Politecnico di Torino

Master Thesis

Nanotechnologies For ICTs

A.y. 2021/2022

Single neuron SNN with Memristor Generated Delays for Real-Time Analysis of Temporal Signals

Supervisors

Prof. Carlo Ricciardi

Dr. Melika Payvand

Candidate

D'Agostino Simone

Abstract

The most powerful computing machine for doing classification with low energy consumption is the brain with its known and unknown behaviors and working principles.

Partially inspired to the brain, neural networks established a paradigm in which the computational power is increased with a trade-off in terms of accuracy and power consumption. Indeed, in such a paradigm, neurons, synapses and the architectures deriving from their combination are used for solving non-linear classification tasks with the objective to reach the highest possible accuracy without any regard to power consumption.

Neuromorphic computing aim is to solve the energy-accuracy trade-off by using all the possible knowledge from biology in order to define a scientific paradigm in which the biology knowledge is not only mathematically represented, but also implemented on-chip for taking advantage from Silicon technology higher efficiency in terms of electrical behavior. Moreover, thanks to the rise of memristor technology, the integrability and scalability of such devices increased in the last years following the energy efficiency trend.

Such chips are based on the so-called spiking neural networks, i.e. neural networks which are quasi-totally inspired by brain mechanism from the signal encoding into spikes -as the name suggests- to neuron and synapse models arriving up to, in certain cases, the network structure. These networks allow to combine deep learning approach for achieving high accuracy results with ultra-low power consumption thanks to brain inspiration and analog electronics circuits.

In this thesis a novel architecture based on multi-synapse on dendrites connections -inspired from novel biological discoveries- exploiting temporal delays generated through memristors is presented for doing ultra-low power real-time classification of time-varying signals. Its results on MIT-BIH ECG dataset obtained through hardware-aware Python-based simulations are presented as a proof of concept of the effective working of the network showing how large memristor employment can allow to reach lower energy consumption and higher scalability while reaching very high accuracy results.

Acknowledgments

The first acknowledgment should go to all the people that accompanied me towards my graduation. My colleagues in the bachelor and master programs that shared studies and exams difficulties, my colleagues and players when coaching basketball that taught me important values such as punctuality and perseverance, my tutors during bachelor internship Ettore and Paolo for having given me a mindset in solving problem no matter how hard they are and to my professors able to transmit passions and competences even during COVID hard times.

Then, many thanks should go to Giacomo's group at Institute of Neuroinformatics for having received me as a part of the big family that they are. In particular, Melika guided me like no other supervisor could have done; Arianna, Matteo, Mattia and Yigit for having always time to talk about common problems and to give me interesting inputs.

Anyway, the most of my gratitude should go to my group of friend "Gli Squali" (The Sharks) for always building a stimulating environment for my personal growth, to my uncles Luca and Anna and my little cousin Mia for having hosted my like a son and a brother, to my girlfriend Martina who always stood by me giving all the possible love and, finally, to my family Eugenio, Laura, Marco, Elena and my grandparents that supported me not only economically but gave me education, possibilities and inspiration even if not living in the best of countries.

Contents

What I cannot create, I do not understand.
Richard Feynman, last words on his blackboard

1. Introduction

Solving non-linear problems, classifying data, recognizing hidden patterns are some of the skills required to new technologies for solving everyday tasks in medicine, engineering, economics and other fields of study.

At the same time, power efficiency become more and more an essential quality for finding environment friendly technologies.

In this section, brain inspired models will be presented as a possible solution to these requirements with their connection to the brain working principles and the improvement of bio-chemical mechanisms that can be found in nature.

1.1. Reasons

In order to solve computing tasks on large amount of data which are not solvable through algorithms, i.e. problems of which solutions are still not known and based on non-linear relations, from the 1940s on, some attempts to project and realize machines able to mimic and improve brain ability to adapt and learn based on what is called Hebbian learning[62] were done using early computers[47].

Among years, in order to reach the brain capability and to accomplish the task difficulties, different models, architectures and approaches were tried starting from the early perceptron[70] which is actually considered the first artificial neural network. In fact, the paper about was opening with three questions:

- How is information about the physical world sensed, or detected, by the biological system?
- In what form is information stored, or remembered?
- How does information contained in storage, or in memory, influence recognition and behavior?

It could be said that in these three questions are expressed all the challenges which researchers in the field of neural networks are nowadays facing.

Moreover, it is possible to find the reasons that lead to the developing of these technologies: knowledge and utility.

The knowledge is well explained by the first two questions and can be summarized in "how does the brain work?". Indeed, despite good progresses in medical research through centuries about the body behavior, the brain has always been -and in part still is- a black box which makes the body properly working and it is able to process information and data to create abstractions. Moreover, there was and there is the problem of interaction with the external world: how does the brain is capable to treat all kind of data, being them varying in space or in time or in both, in the same way?

On the other hand, the third question adds a second layer to the argument: once that the brain has the possibility to collect and store data, how does it use them? This is the utility reason, i.e. the aim becomes to project something capable of collecting, store and use new data of which the correlation and the solution is not know. The need is a machine which can learn and can recognize new patterns instead of only predetermined patterns.

All this must compete with an with the incredible energy efficiency of the brain[37].

1.2. Neuromorphic engineering and brain working principles

Neuromorphic computing is relatively new branch in the technology research in which the aim is to solve real world task while mimicking the brain and nervous system dynamics made of mixing analog and digital components, distributed representations, in-memory computation, self-organization, learning and adaptation[58].

The basic assumption is that the brain takes information from the outside world through its natural sensors such as eyes, ears, nose and so on, process this analog data converting them into a general representation and then performs computation on them.

Since the beginning there is the problem to deal with such a general purpose data representation: an encoding of data which allows to store and treat all the analog signals which can be collected in the same way from a circuit -the brain or the neuromorphic one- which uses the same elements -neurons and synapses- for analyzing them all.

Then, there is the computation problem: once that analog signals from the world are encoded, how does the brain extract information and how does it learn?

The brain is, in fact, a neural network composed by biological elements such as the neurons, the synapses and the dendrites; since first researches on the neuron structure and its role in the brain[51], it is clear that it is a cell composed by a cell body (or soma) connected to an axon along which it transmits electrical signals and to dendrites from which it receives electrical signals from other neurons [Fig. 1.1]; both the cell body and the dendrites are covered with synapses connected to axons from other neurons, these synapses have the role to act as doors for incoming signals.

From electrical pulses across nerves, ions channels open and close sending to the neuron an electrical pulse, each electrical pulse generate a slight change in the potential of the cell's membrane. Once that the difference between the inside and the outside of the cell is high enough, the neuron will send an electrical pulse through the axon. During all the time, synapses are receiving neurotransmitters which can be considered as quanta of neural signals, i.e. carriers for the pulses inside the brain[41].

This is the dance inside the brain which allows animals and humans to interact with the external world by classifying, analyzing and filtering incoming analog output signals by using from 1 billion to 100 billions neurons depending on the species[41] which are interconnected in thousands different ways (a neuron in vertebrate cortex can connect to more than 10^4 postsynaptic neurons[50]).

The final aim of neuromorphic engineering is to take advantage of brain mechanisms in order to improve them for conserving the energy efficiency while improving computational performances. This is reached through two steps which are still the main goals in the

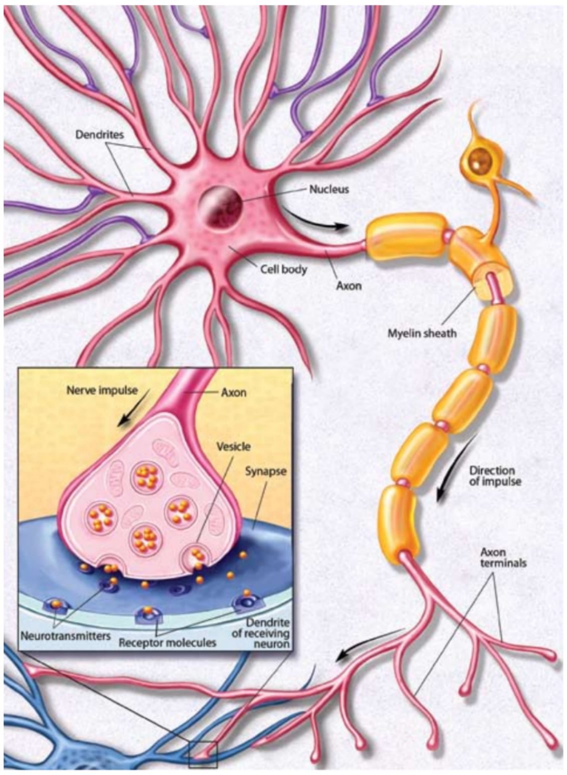


Figure 1.1: Neuron anatomy including the single synapse representation. Image from [41]

research fields of neural networks, neuromorphic computing and machine learning:

- modeling;
- "technological" improvements.

Indeed, the modeling is what allows to keep the energy efficiency of the brain: a proper encoding, the structure and function of each element in the circuit and the computational paradigm could allow to reach energy efficiency in neuromorphic computing.

Then, the change of technology should be intended as the switch from organic and poor performing electrical elements and materials in brain to Si, metals and derived elements allowing to increase significantly the precision and the performance of the neural network out coming from these changes.

Before proceeding with the explanation of these improvements it is important to understand better the role of each component of the brain circuit in order to reach a good modeling. Even if this remains a hard goal for neuroscience, some principal mechanisms are clear and can be explained.

1.2.1. Neuron

The neuron is the core of the computation processes in the neural network constituting the brain and acts as an integrator of electrical pulses coming from other neurons or from inputs. Usually, it is referred as the *central process unit* of the brain[50].

Its most important characteristic is the membrane potential which is defined as the difference in charge between the inside and the outside of the body cell[41] given by concentrations of ions Ca^{2+}, Cl^-, K^+, Na^+ and A^-[65]. In order to compute the potential, it is possible to consider ion energy $E(x)$ as the energy of a particle with charge q in an electric field whose potential is $u(x)$:

$$E(x) = q \cdot u(x)$$

since the probability that the ion will take a state with energy E is proportional to the Boltzmann relation, it easily writable that:

$$n(x) = \exp\left[\frac{q \cdot u(x)}{k_B T}\right]$$

where $n(x)$ is the ion density and it is given by the fact that the number of ions is huge[50].

Considering two points x_1 and x_2 inside and outside the body cell, the potential between the two points is obtained by inverting the relation between $n(x_i) = n_i$ and $u(x_1) - u(x_2) = u$ obtaining the Nernst potential:

$$u = \frac{kT}{q} ln\left(\frac{n_1}{n_2}\right) \tag{1.1}$$

At rest, the neuron is characterized by a so-called resting potential which is negative and corresponds to $u_{rest} = -65\,\text{mV}$. When it is feed with an input pulse, the membrane potential can increase or decrease depending on the synapse -which could be excitatory or inhibitory- before starting to decay back to the rest potential.

The decay could be easily explained by the fact that ions move to bring back the equilibrium in the system moving under the field generated by their displacement[50]. In this case the Hodgkin and Huxley model[57] shows how the semi-permeable membrane can be treated as a capitor which is charged on only one side when a current is injected inside the soma.

If all the presynaptic input are considered, a single neuron is receiving ions -i.e. voltage pulses- and summing them to reach a resulting membrane potential u. E.g. a neuron connected to j presynaptic neurons which send pulses at times $t_j^{(f)}$, where f represent the location in time of the pulse from presynaptic neuron j, the resulting potential at time t will be:

$$u(t) = \sum_j \sum_f \epsilon_j(t - t_j^{(f)}) - u_{rest} \tag{1.2}$$

where ϵ_j is the postsynaptic potential caused by the presynaptic neuron j.

If the membrane potential u overcomes a critical threshold ϑ -which is in the range $[20, 30]\,\text{mV}$- the neuron will emit a pulse of amplitude equal to $100\,\text{mv}$ which has the particular shape of a spike[50]. This is the so-called *action potential* and it is sent to neurons which are considered postsynaptic with respect to the considered one.

Finally, before going back to the resting voltage, the neuron passes through a phase of hyperpolarization in which the potential is lower than u_{rest}.

1.2.2. Dendrites and axon

Dendrites and axons are the wires which bring pulses to the neuron and the movement of charges along them is due to ions generated potentials.

The problem for studying this kind of effect is called *electrodiffusivity* and is very complicated to both study and compute.

Preliminary studies uses cable theory to model these mechanisms[34], but they are not including the extracellular potential which is important to study the network activity.

This is why, in order to properly model the potential, it is necessary to use more complicated models as the Poisson-Nernst-Planck Equations where the movement of ions in a static solvent as a consequence of drift-diffusion is used[69].

1.2.3. Synapses

Synapses are the contact point between an end of a presynaptic neuron's axon and a dendrite or the soma of a postsynaptic neuron.

Chemical synapses work thanks to neurotransmitters which are released in presence of an incoming pulse and, after diffusing across the intrasynaptic space -or cleft-, reach the specific receptors on the target cell[41]. As a key-lock system, the neurotransmitters open specific channels in the membrane which allows ions to enter the cell causing membrane potential variation following (1.1) and (1.2) [Fig. 1.1].

Electrical synapses or gap-junction can also be found, but not too much is known about them[50].

Synapses are a very -maybe the most- important element in brain neural since, thanks to their plasticity, they are responsible of the weight associated to the interaction between two neurons which determines learning and memory.

There are mainly two categories of plasticity:

- short-term plasticity: divided into potentiation (STP) and depression (STD);
- long-term plasticity: divided into potentiation (LTP) and depression(LTD).

Short-term synaptic plasticity

Short-term synaptic plasticity is responsible to the short term memory and adaptation to input signals; it is very frequent and observed virtually in every synapse observed in organism from the simplest to the most evolved[44]. Different short-term plasticity mechanisms have been observed:

- paired-pulse facilitation and depression: depending on the response to the first stimuli if a second one is sent to the neuron in a short time window ($< 20\,\mathrm{ms}$ or $\in [20, 500]\,\mathrm{ms}$); it is very likely due to residual Ca^{2+} ions left by the first stimuli; normally, synapses can show both paired-pulse potentiation or depression depending on recent history;
- potentiation and depression following train of stimuli: thanks to the prolongation of the sequence of input stimuli the longing of the effect to $[0.2, 5]\,\mathrm{s}$;
- modulation of reception by presynaptic receptors: by the release of neuromodulators postsynaptic neuron can influence the release of neurotransmitters;

- Involvement of glia: glia cells can influence short-term plasticity by controlling the speed of neurotransmitters in their clearance or by releasing substances that can influence synaptic efficiency;

Long-term synaptic plasticity

Long-term synaptic plasticity is depending on experience and is a long-lasting influence on the weight that different presynaptic events have on a specified neuron. It was hypothesized more that 100 years ago by S. Ramon y Cajal[44] and furthermore theorized by D. O. Hebb[55].

Despite in [55] there is not a mathematical model -which is essential to develop a scientific theory- of the hypothesis, it is possible to express it in such a way[49].

On the presynaptic side, the learning can be described in function of firing rate or spike time arrival while, on postsynaptic side the neuron can be described as usual in terms of membrane potential, firing rate or backpropagating action potentials (BPAPs).

Starting from the rate-based mechanism, it should start from the description of the activity of a neuron i, which is given by a non-linear function g correlating firing rate ν_i to membrane potential u_i:

$$\nu_i = g(u_i)$$

in such a description the activity of the membrane potential is given by the weighted sum of the presynaptic activity of neurons j:

$$u_i = \sum_j w_{ij}\nu_j$$

since the weight w_{ij} between neurons i and j should increase if they are both active together, six aspects become important: locality, cooperativity, synaptic depression, boundedness, competition and long-term stability.

If the locality, i.e. weight between neurons i and j should not be influenced by other neurons activity, imposes to write the equation of learning as:

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = F(w_{ij}, \nu_i, \nu_j) \tag{1.3}$$

the other five aspects can induce to write such an unknown F as an expansion considering firing rates low enough to be considered stationary, i.e. around $\nu_i = \nu_j = 0$:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} \approx & c_2^{\mathrm{corr}}(w_{ij})\nu_i\nu_j + c_2^{\mathrm{post}}(w_{ij})\nu_i^2 + c_1^{\mathrm{pre}}(w_{ij})\nu_j^2 + \\ & c_1^{\mathrm{pre}}(w_{ij})\nu_i + c_2^{\mathrm{post}}(w_{ij})\nu_j + c_0(w_{ij}) + \mathcal{O}(\nu)\end{aligned} \tag{1.4}$$

where posing all terms equal to zero except $c_2^{\mathrm{corr}}(w_{ij})$ would give the simplest Hebbian learning rule (anti-Hebbian if the parameter is negative).

In order to include also depression to the model, it is possible to use $c_0(w_{ij})$ in combination with $c_1^{\mathrm{pre}}(w_{ij})$ and $c_1^{\mathrm{post}}(w_{ij})$.

Furthermore, the spike-based mechanism can be seen as a generalization of (1.4). Considering the simplest spiking neuron model, i.e. the leaky integrate-and-fire (LIF) neuron, the equation (1.3) derived from locality can be rewritten in terms of membrane potential:

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = F[w_{ij}(t), u_i^{\mathrm{post}}(t'), u_j^{\mathrm{pre}}(t'')] \tag{1.5}$$

where $t' < t$ and $t'' < t$ indicate that weight changes depend also on the history of the potentials.

Notice that, in more complex models, the local Ca^{2+} concentration is influencing F, but it is possible to assume that the Ca^{2+} concentration depends on the previous firing history so that no additional variable should be accounted for[49].

If, for simplicity, $u_{rest} = 0$ and (1.5) is expanded around $u_i^{\text{post}} = u_j^{\text{pre}} = u_{rest}$:

$$\begin{aligned}\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} =& c_0(w_{ij}) + \int_0^\infty \alpha_1^{\text{pre}}(w_{ij}, s) u_j^{\text{pre}}(t-s)\,\mathrm{d}s + \\ &\int_0^\infty \alpha_1^{\text{post}}(w_{ij}, s') u_i^{\text{post}}(t-s')\,\mathrm{d}s' + \\ &\int_0^\infty \int_0^\infty \alpha_2^{\text{corr}}(w_{ij}, s, s') u_j^{\text{pre}}(t-s) u_i^{\text{post}}(t-s')\,\mathrm{d}s'\mathrm{d}s + ...\end{aligned} \tag{1.6}$$

the considerations did for $c_2^{\text{corr}}(w_ij)$, $c_0(w_{ij})$ in (1.4) are still valid for corresponding terms in (1.6).

Synaptic metaplasticity

Finally, there is metaplasticity[1] which is the plasticity of the plasticity. Indeed, it comprises mechanisms which makes LTD or LTP more difficult by changing the thresholds for this mechanisms.

1.3. Neural networks

The main objective for neural networks is to implement brain-like systems where the computation is obtained using connections between artificial neurons through synapses.

In order to collocate them, should be clear that in the field of Artificial Intelligence (AI), in the field of Machine Learning (ML), Neural Networks can be classified in the field of brain inspired ML[24].

Among years, a lot of different architectures and structures have been used for solving a vast amount of tasks, but they can be grouped into two main categories:

- Artificial Neural Networks (ANN);
- Spiking Neural Networks (SNN);

these two categories share some basis and differentiate for others, as they do with respect to the brain.

In such models made of interconnected nodes, the synapses are represented by weights and the neurons are the nodes and represented by a function which takes multiple inputs and gives back a single output. Since this first -very simple- definition of neural networks, are evident the problem posed in the previous section: modeling and technology.

The simplest network is the perceptron [Fig. 1.2a], i.e. a feedforward network with only one neuron which takes input from outside sensors and process it following the equation:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

where $\mathbf{x}$ is the input vector containing the signals from outside sensors, $\mathbf{w}$ is the weight vector in which each component is used to weight an outside sensor output, $f(\mathbf{x})$ is the output of the neuron. Since the output can be 0 or 1, the networks is called *binary classifier*. Here the model of the neuron is given by a weighted sum of its inputs and the input can be encoded into any numerical representation.

Basing on this very simple architecture, feedforward networks started to be developed: neurons are nodes of the network, the signal travels through one direction and it can contain zero or more hidden layers [Fig. 1.2b].

Then, among years, much more complicated architectures such as recurrent NN (RNN), convolutional NN (CNN), time delay NN (TDNN) and many others were presented.

It is is important to always keep in mind what are the main problems and goals when projecting a NN: technology and modeling.

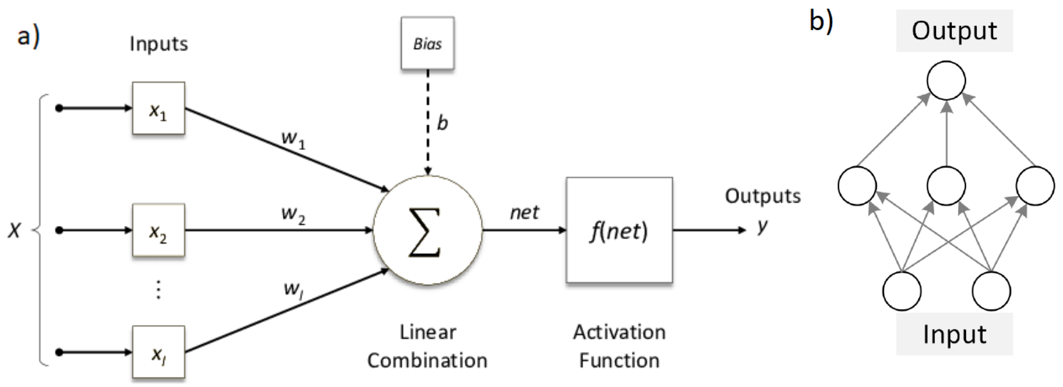


Figure 1.2: a) structure of the perceptron modeled with machine learning standards. Image from [3]; b) structure of a feedforward neural network with one hidden layer. Image from [15];

1.3.1. Artificial Neural Network

Artificial Neural Networks (ANNs) are nowadays the most used and studied neural networks. Being mainly software based, they allow for much more freedom in selecting models when projecting.

Technology

The technology used more frequently for ANN is the classical CPU-GPU-RAM system that can be found on servers or PCs rather than Field-Programmable Gate Array (FPGA) or Application-Specific Integrated Circuits (ASICs) used in order to speed up the computations[72].

The software are implemented using specific libraries such as PyTorch, TensorFlow, Keras, etc. which are normally written using Python or C++;

Modeling

For what concerns the modeling, thanks to the software-based architecture of this NN, it is possible to use a large amount of way to model and combine various components.

Starting from the input signals, they can be encoded in all possible ways allowed from bit-representation and, even if some standard ways emerged among years, the chosen encoding system can influence the quality of results[17].

The neuron model can be generalized using the expression:

$$f(\mathbf{x}(t_k)) = F\left(\sum_i w_i(t_k) \cdot x_i(t_k) + b\right) \tag{1.8}$$

it is important to notice that the base for the model is the perceptron equation (1.7) to which a transfer function F is added. The possibility to choose any mathematical function as transfer function[61] is the great advantage of this kind of NN since it is possible to adapt it to the problem that has to be solved. E.g., the perceptron has the Heaviside function as transfer function centered in a value which will define the threshold of the neuron ϑ (in (1.7) is $\vartheta = 0$).

Synapses are represented by weights as in the hebbian theory, while the dendrites and axons are collapsed into synapses represented through lines in graphic representations.

