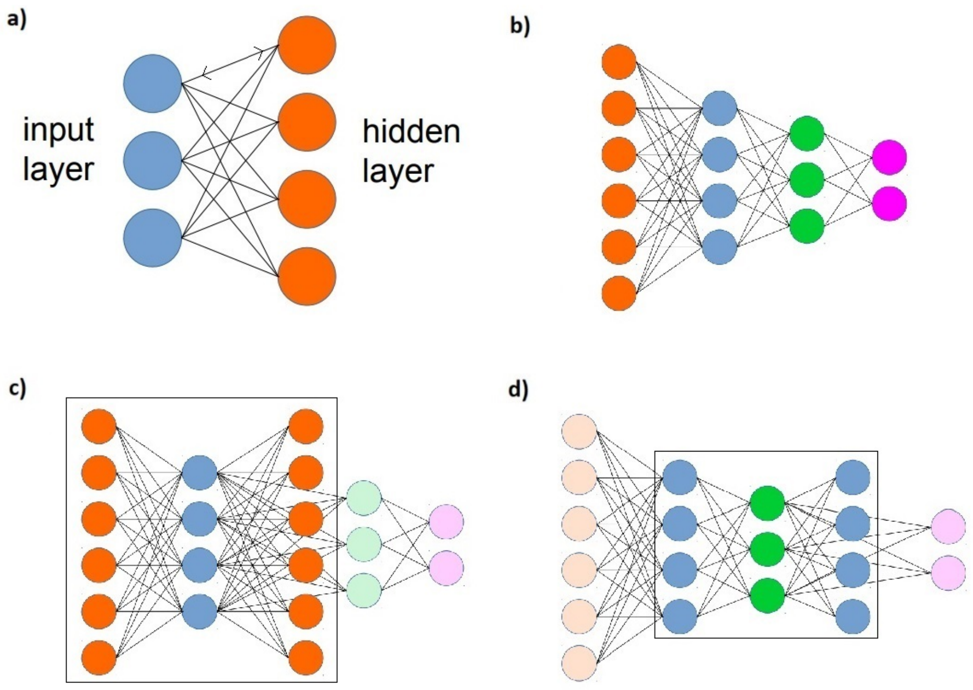


Figure 1.9: **a**) Architecture of a Restricted Boltzmann Machine. **b**) Reference multilayer perceptron. **c**) Illustration of how to autoencode the first hidden layer of a multilayer perceptron. **d**) Autoencoding the second hidden layer of a multilayer perceptron.

- Coding: The output is a coded version of the input using less memory while keeping most of the information content intact (e.g. vector quantization, data compression).
- Feature Mapping: The output neurons are arranged on a regular grid (e.g. a rectangular lattice) and only one may be active at any single moment.

The learning in self-organized networks can be Hebbian, where all the neurons may be simultaneously active, or competitive, where only a single neuron in a group may be active. Furthermore, in competitive learning, only synapses linked to the active neurons are able to perform potentiation. Networks having the latter property are called winner-take-all or WTA networks. Since such networks group or categorize data, they are often used for pattern classification in computer vision or for vector quantization. Another emerging application of unsupervised learning in neural networks is to conduct pre-training for supervised learning, such as Restricted Boltzman Machines (RBM) and autoencoders (Fig. 1.9). They can function as feature detectors which gets "hidden" in the hidden layer and are not directly usable. RBMs are fully connected between layers, with bi-directional synapses and are used to learn the probability distribution of the training set. They are trained in a layerwise fashion by contrastive divergence (CD), which approximates a maximum-likelihood learning algorithm [35].

Autoencoders are structured like a feedforward network, with input (encoding) and output (decoding) layers of the same dimension and a hidden layer with lower dimension (Fig. 1.9 **c**). The training is supervised-like and consists of standard backpropagation using the input as the target vector. The network, this way, will learn to encode the data and perform a perfect reconstruction. The encoded data can be seen as a reduction of the input and so an extraction of its most important features. The process can be done with each layer of a network considering the output of the previous layer the input and making a dummy output layer for training. Such an unsupervised-learning approach is one of the earliest techniques that helps to realize deep learning, because it is used to initialize the synaptic weight to significative values instead of using random initialization.

1.3.3. Reinforcement learning

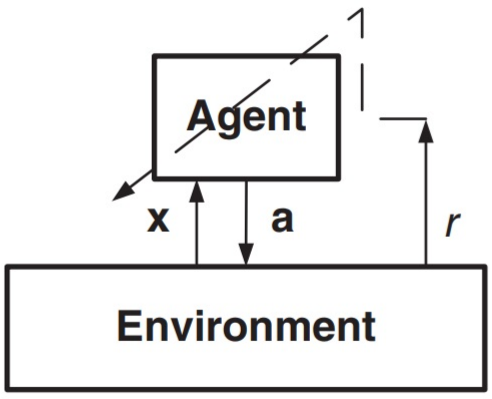


Figure 1.10: Illustration of reinforcement learning paradigm. The agent interacts with the environment. **x** is the state of the environment, **a** the action of the agent and **r** the reward signal. Figure from [39].

Even though the reinforcement paradigm may resemble supervised learning, in some aspects it is completely different. In reinforcement learning, there is no explicit supervisory signal that instructs what the output of the neural network should be. Input-output learning mapping is learned through the iterative process where a measure of learning quality is maximized. The teacher does not present input-output training examples, but only gives a grade representing a measure of learning quality. Reinforcement learning overcomes the problem of supervised learning where training examples are required. The learning can be performed only online i.e. in real-time and the network learns dynamically during the exploitation phase.

The most common task that reinforcement-learning algorithms attempt to solve (Fig. 1.10) consists of an agent interacting with its surrounding environment. At every time step t, the agent is able to observe the states of the environment x(t) and tries to pick the correct action a(t). A reward signal is generated accordingly to the choice. The agent will modify its actions such that the accumulated reward it receives in the future is maximized.

Suppose that the discrete-time system the agent is interacting with can be modeled by

$$x(t+1) = f[x(t), a(t)] \tag{1.5}$$

where x(t) is the k-dimensional state vector at time t, a(t) is the n-dimensional action vector, and $f[\cdot]$ is the model of the system. The paradigm can be described in discrete time as [39]:

$$j[x(t)] = \sum_{k=1}^{\infty} \gamma^{k-1} r[x(t+k)] \tag{1.6}$$

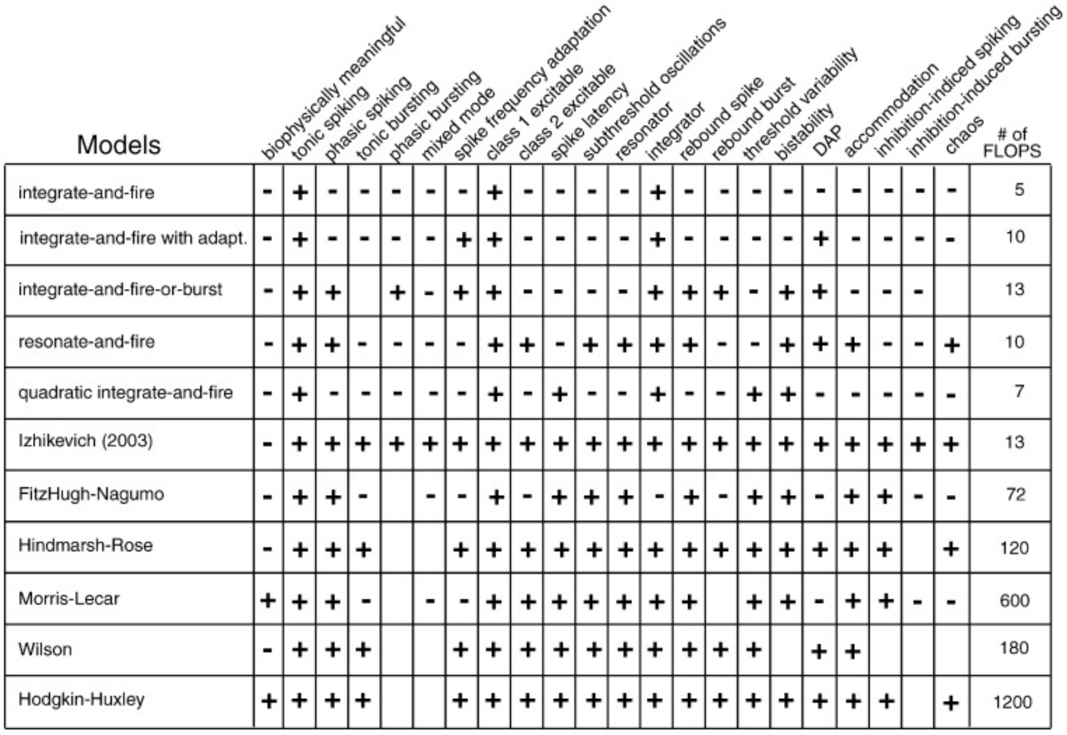
where j is the reward-to-go and γ is the discount factor used to promote the reward received soon over a long-term reward. r[x(t)] is the reward received at state of the environment x(t).

The target of the reinforcement learning is to maximize the reward-to-go (Eq. 1.6). This can be achieved through solving the Bellman equation [39].

$$j^*[x(t)] = max_{a(t)}(r[x(t+1)] + \gamma j^*[x(t+1)]) \tag{1.7}$$

Solving Bellman equation can be solved approximately with adaptive dynamic programming (ADP) or it can be directly solved through dynamic programming, Q-learning, Sarsa, etc [39]. If we consider playing a game as an example of supervised learning, we can understand that it is easy to give a reward at the end of the game given the number of moves and the outcome of the game. However when it comes to assign credits to each move calculating a proper delayed reward is not straightforward. How to assign credits for the ultimate reward is one of the most difficult tasks to solve in a reinforcement-learning problem. It requires the agent to have an internal evaluating algorithm to estimate potential rewards for each possible state. The most common ways are the Monte Carlo (MC) method, where the agent averages all the rewards received starting from one state and uses that average to approximate the expected reward, and the temporal difference (TD) learning, where the agent uses a newer estimated value to update an old estimation before knowing the final result. Furthermore, the rewards-to-go can be stored in look-up tables or generated by a function approximator. The look-up table approach is easy to implement, but has the downside of increasing the size as the number and dimensionality of states increase, while a function approximator would require only to change its parameters.

1.4. Spiking Neuron



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

Figure 1.11: Several mathematical model for neurons and their capability of emulating abilities of the biological neurons.« "♯ of FLOPS" is an approximate number of floating point operations (addition, multiplication, etc.) needed to simulate the model during a 1 ms time span». Figure from [27].

As mentioned in the previous chapter the principal aim of using neuromorphic hardware is to reduce the power dissipation. Even if a greater portion of neural networks literature have been about training ANN with graded values as output, a great portion of signal processing in the brain happens through spikes. Spike coding can be more power consuming than a graded, when the spike rates is too high. The brain is capable of saving power using a sparse code [10] to process signals, i.e. not all the neurons need to be simultaneously active, but only a portion of them. This permits to reduce power consumption even more than a graded network at cost of diminishing the quality of the information.

Various mathematical models of Spiking Neurons have been implemented over the years

(Fig. 1.11), each with different features and a different degree of similarity with biological neurons. This work focuses on learning for Leaky Integrate and Fire model, not only because it is the most used in hardware implementations [50], but also because the reference network's neuron [34], which is nominally a Morris-Lecar model, can be treated as a LIF neuron under certain conditions.

1.4.1. Leaky Integrated and Fire

This neuron model was first introduced by Lapicque in 1907 [12]. It is composed by a membrane capacitance that does not describe the action potential routine but only the graded potential one (section 0.2.1). Such a model is more empirical than biologically accurate and it is much easier to use during simulations and to implement in hardware.

$$\frac{dV_{mem}}{dt} = -\frac{1}{\tau_{mem}}[(V_{mem} - V_{rest}) + I_{input}R_{leak}] + S_{neu}(V_{rest} - V_{th}) \tag{1.8}$$

Where V_{mem} is the membrane potential, V_{rest} is the value of the membrane potential at equilibrium and its maximum value, I_{input} is current injected into the neuron, R_{leak} is the equivalent resistance of the membrane, which simulates the graded potential when no spike occurs and $\tau_{mem} = C_{mem} \cdot R_{leak}$. $S_{neu} = H(V_{mem} - V_{th})$ is a nonlinear function of the membrane voltage, which express an instantaneous voltage change in the membrane potential when a spike occurs, where V_{th} is the threshold voltage of the neuron and $H(\cdot)$ the Heaviside step function.

This model is only able to describe the tonic spiking (i.e while exited the neuron continues to fire a train of spikes), the integration and the dependence between the intensity of current and spiking rate (Fig. 1.11). Even though, due to this simplification, the neuron alone is not able to reproduce complex tasks, with the right training and the right architecture a network of LIF neurons can recognise temporal and spatial patterns like ANN, but with an easier integration into chip and lower power consumption.

1.5. State of the Art training for hardware implemented SNN

Different learning paradigms have been proposed during the years for training ANN. However, only one training technique has been established as enough reliable to make ANN the essential signal processing method known nowadays, the backpropagation. Unfortunately, this same method cannot be applied to SNN. The reason is that, even if SNN can be seen as a particular case of RNN [18], the non-differentiability of the Heaviside function makes it difficult to apply the original algorithm. Biological neurons communicate by sending discrete spikes in opposition with real-valued numbers in ANNs. Furthermore, learning rules in the brain are most likely different from the backpropagation algorithm [8]. Since these facts exist, the SNN research field has been focused on searching for an algorithm close or equally powerful to backpropagation.

The more recent state-of-the-art training methods are divided between mathematical optimization and biological inspired algorithms. Entirely biological inspired methods have local learning rules between layers, in agreement with measured biological data. Mathematical optimization algorithms are adaptations of the backpropagation to work on SNN or adaptation of the sensible information propagated by the network to be differentiable. In this section both these algorithm families are discussed and compared in order to find the one that best suits this project.

1.6. Biologically inspired learnings

As anticipated in the introduction, the brain's synapses alter their strength through long time potetiation (LTP) or long time depression (LTD). Since the 19^{th} century this phenomenon has been studied and, even if there is not still a complete picture of the problem, it has been demonstrated that local interactions around the synapses activates a change of plasticity [19]. Donald O. Hebb theorized a causal strengthening of the synapse i.e. LTP would happen if the presynaptic spike would cause the postsynaptic spike and LTD when the opposite occurs. The later work of Bliss and Lømo (1973) [11] showed that high-frequency presynaptic firing drove LTP, while low frequency firing drove LTD. In 1998 the name "spike-time-dependent plasticity" (STDP) was coined by Bi and Poo [45].

Nowadays, it has been demonstrated that STDP is a multifactor rule that depends on spike timing, synaptic cooperativity, firing rate, and postsynaptic voltage [19]. The firing rate and depolarization requirement demands that multiple spikes are required for associative plasticity, and cooperativity. Furthermore, STDP depends importantly on baseline synaptic weight and on neuromodulators, which can shape it during and after spike pairing (i.e. the same spike pairing causes a different change of the synaptic weight depending on its baseline value and the amount of neuromodulators). However, depending of the area of the brain, each factor can be a major determinant of plasticity in some cases but a minor or negligible factor in others. At this stage of the hardware development for SNN there is no possibility of having a network big enough to have the luxury of implementing different learning in different parts of the network that considers some or all the previously listed factors. More complex models exists [5], but at the moment they can only be used

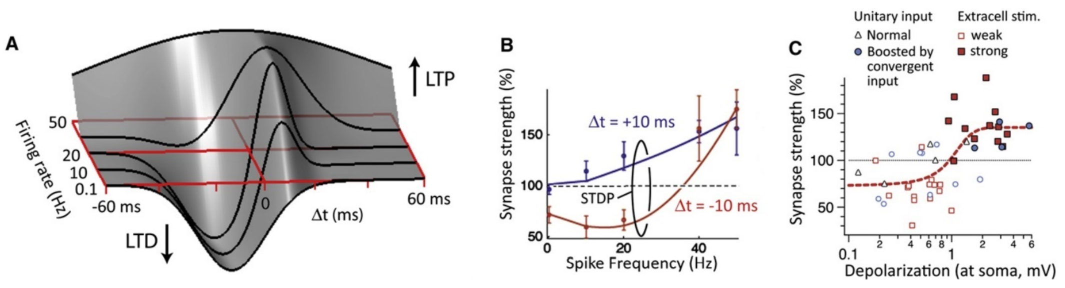


Figure 1.12: **a**) STDP considering joint time between spikes and firing rate at L5-L5 pyramid unitary synapses. **b**) Modeled multifactor STDP rule for the same synapse. **c**) Correlation between cooperativity and couple of spikes with $\Delta t = 10ms$ between presynaptic and postsynaptic spikes in distal synapses on L5 pyramidal cells. Open and filled symbols show inputs with weak and strong cooperativity, respectively. Figures from [19].

by software on CPU or GPU-based processors and will not be considered in this work.

1.6.1. Pair based STDP

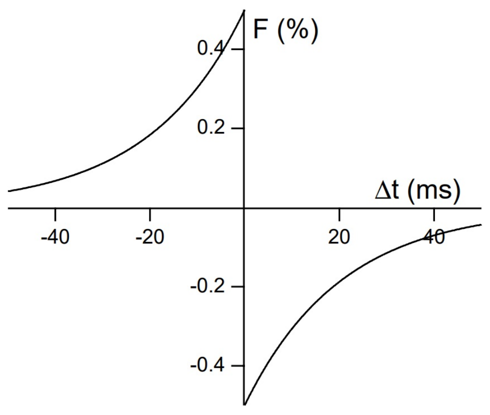


Figure 1.13: Additive pair based STDP modification function. F is the percentage variation of the synaptic weight, while Δt is the time of the presynaptic spike minus the time of the postsynaptic spike. Figure from [51].

Pair-based or canonical STDP (Fig. 1.12) is probably the most known and studied biologically inspired algorithm for local learning. It is an unsupervised training based on the time difference between a single spike of the presynaptic neuron and a single spike of the postsynaptic neuron in a temporal window of usually 10 to 100 ms. Also referred as Hebbian STDP, it is usually bidirectional (in time) and order-dependent, with presynap-

tic before postsynaptic spiking driving long-term potentiation (an increase of strength, related to causal events), and postsynaptic before presynaptic spiking driving long-term depression (decrease of strength, related to anti-causal events). Different variations of the algorithms have been implemented during the years, each with different features.
The additive STDP [51] can be expressed as (Fig. 1.13):

$$\Delta W(\Delta t) = \begin{cases} A_{+}e^{\frac{\Delta t}{\tau_{+}}} & \text{if } \ \Delta t > 0 \\ -A_{-}e^{-\frac{\Delta t}{\tau_{-}}} & \text{if } \ \Delta t < 0 \end{cases}$$

Where $\Delta W(t)$ is the synaptic modification function, Δt is the time difference between presynaptic and postsynaptic spike, τ_{+} and τ_{-} are two different time constants and A_{+} and $A-$ are the maximum weight change. The STDP rule has been proved to automatically achieve a balanced, irregular-firing state in which presynaptic and postsynaptic spike times are causally correlated over a wide range of input rates. Since presynaptic and postsynaptic correlated activity can occur purely by chance, rather than reflecting a causal relationship, to achieve stability is required that the integral of the synaptic modification function $\Delta W(t)$ is negative. This means that synaptic weakening must dominate synaptic strengthening to avoid a possible explosion of the value of the synapses. However, it must also be noted that if the synapses become too weak to make a neuron fire the neuron is lost and will not be able to fire anymore. Finally, a fundamental factor is the size of the time window. As L. F. Abbott et al. write in [51]: "The size of the temporal windows over which synaptic strengthening and weakening occur is critical in determining the effects of STDP. It would seem highly advantageous for window sizes to be different in various brain regions, to be modified during stages of development, and perhaps to be dynamically adjustable over shorter time scales as well".

A more efficient way to limit the synapses into a certain range is to normalize the value of variation by multiplying the positive term with the inverse of the weight W [33]:

$$\Delta W(\Delta t) = \begin{cases} \frac{1}{W}A_{+}e^{\frac{\Delta t}{\tau_{+}}} & \text{if } \ \Delta t > 0 \\ -A_{-}e^{-\frac{\Delta t}{\tau_{-}}} & \text{if } \ \Delta t < 0 \end{cases}$$

Or with an exponential term that takes into account of the weight:

$$\Delta W(\Delta t) = \begin{cases} e^{-W}A_{+}e^{\frac{\Delta t}{\tau_{+}}} & \text{if } \ \Delta t > 0 \\ -A_{-}e^{-\frac{\Delta t}{\tau_{-}}} & \text{if } \ \Delta t < 0 \end{cases}$$

This additional expression of STDP not only normalizes the variation, but also breaks the asymmetric learning of one agent of the neuron. When one neuron starts to gain weight, it experiences a more difficult change to learn further. This effect enhances the learning of the other agents and a fair distribution of synaptic values.

STDP has been proved to be able to perform competitive learning [51] [33] [9] and with the latter algorithm also a Winner-Take-All (WTA) approach. B. Nessler, W. Mass et al. [9] demonstrated that using a probabilistic model STDP is able to perform a "spike-based Expectation Maximization". This means that the WTA approach converges to at least a local optimum in the fitting of the model to the distribution of high-dimensional spike inputs.
One of the most functional SNNs that has been recently developed is the STDP-based spiking deep neural network (SDNN) [48]. This network has been proved to be able to achieve and accuracy of 99.1% on Caltech face/motorbike and 82.8% on ETH-80 datasets only using unsupervised local learning. The learning methods consist of a simplified version of the STDP that takes into account only if the spike difference is positive or negative and normalizes the synapse change using the value of the weight. Furthermore, a simplified but general version of the refractory time has been implemented: a neuron can fire only one time for each image and also the firing of a neuron inhibits the firing of all other neurons that share the same location in different neuronal maps of the same layer. This provides a sparse but highly informative coding, because it indicates the existence of a particular visual feature in a specific location without redundancy.

1.6.2. Triple STDP

Experimental data

Even if pair based STDP has achieved astonishing results, it fails to reproduce some experimental data of neural cells. Sjöström et al. have demonstrated [41] that long-term plasticity in tufted L5 neurons (situated in the visual cortex) depends jointly on firing rate, spike timing and cooperativity among inputs. The experiments have shown that the time window of STDP is in a range of Δt from -50 ms to 20 ms. In this range of pair based spike time difference, the frequency plays a dominant role. From 0.1 to 20 Hz (low frequency) depression is predominant and does not show any frequency (Fig. 1.14 **d** and **f**) or postsynaptic depolarization dependence (Fig. 1.14 **a** and **c**). However, potentiation seems highly affected by frequency (Fig. 1.14 **d** and **f**) and postsynaptic depolarization (Fig. 1.14 **a** and **c**). Furthermore, unlike spike-timing LTD, low frequency pairing of unitary inputs and a single postsynaptic spike could not induce LTP in the absence of preceding postsynaptic depolarization or extracellular stimulation (Fig. 1.14 **b**). This

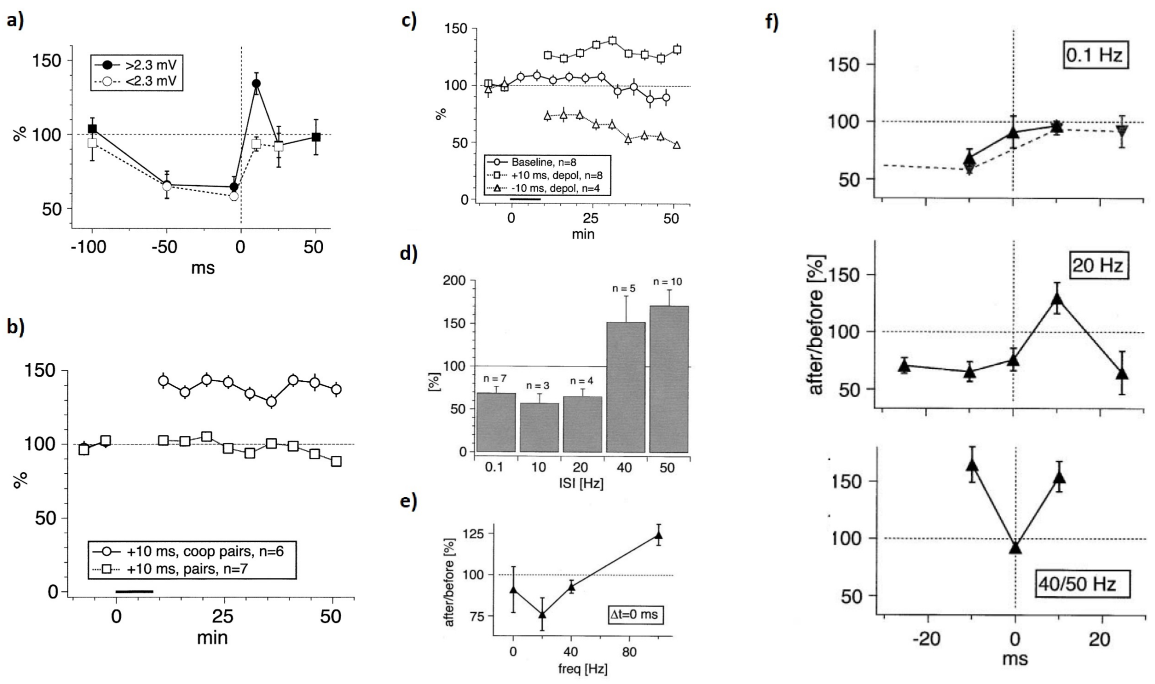


Figure 1.14: **a**) Synaptic change as a function of time between presynaptic and postsynaptic spike and the total postsynaptic depolarization. **b**) LTP induced with (circles) and without (squares) extracellular stimuli with $\Delta t = 10$ ms. "n" is the number of measurements. **c**) Postsynaptic spikes evoked during depolarization to -45.1 mV. **d**) LTD as a function of frequency. It disappears above 40 Hz. **e**) Synchronous firing as a function of frequency. **f**) Synaptic change as a function of time between presynaptic and postsynaptic spike and frequency. Figures from [41].

means that cooperativity from other neurons is needed at low frequency to induce LTP. Above 20 Hz (high frequency), potentiation was more prevalent (Fig. 1.14 **d**). Since the membrane potential did not quite repolarize to rest, the amount of residual depolarization increased proportional to frequency and permitted robust LTP induction. This is in part due to the spike after-depolarization and in part due to the membrane time constant. The magnitude of LTP is inversely correlated with the initial synaptic strength and determined jointly by the membrane potential immediately preceding the postsynaptic spike, by the instantaneous frequency and the time difference between the spikes. Above 40 Hz, depression did not exist at all (Fig. 1.14 **d** and **f**). It happens because the frequency at which the net effect of these multiple pairings shifts from LTD to LTP is determined by the width of the LTP window (Δt of 20 ms i.e. 50 Hz). This absence of LTD at high frequency suggests that other mechanisms must contribute to maintaining stability in the face of plasticity. Finally, synchronous firing at frequencies up to 40 Hz resulted in

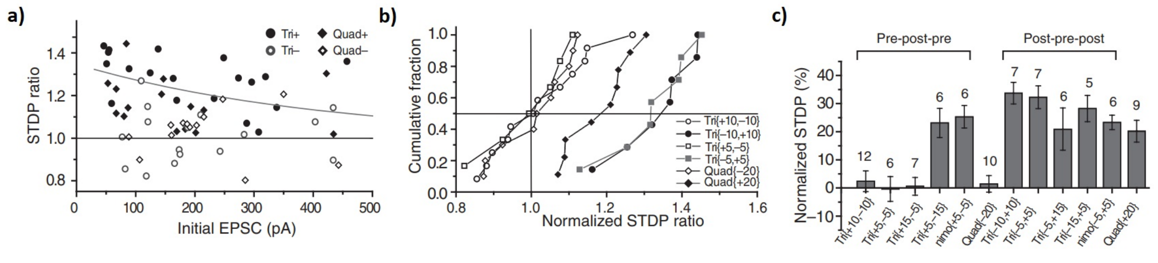


Figure 1.15: Synaptic change in triplet and quadruplet experiments. Positive time intervals means that the presynaptic spike precedes the postsynaptic one. **a**) Synaptic change dependence on initial synaptic strength. The Tri+ data represents –10, +10, –5, +5 and –15, +5 ms triplets of spikes. Tri- includes +10, –10, +5, –5 and +15, –5 ms triplets. Quadruplet experiments are a couple of paired spike -5,+5 and +5,-5 ms. Quad+ represents measurements where the time interval from the midpoint of the depression-inducing spike pair to the midpoint of the potentiation-inducing pair is 25ms, while Quad- when it is -25ms **b**) Cumulative normalized STDP ratio. **c**) Histogram of mean normalized STDP. Figures from [24].

depression, whereas potentiation was obtained at 100 Hz (Fig. 1.14 **e**).

More interactions between spikes were provided by Wang et al. [24] (Fig 1.15). The interaction between triplets of spikes of a hippocampal neural culture seems to suggest that the integration of potentiation and depression processes is nonlinear and temporally asymmetric. The two processes cancel when potentiation is triggered first, while potentiation dominates when it is triggered second (Fig 1.15 **c**). Furthermore, even when depression was activated later, if the time difference between the spikes was less than the potentiation one, the overall synaptic change would result positive. LTP was proved again to be inversely proportional to the synaptic strength, while depression seems to be independent (Fig 1.15 **a**). Even quadruplets of spikes have shown potentiation in a time window of 70 ms, although they produce less potentiation than triplets (Fig 1.15 **a** and **b**). Finally, the potentiation was inversely proportional to the time difference between the depression pair and the potentiation pair.

Learning rules

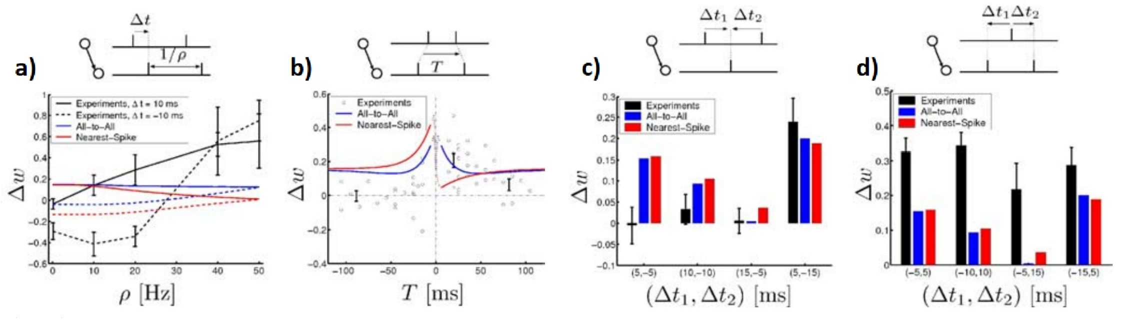


Figure 1.16: Paired STDP fails to reproduce experimental data from [41] and [24]. **a**) Synaptic change as a function of frequency with $\Delta t = +10$ ms (straight line) and $\Delta t = -10$ ms (dotted line). Comparison between data of [41] (black), All-to-All (blue) and nearest-spike (red) simulations. **b**) Comparison in synaptic change in quadruplet protocol from [24] (dots) and simulations (red and blue lines). Δt between couple of spikes of 5ms. **c,d**) Comparison in synaptic change in triplet protocol from [24] (black) and simulations (red and blue). Figures from [42].

