POLITECNICO DI TORINO

Master's Degree in Data Science

Master's Degree Thesis

SpikExplorer: a tool for Design Space Exploration of Spiking Neural Network Architecture

Supervisors

Prof. Stefano DI CARLO

Prof. Alessandro SAVINO

Prof. Alessio CARPEGNA

Candidate

Dario PADOVANO

April 2024

Summary

We live in the age of artificial intelligence, where high level abstractions of human brain consume enormous amount of energy to write an essay for high school kids. In this scenario it could be asked how if there exists a way to improve consumption keeping performance high, the answer is in our brain. The human neural network consumes infinitesimal amounts of energy compared to the stateof-the-art artificial neural networks such as GPT-4, in this way the structure and dynamics of human brain could became an inspiration in the development of pioneering models such as Spiking Neural Networks. In the last few years, Spiking Neural Networks (SNN) heavily spread through the machine learning community as a novelty approach to deep learning. Its basic unit, the spike, take inspiration from the human action potential transmitted by neuron in neuron. These architectures have already found application in popular task as image classification and speech recognition thanks to the intrinsic relationship between spikes and time.

The aim of this work is not only the use of SNN as a way to exploit the spiking dynamic similar to what happens between biological neurons, instead the goal is the creation of a tool capable of generalize as much as possible the construction of the SNN architecture by specifying particular constraints related to the future deployment on FPGA-based (Field Programmable Gate Array) neuromorphic hardware.

This work is divided in two parts, the first will address the problem of finding the optimal method to search the design space, while the seconds will explore the tool structure with an extensive search carried on in order to evaluate the performance of the tool.

Table of Contents

List of Tables

List of Figures

Acronyms

ANN
Artificial Neural Network

SNN
Spiking Neural Network

STDP
Spike Timing Dependent Plasticity

FPGA
Field Programmable Gate Array

IF
Integrate and Fire

LIF
Leaky Integrate and Fire

LTD
Long Term Depression

LTP
Long Term Potentiation

Chapter 1

Introduction

One of the most debated arguments of the last decade is for sure artificial intelligence and neural networks but how much "intelligence" there is here? Actually the driving force behind the development of neural networks is the study of human brain, a subject far from being exhausted. The goal is to mimic the behaviour and connections of the brain in order to make an "artificial" version although the most common neural networks are barely an abstraction of axons and dendrites that everyday drive us through our decisions.

1.1 Human as the machine

1.1.1 Biological Neuron

Before taking into account the artificial counterpart, let us briefly discuss about the biological neuron. Neurons are the fundamental cells that make possible the correct functioning of the nervous system, they are an electrically excitable cell capable of firing electric signals called action potentials, or alternatively *spikes* [1]. Neurons are linked together in order to form the nervous tissue of all animals and, while sharing the same functionality, they form neural circuit. We can split the basic objects that compose the structure of a biological neuron 1.1 into:

1. **Soma**: it is the body of the neuron and it contains the nucleus where most of the protein synthesis occurs.
2. **Dendrites**: they are cellular extensions with a large number of branches capable of intercept inputs and send to the nucleus via the dendritic spine.
3. **Axon**: it is long and fine connection that propagates spikes away from the soma through the *axon hillock* that, thanks to the high density of voltage-dependent sodium channels, works as initiator of spikes.

4. **Axon Terminals**: they represent the end of the axon and contain the synapses where neurotransmitter chemicals make possible the communication with target neurons

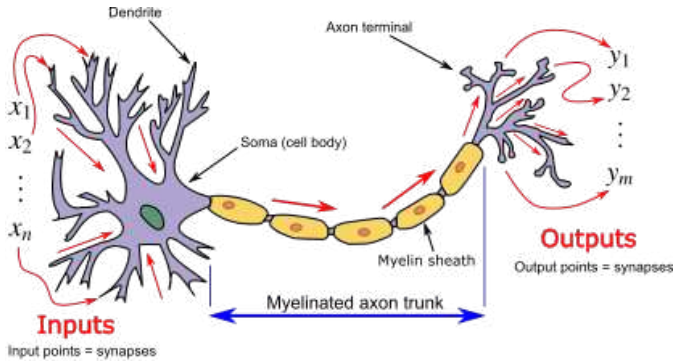


Figure 1.1: Basic structure of a biological neuron. [2]

Therefore, neurons communicate each other making a dense skein made of connections, it was estimated that around 86 billion of neurons are present in our brain, which acts like an electrical circuit more that one might think. The discussion will revolve around the key computational element in this circuit: the *spike*, or more formally, *action potential*. It occurs when the membrane potential of specific cell rapidly increases and decreases causing a depolarization that propagates along the axon: it is generally said that the neuron *fires* spikes to other neurons, propagating the information through the intricate network in our brain. [3]

1.1.2 Neuron Plasticity

A peculiar characteristic of biological neurons is the ability of the synapses to change the synaptic strength over time. One rule that models the biological process that rule the adjustments of connection strengths quite well is Spike-Timing-Dependent Plasticity [4] (STDP). It is based on the relative timing of the neuron's output and input action potentials. STDP plays an important role in the adapting and learning process of a network and it partially explains the development of the nervous system itself, especially when talking about Long-Term Potentiation (LTP) and Long-Term Depression (LTD). This topic will be discussed with major focus in chapter 4.

1.2 Need for computational efficiency

Our brain is extremely efficient: it operates at 12-20 W of power in opposition to one of the most discussed AI at the moment, GPT-4 that, only for training purpose,

consumes around 190.000 kWh [5]. Since the discovery of the neuron in the early years of the 20th century, thanks to spanish neuroscientist Santiago Ramón y Cajal [6], scientist wanted to abstract information from the elemental building block of the brain in order to comprehend the intricate connections formed by cells in our brain. A first development in this direction was the Hodgkin-Huxley model that shows us the relationship between the flow of ionic currents across the neuronal cell membrane and the membrane voltage of the cell [2, 3].

The Hodgkin-Huxley model, also known as conductance-based model is a mathematical model capable of abstract the dynamics of the action potential in neurons. Formally, it is a set of non linear differential equations that try to explain the electrical characteristics of cells.

1.2.1 Artificial Neural Network

A further step towards the abstraction of the functioning of biological neuron was done by Warren McCulloch and Walter Pitts with the theory of the perceptron [8] (also known as McCulloch-Pitts Neuron, from the name of its creator). It is a type of linear classifier, to be precise we are talking about binary classification, functions capable of deciding if an input belongs or not to a certain class. The single layer perceptron laid to a stagnation till the introduction, in 1970, of multi-layered perceptrons.

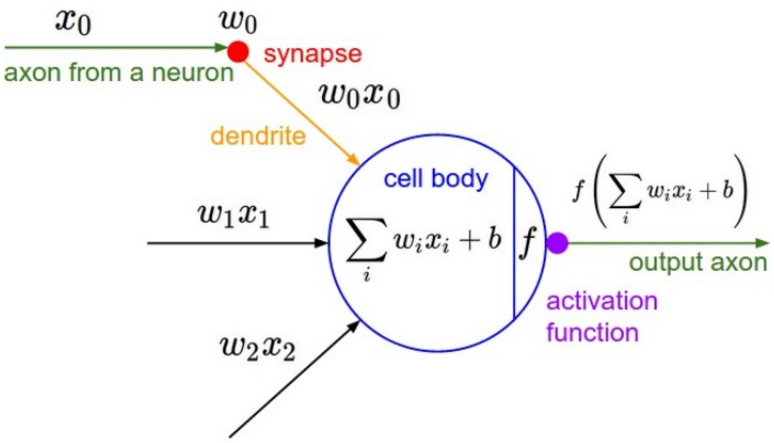


Figure 1.2: Clarifying representation of an artificial neuron, in the figure it was put in evidence the relationship between biological and artificial neuron. [7]

Figure 1.2 [8] is a clear representation of the dynamics of the information that passes through an artificial neuron, as seen for the biological counterpart, input flows into the body of the neuron and after some computation an output spike is fired. To be more precise the neuron's output is the dot product between the

inputs and its weights plus the bias, in order to introduce non-linearity, the result is submitted to an activation function.

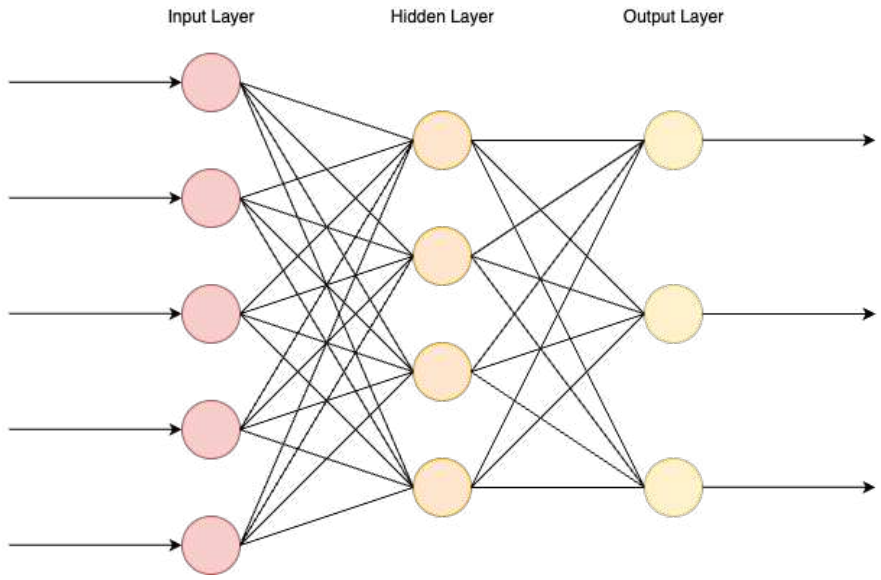


Figure 1.3: Layer structure in a multi-layer perceptron.

What really makes the difference is the aggregation of perceptrons into layer and the overlap of layers themselves. The so obtained MLP, as shown in the figure 1.3, is composed of three type of layer:

1. Input layer: the first layer encountered by the given input, each node of the graph represents an input given to the network. Important to note that the information that is coming to the input layer is formed by a sequence of number. On the contrary, the biological information is formed by electrical impulse. In a fully connected network as MLP, the input of each node is propagated to each hidden layer's neuron.

2. Hidden layer: all the computational layers that lays between the input and output are called hidden.

3. Output layer: the layer charged of providing a comprehensible result as output of the neural network.

The learning process of the artificial network is strictly correlated to the passage of back propagation in which a cost function is computed basing the calculation on the input and the expected output. The network learns through the process of minimization of this cost function, the network resolves into a problem of

minimization in which the goal is to find absolute minimum of the provided error function. In order to achieve this results the approach of gradient descent is used, which will be deeply discussed in the opportune chapter.