As said before, when one or more neurons are connected, a NN is formed and the model used for the structure -architecture- is variable and decided starting on the task that the network has to solve. All architectures can be grouped in categories:

- feedforward networks [Fig. 1.3a]: there is no limit on the number of layers, but the condition is that information must flow from input to output without any backloop;
- recurrent ANN [Fig. 1.3b]: there are no limits to backloop, these guarantee a dynamic temporal behavior and process time-varying signals; all other categories are special cases of RNN;
- Hopfield ANN [Fig. 1.3c]: particular backloop are used to store stable target vectors so that the NN can recall them when feed with similar input; no unit has connections with itself and the connections are symmetric, i.e. $w_{ij} = w_{ji}$;
- Elman and Jordan ANN [Fig. 1.3d and 1.3e]: are simple three layer RNN in which only the hidden layer (Elman) or only the output layer (Jordan) have a recurrent connection with input layer;
- long short term memory [Fig. 1.3f]: are capable of remembering an input of any time interval needed and this is why they outperform simple RNN;
- bi-directional ANN [Fig. 1.3g]: consist of two individual RNN which do not predict only future steps in time series but also past steps; they need two learning phases and used to predict complex timeseries;
- self-organizing map [Fig. 1.3h]: are related to feedforward and commonly arranged in hexagonal or rectangular shape, they uses unsupervised learning paradigm to represent high-dimensional data in low-dimensional representations so that are useful for this kind of dataset and tasks;

- convolutional NN (CNN): the architecture consists of two main parts which are features extractors and the classifier; the feature extractors are then divided into convolution and max-pooling layers and the output is represented on a 2D plane called feature mapping; it is highly optimized for processing 2D and 3D images;

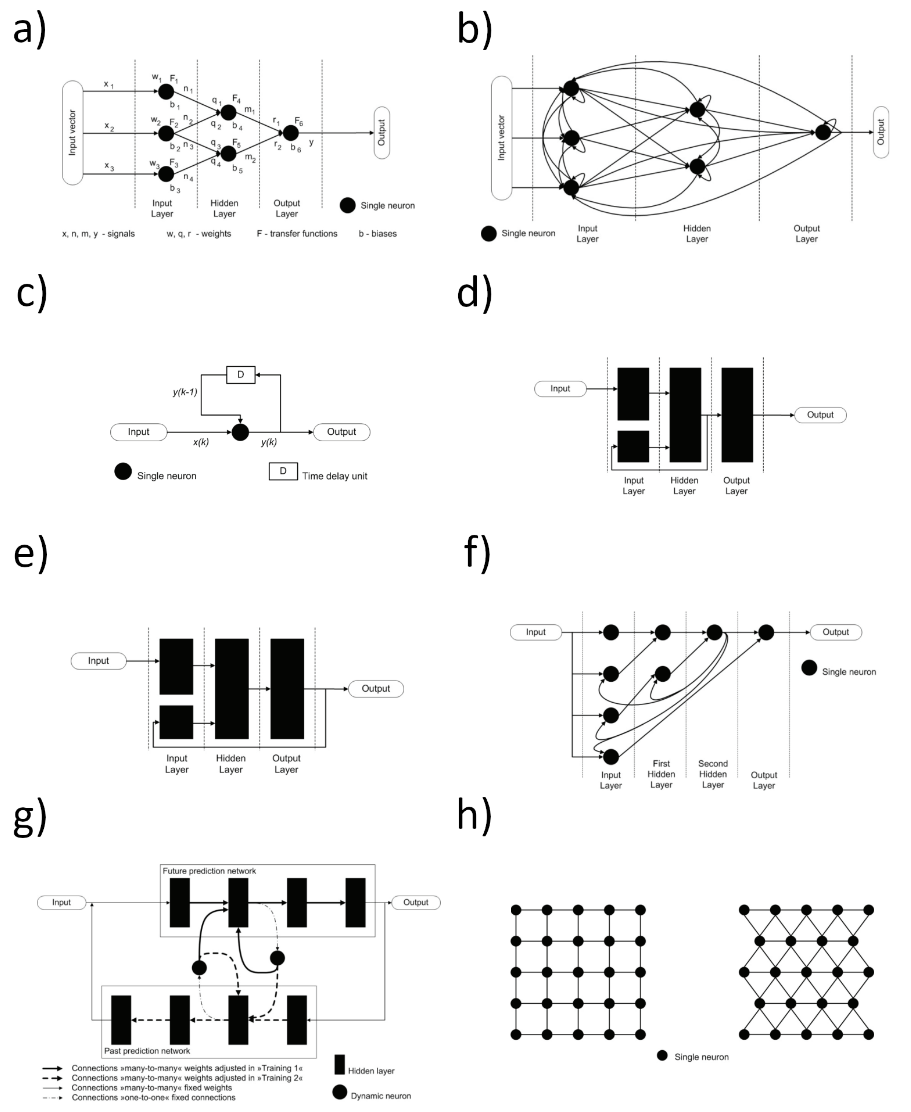


Figure 1.3: a) feedforward ANN; b) recurrent ANN; c) Hopfield ANN; d) Elman ANN; e) Jordan ANN; f) long short term memory ANN; g) bi-directional ANNs; h) self-organization maps: rectangular and hexagonal. Images from [61]

The last brain characteristic that has to modeled is the learning, i.e. the treatment of synapse plasticity. In order to modify the weights during the training phase of the ANN, four different paradigms have been established[24]:

- supervised learning: makes use of labeled data which are shown during each epoch to the NN in order to see what is the predicted value $\hat{y}_k = f(x(t_k))$ and how to modify network parameters to decrease the distance between the label of the data $y_k = y(t_k)$ and the predicted value. Such a distance is evaluated by means of a so-called loss function $l(y_k, \hat{y}_k)$. The objective of modifying the parameters of the network by decreasing the loss function is achieved with the stochastic gradient descendent method coupled with the back-propagation algorithm. Some important parameters in this paradigm are defined: momentum, learning rate and weight decay; all of them have the goal to increase the speed of training, reach the minimum of the loss function and to prevent the overfitting of the network, i.e. to have a good generalization from the training phase;

- semi-supervised learning: makes advantage of partially labeled dataset;
- unsupervised learning: uses unlabeled data in order to make the network learning important hidden features inside input data and often clustering, dimensionality reduction and generative techniques are exploited;
- reinforcement learning: is used to teach the network to interact with unknown environments. From the sampling x_k from the environment and prediction $\hat{y}_k = f(x_k)$ from the network, a cost function $c_k \sim P(c_k|x_k, \hat{y}_k)$ where the probability function P is unknown. Since the cost function is not totally known, the training is much harder, but necessary for unknown environment.

The most used paradigm is, nowadays, the supervised learning whose main interest components for connecting it to the brain is the back-propagation, i.e. the way which is used to combine the loss function minimization with the weights update. Once that the loss function has been computed, the weight update is established as:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta f_i(\mathbf{x}) \delta_j \tag{1.9}$$

where δ_j can be computed recursively using the chain rule and s obtained from all the δ_n in the layer above; $f_i(\mathbf{x})$ is the output of neuron i.

There is no evidence of bio-plausibility of back-propagation, even if NN trained with such a method can account for observed neural response, so the hypothesis that brain uses back-prop-like methods to learn is not to be excluded a priori[32].

1.3.2. Spiking Neural Networks

Here an overview will be given and then explained in details in section 3.

In the field brain inspired ML there is the sub-field of Spiking Neural Networks[24] (SNNs) which is including a big quantity of different approaches to brain inspired NNs spacing from software to hardware with a common input encoding; indeed, spikes are used instead of bit-represented numbers as for ANNs.

Technology

On the technology side, a large quantity of chip have been presented among years for implementing SNNs such as DYNAPs[29], Loihi[21], TrueNorth, etc. supported by PyTorch hardware-aware codes[75] and dedicated libraries such as Brian2[52] or similar.

Modeling

On the modeling side, the input for all these kinds of NNs is modeled through train of spikes, i.e. train of voltage pulses which are weighted and sent to the neuron before being summed. The method for encoding signals into spikes is still an open challenge and models ranging from bio-inspired systems like the cochlea for audio signals[5] to more complex and general models like number representations through train of spikes[73].

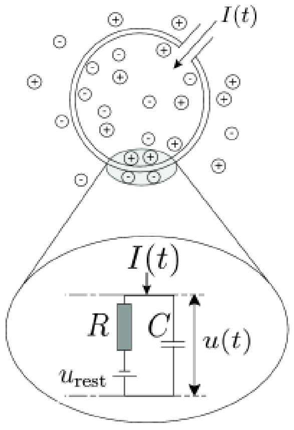


Figure 1.4: Integrate and fire neuron model related to ions movements in the body cell of the neuron

The neuron can be modeled depending on the implementation that should be done of the SNNs starting from the decision between hardware and software. The most elementary and common class of models is the so-called *integrate-and-fire* in which the membrane is modeled as an RC circuit with a voltage supply at u_{rest} an the voltage potential u corresponds to the voltage dropping on the capacitor [Fig. 1.4], but more complicated and functional will be analyzed.

The synapses still have the weight interpretation as in the ANN case, but can be implemented in different ways: from the simple number representation in software implementations to active circuits for synaptic dynamics and spike-based plasticity emulation for short term plasticity[7] to passive circuits for long term plasticity through the use of memristor devices.

Finally, learning policies for SNNs are defined depending on the architecture and the aim of the network. This phase can be done online or offline, on chip or out of chip, taking into account the synapse that has to be implemented, raising to several methods[15, 35]:

- shadow training: an ANN is trained and then weights are transferred to the SNN;
- back-propagation on SNN: can be done by using hardware-aware codes;
- local learning rules: weights updates are functions of signals that are spatially and temporally local;
- mixed precision update: used for taking into account memristors non-idealities and perform on-chip learning;

2. Memristive Technologies

Starting from the late 1970s when L. O. Chua[43] theorized the memristor as the missing circuit element, a lot of research has been done in this direction giving the possibility to implement new technologies for realizing memories and computing systems with an extremely high energy efficiency.

The realization of passive circuit elements with the possibility to preserve the memory of the past history opens a huge quantity of possible applications ranging from classic computation to neuromorphic chips.

2.1. Memristor: the missing circuit element

The first idea of memristor was presented in 1971 by L. O. Chua[43] in which it was referred as the missing element circuit. Indeed, at that time three different passive circuit elements were used:

- resistor: connecting the current to the voltage through the characteristic of resistance $R = \frac{\mathrm{d}v}{\mathrm{d}i}$;
- capacitor: connecting the voltage to the charge through the characteristic of capacitance $C = \frac{\mathrm{d}q}{\mathrm{d}v}$;
- inductor: connecting the current to the magnetic flux through the characteristic of inductance $L = \frac{\mathrm{d}\phi}{\mathrm{d}i}$;

but there was no element connecting the magnetic flux ϕ to the charge q [Fig. 2.1a], i.e. there was a missing relation of the kind:

$$M = \frac{\mathrm{d}\phi(q)}{\mathrm{d}q} \tag{2.1}$$

$$W = \frac{\mathrm{d}q(\phi)}{\mathrm{d}\phi} \tag{2.2}$$

Such a missing element was associated to the resistor also with the name because of its SI unit; indeed, $M(q)$ and $W(\phi)$ are measured in:

$$[M(q)] = \left[\frac{\mathrm{d}\phi}{\mathrm{d}q}\right] = \frac{\mathrm{Wb}}{\mathrm{C}} = \frac{\mathrm{V}\cdot\mathrm{s}}{\mathrm{C}} = \frac{\mathrm{V}}{\mathrm{A}} = \left[\frac{v}{i}\right] = \Omega \tag{2.3}$$

$$[W(\phi)] = \left[\frac{\mathrm{d}q}{\mathrm{d}\phi}\right] = \frac{\mathrm{C}}{\mathrm{Wb}} = \frac{\mathrm{C}}{\mathrm{V}\cdot\mathrm{s}} = \frac{\mathrm{A}}{\mathrm{V}} = \left[\frac{i}{v}\right] = \mathrm{S} \tag{2.4}$$

After that all circuit theory criterion have been satisfied and that the memristor is proved to be a circuit element[43], there is the most important characteristic of such an element to be presented: the hysteresis loop of its *memristance*, i.e. the resistance of the memristor depends on its past history.

If (2.1) is considered, it is clear that it depends on:

$$q(t) = \int_{-\infty}^{t} i(\tau)\mathrm{d}\tau$$

and the same holds for $W(\phi)$, being ϕ the integral over time of the voltage $v(t)$. This demonstrates the memory effect of a memristor and explain its name: a resistance with memory. For the sake of completeness, the relation (2.1) can be rewritten as:

$$\phi = M(q)q$$

where the magnetic flux ϕ can be developed as:

$$\int_{-\infty}^{t} v(\tau)\mathrm{d}\tau = M(q(t))q(t)$$

and, differentiating, it is easy to obtain:

$$v(t) = \frac{\mathrm{d}M}{\mathrm{d}t}q(t) + M(q(t))i(t)$$

so that R_M can be defined:

$$R_M = i^{-1}(t)\frac{\mathrm{d}M}{\mathrm{d}t}q(t) + M(q(t)) \tag{2.5}$$

$$\implies v(t) = R_M i(t)$$

for the chain rule:

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \frac{\mathrm{d}M}{\mathrm{d}q}\frac{\mathrm{d}q}{\mathrm{d}t}$$

and equation (2.5) can be simplified as:

$$R_M(q(t)) = \frac{\mathrm{d}M}{\mathrm{d}q}q(t) + M(q(t))$$

so that, the final relation will read:

$$v(t) = R_M\left[\int_{-\infty}^{t} i(t)\mathrm{d}t\right] i(t) \tag{2.6}$$

when the memristance M depends only on the charge $q(t)$[67], otherwise it can be expressed as a functional depending on x which includes n possible variables:

$$R_M = R_M(x, i, t)$$

This memory of the resistance (or conductance) of a memristor results on a pinched hysteresis loop in the I-V characteristic which defines two types of memristors: self-crossing pinched loop or not [Fig. 2.1a] .

The characteristic of such a loop are depending on the excitation characteristics and it is a necessary feature -for a device defined memristor- when driven by a bipolar periodic signal[30].

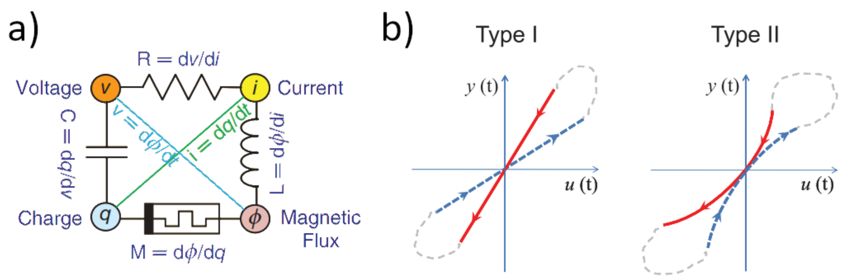


Figure 2.1: a) the missing circuit element with the missing circuit relation, memristor symbol is also represented; b) the two types of pinched hysteresis loops. Image from[67]

Moreover, the hysteresis loop defines the two working states of the memristor: the high resistance state (HRS) and the low resistance state (LRS) which correspond -respectively- to the blue path and the red path in Fig. 2.1b. These two states are extremely important when using memristor for in-memory computing and their characteristics depend on the device. The two operations defined are the *set* when passing from HRS to LRS and *reset* when passing from LRS to HRS.

Finally, the hysteresis loop defines also the switching behavior of a device which can be classified as unipolar or bipolar[20]. In fact, an unipolar switch will happen when the switching from HRS to LRS (or the opposite) is depending only on the magnitude of the applied voltage, while the change of polarity is required to switch on (set) and off (reset) a bipolar mode device [Fig. 2.2].

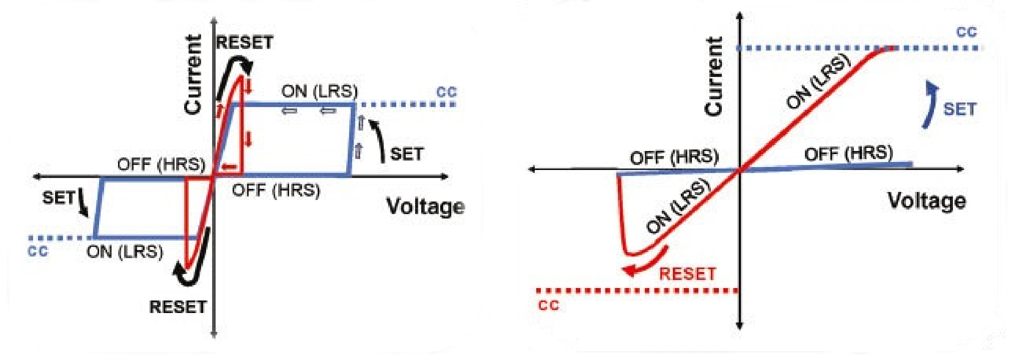


Figure 2.2: memristor switching behavior of a unipolar device (left) and a bipolar device (right). Image from [20]

On the physics point of view, the first memristor model was proposed by HP for oxide-based memristors, since PCM were not existing, and was based on liner ion drift:

$$V = \left(R_{on}\frac{w(t)}{D} + R_{off}\left(1 - \frac{w(t)}{D}\right)\right) I$$

$$\frac{\mathrm{d}w}{\mathrm{d}t} = \mu_v \frac{R_{on}}{D} I$$

where w is the length of dopant filament, D is the total length of the memristor, μ_v is the average mobility of ions, R_{on} is the LRS resistance, R_{off} corresponds to the resistance of the undoped region and w is a state variable that modulate the resistance depending on the applied electric field.

2.2. Types of memristive technologies

The requirements for a memristor device are not easy to satisfy and, thus, the research can be still considered at the beginning of its aim 50 years later the theory by Chua.

Nowadays two main technologies can be found in this field of study: oxide-based memristor and phase-change memristor[36]. In addition, some other materials like electrolytes, heterogeneous materials, perovskite material and others[36, 59] have been proposed for specific tasks.

2.2.1. Oxide-based memristors

The most common structure used for this kind of devices is given by metal-insulator-metal (MIM) structures where the insulator is a binary oxide like HfO_2, SiO_2, TiO_2 or CuO.

The memristive switching in such devices could be analog (gradual) or abrupt (binary) [Fig. 2.3]. This allows to OX-memristors the chance to be selected for two different kinds of work: memory for digital computing where two bits are required, devices for analog computing where a discrete representation with more states is required.

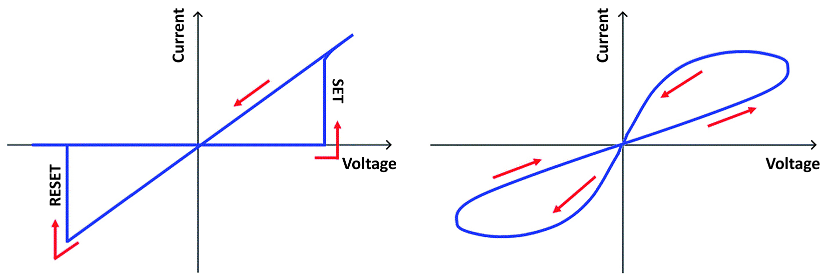


Figure 2.3: Abrupt (left) vs. analog (right) switching behavior of a memristor

Moreover, thanks to the large variety of materials that could be used and their relative common fabrication processes, these devices can be integrated with standard CMOS technology.

The switching mechanism can be generated in two different ways, this will result in two main device categories:

- valence change (VC) device: are also called anionic devices, the conductivity changes thanks to conductive path generated into the oxide from the movement of Oxygen anions under an external electric field;
- electrochemical metallization (EC) device: use an electrochemically active electrode plus a noble counter electrode in order to form a conductive path inside the oxide made of dissolved metal cations from the active electrode; indeed, these devices are also called cationic devices.

Despite the usage of different materials and physical mechanisms, the common aim of all this devices is to form a conductive filament inside the oxide so that the resistance from a very high value can pass to a lower one allowing the device to switch from HRS to LRS [Fig. 2.4].

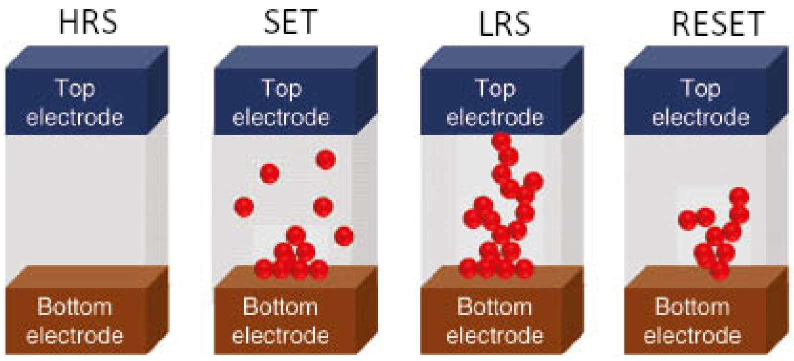


Figure 2.4: Conductive filament representation during the four possible phases. Image from [20]

On the engineering side, a lot of different structures have been implemented among years in order to reduce the set and reset require voltage or improve the conductivity when the device is in the ON state. Some examples are the addition of a capping layer, which could be a film of a different metal in order to increase the switching properties, or the insulator doping with metals or transition metal oxides (TMOs). While the fabrication techniques range from classical physical deposition to more advanced chemical transformation as the atomic layer deposition (ALD).

Since in the following HfO_2 technology is presented, here a model is quickly analyzed[28]. Starting from a piece-wise linear model and adding the non-linear dependence of resistance change with respect to the applied voltage and the resistance saturation, it is possible to write:

$$\frac{dR_M}{dt} = \begin{cases} -C_{LRS}\left(\frac{V(t)-V_{tp}}{V_{tp}}\right)^{P_{LRS}} f_{LRS}(R_M(t)) & \text{if } V(t) > V_{tp} \\ C_{HRS}\left(\frac{V(t)-V_{tn}}{V_{tn}}\right)^{P_{HRS}} f_{HRS}(R_M(t)) & \text{if } V(t) < V_{tn} \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

where C coefficients include the term:

$$\frac{\Delta r}{t_{sw}}$$

with $\Delta r = HRS - LRS$, t_{sw} is the time required to pass from HRS to LRS if is t_{swp} or from LRS to HRS if t_{swn}. Moreover, V_{tp} and V_{tn} are the voltage thresholds for switching and f_{LRS} and f_{HRS} are the terms describing the resistance saturation:

$$f(R_M(t)) = \begin{cases} \left(1+\exp\left[\frac{\theta_{LRS}LRS - R_M(t)}{\beta_{LRS}\Delta r}\right]\right)^{-1} & \text{if } V(t) > V_{tp} \\ \left(1+\exp\left[\frac{R_M(t) - \theta_{HRS}HRS}{\beta_{HRS}\Delta r}\right]\right)^{-1} & \text{if } V(t) < V_{tn} \end{cases} \tag{2.8}$$

with θ_{HRS}, θ_{LRS}, β_{HRS}, β_{LRS} and the values of HRS and LRS which are fitting parameters.

It is important to notice that, in the model above, the change of the resistance in time is described instead of the physical description of much more complicate switching mechanisms taking into account tunneling or exponential ionic drift making the maturity of such models still questionable[28]; furthermore, it cannot be adapted to all memristor types.