It is easy to understand that canonical pair based STDP cannot represent a fitting model for the previous shown experimental data. It fails to show the dependence on frequency of the pairs of spike and cannot reproduce triplets or quadruplet experiments (Fig. 1.16). To better represent the experimental data, J. Pfister and W. Gerstner developed a model inspired by the process of glutamate release of postsynaptic neuron whenever a presynaptic spike arrives [42]. Glutamate that binds to postsynaptic receptors increases when a presynaptic spike happens and after decreases exponentially. They modeled four variables r_1, r_2, o_1, o_2 that change according to these equations:

$$\frac{dr_1(t)}{dt} = -\frac{r_1(t)}{\tau_+} + S_{pre}$$

$$\frac{dr_2(t)}{dt} = -\frac{r_2(t)}{\tau_x} + S_{pre}$$

$$\frac{do_1(t)}{dt} = -\frac{o_1(t)}{\tau_-} + S_{post}$$

$$\frac{do_2(t)}{dt} = -\frac{o_2(t)}{\tau_y} + S_{post}$$

Where τ_x and τ_y are time constant respectively larger than τ_+ and τ_-. S_{post} and S_{pre} are the trains of postsynaptic and presynaptic spikes respectively. The learning rule is based

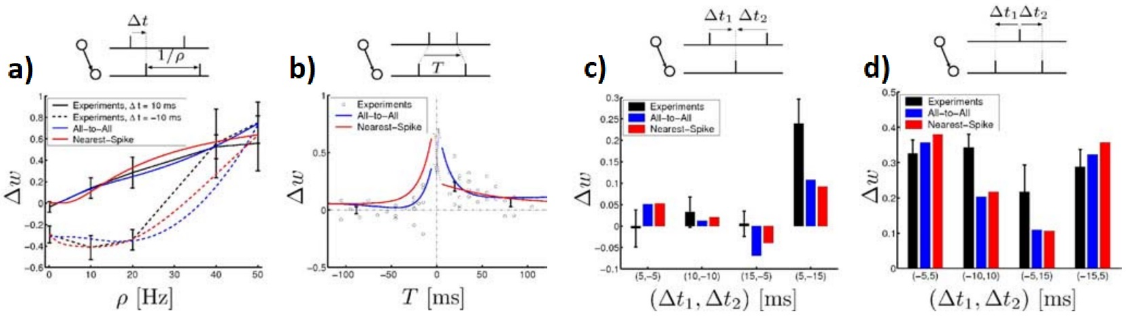


Figure 1.17: Triplet STDP represent data from [41] and [24]. **a**) Synaptic change as a function of frequency with $\Delta t = \pm 10$ ms. Comparison between data of [41], All-to-All and nearest-spike simulations. **b**) Comparison in synaptic change in quadruplet protocol from [24] and simulations. Δt between couple of spikes of 5ms. **c,d**)Comparison in synaptic change in triplet protocol from [24] and simulations. The simulations in figure **d** have a variance 4 times smaller than the pair protocol from experimental data. Figures from [42].

on the previous listed variables and it is and All-to-All interaction:

$$w(t+1) = w(t) - [o_1(t)(A_2^- + A_3^- r_2(t-\epsilon))] * S_{pre}$$

$$w(t+1) = w(t) + [r_1(t)(A_2^+ + A_3^+ o_2(t-\epsilon))] * S_{post}$$

Where A_2^+ and A_2^- are experimental parameters that control the weight change for paired spikes, while A_3^+ and A_3^- denote the amplitude of potentiation and depression for triplets of spikes. All four parameters are assumed greater than or equal to zero and ϵ is an infinitesimal positive quantity to ensure that the weight updates before r_2 and o_2. As it can be easily seen, when there are only pair of spikes r_2 and o_2 are equal to zero and the learning rule behaves like the canonical STDP. However, when triplets and quadruplets of spikes or pairs of spikes above a certain frequency are presented to the network, r_2 and o_2 are not zero anymore and the potentiation or depression is increased. The learning rule has undergone the same experiments of figure 1.16 and the results approximate much better the experimental data from [41] and [24].

Another algorithm that deserve to be mentioned is the triplet of spikes algorithm of J. Gjorgjievaa, J. Pfister et al. [29]. Their approach considers only pairs and triplets of spikes and expresses the weight change as a Volterra expansion of both presynaptic and postsynaptic spike trains:

$$\Delta W(\Delta t_1, \Delta t_2) = \begin{cases} A_3^+ e^{-\frac{\Delta t_1}{\tau_+}} e^{-\frac{\Delta t_2}{\tau_y}} & \text{if} \quad \Delta t_1 \geq 0, \Delta t_2 \geq 0 \\ A_2^- e^{-\frac{\Delta t_1}{\tau_-}} & \text{if} \;\; \Delta t_1 < 0 \end{cases}$$

where $\Delta t_1 = t^{post} - t^{pre}$ denotes the time difference between a postsynaptic and a presynaptic spike and $\Delta t_2 = t^{post} - t'^{post}$ denotes the time difference between the postsynaptic spike and the consecutive postsynaptic spike. τ_+, τ_- and τ_y are potentiation, depression and rate time constants. It is important to note that τ_- is the double of $\tau+$ in order make depression influence more the synaptic change than potentiation and that τ_y is at least 6 times higher than $\tau+$ in order to make the training enough frequency dependent.

The algorithm seems to induce selectivity in rate-based patterns in a similar way of the Bienenstock, Cooper and Munro (BCM) learning rule, which have been observed in biological STDP [19] and that could explain experience-dependent cortical plasticity such as orientation selectivity. BCM, which proposes a sliding threshold for LTP and LTD and states that synaptic plasticity is stabilized by a dynamic adaptation of the time-averaged postsynaptic activity, has been shown to maximize the selectivity of the postsynaptic neuron in a permissive middle range of firing frequency, without letting the synaptic weight explode or go null. Furthermore, this triplet STDP seems to be able to induces selectivity also with third order correlation based patterns, differently from pair based STDP that was only able to induce second order correlations. To validate these simulations results there are also experimental data obtained from memristive devices in [59].

1.6.3. Membrane voltage based STDP

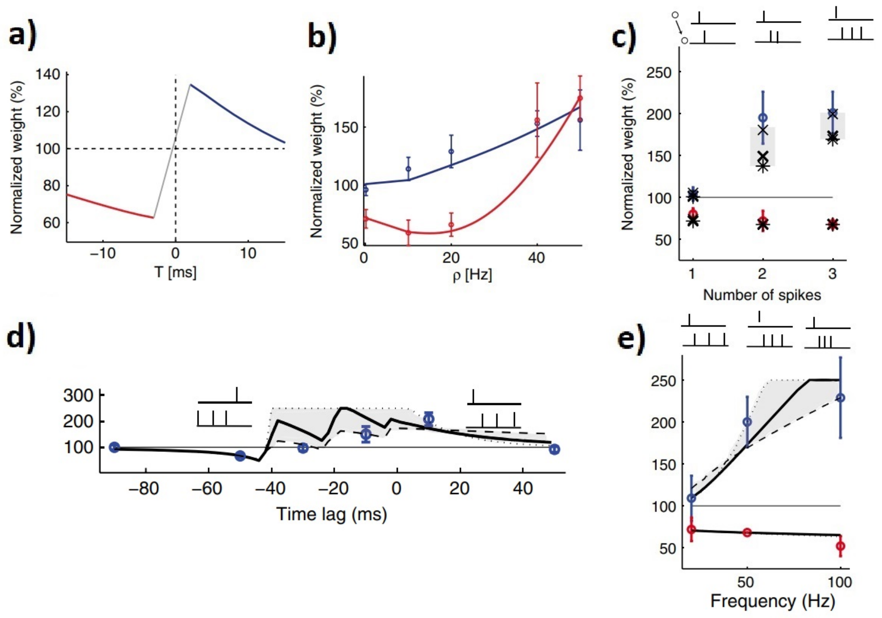


Figure 1.18: **a** and **b** represent the fitting of the model to the experimental data from [41], while **c** and **d** represent the fitting of experimental data from [24]. **a**) Simulation of the weight change with different time intervals Δt between presynaptic and postsynaptic spikes at a frequency of 20Hz. **b**) Normalized weight change at different frequency with constant Δt of 10ms (blue) and -10ms (red). The dots are experimental data. **c**) Weight change as a function of the number of postsynaptic spikes at 50 Hz. The presynaptic spike was paired +10 ms before the first postsynaptic spike (blue) or –10 ms after (red). Dots are experimental data and crosses are simulations. **d**) Synaptic weight change with one presynaptic spike and three postsynaptic spikes at different Δt at 50 Hz frequency. Dots represent experimental data and black and dashed line simulations with different parameters. **e** Synaptic weight change with one presynaptic spike and three postsynaptic spikes at different frequency. Dots represent experimental data and black and dashed line simulations with different parameters. Figures from [15].

Previous explained training algorithms fits more and more accurately the experimental data [41] [24] regarding the spike timing plasticity, but some data [41] shows also a dependence on the postsynaptic neuron membrane potential. The algorithm developed by C. Clopath, W. Gerstner et al. [15] seems to preserve most of the features of the previous algorithms even if it is based on presynaptic spike arrival and postsynaptic potential i.e. takes into account the voltage dependence. The neuron model used for the simulations [15] is an adaptive exponential integrate-and-fire (AdEx) and thus it could lead to different results if a normal integrated and fire (the one used for this project) would have been used instead. However, a working voltage dependent approach is very rare in literature and therefore it is worth mentioning. It could be adapted to LIF neurons and be an inspiration for a future hardware implementation.

The algorithm is based on the idea that potentiation and depression happen when the presynaptic neuron spikes, depending on the instantaneous membrane voltage of the post-synaptic neuron. Unfortunately, this process would retain network's information only for a narrow time window. It depends on the decay time of the membrane voltage, which is usually too fast to fit the experimental data. To solve this problem two different low pass filtered versions of the postsynaptic membrane (one for potentiation and one for depression) along with a low pass filtered version of the presynaptic spike train have been introduced. This way, the training algorithm's memory of the network can be increased or decreased at will and it is possible to create a nonlinear voltage dependence. The method can be summarized as:

$$\begin{cases} \Delta W = A_+\overline{x_i}(u-\theta_+)_+(\overline{u}_+ - \theta_-)_+ - A_-(\overline{\overline{u}})X_i(t)(\overline{u}_- - \theta_-)_+ \\ \tau_-\frac{d}{dt}\overline{u}_-(t) = -\overline{u}_-(t) + u(t) \\ \tau_+\frac{d}{dt}\overline{u}_+(t) = -\overline{u}_+(t) + u(t) \\ \tau_x\frac{d}{dt}\overline{x}_i(t) = -\overline{x}_i(t) + X_i(t) \\ A_-(\overline{\overline{u}}) = A_{0-}\frac{\overline{\overline{u}}^2}{u_{ref}} \end{cases}$$

This is combined with the hard bounds $W_{min} \leq W \leq W_{max}$.
In the system $u(t)$ is the postsynaptic membrane potential, $\overline{u}_+$ and $\overline{u}_-$ its low pass filtered versions for potentiation and depression respectively and $\overline{x}_i(t)$ is the low pass filtered version of the presynaptic spike train $X_i(t) = \sum_n \delta(t - t_i^n)$. $\tau_+ \simeq 10ms$, $\tau_- \simeq 1s$ and τ_x are the time constant of the low filtered variables. A_+ and A_- are amplitude parameters, the first one is a constant, while the second is regulated by the control of an homeostatic process $\overline{\overline{u}}$ [56]. θ_+ and θ_- are threshold that regulates the activation of the STDP when the relative version of the membrane potential crosses them. Finally, the symbol $(\cdot)_+$ indicate rectification, which is equivalent to the operation $max(0, \cdot)$.

It can be easily noted that at low frequency LTP cannot happen if the threshold θ_+ is relatively high, while LTD can happen if θ_+ is relatively low. In Figure 1.18 is shown the ability of the algorithm to have time and frequency dependence similar to the experimental data from [41] and triplets and quadruplets response similar to [24]. The result shows that this method can fit the experimental data in a similar way to the triplet rule and adding the voltage dependence, which is absent in the previous explained biologically inspired learning rules.

1.6.4. Shallow supervised STDP

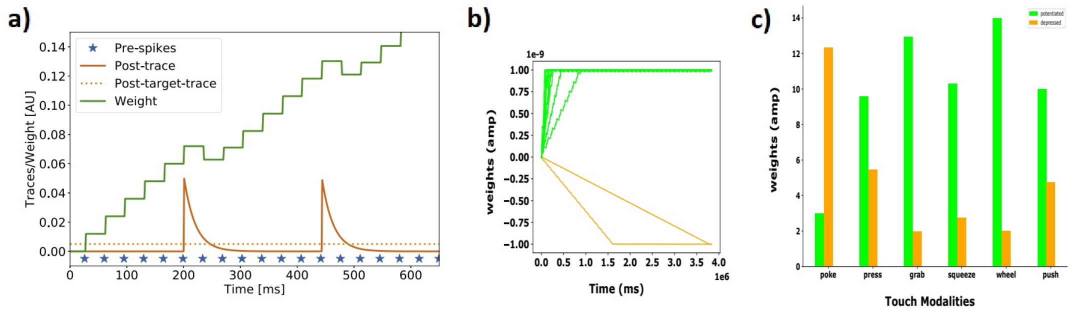


Figure 1.19: **a** Simulated traces of a hidden neuron, an output neuron and the linking synapse during supervised training from [8]. **b** Simulated weight development during training. Figure from [6]. **c** Final state of the simulations of the second layer of synapses from the network of [6]. Figure from [6].

Usually, biologically inspired training rule are unsupervised since there is no clear idea of the global learning in the brain. However, B. Illing, W. Gerstner and J. Brea [8] were able to think of an algorithm able to perform supervised training to the last layer of a SNN in a biologically plausible way. The algorithm is based on the simple concept that each output neuron must have a postsynaptic target trace, which represents the spiking activity that each output neuron should have according to the input presented to the network. Through a STDP like method, the synapses linking hidden and output layers get increased or decreased in order to make each output neuron's mean firing activity as close as possible to the postsynaptic spike trace. In fact, given an input-output pair, where the input is converted into spike trains, the network tends to converge in a state where the firing rates of the output neurons represent the wanted targets. This training method has been used by A. Dabbous, C. Bartolozzi et al. [6] to solve a 6 class classification problem with a 3 layers network. The first layer is composed by 160 neurons responsible for converting the input voltage into spike trains. The second layer is composed by 16 neurons and linked to the first by a randomly generated binary matrix (1 connected, 0 disconnected) in order to compress the input information. Since there were 6 classes, the output layer was composed by 6 neurons that would give a binary output according to their firing rate: output equal 1 for the neuron with the higher firing rate and 0 the others.

The training algorithm for training the synapses connecting hidden and output was:

$$\begin{cases} \tau_{tr}\frac{dtr_i(t)}{dt} = -tr_i(t) + A \cdot S_{out_i}(t) \\ \Delta W_{ji}(t) = \eta(tg_i(t) - tr_i(t)) \cdot S_{hid_j}(t) \end{cases}$$

Where i represent an output neuron, j an hidden neuron and W_{ji} is the synaptic weight linking the two neurons. tg_i and tr_i are respectively the output neuron i target trace and spike trace, with τ_{tr} the time constant of the latter. $S_{hid_j}(t)$ and S_{out_i} are respectively the Spike trains of the hidden neuron j and the output neuron i, η is the learning rate and A is a constant value. The algorithm updates only when a presynaptic spike occurs by taking into account the difference between the target trace of the postsynaptic neuron and its spike trace.

To further increase the accuracy, an additional inhibitory layer was added to the network during the testing phase. This layer, concatenated to the output layer, consists of 6 inhibitory neurons each of which is connected with only one output neuron. Through lateral inhibitory connections, these neurons tend to either prevent from firing or reduce the firing rate of the output neurons not directly connected to them. This method, similar to the location-based inhibition of [48], increases the competition between neurons at the output layer. With this final modification the network was able to achieve an overall accuracy of 88.3% on a dataset of 60 samples of touch modalities. 4 out of 6 classes achieved an accuracy of 90%, one an accuracy of 70% and one an accuracy of 100%. These results are comparable if not even better than other machine learning approaches.

1.7. Mathematical optimization algorithms

As previously mentioned, mathematical optimization algorithms are a derivation of standard backpropagation and unlikely biological inspired algorithms are mainly of the supervised and reinforcement paradigm. The hardware-friendly learning methods can be divided in three major categories. The first and most intuitive is to translate traditional trained "rate-based" ANN into a SNN. The second is smoothing the network model to be continuously differentiable. It can consist of modifying the activation function with soft non-linearity or in changing the model approach and considering the spiking probability, rate or time as the output of the neuron as valid information, instead of the single spike itself. The third method is to use surrogate derivative to relax the non linearity of the gradient. It can be obtained by replacing the spiking nonlinearity derivative by the derivative of a continuously differentiable function. In this way, standard learning as backpropagation through time and real time recurrent learning can be applied. Alternatively, the surrogate gradient can be applied to the loss function itself using feedback alignment or, more in general, random backpropagation matrices. The error can be first backpropagated and then a surrogate loss function and its actual derivative can be locally calculated or the gradient of the surrogate loss function can be calculated and then a matrix can be used to backpropagate it.

1.7.1. ANN converted in SNN

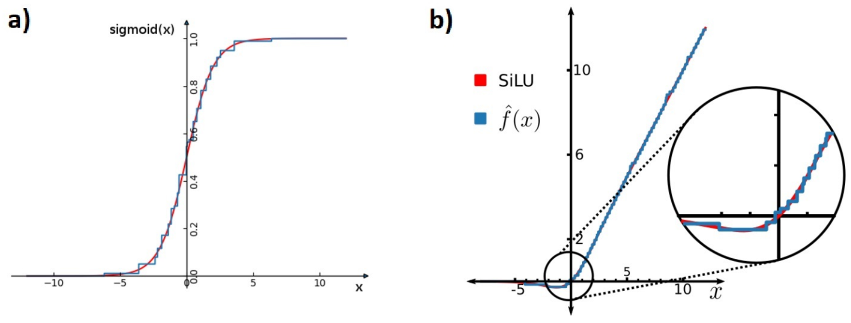


Figure 1.20: **a**) Sigmoid function (red) approximated by AMOS cell (blue) in 8 time steps. Figure from [13]. **b**) SiLu ($x \cdot sigmoid(x)$) function, which contains more complex non linearities than the ReLU function, approximated by an FS-neuron in 16 time steps. Figure from [14].

The method of supervised training based on the conversion of an ANN into a SNN is the most used in literature and probably the most efficient in term of accuracy when it comes to the simulation results. It uses a pre-trained ANN by backpropagation and converts it into a SNN that carries information through rate. By using special cells made by recurrently connected spiking neurons like At Most One Spike (AMOS) [13] or Few-spike (FS) neurons [14] it is also possible to recreate a rate version of some activation functions (Figure 1.20). Unfortunately, rate based networks require a lot of spikes and consequently more power dissipation that other methods. This fact is in opposition to the aim of this project to implement a network with the lowest possible power dissipation and thus rate-based networks have not been taken much in consideration. However, it is important to denote that J. Büchel, D. R. Muir et al. have developed a training algorithm able to fully exploit the temporal memory of LIF neurons and classify time-variant inputs with this method [28]. Training SNN to recognise temporal non-linear sequence is not a simple task. Usually RNN are used for this task and trained with backpropagation through time (BPTT) by backpropagating errors thought the unrolled network and changing the weight of the recurrent connections. In the SNN case there are no recurrent connections

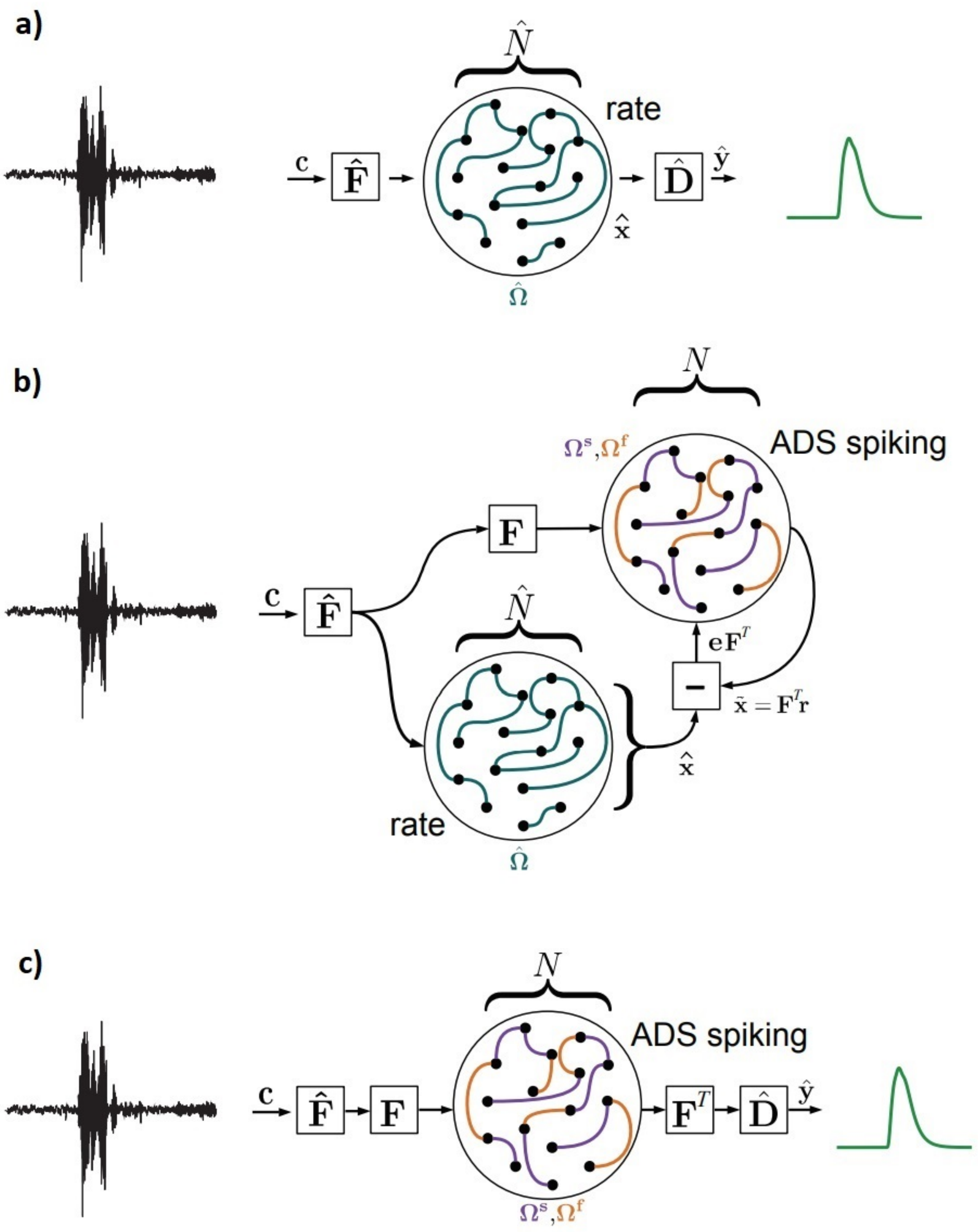


Figure 1.21: **a**) Schematic of the recurrent rate-based neural network with of size $\hat{N}$, with encoding and decoding feedforward connections $\hat{\mathbf{F}}$ and $\hat{\mathbf{D}}$, trained to map $\mathbf{c}(t)$ into $\mathbf{y}(t)$ resulting in the internal temporal representation of neural activity $\hat{x}(t)$. **b**) Schematic of the training of ADS by using the rate-based RNN. The error is calculated from the difference between the RNN output and the filtered and encoded ADS output. The error is then encoded and fed back to the ADS. **c**) Schematic of the resulting classifier composed by two encoding matrices, the ADS network and two decoding matrices. Figure from [28].

because the memory of time is in the neuron itself and the only parameters that a BPTT with a pseudo-gradient could affect are the non recurrent synapses. With only forward synapses as variables for the algorithm, it is very difficult to make the network converge close enough to the global minimum. This happens mainly because the gradient of the current time step and the previous one could be in contrast with each other. A way to control the temporal memory may be to use the synapses time constant and leakage current, but there is not a definite algorithm to minimize a possible loss function with these variables and thus the optimum values can be found only heuristically [36].

The algorithm developed by J. Büchel, D. R. Muir et al. consist in an Efficient Balanced Network (EBN), referred as Arbitrary Dynamical System (ADS) whose training is based on the outputs of a rate-based RNN. The task is defined as mapping a d_1 dimensional input $\mathbf{c}(\mathbf{t})$ into the d_2 dimensional output $\hat{\mathbf{y}}(\mathbf{t})$ and was firstly implemented in a non-spiking EBN in the form of a mono layer rate-based RNN with this dynamic:

$$\begin{cases} \tau\dot{\hat{\mathbf{x}}}(t) = \hat{\mathbf{\Omega}} f(\hat{\mathbf{x}}(t)) + \hat{\mathbf{F}}\mathbf{c}(t) + b + \epsilon \\ \hat{\mathbf{y}}(t) = \hat{\mathbf{D}}\hat{\mathbf{x}}(t) \end{cases}$$

Where $\hat{\mathbf{\Omega}}$ is the $\hat{N}$x$\hat{N}$ ($\hat{N}$ is the number of neurons in the layer) matrix of recurrent connections, $\dot{\hat{\mathbf{x}}}(t)$ is the $\hat{N}$ dimensional output array of the neurons, b is the bias, $f(\cdot) = tanh(\cdot)$ is the activation function and ϵ is a noise factor. Rate-based neurons behaves a little different than standard perceptrons, because their output does not reset every time a new input is injected, but it decays over time (τ is the decay time constant). Furthermore, every time a new input is presented to the network, the new inference output is summed to the previous decayed inference output. In this case, it was also chosen to add decoding and encoding weight matrices $\hat{\mathbf{D}}$ and $\hat{\mathbf{F}}$ with respective dimensions $\hat{N}$xd_2 and d_1x$\hat{N}$ in order to mach input and output dimensions with the size of the network. The decoder weights were initialised using a standard normal distribution and the encoder as $\hat{\mathbf{F}} = \hat{\mathbf{D}}^{\mathbf{T}}$. BPTT or any other suitable approach can be used to train the rate network.

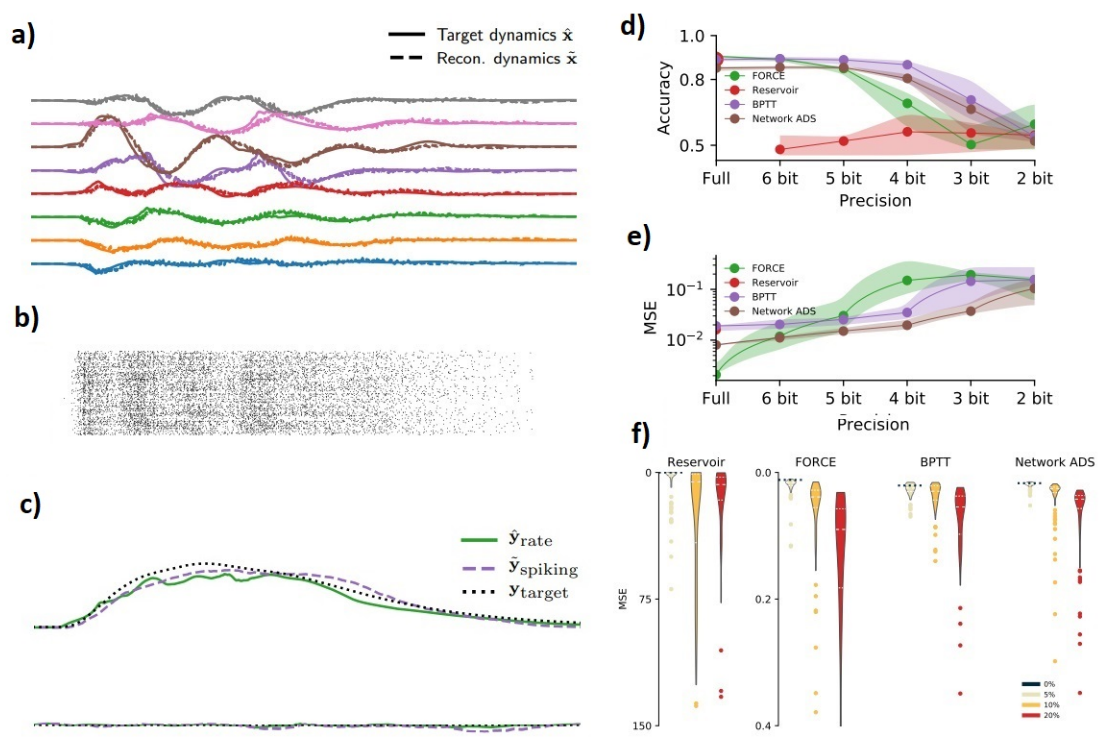


Figure 1.22: **a**) The filtered internal ADS network dynamics are able to match the RNN ones after the training. **b**) Spiking activity of the ADS network. **c**) Whether an input was classified to a specific class the corresponding output signal was high (top trace), otherwise it would remain low (bottom trace). **d**) and **e**) are the accuracy and MSE comparison respect to quantisation noise between this method and the FORCE method, a reservoir network, and RNN trained with standard BPTT. **f**) Comparison of resilience to hardware mismatch. Figure from [28].

Once the RNN network is trained, the next step is to train the ADS network to behave like $\dot{\hat{\mathbf{x}}}(t)$. The network training dynamics (Fig. 1.21) can be expressed as:

$$\begin{cases} \dot{V} = -\lambda V + \hat{\mathbf{F}}\mathbf{F}\mathbf{c}(t) - \mathbf{\Omega}^{\mathbf{f}} S_{neu}(t) + \mathbf{\Omega}^{\mathbf{s}} S_{neu}(t) + k\mathbf{D}e(t) \\ \dot{\mathbf{\Omega}}^{\mathbf{s}} = \mu \mathbf{r}(t)(\mathbf{F}e(t))^T \\ \tilde{x} = \mathbf{F}r \\ e = \tilde{x} - \hat{x} \\ \mathbf{\Omega}^{\mathbf{f}} = b(\mathbf{F}\mathbf{D} + a\mathbf{I}) \\ \mathbf{F} = \mathbf{D}^{\mathbf{T}} \end{cases}$$

Where V is the array of leaky membrane of the N spiking neurons ($N = 5\hat{N}$), λ the leak rate, S_{neu} is the array of spike trains of the neurons, r the spiking output of the ADS network, μ the learning rate, $\mathbf{\Omega^s}$ are the recurrent connections of size NxN where the pseudo-backpropagation is performed and e is the error between the filtered spiking output and the rate-based RNN output. F and D are feedforward weights of dimensions $\hat{N}$xN and Nx$\hat{N}$ needed for encoding and decoding the input and output of the network. Furthermore, Ω^f provides fast balanced feedback and a and b are more complex parameters that have been simplified for the sake of simplicity. Finally, k is a parameter that decays to zero over time and is needed to force the ADS network output to remain close to the desired target dynamics until the learning is completed.