If we want to find a parallelism between artificial and biological counterparts, we can think about the STDP as the real backpropagation in which the connections between the neurons are strengthen or weaken in a way that resembles the ANN, in which the weights on the connections between each nodes of the graph are changed .

1.2.2 Spiking Neural Network

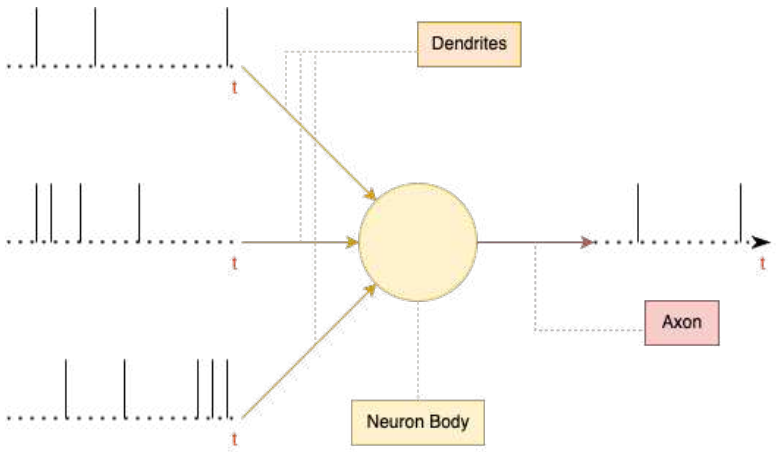


Figure 1.4: Simplified structure of a spiking neuron, it is clear the importance of time that put in relation the input and the output of the neuron.

Standard NN are particular expensive to train and maintain due to the large number of parameters [9] and, as we have seen in the previous chapter, one of the intent of this work is to reduce the power consumption of ready-to-use models by adopting some mechanisms that find inspiration in the nature of human brain's cells connection. In order to achieve this results we can exploit a particular architecture of artificial neural network called *Spiking Neural Network*. As it is from the name, the core concept behind which this architecture was theorized was spikes and their high correlation to time dimension. Let us take into consideration **Figure** 1.4 that takes into account the basic structure of a spiking neuron, similarly to ANN's neuron, the spiking counterpart operates on a weighted sum of inputs. The major difference is the amount of information that is transported trough the network thanks to time encoding of input and output. A standard ANN only brings a value to the neuron while SNN not only tells the neuron what is coming but also *when* it is coming.

As we can imagine, it becomes particularly useful when taking into account time-series or time based events, but SNN can also be used for common task as image classification.

1.2.3 Goal of the work

In a field like that of machine learning, it become meaningful not only to achieve the best performance but also to shrink the cost needed to train and inference[9]. Everyday, we are subject of AI that need a huge amount of time to train an even larger amount of parameters, this is even more emphasized while dealing with SNNs. Moreover, it is not possible to establish which is the best set of parameters by hand, since, in the majority of the cases, neural architectures act as a black box. It becomes fundamental to optimize the research time of convergence, in order to do so we will present in this thesis work "SpikeExplorer" a tool for automatic design space exploration of spiking neural network architectures, which exploits the power to understand the unknown of the Bayesian Optimization to find the best suitable model for a specific use cases. These models will then be compared to state-of-the-art FPGA accelerators for snn.

Chapter 2

Artificial Neural Network

2.1 History of AI

2.1.1 Aspiring to be gods

Since the dawn of time, humans have tried to emulate the purest power of a god, the power of creation. The eternal will to generate artificial creatures capable of thinking independently can be found in ancient myths such as the greek myth of Talos, a giant constructed of bronze who acted as guardian for the island of Crete, capable of recognise enemy ships and shoot them, or the myths of golems described in the writings of Eleazar Ben Judah of Worms, anthropomorphic creatures belonging to the jewish folklore that can be animated by insertion of a piece of paper with any of God's names on it.

Much more of interest is the construction of automata built by craftsman from every civilization, with the belief that they may encapsulate the thoughts and habits of the creatures they inspire. Honorable mention must be done for the oldest ones, the sacred statues from Egypt and Greece.

Artificial intelligence is based on the assumption that the process of human thought can be mechanized. The study of mechanical—or "formal"—reasoning has a long history. Chinese, Indian, and Greek philosophers all developed structured methods of formal deduction in the first millennium B.C.. Much later, in the 17th century, Leibniz, Thomas Hobbes and René Descartes reduced every debate to a bunch of calculus, laying the foundation to a physical symbol system hypothesis that would become the guiding faith of AI research [10].

In the 20th century, it started to seem reasonable to develop an artificial intelligence. From Bertrand Russell and Alfred Whitehead's Principia Matematica, published in 1913, followed a simple question: can all of mathematical reasoning be formalized? Multiple scientists tried to reply to this question, one of them was the Alan Turing with its Turing's Machine: the Church-Turing thesis implied that

Figure 2.1: Vase representing the death of Talos, Jatta National Archaeological Museum, Ruvo di Puglia.

a mechanical device, shuffling symbols as simple as 0 and 1, could imitate any conceivable process of mathematical deduction. The key insight was the Turing machine, a simple theoretical construct that captured the essence of abstract symbol manipulation.

In the 1940s and 50s, a handful of scientists from a variety of fields (mathematics, psychology, engineering, economics and political science) began to discuss the possibility of creating an artificial brain. The field of artificial intelligence research was founded as an academic discipline in 1956.

2.1.2 First Artificial Neuron and the Dartmouth project

The very first milestone in AI field was for sure the computational model proposed by Warren McCulloch and Walter Pitts in 1943, when the first artificial neuron was created and the journey began. Their intuition relied on the idealization of neural networks, reducing neurons to simple logical functions. In their work "A logical calculus of the ideas immanent in nervous activity", they were the first scientists to theorize the artificial neural network, although not calling them in a way familiar

to us as ANN.

After the presentation of their theory, the scientific environment of the fifties was struck by a particular fervor and a peculiar question started to rise: is a machine capable of thinking? Turing tried to give an answer to this question with its Turing Test: in 1950 he published a landmark paper in which he told that in the moment in which a machine will carry on a conversation that was indistinguishable from a conversation with a human being, then it was reasonable to say that machine was thinking. Although it is difficult to believe, at least during the 50s, the problem laid the foundation to the plausibility of developing a artificial intelligence.

Between McCulloch and Pitts students there was also Marvin Lee Minsky, later dubbed as "the father of artificial intelligence". He was an emblematic figure in the community of cognitive sciences. His major contribution in the field of AI was the construction of a random electrical neural network called SNARC, that stands for Stochastic Neural Analog Reinforcement Calculator.

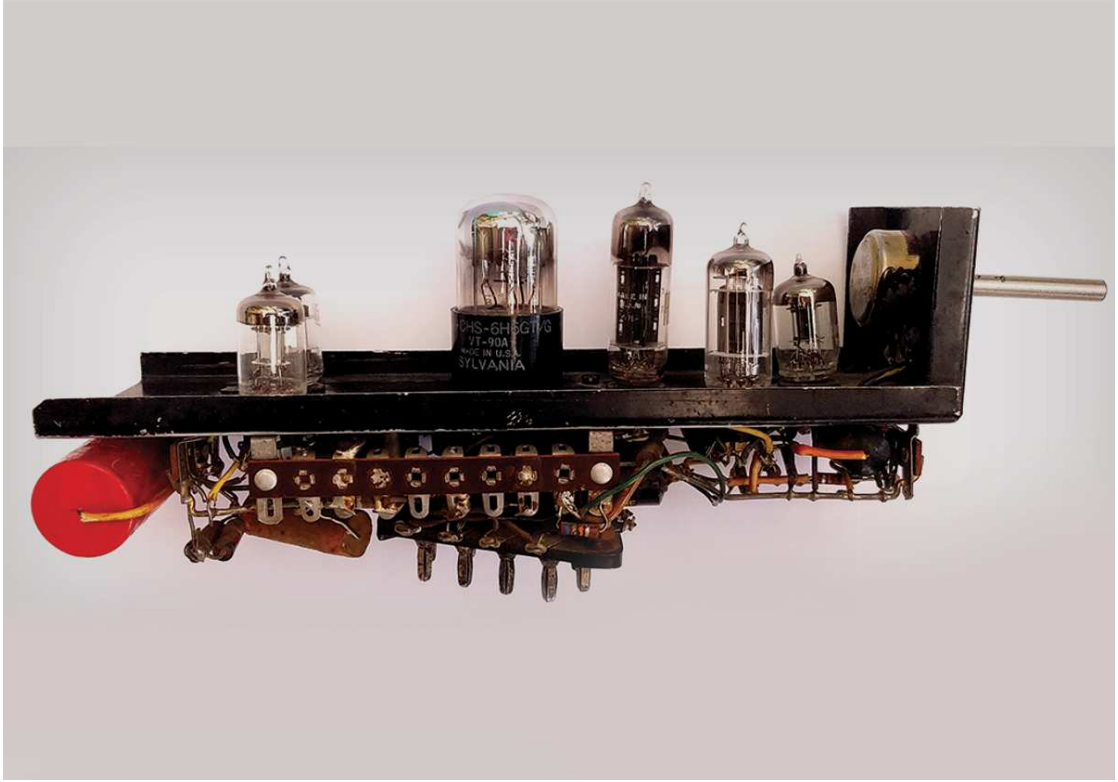


Figure 2.2: The original SNARC machine created by Marvin Minsky. [11]

Its basic functioning consists of a random connected network formed by approximately 40 Hebbs synapses, having a memory capable of holding the probability related to a signal coming as input and another signal exiting as output. This

probability is regulated through a knob that could assume values between zero and one. If the probability signal gets through, a capacitor remembers this function and engages a "clutch". At this point, the operator will press a button to give reward to the machine. After that, a large motor starts, a chain that goes to all 40 synapse machines is activated and checks if the clutch is engaged or not. As the capacitor can only "remember" for a certain amount of time, the chain only catches the most recent updates of the probabilities [11].

Figure 2.3: Some of the participants of Dartmouth Convention. [12]

In 1956 the Dartmouth Summer Project on Artificial Intelligence was the foundational event for the field of AI. Due to the enormous number of different papers that during these years were spreading in the scientific community, chaos reigned sovereign and a point of contact between the major exponents of the subject was needed. For this reason on September 2, 1955, Dr. John McCarthy, Marvin Minsky, Nathaniel Rochester and Claude Shannon proposed the Dartmouth Project to which is credited the introducing of the term artificial intelligence for the first time. The proposal states that for a 2-months period, during the summer of 1956, a 10-man study about AI would be carried out at Dartmouth College in Hanover, New Hampshire. The final number of actual attendees and visitors was much

higher, listing in them names of the caliber of Marvin Minsky, John Nash and Allen Newell. The AI's spring was started.