2.2.2. Phase change memristors

Phase change memory (PCM) are a relatively new type of memristors which rely on Chalcogenide materials[36]. Such materials were already used in the 1990s for information storage systems such as CDs, DVDs and so on, where power optical heating systems as lasers are available.

The switching effect is obtained from the change of the crystalline structure of the material when melted and the cooled. In fact, thanks to Joule self-heating, it is possible to melt the material applying a sequence of current pulses.

When the current pulse is short in time with a high amplitude, the material starts melting and, since the pulse decreases rapidly, the part of the melted material becomes disordered so that, if it is cooled down quickly, the material falls in its amorphous phase inducing the HRS in the device. On contrary, if the current pulse is long in time and low in amplitude, the material melts and stays in a temperature range falling between the complete melting and the crystallization so that, if it is cooled in a relatively slow time, the material becomes crystalline showing a low resistance (LRS) [Fig. 2.5].

After years of engineering about these devices because of their problems about requirements like high set current as well as low switching set time, the philosophy became to minimize the part of material that has to be melted in order to block the current and reach the HRS. This made of the so-called *mushroom cell* the most common design used, even if thanks to innovative fabrication processes as Krypton fluoride (KrF) laser deep-ultraviolet lithography many structures have been exploited, examples are the *pillar cell* or the *bridge cell*[14] that allow to reach lower write currents.

In the case of the mushroom cell, the contact between the phase change material and the bottom electrode is minimized thanks to the fact that this electrode (also called heater) is the smallest part in the design of the cell. After an finite-element analysis of the thermal distribution, it is noticed that the hottest part is close to the heater in a region where the temperature T_{hs} is:

$$T_{hs} = P_{in}R_{th} + T_{room}$$

where P_{in} is the input power, T_{room} is the room temperature and R_{th} is the average thermal resistance defined on different geometries[48]. This influences the design of the cell since, for equivalent P_{in}, a higher R_{th} means a more efficient cell.

Moreover, if also the Thomson effect is considered, the thermal balance can be viewed as:

$$\nabla \cdot (\kappa \nabla T) + \mathbf{J} \cdot (\sigma^{-1}\mathbf{J}) - T\frac{\partial S}{\partial T}\mathbf{J} \cdot \nabla T - q_{loss}$$

where κ is the thermal conductivity, σ is electrical conductivity of the PC material, q_{loss} is the heat transported away from the PC material and S is the Seebeck coefficient. Since the Seebeck coefficient S is positive with a negative temperature dependence in most PC materials, when $\mathbf{J}$ is in the direction of the temperature gradient -from top electrode to the heater- the Thomson coefficient will be negative, generating and additional heating flux which move the hot-spot away from the heater and expand the amorphous region.

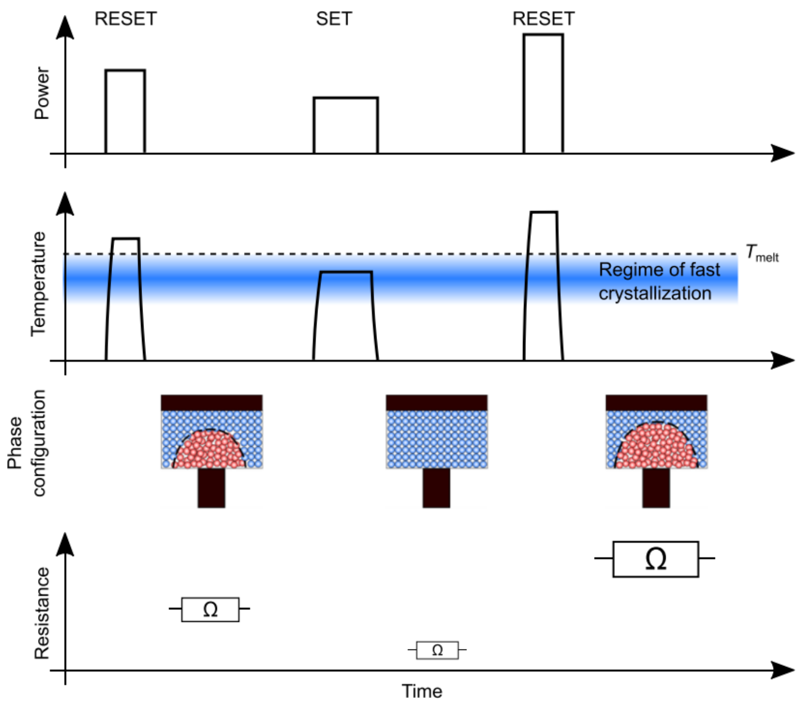


Figure 2.5: Set and reset operations for a mushroom cell. Image from [48]

2.3. OxRAM technology

In the last years oxide based memristors have been exploited in the realization of Resistive Random Access Memory (ReRAM) cells called Oxide RAM (OxRAM) in which the capacitor of Dynamic RAM (DRAM) is substituted by an oxide-based memristor. Thus, the cell is done by a memristor combined with a MOS which allows to read the value of the conductance; such a cell is called 1 Transistor 1 Resistance (1T1R) [Fig. 2.6]. In the following, the ReRAM technology using oxide-based memristor will be described.

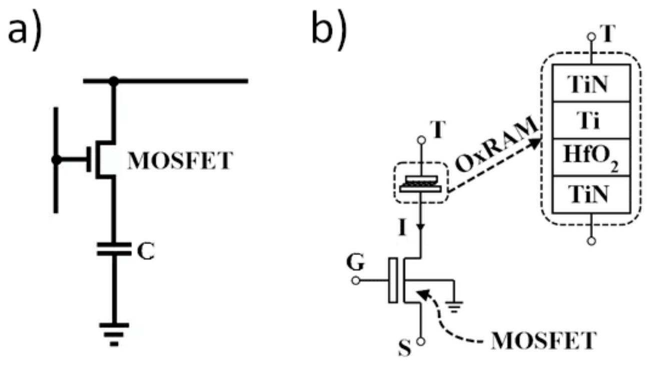


Figure 2.6: a) DRAM cell with the capacitor; b) OxRAM cell with the memristor instead of the capacitor

Such cells can be used both for classic memory application, in which an abrupt switching behavior is required for storing binary information, and for neuromorphic computing in which an analog switching behavior is needed.

Both VC and EC devices are used for this kind of purpose[53], in chapter 4, the VC technology based on HfO_2[8] is simulated and implemented.

The HfO_2 device technology allows to easy integrate the memristor in the back end of line of a 3D cell in which a memristor is connected to a top transistor and one to a bottom transistor that implements FDSOI MOS 65 nm design rule allowing to drastically reduce the cell size ($\sim 1.5\times$).

Moreover, the memristor can be written and read by the same transistor that controls the set and reset operations [Fig. 2.6b] through the control of the compliance current[10]. Indeed, if two voltages V_w for writing and V_r for reading are defined with $V_r < V_w$, it is possible to define three operations referring to Fig. 2.6b:

- read operation: V_G is ON, V_r is applied to T and S is grounded;
- write 0: V_G is ON, V_w is applied to T and S is grounded so that the memristor switches to LRS;
- write 1: V_G is ON, V_w is applied to S and T is grounded so that the memristor switches to HRS;

supposing to work with abrupt switching for binary information storage.

When dealing with neuromorphic computing more conductance levels are required in order to satisfy the analog computation requests. This implies that the memristor conductive filament must be controlled as precise as possible to guarantee the highest bit resolution.

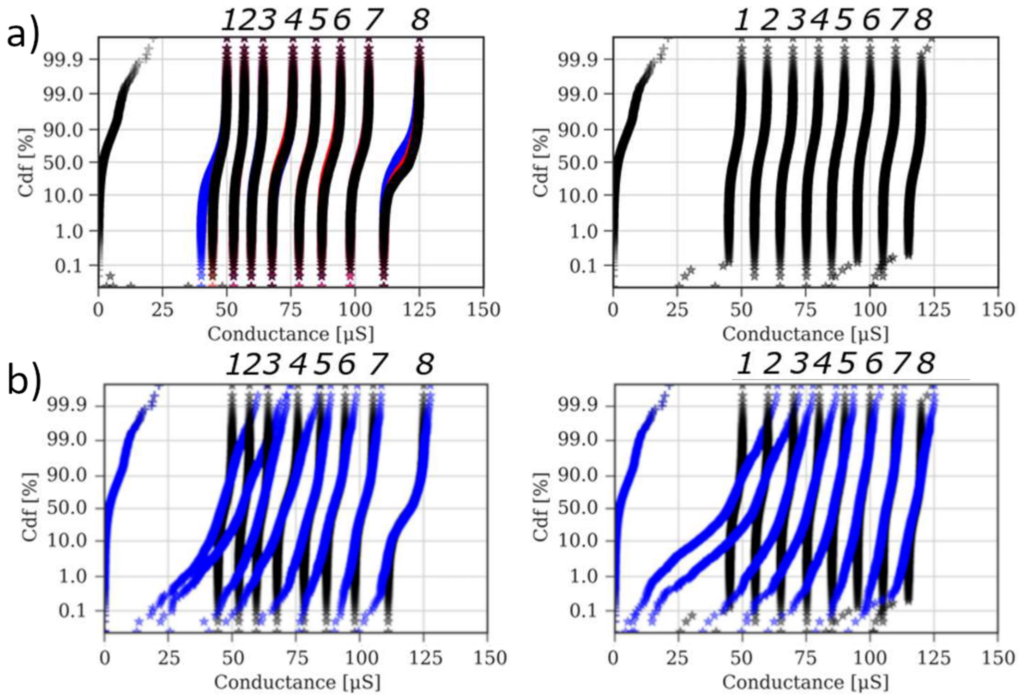


Figure 2.7: a) cumulative density functions for 8-bit precision obtained with SBA (left) and LA (right); b) cumulative density functions for 8-bit precision obtained with SBA (left) and LA (right) showing the time relaxation after 60 s. Images from [8]

The multi-value requirement is accomplished by controlling V_G and so the compliance current flowing in the memristor. In fact, by controlling the intensity and the time of the current, the conductive filament in the HfO_2 layer will grow with different dimensions determining a higher or lower conductance when setting the memristor giving access to more than one conductance level. In such a cell, when V_r is applied, the read current flowing through the memristor is lower allowing to read without modifying the memristor state. This is what is required by neuromorphic computing.

The precision for setting these different LRS values is still an open challenge, but some works established procedures that can be used to obtain up to 3-bit precision [Fig. 2.7]. Using different techniques such as SBA (Sigma-Based Allocation)[9] or LA (Linear Allocation)[8] making advantage of multi programming iterations, it is possible to obtain cumulative density functions for the conductance in the LRS. Finally, the precision is also affected by relaxation, i.e. an amount of time after the programming during which the conductance slightly changes to higher or lower values.

Despite the problems due to conductance programming resolution, 1T1R OxRAM technology is very promising for neuromorphic computing since it results to be less abrupt in switching than PCM ReRAM (PCRAM) technology and it has been demonstrated that it can reach up to 10^5 cycles before a hard fail appearance.

When looking to the HRS and pristine state of the memristor, it turns out that the programming of the conductance value follows a log-normal distribution for the probability density function[31] meaning that by setting two cells in the HRS, two different conductance values will be obtained. Such a log-normal distribution is often present as characterized by:

- the mean μ of the log-normal itself;
- the std σ of the underlying Gaussian distribution.

Typical values are $\mu = 10^8\,\Omega$and $\sigma = 0.4$, in chapter 4 higher values for μ, supposing the characterization to be valid also for the pristine state, are used.

3. Spiking Neural Networks

Spiking neural networks (SNN) are part of the so-called neuromorphic computing in which the structure and the behavior of the brain are modeled into electronic circuits in order to perform non-linear computation tasks.

3.1. Spiking neural network principles

Neuromorphic computing aim is to emulate brain behavior to perform computation and so SNNs' is; but, in order to do it, models of the brain are required and in this section the state-of-the-art is presented for each part of the brain used in such a paradigm.

3.1.1. Information encoding

As suggested by the name, the information inside SNNs is encoded through spikes, i.e. electrical pulses sharp in time which travel through the network for doing computation. This system of encoding is very bio-plausible since in the brain itself the information is encoded through spikes; the difference is that in SNNs the spikes are moving electrons instead of ions bringing a larger velocity in information traveling.

The method used to encode an analog signal into spikes remains an open challenge, but a lot of work have been done in the last years giving rise to different encoding systems. Depending on the task that has to be solved, i.e. what is the input signal, it is possible to choose the better spike encoding.

Numeric encoding

Numeric encoding makes possible to transform each number into a spike train, i.e. to perform a proper digitization. In fact, each numeric value in the input signal is normalized in range $[0, 1]$ and then converted in a spike train of length T which contains $x_i T$ ones and $(1 - x_i)T$ zeros where x_i is the normalized value that has to be encoded[74].

This method allows to encode an input with the desired precision by changing T and can be applied to different desired input signal such as time-varying or space-varying (images); unfortunately, it is not possible to use it for real-time analysis.

Bio-inspired encoding

For temporal analog input signal, models that emulate the working principle of human receptors have been developed.

It is the case of the case of cochlea-mechanism-based conversion[5, 63]. Here three components are modeled and used for spike conversion:

- basilar membrane;

- hair cell;
- bushy cell.

The basilar membrane is simulated following the concept that the interaction between a membrane and a fluid causes spatial frequency dispersion. I.e., the sound arriving to the membrane causes vibration of the last inside a fluid which can be considered inviscid and incompressible; such vibration can be expected to be small so that the fluid can be considered linear. The membrane is furthermore divided into N_{ch} channels that are susceptible to different frequencies, i.e. vibrate differently with respect to the same input frequency.

Then, the membrane movement is converted into spikes by the hairy cell that contains a number of free transmitters equal to $q(x,t)$ which can be released to the synaptic cleft. This implies that in the synaptic cleft there is a varying number of transmitters $c(x,t)$; such a number of transmitters have the probability to excite the postsynaptic potential:

$$\mathcal{P}_{spike} = h \cdot c(x,t)\mathrm{d}t$$

it is also possible to define a refractory period by denying events in a certain range of time. Moreover, a number equal to N_{HC} of hair cells for each channel of the basilar membrane is considered.

Finally, the bushy cell is modeled as a LIF neuron. A number of N_{ch} bushy cells is connected without recurrent connections, in a single layer, to integrate independently all the N_{HC} cells corresponding to a single channel.

E.g., in [5], $N_{ch} = 700$ channels of the basilar membrane are considered, each one is connected to $N_{HC} = 40$ hairy cells and for each group of N_{HC} hairy cells a single bushy cell is used to integrate the spike train giving raise a a spike conversion of an audio signal with 700 channels.

Space-varying signals spike encoding

A lot of work has been done for space-varying input signal -images- spike encoding[33] because SNN are optimized for time-varying inputs while have the key problem of encoding when dealing with images.

Examples are the rate coding and the burst coding. In the first, the input pixels are scaled down by a factor λ and considered as a firing rate which determines a Poisson spike train, i.e. a spike train in which the spike position in time is randomic and the firing rate must be respected. In the second one, a burst of spike which contains a number of spikes and inter-spikes interval proportional to the pixel intensity is sent at one time; this allows to increase the reliability of synaptic communication between neurons.

Sigma-delta modulator

Spike encoding means digitization of a signal so that, theoretically, every ADC (Analog to Digital Converter) is an available choice to perform the operation.

The sigma-delta modulator is an ADC able to perform real-time spike conversion with very low power consumption with respect to other ADC. Moreover, it is a circuit and this makes it very suitable for chip integration [Fig. 3.1].

Given an input analog signal, the sigma-delta modulator is able to produce two spike trains corresponding to up and down spikes. The spikes are generated each time that the signal increase (up spike) or decrease (down spike) more than a given threshold δV with respect to the previous signal or the reference voltage V_{ref}; this implies that such a circuit is clock-less and, thus, more energy efficient. Moreover, the firing rate -spike frequency- can be adjusted by changing δV, keeping into account that it is possible to choose different values for δV_{up} and δV_{dn}.

Finally, a refractory period can also be set by changing the value of C_3 which increase or decrease the hold period when the ADC is reset to V_{ref}.

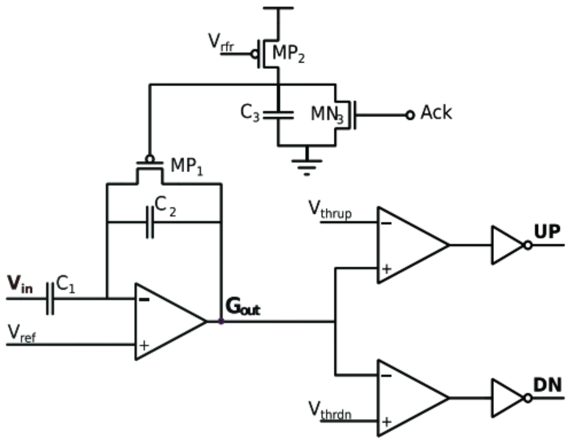


Figure 3.1: Scheme of ADC sigma-delta modulator. Image from [45]

The sigma-delta modulator is a total hardware spike encoder able to perform real-time conversion, this makes it optimal for circuits that have to analyze temporal signals but very difficult to adapt to space-varying signals.

Time-to-first-spike coding and phase

In this paradigm, the importance of an information is contained in the time that passes from a reference signal to an input stimulus[50] and is very suitable for visual stimuli where it is possible to decide that larger a pixel is, higher the information carried is and earlier the spike is emitted[33].

In this case, the stimuli from a presynaptic neuron are not collected in time by a postsynaptic one.

When the input signal is periodic, it is possible to apply a version of the time-to-first-spike encoding where the phase is evaluated and so the variations between one period and another are encoded. Indeed, the key point is to encode the information in the phase of a pulse with respect to the background periodic signal.

3.1.2. Synapse models

Synapses play a crucial role in the brain activity as in NNs functioning. Indeed, they are considered responsible of learning and weighting of information.

Among years, a lot of circuits have been presented for emulating synapse dynamics[38] and, then, have been largely replaced by memristor devices such as OxRAM or PCRAM. In the following the DPI (Differential-Pair Integrator) circuit for emulating synapse dynamics will be presented since has been used for the project in chapter 4.

DPI circuit for synapse dynamics

The DPI circuit aim is to mimic the synapse dynamics including the weight of the incoming signal and the short term integration.

In fact, it acts like a linear filter with the capacity to integrate spike inputs from different sources -presynaptic neurons- by keeping the exponential dynamics and the compactness.

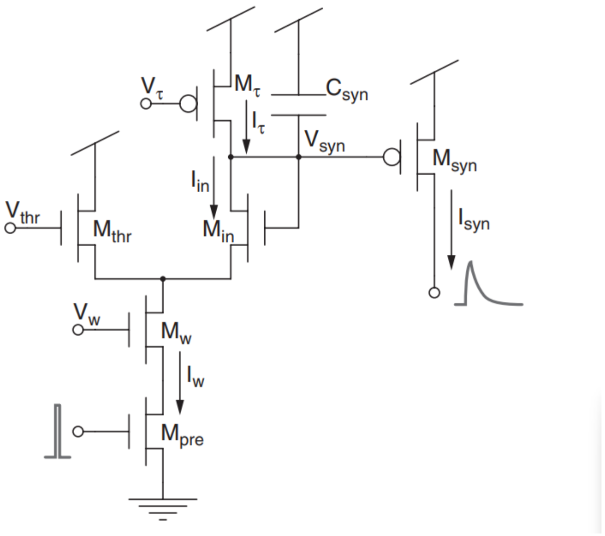


Figure 3.2: DPI circuit for synapse dynamics. Image from [38]

Most of the transistor in the circuit [Fig. 3.2] operates in sub-threshold regimes, so that if $I_g \gg I_{syn}$ holds by setting V_{thr} it is possible to write:

$$\tau \frac{\mathrm{d}I_{syn}}{\mathrm{d}t} + \frac{I_{syn}^2}{I_g} - I_{syn}\left(\frac{I_w}{I_\tau} + 1\right) = 0 \tag{3.1}$$

which can be simplified as:

$$\tau \frac{\mathrm{d}I_{syn}}{\mathrm{d}t} = I_{syn}\left(\frac{I_w}{I_\tau} + 1\right)$$

so that the change in circuit response increases up to when the condition is no more satisfied or even reached at the opposite $I_g \ll I_{syn}$; at this point the condition for low-pass filter behavior is reached:

$$\tau \frac{\mathrm{d}I_{syn}}{\mathrm{d}t} + I_{syn} = \frac{I_g}{I_\tau} I_{in}$$

whose solution is (4.2), where $I_{out} \equiv I_{syn}$. Notice that I_g is a virtual p-type subthreshold current which is not generated by any p-MOS in the circuit.

Moreover, should be noticed that:

- C_{syn} in combination with I_τ control the integration time τ;

- V_w control the weighting of the input signal;
- the output I_{syn} of the circuit follows the exponential decay typical of an action potential.

Finally, it is possible to implement NMDA and conductance-based plasticity mechanisms in addition to restoring mechanisms for short-term plasticity, i.e. for resetting V_w[7].

Memristor synapses

In the last years, memristor devices with an analog switching behavior are replacing circuit for synapse dynamics such as the DPI[10].

Thanks to the resistor nature of memristor, it is easy to implement a ReRAM cell as a weight by setting a resistance value R_M on the memristor and simply converting the incoming spike as:

$$I_w = \frac{V_{in}}{R_M}$$

where I_w is the weighted input that will feed the neuron.

Moreover, it is not necessary to implement additional circuit for emulating long-term and short-term plasticity[38] since they are both part of the memristor behavior. Indeed, the short-term plasticity is part of the memristor volatile nature in which the filament tends to relax to higher conductances after that a pulse ends in order to minimize the energy inside the oxide. On the other hand, long-term plasticity is due to phenomena which makes the memristor such an interesting device: it has memory of the past so that if the conductance have been set to a certain value even if it could be slightly modified by short-term effects, it will be centered around a given value.

3.1.3. Neuron models

The CPU of a neural network is the neuron and, thus, its modeling is very important and determining the performance.

However, a trade-off between bio-plausibility and implementation simplicity must be analyzed as discussed in the following. Moreover, the neuron has to work with spike trains as input, hence the models used for ANNs are not always compatible.

All the neuron models presented act not only as integrators in time but also in space, i.e. the input current is given by the weighted sum of all presynaptic currents as in the McCulloch-Pitts model[54]:

$$I(t) = \sum_i w_i u_i(t)$$

where w_i is the weight associated to the potential u_i of presynaptic neuron i and is measured in S.

Leaky integrate-and-fire model

The most basic neuron model is the leaky integrate-and-fire (LIF) neuron in which an RC circuit is used to model the membrane. Important features are the driving current $I(t) = I_R(t) + I_C(t)$ and the membrane potential $u(t)$.

From the Ohm's law and the capacitor current it is easy to compute the driving current as:

$$I(t) = \frac{u(t)}{R} + C\frac{\mathrm{d}u(t)}{\mathrm{d}t} \tag{3.2}$$

while the membrane potential is given by the multiplication of (3.2) by R and the membrane time constant $\tau_{mem} = RC$, which simply is the integration time of an RC circuit:

$$\tau_{mem}\frac{\mathrm{d}u(t)}{\mathrm{d}t} = -u(t) + RI(t) \tag{3.3}$$

In such a model, the spike event of the neuron occurs at time t_f when the membrane potential:

$$u(t_f) \geq \vartheta$$

and then, immediately after a firing event, the membrane potential is reset to $u = u_{rest}$, with $u_{rest} < \vartheta$.