The previous explained algorithm have been simulated and compared with different training approaches. It seems to perform in similarly to other trainings in a ideal simulation, but the peculiarity of this approach seems to be his robustness to hardware mismatch and quantisation noise (Fig. 1.22 **d**), **e**) and **f**)). This would lead to a great advantage in term of accuracy and reliability with the downside of a more complex implementation and an increase of the synaptic connections.

1.7.2. Smoothed Spiking Neural Networks

The peculiarity of this supervised learning method is the network formulation, which ensures well-behaved gradients that are directly suitable for optimization. Since the neuron model used for this work is a defined LIF neuron, these algorithms have not been considered to be implemented. However, they are widely used in literature and thus the four major kinds will be quickly explained: soft nonlinearity, probabilistic, rate and temporal models.

Soft nonlinearity models

The soft nonlinearity model is implemented by replacing the Heaviside's activation function of SNN with a gated function that can be continuous or more realistically discrete. The function provides an activation of the synapses related to the membrane voltage of their presynaptic neuron [25]. In a discrete implementation there can be different thresholds that increase the synaptic activation gradually.

Probabilistic models

The spiking activity of a neuron averaged over several samples defines its probability density. The binary probabilistic models are based on the concept that the log-likelihood of a spike train is a smooth quantity, which can be optimized using gradient descent. So when it comes to on stochastic neurons the focus is not anymore on the exact value of the synapses, but on the spiking probability that comes from a specific presynaptic neuron respect to the postsynaptic one. A peculiar fact about these networks is that noise can be injected to smooth out the effect of binary nonlinearity.

Rate-coding networks

In rate-based models there is not a modification of the neuron model, but rather at the information level, which is carried out by the spiking rate. Suprathreshold firing rate averaged over a sufficiently large time intervals behaves as a quasi continuous function of the input current and thus permits gradient-based optimization. Differently from stochastic models, they are deterministic networks, which always emit a fixed-integer number of spikes for a given input, with frequency as the only difference. As mentioned, these models perform very well at the expense of low power efficiency. They require either a relatively high firing rate or long averaging time to achieve decent accuracy and thus are not suited for ultra-low power implementations.

Single-spike time-coding networks

The temporal-based methods do not require a modification of the neuron model. In this case, the information is encoded in the firing time that can carry high information in short observation windows just by using one spike. The downside is that every neuron in the network needs to emit one spike per trial, because there is no clear gradient definition for quiescent units. This is a trait that could be considered in contrast with power efficiency.

1.7.3. Surrogate gradients

When surrogate gradients are used there is no modification to the SNN model. There are two ways of applying this method. The first one consists of changing the optimization algorithms by using a virtual surrogate loss function. It approximates the gradient by replacing weights in the backward path of the backpropagation with feedback alignments i.e. random ones. The network would adjust its feedforward weights such that they partially align with the random feedback matrix and thus permitting to convey useful error information (i.e the backpropagated gradient sign and its value, which is close to the real one). The second methodology does not change the algorithms. Instead, it just replaces the Heaviside derivative by the derivative of a continuously differentiable function.

A perfect example that uses both the typologies is SuperSpike [18]. This algorithm is an optimization procedure for time variant classification that leads the output neurons of a SNN to spike at predefined times. In order to do so, a loss function in discrete time is defined as:

$$L = \frac{1}{2}\sum_{n,j}(\lambda * (S_{neu_j}[n] - S^{tg}_{neu_j}[n]))^2$$

Where λ is the continuous-time kernel function that defines the spikes, S_{neu_j} is the spike train of the output neuron j, $S^{tg}_{neu_j}$ the target spike train for the output neuron k and $*$ the convolution symbol. Considering the error as $e[n] = \frac{dL}{dn} = \lambda * (S_{neu_j}[n] - S^{tg}_{neu_j}[n])$, a learning rule for last matrix weights can be defined as:

$$\Delta W^o_{ij}[n] = \eta e_j[n][\sigma'(V_{mem_j}[n] - V_{th})\frac{dV_{mem_j}[n]}{dW^o_{ij}}]$$

where W^o_{ij} is the weight linking the last hidden layer neuron i to the output neuron j, η is the learning rate, $V_{mem_j}[n]$ is the membrane potential of the output neuron j at discrete time n, V_{th} is the threshold voltage of the Heaviside activation function of the LIF neuron while $\sigma'(\cdot)$ is its pseudo or surrogate derivative (i.e. derivative of ReLu, logistic or other activation functions). When applied to multilayer networks, SuperSpike propagates the error directly to deeper layers by using random backpropagation:

$$\Delta W^l_{ik}[n] = \eta[\sum_j G_{ij} e_j[n]][\sigma'(V_{mem_i}[n] - V_{th})\frac{dV_{mem_i}[n]}{dW^l_{ik}}]$$

Where W^l_{ik} is the weight linking the neuron i of the hidden layer $l-1$ to the neuron k of the hidden layer l and G is the random matrix that performs error propagation.

SuperSpike is an example of how random backpropagation can be used to propagate the error backward, but depending on the technology and the network size this method can result not inefficient. Random matrices can eventually be used to map the output of the hidden layer into arrays that have the same dimension of the pseudo target. This way each layer has its own loss function and the weight variation can be calculated locally, like in [18]:

$$\Delta W^l_i[n] = \eta\sigma'(a^l_i[n])\frac{dL^l(G^l y^l[n], y^{l^{tg}}[n])}{dy^l_i[n]}$$

Where G^l is a fixed random matrix for each layer that projects the layer l to an array of dimension of the pseudo-target $y^{l^{tg}}$ and $a^l_i[n]$ is the sum of the inputs to the neuron i of the layer l.

1.8. Conclusion

This chapter can be considered an overview of what the most known training for SNN consist and can be catalogued. The first part was specifically made in order to permit the comprehension of this subject also to electronics engineers new to the field. It is fundamental to clarify some concepts that can result confused in nowadays literature. The second part deepens the State of the Art for the training of SNNs. Different methods have been catalogued, explained and compared to better understand the main ideas that led to the choice of the most common training algorithms. The choice and the algorithm itself will be explained in the following chapters.

2 | Description of the reference artificial neural network

2.1. Introduction

After the study of possible implementations, the neuromorphic circuit proposed first by I. Sourikopoulos et al. [26] and then modified by Mastella [34] and after that realized on chip by Polidori E. [44] and Polidori C. [43] is analyzed in this Chapter with all the final realizations. The Chapter also treats the PCB implemented to test the chip [44].

2.2. Chip realization

The neuromorphic chip was realized in order to gather together the bio-plausible feature with a low power implementation. The chip (Fig. 2.1) was entirely made with CMOS technology and is composed by three main circuits:

- **The neuron**. There are three neuron circuits on the chip: two predisposed as presynaptic neurons and one as postsynaptic. Both the presynaptic neurons are linked to the postsynaptic by a synapse. From now on the presynaptic neurons will be addressed as presynaptic neuron 1 and presynaptic neuron 2, while the postsynaptic one just as postsynaptic neuron.
- **The synapse**. Two floating-gate synapses link the presynaptic neuron 1 and presynaptic neuron 2 to the postsynaptic neuron. When a presynaptic neuron spikes the relative synapse injects a current proportional to their respective weight into the postsynaptic neuron and increases its membrane voltage. Only positive synapses have been implemented on this chip.
- **STDP circuit** The Spike Time Dependent Plasticity circuit is needed to reshape the neurons spikes to a double wave rectangular signal that enables the weight change of the synapses. This signal is then increased to higher voltages in order to permit the tunneling of charge through the floating gate of the synapses.

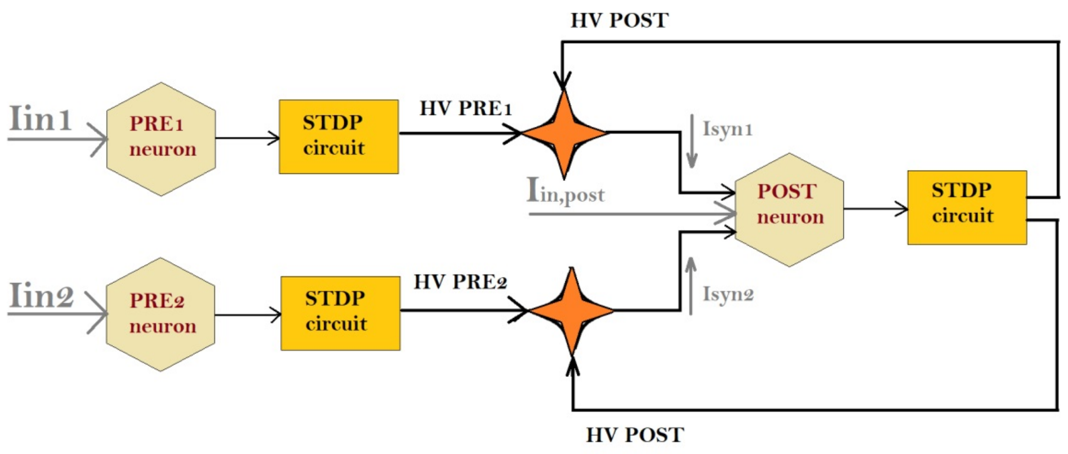


Figure 2.1: Block scheme of the neuromorphic chip. The chip is composed by three neurons, three STDP circuits and two synapses. The neurons have been arranged in order to have two presynaptic neurons and one postsynaptic. The presynaptic neuron 1 and the postsynaptic neuron share all the power supply, while the presynaptic neuron 2 does not share some of the reference voltage that involves starvers current generators. An input current can be injected in all three neurons separately. Figure from [44]

The design has been carried out exploiting the LFoundry 150nm CMOS technology. The toolkit comprises transistors with different voltage ratings: 1.8V, 3.3V, 5V.

2.2.1. The neuron

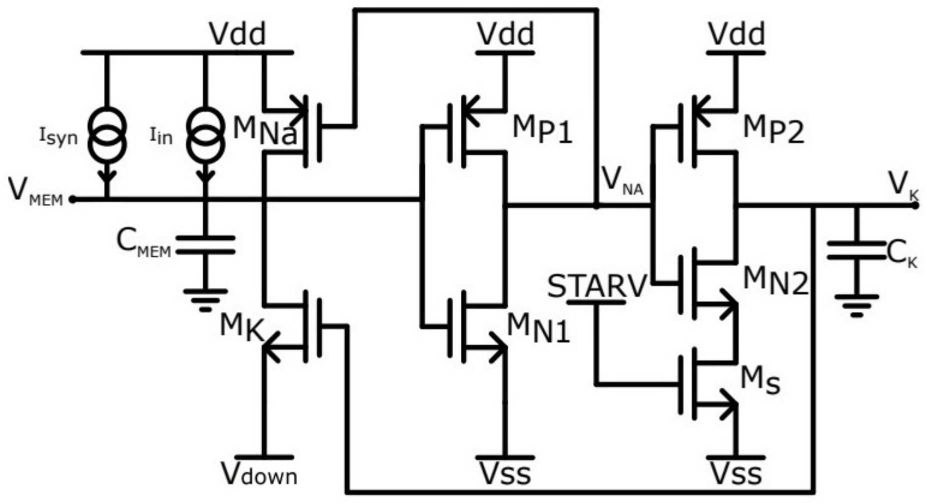


Figure 2.2: Circuital scheme of the designed neuron. The external input current and the synapse current have been modeled respectively with current generators I_{in} and I_{syn}. The membrane capacitance C_{mem} was chosen of about 50fF to be greater than the parasitic capacitances at V_{mem} node. Figure modified from [34]

Transistors sizing

	M_{Na}	M_k	M_{p1}	M_{n1}	M_{p2}	M_{n2}	M_S
$\frac{W}{L}$	$\frac{320nm}{320nm}$	$\frac{1\mu m}{320nm}$	$\frac{1\mu m}{150nm}$	$\frac{1.76\mu m}{350nm}$	$\frac{320nm}{150nm}$	$\frac{320nm}{640nm}$	$\frac{320nm}{1\mu m}$

Table 2.1: Sizing of the transistors of the neuron circuit in Fig. 2.2

The schematic of the neuron is reported in Fig. 2.2. The circuit takes advantages of the physical analogy of electronics with the biology to create a very compact and efficient spiking neuron. In order to resemble a biological neuron, the circuit needs to be able to emulate the neuron membrane and all its states (Fig. 5): the resting state, the depolarization state, the repolarization state and the hyperpolarization state. In order to consume as less power as possible the circuit is powered between $V_{dd} = 0.4V$ and $V_{ss} = 0V$ and V_{down} is controlled with an external trimmer for testing. In quiet conditions, the whole current drained was simulated to be $I_{tot} \approx 2pA$ leading to a dissipated power of $800fW$. When a spike arrives the energy consumed by the whole neuron to create the designed signal should be around $21fJ/spike$. Finally, the neuron occupies an area of $74m^2$.

The cell membrane is modelled as a capacitor C_{mem} and the potassium and sodium ionic channels are implemented respectively by the MOS transistors M_K and M_{Na}. The neuron dynamics are determined by the value of V_{mem}, which is defined by the current injected into C_{mem}:

$$I_{C_{mem}} = I_{Na} - I_k + I_{syn} + I_{in} \tag{2.1}$$

Depending on this equation balance the neuron can enter in a different state.

Resting state

When no current is injected into C_{mem}, the neuron remains in a resting state. At the equilibrium point, the current balance in equation 2.1 becomes:

$$0 = I_{Na} - I_k \tag{2.2}$$

This condition is met when $V_{gs}^K \approx 0V$ and $V_{sg}^{Na} \approx 0V$. This can only happens when the output of the first inverter $V_{Na} = V_{dd}$, while the output of the second inverter $V_K = V_{ss}$. The transistors are sized in order to reach equilibrium $V_{mem} = V_{rest} \approx 0V$ when $V_{down} = V_{ss} = 0V$.

To calculate the exact value os V_{rest} and its relationship with V_{down} we can consider that with $V_{dd} = 0.4V$ the MOSFETs operate in subthreshold regime:

$$I_{p_0}\left(\frac{W}{L}\right)_{Na} e^{\frac{V_{sg}}{n_p V_{thr}}}[1 - e^{-\frac{V_{ds}}{V_{thr}}}] = I_{n_0}\left(\frac{W}{L}\right)_{K} e^{\frac{V_{gs}}{n_n V_{thr}}}[1 - e^{-\frac{V_{ds}}{V_{thr}}}]$$

since $V_{sg} \approx 0V$:

$$I_{p_0}\left(\frac{W}{L}\right)_{Na} [1 - e^{-\frac{V_{dd}-V_{rest}}{V_{thr}}}] = I_{n_0}\left(\frac{W}{L}\right)_{K} [1 - e^{-\frac{V_{rest}-V_{down}}{V_{thr}}}]$$

Assuming V_{rest} lower than $0.2V$ to prevent the first inverter from triggering, we have:

$$[1 - e^{-\frac{V_{dd}-V_{rest}}{V_{thr}}}] \approx 1$$

So in the assumption of $V_{rest} \leq 0.2V$:

$$I_{p_0}\left(\frac{W}{L}\right)_{Na} = I_{n_0}\left(\frac{W}{L}\right)_{K} [1 - e^{-\frac{V_{mem}-V_{down}}{V_{th}}}]$$

$$V_{rest} = V_{th} \ln\left(-\frac{I_{p0}(\frac{W}{L})_{Na}}{I_{n0}(\frac{W}{L})_K} + 1\right) + V_{down} \tag{2.3}$$

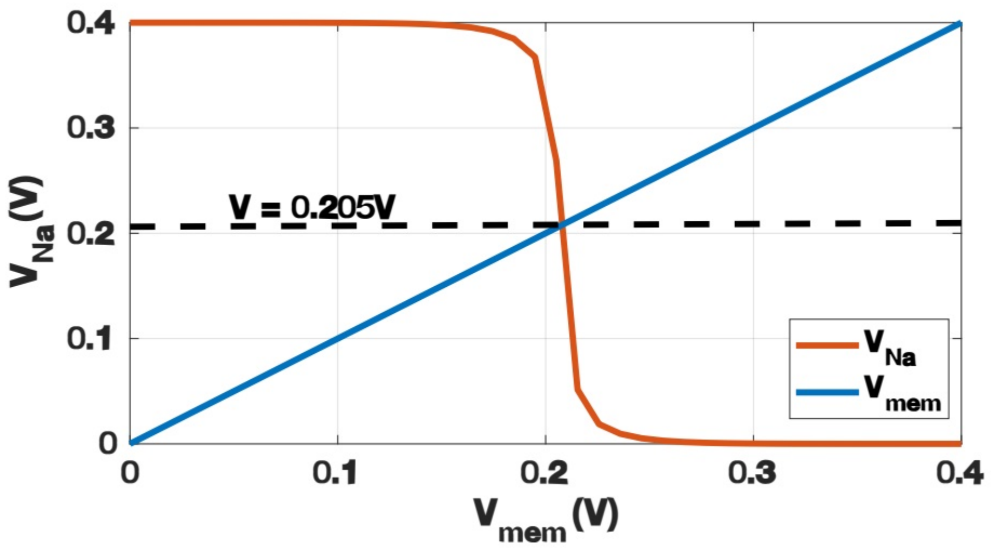


Figure 2.3: Inverter characteristic simulated on the nodes V_{mem} and V_{Na} node during a voltage sweep between $0V$ and $0.4V$. The intersection between the the lines shows $V_{thr} \approx 0.205V$. Figure from [34]

The result shows that in order to achieve a proper V_{rest}, M_k must be more conductive than M_{Na}. If this condition is met, we have $V_{rest} \approx V_{down}$ during the resting state.

Depolarization state

In case a current I_{syn} from the synapses or a current I_{in} is injected, the balance between the leakage currents of M_K and M_{Na} is broken. The extra current integrates into the capacitance C_{mem} and the voltage of V_{mem} evolves toward a new equilibrium at higher voltage. If the equilibrium is below the first inverter commutation voltage, the circuit is still in stationary condition and when no more current is injected, the V_{mem} node will gradually return to its resting state. However, if the V_{mem} value exceeds a certain V_{thr} the first inverter starts to commute leading to the decrease of V_{Na} voltage. With its gate voltage decreasing, the transistor M_{Na} will inject more current to the V_{mem} node. This will lead to a positive feedback until V_{mem} reaches the supply voltage. The threshold of the spiking neuron is the effective threshold of the first inverter. From the input-output characteristic of the inverter in Fig. 2.3, the threshold voltage is $V_{thr} \approx 0.205V$.

Repolarization state

While V_{mem} increases and V_{N_a} decreases, the second inverter charges the capacitor C_K through the transistor M_{p2}. Once this capacitor is charged enough the transistor M_K

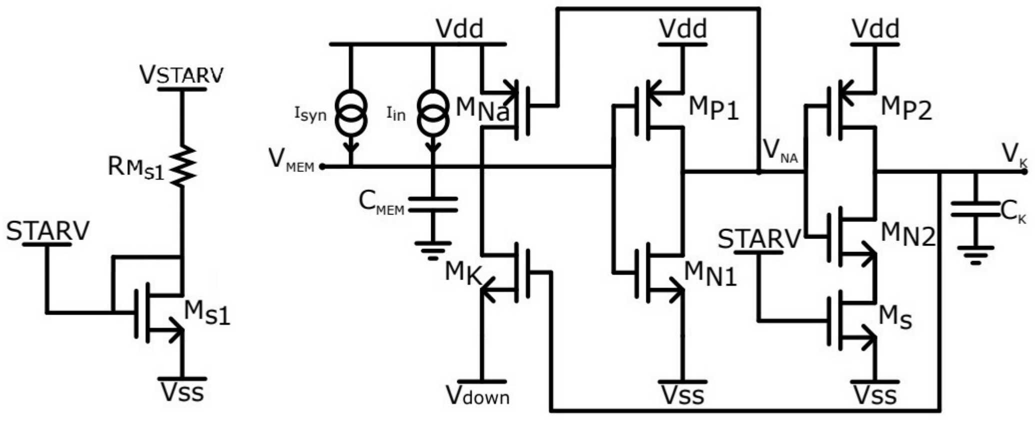


Figure 2.4: Circuital scheme of the neuron and the starver bias circuit. $R_{M_{S1}}$ is a $100k\Omega$ resistor placed in series to M_{S1}, which is a nMOS with the same size as M_S. Figure modified from [34]

starts to drain current out from C_{mem}, leading to a negative feedback. In fact, the decreasing of V_{mem} would make the first inverter commute back to $V_{Na} = V_{dd}$ and the second inverter discharge the capacitor C_k.
The time between the beginning of the charging of C_K and the start of the negative feedback determines the time width of a spike. This time interval should be as small as possible in order to save power. It also has to be slower than the time needed by V_{mem} to reach V_{dd}, otherwise, the spike cannot rise fully during its transient. Based on the sizing of M_{Na}, M_K and the first inverter, a duration of the spike of $T_{width} \approx 1ms$ was chosen.
Considering that the charging current of C_K comes from M_{P2} and that from simulations the critical voltage that activates the negative feedback is $V_{K_{ON}} \approx 0.3V$, the time width of a spike can be calculated as:

$$T_{width} = \frac{V_{K_{ON}} C_K}{I_{P2}}$$

Where:

$$I_{P2} = I_{P0} \left(\frac{W}{L}\right)_{P2} e^{\frac{V_{sg}}{n_p V_{th}}} [1 - e^{-\frac{V_{ds}}{V_{th}}}] = I_{P_{ON}} \left(\frac{W}{L}\right)_{P2} [1 - e^{-\frac{V_{ds}}{V_{th}}}]$$

With the approximation of $V_{sg} \approx V_{dd}$ and assuming $V_{ds} >> V_{th}$, T_{width} can be finally expressed as:

$$T_{width} = \frac{V_{K_{ON}} C_K}{I_{P_{ON}} (\frac{W}{L})_{P2}}$$

Hyperpolarization state

The slow discharge of C_K through the NMOS M_{N2} is limited in current by the starver M_s. The latter transient defines the time when the neuron cannot integrate the input current, called refractory period. This important parameter defines the time between spikes (Interspike Interval) generated by a neuron when the input current is constant. The refractory period also represents the maximum frequency at which a neuron can spike. The boundary is related to the working principle of the starver, which determines the current integrated in C_K and thus its charging time. The value of T_{refr} was chosen around $6ms$ and can be controlled by changing the supply voltage V_{STARV} of the starver (Fig. 2.4). The relationship between the refractory period and the starver current is:

$$T_{refr} = \frac{(V_{dd} - V_{K_{OFF}})C_K}{I_{starv}}$$

Where $V_{K_{OFF}} \approx 0.15V$ is the voltage that the node V_K should assume in order to switch off the transistor M_K.
Considering the charging time of the capacitance, when a constant current is injected, as:

$$T_{charge} = \frac{V_{dd}C_{mem}}{I_{in}}$$

The frequency at which a neuron can spike is:

$$f_{spike} = \frac{1}{T_{refr} + T_{charge} + T_{width}}$$

With a maximum frequency of

$$f_{spike_{MAX}} \approx \frac{1}{T_{refr} + T_{width}}$$

2.2.2. The Synapse

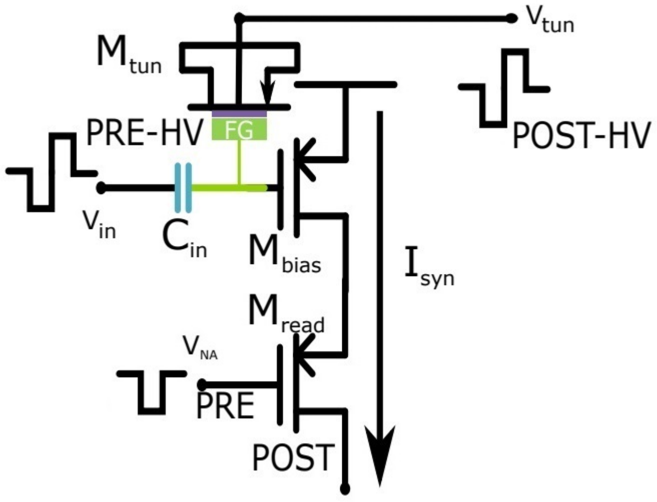


Figure 2.5: Circuital scheme of the floating gate synapse composed by three PMOS transistors. At the gate of M_{read} is applied the signal that activates the synapse. At the source-drain of M_{tun} and V_{in} a double square wave signal with same shape and amplitude are applied to permit Fowler Nordheim tunneling through the floating gate. V_{in} and V_{tun} are generated after a spike of the presynaptic neuron and of the postsynaptic neuron, respectively. Figure modified from [34]

The synapse has the role of injecting current into the capacitance C_{mem} of the neuron. It is composed by three pMOS transistors (Fig. 2.5): two of $3.3V$ technology (M_{tun} and M_{bias}) and one of $1.8V$ technology (M_{read}).

When a presynaptic neuron spikes the STDP circuit generates a double square wave at V_{in} while a negative pulse (the V_{Na} signal from the presynaptic neuron, Fig. 2.4) is applied at the gate of M_{read}. The pulse has a width of $1ms$ and it is synchronized with the negative part of the double square wave signal. The pulse turns on M_{read} generating a current square wave of similar shape as the voltage applied to M_{read} gate, but positive. The current amplitude is controlled by the amount of charge stored in M_{tun}, which is a PMOS transistor with drain and source short circuited that creates a floating gate. After that, M_{read} turns off and and the double square wave signal is applied at V_{in}. If this happens while a similar signal is applied at V_{tun}, a current is injected or ejected from the floating gate according to the STDP learning rule. PMOS transistors have been specifically employed for the floating gate to assure that each tunneling transistor would have its own bulk to prevent charge sharing and cross talk between synapses.

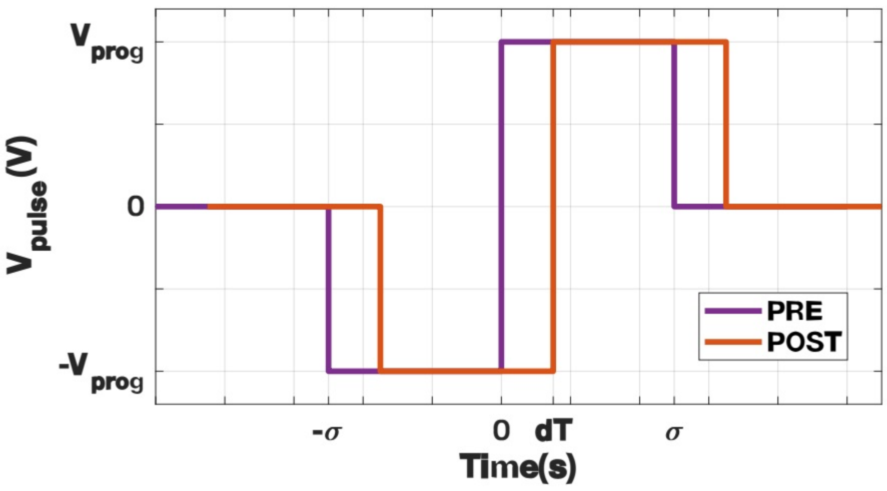


Figure 2.6: Profile of the signals $PRE - HV$ and $POST - HV$. The tunneling happens only n the dT time interval with $V_{PRE-HV} = V_{prog}$ and $V_{POST-HV} = -V_{prog}$ (or viceversa). In all the other cases the ΔV applied to the floating gate is not enough to change significantly its stored charge. Figure from [34]

The injection and ejection of electrons in the floating gate of M_{tun} is performed through tunneling. The Fowler Nordheim tunneling [32] is a phenomenon that happens when a relatively high voltage is applied between a potential barrier. The bands of the barrier starts to bend and the probability of an electron to tunnel through the barrier increases exponentially.

In this particular case the voltage barrier is represented by the oxide and can be expressed as:

$$I_{tun} = -I_0 W L e^{-\frac{V_f}{V_{ox}}} = I_{tun0} e^{-\frac{V_0}{V_{tun}}}$$

Where I_0 is a pre-exponential current, V_{ox} is the voltage across the oxide, W and L are the dimensions of the transistor, and V_f is a constant experimentally obtained that depends on the oxide thickness.

The tunneling current is induced from the bulk to the floating gate node or viceversa if a high potential difference is applied across the oxide of M_{tun}. In fact, a forward tunneling current $I_{\overleftarrow{tun}}$ flows from the gate to the bulk if the difference $V_{in} - V_{tun}$ is positive. Viceversa, a backward tunneling current $I_{\overrightarrow{tun}}$ is injected from the bulk to the V_{fg} node if $V_{in} - V_{tun}$ is negative. By applying high voltages between V_{in} and V_{tun}, the charge can be stored or removed in the floating gate, altering the V_{fg} voltage and thus changing the current I_{out}. The weight of the synapses is the charge integrated into V_{mem} of the postsynaptic neuron during a single spike.

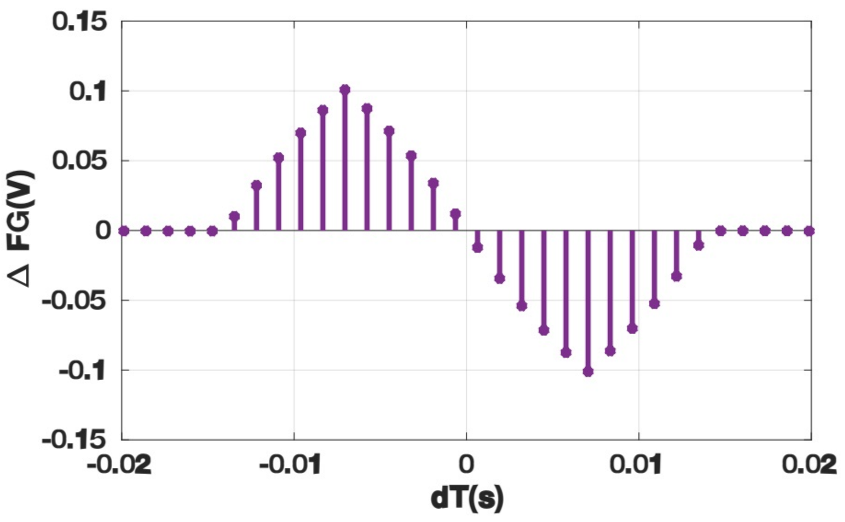


Figure 2.7: Simulated ΔV_{fg} at a function of $\Delta T = T_{PRE-HV} - T_{POST-HV}$. The doube square wave signals have $7ms$ width and $4.5V$ amplitude. Figure from [34]

If we consider both of forward and backward currents, the charge stored in the floating gate can be expressed as:

$$\Delta Q_{fg} = -C_T \Delta V_{fg} = \int_{\Delta T} (I_{\overleftarrow{tun}} - I_{\overrightarrow{tun}})dt = \int_{\Delta T} [I_{\overleftarrow{tun}_0} e^{-\frac{V_0}{V_{in}\frac{C_{in}}{C_T}+V_{fg}-V_{tun}}} - I_{\overrightarrow{tun}_0} e^{-\frac{V_0}{-V_{in}\frac{C_{in}}{C_T}-V_{fg}+V_{tun}}}]dt$$

Where ΔT it the time length of the applied difference i.e. $\Delta T = T_{PRE-HV} - T_{POST-HV}$ and C_T is the equivalent capacitance seen at the V_{fg} node.
In order to be able to inject enough charge two high voltage double square wave signals are applied at the ends of the floating gate (Fig. 2.6). The typical upper and lower value of the signals are $V_{prog} = 4.5V$ and $-V_{prog} = -4.5V$ in order to able to apply $\pm 9V$ at the oxide of M_{tun}. These signals, generated by the STDP circuit, have their own $\pm 4.5V$ voltage supplies and thus the current injected into all the floating gates can be changed just by changing those two power supplies. The simulated voltage change of the V_{fg} node as a function of the time difference between the pre and post spike is shown in Fig. 2.7.