2.1.3 *Perceptrons* and AI winter

The following year, in 1957, the American psychologist Frank Rosenblatt started to formulate his theory about perceptrons that culminated, in 1960, with the creation of the Mark I Perceptron, basically an hardware solution capable of learn through the process of trial and error. Later on, in 1962, he published Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms [13] in which the importance of multi-layered perceptrons networks is highlighted for the first time . Even if the New York Times billed it as revolutionary, it would soon began the incipit of a long halt in AI field.

Marvin Minsky in the early 70s took into consideration the challenges behind perceptron's development, not only addressing their limitation in terms of capability and growth possibility, in his works "Perceptrons" he had heavily criticized the work done by Rosenblatt, emphasizing his skepticism about connectionism. This was not the only reason for which during the seventies there was a first arrest in the research field of AI: limited computer power, intractability and combinatorial explosion, commonsense knowledge and reasoning, Moravec's Paradox: "proving theorems and solving geometry problems is comparatively easy for computers, but a supposedly simple task like recognizing a face or crossing a room without bumping into anything is extremely difficult"[10]. This, together with the lack of frame and qualification cause the start, in the early seventies, of the AI Winter.

2.1.4 Neural Networks Revival

In such a non-proactive environment, new developments in AI seemed impossible. The previously mentioned limitations caused the progressive end of investments in the research by states, dissuading many scientists from further exploring the field, but in 1980s something would change AI forever. Multiple reasons caused the reigniting of AI[14]:

- The revival of connectionism done by John Hopfield and David Rumelhart popularized deep learning techniques
- Edward Feigenbaum's expert systems was capable of mimic the decision making process of a human expert
- Japan hardly invested with the goal of revolutionizing computer processing with its Fifth Generation Computer Project (FGCP)

Even if the lack of funds was still present and the ambitions of the time did not find the desired results, a new generation of scientists was born and was hungry for knowledge. During 1990s and 2000s the world saw the unstoppable pace of the AI revolution which led to the reality that we all know today.

2.2 Basics of an Artificial Neural Network

It has already been shown previously the structure of an ANN, but let us dive more in detail about its basic components. In this part it will be addressed the following question: how an ANN learns? the answer to this question will be useful later on while talking about SNN and their peculiar way to learn.

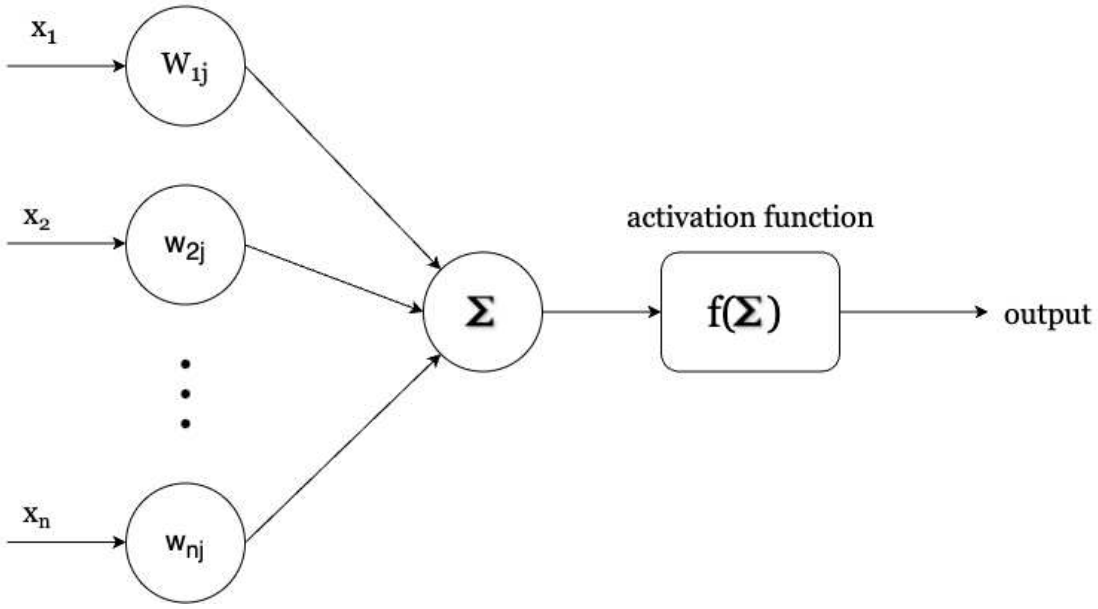


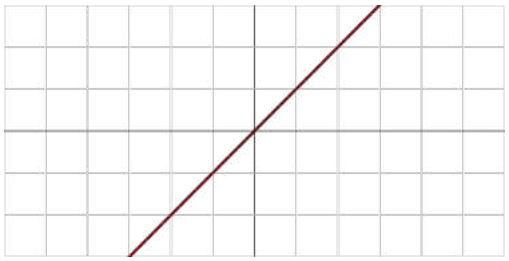
Figure 2.4: A more detailed figure of an artificial neuron.

The information is sliced into n parts, the so obtained input x_i flow to the artificial neuron through synapses, each equipped with a weight w_{ij}, in this way x_i is multiplied by its corresponding weight and summed to the other parts of the information obtaining (1).

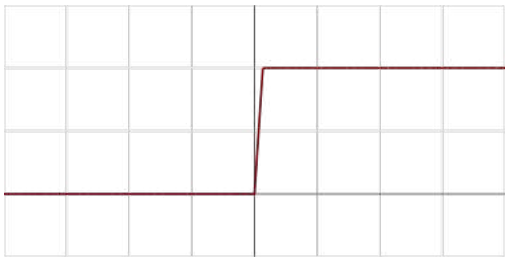
$$net_j = \sum_{i=1}^{n} x_i * w_{ij}$$

The net_j is passed as input to an activation function, that plays a important role in the process of learning of the neural network. Activation functions introduce

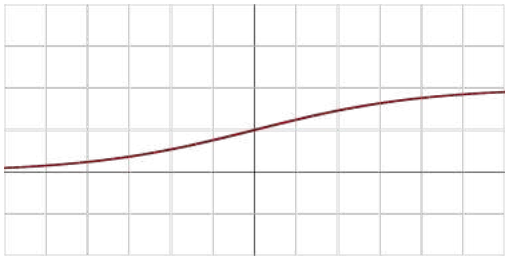
non-linearity into the network, making it possible to learn non-linear functions that would be impossible to learn otherwise.



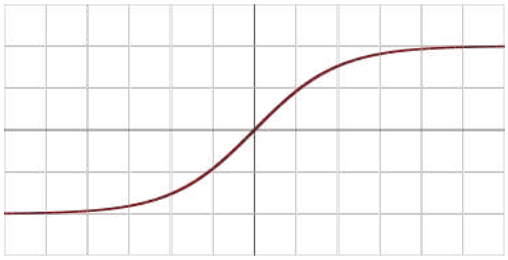
(a) Identity



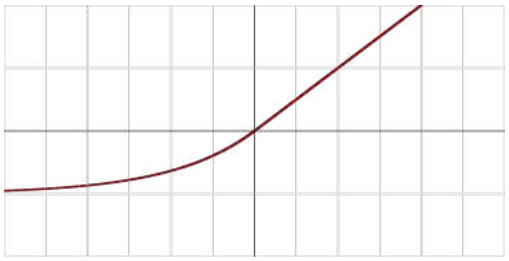
(b) Binary Step



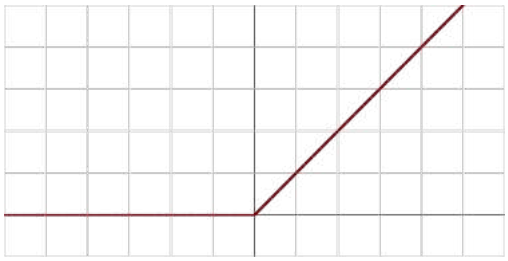
(c) Logistic



(d) Hyperbolic



(e) Exponential LU



(f) Rectified LU

Figure 2.5: Some of the most used activation functions in ML. [15]

This process is repeated through the network for each neuron until the last layer is reached, returning the evaluated output. This is the forward phase, but how the network actually learns?

2.2.1 Learning Paradigms

There are a lot of different learning paradigms that have been used in machine learning, let us analyze the main three paradigms.

Supervised Learning

As it is suggested by the name, in supervised learning the network is fed with inputs that are paired to the desired outputs. In this way the cost function can be easily computed by eliminating incorrect deductions [16]. Formally, a Feed-Forward Neural Network is trained by passing a pair (X, Y) where $X = (x_1, x_2, ..., x_N)$ and $Y = (y_1, y_2, ..., y_N)$, each input is a p-dimensional vector $x_i = \left\langle x_{i1}, x_{i2}, ..., x_{ip} \right\rangle$ associated to the corresponding q-dimensional output $y_i = \left\langle y_{i1}, y_{i2}, ..., y_{iq} \right\rangle$. The real target associated to a given input is then compared to the output of the net ($\widehat{Y} = (\widehat{y_1}, \widehat{y_2}, ..., \widehat{y_N})$ and $\widehat{y_i} = \left\langle \widehat{y_{i1}}, \widehat{y_{i2}}, ..., \widehat{y_{iq}} \right\rangle$) by computing some cost function, the ultimate objective of the learning is the reduction of this given error function.

Unsupervised Learning

Used in estimation problems such as clustering, the unsupervised learning is characterised by input data not anymore paired with the correct output (since it is not labeled) but instead there is a pre-defined cost function created a priori on the basis of some assumptions. So the difference are not only in the architecture and in the fact that here we do not have a correct label, but also and above all in the tasks that each learning paradigms is supposed to address.

Reinforcement Learning

Reinforcement learning is a paradigm modeled as a Markov decision process. In order to be functioning it is needed:

- An environment in which actions will be taken
- A set of action states S_i
- A set of actions for the agent A
- The transition probability between two states of the process

$$P_a(s, s') = Pr(S_{t+1} = s' | S_t = s, A_t = a)$$

- A function that tells the reward obtained by passing from s to s'

$$R_a(s, s')$$

2.2.2 Error functions for supervised learning

When we talk about supervised learning it is equivalent to consider the minimization problem of the distance between the desired output and the obtained one. In order to compute a measure of the difference, an error function can be used. Let's dive into two different pretty common functions.