It is clear that the circuit integrates incoming spikes with a leakage in time due to the capacitor discharge. Moreover a refractory period during which the dynamics is interrupted after an output spike event in order to limit the firing frequency.

Non-linear IF models

In the non-linear integrate-and-fire family of models (3.3) is replaced by:

$$\tau_{mem}\frac{\mathrm{d}u(t)}{\mathrm{d}t} = F(u) + G(u)I \tag{3.4}$$

where $G(u)$ is a voltage dependent input resistance and $F(u)/(u - u_{rest})$ is a voltage dependent time constant[50].

To this family belongs the quadratic integrate-and-fire model (QIF) whose equation is:

$$\tau_{mem}\frac{\mathrm{d}u}{\mathrm{d}t} = a_0(u - u_{rest})(u - u_c) + RI \tag{3.5}$$

with the parameters $a_0 > 0$ and $u_c > u_{rest}$. The spiking mechanism as for LIF is generated whenever $u \geq \vartheta$. The difference with respect to the LIF model is that using this model is possible to reproduce an action potential.

Adaptive exponential neuron model

The adaptive exponential (AdExp) model equation is (3.4) expressed in terms of currents with:

$$F(u) = f(u) - w$$

$$G(u) = 1$$

so that becomes:

$$C\frac{\mathrm{d}u}{\mathrm{d}t} = f(u) - w + I \tag{3.6}$$

where w is the adaptation term and $f(u)$ is a combination of linear and exponential functions[40]:

$$f(u) = -g(u - u_{rest}) + g\Delta_\vartheta \exp\left[\frac{u - \vartheta}{\Delta_\vartheta}\right]$$

where $g = 1/R$ is the leak conductance and Δ_ϑ is the slope factor and determines the sharpness of the threshold. Again, when a spike event occurs, the potential is reset to u_{rest}.

Finally, the adaptation current w is described by:

$$\tau_w \frac{\mathrm{d}w}{\mathrm{d}t} = a(u - u_{rest}) - w$$

with a indicating the level of adaption.

An example of application of such a model is the DPI neuron[12] in which a log-domain low-pass filter neuron is implemented through four circuits:

- a DPI for low-pass filtering;
- a spike event generator and a current-based positive feedback;
- a reset-refractory pulse generator;
- an adaptation low-pass filter implemented through another DPI filter.

Despite the simplicity, such a model predicts with 96 % accuracy the spike timing (± 2 ms) of much more complex models such as the Hodgkin-Huxley which has more than 100 fitting parameters[40].

Hodgkin-Huxley: the biological model

The most complicated models are the ones strictly connected to biology dynamics; examples are Hodgkin-Huxley model and their reduction to two dimensions[50]. The complicated formalism is partially due to the pragmatic nature of the model, i.e. it is a mathematical set of equations derived as a consequence of experimental data instead of being the origin of predictions from a mathematical paradigm. This process revert the cause-effect flow in scientific research and produces the quasi-total absence of mathematical formalism in disciplines such medicine or biology because of the huge quantity of parameters needed to fit the model to experimental data.

By the way, the model has been obtained by the study of giant squids axons and represent the neuron membrane as composed of three ions channel: Sodium (Na^+, Potassium (K^+) and Chloride (Cl^-) which is indicated as a leaky channel. Since here is a difference in concentrations of difference ions between inside and outside the membrane, a Nernst potential is generated and represented by a voltage supply. In addition, two variables conductances are inserted in order to describe the lower -or higher- probability of moving ions when the concentration is too high -or too low- with a constant leakage conductance [Fig. 3.3a]. Thus, the current can be expressed as:

$$I(t) = C\frac{\mathrm{d}u}{\mathrm{d}t} + \sum_j I_j(t)$$

where j indicates different ions channels.

Such a model will not be further investigated since it is never used in spiking neural network models because of its complexity and the high specificity variables. It is relevant to notice that the potential of the LIF model is not equal to the action potential of biological models [Fig. 3.3b], while the one generated by non-linear IF models is similar.

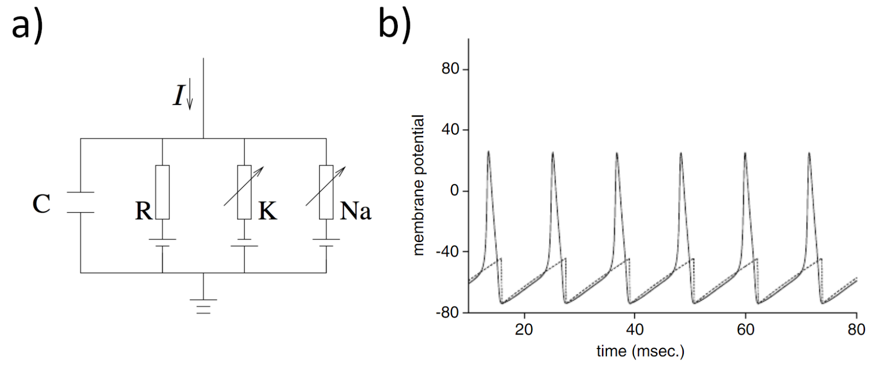


Figure 3.3: a) Hodgkin-Huxley model circuit representation. Image from [50]; b) comparison between Hodgkin-Huxley action potential (solid line) and LIF potential (dashed line). Image from [71]

3.1.4. Learning in SNNs

From deep learning it is possible to learn lessons about learning on NNs that can be applied to spiking architectures in order to enjoy the benefits deriving from these last ones[15]. The focus is always to solve the so-called *dead neuron problem* [Fig.3.4] in which back-propagation is useless because of the non-differentiability of spikes, indeed looking at the solution of (3.4) discretized in time it reads:

$$u(t+\Delta t) = \beta u(t) + W_w R x_{in}(t) \tag{3.7}$$

where β is the exponential decay of the membrane potential, Δt is the time step for time discretization, W_w is the weight conductance, R is the leak resistance, $x_{in}(t)$ is the input spike voltage and the reset term has been subscript. Since the spiking output is normally given by:

$$S(t) = \begin{cases} 1 & \text{if } u(t) > \vartheta \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

a weight update ΔW_w (which is the base of back-propagation as in (1.9)) corresponds to a change in the potential of Δu, but such a change fails to precipitate a further change in in (3.8):

$$\frac{\mathrm{d}S}{\mathrm{d}u} \begin{cases} \rightarrow \infty & \text{if } u = \vartheta \\ = 0 & \text{otherwise} \end{cases}$$

which can be expressed in terms of the gradient of loss in the weight space

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_w} \in \{0, \infty\}$$

which does not enable learning.

Anyway, learning methods remain an open challenge in the field of deep learning for ANNs as for SNNs. Each policy described in the following has its field of shining where it can outperform the others, so that it is possible to adapt the rule to the task that has to be performed.

Shadow training

When the input signal is not time-varying, as in case of static images, and the inference efficiency is much more important than training efficiency a good way to go is the training of an ANN whose parameters are then converted into SNN parameters; e.g., the activation function of the neurons is converted into either a spike rate or a latency code[15].

The two main disadvantages in such a training method are:

- the conversion of sequential ANNs to SNNs is not that explored;
- the activation function in ANNs is much more precise than the spiking conversion;

that make the trained SNN never reaching the original ANN performance.

Surrogate gradient

When using surrogate gradient (SG) approach[66], the spiking function (3.8) is substituted with a smoother function [Fig. 3.4] that allows the derivative of the spiking function to be replaced with a similar one which is differentiable, in other words:

$$\frac{\mathrm{d}S}{\mathrm{d}u} \leftarrow \frac{\mathrm{d}\tilde{S}}{\mathrm{d}u}$$

so that the loss function derivative is now indicating a gradient that can be followed by deep learning gradient descendent algorithms.

This method allows to train native SNNs using deep learning algorithms without requiring higher computational power or parameters transferring. On the other hand, it is adding a hyperparameter to the training phase: the choice of the surrogate gradient.

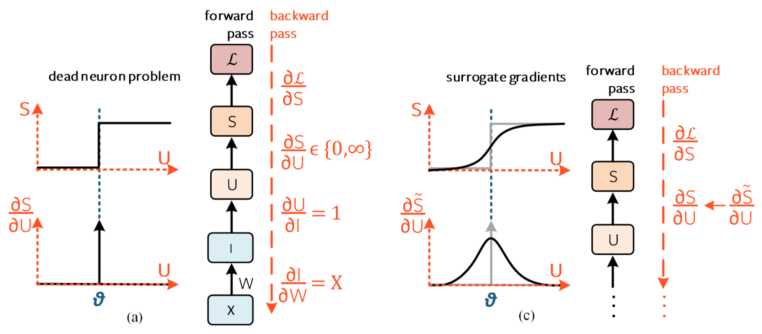


Figure 3.4: Dead neuron problem graphical representation and surrogate gradient solution. Image from [15]

Learning on memristor devices

When introducing memristor devices inside neuromorphic chips and SNNs, it is mandatory to consider the hardware nature of such devices even when simulating or interacting with them through learning algorithms.

This is why the training of memristor-based SNNs remains an open challenge.

Considering that a change ΔW corresponds to a change of the conductance in the device and that the precision in the programming of such conductance cannot reach the precision of software simulations -32-bit or even more- innovative mixed-precision learning policies have been proposed[35].

Using these policies, it is possible to use a learning block on the chip to reach also online learning in which data are available in a sequential order.

The mixed-precision update provides tools for keeping the trace of gradient by storing its changes and then deciding if it is necessary to update the weight. E.g. suppose that the gradient is stored for j and that the original conductance $G_i(0)$ -weight- of a memristor synapse is falling inside the distribution with mean G_1, the accumulated gradient is requiring a change of $\Delta W = \Delta G$ so that the conductance after j epochs should be $G_i(0) + \Delta G$, there are two available options:

- $|G_i(0)+\Delta G-G_2| > |G_i(0)+\Delta G-G_1|$: the weight is not updated and kept constant to its original value $G_i(0)$;
- $|G_i(0) + \Delta G - G_2| < |G_i(0) + \Delta G - G_1|$: the weight is updated to $G_i(j)$ which is sampled from the distribution with mean G_2.

3.1.5. SNNs implementations

Spiking neural networks offer a wide range of application through a large set of different implementations spacing from the software-based to the on-chip implementation. Of course, it is evident that they can fully prove their efficiency -especially in terms of power consumption- when following on-chip approach.

When looking at hardware implementation, it becomes important to consider one of the biggest issues of the last decades in technology: the von Neumann bottleneck, i.e. the limited throughput between the CPU and the memory compared to the memory in a von Neumann architecture. Indeed, von Neumann architecture is obtained when designing a chip which has different blocks for storing data (memory) and for doing operations on data (CPU); these two blocks need to communicate because in order to elaborate data it is mandatory to send them from the memory to the CPU and back to the memory, but if the communication speed is limited the velocity of operations results limited as well. In the following, three chips that may be considered as the state-of-art about neuromorphic chips overcoming von Neumann bottleneck are presented.

Software implementations

Simulating SNNs on software is not a simple task since they show the same hyperparameters of ANNs plus others and require the implementation of different differential equations with respective solutions discretized in time. On the other hands this is very often a mandatory step, especially for custom chips, and allows to simulate architectures which can be much more complicated that on-chip respective.

Anyway, there are two main philosophies for doing software-based implementations of SNNs.

The first one is more classical from deep learning point of view and requires the knowledge of existing Python library also used for ANNs like PyTorch, TensorFlow or Keras adapting the structure of the simulated NN to the SNN architecture requirements[75].

The second is based on specific libraries such as Brian2[52] which already includes neuron, synapses and other components models allowing to write a code more similar to the hardware real behavior.

They both have pros and cons, e.g. the use of specific libraries gives less flexibility when customizing the network for introducing new components or new methods for training.

DYNAPs chip family

Dynamic Neuromorphic Asynchronous Processors (DYNAPs) is a family of chips designed using non von Neumann architecture for address-event representation (AER) using asynchronous digital circuits[29]. In such a system, neurons are process units which are connected in different ways in order to form a hardware implementation of a SNN.

The difference with ASICs used for standard deep learning is that spike events are substituting the classic clock-based floating point computation and feed directly the process units avoiding the von Neumann bottleneck obtaining an asynchronous spike-based computation.

Both neurons and synapses are implemented through the DPI models of the two components[38, 12] while the communication between them is obtained by a two phase routing scheme. Indeed, the N neurons used for a layer of the network are grouped in N/C clusters of C neurons each and the fan-out operation -i.e. the communication with the successive layer- is divided into two stages:

1. neurons use the source-address routing for targeting a subset of intermediate nodes of dimension F/M with $F > M$ and where F is the fan-out of the first layer neurons; the number of intermediate nodes in N/C which is $N/C \leq F/M$;

2. the intermediate nodes target a number $M \leq C$ of neurons in the second layer within each cluster C; each neuron in the end-point cluster uses a set of tags (one of K tags for each cluster) and decides if accept or ignore the incoming input.

This two-stage fan-out operation allows to overcome von Neumann bottleneck by distributing the memory across the different neurons and to minimize the total digital memory by keeping a higher fan-out M so that the total memory required by each neuron can be divided into:

- source memory MEM_S;

- target memory MEM_T.

Finally, the chip architecture provides the possibility to implement multi-core neuromorphic processors in which there are three different asynchronous routers and embedded SRAM and CAM (Content Addressable Memory) cells distributed across cores and routers.

The chip allows to reach impressive specifications and ultra-low power consumption per event [Tab. 3.1]. Moreover, thanks to the memory distribution on neuron and synapse arrays, it is already possible to implement innovative technologies such as ReRAM for further reducing the energy consumption.

Specifications		
Process technology	0.18 µm 1P6M	
Die size	43.79 mm^2	
Voltage supply (core)	1.3 V-1.8 V	
Supply voltage (I/O)	1.8 V-3.3 V	
Number of cores	4	
Number of neurons	1 k	
Number of synapses	64 k	
Total memory	64 k CAM and 4 k SRAM	
Energy consumption		
Operation	Core voltage supply	
	1.3 V	1.8 V
Single spike generation	260 pJ	883 pJ
Encode one spike and append destinations	507 pJ	883 pJ
Broadcast event to same core	2.2 nJ	6.84 nJ
Route event to different core	78 pJ	360 pJ
Extend pulse generated from CAM match	26 pJ	324 pJ

Table 3.1: DYNAPs family chip specifications and power consumption per event from [29]

Loihi

Loihi is a NC chip produced to overcome the von Neumann bottleneck, to use all the knowledge from neuroscience and to respect all the requirements for efficient running of SNNs[22]:

- sparse network compression: the fan-out neuron indices are computed basing on states stored with each synapse state in a sparse manner; this can be done using three different sparse matrix compression methods;
- core-to-core multicast: any neuron can send spikes to any other neuron in the network following the SNN architecture;
- variable synaptic formats: the weight resolution can vary from 1 to 9 bits representations and can be signed or unsigned;
- Population-based hierarchical connectivity: connectivity templates can be defined as generalized weight sharing mechanisms;

all this taking advantage of asynchronous design thanks to SNNs properties, including bio-plausibility structures and elements such as dendrites and axons.

Moreover, each core implement spike-time-dependent plasticity (STDP) based learning engines for changing synapses variables and perform on-chip learning.

Finally, it is the most dense neuromorphic chip implementation [Tab. 3.2] which allows to implement SNNs using all possible architectures and with ultra-low power consumption.

Specifications	
Process technology	14 nm FinFET
Die size	$60.00\,\mathrm{mm}^2$
Voltage supply	$[0.5, 1.25]$ V
Number of cores	128
Number of neurons	1024 per core
Synaptic memory	16 MB
Total memory	33 MB SRAM
Energy consumption	
Operation	Voltage supply
	0.75 V
Synaptic spike	23.6 pJ
Synaptic update	120 pJ
Neuron update (active/inactive)	81 pJ/52 pJ

Table 3.2: Loihi specifications and energy consumption per event from [22]

TrueNorth

TrueNorth is the largest neuromorphic chip and was the most dense up to Loihi coming. The basic block for such a structure is a core with 256 input lines (called axons), 256 output neurons and 256-by-256 synaptic connections.

Then, 4096 cores are connected through on-chip connections and each neuron is able to send signals to any other axon in the chip; such a communication is done in two stages:

1. a single connection travels a long distance between a first and a second core;
2. a second multiple connection connect the first to each axon on the second core.

This allows to the chip to reach high efficiency thanks to the fact that neurons form clusters that draw their inputs from a similar pool of axons and only spike events sparse in time are communicated. Of course, off-chip connections are also present in order to communicate with the external world input signals coded in form of spikes.

The neuron dynamics is discretized through 1 ms time step with a global 1 kHz global clock. The spike advantage in terms of clock on such a chip is represented by fully asynchronous inter-core and event-driven intra-core computation. This makes the active power proportional to the firing activity. A lot of other benefits such as scalability and flexibility are guaranteed by this chip[27].

Both online and offline learning are possible with TrueNorth for implementing a large quantity of different SNNs architectures such as liquid state machines, restricted Boltzmann machines or convolutional networks.

The chip characteristics [Tab. 3.3] are still very innovative even if the first presentation of the chip was in 2014.

Specifications	
Process technology	Samsung's 28 nm
Die size	4.3 cm^2
Voltage supply	0.775,V
Number of cores	4096
Number of neurons	1 M
Number of synapses	256 M
Total memory	6 T SRAM
Energy consumption	
Operation	Voltage supply
	0.775 V
Synaptic event	26 pJ

Table 3.3: TrueNorth chip specifications and power consumption per event from [27, 2]

3.2. Spiking neural network time signal analysis

In the previous section, all the instruments for projecting and implementing a SNN were given. In the following, the state-of-art about SNN architectures used for doing time-varying signal processes will be explained, knowing that this architectures can be implemented with the previously explained methods or with custom chips.

The main problem regards memory. Indeed, differently from space-varying signals, in time-varying signals the information is not all available immediately but is distributed in time so that for real-time computation storing information for a certain amount of time becomes mandatory. In certain cases, short-period signals, neurons models provide a very short memory corresponding to the integration time of the circuit, but for longer period signals a larger memory and more complicated mechanisms for obtaining it are necessary.

The key-point is that SNNs are built for analyzing spike trains which are already time-varying signals. I.e., in principle, SNNs are always doing real-time computation. The problem is when are they doing true real-time with respect to analog input signal?

3.2.1. Feed-forward SNN

Feed-forward architectures represent the most simple possible architecture for SNNs, they are built using one or more layers of neurons which are connected in a way that allows the spikes to travel following only one direction. They can use both excitatory and inhibitory synapses for connecting the neurons in the layers.

An example of such a structure has been presented for EMG (electromyography) signal classification using two layers of AdExp neurons connected with DPI synapses for having both excitatory and inhibitory connections[46]. The two layers are structured in a way that:

- the spike conversion is obtained through sigma-delta modulation;
- a first hidden layer of 192 neuron receives a linear combination of fixed-weight randomly generated from four input EMG channels through excitatory and inhibitory

connections; this allows each neuron to receive 16 input spike trains (1 up and 1 dn spike trains for each of the four channels connected twice with excitatory and inhibitory synapses);

- an output layer of 3 neurons for doing classification; each neuron is connected through trainable weights to all the neurons in the hidden layer.

The network is implemented on DYNAP chip using off-chip online training for the output layer weights following the delta rule:

$$\Delta w_{ij} = \alpha(T_j - y_j)x_i$$

where i indicates the ith hidden neuron, j represents the jth output neuron, T_i is the target, y_i is the network output and x_i represents the input pattern.

The classification is done by observing what is the neuron spiking with the highest firing rate among the three and connecting it to a class of gestures.

The network is said to be "a step towards realizing an end-to-end solution, from sensors to classification, for the real-time processing of EMG signals"[46], so that it is still not doing real-time classification with respect to incoming analog signal.

Works based on feed-forward SNN on others dataset for time-varying signals like EGG have also been presented[23].

Balanced SNN

Balanced SNN is a particular architecture of the feed-forward family in which excitatory and inhibitory synapses are used in a different ways between two layers.

A three layer architecture of balanced SNN has been used for analyzing vibration patterns of bearing test rings for detecting anomalies[25].

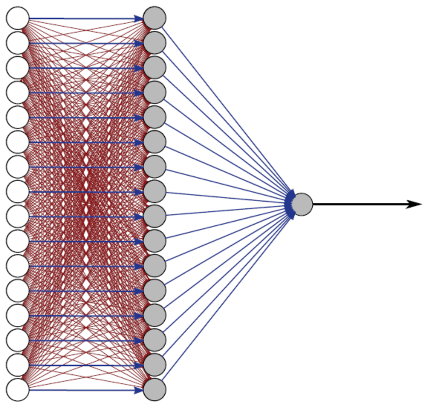


Figure 3.5: Balanced SNN example: in red inhibitory synapses, in blue excitatory synapses. Image from [25]

In the work, the vibration patterns are converted into spikes through a sigma-delta ADC with a refractory period and the spike trains are used to feed a SNN made of an input layer whose dimension N corresponds to the input number of channels; the layer is connected to a N dimensional hidden layer where each n_j neuron receive an excitatory connection with the respective h_j neuron in the input layer and combines it with $N-1$

inhibitory connections from all other neurons in the input layer [Fig. 3.5]; the weight between h_j and n_j is set to be:

$$w_{jj} = \alpha(N-1)w_{ij,i\neq j}$$

where α is a scaling parameter; this is done in order to balance the difference between excitatory and inhibitory synapses.

The hidden layer neurons, finally, send their output to the output neuron through only excitatory connections. The network has been successfully run on both software implementations like Brian2 and hardware implementation through DYNAP chip.

3.2.2. Recurrent SNN

Recurrent SNNs (RSNNs) are the equivalent of recurrent network in the ANN paradigm. The architecture provides the possibility to send information through multiple dimensions in order to form a short-term memory whose importance is determined by the weight related to recurrency.

If the most simple form of recurrency is considered, the signal is sent back just to the neuron itself and the equation (3.7) for a fully connected single neuron becomes:

$$u(t+\Delta t) = \beta u(t) + \sum_i W_i R x_{in,i}(t) + VwRS(t) \tag{3.9}$$

where $S(t)$ is formally (3.8) but can be substituted with the surrogate gradient function if used and V is the weight associated to the recurrent signal of the neuron, i.e. the neuron is sending its weighted output to itself.

Reservoir computing

Reservoir computing (RC) paradigm is derived from recurrent network so that the principle of keeping memory of the signal through recurrency is maintained.

The basic structure for such an architecture is formed by a reservoir connected to the input and to a read-out layer [Fig. 3.6a]. The reservoir is, then, formed by a sequence of randomly and recurrently interconnected virtual nodes -i.e. neurons- to constitute a network with internal feedback loops. This allows to the reservoir state to be influenced only by recent past, i.e. to form a short term memory of the input.

Generally, the read-out layer is fully connected to all the nodes in the reservoir and the weights of such connections are the only trained among the entire architecture.