2.2.3. The Spike-Time Dependent Plasticity circuit

In order to generate the HV double square wave signals (Fig. 2.6) a proper circuit has been designed (Fig. 2.8). It was called STDP circuit and it is composed of three main blocks: the timing circuit, the voltage shifters and the 5V MOSs.

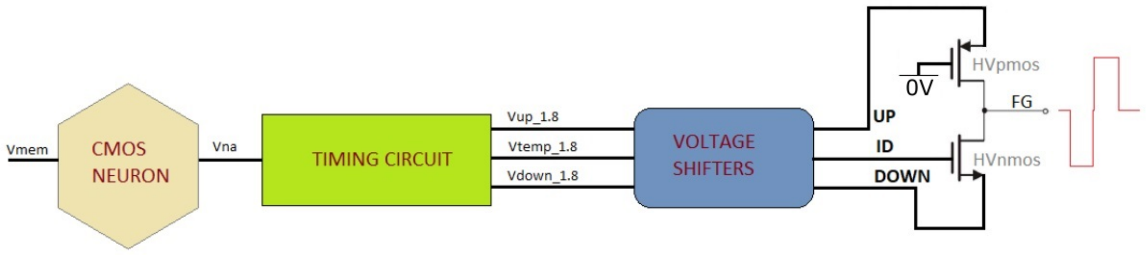


Figure 2.8: STDP circuit scheme. The timing circuit generates the shape of $Vdown\_1.8$, $Vup\_1.8$ and $Vtemp\_1.8$ signals. The voltage shifters extends the voltage range of these signals in order to be suitable to control the HV MOS and generate the HV signals. Figure from [44]

The timing circuit

The timing circuit, shown in Fig. 2.9, creates the proper timing distance to drive the 5V MOSs from the V_{Na} signal. The voltages of the signals go from $0V$-$0.4V$ supplies to $0V$-$1.8V$ supplies.
The first stage is composed by a couple of inverters in cascade that regenerates the V_{Na} signal of the neuron and reshape it in a rectangular fashion. This signal is then processed by two NOR gates and one AND gate. Furthermore, one of the AND inputs and one of the NOR inputs is slowed down by a starver inverter (Fig. 2.10) in order to control the time shape of the signals by changing their biases.

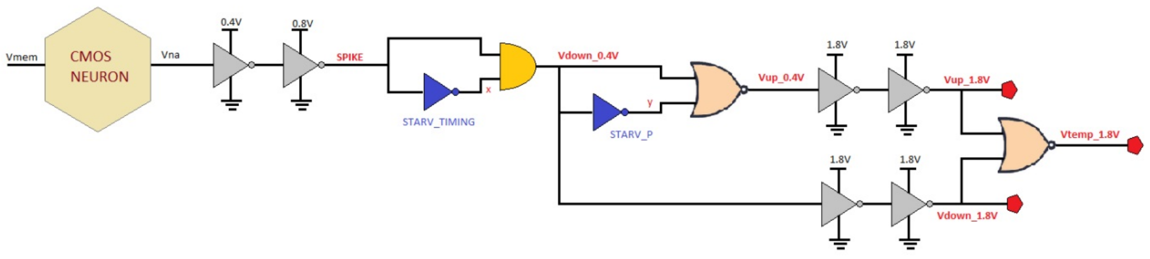


Figure 2.9: Timing circuit scheme. The first stages are two inverters in cascade which regenerate the V_{Na} and then extend it from $0V - 0.4V$ to $0V - 0.8V$. After that, some logic ports reshape the signal into $Vdown\_0.4$ and $Vup\_0.4$ signals. Finally, these signals get shifted to $0V - 1.8V$ range with two standard inverters and enter the output NOR gate. Figure from [44]

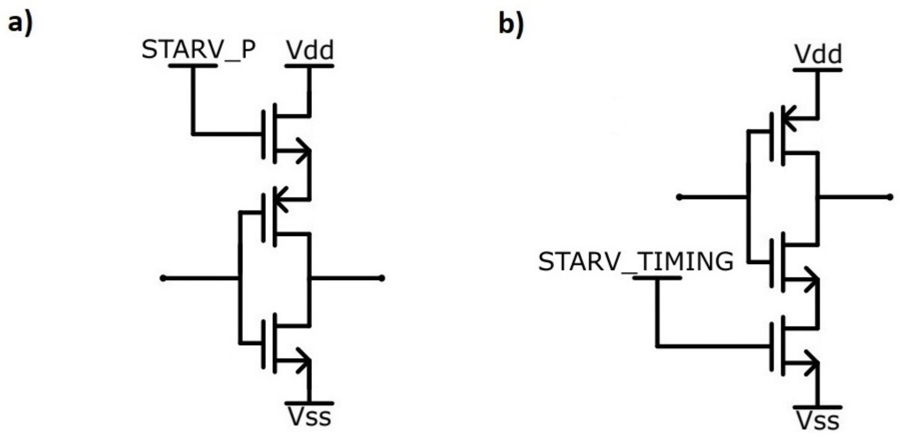


Figure 2.10: Circuital scheme of the starvers. **a**) The starver inverter before the first NOR generates a limitation in the rising edge, in order to slow it down by a time interval D2 (referred as T_{up} in the next chapter). **b**) the starver inverter before the AND gate limits the falling edge by a D1 delay (referred as T_{down} in the next chapter).

The voltage shifters

The voltage shifters (Fig. 2.11) stretch the signals coming from the timing circuit to higher voltages. Depending on the signal involved, they can extend the range up to $0V$-$4.5V$ and -$4.5V$-$0V$, maintaining the correct delays imposed by the timing circuit. Furthermore, an output path (Fig. 2.20) have been designed for the ID signals (Figure 2.11 **b**) of all the neurons. These signals share the same time lengths of the HV signals and thus can be used to check the STDP circuit functioning.

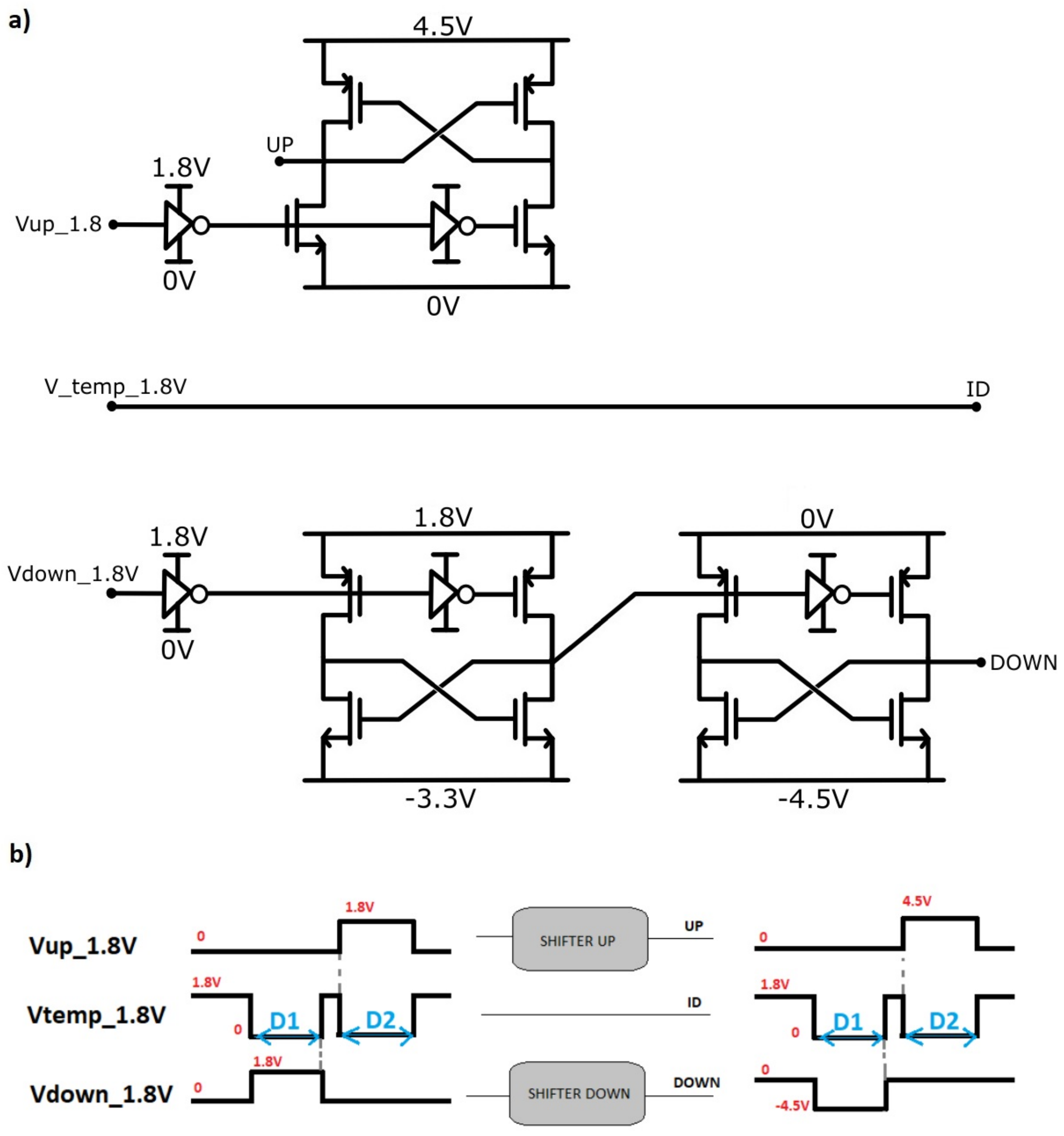


Figure 2.11: **a**) Circuit schematic of the voltage shifters **b**) Signals before and after the voltage shifters. Figure **b** from [44]

The high voltage MOSes

The output stage uses an NMOS and a PMOS to selectively connect the output node to +4.5V or to -4.5V (Fig. 2.12). The transistors are controlled by the three signals generated by the voltage shifters. By controlling the gate and sources of this circuit we can obtain the shape we found useful in 2.2.2 subsection. In LFoundry technology, the chosen nmos5v0rvt and pmos5v0rvt are able to withstand voltages up to 6V of V_{gd}, V_{gs}

and V_{ds}.

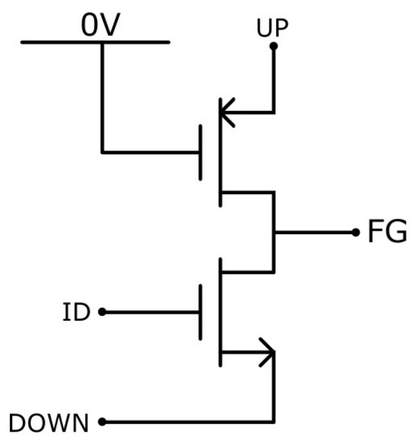


Figure 2.12: HV MOSes configuration. The UP, $DOWN$ and ID signals pilots the generation of the HV signals.

2.2.4. Neuron interface circuits

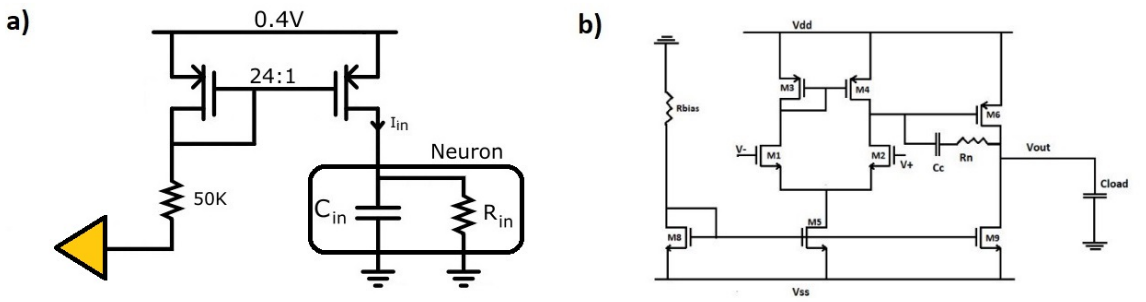


Figure 2.13: **a**) Circuit schematic of the current mirror. The trandiode is connected through a 50kΩ resistor to an external pin of the chip. **b**) Circuit schematic of the buffers. Figure **b** from [44]

In order to controll and interface with every single neuron, dedicated circuits have been designed.

The first one is a current mirror (Fig. 2.13 **a**) working in the subthreshold regime that serves as a current generator. Each neuron has its own current mirror that can be driven by an external signal generator. The mirror ratio is such that the current injected into the neuron is reduced by a factor 24 in order to increase the control of V_{mem}. To help in reducing further the current an external $50G\Omega$ resistor is placed on the PCB test board.

The second circuit integrated in the chip is a buffer designed to read the V_{mem} of the neurons (Fig. 2.13 **b**). These three buffers are fundamental to keep low the equivalent capacitance at C_{mem} node which is responsible for the integration. The buffers are composed by a CMOS OTA and a high impedance output compensated through a nulling resistor R_n and a Miller capacitance C_c. The R_{bias} is connected to an external pin, where a specific voltage is given. Furthermore, the voltage supplies of the buffers have been decided starting from the voltages already needed by the chip: $V_{dd} = 1.8V$ and $V_{ss} = -3.3V$.

2.3. PCB layout

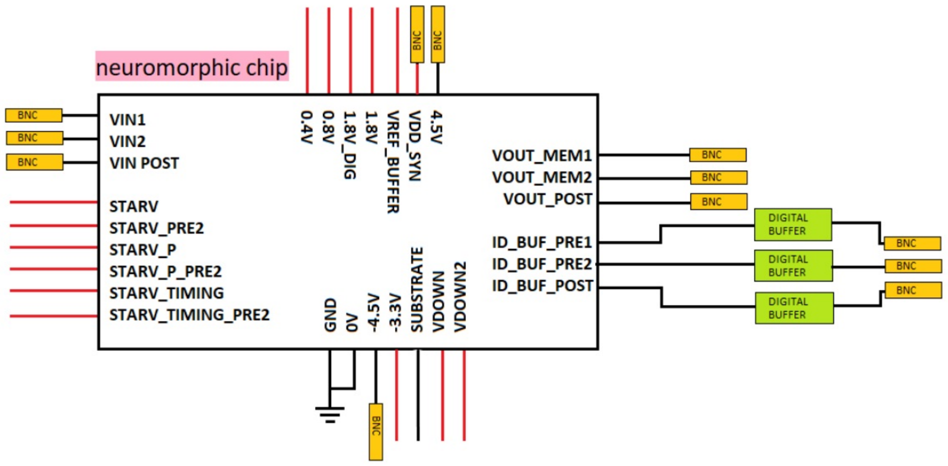


Figure 2.14: Block scheme of the neuromorphic chip pins and connections. The connections of the chip are used to provide power supplies, input signals and output signals. Figure modified from [44]

In order to interface with the chip, a test PCB have been developed. The design has been done using Altium program and then fabricated by Millenium Dataware. The PCB test board is designed to:

- **give the proper input signals to the chip**. $VIN1$, $VIN2$, VIN_POST are externally imposed voltages that bias the current mirrors and provide the current to the CMOS neurons;
- **provide the proper power supplies**. Pos_Ref and Neg_Ref are the main power supplies of the board that provide the right bias of all the components. $+4.5V$ and $-4.5V$ are the power supplies that directly bias the HV MOSs in the STDP circuit. A dedicated bias was chosen in order to directly affect the tunneling and the STDP characteristic;

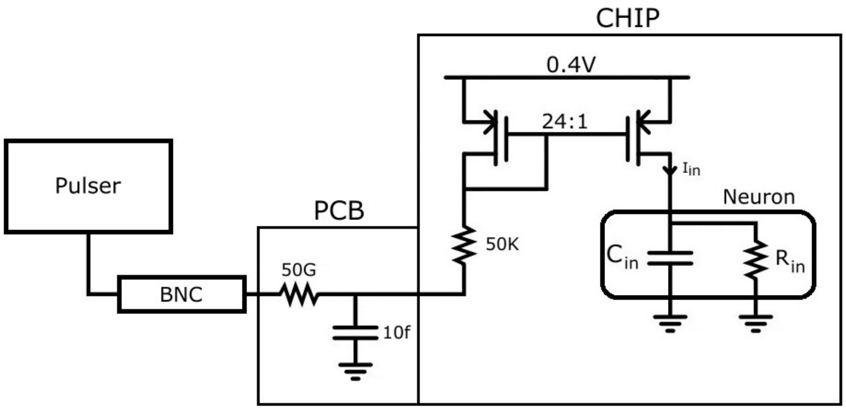


Figure 2.15: Circuit schematic of the path that leads from the voltage applied by the pulser to the generation of the current injected into the neuron.

- **read the output signals**. $VOUT\_MEM1$, $VOUT\_MEM2$ and $VOUT\_POST$ are the output voltage signals of the three output buffers. They are needed to show the membrane voltage of the neurons. ID_BUF_PRE1, ID_BUF_PRE2 and ID_BUF_POST show the ID signals of the timing circuits. The possibility of measuring these signals will be beneficial to verify the correct functioning of the timing circuit.

2.3.1. Input path

In order to charge externally the capacitance C_{mem} of each neuron the chip includes three current generators. However, the needed current to cope with the time requirements of the neurons is in the order of pA. To have such a small currents, the BNCs designated to provide the input signals ($VIN1$, $VIN2$, VIN_POST) have been placed in series with a $50G\Omega$ resistor and then to the current mirror described in the previous section (Fig. 2.15). A $10pF$ capacitance has been also added to filter out unwanted high frequency signals coming from the external voltage generators. The resulting RC low pass filter has a fixed pole at $0.3Hz$ which increases in frequency if a current is injected into the mirror. In fact, the current would decrease the equivalent resistance of the transdiode and change the pole.

In case of DC current, the resulting current injected into the neuron would be:

$$I_{in} = \frac{0.4V - V_{ds} - V_{in}}{24 \cdot 50G\Omega}$$

Where V_{ds} of the transdiode is $\approx 200mV$

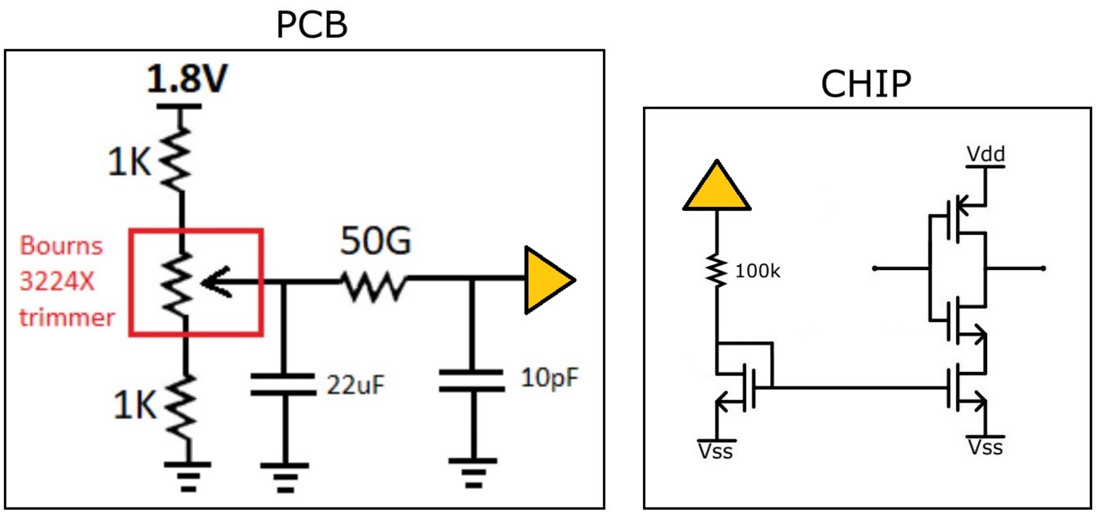


Figure 2.16: Circuit schematic of the path that leads from the trimmer to the bias of the starvers. The circuit comprises a resistive partition with $1k\Omega$ resistors and a Bourns 3224X trimmer, a $50G\Omega$ resistor to reduce the current, a $10pF$ decoupling capacitor close to the pad of the neuromorphic chip and a $22\mu F$ capacitor for an RC LPF. The adopted trimmers are featured by a maximum of $10k\Omega$ resistance. Figure modified from [44]

2.3.2. The starvers regulators

The starver configuration has been used widely during the design of the chip. In fact, by changing its bias it is possible to slow down or speed up an inverter commutation. To have the possibility of changing the transient time of the starver inverters, their bias has been implemented tunable through trimmers. The trimmers can generate a reference current by tuning the voltage drop across a $50G\Omega$ resistor from $0.15V$ to $1.65V$. The current is then injected into a current mirror that biases the starvers (Fig. 2.16) with a range from $3pA$ to $33pA$.

There are 6 trimmers that control the starvers: $STARV$, $STARVP$, $STARV\_TIMING$, $STARV\_PRE2$, $STARVP\_PRE2$, $STARV\_TIMING\_PRE2$ (Fig. 2.17). The first three biases the presynaptic neuron 1 and the postsynaptic neuron, while the latter three are dedicated to bias the presynaptic neuron 2. This choice allowed separate measurements and timings during the testing phase. $STARV$ and $STARV\_PRE2$ determines the refractory periods. $STARVP$ and $STARVP\_PRE2$ control the time length of the positive voltage of the HV signals. Finally, $STARV\_TIMING$ and $STARV\_TIMING\_PRE2$ determines the time length of the negative voltage of the HV signals (Fig. 2.17).

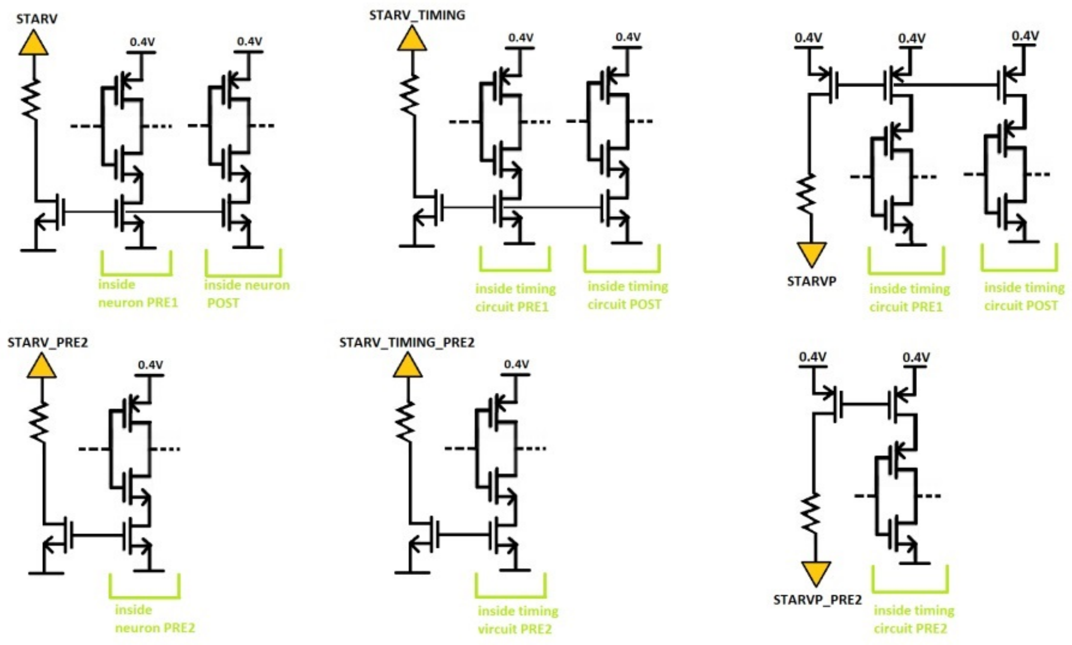


Figure 2.17: Circuital scheme showing the starvers that share the same current mirrors. Figure from [44]

2.3.3. The power supplies

The power supplies needed by the neuromorphic chip are eleven in total and can be categorized into three different kinds:

- **Tunable power supply**. $0.8V$, $1.8V$, $1.8V\_dig$, $0.4V$, $Vdown$, $Vdown2$, $-3.3V$ and $Vref\_buffer$ are power supplies that needed to be tunable, therefore a trimmer performing a resistive voltage divider between 0 and 1.8V has been involved for each of them. Furthermore, a 22µF capacitor is set in a RC low pass filter configuration followed by a buffer and a 50Ω resistance at its output directly connected to the chip (Fig. 2.18 **a**).
- **Fixed power supply**. The BCN of -4.5V and 4.5V power supplies are just connected to the pin of the chip through a low pass filter composed by a 500Ω resistance and a 22μF capacitor.
- **Hybrid**. The voltage supply of the synapses VDD_SYN has a nominal value of 0.4V, but the possibility of tuning it has been taken into account if necessary during the test phase. For this purpose, a trimmer has been inserted to change the voltage if needed, otherwise a simple connection to the external generator (properly filtered) imposes the required voltage (Fig. 2.18 **b**).

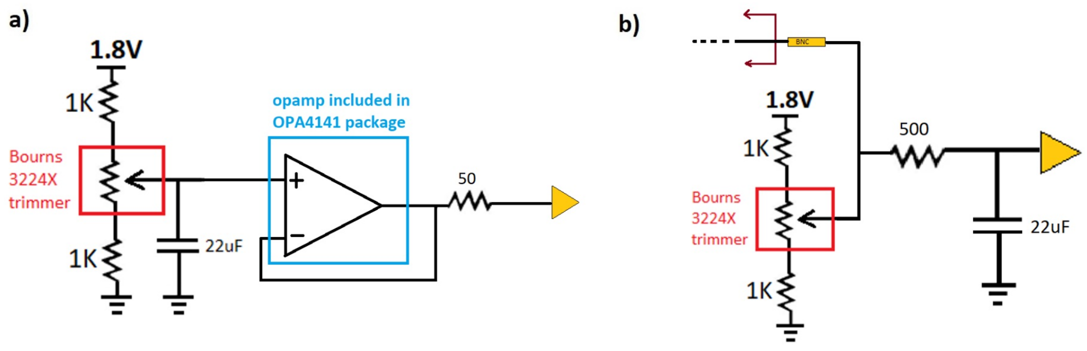


Figure 2.18: Circuital scheme of the power supplies chain from the external voltage generator to the pin. **a**) The tunable approach comprises a trimmer Bourns 3224X, $1K\Omega$ resistors, a $22\mu F$ capacitor, one of the opamp included in OPA4141AID package, a 50Ω resistor. **b**) Hybrid approach to generate the synapse supply voltage. Figures from [44]

As mentioned before, the BNC connectors of Pos_Ref and Neg_Ref bring the external reference voltages to the board. Then they are linked to four LDO regulators which generate all the lower voltage required. From the power supplies four supply voltages have been generated (Fig. 2.19 **a**):

- $1.8V$. Generated from the positive power supply, this voltage is the positive reference voltage of all the trimmers in the PCB that also biases the 1.8V pin of the chip. The positive regulator used is the LM1117MP1.8 [53] with a $22\mu F$ input and $10\mu F$ output capacitances (Fig. 2.19 **b**).

- $1.8V\_DIG$. This voltage is the positive reference voltage of the digital buffer and the digital $1.8V$ pin of the chip. The positive regulator used is the same as for the $1.8V$ supply: LM1117MP1.8 LDO with a $22\mu F$ input and $10\mu F$ output capacitances (Fig. 2.19 **b**).

- $5V$. This voltage, generated from the positive power supply, biases the OPAMPs. The positive regulator used is the MC78M05ABDT [40] with a $3.3\mu F$ input capacitance (Fig. 2.19 **c**).

- **–3.3V**. Generated from the negative power supply, it is used as negative power supply of all the buffers mounted on the PCB, negative reference of some of the trimmers on the board and the –3.3V power supply of the neuromorphic chip. The negative regulator used is the UCC384DP-ADJG4 placed in the same configuration as Fig. 2.19 **d**. It has been chosen $R1 = 200k\Omega$ and $R2 = 120k\Omega$ in order to satisfy the relationship $V_{out} = -1.25V(1 + \frac{R1}{R2})$. The capacitor $C1 = 20pF$ is placed to cancel the pole introduced by $R1$ and the parasitic capacitance at $VOUT$ node.

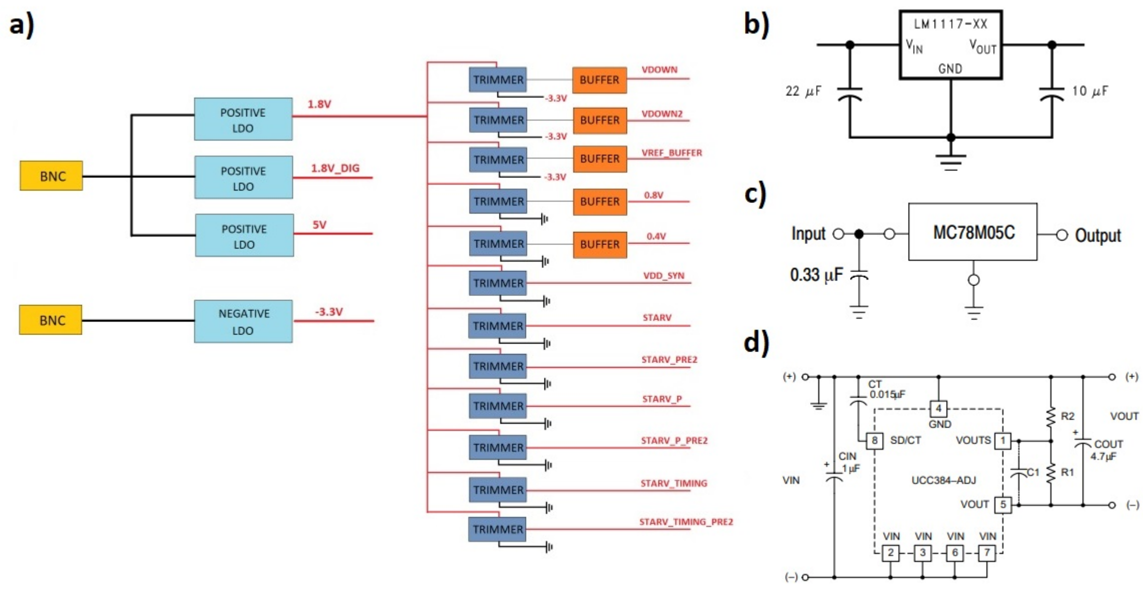


Figure 2.19: **a**) Schematic blocks of hierarchical organization of the power supplies. The ramification reduces the number of external power supplies needed. Figure from [44]. **b**) Circuit schematic of the positive LDO of $1.8V$ and $1.8V\_DIG$ power supplies. Figure modified from [53] **c**) Circuit schematic of the positive LDO of $5V$ power supply. Figure modified from [40] **d**). Circuit schematic of the negative LDO of $-3.3V$ power supply. Figure from [54]

2.3.4. Output path

Two kinds of signals have been chosen as output of the chip in order to test it: the V_{mem} and the ID signals. The first one is needed to check the correct functioning of the neuron, while the second is needed to have proper information on the timing circuit functioning and thus to check on the HV signals and the STDP.