Error or Loss Function?

Actually, the real difference depends on the context in which the two terminologies are addressed. With the term error function we usually intend how much the predicted value deviates from the real one. So it only depends on the behaviour of the quantities evaluated and the mechanism used to evaluate them. Much more complex is the understanding of the loss function. It can be said that the loss tends to interpret how bad a prediction is with respect to the real one. The terminologies even if seems to be synonymous tends to adopt different meaning once you get to the practical side.

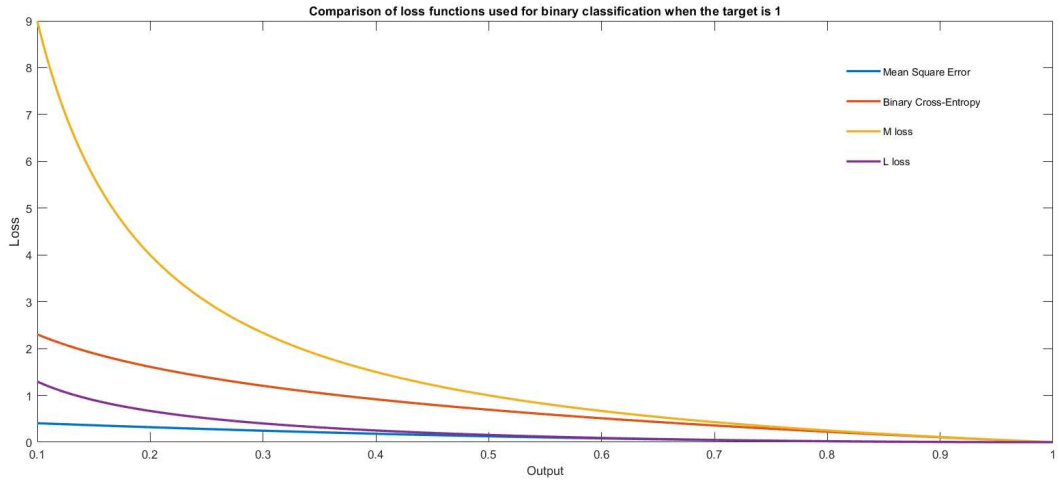


Figure 2.6: In the figure it can be seen how for different loss function reacts in the training of a binary classifier.[17]

Mean Squared Error

The mean squared error is probably the most known and used cost function, and can be expressed as follow:

$$c_f(y_i, \widehat{y}_i) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{q} (y_{ij} - \widehat{y}_{ij})^2$$

- y_{ij}: the desired response,
- $\widehat{y}_{ij}$: the response returned by the artificial neural network.

The difference between y_{ij} and $\widehat{y}_{ij}$ is summed over the N data pairs.

Cross-Entropy Loss

In information theory, it is called cross-entropy the measure of the average number of bits needed to identify an event drawn from a given set if the solution is optimized with respect to an estimated probability q rather than the true distribution p [18]. The definition can be formalized as follows, given a distribution q relative to a distribution p, we have:

$$H(p, q) = -E_p[logq] \tag{2.1}$$

where E_p is the expectation of the distribution p. The equation can be re-written both for discrete probability distributions as:

$$H(p, q) = -\sum_{x \in X} p(x) \log q(x) \tag{2.2}$$

and for continuous as:

$$H(p, q) = -\int_X P(x) \log Q(x) dx \tag{2.3}$$

With respect to what we previously said, cross-entropy could be translated to an optimal loss function for machine learning and optimization problems[19]. From this point of view we have that the true probability distribution p_i become the true label of our output while the given distribution q_i corresponds to the predicted value returned by our model. Consider a binary classifier, here the cross-entropy works as a measure of dissimilarity between the two probability distribution.

$$H(p, q) = -\sum_i p_i \log q_i = -y \log \widehat{y} - (1 - y) \log(1 - \widehat{y}) \tag{2.4}$$

where:

$$q_{y=0} = 1 - \widehat{y} \tag{2.5}$$

$$q_{y=1} = \widehat{y} \equiv g(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}} \tag{2.6}$$

In this case the weights w can be optimized through algorithm such as gradient descent.

2.2.3 Backpropagation

The term "back-propagation error connection" was firstly introduced by Frank Rosenblatt in [13] but he did not know how to implement it. As we can learn from [21], backpropagation is an efficient implementation of the Leibniz's chain

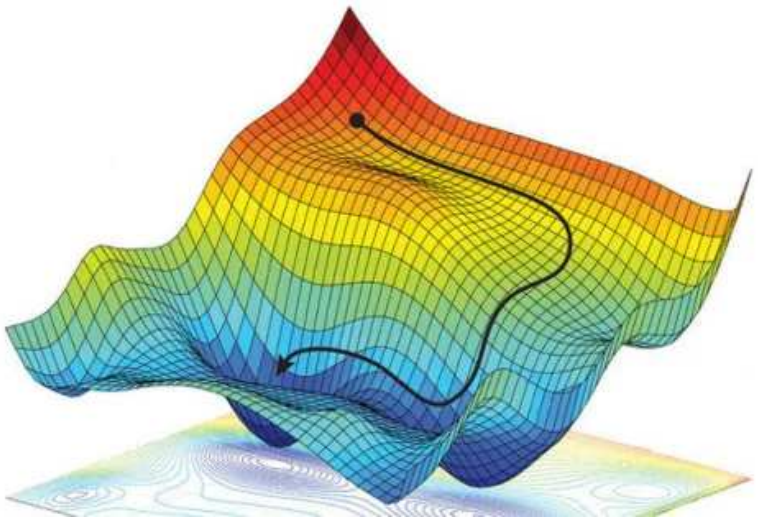


Figure 2.7: "Non-convex optimization utilizing stochastic gradient descent to find a local optimum in our loss landscape." [20]

rule introduced in the 1676, even if various systems spread during 1960s, only in 1970, with the work of Seppo Linnainmaa backpropagation was implemented as we know today [22]. In his master's thesis in fact he described the "reverse mode of automatic differentiation", the concepts discussed where purely mathematical and he did not mentioned neural networks, but the importance of this scientific treatise remains unchanged.

The concepts developed by Linnainmaa would be unthinkable due to its complexity, but it is possible to abstract the reasoning behind it. Let us consider backpropagation as an optimization problem, for general purpose let us consider as the loss function the previously mentioned cross-entropy loss:

$$H(p,q) = -\sum_i p_i \log q_i = -y \log \widehat{y} - (1-y)\log(1-\widehat{y}) = H(x,y) \tag{2.7}$$

The goal is to minimize the loss, which concretely turns into finding the minimum value assumed by the function $H(x,y)$. In this way the objective is being remapped as an optimization problem that could be solved through different algorithms, one of the most common is gradient descent.

Gradient Descent

One of the best iterative algorithms for unconstrained mathematical optimization is gradient descent. The idea is to compute the gradient (or an approximate version of it) than use the opposite of its direction to move towards a local minima. In

order to visualize the concept, lets think about a water drop on a three dimensional space, as we can imagine the drop, pushed by force of gravity, will descents towards a minimum stationary point in the space (even local or global minimum). The same happens here with GD. Gradient tells us which direction to choose in order to reach the minimum value in a determined function, in our case the cost/loss function.

In deep learning algorithms is more frequent to use Stochastic Gradient Descent (SGD) capable of optimizing an objective function with suitable smoothness at a much lower effort since it uses a stochastic version of the gradient calculated on a randomly selected subset of the data set. This practice reduces by a lot the amount of computational power needed to evaluate the gradient.

Chapter 3

Biological Neuron

The key processing element in our brain is the neuron. It was estimated that in our brain there are approximately 100 billions neurons, with a density of 100000 unit on mm^3 [23]. They have different forms and dimensions that vary from $4\mu m$ to 1 or 2 meters (this refers to the Dorsal Root Ganglion that carries sensory information from the body to the brain, it is uni-polar which means that axons connect receptors on the skin all the way up to the brain stem [24]). In vertebrates, more than ten thousands different morphological classes have been observed. When talking about functional classes the number is even higher. The basic structure of the neuron, as a cell formed by dendrites, axon and soma, was already discussed in 1, for this reason it is the interest of this chapter to discuss more about how the information is transmitted, its chemical reasoning and similarities with circuits.

3.1 Biological Neuron Functioning

3.1.1 Neuronal Signals

Between the external part and the internals of the neuronal cell there is a potential difference. Generally the neuron is characterized by its membrane potential V_m, which is the internal potential measured taking the external of the cell as a reference. In rest situations $V_m \sim -60\ mV/-75\ mV$, creating what is called dynamic balance.

The neuronal signal is the index of the temporal and spatial variation of V_m. During this dynamic changes impulses are generated. These are called action potentials (PA), and present a stereotyped form. In general, an action potential is generated when a stimulus increase the membrane potential above a threshold value, $\Theta \sim -55\ mV$. The duration of the impulse is time scoped to $1-2\ ms$, having and amplitude of $100-120\ mV$. After the peek, the action potential returns to the rest value but not before the refractory phase of approximately 10 ms in

which there is hyper-polarization of the membrane where the potential reaches its minimum of about $-90\ mV$.

3.1.2 Synapses

Neurons can be joint through synapses, by which inter-neurons information is transmitted. It can be done a distinction between chemical synapse and electrical synapse.

- In a chemical synapse, the action potential generated by the pre-synaptic neuron at the end of the axon generates a depolarization on the membrane, causing the release of vesicles into the synaptic cleft. The latter contain neurotransmitters capable of opening the canals of the other neuron, through which the ionic current flows, causing a variation of V_m. In this sense, in a chemical synapse there is the transformation of a electrical signal into a chemical one which is re-transformed back into an electrical signal once reached the next neuron.
- The electrical counterpart, instead, generates an electric joint between two neurons through highly specialized ion channels, i.e. gap-junctions, that connect the pre-synaptic and post-synaptic membrane. The electrical allows a direct current flow through neurons.

3.1.3 Neuronal Membrane

The neuronal membrane is formed by a double layer of phospholipids with hydrofilic heads facing the intracellular cytoplasm and the extracellular space. The heads form an insulating layer that divide the conductive solutions, in this way it can be said that the membrane divides electric charges acting as a capacity with:

$$C_m = 1\ \mu F/cm^2 \tag{3.1}$$

The resting potential difference of the cell lies in the separation of the charges, since the potential is $V_m \sim -65\ mV$, we have:

$$Q_m = C \times V_m = 6.510^{-8} C/cm^2 \tag{3.2}$$

3.1.4 Ion Channels

As seen before, the lipid bi-layer of the neuronal membrane tends to repel electrically charged, hydrated ions, making virtually impossible the movement across the membrane that is necessary for the generation of nerve impulses. The transmembrane

movement of ions is actually carried out by molecular mechanism—specifically, by protein molecules embedded in the lipid layers. One mechanism, the sodium-potassium pump, maintains the resting potential, while the various ion channels help creating the action potential.