When the reservoir computing system is using SNNs models it is usually called liquid state machine (LSM) because it response is similar to a liquid forming ripples in response to an input. The reservoir in this case is not guarantee to be fully-connected, but the probability of connection between two neurons is inversely proportional to their respective distance as in brain.

The reservoir dynamics can be expressed as:

$$x^M(t) = (L^M x_{in})(t) \tag{3.10}$$

where x^M indicates the reservoir state, $x_{in}(\cdot)$ is the input spike train and L^M is the filter used for transforming the input in the reservoir state[13].

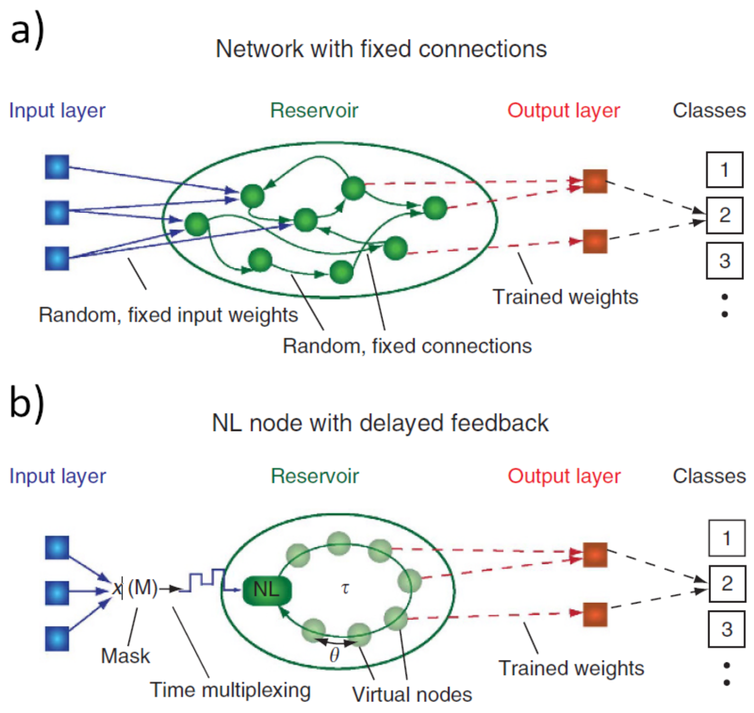


Figure 3.6: a) classic RC network with fixed connections; b) RC network with non-linear node with delayed feedback as reservoir. Images from [18]

Recent trends tend to substitute the classical reservoir with a non-linear node with delayed feedback[18] where virtual nodes are used to equally divide a delay time τ into N part, where N is the number of nodes [Fig. 3.6b]. The state of each node is then used to describe the reservoir state at time t:

$$x^M(t) = x(t-(N-i)\theta)$$

where $\theta = \tau/N$. This represents one of the first attempts to use temporal delay as memory in NNs.

When the input signal is space varying it is possible to treat it as time varying for using LSM and RC for doing image classification. This is done by using a sequence of pixels as a train of spikes and using a reservoir entirely realized through memristor synapses with optoelectronic response[19]. Also on-chip implementations for ECG signals classification of RSNN reservoir-inspired have been done for real-time analysis[39].

4. Single Neuron SNN for Real-Time Analysis of Temporal Signals

In the last years, the evidence that two neurons are connected though multiple synapses have been suggested in medical and biological papers even if the reasons behind this characteristic remain unknown[56].

In the presented project, this multi-synapse connections are supposed to be used to introduce time delays in order to process temporal signals. Indeed, through delays applied to the incoming signal -by exploiting non-idealities of memristors- it is possible to generate a short-term memory for storing important signal features.

4.1. Architecture

The SNN is a perceptron-inspired architecture for analysis of time-varying signals made of 4 dendrites, each dendrite is then connected to 64 synapses, each synapse is finally associated to a delay [Fig. 4.1].

The first synapse of each dendrite is associated to a 0 s delay, i.e. to a memristor in its LRS.

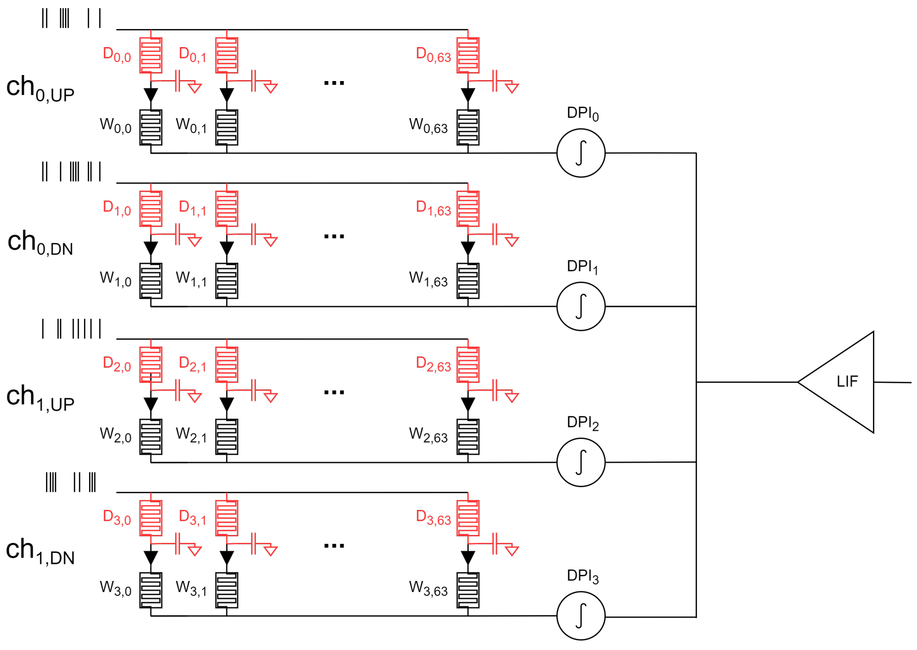


Figure 4.1: SNN architecture represented in a code-aware version in order to accomplish both PyTorch and circuit representations

Spike encoding

The spike encoding for the input signal is done through a sigma-delta modulator[46, 45], i.e. a self-timed clock-less ADC which produces for an input signal two train of spikes, respectively up and down (dn).

These spikes are generated when a change in the input signal is equal or higher than a fixed δV.

Delays

The delay is obtained from a RC circuit made of a memristor which can be set in HRS or in LRS and a capacitor whose delay is given by:

$$D_{i,j} = R_{M,i,j}C$$

where $R_{M,i,j}$ is the memristor resistance associated to synapse j on dendrite i and C is the capacitance.

To optimize the area efficiency of the circuit, the capacitance is fixed at 100 fF while the memristor is used to select different delays. Since the LRS resistance R_M of the memristor is much lower than its HRS resistance, the delay of an ON memristor is considered $\tau \sim 0\,\mathrm{s}$.

Since R_M for a memristor in HRS is not fully determined but it is a value sampled from a log-normal distribution[31], the non-ideality of the memristor is used to obtain one different delay for each synapse on the dendrite; moreover the first delay of each dendrite i -$D_{i,0}$- is set to 0 s.

Weights

The weight of each synapse is again a memristor which can be set to HRS to have a zero weight, i.e. no signal from the synapse, or in LRS knowing that the LRS conductance $W_{i,j} = 1/R_{M,i,j}$ is sampled from a Gaussian distribution whose mean can be fully determined in 8 values[8] obtaining a total of 9 available weight values not fully-determined but sampled from Gaussian distributions. In such a way, the current entering the DPI -or directly the neuron- is:

$$I_i(t) = \sum_j W_{i,j} V_{ADC}(t - D_{i,j}) \tag{4.1}$$

The aim of the SNN during the training will be to learn what are the desired delays in order to keep in memory the features characterizing the input signal.

Synapse

The synapse dynamics is given by a DPI circuit[6] connected at the end of each dendrite which is receiving the sum of all weighted signals. As will be analyzed in next sections, the DPI can be avoided if required from the chip design.

The time-response of the DPI$_i$ is modeled as follows:

$$\begin{aligned} I_{out,i}(t) =& g_{DPI} I_{in,i}(t) \left(\exp\left[-\frac{\tau_s}{\tau_{DPI}} \right] - 1 \right) + \\ & I_{out,i}(t - \tau_s) \exp\left[-\frac{\tau_s}{\tau_{DPI}} \right] \\ g_{DPI} =& \frac{I_g}{I_\tau} \end{aligned} \tag{4.2}$$

where I_g is a virtual p-MOS sub-threshold current, I_τ is the current controlling the decay time constant τ_{DPI}, τ_s is the pulse width and $I_{in,i}$ is the current computed in (4.1).

Neuron

The neuron is a simple LIF neuron described by (3.2) considering the rest potential:

$$\tau_{mem} \frac{\mathrm{d}u}{\mathrm{d}t} = -\left(u - u_{rest}\right) + R_n I_{in}(t) \tag{4.3}$$

where τ_{mem} is the membrane time constant and can be tuned and R_n is the resistance associated to the time constant

$$R_n = \frac{\tau_{mem}}{C}$$

with $C = 100\,\mathrm{fF}$; in addition to the membrane time constant, also the threshold ϑ is tunable. The current entering the neuron $I_{in}(t)$ is given by:

$$I_{in}(t) = \sum_i I_{out,i}(t)$$

where $I_{out,i}$ is the one from (4.2) or, if the DPI is avoided, the one from (4.1); finally, it is adapted through a current limiter in order to use reasonable values for ϑ.

Everything is simulated using PyTorch and an hardware-aware coded so that the circuit parameters are accounted and the full power of deep learning is used.

4.2. Hardware-aware coding

The main objective for the network during the training is to learn the weights in a way that allows to keep the right delays for storing memory.

The principal idea is that, using different delays, the signal is sent to the neuron multiple times so that old features can be easily connected in time with the last arrived. Such a method is used in reservoir computing or in RNN where the memory is kept using recurrent connections to send the signal back to the neuron itself. Anyway, using dendrites and delays with multi synapses connection allows to increase scalability while reducing power consumption.

In order to simulate the network dynamics while using deep learning features, a Python code is required. Such a code must take into account the hardware features and this is the reason why it is called *hardware-aware code*.

The first step is to discretize time in the equations describing the architecture, then the so-called hyperparameters of SNN should be defined and, finally, scaling issues regarding currents and voltages amplitude must be considered.

4.2.1. Time discretization

Instead of using the usual continuous time conception and description, is normal to use discrete formalism in NN which are solving temporal tasks.

In SNN this requires the definition of a time step Δt and the redefinition of equations describing the system.

Spike encoding

The sigma-delta modulator is coded with a loop that given the analog signal -sampled with frequency $f_s = 1/\Delta t$- compares the current analog value with the previous one or the reference -set to the first incoming value- and generates two vectors with the same length of the input array [Algorithm 4.1].These are the up and dn spike trains that will be used for feeding the network.

Algorithm 4.1 *delta_modulation*(x_analog, dv)

```
Input: analog signal $x\_analog$ of shape $(t\_steps, n\_ch\_analog)$ where $n\_ch\_analog$ is
    the number of analog channels; voltage threshold for spike emission $dv$;
Output: spike train $delta\_mod$ of shape $(t\_steps \times n\_ch = n\_ch\_analog * 2)$
1: $n\_ch = n\_ch\_analog * 2$
2: $delta\_mod =$zeros$[t\_steps][n\_ch]$
3: for $i \leftarrow 1$ to $n\_ch\_analog$ do
4:     $prev = x\_analog[1][i]$
5:     for $j \leftarrow 1$ to $t\_steps$ do
6:         if $x\_analog[j][i] - prev > dv$ then
7:             $delta\_mod[j][2*i-1, 2*i] = [1, 0]$        ▷ up spike generation
8:             $prev = x\_analog[j][i]$
9:         else if $x\_analog[j][i] - prev < dv$ then
10:            $delta\_mod[j][2*i-1, 2*i] = [0, 1]$        ▷ dn spike generation
11:            $prev = x\_analog[j][i]$
12:        end if
13:    end for
14: end for
15: return $delta\_mod$
```

Delays

The delays are described by an integer number of spike given by:

$$D_{i,j}^{\text{discrete}} = \text{int}\left(\frac{D_{i,j}}{\Delta t}\right)$$

where int(·) function takes only the integer part of the division. Such a delay is added at the beginning of each spike train entering the network though a specific function which computes the discrete delay and the concatenates the original vector at the bottom of the delays vector [Algorithm 4.2].

In order to take into account non-idealities of the ADC, random spikes are added with a probability of 0.01 %, i.e. the probability that a noisy spike happens at time t_i in the interval $[0, D_{i,j}^{\text{discrete}}]$ is 0.01 %. Moreover, the length of simulation for each run of the network in terms of time steps will be given by:

$$n_{steps}^{\text{tot}} = n_{steps}^{\text{analog}} + \max_{i,j} D_{i,j}^{\text{discrete}}$$

where $n_{steps}^{\text{analog}}$ corresponds to the length of the two vectors computed in the ADC loop.

Algorithm 4.2 $delay(delta\_mod, t\_s, mu, sigma, c, n\_syn, n\_ch)$

```
Input: mean of the log-normal $mu$; std of the underlying Gaussian $sigma$; capacitance
    $c$; time step $t\_s$; number of dendrites $n\_ch$; number of synapses $n\_syn$; input spike
    train $delta\_mod$ with shape $(t\_steps \times n\_ch)$
Output: input spike train with delays $x\_delayed$ of shape $(t\_steps + d\_max \times n\_syn \times$
    $n\_ch)$
 1: $r$ =zeros$[n\_syn][n\_ch]$
 2: $r[2 : \text{end}][\text{all}] = lognormal(mu, sigma, size = n\_ch * n\_syn - n\_ch)$  ▷ sample from
    log-normal
 3: $tau = r * c$
 4: $t\_steps = length(x\_data)$
 5: $d\_max = (int)(max(tau)/t\_s)$
 6: $x\_delayed$ =zeros$[t\_steps + d\_max][n\_syn][n\_ch]$
 7: for $i \leftarrow 1$ to $n\_syn$ do
 8:     for $j \leftarrow 1$ to $n\_ch$ do
 9:         $d = (int)(tau[i][j]/t\_s)$
10:         $x\_delayed[\text{all}][i][j] = concatenate(\text{zeros}[d], x\_data[\text{all}][j], \text{zeros}[d\_max - d])$
11:     end for
12: end for
13: return $x\_delayed$
```

Weights

The input spike train, after the delay, has to be weighted since it is passing through conductances of different magnitude.

This operation can be simply performed by a matrix multiplication of the type $abc, cd \rightarrow abd$:

$$\mathbf{I_W} = \mathbf{I_{delay}} \cdot \mathbf{W}$$

where $\mathbf{I_W}$ will be a matrix $(n_{steps}^{\text{tot}} \times n_{ch})$, $\mathbf{W}$ is a matrix $(n_{syn} \times n_{ch})$ and $\mathbf{I_{delay}}$ is a matrix $(n_{steps}^{\text{tot}} \times n_{syn} \times n_{ch})$.

Notice that in Algorithm 4.5 the matrix multiplication is performed through a for loop, while on PyTorch much more efficient tools such as *torch.einsum* are provided.

Synapse

The DPI are modeled with the respective equation in discrete time. Starting from the single pulse response:

$$I_{out}(t_i + n\Delta t) = g_{DPI} I_{in}(t_i) \left(\frac{\exp\left[\dfrac{\Delta t}{\tau_{DPI}}\right] - 1}{\exp\left[\dfrac{(n+1)\Delta t}{\tau_{DPI}}\right]} \right) \tag{4.4}$$

which is directly derived from the discretization of (4.2) using $\tau_s = \Delta t$ and describes the DPI output after n time steps.

Using (4.2) discretized in time is possible to simulate the time behavior of the DPI by considering that $I_{in,i}(t_k)$ is the weighted and delayed current at time t_k

Indeed, the python code is a loop over time steps of weighted and delayed input spike trains [Algorithm 4.3] which is, in fact, a current since weight dimension is expressed in S.

Algorithm 4.3 *dpi_convolution*$(x\_weighted, tau\_dpi, t\_s, g\_dpi)$

```
Input: weighted and delayed spike train $x\_weighted$ with shape $(t\_steps + d\_max \times n\_ch)$; DPI time constant $tau\_dpi$; time step $t\_s$; DPI gain from (4.2) $g\_dpi$
Output: DPI output current array $dpi\_out$ with shape $(t\_steps + d\_max \times n\_ch)$
1: $decay = exp(-t\_s/tau\_dpi)$
2: $dpi\_out = \text{zeros}[t\_steps + d\_max][n\_ch]$
3: $dpi\_out[1][\text{all}] = x\_weighted[1][\text{all}] * g\_dpi * (1 - decay)$
4: for $i \leftarrow 2$ to $t\_steps + d\_max$ do
5:     $dpi\_out[i][\text{all}] = x\_weighted[i][\text{all}] * g\_dpi * (1 - decay) + dpi\_out[i-1][\text{all}] * decay$
6: end for
7: return $dpi\_out$
```

Neuron

The neuron, at the end, needs adapted equations too in order to be inserted in the discrete time system:

$$u(t + \Delta t) = \beta u(t) + R_n \sum I_{DPI,i}(t) - S(t)$$

where R_n is defined in (4.3) and the new terms are:

$$\beta = \exp\left[-\frac{\Delta t}{\tau_{mem}}\right] \text{ exponential decay}$$

$$S(t) = \Theta(u(t) - \vartheta) \text{ neuron output}$$

notice that the neuron output can be either 0 V or 1 V -indicating the emission or not of a spike- since $\Theta(\cdot)$ is denoting the Heaviside function.

When writing the code associated to the neuron [Algorithm 4.4], the neuron output function will be modified accordingly to the surrogate gradient technique[66].

Algorithm 4.4 *neuron*($x\_in, tau\_mem, t\_s, theta$)

```
Input: neuron input current $x\_in$ with shape ($t\_steps + d\_max \times n\_ch$); neuron time constant $tau\_mem$; time step $t\_s$; spiking threshold $theta$
Output: neuron membrane potential $u\_mem$ with shape ($t\_steps + d\_max + 1$); neuron spiking output $out$ with shape ($t\_steps + d\_max$)
 1: $\beta = exp(-t\_s/tau\_mem)$
 2: $r\_n = tau\_mem/10^{-13}$
 3: $out = zeros[t\_steps + d\_max]$
 4: $i\_in = zeros[t\_steps + d\_max]$
 5: for $i \leftarrow 1$ to $n\_ch$ do
 6:     $i\_in[\text{all}] = i\_in[\text{all}] + x\_in[\text{all}][i]$
 7: end for
 8: $u\_mem = zeros[t\_steps + d\_max + 1]$
 9: for $i \leftarrow 1$ t0 $t\_steps + d\_max$ do
10:     $out[i] = spike_fn(u\_mem[i] - theta)$                    ▷ [66]
11:     $u\_mem[i+1] = (beta * u_mem[i] + r\_n * i\_in[i]) * (1 - out)$
12: end for
13: return $u\_mem$, $out$
```

Neural network

The total network will be coded using an object whose call function is a cascade of the previous methods for simulating the current flow [Algorithm 4.5].

Algorithm 4.5 *network_call*($x\_in, t\_s, mu, sigma, c, n\_syn, n\_ch, w, tau\_dpi, g\_dpi, tau\_mem, theta$)

```
Input: delta modulated input array $delta\_mod$ of shape ($t\_steps \times n\_ch$); time step $t\_s$; mean of the log-normal $mu$; std of the underlying Gaussian $sigma$; capacitance $c$; number of dendrites $n\_ch$; number of synapses $n\_syn$; weight array $w$ with shape ($n\_syn \times n\_ch$); dpi time constant $tau\_dpi$; dpi gain $g\_dpi$; membrane time constant $tau\_mem$; spiking threshold $theta$
Output: neuron membrane potential $u\_mem$ with shape ($t\_steps + d\_max + 1$); neuron spiking output $out$ with shape ($t\_steps + d\_max$)
 1: $x\_delayed = delay(delta\_mod, t\_s, mu, sigma, c, n\_syn, n\_ch)$
 2: $t\_steps = length(x\_delayed[\text{all}][0][0])$
 3: $x\_weighted = zeros[t\_steps][n\_ch]$
 4: for $i \leftarrow 1$ to $n\_ch$ do
 5:     for $j \leftarrow 1$ to $n\_syn$ do
 6:         $x\_weighted[\text{all}][i] = x\_weighted[\text{all}][i] + x\_delayed[\text{all}][j][i] * w[j][i]$
 7:     end for
 8: end for
 9: $dpi\_out = dpi\_convolution(x\_weighted, tau\_dpi, t\_s, g\_dpi)$
10: $u\_mem, out = neuron(dpi\_out, tau\_mem, t\_s, theta)$
11: return $u\_mem, out$
```

4.2.2. Network hyperparameters

In the presented architecture, the number of layers and neurons per layer is not a hyperparameter since the network is a perceptron-inspired model, but the SNN hyperparameters such as weight initialization, membrane time constant and synapse time constant -corresponding to the DPI one- have to be defined and set in the right way in order to obtain a successful training. In the following, hyperparameters concerning the training phase are not considered and will be analyzed in section 4.3.

Being the network simulated on software, the quantity of parameters that can be considered *hyper* is huge, but some are constrained by the hardware requirements:

- neuron spiking threshold $\vartheta = 1\,\mathrm{V}$;
- input spike amplitude $V_{in,ADC} = 100\,\mathrm{mV}$;

some other parameters, that should require a strict hardware consideration, can be set with more freedom since the training is not online as explained in section 4.3; this is the case of:

- weight initialization, which does not require to take into account low-bit precision;
- spike pulse width for training;

Finally, for bio-inspired parameters, the bio-plausibility is always tried to be respected[60]:

- membrane time constant τ_{mem};
- synapse time constant $\tau_{syn} = \tau_{DPI}$;

Moreover, in order to set good values, the $f_{out} - f_{in}$ characteristic, in which the frequency response of the network to a certain input frequency is represented, should be considered; but, to consider such a characteristic the dataset should be know in order to adapt the parameters to the task that should be chosen. As analyzed in section 4.4, the dataset used for demonstrating the capability of the network to pick the right delay is the MIT-BIH[64].

In the following, the procedure is presented for the first proof of concept presented in section 4.4, but it has been done for every different train session.

After the creation of an input spike train whose maximum frequency is $f_{dataset} = 360\,\mathrm{Hz}$ and keep into account that when a spike up occurs a spike dn cannot be present (resulting in a maximum frequency of $f_{spiking} = 180\,\mathrm{Hz}$ for a single channel), the network with randomly initialized weights is feed with it.

The hyperparameters that must be studied are:

- w_{scale} used for initialize the weights using a Gaussian distribution with $\mu = 0$ and $\sigma = w_{scale}/\sqrt{n_{ch}}$;
- membrane time constant τ_{mem};
- DPI time constant τ_{DPI} that is used for synapse dynamics.

It is important to notice that the network allows only positive weights while PyTorch can deal with both positive and negative weights, so the results of the distribution for weights initialization will be used after taking the absolute value of each weight. Moreover, since the amplitude of the output current of the DPI decreases as the time constant increases [Fig. 4.2], a factor of compensation must be included in the gain of the DPI by setting it to:

$$g_{DPI} = \frac{\exp\left[\dfrac{\Delta t}{\tau_{DPI}}\right]}{\exp\left[\dfrac{\Delta t}{\tau_{DPI}}\right] - 1}$$

which is derived from (4.4) by setting $n = 0$ and $I_{in} = I_{out}$.