$VOUT\_MEM1$, $VOUT\_MEM2$ and $VOUT\_POST$ can be monitored through the output buffers already implemented on the neuromorphic chip, therefore a simple connection from the output pins to the BNC connectors is done.

For what concern ID_BUF_PRE1, ID_BUF_PRE2 and ID_BUF_POST, which are double-waves rectangular voltage spanning from $0V$ to $1.8V$, two CMOS digital buffers on chip provided by LFoundry and another digital buffer as IC on the PCB have been chosen to boost the signal (Fig. 2.20). The buffer mounted on the PCB is the SN74LVC3G17DCTRE4 [55], which performs the Boolean function Y = A, following a non-inverting Schmitt trigger logic. The power supplies determine the high and low logic levels and are taken accordingly to transmit the digital ID signal: 0V and 1.8V.

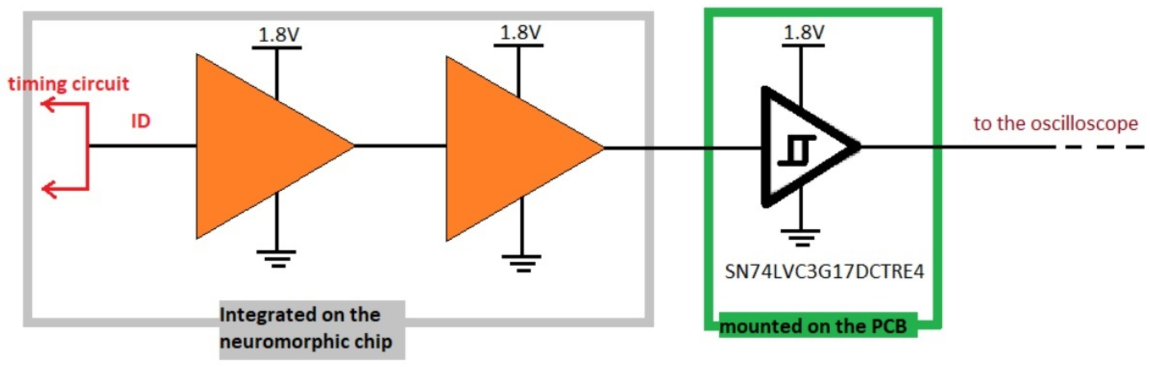


Figure 2.20: Schematic blocks of the output path for the ID signals. Two LFoundry digital buffers are integrated at the ID node, followed by another digital buffer put in cascade on the PCB. The oscilloscope then reads the output signal through a BNC connector. Figure from [44]

3 | Experimental results

3.1. Introduction

The neuromorphic chip described in the previous chapter has been fabricated and mounted on its PCB for testing. Different experiments have been carried through to validate the performance of this implementation with its relative advantages and problems.

3.2. Measurement setup

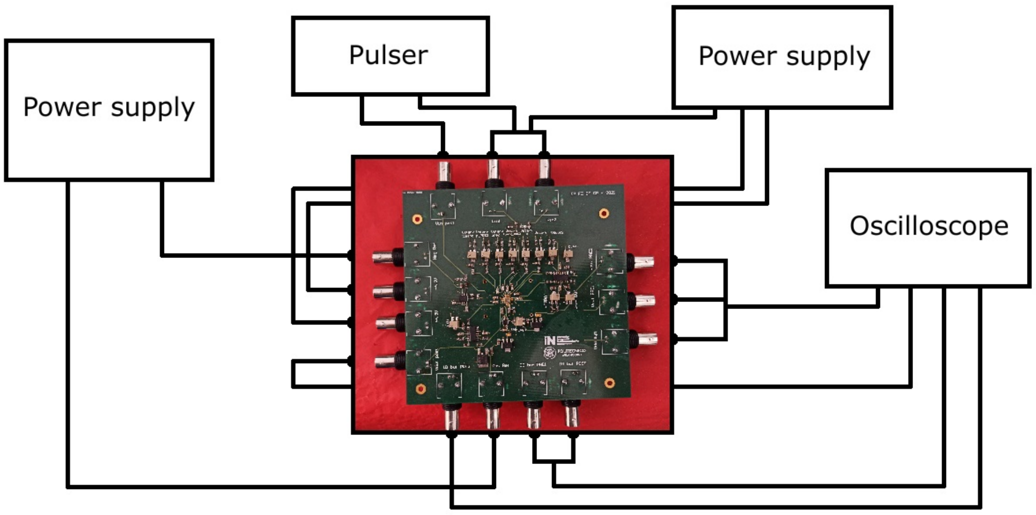


Figure 3.1: Configuration of the PCB connections used to test the chip.

The PCB used for testing the chip was built with a total of twelve BNC: three for input signals, five for power supplies and six for output signals.
The pulser used to inject current into the neurons is the 33522A series Waveform generator by Agilent. It only has two outputs and the experiments have been done only with a maximum of two neurons at the same time. The most common configuration was a presynaptic neuron measured with the postsynaptic one, while the input of the other

presynaptic neuron was biased in order to do not inject any current.
To power up the board the ROHDE & SCHWARZ DC power supply was used. The BNC inputs involved were the POS_REF and NEG_REF.
To bias the dedicated power supply of the STDP circuits and the input of one of the presynaptic neurons the HF2LI Lock-in Amplifier by Zurich Instruments was used. Its AUX outputs have been put in manual mode in order to be easily controlled by a console.
Finally, the HDO6054A-MS oscilloscope by TELEDYNE LECROY was used to record the measurements.

3.3. Chip damaging factors

Before entering in the details of the measurements, it is important to highlight some problems that occurred during the process. In fact, the chip was more fragile than expected and sensitive to low voltage differences. As a consequence the chip has been changed up to six times during the measurement process.

The first problem encountered resulted in having a static $V_{mem} \approx 100mV$ in the postsynaptic neuron that was higher than the applied $V_{down} \approx 0V$. The solution found to lower the V_{mem} was to decrease $V_{down} \approx -75mV$ and thus increase the leakage current flowing through the transistor M_K. This approach caused the decrease of $V_{mem} \approx 0V$, but did not answer the question of how the V_{mem} stationary value could be so much higher than V_{down}. The cause of the extra leakage injected in the V_{mem} node was found when experiments on the synapse were performed. In fact, it did not matter how much charge was stored into the floating gate of any of the synapses through STDP, they were not able to inject any additional current to the V_{mem} node. The formulated hypothesis was that, since a $V_{syn} \approx 0V$ was forced during the starting of the chip, when the first postsynaptic neuron spiked a voltage difference of $0.4V$ was applied between V_{mem} and V_{syn}. This voltage difference caused the M_{bias} and M_{read} pMOSes junctions of both synapses to go in direct mode. This would have caused a current relative low, but apparently high enough to damage the transistors. To validate this hypothesis different voltages were applied to the V_{syn} node and it was observed that the V_{mem} voltage would increase or decrease proportionally. Furthermore, after the bonding of the other chips on the PCB, the V_{syn} voltage forced was at $\approx 200mV$ instead of $\approx 0V$, always resulting in the correct functioning of the synapses.

The second problem encountered during the measurements involved, again, some of the V_{mem} voltages, which were stuck around $0V$. This time, there was not a correlation between the neurons as in the previous case, because it happened once to the presynaptic

1 neuron and the next time to both the postsynaptic neuron and the presynaptic neuron 2. The measurements showed also a signal superimposed at the $\approx 0V$ bias. From the undulating shape of the signal and its frequency of $\approx 50Hz$ was deduced that the node must have been in a floating state. However, the ID signals seemed to work fine. When a current was injected into the neuron they would commute just like they would have done if the neuron was spiking. This would have been impossible if V_{mem} was floating and thus it was further deducted that the cause of the faulty signal could only have been the breaking of the analog buffer on the chip. In fact, the output nodes of the OPAMPs are directly connected with the BNCs, which in those occasions have been re-soldered. To validate this hypothesis other soldering tests have been performed and the resulting behaviour of the amplifier outputs was similar: floating at some voltage, with a $50Hz$ sinusoid superimposed. Probably, the soldering caused some electrostatic shock that was enough to pierce through the oxide of the Miller capacitance, causing the V_{out} node to remain floating.

The third problem encountered during the measurement involved the function of the presynaptic neuron 2. During the measurement of the last two chips the V_{mem} node was stuck at $0.4V$ forcing its synapse to remain opened. This fact kept the postsynaptic neuron spiking at the higher possible frequency that its refractory period permitted. In the attempt to recover at least the postsynaptic neuron, the V_{down} of the presynaptic neuron 2 was lowered up to $-0.5V$ in order to fully open the M_K transistor. The response was that V_{mem} would come back to its normal resting state where $V_{mem} \approx V_{down}$. An increase of V_{down} would have been followed by V_{mem} up to $\approx 0.2V$, which is the spiking threshold. However if the positive feedback was started the V_{mem} would get stuck again at $0.4V$. This time the hypothesis was that during the bonding or the starting of the PCB the M_{S1} nMOS broke causing the shutting down of M_S and thus impeding the negative feedback to bring back V_{mem} to $\approx V_{down}$.

3.4. Two neuron comparison

As stated in the previous chapter, the presynaptic neuron 1 and the postsynaptic neuron shares the same power supplies and starvers' current mirrors. Those neurons, excluding the synaptic connection, are nominally the same and thus perfect to be compared in order to understand the limits of this technology, dependent on the process variability.

PCB voltage configuration

	Refractory period	LHV signal	HHV signal	STDP
VSTARV_TIMING	250 mV	VAR	250 mV	250 mV
VSTARVP_PRE2	550 mV	550 mV	550 mV	550 mV
STARV_PRE2	0 V	0 V	0 V	0 V
VSTARV_TIMING _PRE2	900 mV	900 mV	900 V	900 mV
VSTARVP	630 mV	630 mV	VAR	630 mV
VSTARV	VAR	300 mV	300 mV	300 mV
VDOWN2	-20 mV	-20 mV	-20 mV	-20 mV
0.4V	400 mV	400 mV	400 mV	400 mV
VDOWN	-20 mV	-20 mV	-20 mV	-20 mV
VREF_BUF	1 V	1 V	1 V	1 V
VDD_SYN	200 mV	200 mV	200 mV	200 mV
0.8V	800 mV	800 mV	800 mV	800 mV
+4.5V	0 V	0 V	0 V	4.5 V
-4.5V	0 V	0 V	0 V	-4.5 V
POS_REF	7.3 V	7.3 V	7.3 V	7.3 V
NEG_REF	-4.2 V	-4.2 V	-4.2 V	-4.2 V

Table 3.1: PCB configurations used to perform measurements. The first three measurement focuses on the variations between the presynaptic neuron 1 and the postsynaptic neuron, thus each neuron has been individually measured. During these measurements $VSTARV$, $STARV\_TIMING$ and $STARVP$ have been respectively varied, while all the other voltages have been kept the same. The fourth measurement was needed to understand the effective characteristic of the STDP with two neurons working simultaneously.

3.4.1. Refractory period dynamic

The refractory period is a fundamental feature of a spiking neuron. It defines the maximum frequency that a neuron can spike and its maximum speed of operation. However, not only the characteristic of the refractory period of a single neuron is important. It is fundamental to understand the effect of the process variability on the refractory period of all the neurons of a possible network. It is also important to understand if the frequency

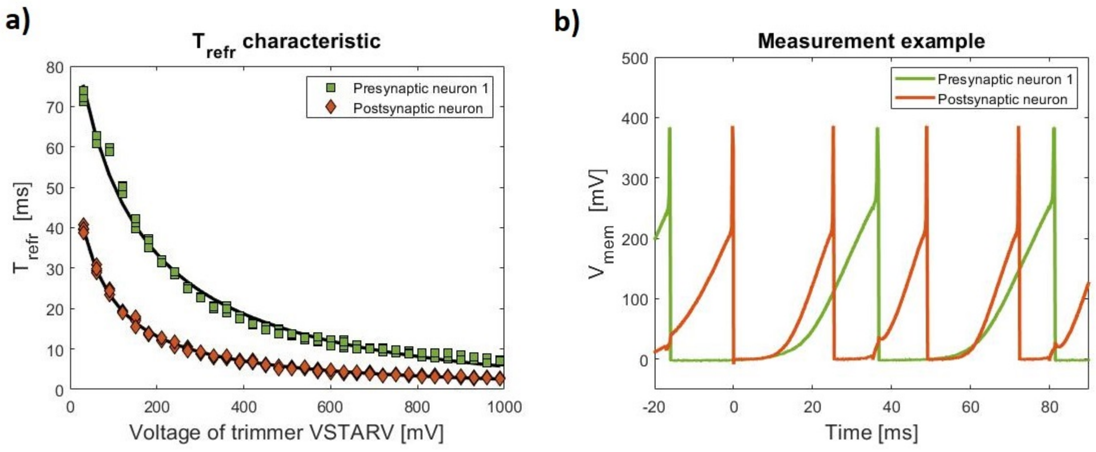


Figure 3.2: **a**) T_{refr} characteristic of presynaptic neuron 1 and the postsynaptic neuron as compared to the variation of $VSTARV$ voltage. The voltage range is from $30mV$ to $990mV$ with 4 measurements performed after each $30mV$. Black lines are the interpolated curves. **b**) Measurement example with $VSTARV$ voltage equal to $300mV$.

range is wide enough to perform training based on rate or spiking probability and to understand what is the operational speed that should be imposed to the network to prevent unexpected spiking.

To understand these limitations a measurement of the V_{mem} of the presynaptic neuron 1 and the postsynaptic neuron have been performed through the outputs of the board $VOUT\_MEM1$ and $VOUT\_POST$. The PCB configuration is listed in table 3.1. At the input of $Vin2$ was applied a constant voltage of $400mV$ in order to prevent any current injection, while a constant bias of $-1.5V$ was applied at $Vin1$ and $Vin\ post$. The refractory period (T_{refr}) has been considered the time difference between the moments when, after a spike, V_{mem} drops lower than $4mV$ from its resting state and when it rises to $4mV$ again (Fig. 3.2 **b**). $4mV$ has been chosen because it is 1% of the spike peak ($400mV$).

The acquired data (Fig. 3.2 **a**) show a significant variability of the refractory periods. In fact, the refractory period of the presynaptic neuron 1 is almost the double of the refractory period of the postsynaptic neuron, no matter the bias. The cause could be the high variability of the transistor M_s when it is biased in subthreshold regime. In fact, both the M_S transistor of the presynaptic neuron 1 and the postsynaptic neuron are biased by the same M_{S1}. Because of this variability it has been necessary to implement an algorithm resilient to the variation of the refractory period.

3.4.2. HV signals characteristic

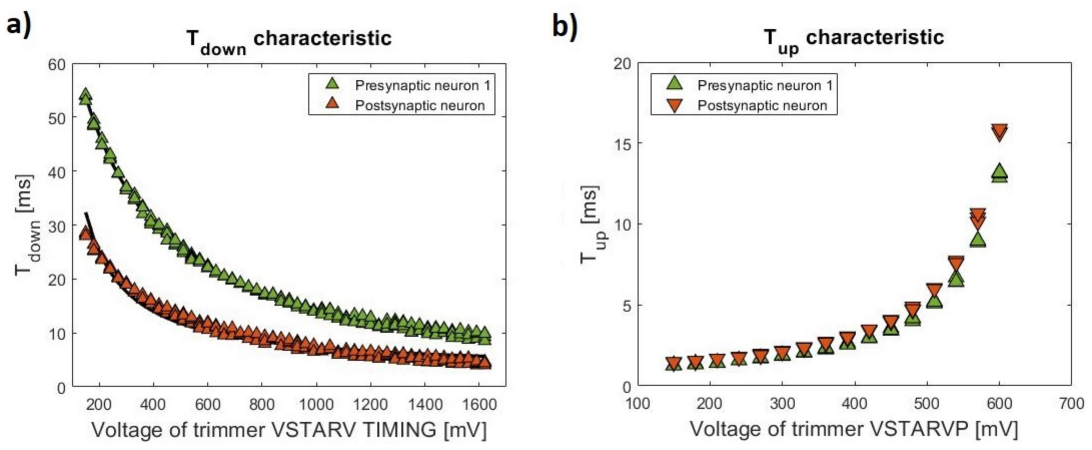


Figure 3.3: **a**) T_{down} characteristic of presynaptic neuron 1 and postsynaptic neuron as compared to the variation of $VSTARV\_TIMING$ voltage. The voltage range is from $30mV$ to $1.62V$ with 4 measurements performed after each $30mV$. Black lines are the interpolated curves. **b**) T_{up} characteristic of presynaptic neuron 1 and the postsynaptic neuron as compared to the variation of $VSTARVP$ voltage. The voltage range is from $30mV$ to $600mV$ with 4 measurements performed after each $30mV$.

$STARV\_TIMING$ and $STARVP$ trimmers determines the time shape of the HV double square wave signals that inject and eject charge into the floating gate of the synapses. These voltages bias the starvers which control the time lengths of the negative and positive voltages of the HV signals. The real relationship between the trimmers voltage and these time length is needed to understand the best possible bias voltage for an efficient STDP and its limits.

Unfortunately, there is not a direct access to the HV signals, but instead an output path to the main signals that pilot the 5V MOSs have been provided (Fig. 2.20). The signals in question are the ID signals (Fig. 2.12), which can be sensed from the BNC outputs $ID\ buf\ PRE1$ and $ID\ buf\ POST$. The PCB configuration during the measurement is listed in table 3.1 (HHV and LHV columns). At the $Vin2$ input was applied a constant voltage of $400mV$ in order to prevent any current injection, while voltage pulses of $-1.5V$ with $400mV$ baseline and $\approx 40ms$ duration were applied at $Vin1$ and $Vin\ post$. The stationary state of an ID signal is at $1.8V$ until a spike occurs. When this happens it commutes to $0V$ for a time T_{low} that corresponds to the time length of the negative voltage of its respective HV signal. Then it commutes back to $1.8V$ for $\approx 150ns$ and after that it commutes again to $0V$ for a period of time T_{up} that corresponds to the time length of

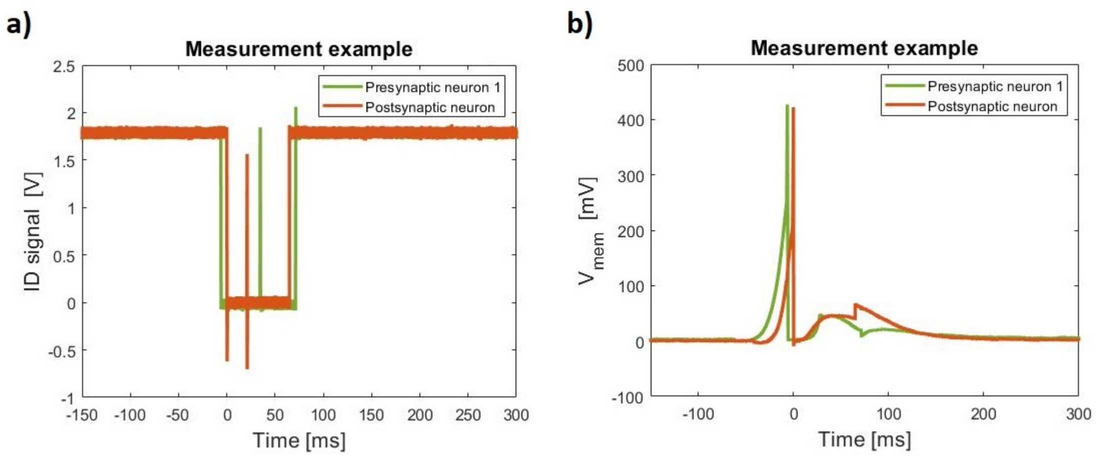


Figure 3.4: **a**) $ID\ buf\ PRE1$ and $ID\ buf\ POST$ signals when and $STARVP = 650mV$. **b**) $Vout\ PRE1$ and $Vout\ post$ signals when and $STARVP = 650mV$. In both the measurements $STARV\_TIMING = 250mV$.

the positive voltage of its respective HV signal. Finally, it returns to its resting state of $1.8V$ (Fig. 3.4 **a**).

The acquired data show a huge difference between the T_{down} characteristics of the two neurons (Fig. 3.3 **a**), while T_{up} has less variability. The T_{down} of the presynaptic neuron 1 is almost the double of the T_{down} of the postsynaptic neuron, no matter the bias. This result resembles the measurements of the refractory period. However, T_{down} measurements show much higher jitters, which increases proportionally to the applied voltage. For what concern the T_{up} measurements it is clear that PMOS starvers have much less variability. Furthermore, Figure 3.5 **c** suggests the presence of a telegraphic noise, which produces two T_{down} levels.

Moreover, to better understand the jitters of T_{down} and T_{up} more than 1000 measurements have been performed on the ID signals with the bias chosen for the STDP characteristic measurement (section 3.5). From this measurements the effective probability distribution of these time length have been extracted and reported (Fig. 3.5). The analysis shows that the variance of the time lengths is $\sigma \approx 1ms$, with some outliers of $4ms$ and $2.5ms$ in the T_{up} distributions. The effect of these jitters will appear in the STDP characteristic, however the variance is not high enough to compromise the effectiveness of the learning.

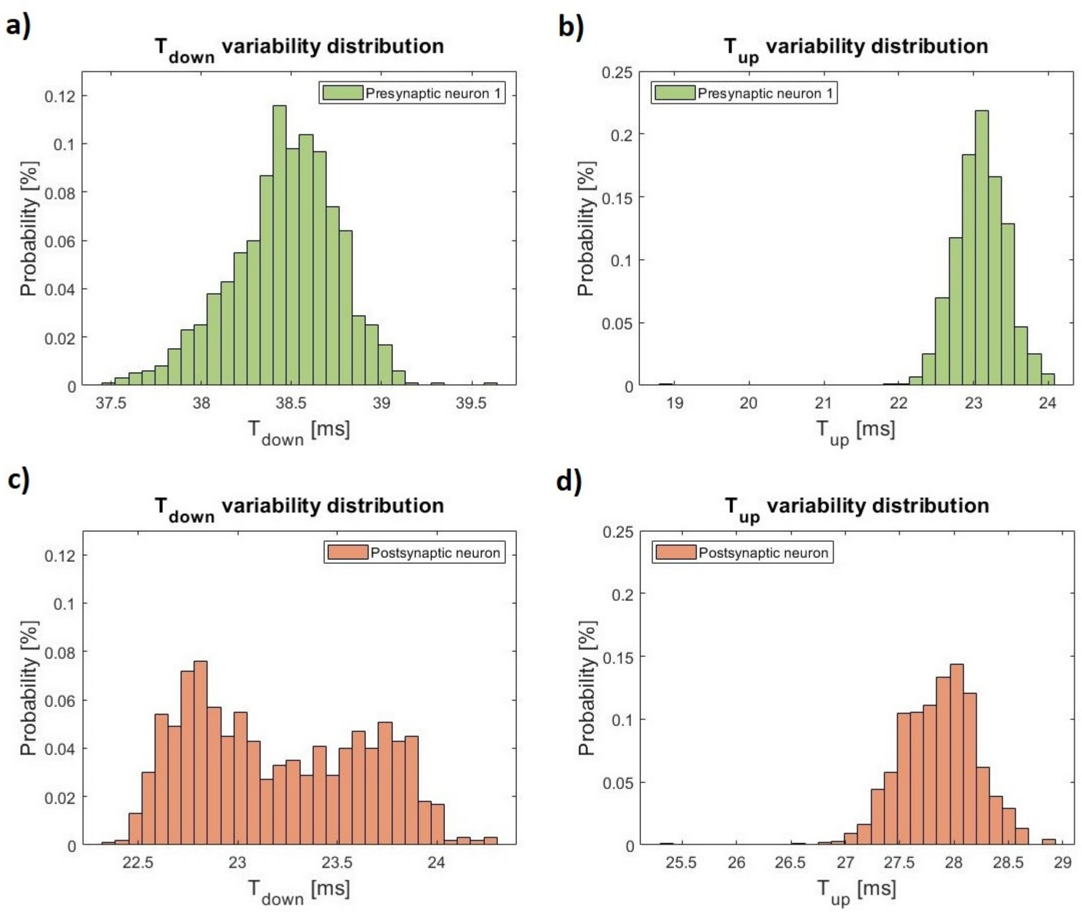


Figure 3.5: **a**) Probability distribution of T_{down} of the presynaptic neuron 1. **b**) Probability distribution of T_{up} of the presynaptic neuron 1. **c**) Probability distribution of T_{down} of the postsynaptic neuron. **d**) Probability distribution of T_{up} of the postsynaptic neuron. All the measurements have been performed with $STARV\_TIMING = 250$ and $STARVP = 630$.

3.5. STDP characteristic

The previously chosen method to train the network is the pair based Spike Time Dependence Plasticity. This training method belongs to the unsupervised paradigm and can perform competitive learning and a Winner-Take-All approach. Furthermore, during the thesis work of Mastella [34] the simulations on the neurons and synapses showed clearly the ability of this technology to develop competitive learning.

To understand if it was possible to implement a bigger network with this kind of training, measurements to trace the effective STDP characteristic have been performed. The PCB has been configured as it is listed in table 3.1 (STDP column) and at the $Vin2$ input was applied a constant voltage of $400mV$ in order to prevent any current injection. The

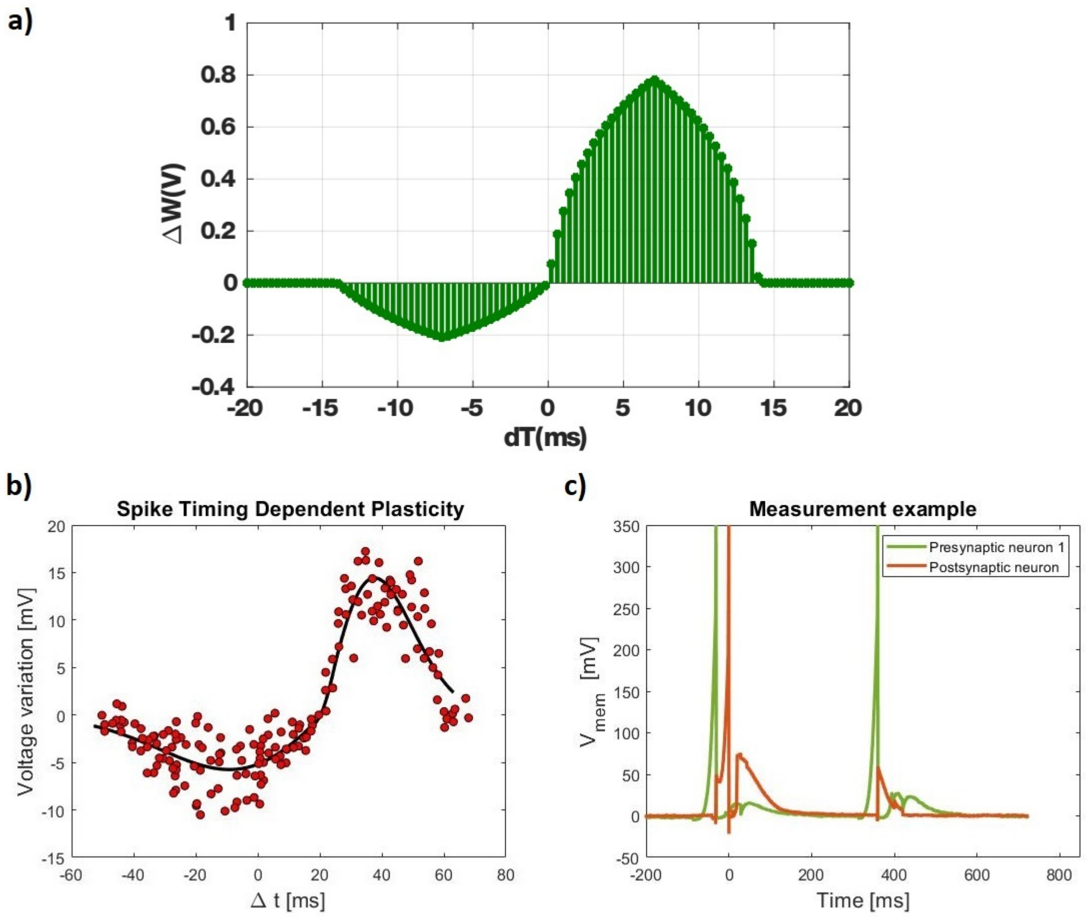


Figure 3.6: **a**) Expected STDP simulated in [34]. ΔW is the floating gate voltage. **b**) STDP characteristic acquired from measurements. The Δt range is from $-50ms$ to $68ms$ with 3 measurements performed after each $2ms$. The black line represents the interpolated curve. **c**) Measurement example with $\Delta t = 10ms$ (effective spike delay).

$STARV\_TIMING$ and $STARVP$ voltages have been specifically chosen to have T_{down} of the postsynaptic neuron and T_{up} of the presynaptic neuron 1 equal to $\approx 20ms$ to ensure biological plausibility. To measure the effectiveness of the STDP, a voltage pulse of $-1.5V$ with $400mV$ baseline and $\approx 40ms$ duration was applied at $Vin\ post$, while at $Vin1$ were applied two of this pulses. The first impulse fed to $Vin2$ was synchronized with different delays ($1ns$ precision) with the one fed to $Vin\ post$. The second pulse was used to trigger the synapse linking the presynaptic neuron 1 and the postsynaptic neuron to measure its weight change. The initial strength of the synaptic weight was set to $\approx 50mV$ (i.e. the voltage increase of V_{mem} of the postsynaptic neuron when the presynaptic neuron 1 spikes) and its value was restored every time a new measurement was performed. Both $Vout\ Pre1$ and $Vout\ post$ were measured 3 times for each different delay (Fig 3.6 **c**).

The resulting STDP characteristic (Fig 3.6 **b**), a part from the high jitters probably caused by the ID signals jitters, resemble the previously simulated characteristic. This measure have effectively validated the correct functioning of all the blocks of the chip, since all necessary to implement the STDP. However, respect to the simulated one (Fig. 3.6 **a**) the STDP characteristic is shifted in time of $\approx 20ms$. This effect, probably due to the high variability of T_{down} and T_{up}, must not be underestimated, since it makes the algorithm treat causal spike pairs, up to $\Delta t \approx 20ms$, as anti-causal. With this characteristic it would be impossible to perform competitive learning. In fact, the pair STDP learning (subsection 1.6.1) performs a WTA learning that increases the synaptic weights of the presynaptic neurons that contribute most at the spiking of the postsynaptic neurons (i.e. causal relationship). The measured STDP would address as anti-causal the presynaptic neurons that contributes most at the spiking of the postsynaptic neurons ($0ms < \Delta T < 20ms$) and thus would penalize those synapses and support more the synapses linking presynaptic neurons that contribute less at the spiking of the postsynaptic neurons ($\Delta t < 40ms$).
To correct the STDP characteristic the layout of the chip must be adjusted in order to have less variability between the neurons. More specifically, it must be reduced the mean variability of the T_{up} and T_{down} time lengths, since the noise jitter itself does not really alter the learning.