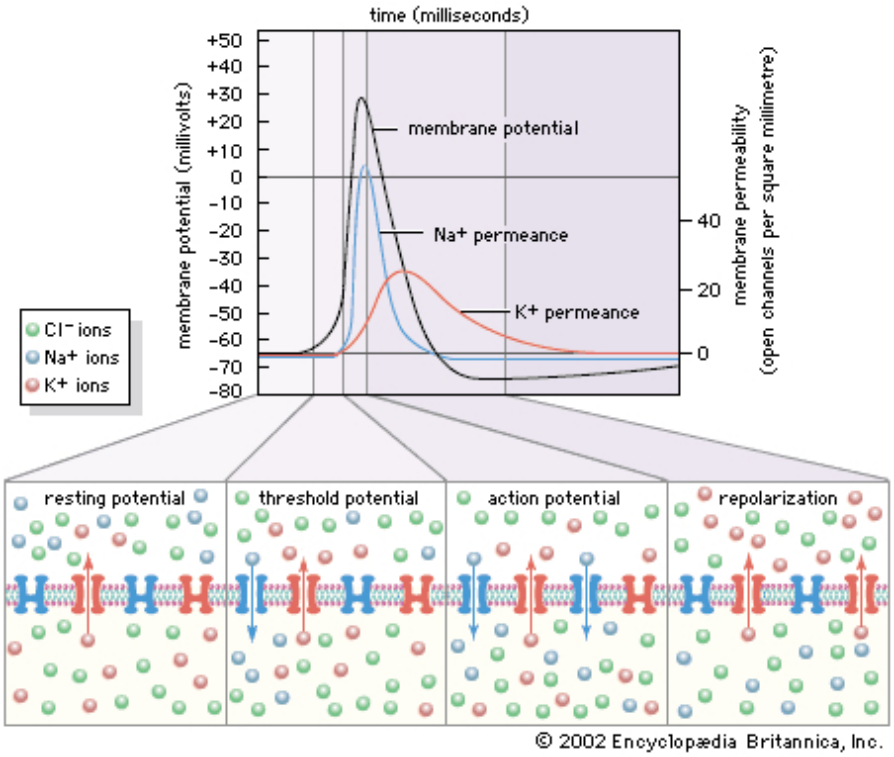


Figure 3.1: "Changes in ion permeance underlying the action potential. Electrical potential is graded at left in millivolts, ion permeance at right in open channels per square millimetre. At the resting potential, the membrane potential is close to E_K, the equilibrium potential of K+. When sodium channels open, the membrane depolarizes. When depolarization reaches the threshold potential, it triggers an action potential. Generation of the action potential brings the membrane potential close to E_{Na}, the equilibrium potential of Na+. When sodium channels close (lowering Na+ permeance) and potassium channels open (raising K+ permeance), the membrane repolarizes."[25]

Active transport

Despite the natural tendency for potassium to flow out and sodium to flow in due to their concentration gradients, the pump actively transports them against these gradients. This process, powered by ATP, creates a polarized membrane with

a negative inner surface, crucial for neuronal function. The pump's electrogenic nature and reliance on ATP illustrate its significance in regulating membrane potential and neuronal activity.

Passive transport

The sodium-potassium pump sets the membrane potential of the neuron by keeping the concentrations of Na+ and K+ at constant disequilibrium. They are thought to be cylindrical, with a hollow, water-filled pore wider than the ion passing through it except at one region called the selectivity filter. Given the relative impermeability of the plasma membrane to Na+, this influx itself implies a sudden change in permeability.

3.1.5 Action Potential

Action potential is the brief (about one-thousandth of a second) reversal of electric polarization of the membrane of a nerve cell (neuron) or muscle cell. In the neuron an action potential produces the nerve impulse, and in the muscle cell it produces the contraction required for all movement. Before stimulation, a neuron or muscle cell has a slightly negative electric polarization; that is, its interior has a negative charge compared with the extracellular fluid. Sometimes called a propagated potential because a wave of excitation is actively transmitted along the nerve or muscle fibre, an action potential is conducted at speeds that range from 1 to 100 meters per second, depending on the properties of the fibre and its environment [26].

Subsequently, protein transport molecules pump sodium ions out of the cell and potassium ions in. This restores the original ion concentrations and readies the cell for a new action potential. In the generation of the action potential, stimulation of the cell by neurotransmitters or by sensory receptor cells partially opens channel-shaped protein molecules in the membrane. If this local potential reaches a critical state called the threshold potential (measuring about - 60 mV), then sodium channels open completely.

3.1.6 Spike-Timing-Dependant Plasticity

Associative synaptic plasticity is a mechanism for learning and memory that involves changes in the strength of synapses between neurons. Long-term potentiation (LTP) and depression (LTD) are responsible for associative synapse strengthening and weakening. The rules governing LTP and LTD induction involve temporally correlated presynaptic spiking and postsynaptic depolarization, with strong depolarization leading to LTP and weaker, sustained depolarization leading to LTD.

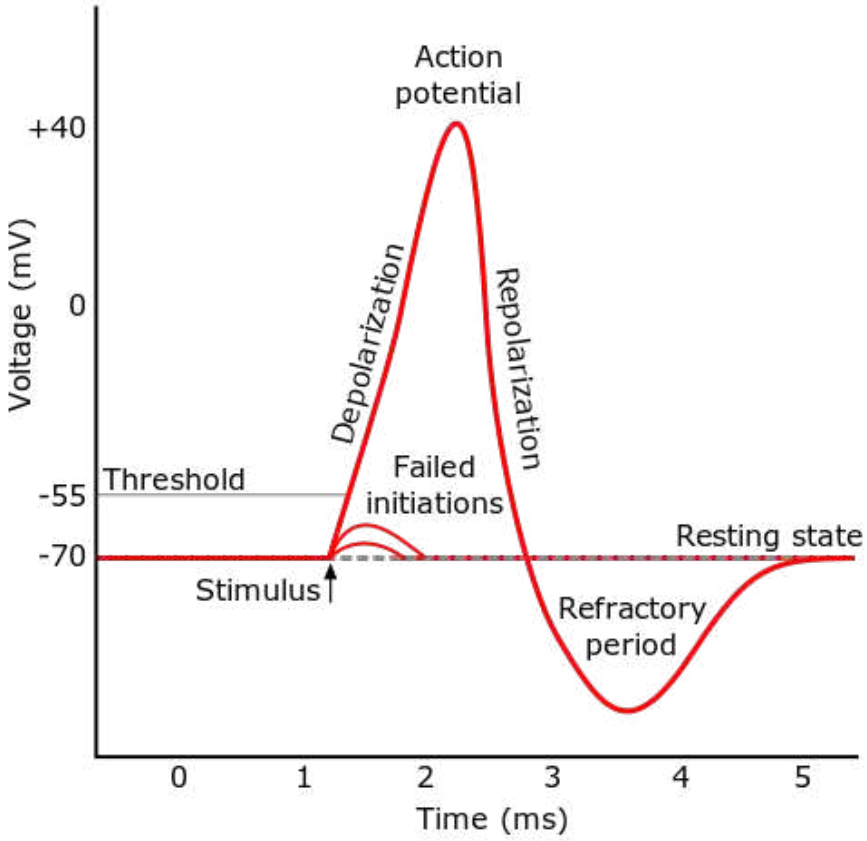


Figure 3.2: "Approximate plot of a typical action potential shows its various phases as the action potential passes a point on a cell membrane. The membrane potential starts out at approximately - 70 mV at time zero. A stimulus is applied at time = 1 ms, which raises the membrane potential above - 55 mV (the threshold potential). After the stimulus is applied, the membrane potential rapidly rises to a peak potential of + 40 mV at time = 2 ms. Just as quickly, the potential then drops and overshoots to - 90 mV at time = 3 ms, and finally the resting potential of -70 mV is reestablished at time = 5 ms." [26]

Spike timing-dependent plasticity (STDP) is a precise timing- and order-dependent plasticity that explains the development of phase locking in sound localization. STDP depends on the order and timing of pre- and postsynaptic spikes on the 10-ms time scale[27].

Canonical STDP

Canonical STDP is a type of synaptic plasticity that is bidirectional and order-dependent, with pre-leading-post spiking driving LTP and post-leading-pre spiking driving LTD. It has precise temporal windows for LTP and LTD, and several basic forms of STDP exist at different synapses. However, there is substantial variation within each form, reflecting both synapse specialization and variation in experimental conditions. STDP has expanded to include other plasticity that depends on spike timing but is not bidirectional or order-dependent.

Hebbian STDP

Hebbian STDP is a type of synaptic plasticity that strengthens synapses whose activity is causal for postsynaptic spiking and weakens non-causal synapses. It occurs when presynaptic spikes precede postsynaptic spikes by 0 to 20 ms, while LTD is induced when post leads pre by 0 to 20–100 ms. It is prevalent at excitatory synapses onto neocortical and hippocampal pyramidal neurons, excitatory neurons in auditory brainstem, parvalbumin-expressing fast-spiking striatal interneurons, and striatal medium spiny neurons in the presence of dopamine. Some synapses exhibit long LTD windows producing a net bias toward LTD. It can also occur at inhibitory synapses.

Chapter 4

Biological Neuron Model

4.1 Biological Neuron Models

4.1.1 Lapicque's discovery

While talking about artificial neurons, they can be classified into three different generations [28]. The first generation, the *old one*, laid its foundation on the McCulloch-Pitts neurons commonly known as perceptrons (or threshold gates); popular networks based on this model are the MLP, Hopfield networks or Boltzmann machines. The second generation begin with the introduction of computational units that use an activation function on a continuous set of possible output value to a weighted sum of the inputs. Typical nets as feed-forward and recurrent neural networks belong to this category, moreover an important feature of second generation networks is the support to gradient descent based algorithms such as back propagation. But even if the second type of nets is more biologically plausible than the first one, it is far from being realistically accurate. This caused a shift in the research field in which there is the propulsion to see furthermore into new horizons, experimental evidences in the neurobiological field brought to light how much the timing of single action potentials is important in biological neural systems to encode information. All these evidences led to the exploration of a third generation of neural network models that will use spiking neurons as primary computational units [28]. Mathematical models for these integrate-and-fire neurons could be traced back to 1907 with the researches carried on by Louis Lapicque. The success from Lapicque was highly remarkable to the fact that in neural modeling, studies of function do not necessarily imply the understanding of the biological basis of the phenomena, simplifing and abstracting the real model behavior [29]. Lapicque was capable of modeling the human neuron behavior through the use of an electrical circuit consisting of a resistor and a capacitor in parallel representing each the leakage and the capacitance of the cell membrane. This circuit cannot

generate for sure spikes but the intuition of Lapicque was that when the membrane capacitor was charged to certain threshold potential, a spike would be generated discharging the capacitor, resetting the membrane potential. [29].