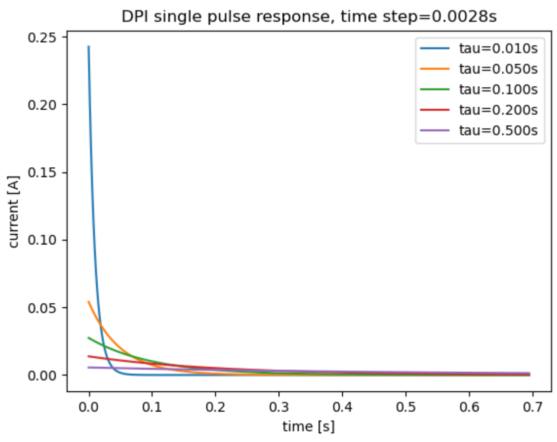


Figure 4.2: DPI single pulse response

The objective is to find a combination of the three parameters presented above that makes the neuron firing with a frequency comprised in the linear region of the $f_{out} - f_{in}$ characteristic.

Once that results are obtained [Fig. 4.3], it is possible to see how in the region $f_{in} \in [100, 130]$ Hz the quality of the curve is very unrealistic due to a bad spike encoding: it is difficult to keep a relation between 360 Hz in which the input tensor has always a spike alternating on up or dn channel and frequencies in that range.

A good solution for the hyperparameters has been found to be:

- $w_{scale} = 3 \cdot 10^{-2}$;
- $\tau_{mem} = 10\,\text{ms}$;
- $\tau_{DPI} = 7\,\text{ms}$.

This choice allows to run the network in a linear region on its characteristic while keeping bio-plausibility constraint. On the other hand, basing on the task that has to be solved and to the firing frequency required to the neuron, it is possible to change the curve of reference by changing the hyperparameters or by reducing the gain of the DPI as will be seen in the next section.

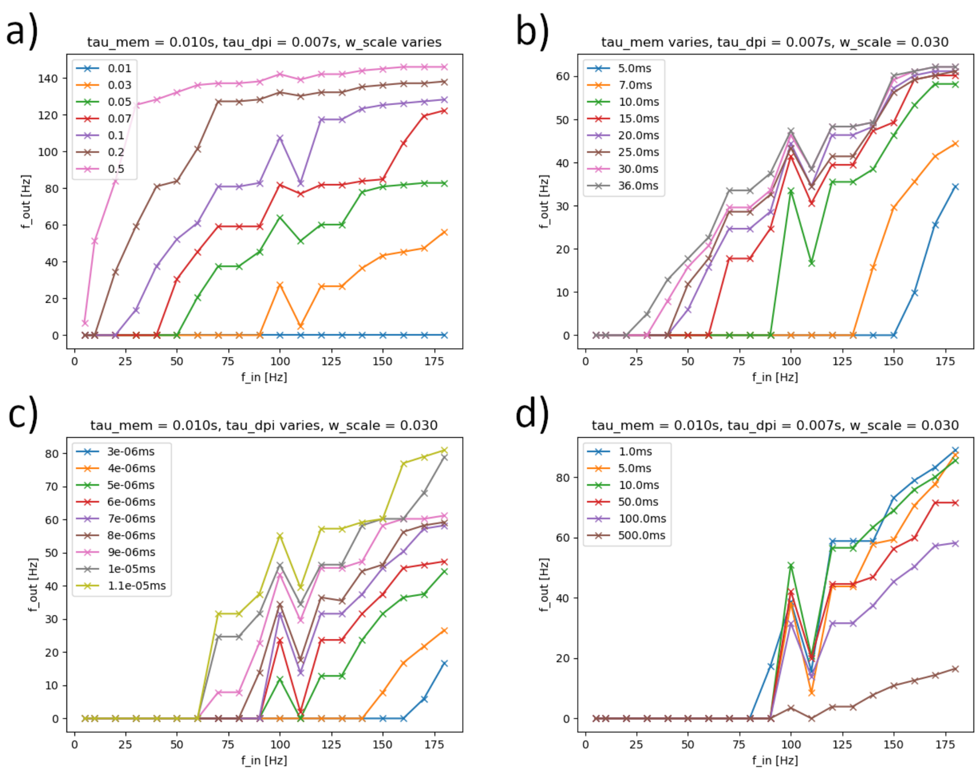


Figure 4.3: a) w_{scale} variation keeping τ_{DPI} and τ_{mem} constant; b) τ_{mem} variation keeping τ_{DPI} and w_{scale} constant; c) τ_{DPI} variation keeping w_{scale} and τ_{mem} constant; d) mean of the delay lognormal variation keeping τ_{DPI}, w_{scale} and τ_{mem} constant;

In this sense, it is important to say that the parameter that can be changed easily is w_{scale} while letting the DPI gain at 1 at least when configuring the network during first training sessions. Indeed, the change of other hyperparameters could not only bring to a hardware unaware use of Python but also to non-bio-plausible configurations which can be undesired.

4.2.3. Power and energy consumption

Both power and energy consumption are computed when doing inference on all the elements present in the network.

It is important to know that the consumption is optimized and engineered only in the signal analysis, i.e. for weights and delays. Moreover is very hard to estimate the power and energy consumption for training phase since it can be performed in many different ways.

Starting from the sigma-delta modulator, its power consumption is computed on a 180 nm CMOS technology based circuit[45] adapting the results to the dataset used:

$$P_{ADC} = 55\,\mu\mathrm{W}\frac{f_{dataset}}{100\,\mathrm{Hz}}$$

while for the power, it is simply computed by multiplying the power by the time of inference t_{inf}:

$$E_{ADC} = P_{ADC}t_{inf}$$

this will result to be the most expensive circuit element.

Delays

The delays are supposed to be read at every time step since is not possible to know what are the non-zero weights a priori. Since they obtained through an RC circuit in which the resistance is given by a memristor in its HRS, the power in time is computed as:

$$p^D_{i,j}(t) = Vi(t) = \frac{V_{read}(t) - v_C(t)}{R_{i,j}} = \frac{V_{read}(t) - V_{read}(t)\left(1 - \exp\left[-\frac{t}{R_{i,j}C}\right]\right)}{R_{i,j}} =$$
$$= \frac{V_{read}(t)^2}{R_{i,j}} \exp\left[-\frac{t}{R_{i,j}C}\right]$$

where i is indicating the dendrite and j the synapse, $C = 100\,\mathrm{fF}$, $V_{read}(t)$ is the input spike train exiting the sigma delta modulator with the amplitude used for reading delays. Thus, for computing the energy of each RC circuit:

$$E^D_{i,j} = \sum_{k=0}^{n^{\mathrm{tot}}_{steps}} \int_{t_k}^{t_{k+1}} \frac{V^2_{read}(t_k)}{R_{i,j}} \exp\left[-\frac{t}{R_{i,j}C}\right] \mathrm{d}t$$

knowing that each integration between t_k and t_{k+1} is equal to the integration in range $[0, \tau_s]$, the energy and power of each delay are respectively given by:

$$E^D_{i,j} = \sum_{k=0}^{n^{\mathrm{tot}}_{steps}} CV^2_{read}(t_k)\left(1 - e^{-\dfrac{\tau_s}{R_{i,j}C}}\right)$$

$$P^D_{i,j} = \sum_{k=0}^{n^{\mathrm{tot}}_{steps}} \frac{CV^2_{read}(t_k)}{n^{\mathrm{tot}}_{steps}\tau_s}\left(1 - e^{-\dfrac{\tau_s}{R_{i,j}C}}\right)$$

the total values are derived by summing over $i \in [0, 3]$ and $j \in [0, 63]$. This is a good approximation which takes into account that the pulse width of a single spike will be $\ll t_s$ on the chip.

Weights

The weights energy and power consumption corresponds to the dissipation of a resistor which is read every time that a spike occurs. Of each resistor the conductance $W_{i,j}$ corresponding to the weight is known and will be used in the following.

The power is computed referring to a AC signal on a resistor:

$$P^W_{i,j} = \left(V^{\mathrm{RMS}}_{i,j}\right)^2 W_{i,j}$$

where RMS indicates the root mean square of the voltage $V_{i,j}(t_k)$ which is $V_{read,W}(t_k)$, i.e. the read voltage for weights after the delay. As the delay the total power is given by the summation over i and j. It is important to notice that the spike pulse width will influence the power consumption as $V^{\mathrm{RMS}}_{i,j}$ decreases as the pulse width decreases.

For what concerns the power, time cannot be reduced through the use of root mean square but must be considered:

$$E^W_{i,j} = n^{\text{tot}}_{steps}\tau_s \sum_{k=0}^{n^{\text{tot}}_{steps}} V^2_{i,j}(t_k)W_{i,j}$$

of course, the total power is given by the sum over i and j. Moreover, the energy indicates that there are two possible situations in which power is not dissipated through weights at time t_k:

- $V_{i,j}(t_k) = 0\,\text{V}$, i.e. no spike occurs and the weight is not read;
- $W_{i,j} = 0$ so that the weight will be never be read.

It is clear that depending on the training and the number of delays picked through $W \neq 0$, the power consumption will slightly vary.

Neuron

The LIF neuron, finally, is computed basing on a 22 nm CMOS technology based circuit[4] on which the energy per spike is computed for every spike, knowing that the neuron has very low chance to overcome 30 Hz of firing rate.

Neural network

The total power and energy consumption is computed as:

$$P_{TOT} = P_{ADC} + P_{LIF} + \sum_{i=0}^{3}\sum_{j=0}^{63}\left(P^D_{i,j} + P^W_{i,j}\right)$$

$$E_{TOT} = E_{ADC} + E_{LIF} + \sum_{i=0}^{3}\sum_{j=0}^{63}\left(E^D_{i,j} + E^W_{i,j}\right)$$

in which only the terms regarding delays and weights are optimized in this work.

4.3. Training method

The training of the network is performed on computer using PyTorch 1.10 through a dedicated object, it is structured -as in classical deep learning paradigm- in epochs during which the dataset is showed divided in batches, the loss function is computed between the LIF neuron output and the label and the back-propagation function is called on the weights. Hence, the training is supervised since each batch contains both the input spike train and the label.

Like the network, the training has some hyperparameters that will be discussed in the following.

The training phases are projected to be done offline with some phases that can be performed online using a learning block which is able to accumulate the gradient. Thus, thanks to hardware-aware coding, the initialized network chip with the respective delays can be uploaded on a computer and then the weight can be set on the chip considering the training results. In this way back-propagation and PyTorch built-in functions can be fully exploited basing on the task to be solved.

The only parameters undergoing to back-propagation and, thus, to training is the weights vector since the aim is to chose the right delays for recognizing signal features and this is done by keeping high the corresponding weights.

4.3.1. Training hyperparameters

The training phase is characterized by some hyperparameters which determine the speed and accuracy of training phases. Such hyperparameters are:

- loss function: a function that maps the difference of the output of the network with respect to the label;
- optimizer: an algorithm which changes the parameters of the SNN basing on the loss function result;
- learning rate lr: how fast is the learning of the network, i.e. what is the step of change after each back-propagation call;
- number of epochs n_{epochs}: how many time the training set is showed to the network and the back-propagation method is called.

Unfortunately, in deep learning there are not specified criteria for setting these parameters outside the experience and some general considerations[15].

Being the proof of concept built on binary classification performed on MIT-BIH dataset, the selected loss function is *BCEWithLogitsLoss* method belonging to PyTorch *torch.nn* class in which the binary cross entropy loss is computed after a sigmoid function is applied to the input, i.e. to the output of the neuron. Since the possibility to define a threshold of tolerance for the number of spikes -i.e. a number of spikes which are ignored and considered as noise related- the network output has to be clamped before computing the loss [Algorithm 4.8].

On the optimizer side, three different PyTorch algorithm belonging to the *torch.optim* class have been exploited: *Adam*, *Adamax* and *RMSprop*. The best trade-off on speed, accuracy and overfitting was found to be *RMSprop*.

Finally, there is a relationship between the learning rate and the number of epochs to be used. Starting from the assumption that the learning rate cannot be too low otherwise the optimizer gradient descendent algorithm could stuck on a local minimum during the training, across all the learning routines.

To sum up, training hyperparameters are set to be:

- optimizer: *RMSprop*;
- loss function: *BCEWithLogitsLoss* for binary classification;
- learning rate: $lr \in [10^{-4}, 10^{-3}]$;
- number of epochs: $n_{epochs} < 30$;

4.3.2. Positive weights

The architecture of the network, in order to be more area and power efficient, uses only positive weights [Fig. 4.1] and this should be considered while training the SNN because PyTorch do not consider any constraint on the weight while training the network. Moreover, a PyTorch built-in function for keeping the weights positive does not exist so that a personalized method has been coded for this purpose.

The method is relatively simple [Algorithm 4.6] and it is applied to the network weights after each batch back-propagation call. It consists of a weight minimum clamping to zero, i.e. all the negative weights are set to zero when the method is called.

Algorithm 4.6 $w\_clamping(w)$

```
Input: weight array $w$ of shape $(n_{syn} \times n_{ch})$
Output: weight array clamped to be $< 0$
1: for $i \leftarrow 1$ to $n_{ch}$ do
2:     for $j \leftarrow 1$ to $n_{syn}$ do
3:         if $w[j][i] < 0$ then
4:             $w[j][i] = 0$
5:         end if
6:     end for
7: end for
8: return $w$
```

Of course, PyTorch provides computational efficient tool to solve this request such as *torch.clamp* function, but the intent of algorithms is to show the idea behind a function.

4.3.3. Low-bit precision for weights training

Writing memristors with analog switching on I-V characteristics does not allow to select the conductance in a fully deterministic way with the all possible wanted values. By the way, methods for sampling the conductance value from a Gaussian-like distribution with 8 available mean values have been presented[8].

In other words, it means that the resolution for the weights values is 3-bit and the value when a memristor is written in a new state is not fully determined, but sampled from a Gaussian distribution. Such a behavior cannot be neglected during training and different techniques can be used for training networks considering it.

For this project a mixed-precision learning[35] approach is chosen. The objective is to accumulate gradient for a given number of epochs and then update the weights using the 3-bit resolution; since each time a memristor is written a new sample from the Gaussian must be done, only the weights which move from a distribution to another will be written and resampled [Algorithm 4.7].

This kind of function is called in two different ways: the first one uses a list of possible distribution including HRS -$\mu = 0$ and $\sigma = 0$-, the second uses a list containing only LRS mean values.

Every time that the low-bit precision training has to be performed, it must be preceded by a32-bit precision training -i.e. without any constriction on weight resolution- so that a

Algorithm 4.7 $low\_bit(w, w\_old, lrs\_list, std\_list)$

```
Input: weight array after back-propagation $w$; weight array of previous sampling $w\_old$;
    array containing the means for the Gaussian sampling $lrs_list$ with length equal to 8;
    array containing the std for the Gaussian sampling $std\_list$ with length equal to 8;
Output: new weight array with low-bit resolution $w$
1: $mean$ = array of the closest mean to each weight with the same shape of $w$
2: $old\_mean$ array of the closest mean to each weight in $old\_w$
3: for $i \leftarrow 1$ to 8 do
4:     $n$ = take the number of elements for which $mean \neq old\_mean$ and $mean = lrs\_list[i]$
5:     $new\_w = normal(lrs\_list[i], std\_list[i], size = n)$    ▷ sample $n$ element from normal
6:     update every $w$ that changed distribution from $old\_mean$ to $mean = lrs\_list[i]$ posing it equal to values in $new\_w$
7:     all the $w$ values not updated must be posed equal to $old\_w$
8: end for
9: return $w$
```

magnitude scale for weights is obtained, then the lists of means and std for the distribution must be scaled by a quantity equal to

$$s_w = \frac{\max(lrs)}{max(w)}$$

this will ensure that PyTorch built-in algorithm are able to train the network without modifying training hyperparameters each time that the distributions for LRS are changed. The scaling factor s_w is then saved for doing hardware-aware inference.

4.3.4. Training loop and inference

The training loop [Algorithm 4.8] is defined implementing all the above methods.

It is important to notice to notice that PyTorch simulates and train the network basing only on its dynamics, so the networks parameters will be set to:

- neuron spiking threshold $\vartheta^{\mathrm{d}} = 1\,\mathrm{V}$;
- input spiking amplitude $V^{\mathrm{d}}_{in,ADC} = 1\,\mathrm{V}$;
- $R^{\mathrm{d}}_n = 1\,\Omega$from (4.3);
- weights arbitrarily scaled by PyTorch;
- $t^{\mathrm{d}}_s = 1/f_{sampling}$;

where the index d refers to the quantities used for dynamics-only simulations.

This implies that for the inference phase all these factors must be reset to the chip values. Thus an analysis of currents and voltages for going back to hardware-aware coding should be done.

Algorithm 4.8 $training\_loop(..., w\_scale, x\_in, labels, n\_epochs, tolerance\_thr, f)$

Input: $network\_call$ requirements [Algorithm 4.5]; low_bit requirements [Algorithm 4.7]; w_scale for weight initialization; $n\_samples$ spike trains divided into $n\_batches$ batches x_in with shape $(n\_samples/n\_batches \times n\_batches)$; $n\_samples$ labels $labels$ divided into $n\_batches$ batches with shape $(n\_samples/n\_batches \times n\_batches)$; number of epochs n_epochs; threshold tolerance $tolerance\_thr$ for spiking ; frequency for low-bit f

Output: trained weights array w; loss history $loss\_hist$

```
1:  $loss\_hist = zeros[n\_epochs]$
2:  $w = normal(0, w\_scale/\sqrt{n\_ch}, shape = n\_syn \times n\_ch)$   ▷ initialize random array
3:  for $i \leftarrow 1$ to $n\_epochs$ do
4:      $loss\_local = 0$
5:      for $j \leftarrow 1$ to $n\_batches$ do
6:          $out = network(x\_in[\text{all}][j], w, ...)$   ▷ shape is $(n\_samples/n\_batches \times t\_steps + d\_max)$
7:          sum $out$ spikes on the time step axis
8:          $out = out - tolerance\_thr$
9:          clamp $out$ minimum to 0
10:         $loss = BCEWithLogitsLoss(out, labels)$   ▷ PyTorch built-in function
11:         $loss\_local = loss\_local + loss$
12:         back-propagation of $loss$ on $w$
13:         $w = w\_clamping(w)$
14:     end for
15:     if $i$ is a multiple of $f$ then
16:         $w = low\_bit(w, ...)$
17:     end if
18:     $loss\_hist[i] = loss\_local/n\_batches$
19: end for
20: return $w, loss\_hist$
```

If it is considered that the spiking threshold ϑ remains constant from the dynamics simulation to the chip scaling, it is clear that a current normalizer should be inserted in the chip just before the neuron in order to scale linearly the current up to a maximum value given by

$$I_{norm,MAX} = \frac{I^{\text{d}}_{MAX}}{R_n s_w V_{in,ADC}} \tag{4.5}$$

where at the numerator $V^{\text{d}}_{in,ADC}$ and R^{d}_n have been omitted being equal to a unit value. Such a current limiter make advantage of an $I_{out} - I_{in}$ characteristic which is linear in the region comprised between 0 A and a given values after which the current saturates to a maximum value $I_{norm,MAX}$.

For what concerns the time scaling, it is possible under certain condition as explained in section 4.4.

4.4. Results

In order to demonstrate that the network is able to learn what are the right delays in order to recognize features in temporal signals for real-time analysis the MIT-BIH dataset[64] has been used in different ways.

During years, a lot of works have been presented on this dataset for recognizing and classifying arrhythmia and heartbeat, unfortunately the entire dataset requires a lot of medical competences to be used in its standard division[42]. Indeed, the dataset is not very clear to scientists with non-medical background because the standard notation used for labeling the dataset is very complicated and leads to different interpretations of labels in different works. Moreover, problems on the segmentation of the dataset with fixed length occurs when considering different patients because of singular temporal properties of ECG records[11]. Finally, being the dataset very unbalanced, it is very hard to train a network on it, even if this problem can be solved with oversampling algorithm such as SMOTE[26] it does not lead to results comparable to single patient even on CNN artificial networks[74].

In this work, the dataset will always be used for doing binary classification between arrhythmia and normal beats which are classified by the LIF neuron as follows:

- no spikes corresponds to normal beat;
- one or more spikes correspond to arrhythmia.

All the non-idealities of memristors are controlled by setting a different seed for random samplings that must be performed before each different training and simulation.

During the inference phase of all the results presented in the following, the values are re-scaled to the chip values accordingly to what is explained in section 4.3 except for the time. Indeed, because of the very low sampling frequency for the digitization of the dataset -$f_{dataset} = 360\,\text{Hz}$- due to the relative slow variations of ECG signals, the computational power required to use a time step comparable with the real one would be too high for a normal computer, but it has been seen that decreasing the time step up to $1/(10f_{dataset})$ the results are stable. For the sake of completeness, the simulation for both training and testing phases are done using a time step for time discretization equal to $1/t_{dataset}$, which corresponds to use spikes whose pulse width is equal to $\sim 2.78\,\text{ms}$.

4.4.1. Single Patient classification

The first and most complete analysis has been done on patient 208 which is the most balanced in the dataset and whose labels are divided into arrhythmia and normal beats accordingly to Tab. 4.1.

Class	0	1
Label	'L' 'R' 'N'	'e' 'j' 'A' 'a' 'J' 'S' 'V' 'E' 'F' '/' 'f' 'Q'
Samples	1369	1586

Table 4.1: Labels and samples for arrhythmia classification of patient 208

At first, the dataset is segmented around the R peak of the heartbeats using 125 time steps before and after ignoring the segments which are incomplete because at the beginning or at the end of the record and the segment which includes more than one R peak (heart beat peak); with this approach, the number of samples per classes reduces to 1585 class 0 and 1368 class 1. Then, three different sets are created:

- train set: first 1476 samples;
- test set: last 1477 samples;
- validation set: first 96 and last 128 samples of the test set.

The delta modulation for converting the analog input to spike trains is performed using a threshold:

$$\delta V = \frac{|0.991\max(A_{ch}) - 0.009\max(A_{ch})|}{10}$$

where $ch \in [ch_1, ch_2]$ is indicating the corresponding electrode and A_{ch} the amplitude of the record of electrode ch.

The network simulated is composed of 4 channel which receive the delta-modulated output of electrode 1 and 2 of patient 208, a DPI with $\tau_{DPI} = 10\,\text{ms}$ and $g_{DPI} = 1$ and membrane time constant $\tau_{mem} = 10\,\text{ms}$.

The training set is used to train the network for 15 epochs with $lr = 10^{-3}$ and only positive weights constraint [Algorithm 4.6] for understanding what is the minimum mean delay D_μ -i.e. μ of the log-normal- required for obtaining a good accuracy and what is the effect of device variability, i.e. how changes the accuracy when varying the σ of the underlying Gaussian for log-normal sampling of $R_{i,j}$, even if the reference is considered $\sigma = 0.4$[31].