3.6. Power consumption estimations

As previously explained, the major advantage of this technology is the high power efficiency. To estimate the power dissipation of the neuromorphic chip, another board have been assembled with the only difference of having the $0.4V$ power supply and the $0.8V$ power supply disconnected from the LDOs. Those power supplies have been directly connected to a semiconductor parameter analyzer in charge of providing the required voltage and measure expected low currents. Furthermore, the presynaptic neuron 2 was selected, instead of the presynaptic neuron 1, because of the possibility of synchronizing its spiking activity with the postsynaptic neuron one. In fact, their starvers are biased by different trimmers and thus it is possible to overcome the variability problem.

The measurements have been performed by configuring the trimmers and applying a constant bias to both $Vin2$ and $Vin\ post$ in order to make the presynaptic neuron 2 and the postsynaptic neuron spike at $\approx 91Hz$, i.e. 1 spike each $\approx 11.1ms$ (Fig. 3.7 **b**). At the input $Vin2$ was also applied a constant voltage of $400mV$ in order to prevent any current injection as was done in the previous measurements. With the parameter $0.4V$ and $0.8V$ voltages were forced and their absorbed currents measured.

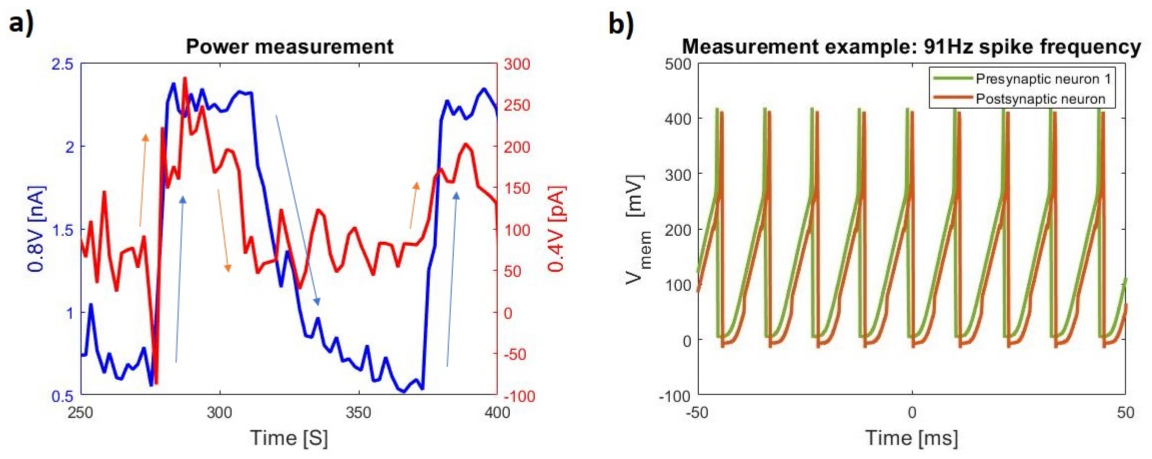


Figure 3.7: **a**) Measurement of the current required by the neurons and a portion of the timing circuit. When the neurons starts spiking there is a visible increase of the current required by the two power supplies (up pointing arrows). When the neurons stop spiking the current decreases by the same amount (down pointing arrows). **b**) Measurement example of the $Vout$ signals when the neurons have been turned on.

The resulting measurement (Fig. 3.7 **a**) clearly shows the effective power dissipation of the neurons ($0.4V$) and a portion of the timing circuit ($0.8V$). When the current is injected into the neurons there is a visible increase by both the currents required by the $0.4V$ and $0.8V$ supplies. The current drained by the core circuits of the two neurons is $\approx 100pA$. From this value, the real power consumption of a single neuron can be estimated as $Power = 0.4V \cdot 50pA = 20pW$ and the real energy consumption for a spike $E_{spike} = 20pW \cdot 11.1ms/spike = 222fJ$ (overestimation, because it includes also the power dissipation of the neurons input current mirrors, that would require $\approx 40pA$ of input current each wit a power dissipation of $\approx 16pW$ each). These results are still an order of magnitude greater than the simulated one [34], but still impressive since a biological neuron consumes $100pJ$ per action potential and $10fJ$ per synaptic transmission [49]. Furthermore, it has been discovered that a portion of the timing circuit alone consumes one order of magnitude more than the neuron circuit. This was not really expected since it is just composed by some digital logic ports, however is not a big issue, since the circuit would be turned on only during learning.

4 | Results and simulations

4.1. Introduction

As shown in the previous chapter, the process variability alters considerably the timing shape of the HV signals resulting in the fault of the STDP, which could not even recognize the causality of the spikes. However, even if the STDP would have worked fine, the fact that the chosen training algorithm is an unsupervised learning only based on the time and causality of the local spikes limits the network capability of understanding the input data and generating a proper prepossessing or classification.

In order to still manage to use this incredibly low power dissipation technology another training algorithm has been thought and will be presented in this chapter. A supervised learning was preferred because of the purpose of performing real time classification.

4.2. The datasets

Two different datasets and different network sizes have been used to train the network. The first dataframe is a dummy and very simple dataset, based on colors, that have been used to understand and test more easily the algorithm. The second is the effective dataset used to evaluate the performance of the training algorithm.

4.2.1. Binary color dataset

The dummy dataframe is composed by the bits that determine a simple batch of 8 colors (Table 4.1). The task to perform on this dataset is to train the SNN to be able to classify the colors between two classes: cold colors and hot colors. This dataframe has been used to start testing the algorithm mainly because at the beginning of the testing phase the effective dataset was not available yet. However, every version of the algorithm was first tested with this dataset because of its simplicity and reduced size. In fact, it helped to debug more easily the code and to catch errors in the algorithm that would prevent convergence.

Color dataset

	White	Blue	Red	Cyan	Lime	Magenta	Yellow	Black
Blue	0	1	0	1	0	1	0	1
Green	0	0	0	1	1	0	1	1
Red	0	0	1	0	0	1	1	1
Cold	x	x		x				x
Hot			x		x	x	x	

Table 4.1: The color dataset used to test and understand the working principle of the training algorithm.

4.2.2. Binary HAR dataset

HAR dataset description

	Axis 1	Axis 2	Axis 3
Count	1630564	1630564	1630564
Mean	0.123	-0.020	0.347
Std	0.732	0.638	0.559
Min	-4.000	-5.391	-4.000
25%	-0.438	-0.375	0.000
50%	0.180	0.000	0.406
75%	0.562	0.266	0.828
Max	4.000	5.812	4.266

Table 4.2: In the table are listed statistical value representing the data acquired from each of the accelerometers. They represent: the number of samples, the mean value, the standard deviation, the minimum value, the percentiles at 25%, 50%, 75% and the maximum value.

The effective dataset consists of approximately 1.600.000 raw data samples (Table. 4.2) coming from a 3-axis accelerometer installed on a phone. The dataset is provided by STMicroelectronics in arbitrary unit. They represent the instantaneous accelerations measured by the accelerometers and have been labelled into two different classes: still

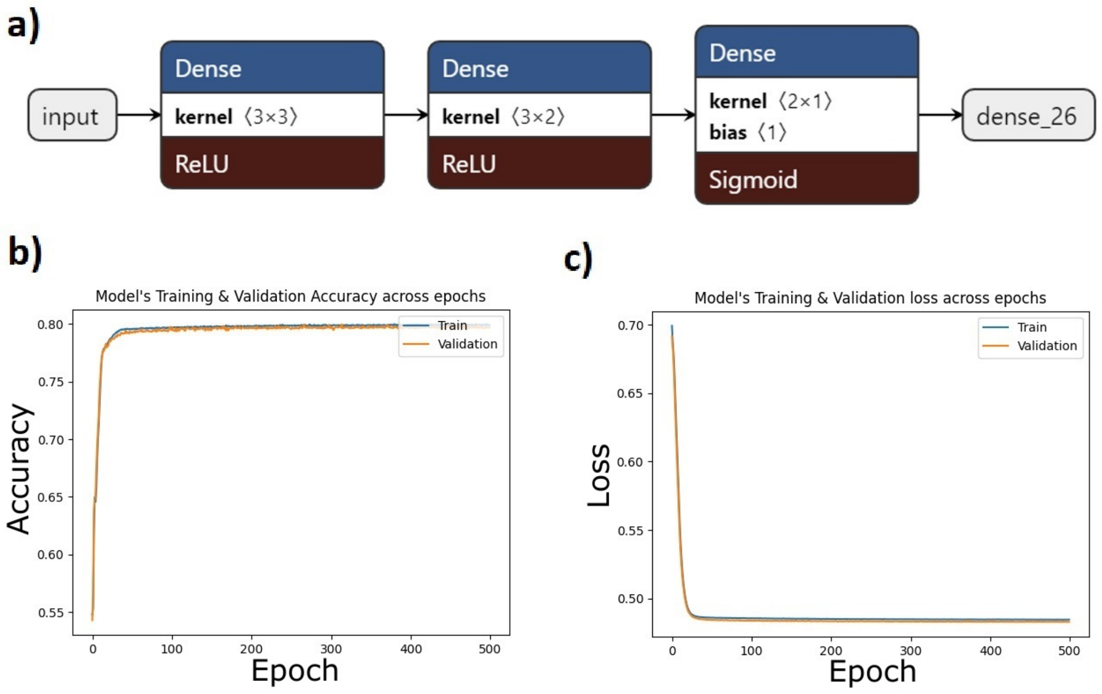


Figure 4.1: **a**) Shallow ANN. It is composed by three layers: 3-3-2. The first two layers uses the ReLu as activation function, while the last one uses the logistic function. Furthermore, only the last layer uses a bias. **b**) Graph of the accuracy achieved by the network during each epoch. **c**) Graph of the Loss function per epoch.

and walk. The idea behind it is to be able to identify if a person is walking or staying still only by using the output of the accelerometers and a neural network.

As a reference It was also provided a fully connected Shallow ANN (Fig. 4.1 **a**)compatible with an implementation on phones and is implemented on a microcontroller. The network is a three layers feedforward network (3-3-2). The first two neuron layers uses the ReLu activation function, while the last one uses the sigmoid function (i.e. logistic function). The network has been trained with about 100.000 samples to speed up the training process. Furthermore, no Features Engineering operation was performed on the dataset but only basic data pre-processing operations (like centering and variance normalization). The ANN achieves an accuracy close to 80% both on the training set and on the test set (Fig. 4.1 **b**, **c**).

4.3. The learning algorithm

The choice of the algorithm is the result of a long study of the state of the art spiking neural networks training combined with the need of having efficient results that could be obtained in reasonable time. Furthermore, the algorithm should have been compatible with our technology or adapted to it.

The need of fast results and the high variability of the technology have suggested the implementation of an online learning. This way, the design of the dedicated hardware would have been limited to the acquisition-writing chain, without needing of more sophisticated hardware. Furthermore, it was decided to uncouple learning an inference. This way, the network could have been designed accordingly to the task and programmed only once.

The second step to determine the final choice has been to decide if it was better to use a mathematical optimization algorithm or keep going with a biologically inspired one. Since the objective was to obtain practical and accurate results and since it was already decided that the implementation would have been online, it was set that a mathematical optimization algorithm would suit better the circumstances. In fact, after the choice of an online learning the only constraint was to keep a low power dissipation of the inference, which did not change much if the learning is biologically inspired or mathematical optimizations. Because of that, it was established that a mathematical optimization one would have been better, since they have, on average, an higher accuracy.

4.3.1. Reference algorithm

For the algorithm choice the standards ANN converted into SNN techniques were discarded from the beginning, because of their high spiking rates and the need of rate-based neurons. The algorithm developed by J. Büchel, D. R. Muir et al. [28] was taken into account, however, the strength of the latter algorithm is the ability of recognizing time-variant data, which was useless for this task and would have just added complexity to the layout of the hardware implemented network because of the needed recurrent connections. Since also the Smoothed SNN techniques requires to change the neuron model the choice fell on the surrogate gradient method. The decreed algorithm was originally developed by A. Renner, A. Sornborger et al. [7]. It is a mathematical optimization supervised training belonging to the family of surrogate gradient learnings. It was preferred because the algorithm encourages only the neurons strictly necessary to stimulate the right output neurons, without the need of making the neurons responsible to inhibit all the others output neurons to spike.

This method have been applied to a four layer SNN ($x = 3 - h_1 = 4 - h_2 = 3 - o = 2$), whose functioning can be described as follows:

$$o = f(W_3 f(W_2 f(W_1 x)))$$

$$f(x) = H(x - V_{th})$$

$$H(x) = \begin{cases} 0 & \text{if} \quad x < 0 \\ 1 & \text{if} \quad x \geq 0 \end{cases}$$

Where x is a binary input array, o the binary output array, W_1 $(4x3)$, W_2 $(3x4)$ and W_3 $(2x3)$ are the three synaptic weight matrices, $f(\cdot)$ is the spiking activation function and $V_{th} = 200mV$. The network was trained with the targets t using the loss function $L = \frac{1}{2}\|o - t\|^2$. It is also important to denote that differently from standard procedure, the pseudo backpropagation was performed after each inference.

The algorithm in question is:

$$d_3 = (o - t) \circ f'(W_3 h_2)$$

$$d_2 = sgn(W_3^T d_3) \circ f'(W_2 h_1)$$

$$d_1 = sgn(W_2^T d_2) \circ f'(W_1 x)$$

$$\frac{\partial L}{\partial W_l} = d_l (a_{l-1})^T$$

$$W_l^{new} = W_l^{old} - \eta \frac{\partial L}{\partial W_l} \qquad l = 1, 2, 3 \tag{4.1}$$

Where t is the two elements binary target array (one for each class), d_l are the backward propagated local gradients (same sizes as the weight matrices), which represent the amount by which the loss changes when the activity of a neuron changes. $\circ$ represent the elementwise product between vectors, $sgn(x)$ is the sign function, and a_l denotes the activation of the layer l (1 if a spike occurred, 0 otherwise) i.e. $f(W_l a_{l-1})$ with $a_0 = x$, $a_1 = h_1 = f(W_1 x)$, $a_2 = h_2$, $a_3 = o$. The only hyperparameter of the algorithm is the learning rate η. Finally, $f'(\cdot)$ is the pseudo derivative of the activation function. The latter was selected as:

$$f_{surrogate}(x) = min(max(x, 0), 1)$$

I.e. a truncated ReLu function between 0 and 1, whose derivative can be calculated as:

$$f'(x) = H(x) - H(x - 1)$$

This pseudoderivative was the one originally chosen in [7] because of its simple hardware implementation. It have been decided to keep it, in order to test the effectiveness of the algorithm as it was originally thought.

This method recalls the backpropagation, however the weight change is forced to happen only when spikes occur. With no spiking neurons the weight could not be updated and thus the network would not be able to learn. In fact, after the inference, the algorithm understands the path of spiking neurons that caused the output neurons to spike. By comparing this knowledge with the wanted target, it is able to calculate the sign of the weight update (only the weight between two neuron that spiked are considered) to approach the network convergence. After that, the weight gets updated accordingly to the pseudo derivative and the input of the involved neurons. If the backpropagated surrogate gradient is negative, the corresponding weights would be reduced, while the corresponding weights would be increased if the surrogate gradient is positive. This way, the loss function calculated after the next inference would result reduced and the network would have moved closer to the global minimum, i.e convergence.

The algorithm was implemented on Intel's Loihi [16] by A. Renner, A. Sornborger et al. in a three layer SNN (400-400-10) [7] and was able to achieve a 95.7% accuracy on the MNIST test dataset. This is comparable with the performance of other shallow, stochastic gradient descent (SGD) trained multilayer perceptron models without additional allowances. However, this impressive result was achieved with a network size of 810 neurons and a total of 164000 synaptic connection. Sizes which are far behold the capability of our technology. This is the reason why the number of neurons synapses have been reduced for our implementation. However, this reduction of the sizes of the network trained with the HAR dataset have resulted in the total non-convergence of the network. In the next section the emerged problematic will be discussed and the actuated solutions explained.

4.4. Simulation models

To evaluate the effectiveness of the algorithm different conditions have been simulated. All the simulations of the SNNs have been performed on MatLab with the same neuron and synapse models. During the simulations the algorithm itself has been modified, together with the input encoding. On the contrary, the reference ANN has been simulated by python code using standard tools (TensorFlow).

4.4.1. Neuron and synapse models

The neurons of the simulated spiking neural networks are LIF neurons that behaves in a similar way to the ones on the chip. They have a simulated capacitance with a linear discharge time of $-0.0541V/s$, approximated from the measurements data. When a spike occurred an amount of current would be injected into the simulated capacitance causing

the increase of its relative V_{mem}. After crossing a threshold of $200mV$ a spike would occur and the capacitance would reset. Furthermore, to simulate the refractory period, for a certain amount of time (usually $10ms$) the capacitance is forced to stay at $0V$ (the chosen value for $V_{rest} \approx V_{down}$). The current injected into the capacitance can be positive or negative depending on the synaptic weight.
The synaptic weight chosen was the change of the membrane voltage of the postsynaptic neuron given by a single spike.

4.4.2. Input encoding

Rate coding

Since the used model has memory, to make the pseudo backpropagation work the input current was kept flowing until the input neurons would spike multiple times. To do so, it was chosen to encode the dataframes with a sort of rate coding.
For what concern the Color dataframe, the idea was that when a 1 was presented to an input neuron, the latter would have spiked once each $20ms$ (i.e. the maximum spiking frequency), while when a 0 was presented, it would have taken $80ms$ (i.e $\frac{1}{4}$ of the maximum spiking frequency). The input was presented for $200ms$ to the network in order to have at least 2 spikes when the input data was 0.
For what concern the HAR dataframe, positive and negative data were separated and fed to different neurons. After applying the module operation, it was set a range from $20ms$ per spike for the highest absolute value of the dataset to $400ms$ per spike for the lowest absolute value. The input was presented for $10s$ to the network, before evaluating the result of the inference.
This encoding showed some results, however during the various tests with the HAR dataframe it was found that this encoding is not really compatible with the neuron model and the learning algorithm. When the frequency range is too wide, the algorithm and the network fail to distinguish the different inputs.

LSB coding

To solve these issues a new encoding based on ranges was thought. Similar to how an ADC works, the input range is divided into intervals (LSB) based on the number of wanted bits. Each interval is associated with a neuron that produces a spike each $20ms$ when the input is included in its interval. As it was done before, the input would be presented to the network for a certain amount of time (multiple of $20ms$) until the evaluation and training of the network was completed. As an example, we consider input data in the range between -5 and 11 and a 3-bit coding. For this case 8 input neurons would be needed to encode the data and the LSB would be of 2. If, for example, the input of the network would be equal to 0.90, i.e 3 LSB, it would make the 3rd neuron spike. By using this encoding, the spiking frequency could be adjusted to be almost the same for all the input neurons. In fact, with a low enough refractory period, the only factor determining the spiking frequency of the input neurons would be the input current, which can easily be tuned.

4.4.3. Accuracy and MSE calculation

To evaluate the performance of the neural network it was decided to use the accuracy and mean squared error. Both these metrics are used to evaluate how much of the dataset is rightly catalogued by the network. The accuracy was calculated independently of the algorithm variations, while the MSE would be calculated in different ways depending on when the pseudo backpropagation was performed.

The output layer of all the simulated networks was composed by 2 neurons, each one representative of one of the dataset classes. When a new data is presented to the network, the input neurons would keep spiking for a certain amount of time. This, through the inference of the network, would also generate the output neurons to spike. The output neuron that happens to spike more frequently during this period would be considered as $output = 1$, while all the other output neurons would be considered as $output = 0$. After the whole set data is presented to the network the accuracy is then calculated as:

$$accuracy = \frac{N_{right}}{N_{tot}}$$

Where N_{right} is the number of right classified data, while N_{tot} the number of data in the evaluated set.

The other parameter chosen to understand the convergence of the network is the MSE. Two different MSE have been calculated, based on when the pseudo backpropagation was performed. The first one was used when the error was updated after $20ms$, i.e. at the maximum neuron frequency (with both coding). It was calculated as:

$$MSE_{per\_cyc} = \frac{1}{2}\sum_{i=1}^{N_{tot}}\sum_{k=1}^{n_{cyc}}\frac{\|o_{i,k} - t_i\|^2}{N_{tot}n_{cyc}}$$

Where n_{cyc} is calculated as the updating time of the error ($20ms$) divided by the inference time selected for one input data, which is also the number of times a spike would have occurred in a neuron that was spiking at maximum frequency. It was chosen to use the squared norm of error to obtain a scalar value, because the considered error is a vector.

The second method of calculating the MSE does not have any dependence on the number of times a spike would have occurred in a neuron that was spiking at maximum frequency. It is introduced because in the latter version of the algorithm the meaningful information for the backpropagation of the error was if a neuron spiked more than a certain number of times, instead of if the neuron spiked at all. In fact, instead of updating the error every $20ms$, the error was updated after every evaluation (inference) of one input data. The second MSE was calculated as:

$$MSE = \frac{1}{2}\sum_{i=1}^{N_{tot}} \frac{\|o_i^{th} - t_i\|^2}{N_{tot}} \tag{4.2}$$

Where o_i^{th} is a two element vector, whose elements are 1 if their equivalent output neurons have spiked more than a selected amount of time and 0 otherwise.

4.5. Algorithm simulations and development

The process of understanding the classification capability of the chosen algorithm and its further development happened in different phases, that will be explained in a temporal order. This way, it should result simpler to understand the thinking process behind it, which is affected by the knowledge and resources of that specific phase.

4.5.1. Simulations to understand the algorithm

The first phase was dedicated to understand how the training was performed and how to actually make the network learn. The dataset used for these experiments was the Color dataframe for its simplicity.
With the previously explained neuron and synapse models (subsection 4.4.1) a four layer network (3-4-2-2) has been implemented. This arrangement was chosen both to see if problems of vanishing gradients would occur and to try to increase the network accuracy by increasing the input dimensionality with the second layer. During the training, the input data were encoded with the rate coding (subsection 4.4.2) listed in table 4.3. The order of the input data was randomly chosen and the same input was presented to the network for $200ms$. During this period, the spiking activity of the output neurons were monitored and every $20ms$ the error and equivalent weight updates were calculated with the algorithm previously described (subsection 4.3.1). After a batch of 8 data (the whole

inputs and outputs of 3-4-2-2 network

	Input 1	Input 2	Input 3	Hot	Cold
White	50 mV	50 mV	50 mV	0	1
Blue	50 mV	50 mV	200 mV	0	1
Red	50 mV	200 mV	50 mV	1	0
Cyan	50 mV	200 mV	200 mV	0	1
Lime	200 mV	50 mV	50 mV	1	0
Magenta	200 mV	50 mV	200 mV	1	0
Yellow	200 mV	200 mV	50 mV	1	0
Black	200 mV	200 mV	200 mV	0	1

Table 4.3: The table lists the voltage increase of the V_{mem} voltage of the three input neurons each $20ms$ and the desired spiking activity of the output neurons for the color dataframe.

dataset), the weight updates would be summed and normalized by the number of presented inputs before applying the effective weight update:

$$W_l^{new} = W_l^{old} - \frac{\eta}{N_{tot}} \sum_{k=1}^{N_{tot}} (\frac{\partial L}{\partial W_l})_k$$

where $N_{tot} = 8$.

Unexpectedly, even if different learning rates from 0.001 to 1 have been used, the network was not able to reach convergence on the dataset (Fig. 4.2 **a**). In order to solve this issue and inspired by [28], the pseudo activation function (truncated ReLu) used for backpropagating the error was changed with an hyperbolic tangent function:

$$f(x) = \frac{1}{V_{th}} tanh(\frac{x}{V_{th}} - V_{th})$$

Whose derivative is:

$$f(x) = 1 - tanh^2(\frac{x}{V_{th}} - V_{th})$$

Where $V_{th} = 200mV$.

With this modification the network was finally able to converge. The tanh derivative (Fig. 4.2 **d**) approximates much better the derivative of a spiking neuron (δ function) respect to the derivative of the ReLu function (Fig. 4.2 **c**). In fact, the new smoother derivative

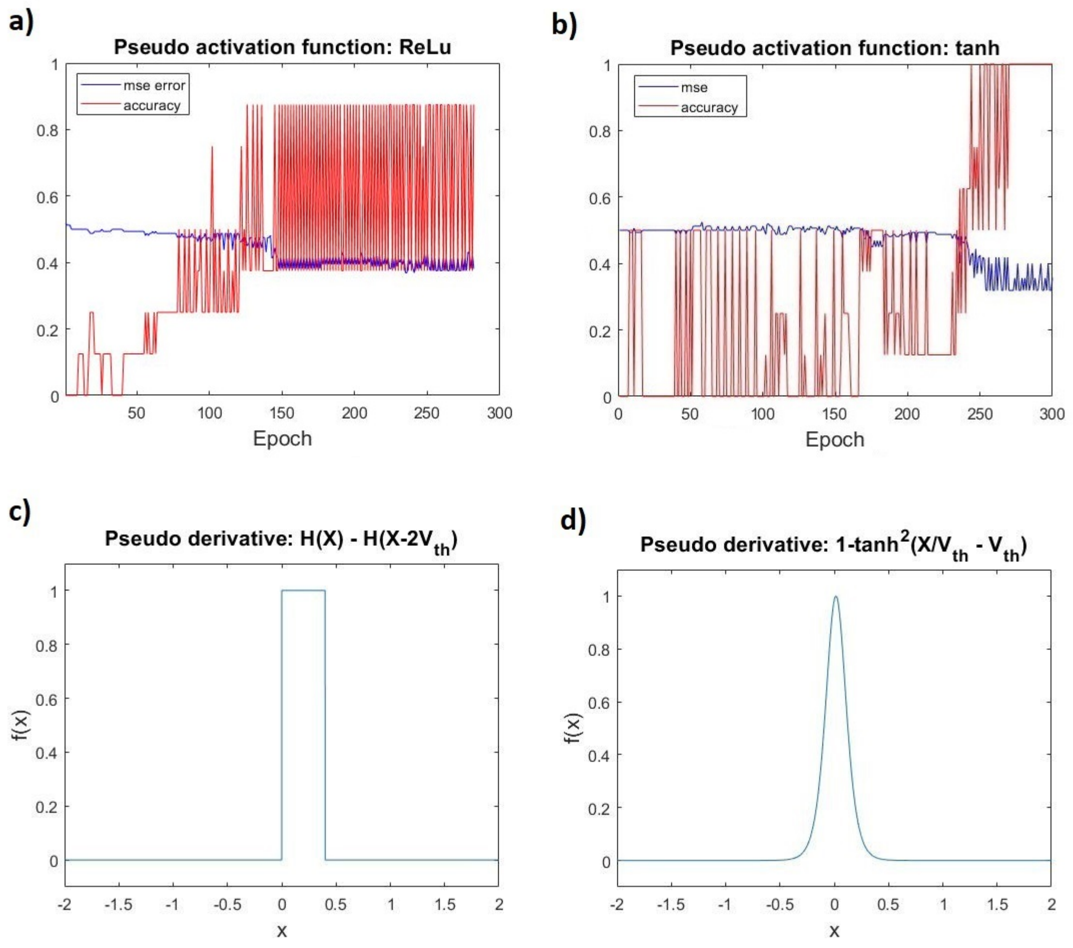


Figure 4.2: **a**) Accuracy and MSE of the (3-4-2-2) network trained with ReLu as pseudo activation function. The algorithm is unable to properly approximate the real gradient and thus cannot achieve convergence. **b**) Accuracy and MSE of the (3-4-2-2) network trained with tanh as pseudo activation function. **c**) Truncated Relu derivative. **d**) tanh derivative.

permits to have a smoother gradient, which approximates better the real gradient. Another important modification was the weight initialization. In [7] the weight matrices were initialized by sampling them from a Gaussian distribution with mean of 0 and a standard deviation of $1/\sqrt{2/(N_{fanin} + N_{fanout})}$, where N_{fanin} denotes the number of neurons of the presynaptic layer and N_{fanout} the number of neurons of the postsynaptic layer. However, as previously mentioned, the algorithm needs spikes to backpropagate the error. Without spikes in the network, the weight update would always be $\partial L \partial W_l = 0$. Moreover, by centering the distribution of weight initialization in 0 and without a big variance, the probability of spiking is really low. Since the simulated network has much smaller size than the one in [7], a portion of neurons that could not spike from the beginning was lost

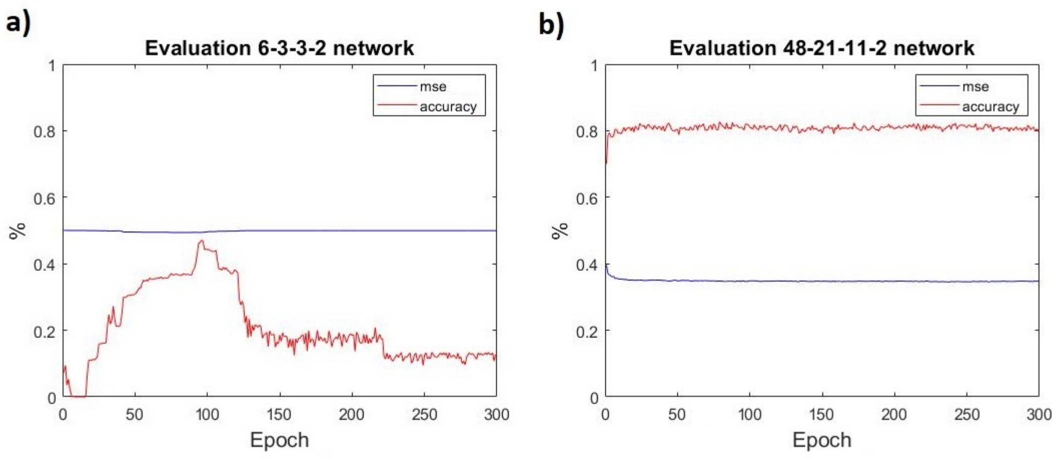


Figure 4.3: **a**) 6-4-3-2 network evaluation. It is clear that the network is unable to fit the training set. **b**) 48-21-11-2 network evaluation.

and could not be recovered. To overcome this issue, the synaptic weight initialization has been sampled from a random distribution still centered in 0, but with $1.1V_{th}$ variance. This new weight initialization granted that almost all the neuron could spike after all the different inputs were propagated through the network (inference).

However even with these modifications, the convergence of the network (Fig. 4.2 **b**) was achieved in a very irregular way and with 300 cycles i.e. 1min of training time. This was still a poor performance if taken into account the simplicity and reduced size of the dataframe.

4.5.2. Simulation with HAR dataframe

Since the HAR dataset happens to have both positive and negative values it was thought to use the first three input neurons to encode the positive values and the latter three input neurons to encode the negative ones. After that a portion of 100.000 data equally distributed between the two classes were randomly chosen from the 1600000 available. This dataframe was further divided into a training set (80%) and a test set (20%). The data were also converted into currents to be fed to the input neurons and were normalized in order that the maximum possible input would increase V_{mem} by $200mV$, i.e. make the neuron spike (rate coding, previously introduced in subsection 4.4.2). A 6-4-3-2 was simulated and trained in the exact same way of the 3-4-3-2 network, with the only difference of this new training set and the new pseudo derivative. However, once again the network was not able to converge (Fig. 4.3 **a**) even if a wide range of numbers were used for the learning rate.