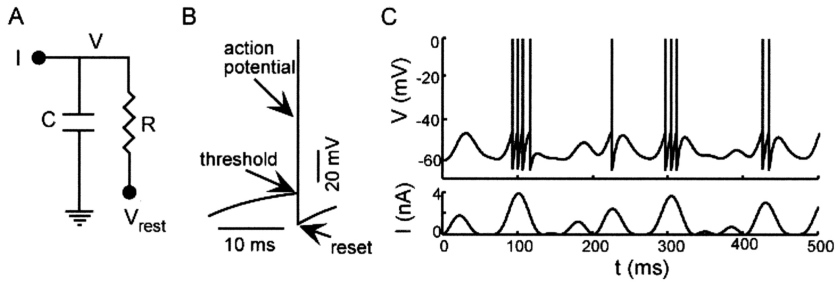


FIG. 1. The integrate-and-fire model of Lapicque. (A) The equivalent circuit with membrane capacitance C and membrane resistance R. V is the membrane potential, V_{rest} is the resting membrane potential, and I is an injected current. (B) The voltage trajectory of the model. When V reaches a threshold value, an action potential is generated and V is reset to a subthreshold value. (C) An integrate-and-fire model neuron driven by a time varying current. The upper trace is the membrane potential and the bottom trace is the input current.

[29]

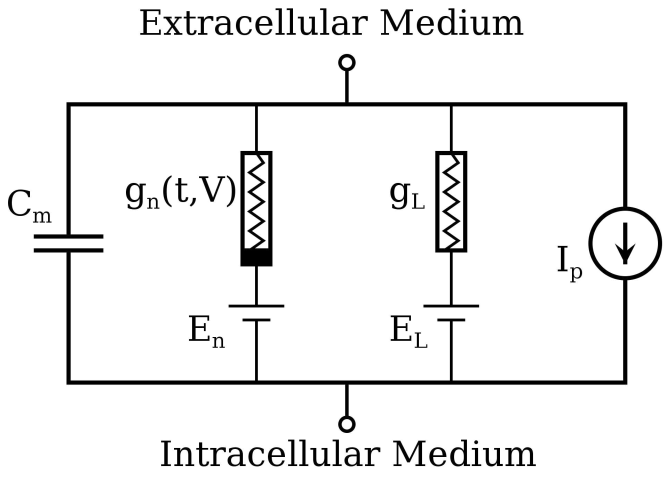
4.1.2 Hodgkin-Huxley Model

The major contribution after Lapicque's work, is for sure the Hodgkin-Huxley interpretation of the neuron. Through it is possible to construct models that include the dynamics of the voltage-dependent membrane conductances capable of generating spikes [29]. The conductance-based model described in the works of Alan Hodgkin and Andrew Huxley is a mathematical model that meticulously tell how actions potentials in neurons are propagated and initiated. The studies were possible by analyzing the axon of a giant squid [3], for these studies they received a Nobel Prize in Physiology or medicine in 1963.

As we can see in the image 4.1, every component of an excitable cell as an electrical element corresponding to it: the capacitance C_m is equivalent to the lipid bilayer as well as the electrical conductances g_n are representing the Voltage-gated ion channels while the linear conductances g_L are the leak channels. Moreover, voltage sources E_n and current sources I_p are respectively the electrochemical gradients and the ion pumps, last but not least is the membrane potentials represented by V_m [30].

From a mathematical perspective, by resolving the equations related to the electrical circuit the result is that the current flowing through the lipid b-ilayer and the current that pass through a give ion channel are respectively[3]:

$$I_c = C_m \frac{dV_m}{dt} \tag{4.1}$$



[30]

Figure 4.1: "Basic components of Hodgkin–Huxley-type models which represent the biophysical characteristic of cell membranes. The lipid bi-layer is represented as a capacitance (Cm). Voltage-gated and leak ion channels are represented by nonlinear (gn) and linear (gL) conductances, respectively. The electrochemical gradients driving the flow of ions are represented by batteries (E), and ion pumps and exchangers are represented by current sources (Ip)."[31, 30]

$$I_i = g_i(V_m - V_i) \tag{4.2}$$

where V_i is the reversal potential of a specific ion channel. In this way, it is possible to compute the total current that pass through the membrane as it follow:

$$I = C_m \frac{dV_m}{dt} + g_K(V_m - V_K) + g_{Na}(V_m - V_{Na}) + g_l(V_m - V_l) \tag{4.3}$$

As it can be seen from above, the total membrane current per unit area I can be decomposed as the sum of the current passing through the lipid layer plus the current flowing through each channels. Note that with g_l and V_l we are representing respectively the leak conductance per unit area and leak reversal potential. The leak channels account for the natural permeability of the membrane to ions and are responsible for the leakage of the potential. Using a series of experimental measurements for the ion currents of the cell's membrane called voltage clamps,(4.3) can be rewritten as follows:

$$I = C_m \frac{dV_m}{dt} + G_K n^4(V_m - V_K) + G_{Na} m^3 h(V_m - V_{Na}) + G_l(V_m - V_l) \tag{4.4}$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1 - m) - \beta_m(V_m)m \tag{4.5}$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n \tag{4.6}$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h \tag{4.7}$$

where m, n, and h are dimensionless probabilities between 0 and 1 that are associated with potassium channel sub-unit activation, while α_i and β_i are rate constants for the *i-th* ion channel that do not depend on time but depend on the voltage. In the voltage clamp situation, when the membrane potential is kept at the same value, it could be possible to study each non-linear gating equations, rewriting them as:

$$m(t) = m_0 - [(m_0 - m_\infty)(1 - e^{-t/\tau_m})] \tag{4.8}$$

$$n(t) = n_0 - [(n_0 - n_\infty)(1 - e^{-t/\tau_n})] \tag{4.9}$$

$$h(t) = h_0 - [(h_0 - h_\infty)(1 - e^{-t/\tau_h})] \tag{4.10}$$

The Hodgkin-Huxley model can be seen as a differential equations system with four state variables that change with respect to the time: $V_m(t)$, $m(t)$, $n(t)$ and $h(t)$. This system is really difficult to study because it is a non-linear problem without a closed-form solution that cannot be solved in an analytical fashion. For this reason a lot of researchers tried to simplify the HH model [32, 30].

4.1.3 Didactic *Toy* Models

Starting from the '60s, some models became popular as a valid alternative to the Hodgkin-Huxley model in term of complexity, even if are not considered as a valid option of neuron model while talking about large-scale simulation. The first one, born between 1961 and 1962, is the FitzHugh-Nagumo model, it captures the essence of neuronal excitability through a simplified set of equations involving fast and slow variables[2, 33, 34].

$$\frac{dV}{dt} = V - \frac{V^3}{3} - w + I_{ext} \tag{4.11}$$

$$\tau\frac{dw}{dt} = V - a - bw \tag{4.12}$$

It is an example of a relaxation oscillator because, if the external stimulus I_{ext} exceeds a certain threshold value, the system will exhibit a characteristic excursion

in phase space, before the variables v and w relax back to their rest values. The model presented by Richard FitzHugh [35] and modeled as a circuit by Jinichi Nagumo [36] acts as a two-dimensional version of the Hodgking-Huxley model.

One another important didactic model is the Morris-Lecar model, it was introduced in the 1980s and offers a more detailed representation of neuronal dynamics, in particular to the ones related to the membrane conductance. It was developed by Cathrine Morris and Harold Lecar to reproduce the variety of oscillatory behavior in relation to Ca++ and K+ conductance in the muscle fiber of the giant barnacle [34, 37]. The Morris-Lecar model is simpler than Hogkin-Huxley model because there are fewer nonlinear differential equations, to be more precise, it is composed by:

$$C\frac{dV}{dt} = I + g_L(V - V_L) + g_{Ca}M_{SS}(V - V_{Ca}) + g_K N(V - V_K) \tag{4.13}$$

$$\frac{N}{t} = \frac{N_{SS} - N}{\tau_N} \tag{4.14}$$

where:

$$M_{SS} = \frac{1}{2}(1 + tanh[\frac{V - V_1}{V_2}]) \tag{4.15}$$

$$N_{SS} = \frac{1}{2}(1 + tanh[\frac{V - V_3}{V_4}]) \tag{4.16}$$

$$\tau_N = \frac{1}{(\phi cosh[\frac{V - V_3}{2V_4}])} \tag{4.17}$$

Qualitatively, this system of equations describes the complex relationship between membrane potential and the activation of ion channels within the membrane: the potential depends on the activity of the ion channels, and the activity of the ion channels depends on the voltage [37].

4.1.4 Perfect Integrate-and-Fire

The *IF* neuron is also known as perfect integrate-and-fire (or non-leaky integrate-and-fire). It is the simpler spiking neuron and it was introduced by Louis Lapicque in 1907 [29]. Similarly to the one from Hodgkin-Huxley, its membrane voltage evolves with time during the stimulation accordingly to the input current, following:

$$I(t) = C\frac{dV(t)}{dt} \tag{4.18}$$

that is the time derivative of the law of capacitance, $Q = CV$, as the current value varies there is an increment of the membrane voltage until a certain threshold value V_{th} is reached. At this moment a spike (Dirac delta function spike or unit impulse) is generated, consequently the voltage value is reduced to its resting potential. The accuracy of the model can be increased by the introduction of a refractory period t_{ref} in which it is impossible to fire another spike. The fire frequency can be expressed as follows:

$$f(I) = \frac{I}{CV_{th} + t_{ref}I} \tag{4.19}$$

4.1.5 Leaky Integrate-and-Fire

The previous model does not take into account the natural leakage that there is in a biological neuron, to the previous function that describe the variation of the membrane potential, it is possible to add a term corresponding to the leak in order to increment its biological faithfulness.

$$C_m \frac{dV_m(t)}{dt} = I(t) - \frac{V_m(t)}{R_m} \tag{4.20}$$

where $V_m(t)$ is the voltage across the cell membrane and R_m is the membrane resistance responsible for the leakage (when $R_m \longrightarrow \infty$ the LIF neuron reduces itself to the IF neuron). Like before, when a certain value is reached by the voltage, a spike is generated and the membrane potential is reset. In this case the minimum input current needed to reach V_{th} is:

$$I_{th} = \frac{V_{th}}{R_m} \tag{4.21}$$

As a conserquence, the firing rate can be re-written as:

$$f(I) = \begin{cases} 0, & I \leq I_{th} \\ [t_{ref} - R_m C_m \log(1 - \frac{V_{th}}{IR_m})]^{-1}, & I > I_{th} \end{cases} \tag{4.22}$$

Also for this equation, it can be seen that for large input current it converges to the IF model with t_ref [38]. One disadvantage of the LIF neuron is that it does not contain neuronal adaptation.