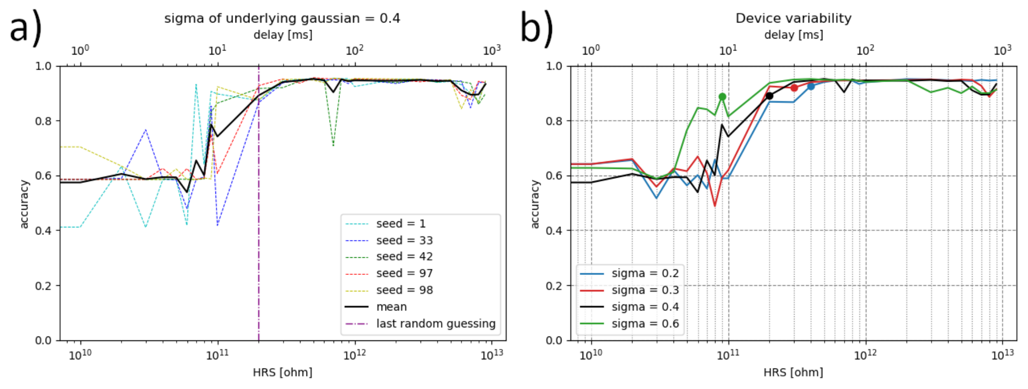


Figure 4.4: a) accuracy variation with respect to mean delay from sampling changing μ and $\sigma = 0.4$; b) mean accuracy over 5 seeds by varying both μ and σ, the dots are indicating the D_μ required for the network to stabilize its accuracy on all the seeds

Finally, 5 different seeds are used to study the device non-idealities. Results show as there is a window of mean delay in which the accuracy is ~ 0.95, this window for $\sigma = 0.4$ [Fig. 4.4a] start at a mean delay $D_\mu = 50\,\text{ms}$ -i.e. $\mu = 5 \cdot 10^{11}\,\Omega$- and is shifted to lower μ values when σ increases [Fig. 4.4b]. The random guessing is defined when the accuracy difference between the training and test accuracy is larger than 0.2 and it is considered ended when all the networks with different seeds start learning properly.

Analysing the weight values, it is possible to see what are the delays chosen by the network after the training. For $\mu = 5 \cdot 10^{11}\,\Omega$, the delays picked are at the two extremities corresponding to a difference in time comprised in the interval $D_0 - D_1 \in [80, 135]$ ms; this choice of delays is common to higher mean delays such as $D_\mu = 100$ ms [Fig. 4.5]. On the other hand, the weights chosen on the dendrites corresponding to the second electrode are very low and does not show any difference in the delay choice.

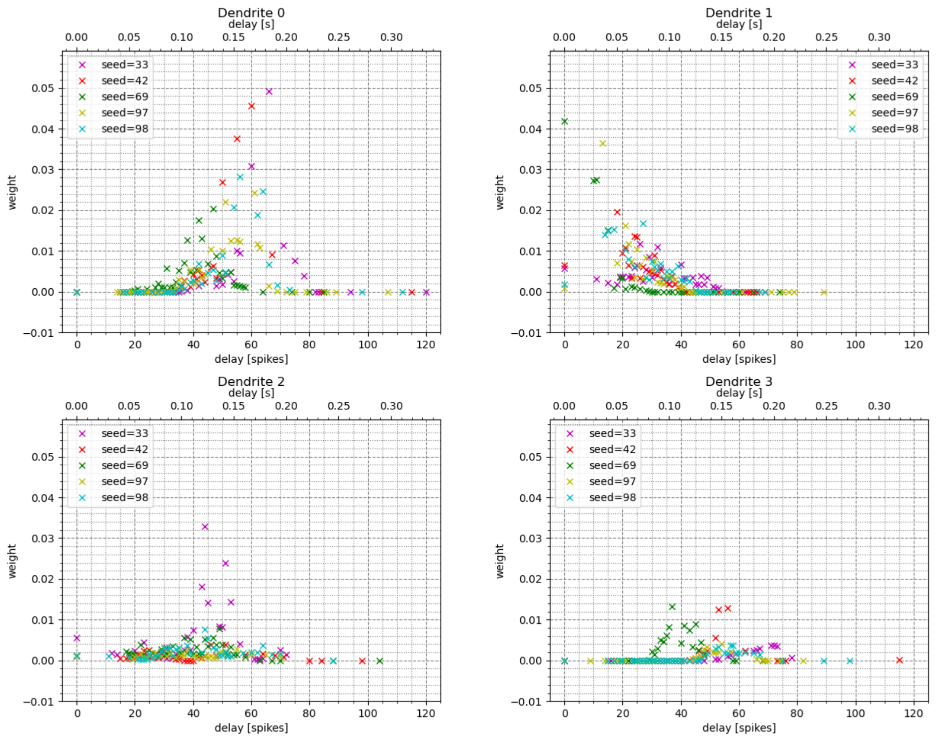


Figure 4.5: Weights values for each delay on the dendrites after the training phase without low-bit resolution; the mean delay used is $D_\mu = 100$ ms because for $D_\mu = 50$ ms the difference between D_0 and D_1 would not be perceptible as stable since the values are at the extremities of the distributions

Once that is demonstrated that the network can learn correctly reaching an accuracy $> 95\,\%$, the low-bit constraint must be applied [Algorithm 4.7] in order to see what are the effects on the results.

Moreover, to be more hardware-aware, it is investigated what happens if the DPI is removed from the network architecture since the simulated chip will not include such a circuit

Taking as reference $D_\mu = 50$ ms, a learning routine based on the following sequence is scheduled:

1. n_{epochs} with 32-bit precision for bit and only positive weights constraint obtaining $W_{trained}$;
2. scaling of LRS Gaussian parameters by $s_w = \max LRS / \max W_{trained}$;
3. int($n_{epochs}/2$) with 3.17-bit precision, i.e. using 8 μ values for LRS sampling plus HRS state corresponding to $\mu = 0$ S and $\sigma = 0$ [Fig 2.7a], and positive weights constraints;

4. n_{epochs} with 3-bit precision, i.e. only LRS values[Fig 2.7a] scaled by s_w, and positive weights constraints.

Then, it has be demonstrated in past works that one electrode is sufficient to perform classification[42], since the second electrode is not always the same, and that a segmentation of 180 time steps around the R peak can be considered a sort of optimal fixed length segmentation[74]. As a consequence, these changes to the architecture have been made:

- segmentation of 180 time step around R peak instead of 250 time steps;
- DPI removal from the architecture;
- D_2 and D_3 exclusion in order to use only one electrode.

These changes led to a study over 15 seeds to analyze what is the accuracy behavior when passing to more complex systems to a simpler one with optimizations on signal processing for training; moreover, the study is also used to find the best learning routine for the new consideration.

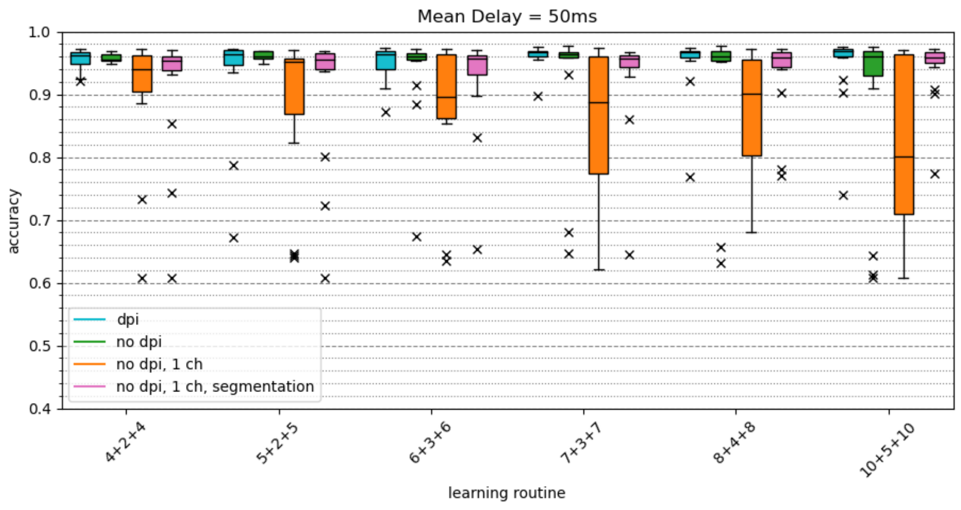


Figure 4.6: Box plots of different accuracy obtained by different architectures with different signal processing optimizations vs. the learning routine scheduled

Learning routine	Mean accuracy	Median accuracy	[min, max] accuracy range
8+4+8	0.9324	0.9594	[0.7698, 0.9729]

Table 4.2: Best learning routine result in terms of accuracy

Basing on the median, mean and [minimum, maximum] range or all the accuracy results on the 15 seeds [Fig. 4.6], the best model is obtained with a learning routine made of 8+4+8 [Tab. 4.2]. They show not only that the synapse dynamics is not influencing the performance of the network and that despite memristors non-idealities the accuracy is still good, but that when passing to one electrode (which is mandatory for the use of more than one patient) the signal processing becomes very important; this increase the initial argument about the susceptibility of the dataset to signal process engineering.

Real-time analysis

The real-time behaviour of the network must be analyzed, before passing to power consumption analysis. This task is executed on the entire record of patient 208 (without respecting the division in training and test set). Differently from segmented classification in which one heartbeat per time is passed to the network and classified, non-centered time windows of different length of the record are passed to the network, then the spike output is analyzed and if one or more spikes are found in the interval $[t_{peak} - w_l/2 + \text{mean}(D_{i,j}), t_{peak} + w_l/2 + \text{mean}(D_{i,j})]$ -w_l indicates half of the windows used for segmentation-, the heartbeat occurring at time t_{peak} is labeled as belonging to class 1, otherwise to class 0 [Fig. 4.7].

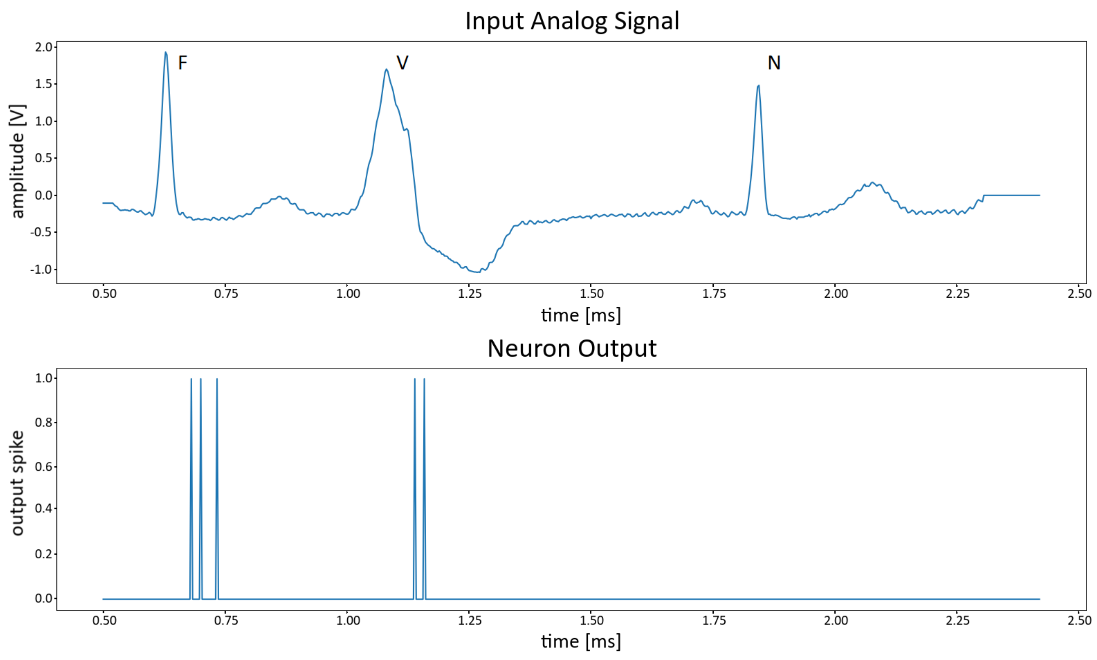


Figure 4.7: Example of real-time classification on a window with 650 time steps length

In this way, a model trained on segmented heartbeats is used for doing inference o non-centered windows of increasing length to test the stability of the network for real-time classification basing on the mean values of:

$$\text{Accuracy} = \frac{t^+ + t^-}{t^+ + t^- + f^+ + f^-}$$

$$\text{Predictivity} = \frac{t^+}{t^+ + f^+}$$

$$\text{Sensitivity} = \frac{t^+}{t^+ + f^-}$$

$$\text{Specificity} = \frac{t^-}{t^- + f^+}$$

obtained on 5 seeds. Here t^+ indicates true positives, t^- true negatives, f^+ false positives and f^- false negatives.

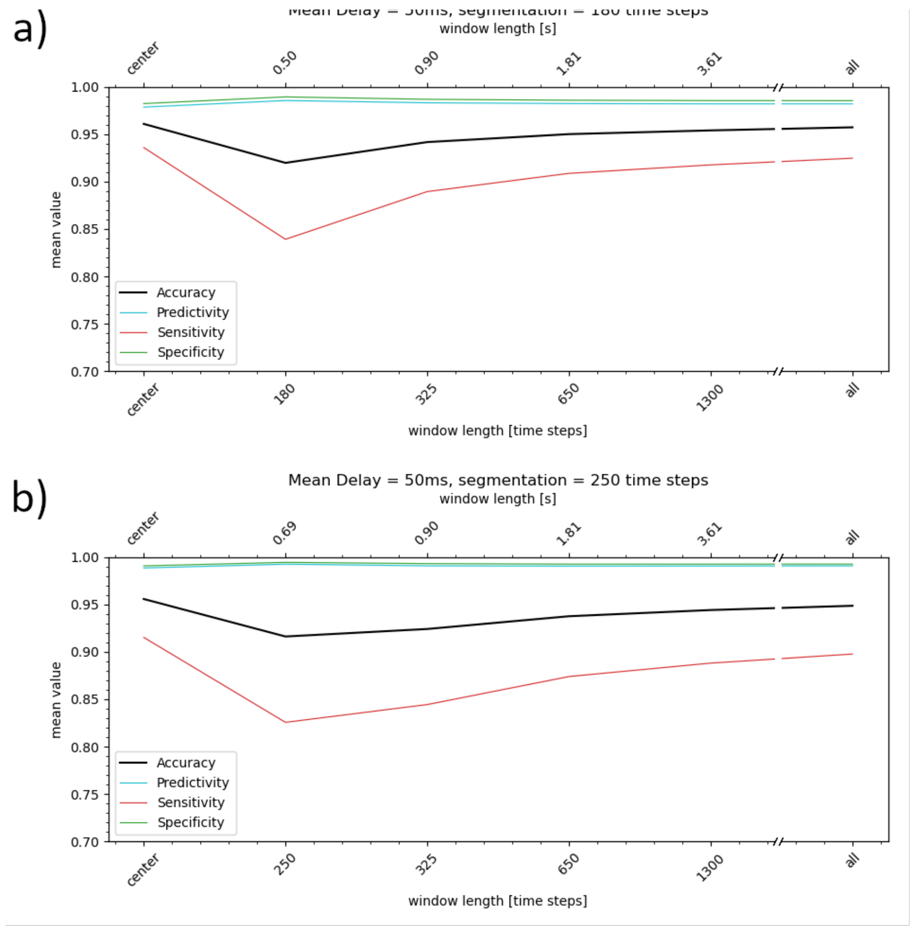


Figure 4.8: a) accuracy, predictivity, sensitivity and specificity of a model trained on segment of 180 time step length vs window length; b) accuracy, predictivity, sensitivity and specificity of a model trained on segment of 250 time step length vs window length

When the network is feed with very short samples which are not centered, accuracy and sensitivity drops while predictivity and specificity raise a little bit because of insufficient number of spikes for making the neuron spiking in case of a positive label due to a missing part of the heartbeat in the input window. Indeed, this will cause less class 1 labels and a consequent drop of f^+ and t^+ with a raise of t^- and f^-.

While the window length increases, the network output stabilize up to results comparable to the segmented classification [Fig. 4.8] when the window length equals the entire record of 30 min. Once again, th processing of the signal used for training the network is very important.

This result further emphasizes how the SNN is able to learn temporal features of the signal by using a short-term-like memory generated by the delays, instead of other kinds of properties of the signal as an ANN could do.

Power and energy consumption

The power and energy consumption is computed on real time inference with the window length equal to the entire record of patient 208.

Referring to formulas in section 4.2, all the power consumption contributes are computed doing the mean over 5 seeds. The ADC power remains unchanged over the seeds and it will be the greatest consumption in the network:

$$P_{ADC} = 55\,\mu\text{W}\frac{306\,\text{Hz}}{100\,\text{Hz}} = 198\,\mu\text{W}$$

For the delays a read voltage $V_{D,read} = 0.5\,\text{V}$ is used and results to be negligible with respect to other contribution since the resistances for the RC circuits are very high:

$$P^D = \sum_{i=0}^{1}\sum_{j=0}^{63}\sum_{k=0}^{n_{steps}^{\text{tot}}} \frac{CV_{D,read}^2(t_k)}{t_{208}^{\text{tot}}}\left(1 - e^{-\frac{\tau_s}{R_{i,j}C}}\right)$$

The weights power is not negligible and reads:

$$P^W = \sum_{i=0}^{1}\sum_{j=0}^{63}(V_{i,j}^{\text{RMS}})^2 W_{i,j}$$

with the read voltage for weights equal to 0.1 V. It is influenced by the pulse width of the incoming spike -i.e. read voltage- since the RMS value for a sequence of spikes would decrease by decreasing the pulse width in time; in the above equation the pulse width is kept constant and equal to the time step of time discretization -which is equal to $1/f_{dataset} \sim$ 2.78 ms- and it results to be bigger than a real pulse width of five order of magnitude. This means that, in the following, the weight power consumption will be overestimated.

Finally, the LIF neuron power consumption is too low to be considered in the computation:

$$P_{LIF} \ll 1\,\text{nW}$$

The mean results over the seeds are summed to obtain the overall power consumption. Here, for the sake of completeness, the power consumption of signal conversion to spikes is reported, but the comparison with other works [Tab. 4.3] is done only on the engineered part, i.e. on weights and delays, since the digitization power is never computed.

The energy consumption is computed referring to the method explained in section 4.2 and taking into consideration the real-time classification of patient 208 on 30 min continuous record.

For the comparison with other works, the quantity that has been taken into account is the energy per classification -when given- considering the model performing a classification on the number of labels indicated in Tab. 4.1. Otherwise, it is obtained just by considering the network as it is performing real time classification and computing:

$$E_{work} = P_{work} \cdot t_{208}^{\text{tot}}$$

The perceptron-like SNN is able to outperform state of art works both in terms of power and energy consumption, despite the computation based on real-time classification instead of more convenient segment-based classification. Compared to recurrent SNN architecture implemented on neuromorphic analog chips, the power consumption is three order of magnitude lower[39] and the difference increases up to seven order of magnitudes when looking at CNN[74]. Even when compared to very engineered systems[16], the power consumption is one order of magnitude lower and very promising.

When looking at energy consumption, differences seen in power consumption hold, with the exception of power consumption with non-real-time systems in which the inference time is significantly decreased.

Work	This work	[16]	[39]	[74] SNN	[74] CNN
Power	0.53 µW	48.6 µW	516.1 µW	64 mW	8.94 W
Energy	965.78 µJ	6.64 mJ	931.85 mJ	116 J	16 kJ

Table 4.3: Energy and power consumption comparison with the state of art

The extension on other patient for such a task is not straight forward since in the dataset there are some labels which are not used for the standard classification[42] but can be classified as arrhythmia and makes the neuron spiking. E.g., label '!' is indicating a ventricular flutter wave and has a very high spike encoding frequency making the network spiking but -since there is not a standard classification for it- the classification periods containing '!' labels cannot be evaluated. Anyway, the network is able to learn on patients with more than 5 arrhythmia labels using the SMOTE algorithm for balancing the training set; some problems occur with the classification of SVEB arrhythmia [Tab. 4.4] as in other works using fixed length segmentation for training[11].

To sum up, patient 208 is used for having a proof of concept of the delay-based dendrites utility for real-time classification, but deeper studies on other patients can be done in the future.

4.4.2. Multi patient classification

A second proof of concept about the capacity of the network to recognize temporal features after a proper training can be done on a dataset made of data from more patients inside the MIT-BIH dataset.

Because of the very complicated labeling system used on ECG signals[68], people which are non-medical doctor make a lot of fatigue to read it. This fact makes binary classification very hard to do, because the distinction between normal and abnormal signals should be done and this could lead to mistakes when the labels given by the source are not clear.

In the following, in order to compensate the unbalanced properties of the dataset and to have the less signal processing possible, a subsampled dataset is used by dividing the labels into five main groups -following the labels division of Tab. 4.4- and random sampling 1000 samples for each group except for group F which has in total 802 samples.

The samples are then segmented using 180 or 250 time steps and normalized between $[0, 1]$ V. It is important to notice that each sample is normalized instead of the entire signal. This action enables the sigma-delta modulator to have a stabler input signal, otherwise due to the differences of records between patients, there could be some cases in which the spike frequency is too high or too low for the network to recognize features.

For the sigma-delta modulation, a fixed value $\delta V = 50$ mV has been chosen and then the spike trains are generated.

Finally, three set are built:

- 70 % of samples is used for the train set;

Class	0	1			
Group	N	SVEB	VEB	F	Q
Labels	'L' 'R' 'N'	'e' 'j' 'a' 'A' 'J' 'S'	'V' 'E'	'F'	'/' 'f' 'Q' '"' '!' ' ' '|' '+' 'x' '[' ']'
Samples	90333	3024	7235	802	11149

Table 4.4: Labels and samples for binary classification on multi-patient dataset

- 10 % of samples is used for the validation set;
- 20 % of samples is used for the test set;

The network used to solve this task does not include the DPI, but has 2 dendrites and $\tau_{mem} = 10\,\text{ms}$. For the delays sampling log-normal mean is $\mu = 10^{12}\,\Omega$and underlying Gaussian $\sigma = 0.4$ which correspond to a mean delay on dendrites $D_\mu = 100\,\text{ms}$; this choice allow to have a longer short term memory thanks to the fact that $D_{0,j} = 0\,\text{S}$, i.e. memristor in LRS.

The training is performed using segmented signals made of 180 time steps and 250 time steps around the R peak for each sample and searching for an optimized learning routine with a structure equal to the one presented in the previous section taking into account the mean, median and [minimum, maximum] range of accuracy, sensitivity and predictivity. The study of non-idealities has been done over 15 different seeds.