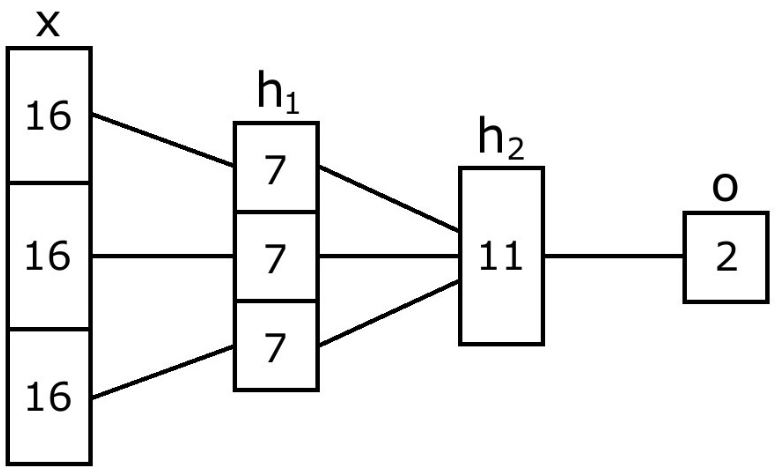


Figure 4.4: 48-21-11-2 network feedforward connections. The neuron clusters connected by straight lines are fully connected.

The solution of this problem consisted of changing the input encoding. Apparently the wide range of rates could not be understood by a non rate based algorithm. The new coding was the LSB coding explained in subsection 4.4.2. In order to cope with the new size of the dataset, i.e. 16 neurons for each axis input data, the network size was modified into 48-21-11-2. The sizes of the hidden layers have been chosen odd in order to break the network symmetry and force different and complimentary connections between the layers i.e force different feature extraction. Moreover, it has been chosen to use the first hidden layer to reduce the input size. In fact, the input layer is not fully connected to the first hidden layer, but the input neurons corresponding to each axis are only connected to 7 neurons of the hidden layers (Fig. 4.4). The order of the input data was randomly chosen and the same input was presented to the network for $120ms$ this time, i.e. the time equivalent of 6 input neuron spikes, because it was the lowest evaluation time that still permitted to achieve convergence. During this period, the spiking activity of the output neurons was monitored and each $20ms$ the error and equivalent weight updates would be calculated ($output = 1$ if there was a spike, 0 otherwise). This time, inspired by the Shallow ANN, a batch of 1000 data was chosen. The weight updates would be summed and normalized by $N_{batch} = 1000$ and then performed :

$$W_l^{new} = W_l^{old} - \frac{\eta}{N_{batch}} \sum_{k=1}^{N_{batch}} (\frac{\partial L}{\partial W_l})_k$$

Finally, the network has proven able to fit the data and achieve convergence (Fig. 4.3

b). The final accuracy of 80% means that the network was not able to achieve the global minimum. However, since also the Shallow Net could not pass the 80% accuracy (Fig. 4.1 **b**), it means that probably the instantaneous acceleration alone is not enough to achieve a better classification accuracy. Furthermore, this version of the network was able to converge in less than 10 training cycles, which is a huge improvement compared to the previous versions.

4.5.3. Algorithm with memory adaptation

The algorithm previously explained (subsection 4.3.1) was thought for a SNN without memory, and thus theoretically a network can learn with its LIF neurons spiking only one time per inference and without retain memory of the past. It is intuitive that a network without memory would achieve lower performance that a network with memory of the same size. Since our neuron model has memory, it has been decided to further adapt the algorithm to our model by imposing the need of multiple spikes from a single neuron to identify the backwards path that lead to the right classification. The training algorithm used this time is the following:

$$d_3^{th} = (o - t) \circ f'(W_3 h_2^{th})$$

$$d_2^{th} = sgn(W_3^T d_3^{th}) \circ f'(W_2 h_1^{th})$$

$$d_1^{th} = sgn(W_2^T d_2^{th}) \circ f'(W_1 x^{th})$$

$$\frac{\partial L}{\partial W_l} = d_l^{th} (a_{l-1}^{th})^T$$

$$W_l^{new} = W_l^{old} - \eta \frac{\partial L}{\partial W_l} \qquad l = 1, 2, 3 \tag{4.3}$$

Where t is the two elements binary target array (one for each class), d_l^{th} are the surrogate backward propagated local gradients (same sizes as the weight matrices), which represent the amount by which the loss changes when the activity of a neuron changes in a certain amount of time. $\circ$ represent the elementwise product between vectors, $sgn(x)$ is the sign function, and a_l^{th} is a vector in which the elements denote if the neurons in the layer l were activated, i.e. $f(W_l a_{l-1})$ with $a_0 = x$, $a_1 = h_1 = f(W_1 x)$, $a_2 = h_2$, $a_3 = o$, following this rule: $a_l(k) = 1$ if the k-neuron of the layer l was activated more than N_{th} times when the same data is presented n_{cyc} times to the network, while $a_l(k) = 0$ if the k-neuron of the layer l was activated less than N_{th} times when the same data is presented n_{cyc} times to the network. The hyperparameter of the algorithm is the learning rate η. Furthermore,

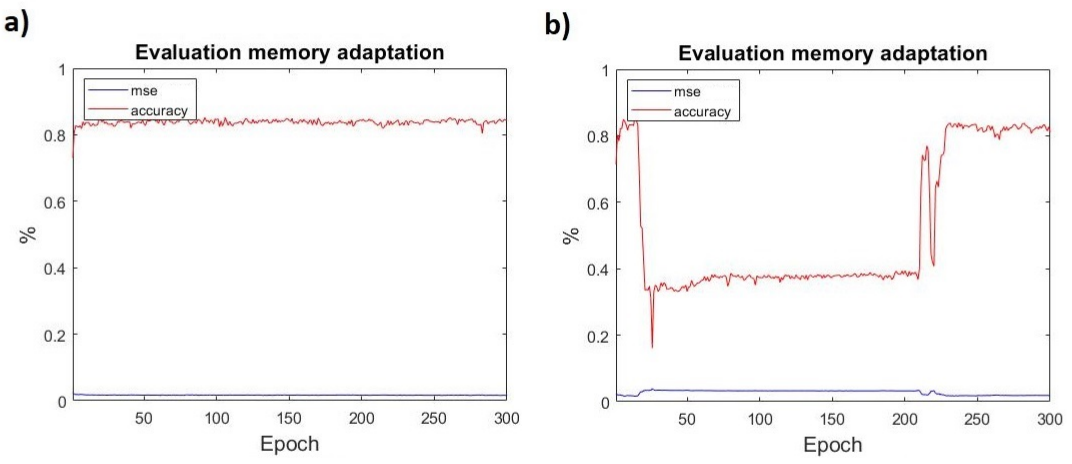


Figure 4.5: **a**) Evaluation of 48-21-11-2 network trained with the new algorithm adapted to force the use of memory. The network almost instantly converges to the minima. **b**) Evaluation of 48-21-11-2 network trained with the new algorithm adapted for force the use of memory. After reaching a local minima, the network jumps out to reach another local minima.

$f'(\cdot)$ is the pseudo derivative of the activation function $tanh(x/V_{th} - V_{th})$:

$$f(x) = 1 - tanh^2(\frac{x}{V_{th}} - V_{th})$$

This approach can be considered a mixed approach of the surrogate loss function and surrogate derivative.

Another 48-21-11-2 have been simulated and trained with the HAR dataset encoded with the LSB encoding. The order of the input data was randomly chosen and the same input was presented to the network for $220ms$, i.e. the time equivalent of 11 input neuron spikes. The N_{th} chosen to trigger the backpropagation is three spikes. Furthermore, differently from the previous simulations, the MSE have been calculated with the second method expressed in equation 4.2.

The modified algorithm seems to perform in a similar way to the previous one, a part from the fact that sometimes it seems to be able to jump from a local minimum to another. In fact, from the accuracy graph (Fig. 4.5 **b**) it can be noted that at around 30 epochs the accuracy drops significantly, to recover again around 230 epochs.

4.6. Shallow ANN and SNN comparison

4.6.1. Accuracy comparison

The fact that the initialization of weight matrices of a NN and the order of the input presented to the network are random processes makes also the process of reaching the minimum of the loss function random. To cope with this issue and properly evaluate the performance of the algorithm to train the SNN, multiple networks have been trained with the same sizes and same learning rate for both the training algorithms mentioned in the previous section (subsection 4.5.2 and 4.5.3). The final accuracy on both training set and test set achieved after 300 training epochs have been compared. Furthermore, the performances of the algorithms have also been compared also with the final accuracy achieved by the Shallow ANN previously mentioned (subsection 4.1).

The Shallow ANN have been trained with the Adam optimizer on a binary crossentropy loss function with a batch size of 1000 data for 500 epochs. This process has been repeated for 500 times with the original HAR dataset and 500 with a 4bit quantized version of it (similar to the LSB encoding) to evaluate the effect of the quantization of the dataset. The resulting accuracy variability obtained on the test sets (Fig 4.6 **a** and **b**) shows not much difference in the distributions, since they are both centered around 78% accuracy and have the same variability. The only visible effect is that the probability of achieving accuracy greater than 83% for a network trained with the quantized dataset is lower than the one of a network trained with the original dataset. The maximum accuracy achieved with the original dataset is 83.8%, while with the quantized dataset it is 84.2%.

The SNN have been trained 70 times with the normal version of the algorithm (subsection 4.5.2) and 70 times with its memory adapted version (subsection 4.5.3). The variability distributions of both test and training sets (Fig 4.6 **c**, **d**, **e** and **f**) show higher reliability of the first version, although it tends to overfit the training set. The memory adapted version of the algorithm, instead, can achieve higher accuracy overall on the test set at the expense of having an higher probability of failing to fit the dataset. However failing sometimes to fit the dataset is normal for ANN as it can be seen that also the Shallow ANN has the same behaviour ((Fig 4.6 **a** and **b**)). The maximum accuracy achieved by the normal algorithm on the test set is 73.5%, while it is 79.1% for the memory adapted algorithm.

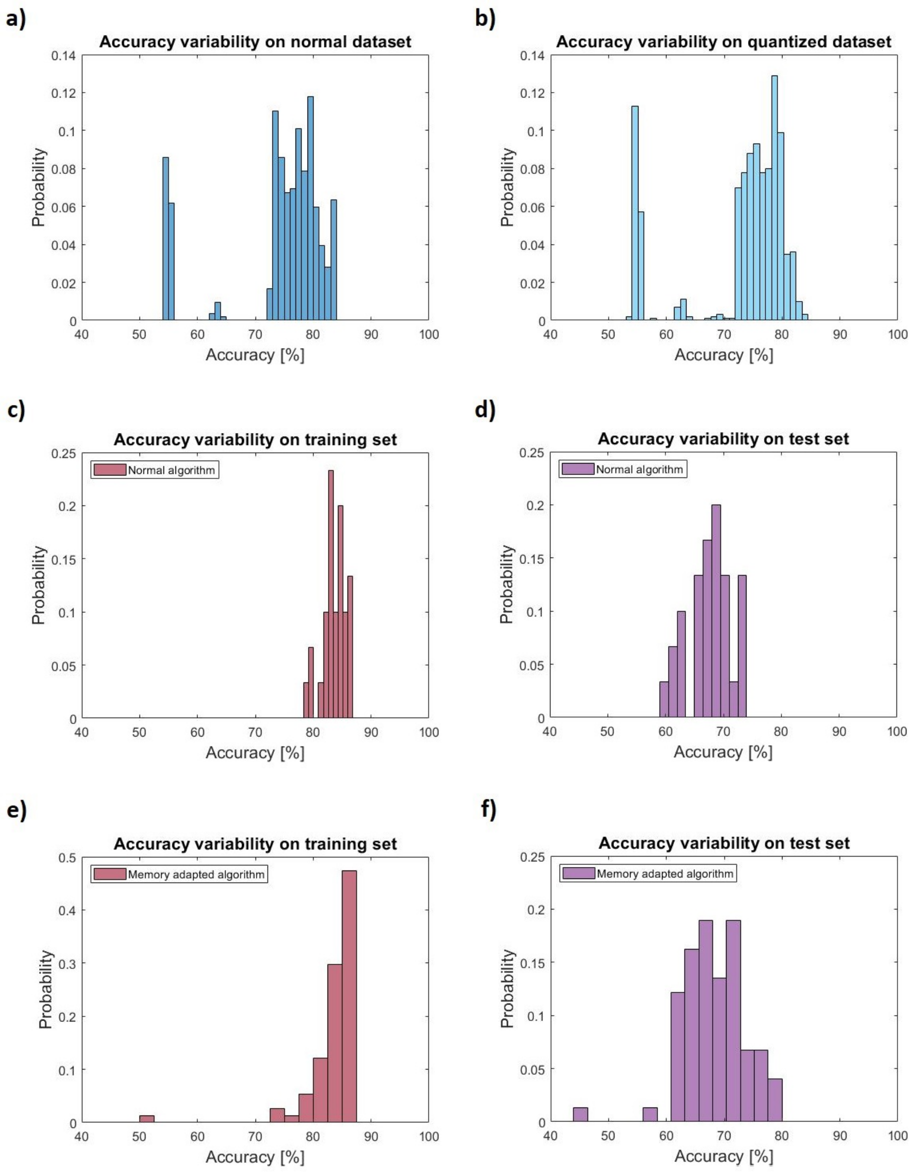


Figure 4.6: **a**) Probability distribution of the accuracy achieved by the Shallow ANN after being trained 500 times with the original HAR dataset. Accuracy calculated on the test set. **b**) Probability distribution of the accuracy achieved by the Shallow ANN after being trained 500 times with the quantized HAR dataset. Accuracy calculated on the test set. **c**) Probability distribution of the accuracy achieved by the SNN after being trained 70 times with the normal algorithm. Accuracy calculated on the training set. **d**) Probability distribution of the accuracy achieved by the SNN after being trained 70 times with the normal algorithm. Accuracy calculated on the test set. **e**) Probability distribution of the accuracy achieved by the SNN after being trained 70 times with the memory adapted algorithm. Accuracy calculated on the training set. **f**) Probability distribution of the accuracy achieved by the SNN after being trained 70 times with the memory adapted algorithm. Accuracy calculated on the test set.

4.6.2. Power consumption comparison

From the simulation results and the measurements, it has been possible to estimate the power dissipation of a SNN trained with the previous algorithms in standard CMOS technology during inference. The simulations show that a 48-21-11-2 SNN trained with the normal algorithm spikes on an average mean of 37 times to catalogue an input data, with a minimum of 24 and a maximum of 48 spikes. A SNN of the same size trained with the memory adapted version of the algorithm spikes on average 58 total times to catalogue an input data, with a minimum of 46 and a maximum of 75 spikes. Considering the worst case scenario of 75 spikes occurring simultaneously during inference, the total power dissipated by the network to catalogue an input data would be of $P_{neurons} = 75 \cdot 20pW = 1.5nW$ (high overestimation, considering that not all the neuron spikes at the same time and considering the power overestimation in section 3.6) and an energy consumption of $E_{neurons} = 75 \cdot 222fJ = 16.65pJ$ (high overestimation, see section 3.6 for power evaluation). It has not been possible to measure the exact power dissipation of a synapse, however it is possible to estimate its value by comparing the data obtained form the measurements with the data obtained from the technology simulations [34]. In its thesis work Mastella [34] estimated the neuron of dissipating $P_{neuron} = 800fW$ and $E_{neuron} = 21fJ/spike$ while estimated the maximum synapse of dissipating $P_{synapse} = 3.28pW/spike$ and $E_{synapse} = 1.65fJ/spike$. The estimated power dissipation of the synapse is one order of magnitude more than the neuron, while the energy dissipation is one order of magnitude less. If these proportions are compared with the measurements it can be estimated a real power dissipation of the synapse of $P_{synapse} \approx 50pW/spike$ and an energy consumption of $E_{synapse} \approx 22,2fJ/spike$. If it is considered that in the worst simulated scenario of 75 total spikes the input neurons spiked 33 times, the hidden layer 1 neurons spiked 20 times, the hidden layer 2 neurons spiked 17 times and the output neurons spiked 5 times, it can be estimated a total dissipation of $P_{synapses} = (33 \cdot 7 + 20 \cdot 11 + 17 \cdot 2)50pW = 24,3nW$ (overestimation, considering that not all the neuron spikes at the same time) and $E_{synapses} = (33 \cdot 7 + 20 \cdot 11 + 17 \cdot 2)22fJ = 10,67pJ$. This would lead to the estimation of a total dissipation of $P_{inference} = 1.5nW + 24,3nW \approx 26nW$ and $E_{inference} = 16,65pJ + 10,67pJ = 27.32pJ$.

Since the start of art commercial solution (LSM6DSOX by ST) based on a dedicated machine learning core, consumes $P_{micro} = 1.8V \cdot 4uA$ for the same classification task, it can be concluded that the simulated SNN implemented in standard CMOS technology would dissipate 2 orders of magnitude less. Furthermore, this SNN, differently from a microcontroller, would consume energy only during inference, since without spikes the power consumption of the network is practically negligible.

5 | Conclusion and future perspective

The aim of this thesis work was to develop a suitable supervised online-learning that could be implemented on an analog spiking neural network with floating gate synapses in Standard CMOS technology and able to achieve state of the art performance.

The first part of the thesis was focused on understanding the limitations of the previously designed neuromorphic chip of a CMOS-based spiking neural network. The prototype chip presented three neurons and two synapses and was designed to learn through Spike Timing Dependent Plasticity. The technology revealed to be extremely power efficient ($P_{neuron} \approx 20W/spike$, $E_{neuron} \approx 222fJ/spike$, $P_{synapse} \approx 50pW/spike$, $E_{synapse} \approx 22,2fJ/spike$) and to effectively be able to perform learning with the floating gate synapses. However, the design driven by low power consumption and area occupation resulted to have a greater variability than expected. This huge variability may prevent learning with the STDP rule because the network could not be able to recognize the causality of the spikes. Nevertheless, even if the STDP would have worked fine, the fact that the previously chosen training algorithm was an unsupervised learning only based on the time and causality of the local spikes, limits the network capability of understanding the input data and to generate a proper prepossessing or classification.

In the second part of the thesis, it was studied, looking at the state of the art, a better training algorithm solution for this technology. The design was driven by the will of having efficient results that could be obtained in reasonable time without increasing the power dissipation. The choice fell on a supervised mathematical optimization algorithm based on the surrogate gradient technique. The algorithm, originally hardware implemented for the Loihi neuromorphic chip, has been converted into an online learning, because of the possible problems that could emerge from the high variability of the technology. Furthermore, since the original algorithm was thought for a SNN without memory, it has been further adapted to fully exploit the memory of the designed neurons. The modification consisted in using multiple spikes from a single neuron to identify the backwards path that leads to the right classification.

The last part of this work consisted in training SNN simulated according to the data acquired from the previously designed chip. With an accurate choice of the input encoding (16 neurons for each input), random distribution to sample the weight initialization and the hyperparameter tuning the simulated networks were able to achieve performance comparable with ANNs. Furthermore, it has been estimated the dissipation of a plausible implementation of the network, which resulted to be three orders of magnitude less than the current algorithms implemented on a microcontroller for the same classification task.

The future perspective of this project would be to find a way to implement the acquisition-writing chain to read the neuron activity, and the synaptic weight. For what concerns the neuron activity it would be best to implement a counter on chip able to be turned on during inference and off once the learning is completed. Instead, the programming of the synaptic weight could be performed with a number of high voltage pulses proportional to the weight change.
Furthermore, a proper layout technique for this architecture should be investigated. The chip already showed capacitance coupling that can only worsen with the increase of the network size. An increase of these factors or the generation of resistive paths can result detrimental, considering the low currents and low voltages employed.

Bibliography

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

[2] URL https://intellipaat.com/community/9868/what-is-the-difference-between-deep-learning-and-traditional-artificial-neural-

[3] URL https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/.

[4] URL https://en.wikipedia.org/wiki/Synapse#%20/media/File:SynapseSchematic_lines.svg.

[5] J. A. G. N. R. L. F. N. E. R. E. D. Alberto Antonietti, Claudia Casellato and A. Pedrocchi. Spiking neural network with distributed plasticity reproduces cerebellar learning in eye blink conditioning paradigms. *IEEE Ttransactions ON BIOMEDICAL ENGINEERING*, 63(1):210–219, 2016.

[6] M. V. C. B. Ali Dabbous, Ali Ibrahim. Touch modality classification using spiking neural networks and supervised-stdp learning. *IEEE*, 2018.

[7] A. Z. L. T. Alpha Renner, Forrest Sheldon and A. Sornborger. The backpropagation algorithm implemented on spiking neuromorphic hardware. *arXiv*, 2021.

[8] J. B. Bernd Illing, Wulfram Gerstner. Biologically plausible deep learning — but how far can we go with shallow networks? *Neural Networks*, 118:90–101, 2019.

[9] L. B. W. M. Bernhard Nessler, Michael Pfeiffer. Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS computational biology*, 9.4(e1003037), 2013.

[10] M. B. S. Biswa Sengupta. Power consumption during neuronal computation. *IEEE*, 2014.

[11] T. V. P. Bliss and T. Lømo. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following simulation of the perforant path. *The Journal of physiology*, 1973.

[12] N. Brunel and M. C. Van Rossum. Lapicque's 1907 paper: from frogs to integrate-and-fire. *Biological cybernetics*, 97(5):337–339, 2007.

[13] W. M. Christoph Stockl. Recognizing images with at most one spike per neuron. *arXiv*, 2020.

[14] W. M. Christoph Stockl. Optimized spiking neurons classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 2021.

[15] E. V. Claudia Clopath, Lars Büsing and W. Gerstner. Connectivity reflects coding: a model of voltage-based stdp with homeostasis. *nature neuroscience*, 2010.

[16] M. e. a. Davies. Loihi: A neuromorphic manycore processor with on-chip learning. pages 82–99, 2018.

[17] R. W. DE Rumelhart, GE Hinton. Learning representations by back-propagating errors. *nature*, 1986.

[18] H. M. Emre O. Neftci and F. Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 2019.

[19] D. E. Feldman. The spike-timing dependence of plasticity. *Neuron*, 2012.

[20] J. B. Francisco L ́opez-Mu ̃noz and C. Alamo. Neuron theory, the cornerstone of neuroscience, on the centenary of the nobel prize award to santiago ram ́on y cajal. *Brain research bulletin*, 70(4-6):391–405, 2006.

[21] J. Guo. Backpropagation through time. 2013.

[22] D. O. Hebb. *The Organization of Behavior*. 1949.

[23] A. L. Hodgkin and A. F. Huxley. Propagation of electrical signals along giant nerve fibres. *Proceedings of the Royal Society of London*, 140(899):177–183, 1952.

[24] D. W. N. Huai-Xing Wang, Richard C. Gerkin and G.-Q. Bi. Coactivation and timing-dependent integration of synaptic potentiation and depression. *Nature Neuroscience*, 8(2):187—-193, 2005.

[25] D. Huh and T. J. Sejnowski. Gradient descent for spiking neural networks. *Advancement in Neural Information Processing Systems*, 2018.

[26] C. L. F. D. V. H. E. M. Ilias Sourikopoulos, Sara Hedayat and A. Cappy. A 4-fj/spike artificial neuron in 65 nm cmos technology. *Frontiers in Neuroscience*, 11:123, 2017.

ISSN 1662-453X. doi: 10.3389/fnins.2017.00123. URL `https://www.frontiersin.org/article/10.3389/fnins.2017.00123`.

[27] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE: transactions on neural networks 15.5*, pages 1063–1070, 2004.

[28] S. S. G. I. Julian Buchel, Dmitrii Zendrikov and D. R. Muir. Supervised training of spiking neural networks for robust deployment on mixed-signal neuromorphic processors. *Nature Scientific reports*, 2021.

[29] J. A. Julijana Gjorgjievaa, Claudia Clopathb and J.-P. Pfister. A triplet spike-timing–dependent plasticity model generalizes the bienenstock–cooper–munro rule to higher-order spatiotemporal correlations. *PNAS*, 18(48):19383—-19388, 2011.

[30] H. W. K. Hornik, M. Stinchcombe. Multilayer feedforward networks are universal approximators. 1989.

[31] B. Katz. The release of neural transmitter substances. *Liverpool University Press*, pages 5–39, 1969.

[32] M. Lenzlinger and E. H. Snow. Fowler-nordheim tunneling into thermally grown sio2. *Journal of Applied Physics*, 40.1:278–283, 1969. doi: 10.1063/1.1657043. URL `https://doi.org/10.1063/1.1657043`.

[33] G. Q. B. M. C. W. van Rossum and G. G. Turrigiano. Stable hebbian learning from spike timing-dependent plasticity. *The Journal of Neuroscience*, 20(23):8812—-8821, 2000.

[34] M. Mastella. Analog spiking neural network with floating gate synapses in standard cmos technology. Master's thesis, Politecnico di Milano, 2019.

[35] C. Y. S. W. P. S. M. S. N. B. C. V. E. A. A. S. A. Md Zahangir Alom, Tarek M. Taha and V. K. Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv*, 2018.

[36] J. S. Mohammadali Sharifshazileh, Karla Burelo and G. Indiveri. An electronic neuromorphic system for real-time detection of high frequency oscillations (hfo) in intracranial eeg. *Nature COMMUNICATIONS*, 2021.

[37] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38 (8):114–117, 1965.

[38] G. E. Moore. Progress in digital integrated electronics. *International Electron Devices Meeting*, pages 11–13, 1975.

[39] P. M. N. Zheng. *Learning in Energy-Efficient Neuromorphic Computing.*

[40] *MC78M00, MC78M00A Series, 500 mA Positive Voltage Regulators.* ON Semiconductor, 18 edition, 8 2006. URL `https://pdf1.alldatasheet.com/datasheet-pdf/view/173643/ONSEMI/MC78M05ABDT.html`.

[41] G. G. T. Per Jesper Sjöström and S. B. Nelson. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron*, 32:1149—-1164, 2001.

[42] J.-P. Pfister and W. Gerstner. Triplets of spikes in a model of spike timing-dependent plasticity. *The Journal of Neuroscience*, 26(38):9673—-9682, 2006.

[43] C. Polidori. Analog spiking neural network with floating gate synapses in standard cmos technology. Master's thesis, Politecnico di Milano, 2021.

[44] E. Polidori. Analog spiking neural network with floating gate synapses in standard cmos technology. Master's thesis, Politecnico di Milano, 2021.

[45] G. qiang Bi and M. ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 1998.

[46] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958.

[47] K. N. S. Salahuddin and S. Datta. The era of hyper-scaling in electronics. *Nature Electronics*, 1(8):442–450, 2018.

[48] S. J. T. T. M. Saeed Reza Kheradpisheh, Mohammad Ganjtabesh. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018.

[49] A. Sandberg. Energetics of the brain and ai. *arXiv*, 2016.

[50] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017.

[51] K. D. M. Sen Song and L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 2000.

[52] S. S. H. B. P. P. P. K.-S. N. S.-E. C. Sina Shaffiee Haghshenas, Behrouz Pirouz and Z. W. Geem. Prioritizing and analyzing the role of climate and urban parameters in the confirmed cases of covid-19 based on artificial intelligence applications. *Evioronmental Research and Public Health*, 2020.

[53] *LM1117 800-mA, Low-Dropout Linear Regulator.* Texas Instruments, 2 2000. URL `https://www.datasheets.com/en/part-details/lm1117mp-1-8-nopb-texas-instruments-19638970#datasheet`. Revised June 2020.

[54] *UCC284–5, UCC284–12, UCC284–ADJ, UCC384–5, UCC384–12, UCC384–ADJ LOW-DROPOUT 0.5-A NEGATIVE LINEAR REGULATOR.* Texas Instruments, 1 2000. URL `https://pdf1.alldatasheet.com/datasheet-pdf/view/110138/TI/UCC384DP-ADJG4.html`. Revised February 2002.

[55] *SN74LVC3G17 Triple Schmitt-Trigger Buffer.* Texas Instruments, 8 2003. URL `https://www.datasheets.com/en/part-details/sn74lvc3g17dctre4-texas-instruments-21573272`. Revised August 2015.

[56] G. G. Turrigiano and S. B. Nelson. Homeostatic plasticity in the developing nervous system. *Nature reviews neuroscience*, 2004.

[57] M. M. Waldrop. The chips are down for moore's law. *Nature*, pages vol. 530, no. 2, pp. 144–147, 2016.

[58] H.-S. P. Wong and S. Salahuddin. Memory leads the way to better computing. *Nature nanotechnology*, 10(3):191–194, 2015.

[59] Y. R. Y. L. H. X. X. Z. Y. L. Zhongqiang Wang, Tao Zeng and D. Ielmini. Toward a generalized bienenstock-cooper-munro rule for spatiotemporal learning via triplet-stdp in memristive devices. *Nature COMMUNICATIONS*, 2020.