4.1.6 Izhikevich Model

Between the Leaky Integrate-and-Fire and the Hodgkin-Huxley lies the model proposed by Eugene M. Izhikevic in 2003 [39]. The Izhikevich Model combines the biological plausibility of the HH model's dynamics to the computational efficiency

of the LIF neurons [40]. It is represented by a two dimensional system composed by the following ordinary differential equations:

$$C_m \frac{dV_m}{dt} = k(V_m - E_L)(V_m - V_{th}) - u + I_{syn}(t) \tag{4.23}$$

$$\frac{du(t)}{dt} = a(b(V_m - E_m) - u) \tag{4.24}$$

Also in this situation the membrane potential is reset after the spike emission, that happens when V_m reaches V_{th}. The power of this model is that, with the right parameters it can mimic the firing patterns of all known types of cortical neurons[41].

4.1.7 Adaptive Exponential Integrate-and-Fire

When the neuron is subjected to a constant input current, the temporal window between two consecutive spikes increases over time, this phenomena is called neuronal adaptation. An Adaptive Integrate-and-Fire is the results of the combination between the LIF neuron and one or more adaptation variables. The aforementioned Izhikevich neuron could be classified as an adaptive quadratic integrate-and-fire model. In this model there is an exponential voltage dependence that, when coupled with a slow variable, can modify the threshold adaptation. It can be described by the following set of equations:

$$C_m \frac{dV_m}{dt} = -G_L(V_m - E_L) + G_L \Delta_T \exp \frac{V_m - V_T}{\Delta_T} - w + I_{syn} \tag{4.25}$$

$$\tau_w \frac{dw}{dt} = a(V_m - E_L) - w \tag{4.26}$$

with reset conditions:

$$\text{if } V_m \geq V_{th} \text{then } \begin{cases} V_m \longleftarrow V_{reset} \\ w \longleftarrow w + b \end{cases} \tag{4.27}$$

The variable in the equations have the following meaning:

- w: the slow variable that make the consideration of the adaptation possible,
- V_T: rheobase current, i.e. *the minimal electrical current that is necessary to elicit an action potential when current is injected into a cell* [42],
- Δ_T: to this variable is attributable the modification of the sharpness of the sodium channels' activation function.

Chapter 5

Spiking Neural Network

5.1 A new generation of neural networks

As we have already seen in the last chapter, it becomes clear the need to divide neural networks into three different category, or better, into three different *generations* [43].

5.1.1 The first generation

The McCulloch-Pitts's *perceptrons* [44] are computational units with the characteristic feature of a binary variable as output, used in the *Multi Layer Perceptrons* architecture. The value assumed by the output can be describe as follows:

$$y = f\left(\sum_{i=1}^{N_I} w_i x_i - \theta\right) = \begin{cases} 1, & \text{if } \sum_{i=1}^{N_I} w_i x_i - \theta \geq 0 \\ 0, & \text{if } \sum_{i=1}^{N_I} w_i x_i - \theta < 0 \end{cases} \tag{5.1}$$

The output, since it is binary, can only assume value in the set $\{0, 1\}$, with $f()$ identifying the activation function (in this case can also been called state transition function, N_I is the number of input neurons, x_i the single i-th input of the neuron while w_i is the associated synaptic weight, θ is the activation threshold.

5.1.2 The second generation

Surely more valuable and reliable are the neurons belonging to the second generation, the artificial neurons are slightly different from the perceptrons. Now the activation function is continuous, this neurons are capable of processing real input and output, encoding in this way more information in the process. Neurons belonging to this generation can be written as:

$$y = f\left(\sum_{i=1}^{N_I} w_i x_i + b\right) \tag{5.2}$$

where $b \in \mathbb{R}$ is the bias. While in the previous generation the representative computational model was the MLP, now it is the feed-forward neural network trainable with back-propagation[43]. They are for sure powerful networks but there is an important bottleneck: the artificial neurons does not take into account the temporal information of individual spikes [43].

5.1.3 The third generation

The last category is the one of biological plausible neural networks, that are Spiking Neural Networks. They are based on spiking neurons, basic computational units that can be described with a hybrid formalism [45]

$$\begin{cases} \frac{dX}{dt} = f(X), \\ X \longleftarrow g_i(X), \end{cases} \tag{5.3}$$

where X is a vector consisting of state variables of the neuron, $f()$ represents the differential equations for the evolution of the state variables and $g_i()$ describes the change of the state variables caused by spikes events from the synapse i. Since this system take into consideration precisely timed spike trains, it is more representative of the biological nervous system than ANNs, achieving efficient information processing [43].

5.2 SNN Architecture

In the previous chapters where given definitions for both artificial neural network and neuron model that can approximate the functioning of biological neuron. But why SNN are so useful and innovative? let us take into consideration some factors:

- Spikes: contrary to what happens in ANN, SNNs use spikes instead of high-precision values, this make the spike-approach heavily efficient trading the wide number of multiplications with simple memory read-out of the weight value[5].

- Sparsity: as biological neuron spends most of its life in a quiet state, also the data structures in which are contained spike are equally sparse, this implies that much less memory is used to store sparse matrices.

- Static Suppression: also known as event-driven processing, SNNs enhance the usage of sensory system that by nature have a much higher responsiveness to changes than conventional hardware that work with static input. As we will see in 11, DVS cameras are a real-world example of static suppression, with them it is possible to achieve enormous energy savings due to the reduction of the number of active pixels.

While leaving to 11 the burden of dealing with input and output decoding, we will now focus on how SNNs are able to learn, addressing the difference between ANN and SNN with respect to Objective Function, Learning Rules and Backpropagation.

5.2.1 Objective Function in SNNs

The loss functions seen in 2 are far from being biologically accurate, nature acts in a way much more similar to reward functions seen in reinforcement learning, such us releasing dopamine while some actions causes pleasure [5]. While dealing with SNNs, what can be made is the translation of canonical loss functions into their spiking counterparts, they can be divided into:

- Spike Rate Objectives Functions: in these case correct neuron class are promote to fire with higher frequency. In this class of objective functions, they can be found spiking version of both CE-Loss and MSE, with the foresight of using versions that rely on Spike Count while dealing with a large number of time steps and versions that use Membrane Potential in situations where the number of time steps is constrained.
- Spike Time Objectives Functions: as the name lets guess, this class leverages the time characteristic of spikes instead of their frequency. Unfortunately, these functions are far from being used as the previous ones, perhaps for the following reasons:
    1. rate codes are more tolerant to noise and, since in ML literature error rates are perceived as the principle metric, this leads to choose the previous one.
    2. moreover, from a technical point of view, temporal codes are much more difficult to implement.

5.2.2 How does SNN learn?

After the choice of the loss function, it has to be addressed how the network will learn, i.e. how the weight will be updated. While doing the so called "credit assignment", identifying how much each neuron competes in learning, there are two factors that need to be considered:

- Spatial Credit Assignments: focused on finding the location of the weights that contribute to the error.
- Temporal Credit Assignments: focused on finding where the weights contribute to the error.

In ANN, backpropagation has established itself as a valid method to address the problem of credit assignments to the weights but, other than the fact that it is not biologically plausible, there is only one problem: while dealing with spikes, backpropagation is not feasible.

The unfeasibility of backpropagation is caused by the dead neuron problem: the non-differentiability of spikes results in a gradient that does not enable the learning procedure. This was one of the major blocking points in developing SNN, in order to stem it, different methods were born.

- Shadow Training: consists on training a shadow ANN and the convert it into a SNN, this make possible to apply canonical advances in DL directly to SNN [5]. It is an optimal choice when applied to static classification tasks but it is strictly constrained to the task's nature since it does not exploit the temporal dynamics of SNNs.
- Backpropagation Using Spike Times: it is the first method proposed in the timeline, the intuition behind it resides in the fact that time, contrary of spikes, is a continuous variable and so it can be differentiated. Even if it had remarkable results in data-driven tasks, it entails several drawbacks.
- Backpropagation Through Time: BPTT has soon become one of the most commonly adopted method to trait backpropagation in SNNs, it consists in applying a generalized backpropagation algorithm to the unrolled computational graph.
- Surrogate Gradient: another way to overcome the dead neuron problem is the surrogate gradient, i.e. approximating the gradient to some term that not becomes null. This is done by replacing a non differentiable term in the equation with a differentiable one (such as threshold-shifted sigmoid function). It achieved successful results also due to the variety of surrogate gradients present in literature.

In this work of thesis surrogate gradient will be adopted as the backpropagation method due to the possibility to trait it as an hyperparameter during the design space exploration.

Chapter 6

Design Space Exploration

In the past chapters the topic of hyper-parameters tuning and how does it play an important role during the optimization of a neural network has been presented. This can be seen as the extensive exploitation of some configuration of the net. What if we want to explore the space of possible configurations while searching for an optimal solution?

6.1 Exploration-Exploitation Dilemma

The trade-off between exploration and exploitation is a multidisciplinary concept that become useful when choosing the way to search an optimal solution [46]. In this dilemma we have two solution:

- Search the perfection in a pre-existing system, following in this way the road of full exploitation of a solution, trying to push as much as possible its performance.

- Explore new possible solution. Sometimes less-performing configurations will be found, while in other cases, the desired ones, a new system that could achieve the task in a much more efficient way can be obtained.

Exploration is more suited for solution that will take place more in future since there is no immediate additional value and costs are much higher in this phase. While exploitation will continue to give results in the short period and then fade away over time. Since the goal is not a simple hyper-parameters optimization of a neural network, next sections dive into the field of Design Space Exploration.