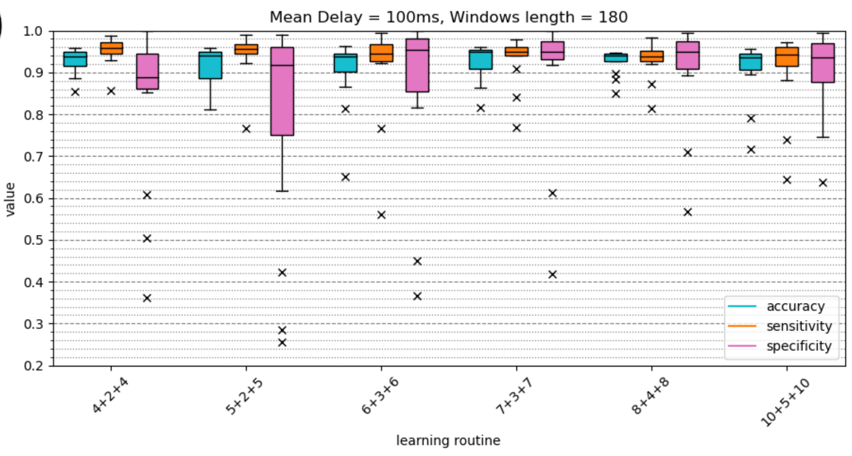
Characteristic	Mean	Median	[min, max] range
segmentation= 180 time steps, 8+4+8 epochs			
Accuracy	0.9268	0.9396	[0.8500, 0.9479]
Sensitivity	0.9310	0.9372	[0.8128, 0.9830]
Specificity	0.9106	0.9490	[0.5663, 0.9949]
segmentation= 250 time steps, 5+2+5 epochs			
Accuracy	0.9301	0.9406	[0.8083, 0.9677]
Sensitivity	0.9362	0.9437	[0.7657, 0.9843]
Specificity	0.9065	0.9490	[0.5867, 1.0]

Table 4.5: Results of the best learning routines for the two different fixed length segmentation windows used for training on the multi patient dataset

The highest variations due to devices non-idealities are related to specificity which is the most variable parameter as in other works[11] and the difference with other parameters is a little bit more evident when the segmentation is 250 time steps; this underlines again how much the dataset is susceptible to preprocessing changes and optimization which is not an aim of this work.

Looking at the results [Fig. 4.9], a learning routine for the two segmentation length is chosen and analyzed in Tab. 4.5.

For this subset of the MIT-BIH dataset has not been possible to do real time inference because of the random sampling used for constructing it. Instead, a classification per

a)


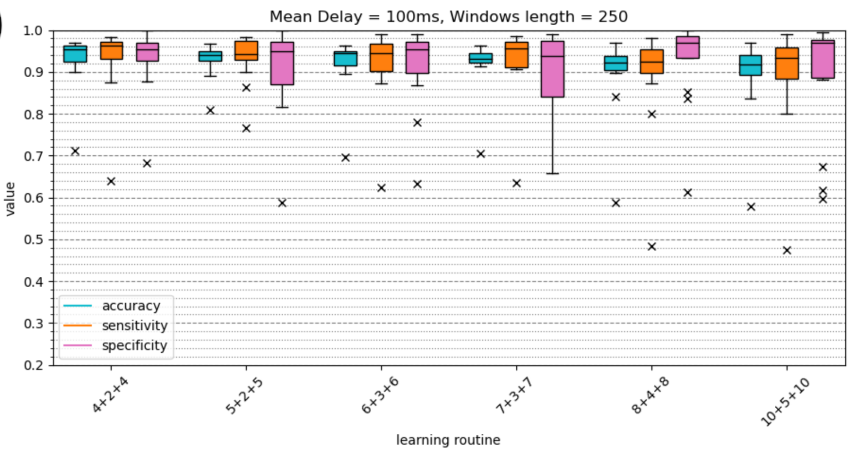
b)


Figure 4.9: a) accuracy, sensitivity and predictivity variation with respect to the learning routine when using a segmentation of 180 time steps; b) accuracy, sensitivity and predictivity variation with respect to the learning routine when using a segmentation of 250 time steps

segment has been used for evaluating the network performance and computing the power and energy consumption.

This last characteristics allows to estimate power and consumption accordingly to formulas in section 4.2 by keeping into account that:

- $V^2_{D,read}(t_k)$ is the mean over samples for a single seed;
- $V^{\text{RMS}}_{i,j}$ is the mean over samples for a single seed;
- the energy consumption obtained will be an energy per classification.

The read voltages for delays and weights are kept unchanged with respect to the single patient analysis and thus is 0.5 V for the delays and 0.1 V for weights.

The sigma-delta modulator consumption remains the same from the previous analysis and has not been considered as explained in the previous section.

When the energy consumption per classification in other works is not indicated, it was computed by:

$$E_c = P \cdot t_{seg}$$

Work	This work	[16]	[39]	[74] SNN	[74] CNN
Power	0.47 µW	48.6 µW	516.1 µW	64 mW	9.94 W
Energy per classification	80.78 nJ[1] 88.23 nJ[2]	2.25 µJ	258.08 µJ	32 mJ	4.47 J

[1]segmentation = 180 time steps

[2]segmentation = 250 time steps

Table 4.6: Power and energy per classification consumption comparison with the state of art

where E_c indicates the energy per classification and t_{seg} is the segmentation time used for a single classification, e.g. $t_{seg} = 180\tau_s$ when using a segmentation equal to 180 time steps. Concerning the current work, two results are given based on two difference segmentation length. If the segmentation length in other works is not presented, is considered to be 180 time steps.

Compared with the state of art, the SNN is still very well behaving. Indeed, the energy consumption is showing even better results due to the fact that there is less signal between one classification and another because of real-time absence with respect to Tab. 4.3. On the other hand, the energy consumed by the segmentation is not computed because such a circuit has not been implemented in this work.

The power consumption is s little bit lower than the single patient one meaning either that the selected weights are lower in conductance or that are sparser with respect to the selection of delays; another possible explanation could be a sparser input spike train which would decrease V^{RMS}.

5. Conclusions

The state-of-art for what concerns NNs with a main focus on the spiking neuromorphic implementations for analyzing temporal signals has been presented. Always keeping in consideration the strict relation that occurs between the real brain and such applications, the bio-plausibility of these system has been explained with a comparison between the different and more advanced chips.

Then, a second focus on the developing memristor technologies for implementing existing models -like synapses- in a different and more efficient way has been reported. The different materials, behaviors and integration have been analyzed by starting from the memristor definition and arriving to the OxRAM cell characterization.

The union between these two introductory topics has led to the thesis project that is focused on the possible explanation for multi-synapse connections observed in the brain between two neurons by supposing that they can be used to generate a short-term memory by means of time delays to the input signal. Indeed, it has been demonstrated that the perceptron-like network is able to keep a short-term memory by means of memristor generated delays by learning through back-propagation deep learning algorithms what is the best delay for each dendrite for keeping only relevant features. Such a choice is made through memristor implemented synapses which are trained following the characterization proper of this technology. In addition, the importance of memristor characterization in both LRS and HRS -up to pristine state- has been showed, implicating a growth of the importance of such devices not only for memory application but also for analog circuits.

Moreover, the implementation of the network through hardware-aware codes allowed to simulate what would be its behavior during real-time classification of time-varying signals like ECG. This step showed that results comparable to the state-of-art -in terms of accuracy, predictivity and sensitivity- can be obtained on single patient classification and on multi patient classification by random sampling among the entire MIT-BIH dataset.

All these results have been obtained with ultra-low power and energy consumption, also when compared with highly engineered systems belonging to the state-of-art. Thanks to the large use of memristor technology, also scalability and integrability reached a very good level as the decrease of memory element per chip.

5.1. Future works

Such a new architecture could represent a novel way not only to implement short-term, but also to explain realistic brain behaviors and to make memristor devices interesting for new fields of application outside the classic synapse and memory implementations. On the coding side, an on-chip characterization will surely help to understand what is the reliability of hardware-aware codes and maybe push deep learning libraries such as PyTorch and TensorFlow to open more to this programming paradigm, maybe starting from the building of integrated algorithms for memristor simulation and learning.

The architecture is able to learn and solve nonlinear tasks on MIT-BIH dataset, but it is just a starting point for developing new and more complicated architectures that can be applied to different dataset. Indeed, an immediate consequence of this thesis can be the application of the one neuron network to different kind of signals by changing the mean delays for the dendrites distributions showing the flexibility of the network.

Then, the development of more complicated multi-layers architectures can show what are the implication of such a multi-synapse connection between neurons when applied to inter layer synapses. Furthermore, the development of more complicated structures, or the application of the multi-synapse connection concept to existing architectures, could show what is the effective computational power introduced.

In the end, trying to exploit both delays and recurrent connections for having different levels of short-term memory could give vaster comprehension of memory behavior and generation in SNNs.

References

[1] W. C. Abraham and M. F. Bear. “Metaplasticity: the plasticity of synaptic plasticity”. In: *Trends in Neurosciences* 19.4 (1996), pp. 126–130. DOI: 10.1016/S0166-2236(96)80018-X.

[2] A. G. Andreou et al. “Real-time sensory information processing using the TrueNorth Neurosynaptic System”. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 2911–2911. DOI: 10.1109/ISCAS.2016.7539214.

[3] A. Parmezan et al. “Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model”. In: *Information Sciences* (2019). DOI: 10.1016/j.ins.2019.01.076.

[4] A. Rubino et al. “Ultra-Low-Power FDSOI Neural Circuits for Extreme-Edge Neuromorphic Intelligence”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.1 (2021), pp. 45–56. DOI: 10.1109/TCSI.2020.3035575.

[5] B. Cramer et al. “The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–14. DOI: 10.1109/tnnls.2020.3044364.

[6] C. Bartolozzi et al. “An ultra low power current-mode filter for neuromorphic systems and biomedical signal processing”. In: *2006 IEEE Biomedical Circuits and Systems Conference*. 2006, pp. 130–133. DOI: 10.1109/BIOCAS.2006.4600325.

[7] E. Chicca et al. “Neuromorphic Electronic Circuits for Building Autonomous Cognitive Systems”. In: *Proceedings of the IEEE* 102.9 (2014), pp. 1367–1388. DOI: 10.1109/JPROC.2014.2313954.

[8] E. Esmanhotto et al. “High-Density 3D Monolithically Integrated Multiple 1T1R Multi-Level-Cell for Neural Networks”. In: *2020 IEEE International Electron Devices Meeting (IEDM)*. 2020, pp. 36.5.1–36.5.4. DOI: 10.1109/IEDM13553.2020.9372019.

[9] E. R. Hsieh et al. “High-Density Multiple Bits-per-Cell 1T4R RRAM Array with Gradual SET/RESET and its Effectiveness for Deep Learning”. In: *2019 IEEE International Electron Devices Meeting (IEDM)*. 2019, pp. 35.6.1–35.6.4. DOI: 10.1109/IEDM19573.2019.8993514.

[10] E. Vianello et al. “Metal Oxide Resistive Memory (OxRAM) and Phase Change Memory (PCM) as Artificial Synapses in Spiking Neural Networks”. In: *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2018, pp. 561–564. DOI: 10.1109/ICECS.2018.8617869.

[11] F. Li et al. “Automated Heartbeat Classification Exploiting Convolutional Neural Network With Channel-Wise Attention”. In: *IEEE Access* 7 (2019). DOI: 10.1109/ACCESS.2019.2938617.

[12] G. Indiveri et al. “Neuromorphic Silicon Neuron Circuits”. In: *Frontiers in Neuroscience* 5 (2011). DOI: 10.3389/fnins.2011.00073.

[13] G. Tanaka et al. “Recent advances in physical reservoir computing: A review”. In: *Neural Networks* 115 (2019), pp. 100–123. DOI: 10.1016/j.neunet.2019.03.005.

[14] G. W .Burr et al. “Phase change memory technology”. In: *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena* 28.2 (2010), pp. 223–262. DOI: 10.1116/1.3301579.

[15] J. K. Eshraghian et al. “Training Spiking Neural Networks Using Lessons From Deep Learning”. In: *arXiv* (2021). DOI: 10.48550/ARXIV.2109.12894.

[16] J. Liu et al. “4.5 BioAIP: A Reconfigurable Biomedical AI Processor with Adaptive Learning for Versatile Intelligent Health Monitoring”. In: *2021 IEEE International Solid- State Circuits Conference (ISSCC)*. Vol. 64. 2021, pp. 62–64. DOI: 10.1109/ISSCC42613.2021.9365996.

[17] K. Potdar et al. “A comparative study of categorical variable encoding techniques for neural network classifiers”. In: *International journal of computer applications* 175.4 (2017), pp. 7–9. DOI: 10.5120/ijca2017915495.

[18] L. Appeltant et al. “Information processing using a single dynamical node as complex system”. In: *Nature communications* 2.1 (2011), pp. 1–6. DOI: 10.1038/ncomms1476.

[19] L. Sun et al. “In-sensor reservoir computing for language learning via two-dimensional memristors”. In: *Science Advances* 7.20 (2021). DOI: 10.1126/sciadv.abg1455.

[20] M. Baker et al. “State of the art of metal oxide memristor devices”. In: *Nanotechnology Reviews* 5.3 (2016), pp. 311–329. DOI: doi:10.1515/ntrev-2015-0029.

[21] M. Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359.

[22] M. Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359.

[23] M. Sharifshazileh et al. “An electronic neuromorphic system for real-time detection of high frequency oscillations (HFO) in intracranial EEG”. In: *Nature communications* 12.1 (2021), pp. 1–14. DOI: 10.1038/s41467-021-23342-2.

[24] M. Z. Alom et al. “A State-of-the-Art Survey on Deep Learning Theory and Architectures”. In: *Electronics* 8.3 (2019). DOI: 10.3390/electronics8030292.

[25] N. Dennler et al. “Online detection of vibration anomalies using balanced spiking neural networks”. In: *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE. 2021, pp. 1–4.

[26] N. V. Chawla et al. “SMOTE: Synthetic Minority Over-sampling Technique”. In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357. DOI: 10.1613/jair.953.

[27] P. A. Merolla et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* 345.6197 (2014), pp. 668–673. DOI: 10.1126/science.1254642.

[28] S. Amer et al. "A practical hafnium-oxide memristor model suitable for circuit design and simulation". In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050790.

[29] S. Moradi et al. "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)". In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1 (2018), pp. 106–122. DOI: 10.1109/TBCAS.2017.2759700.

[30] S. P. Adhikari et al. "Three Fingerprints of Memristor". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 60.11 (2013), pp. 3008–3021. DOI: 10.1109/TCSI.2013.2256171.

[31] T. Dalgaty et al. "Hybrid neuromorphic circuits exploiting non-conventional properties of RRAM for massively parallel local plasticity mechanisms". In: *APL Materials* 7.8 (2019). DOI: doi.org/10.1063/1.5108663.

[32] T. P Lillicrap et al. "Backpropagation and the brain". In: *Nature Reviews Neuroscience* 21.6 (2020), pp. 335–346. DOI: 10.1038/s41583-020-0277-3.

[33] W. Guo et al. "Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems". In: *Frontiers in Neuroscience* 15 (2021). DOI: 10.3389/fnins.2021.638474.

[34] W. Rall et al. "Matching dendritic neuron models to experimental data". In: *Physiological Reviews* 72.suppl_4 (1992), S159–S186. DOI: 10.1152/physrev.1992.72.suppl_4.S159.

[35] Y. Demirag et al. *Online Training of Spiking Recurrent Neural Networks with Phase-Change Memory Synapses*. 2021. DOI: 10.48550/ARXIV.2108.01804.

[36] Yibo Li et al. "Review of memristor devices in neuromorphic computing: materials sciences and device challenges". In: *Journal of Physics D: Applied Physics* 51.50 (2018). DOI: 10.1088/1361-6463/aade3f.

[37] V. Balasubramanian. "Brain power". In: *Proceedings of the National Academy of Sciences* 118.32 (2021). DOI: 10.1073/pnas.2107022118.

[38] C. Bartolozzi and G. Indiveri. "Synaptic Dynamics in Analog VLSI". In: *Neural Computation* 19.10 (2007), pp. 2581–2603. DOI: 10.1162/neco.2007.19.10.2581.

[39] F. C. Bauer et al. "Real-Time Ultra-Low Power ECG Anomaly Detection Using an Event-Driven Neuromorphic Processor". In: *IEEE Transactions on Biomedical Circuits and Systems* 13.6 (2019), pp. 1575–1582. DOI: 10.1109/TBCAS.2019.2953001.

[40] R. Brette and W. Gerstner. "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity". In: *Journal of Neurophysiology* 94.5 (2005), pp. 3637–3642. DOI: 10.1152/jn.00686.2005.

[41] J. Carey. *Brain facts: A primer on the brain and nervous system.* Society for Neuroscience, 1990.

[42] P. de Chazal, M. O'Dwyer, and R.B. Reilly. "Automatic classification of heartbeats using ECG morphology and heartbeat interval features". In: *IEEE Transactions on Biomedical Engineering* 51.7 (2004), pp. 1196–1206. DOI: 10.1109/TBME.2004.827359.

[43] L. O. Chua. “Memristor-The missing circuit element”. In: *IEEE Transactions on Circuit Theory* 18.5 (1971), pp. 507–519. DOI: `10.1109/TCT.1971.1083337`.

[44] A. Citri and R. C. Malenka. “Synaptic Plasticity: Multiple Forms, Functions, and Mechanisms”. In: *Neuropsychopharmacology* 33 (2008), pp. 18–41. DOI: `10.1038/sj.npp.1301559`.

[45] F. Corradi and G. Indiveri. “A Neuromorphic Event-Based Neural Recording System for Smart Brain-Machine-Interfaces”. In: *IEEE Transactions on Biomedical Circuits and Systems* 9.5 (2015), pp. 699–709. DOI: `10.1109/TBCAS.2015.2479256`.

[46] E. Donati and M. Payvand et al. “Discrimination of EMG Signals Using a Neuromorphic Implementation of a Spiking Neural Network”. In: *IEEE Transactions on Biomedical Circuits and Systems* 13.5 (2019), pp. 795–803. DOI: `10.1109/TBCAS.2019.2925454`.

[47] B. Farley and W. Clark. “Simulation of self-organizing systems by digital computer”. In: *Transactions of the IRE Professional Group on Information Theory* 4.4 (1954), pp. 76–84. DOI: `10.1109/TIT.1954.1057468`.

[48] M. Le Gallo and A. Sebastian. “An overview of phase-change memory device physics”. In: *Journal of Physics D: Applied Physics* 53.21 (2020). DOI: `10.1088/1361-6463/ab7794`.

[49] W. Gerstner and W. M. Kistler. “Mathematical formulations of Hebbian learning”. In: *Biological Cybernetics* 87 (2002), pp. 404–415. DOI: `10.1007/s00422-002-0353-y`.

[50] W. Gerstner and W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, 2002. DOI: `10.1017/CBO9780511815706`.

[51] E. M. Glaser and H. Van Der Loos. “A Semi-Automatic Computer-Microscope for the Analysis of Neuronal Morphology”. In: *IEEE Transactions on Biomedical Engineering* BME-12.1 (1965), pp. 22–31. DOI: `10.1109/TBME.1965.4502337`.

[52] D. Goodman and R. Brette. “Brian: a simulator for spiking neural networks in Python”. In: *Frontiers in Neuroinformatics* 2 (2008). DOI: `10.3389/neuro.11.005.2008`.

[53] L. Goux. “OxRAM technology development and performances”. In: *Advances in Non-volatile Memory and Storage Technology* (2019), pp. 3–33. DOI: `10.1016/b978-0-08-102584-0.00001-2`.

[54] S. Hayman. “The McCulloch-Pitts model”. In: *IJCNN’99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339).* Vol. 6. 1999, pp. 4438–4439. DOI: `10.1109/IJCNN.1999.830886`.

[55] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Psychology Press, 1949. DOI: `10.4324/9781410612403`.

[56] N. Hiratani and T. Fukai. “Redundancy in synaptic connections enables neurons to learn optimally”. In: *Proceedings of the National Academy of Sciences* 115.29 (2018). DOI: `10.1073/pnas.1803274115`.

[57] A. L. Hodgkin and A. F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of physiology* 117.4 (1952), pp. 500–544. DOI: `10.1113/jphysiol.1952.sp004764`.

[58] G. Indiveri and T. K. Horiuchi. “Frontiers in Neuromorphic engineering”. In: *Frontiers in Neuroscience* 5 (2011). DOI: 10.3389/fnins.2011.00118.

[59] R. A. John and Y Demirağ et al. “Reconfigurable halide perovskite nanocrystal memristors for neuromorphic computing”. In: *Nature Communications* 13.1 (2022). DOI: 10.1038/s41467-022-29727-1.

[60] F. Jug. “On Competition and Learning in Cortical Structures”. PhD thesis. Zürich: ETH Zurich, 2012. DOI: 10.3929/ethz-a-007140134.

[61] A. Krenker, J. Bešter, and A. Kos. “Introduction to the artificial neural networks”. In: *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech* (2011), pp. 1–18.

[62] B. Macukow. “Neural Networks – State of Art, Brief History, Basic Models and Architecture”. In: *Computer Information Systems and Industrial Management.* Springer International Publishing, 2016, pp. 3–14. ISBN: 978-3-319-45378-1.

[63] R. Meddis. “Simulation of mechanical to neural transduction in the auditory receptor”. In: *The Journal of the Acoustical Society of America* 79.3 (1986), pp. 702–711. DOI: 10.1121/1.393460.

[64] G.B. Moody and R.G.Mark. “The impact of the MIT-BIH Arrhythmia Database”. In: *IEEE Engineering in Medicine and Biology Magazine* 20.3 (2001), pp. 45–50. DOI: 10.1109/51.932724.

[65] P. M. Nanninga. “A computational neuron model based on Poisson–Nernst–Planck theory”. In: *Proceedings of the 14th Biennial Computational Techniques and Applications Conference, CTAC-2008.* Ed. by Geoffry N. Mercer and A. J. Roberts. Vol. 50. ANZIAM J. 2008, pp. C46–C59.

[66] E. O. Neftci, H. Mostafa, and F. Zenke. *Surrogate Gradient Learning in Spiking Neural Networks.* 2019. DOI: 10.48550/ARXIV.1901.09948.

[67] Y. V. Pershin and M. Di Ventra. “Memory effects in complex materials and nanoscale systems”. In: *Advances in Physics* 60.2 (2011), pp. 145–227. DOI: 10.1080/00018732.2010.544961.

[68] *PhysioBank ECG Annotations.* https://archive.physionet.org/physiobank/annotations.shtml. 2016.

[69] J. Pods, J. Schönke, and P. Bastian. “Electrodiffusion Models of Neurons and Extracellular Space Using the Poisson-Nernst-Planck Equations—Numerical Simulation of the Intra- and Extracellular Potential for an Axon Model”. In: *Biophysical Journal* 105.1 (2013), pp. 242–254. DOI: doi.org/10.1016/j.bpj.2013.05.041.

[70] F. Rosenblatt. “The perceptron: A probabilistic model for information”. In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[71] L. Sirovich. “Dynamics of neuronal populations: Eigenfunction theory; some solvable cases”. In: *Network (Bristol, England)* 14 (June 2003), pp. 249–72. DOI: 10.1088/0954-898X/14/2/305.

[72] Y. E. Wang, G. Wei, and D. Brooks. “Benchmarking tpu, gpu, and cpu platforms for deep learning”. In: *arXiv preprint arXiv:1907.10701* (2019). DOI: 10.48550/arXiv.1907.10701.

[73] Z. Yan, J. Zhou, and W. F. Wong. “Energy efficient ECG classification with spiking neural network”. In: *Biomedical Signal Processing and Control* 63 (2021). DOI: `10.1016/j.bspc.2020.102170`.

[74] Z. Yan, J. Zhou, and W.F. Wong. “Energy efficient ECG classification with spiking neural network”. In: *Biomedical Signal Processing and Control* 63 (2021). DOI: `10.1016/j.bspc.2020.102170`.

[75] F. Zenke. *SpyTorch*. Version v0.3. Mar. 2019. DOI: `10.5281/zenodo.3724018`.