A | Appendix A

A.1. MatLab code

A.1.1. LSB coding

```
clear; close

% % Preprocessing parameters %

divisions = 16;

reduced_data.axis = csvread("x_test.csv"); %read csv file containing the dataset

reduced_data.labels = csvread("y_test.csv"); %read csv file containing the dataset classes

spiking_data.axis = [];

folder_save = ""; %path for saving the encoded dataset

for i = 1:3

eval("max_"+ num2str(i) +" = max(reduced_data.axis(:,i));");


eval("min_"+ num2str(i) +" = min(reduced_data.axis(:,i));");

eval("division_"+ num2str(i) +" = (1:1:divisions)*((max_"+ num2str(i) +"-min_"+
num2str(i) +")/divisions) + min_"+ num2str(i) +";");


eval("division_"+ num2str(i) +" = cat(2,min_"+ num2str(i) +",division_"+ num2str(i)
+");");

end

spiking_data.axis = [];

for i = 1:divisions
```

```
temp_1 = (reduced_data.axis(:,1) >= division_1(i)).*(reduced_data.axis(:,1) < division_1(i+1));

temp_2 = (reduced_data.axis(:,2) >= division_2(i)).*(reduced_data.axis(:,2) < division_2(i+1));

temp_3 = (reduced_data.axis(:,3) >= division_3(i)).*(reduced_data.axis(:,3) < division_3(i+1));

spiking_data.axis = cat(2,spiking_data.axis,temp_1,temp_2,temp_3);

end

spiking_data.labels = cat(2,reduced_data.labels,not(reduced_data.labels)).';

save(folder_save+"test_spiking_data.mat","spiking_data"); % save LSB coded data
```

A.1.2. Network training with normal algorithm

```
clear;close;

%% Network parameters %random_initialization = 1;
dt = 80e-3; %[sec]
threshold = 0.2; %[V]
neuron_for_layer = [48,21,11,2];
current_sinking_time = 79.99e-3; %[sec]
max_weigth = 0.4; %[V]
max_epoch = 500;
epoch_break_point_0 = 25;


batch_size = 500;
learning_rate = 0.4;
num_train_cyc = 6; %number of consecutive training for each input
mse_min = 0.3; %mse to be achieved to stop the loop
slope = -0.0541; %slope of the leakage


reload_network = 0; %choose the netwok you want to keep training

num_layer = length(neuron_for_layer);

%designed folder were the previously LSB encoded data are stored
folder = "";
```

```
folder_network = folder+"networks";
folder_parameters = folder+"network_parameters";

%% Loading input data %

load(folder+"spiking_data.mat");

num_max_input = length(spiking_data.axis(:,1));
spiking_data.axis = spiking_data.axis/max(max(spiking_data.axis))*(threshold);

for i = 1:num_layer eval("layer"+num2str(i)+".spike = zeros(neuron_for_layer(i),1)*nan;");
eval("layer"+num2str(i)+".capacitance = zeros(neuron_for_layer(i),1);");
eval("layer"+num2str(i)+".last_activation = ones(neuron_for_layer(i),1)*
current_sinking_time;")
end

%eliminating unused connections between input layer and first hidden layer


num_l1 = neuron_for_layer(1);
num_l2 = neuron_for_layer(2);
w_cancel = ones(num_l2,num_l1);

w_cancel(1:num_l2/3,num_l1/3+1:end) = zeros(num_l2/3,num_l1/3*2);
w_cancel(num_l2/3+1:num_l2/3*2,num_l1/3*2+1:end) = zeros(num_l2/3,num_l1/3);
w_cancel(num_l2/3+1:end,1:num_l1/3) = zeros(num_l2/3*2,num_l1/3);
w_cancel(num_l2/3*2+1:end,num_l1/3+1:num_l1/3*2) = zeros(num_l2/3,num_l1/3);

%% Initializing weigth %

if random_initialization == 1
for i = 1:num_layer-1 eval("w"+num2str(i)+" = randn("+num2str(neuron_for_layer(i+1))+",
"+num2str(neuron_for_layer(i))+")*threshold*1.1;");
end
end w1 = w1.*w_cancel;

%% Loading network if needed %

if reload_network >0

load(folder_parameters+"net_parameters_"+num2str(reload_network)+".mat");

dt = net_parameters.dt;
threshold = net_parameters.threshold;
neuron_for_layer = net_parameters.neuron_for_layer;
```

```
current_sinking_time = net_parameters.current_sinking_time;
max_weigth = net_parameters.max_weigth;
learning_rate = net_parameters.learning_rate;
num_train_cyc = net_parameters.num_train_cyc;
mse_min = net_parameters.mse_min;
num_layer = length(neuron_for_layer);


load(folder_network+"network_"+num2str(reload_network)+".mat");

for i = 1:num_layer -1

eval("w"+num2str(i)+" = network.w"+num2str(i)+";");

end

end

%% Initializing counters %

if reload_network <=0 mse_errors = []; accuracy = []; else mse_errors = network.mse_errors;
accuracy = network.accuracy; end

%% Starting_training %

epoch_break_point = epoch_break_point_0;

for epoch = 1:1:max_epoch

%% resetting parameters %%loss function = mean squared error, at first cycle mse in
unknown mse = 0; de_dw1 = 0; de_dw2 = 0; de_dw3 = 0;

% counter of right classiifications of the network right_counter = 0;

%% Inference %

num_batch = 0;

for num_input = randperm(num_max_input)

output = layer4.spike*0;
num_batch = num_batch + 1;

for num_train = 1:num_train_cyc

input = spiking_data.axis(num_input,:)';
layer1.capacitance = layer1.capacitance + input;
layer1 = trigger_layer(layer1,threshold,dt,slope);
```

```
layer2 = inference(w1,layer1,layer2,current_sinking_time);
layer2 = trigger_layer(layer2,threshold,dt,slope);


layer3 = inference(w2,layer2,layer3,current_sinking_time);
layer3 = trigger_layer(layer3,threshold,dt,slope);


layer4 = inference(w3,layer3,layer4,current_sinking_time);
layer4 = trigger_layer(layer4,threshold,dt,slope);

%% Backpropagation %

e = spiking_data.labels(:,num_input) - layer4.spike;
d3 = e.*tanh_prime(w3*layer3.spike,threshold);
de_dw3 = de_dw3 + d3*(layer3.spike.');

d2 = sign(w3.'*d3).*tanh_prime(w2*layer2.spike,threshold);
%calculating de/dw as d_l(a_(l-1))^Tandsumminghispreviousvalue
de_dw2 = de_dw2 + d2 * (layer2.spike.');

d1 = sign(w2.'*d2).*tanh_prime(w1*layer1.spike,threshold);
de_dw1 = de_dw1 + d1*(layer1.spike.');

%% Parameters update %

mse = mse + sum((e).^(2))/2;

output = output + layer4.spike;

end

%% resetting capacitance %

layer1.capacitance = layer1.capacitance*0;
layer2.capacitance = layer2.capacitance*0;
layer3.capacitance = layer3.capacitance*0;
layer4.capacitance = layer4.capacitance*0;

%% Evaluating if there are more pulses on the right or wrong output %

right = sum(output.*spiking_data.labels(:,num_input));
wrong = sum(output.*flip(spiking_data.labels(:,num_input)));

if right > wrong
```

```
right_counter = right_counter + 1;

end

if num_batch == batch_size
%dividing de/dw for the number of cycles obtaining the mean
de_dw1 = de_dw1/(batch_size*num_train_cyc);
%weigth update
w1 = w1 + learning_rate*de_dw1.*w_cancel;
de_dw2 = de_dw2/(batch_size*num_train_cyc);
w2 = w2 + learning_rate*de_dw2;
de_dw3 = de_dw3/(batch_size*num_train_cyc);
w3 = w3 + learning_rate*de_dw3;

num_batch = 0;

de_dw1 = 0;
de_dw2 = 0;
de_dw3 = 0;


end

end

accuracy = cat(1,accuracy,right_counter/num_max_input);
mse = mse/(num_max_input*num_train_cyc);
mse_errors = cat(1,mse_errors,mse);


if max(accuracy) == right_counter/num_max_input

network.w1 = w1;
network.w2 = w2;
network.w3 = w3;
network.mse_errors = mse_errors;
network.accuracy = accuracy;


end

if (mse<mse_min)  (accuracy(end)==1)
disp("Leaning sucessfull");
```

```
break
end

end

figure(1)
plot(mse_errors,"Color","b");
hold on;
plot(accuracy,"Color","r");
legend("mse error","accuracy");

%% Save data %

num_network = length(dir(folder_network)) - 1;

net_parameters.dt = dt;
net_parameters.threshold = threshold;
net_parameters.neuron_for_layer = neuron_for_layer;
net_parameters.current_sinking_time = current_sinking_time;
net_parameters.max_weigth = max_weigth;
net_parameters.learning_rate = learning_rate;
net_parameters.num_train_cyc = num_train_cyc;
net_parameters.mse_min = mse_min;
save(folder_parameters+"net_parameters_"+num2str(num_network)+".mat",
"net_parameters");
save(folder_network+"network_"+num2str(num_network)+".mat","network");


num_network = length(dir(folder_network)) - 1;

network.w1 = w1;
network.w2 = w2;
network.w3 = w3;
network.mse_errors = mse_errors;
network.accuracy = accuracy;
save(folder_network+"network_"+num2str(num_network)+".mat","network");
save(folder_parameters+"net_parameters_"+num2str(num_network)+".mat",
"net_parameters");

function layer_post_pre_spike = inference(w,layer_pre,layer_post,current_sinking_time)

infer = w*layer_pre.spike;
```

```
update = double(layer_post.last_activation>=current_sinking_time);
infer = infer.*update;

layer_post_pre_spike.capacitance = infer + layer_post.capacitance;
layer_post_pre_spike.last_activation = layer_post.last_activation;
layer_post_pre_spike.spike = layer_post.spike;

end

function layer_update = trigger_layer(layer,threshold,dt,slope)

layer.last_activation = layer.last_activation + dt;
layer.capacitance = layer.capacitance + slope*dt;
layer.capacitance = layer.capacitance.*double(layer.capacitance>0);
spike_negat = double(layer.capacitance<threshold);
layer_update.capacitance = layer.capacitance.*spike_negat;
layer_update.last_activation = layer.last_activation.*spike_negat;


layer_update.spike = double(layer.capacitance>=threshold);

end

%derivative of the tanh
function f_prime = tanh_prime(vector,threshold)

f_prime = 1-tanh(vector/threshold-threshold).^2;

end

function f = heaviside(vector,threshold) %heaviside function

f = double(vector>=threshold);

end

%surrogate heaviside derivative function. Derivate of min(max(vector,0),max_weigth)
function f_prime = pseudo_heaviside_prime(vector,max_weigth,threshold)


f_prime = heaviside(vector,threshold) - heaviside(vector-max_weigth,threshold);

end
```

A.1.3. Network training with memory adapted algorithm

```
clear;close;
%% Network parameters %
random_initialization = 1;
dt = 80e-3;%[sec]
threshold = 0.2;%[V]
neuron_for_layer = [48,21,11,2];
current_sinking_time = 79.99e-3;%[sec]
max_weigth = 0.4; %[V]
max_epoch = 200;
epoch_break_point_0 = 25;

batch_size = 1000;
learning_rate = 0.5;
num_train_cyc = 11; %number of consecutive training for each input
threshold_spike = 3; %number of spike to get a 1 in backpropagation
mse_min = 0.1; %mse to be achieved to stop the loop
slope = -0.0541; %slope of the leakage

reload_network = 0;%choose the netwok you want to keep training

num_layer = length(neuron_for_layer);

folder = "";
folder_network = folder+"networks";
folder_parameters = folder+"network_parameters";

%% Loading input data %

load(folder+"spiking_data.mat");

num_max_input = length(spiking_data.axis(:,1));
spiking_data.axis = spiking_data.axis/max(max(spiking_data.axis))*(threshold);

for i = 1:num_layer
eval("layer"+num2str(i)+".spike = zeros(neuron_for_layer(i),1)*nan;");
eval("layer"+num2str(i)+".capacitance = zeros(neuron_for_layer(i),1);");
eval("layer"+num2str(i)+".last_activation =
ones(neuron_for_layer(i),1)*current_sinking_time;")
eval("layer"+num2str(i)+"_memory_spike = zeros(neuron_for_layer(i),1)*nan;");
end
```

```
%eliminating unused synaptic connections between input layer and first hidden layer
num_l1 = neuron_for_layer(1);
num_l2 = neuron_for_layer(2);
w_cancel = ones(num_l2,num_l1);

w_cancel(1:num_l2/3,num_l1/3+1:end) = zeros(num_l2/3,num_l1/3*2);
w_cancel(num_l2/3+1:num_l2/3*2,num_l1/3*2+1:end) = zeros(num_l2/3,num_l1/3);
w_cancel(num_l2/3+1:end,1:num_l1/3) = zeros(num_l2/3*2,num_l1/3);
w_cancel(num_l2/3*2+1:end,num_l1/3+1:num_l1/3*2) = zeros(num_l2/3,num_l1/3);
%% Initializing weight %

if random_initialization == 1
for i = 1:num_layer-1
eval("w"+num2str(i)+" = randn("+num2str(neuron_for_layer(i+1))+",
"+num2str(neuron_for_layer(i))+")*threshold*1.1;");
end
end w1 = w1.*w_cancel;

%% Loading network if needed %

if reload_network >0

load(folder_parameters+"net_parameters_"+num2str(reload_network)+".mat");

dt = net_parameters.dt;
threshold = net_parameters.threshold;
neuron_for_layer = net_parameters.neuron_for_layer;
current_sinking_time = net_parameters.current_sinking_time;
max_weigth = net_parameters.max_weigth;
learning_rate = net_parameters.learning_rate;
num_train_cyc = net_parameters.num_train_cyc;
mse_min = net_parameters.mse_min;
num_layer = length(neuron_for_layer);

load(folder_network+"network_"+num2str(reload_network)+".mat");

for i = 1:num_layer -1

eval("w"+num2str(i)+" = network.w"+num2str(i)+";");

end

end
```

```
%% Initializing counters %

if reload_network <=0
mse_errors = [];
accuracy = [];
else
mse_errors = network.mse_errors;
accuracy = network.accuracy;
end

%% Starting_training %

epoch_break_point = epoch_break_point_0;

for epoch = 1:1:max_epoch

%% resetting parameters %

mse = 0; %loss function = mean squared error, at first cycle mse in unknown
de_dw1 = 0;
de_dw2 = 0;
de_dw3 = 0;

right_counter = 0; % counter of right classifications of the network

%% Inference %

num_batch = 0;

for num_input = randperm(num_max_input)

output = layer4.spike*0; num_batch = num_batch + 1;

for num_train = 1:num_train_cyc

input = spiking_data.axis(num_input,:)';

layer1.capacitance = layer1.capacitance + input;
layer1 = trigger_layer(layer1,threshold,dt,slope);


layer1_memory_spike = layer1_memory_spike + layer1.spike;

layer2 = inference(w1,layer1,layer2,current_sinking_time);
layer2 = trigger_layer(layer2,threshold,dt,slope);

layer2_memory_spike = layer2_memory_spike + layer2.spike;
```

```
layer3 = inference(w2,layer2,layer3,current_sinking_time);
layer3 = trigger_layer(layer3,threshold,dt,slope);

layer3_memory_spike = layer3_memory_spike + layer3.spike;

layer4 = inference(w3,layer3,layer4,current_sinking_time);
layer4 = trigger_layer(layer4,threshold,dt,slope);

layer4_memory_spike = layer4_memory_spike + layer4.spike;

%% Parameters update %

output = output + layer4.spike;

end

%% Backpropagation %

layer1_memory_spike = layer1_memory_spike>=threshold_spike;
layer2_memory_spike = layer2_memory_spike>=threshold_spike;
layer3_memory_spike = layer3_memory_spike>=threshold_spike;
layer4_memory_spike = layer4_memory_spike>=threshold_spike;

e = spiking_data.labels(:,num_input) - layer4_memory_spike;

d3 = e.*tanh_prime(w3*layer3_memory_spike,threshold);
de_dw3 = de_dw3 + d3*(layer3_memory_spike.');

d2 = sign(w3.'*d3).*tanh_prime(w2*layer2_memory_spike,threshold);
%calculatind de/dw as d_l(a_(l-1))^T andsumminghispreviousvalue
de_dw2 = de_dw2 + d2 * (layer2_memory_spike.');

d1 = sign(w2.'*d2).*tanh_prime(w1*layer1_memory_spike,threshold);
de_dw1 = de_dw1 + d1*(layer1_memory_spike.');

mse = mse + sum((e).^(2))/2;

%% resetting capacitance %

layer1.capacitance = layer1.capacitance*0;
layer2.capacitance = layer2.capacitance*0;
layer3.capacitance = layer3.capacitance*0;
layer4.capacitance = layer4.capacitance*0;

%% Evaluating if there are more pulses on the right or wrong output %
```

```
right = sum(output.*spiking_data.labels(:,num_input));
wrong = sum(output.*flip(spiking_data.labels(:,num_input)));

if right > wrong

right_counter = right_counter + 1;

end

if num_batch == batch_size

%dividing de/dw for the number of cycles obtaining the mean
de_dw1 = de_dw1/(batch_size);
%weigth update
w1 = w1 + learning_rate*de_dw1.*w_cancel;
de_dw2 = de_dw2/(batch_size);
w2 = w2 + learning_rate*de_dw2;
de_dw3 = de_dw3/(batch_size);
w3 = w3 + learning_rate*de_dw3;

num_batch = 0;

de_dw1 = 0;
de_dw2 = 0;
de_dw3 = 0;

end

end

accuracy = cat(1,accuracy,right_counter/num_max_input);
mse = mse/(num_max_input*num_train_cyc);
mse_errors = cat(1,mse_errors,mse);

if max(accuracy) == right_counter/num_max_input

network.w1 = w1;
network.w2 = w2;
network.w3 = w3;
network.mse_errors = mse_errors;
network.accuracy = accuracy;

end

if (mse<mse_min)  (accuracy(end)==1)
```

```
disp("Leaning sucessfull");
break
end

end

figure(1)
plot(mse_errors,"Color","b");
hold on;
plot(accuracy,"Color","r");
legend("mse error","accuracy");

%% Save data %

num_network = length(dir(folder_network)) - 1;

net_parameters.dt = dt;
net_parameters.threshold = threshold;
net_parameters.neuron_for_layer = neuron_for_layer;
net_parameters.current_sinking_time = current_sinking_time;
net_parameters.max_weigth = max_weigth;
net_parameters.learning_rate = learning_rate;
net_parameters.num_train_cyc = num_train_cyc;
net_parameters.mse_min = mse_min;
save(folder_parameters+"net_parameters_"+num2str(num_network)+".mat","net_parameters");
save(folder_network+"network_"+num2str(num_network)+".mat","network");


num_network = length(dir(folder_network)) - 1;

network.w1 = w1;
network.w2 = w2;
network.w3 = w3;
network.mse_errors = mse_errors;
network.accuracy = accuracy;
save(folder_network+"network_"+num2str(num_network)+".mat","network");
save(folder_parameters+"net_parameters_"+num2str(num_network)+".mat","net_parameters");

function layer_post_pre_spike = inference(w,layer_pre,layer_post,current_sinking_time)

infer = w*layer_pre.spike;
update = double(layer_post.last_activation>=current_sinking_time);
```

```
infer = infer.*update;

layer_post_pre_spike.capacitance = infer + layer_post.capacitance;
layer_post_pre_spike.last_activation = layer_post.last_activation;
layer_post_pre_spike.spike = layer_post.spike;

end

function layer_update = trigger_layer(layer,threshold,dt,slope)

layer.last_activation = layer.last_activation + dt;
layer.capacitance = layer.capacitance + slope*dt;
layer.capacitance = layer.capacitance.*double(layer.capacitance>0);
spike_negat = double(layer.capacitance<threshold);
layer_update.capacitance = layer.capacitance.*spike_negat;
layer_update.last_activation = layer.last_activation.*spike_negat;


layer_update.spike = double(layer.capacitance>=threshold);

end

%derivative of the tanh
function f_prime = tanh_prime(vector,threshold)

f_prime = 1-tanh(vector/threshold-threshold).^2;

end
```

List of Figures

List of Tables

Ringraziamenti

Innanzitutto, metto le mani avanti. Premetto che la maggior parte di questi ringraziamenti sono stati scritti mentre ero fuori come un balcone per via delle notti insonne. Quindi chiedo gentilmente di avere pazienza nel caso mi fossi dimenticato qualcuno o qualcosa, siccome, purtroppo la situazione era quella e "the show must go on".

Innanzitutto, non potrei mai e poi mai ringraziare abbastanza il professor Giorgio Ferrari. Penso sia la persona più disponibile e dedita al lavoro che abbia mai conosciuto. Con una modesta media di 15 ore di lavoro al giorno, wekend inclusi, mi ha fornito la saggezza e l'aiuto necessari per concludere la mia tesi in tempo. Tutto questo mentre seguiva altri 10 progetti. Davvero, io non so come ringraziarla professore. Penso sinceramente che non avrei potuto avere un relatore migliore e che probabilmente il LabSamp non sarebbe quello che è senza di lei.

Volevo ringraziare il professor Ielmini con cui ho intrapreso il mio primo percorso di tesi. Grazie a lui ho imparato tanto sui dispositivi elettronici. Purtroppo, grazie a lui ho anche scoperto che l'arte della misura dei dispisitivi non fa per me e questo mi ha portato su un percorso diverso. La ringrazio comunque per quello che mi ha insegnato e il tempo che mi ha dedicato.

Ringrazio il professor Sampietro che mi ha insegnato l'arte dell'elettronica. Con il suo tono pacato e le sue lezioni interattive è riuscito a farmi piacere questa materia e a farmi sparire ogni dubbio sulla scelta del mio indirizzo di studi. Penso che il professor Sampietro sia stato il professore che mi abbia dato di più in assoluto. Non che gli altri professori non mi abbiano lasciato nulla, chiariamoci. Il fatto è che il corso del professore Sampietro mi è caro come ad un elettronico è caro il segnale. Il segnale è tutto per noi elettronici. é la parte nobile dello studio degli elettroni. Il principio ed il fine ultimo. Quindi ringrazio la persona che per prima ha insllato in me quella che oggi è la mia più grande passione.

Un ringraziamento speciale va a Paola, Vittorio, Giuseppe, Michele, Fabio e Francesco, sempre super disponibili e gentili. Non si sono mai permessi di rifiutare una mia richiesta di aiuto, per quanto fosse banale o superficiale. Insieme a loro, voglio ringraziare Federico, Gabri, Diego, Monica, Luca, Silvano, Emanuele, Cristina, Alex, Elisabetta, Riccardo,

Andrea e Giulio sono stati i miei preziosissimi compagni di percorso (l'ordine dei loro nomi e quello della precedente lista è stato ottenuto mettendoli in ordine alfabetico e generando un vettore di ugual lunghezza derivante da una distribuzione gaussiana, il quale è stata riordinato in ordine crescente e dal quale a sua volta è derivato l'ordine finale dei nomi (DISCLAIMER non vi è stata alcuna decisione o preferenza che determinasse l'ordine dei nomi)). Con cui ho condiviso questi mesi di duro lavoro alternato a pranzi, pause caffè e scale di Bristol. Hanno aggiunto ad ogni momento di duro lavoro un altro momento di schezi e spensieratezza. Non dimenticherò mai i ceci di Vittorio, la pugliesanza di Giulio, il pollice verde di Emanuele, i ritardi di Gabri, l'high power efficiency di Paola, le storie di Diego, la bici di Fede e il posto non mio, ma di Silvano. Mi dispiace di non essere stato presente in laboratorio quanto avrei voluto, ma so che vi porterò sempre nel cuore.

Un ringraziamento particolare va a Saverio, Matteo, Shahin e Alessandro, i miei compagni di misure. Con loro ho condiviso giornate indimenticabili a misurare device in uno sgabbuzzo con una temperatura media 40 gradi in inverno. Li stimo molto per il loro lavoro e li ringrazio per avermi insegnato l'arte della misura e a fare i grafici belli.

Un ringraziamento speciale va a Libero, Paolo, Francesca, Roberto, Andrea, Renata, Katerina, Katia, Cinzia e tutti i maestri e compagni di karate. Chi più, chi meno mi hanno insegnato cosa voglia dire avere un corpo e come usarlo. Sono stato fortunato ad aver fatto parte di una comunità così meravigliosa. Tanto volta al dovere quanto allo svago dopo un allenamento. Sono le persone che mi hanno mostrato che i supereroi esistono anche nella realtà e sono molto più forti e cazzuti di Batman (non c'è paragone). Li ringrazio perchè grazie a loro ho acquisito una confidenza in me stesso che non avrei mai potuto avere altrimenti. Un ringraziamento particolare va al Maestro Libero e a Francesca che mi hanno fatto riprendere (in realtà costretto, ma son dettagli) e un R.I.P per le dita di Robi.

Ringrazio Pino, Gabri, Fenzi, Stè, Mirko, Lore, Church, Manga e Marchetti. La mia churma del Politecnico. Coloro che non si sono mai fatti remore a distrarmi durante le lezioni per fare battute becere e che mi hanno sempre incoraggiato nello studio (falsissimo). Le nostre serate leggendare saranno tramandate di padre in figlio per generazioni. Serate leggendarie che in verità si concludono alla prima serata. Serata che è terminata con la mia perdizione nei più reconditi meandri di Lambrate, lo scippo di Pino, il coma etilico di Stè e la decisione di Marchetti di non rivolgerci mai più la parola dopo le 7 di sera. Mirko e Fenzi erano assenti, ma solo perchè sono asociali (Fenzi anche un po' psicopatico).

Ringrazio mia Mamma, mio Papà e Raffi e la mia famiglia (Stefi e Simona incluse). Mi hanno sempre sostenuto nelle difficoltà e nelle scelte della vita. Soprattuto rigrazio mia mamma Augusta, alla quale dedico questo lavoro di tesi. Vorrei scrivere un'infinità di aneddoti e ringrziamenti specifici, ma sarebbe inusuale scrivere più pagine di ringraziamenti che di tesi effettiva, quindi evito.

Devo ringraziare la mia ragazza Lucija. Non sarei la persona che sono oggi senza di lei. Mi ha sostenuto e dato forza e motivazione per finire l'università. Non so come faccia a sopportarmi o perchè sia felice di stare insieme ad un personaggio come me, ma gliene sono grato. Grazie per farmi svelgiare ogni giorno con il sorriso.

Volevo ringraziare Pando, Richi, Sere, Perri, Fisco e Claudio. I miei amici di lunga data che trovano sempre il modo di arrivare in ritardo o paccare all'ultimo. Non so come avrei fatto senza tutte le risate che ci siamo fatti in questi anni. Un ringraziamento speciale va a Pando e Richi che per fare una serata "diversa" a Bratto hanno rischiato di non farmi laureare.

Un ringraziamento particolare va a Giovanni, Paolo, Carlo, Lorenzo e Lara. Mi hanno riportato una spensieratezza che non avevo da tempo e che mi porterò per sempre con me. Soprattuto volevo ringraziare Gio. Sei un pazzo maniaco, ma mi hai permesso di fare cose che ho sempre desiderato, ma che non ho mai avuto l'ardore di fare. Conterò sempre su di te per i futuri progetti.

Un ringraziamento speciale a Ale e Iris, che non hanno alcun motivo di essere separati dal paragrafo precedente. Ma proprio nessuno nessuno. Però ero sicuro che avrebbe dato fastidio ad Ale quindi l'ho fatto.

Un ringraziamento particolare a Trenord. Che mi ha permesso di arrivare in ritardo al primo colloquio col professor Ferrari. Il loro servizio che fa arrivare i treni in ritardo 15 volte su 16 and in anticipo la restante volta è encomiabile. Hanno aumentato di due livelli la difficolta di questi 5 anni di studio. Putroppo, devo anche ammettere che mi hanno fatto vivere alcune delle mie migliori esperienze. Come quando ho perso la coincidenza per andare ad assistere alla mia prima conferenza dal vivo perchè il Milano centrale era in ritardo di 30 minuti. Ancora mi ricordo lo stupore negli occhi delle mie compagne di viaggio quando mi hanno visto salire alla stazione di Piacenza sul regionale diretto a Modena che avevo perso. Quel giorno ho scoperto quanto veloce possa essere un Freccia Argento. Specialmente se ti godi il viaggio dal bagno.

Un ringraziamento alla parrocchia di Fornovo San Giovanni e alle sue bellissime campane che mi hanno impedito di dormire per la maggior parte del periodo dei miei studi. Superando i 90dB di potenza acustica (misurazioni effettuate dal mio cuscino) si sono assicurati che la media di sonno dei miei weekend non superasse le 4 ore. Perché solo gli stolti si alzano dopo le 7 la domenica. Per non parlare della paranoia e agitazione che mi hanno causato e che mi ha permesso di dormire ancora di meno in settimana. Mi hanno insegnato che non bisogna mollare mai. Mi hanno insegnato che il buon udito, il sonno e la sanità mentale solo per i deboli, gli inetti, gli atei, i miscredenti e per chi vuole superare i cinquant'anni. Mi hanno insegnato che l'Italia è un paese laico solo di facciata. A quanto pare finché vivrò in questo paese i miei diritti fondamentali, teoricamente inalienabili, potranno essere calpestati da un concetto. Il che fa capire meglio la mentalità della maggior parte della popolazione del paese con minor fondi dediti alla ricerca in Europa. Nonché patria del nepotismo e della Mafia. Il paese in cui sono nato, che amo e rispetto con tutto me stesso. Grazie per tutte queste nozioni fondamentali.

Ringrazio i Bergamaschi per la loro remarcabile dedizione al lavoro e i Milanesi per la loro insaziabile voglia di Spritz. Senza non saremmo mai riusciti ad essere i primi al mondo ad andare in lockdown e di conseguenza non sarei mai riuscito a recuperare 50 crediti tra giugno-settembre 2020. Ovviamente non ci sono riuscito stesso, ma tentare 50 crediti in 2 settimane e mezzo a luglio è stata un'esperienza che sono sicuro in pochi se ne possano vantare.

Ringrazio Adele, che è riuscita a spezzarmi il cuore in uno dei momenti di fragilità e debolezza maggiori nella mia vita (che significa nello stesso semestre di Analog design) e a dividere (per poco tempo) un gruppo di amici incredibile. La ringrazio perché se non fosse successo non sarei neanche minimamente la persona che sono oggi. Per molte persone l'uscita dall'adolescenza corrisponde alla crescita della barba o dei peli sul petto, per me è stata quella fatidica notte. Quell'esperienza mi ha fatto crescere e maturare come nient'altro nella mia vita. Mi ha anche permesso di conoscere la persona che adesso è più importante e cara per me in assoluto. Grazie XP.

Ringrazio il professor Lacaita per motivi che non posso citare perché, non laureandomi in giurisprudenza, non ho la più pallida idea di quanto denunciabile possa essere. Dico solo che è stata una grande esperienza formativa, concentratasi al termine di 50 fatidici crediti fatti in due settimane e mezzo. La cigliegina sulla torta insomma.

Ringrazio Putin che ha deciso di incominciare un percorso di genocidio di massa proprio in questo periodo e avermi fatto perdere ogni speranza nell'umanità.

La professoressa MM. il quale accanimento, quasi morboso, mi ha permesso di avere una pubertà orribile e di rovinare per sempre la mia psiche e sanità mentale. Ma no, scherzo. Il mio periodo adolescenziale è stato spaziale. Miglior periodo della mia vita. La ringrazio perché senza di lei non sarei mai riuscito a scrivere la tesi o questi fantastici ringraziamenti. Essendo lei ad avermi effettivamente insegnato a scrivere.

E infine ringrazio te caro lettore. Non so chi tu sia, ma dopo più di 100 pagine di tesi più ringraziamenti direi che posso permettermi questa chicca. Tu che hai perso tempo per leggere un lavoro che probabilmente risulterà inutile e perso negli annali del Politecnico. Probabilmente essendo dopo il ringraziamento a Putin ti potresti sentire un po' screditato e di marginale importanza per me, ma non è vero. Senza di te questo lavoro non avrebbe senso alcuno o ragione di esistere. Un libro senza lettore è come un transistor senza ossido. Un doppio diodo con giunzione p-n-p o n-p-n. Bello, ma sostanzialmente inutile. A meno che tu non sia un genio. In quel caso hai un BJT e un nobel, ma non era questo il punto. Il fatto è che sei la persona che più ho pensato, adorato e odiato durante la stesura della tesi. Quindi direi di sostanziale importanza.
Ho giusto due parole da dirti. Se sei il mio successore nel progetto, spero di non averti annoiato troppo. So che è lunga e non troppo utile, ma ho fatto del mio meglio con quello che avevo e nelle condizioni in cui mi trovavo. Fidati che è pesato più a me che a te. Preparati perché l'ultimo mese di tesi non è neanche paragonabile alla sessione. Soprattutto, se ci tieni e vuoi laurearti in tempi decenti. Vedi, inoltre, di stare attento e non fare casini, anche se sotto l'ala di Giorgio è difficile e dovresti proprio impegnarti. Se sei un ricercatore ti ringrazio ancora di più perché non avrei mai pensato che questo lavoro potesse effettivamente interessare a qualcuno fuori dal SampLab. Mi sento estremamente onorato. Infine se non sei ne un ricercatore ne uno studente del DEIB... Beh... Probabilmente dovresti trovarti un hobby.

Grazie. Grazie davvero davvero a tutti, senza di voi non sarei la persona che sono oggi.