6.2 Design Space Exploration

As presented before, in this work of thesis we want to optimize the architecture of a Spiking Neural Network in order to deploy the model on an ad-hoc configured FPGAs. This could be extremely different and to obtain the most general tool we need to take into account different objectives, with different constraints each. We have to face a multi-objectives optimization problem (MOP) since we have to consider performance, power/energy consumption and latency simultaneously. Our objectives are often in conflict, if high-performance is desired, rarely it will be achieved with low consumption and latency. We have to deal with multi-objective DSE, finding all the optimal solutions that need to be taken in the presence of trade-offs between criteria[47].

Initially we have a set of variables corresponding to the parameters (such as learning rate, beta, type of neuron etc.) that we can vary, these are the degrees of freedom explored during the DSE. The goal is to optimize the so-called fitness function on the n objectives. Formally, given m decision variables, n objectives, the fitness function is defined as follows:

$$f_i : R^m \rightarrow R^1(1) \tag{6.1}$$

Each realization of the space R^m corresponds to the assignment of m decision variables, equally speaking, it is a potential solution for our system. The fitness function in Equation 6.2 put in relation a combination of assigned variables to a point in the objective space, each f_i function refers to the i-th objective that we want to optimize.

When the search takes into account only a single-objective, the results are already non-trivial: major accuracy stands for better performance. But now we have to deal with multiple objectives simultaneously, let now formalize the previously mentioned MOP[47]:

$$y = f(x) = (f_1(x), f_2(x), ..., f_n(x)) \tag{6.2}$$

$$x = (x_1, x_2, ..., x_m) \in X(2) \tag{6.3}$$

$$y = (y_1, y_2, ..., y_n) \in Y(3) \tag{6.4}$$

For each configuration in 6.3 we will obtain a result (6.4) that is a tuple corresponding to the value of each objective assumed by our configuration. For a given system not all configurations are valid, for this reason we could think about putting constraints on the parameters that define our design space, in this way not only we avoid invalid configuration but we reduce also the search space. The valid

configuration x_i belong to the so-called feasible set. The goal is to minimize 6.2 but, without loss of generality[47], we can multiply y_i by -1 facing in this way a maximization problem.

Therefore, an efficient method to represents data related to a multi-objective search is needed in order to easily compare the obtained results. Keep in mind that results are not unambiguous, some outcomes will be more useful in certain cases more than other and vice versa. We can adopt the Pareto Front as an effective representations of our results. Formally speaking, it is the set of all Pareto efficient solutions [48], i.e. a situation where no action or allocation is available that makes one individual better off without making another worse[49]. Relating to our problem, consider: function $f : X \longrightarrow R^m$, a set X of feasible decisions in the metric space R^n and a set Y of feasible criterion vectors in R^n, we have:

$$Y = y \in R^n : y = f(x), x \in X \tag{6.5}$$

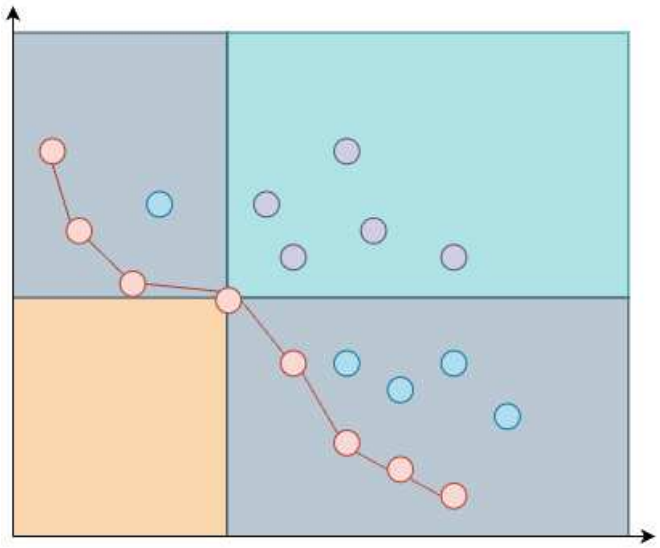


Figure 6.1: Representations of point in the Pareto Front.

In 6.1, the Pareto Frontier is represented by the red dots, it is interesting how, by taking into account one of them, it can be deduced that on right upper side there are the points that are dominated, i.e. it is been said that a point dominate the others in the graph if it is not possible to find points in the graph that outclass in both metrics the dominating points on the pareto frontier, on the left lower side the points that dominate while for the remaining parts of the so-divided graph points are not comparable (some results could be better while others worse).

6.2.1 DSE Needs

When projecting a Design Space Exploration we have to address to at least two major problem [50, 51]:

- How it can be evaluated a single point with respect to the fitness function, considering simultaneously all the objectives?
- Now that it is known how to evaluate a single point in the space, how can the space itself be navigated? Which strategy needs to be followed?

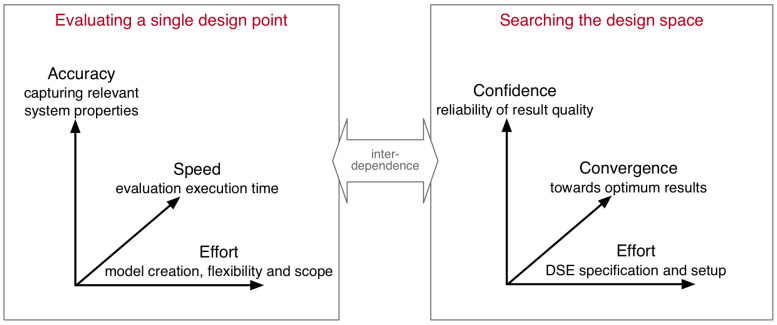


Figure 6.2: In the figure it can be appreciated the taxonomy of Design Space Exploration with its dual problem, the conflict between evaluation and search. [47]

Evaluation of a single design point

While evaluating a single configuration one can choose between different roads, maybe the one most accurate is the deployment on the final system of the model in order to evaluate directly on the system its performance and consumption, but it is fairly clear that this is impossible in term of practicality and time efficiency. One another solution could be act on another level of abstraction, creating a plausible simulation and trait the evaluation only from a software point of view. For our specific case it will be used the second one, more on the methodology in Chapter 10.

Searching the design space

Even if the previously problem seems problematic, the efficient searching of the design space is for sure more challenging. Also in this situation we have to face a trade-off problem, now between convergence, effort and confidence. We could have the best solution by scanning the entirety of the search space but it is obviously not feasible in terms of time and cost. Without known informations about model's

optimal solution, metaheuristic procedures can be exploited in order to effectively scan the target search space. Below are listed the three most used heuristics.

- **Bayesian Optimization**: largely used to optimize the hyperparameters of a ML model in an efficient way, can also be applied to DSE. The idea behind is to map the architecture to its validation error through the objective function. BO uses a surrogate to model this objective function based on the previously obtained architectures and validation errors. Then, the next architecture to evaluate is chosen by maximizing an acquisition function, providing a balance between exploration and exploitation. Objective function and acquisition function maximization could be computationally expensive while applied to DSE.

- **Genetic Algorithms**: a genetic algorithm is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm. It uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions. Evolution of the population then takes place after the repeated application of the above operators.

- **Reinforcement Learning**: it is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. As seen before, reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. Reinforcement Learning can be considered as a DSE search strategy.

6.2.2 Neural Architecture Search

Until now we talked about DSE but, while referring to machine learning and deep learning it is not properly correct to refer to the search of a neural architecture only as design space exploration. Novelty literature has taken into account the Neural Architecture Search as a revolutionary approach in DL models designing process. As we have seen in previous chapters, AI field is in constant evolution an letting humans introduce a cognitive bias during the search of optimal neural model could lead to sub-optimal solutions, for this reason in the past years emerged the willing to automatize the process of research of the neural architecture of a model, this field, called Auto ML (or Auto DL depending on the literature taken into consideration), can be seen as a subset of DSE since the latter is used also in more disparate research fields.

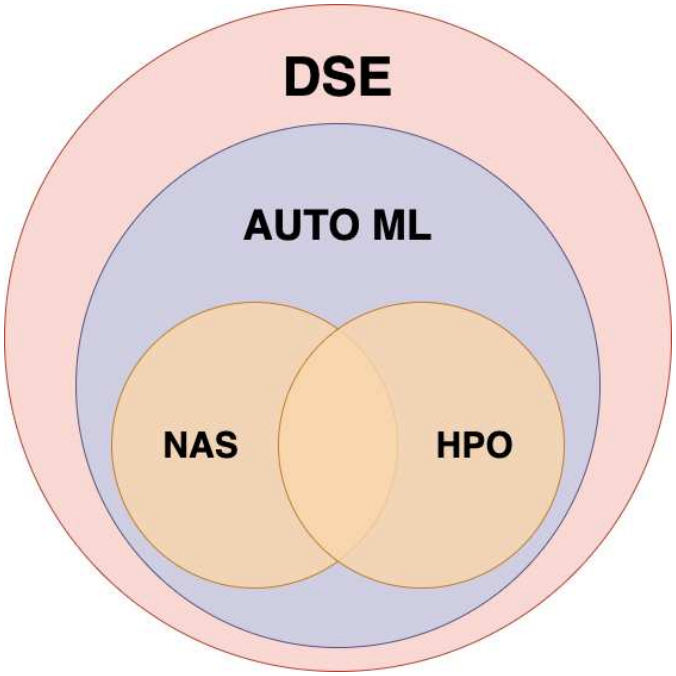


Figure 6.3: A graphical representation of the hierarchy that exists between Design Space Exploration, Auto ML, Neural Architecture Search and Hyper-Parameters Optimization.

NAS can be seen as subfield of AutoML and has significant overlap with hyper-parameter optimization[52]. Methods for NAS are cathegorized according to three dimensions: search space, search strategy and performance estimation strategy.

Search Space

In principle, the search space tells us what architectures can be captured by this approach. Typical attributes of an artificial neural network architecture are available in a database of prior knowledge and are used to guide its design and evolution.

By using architectures that are task-specific, we can reduce the search space and complexity. This, in turn, is able to introduce human bias that precludes the discovery of new architectural building blocks that do not conform with current human knowledge.

Search Strategy

The search algorithm shows the way to traverse a space (which could be very large, possibly exponential or infinite). It is characterized by the classical exploration-exploitation dilemma as it must seek a balance between finding highly successful architectures quickly and premature convergence on suboptimal areas.

Performance Estimation Strategy

Estimation of performance is the process that guides NAS in seeking architectures that will deliver high predictive performance based on unseen data. The simplest way is through standard training and validation, but this becomes computationally expensive and limits the number of architectures that can be tested. As such, much of the current research has focused on finding methods to reduce these estimations in terms of cost.

Chapter 7

Search Space

7.1 Our Search Space

Is design space exploration really needed in our case? Perhaps, traditional methods could be implemented but trying to automatize the process can only bring some help in our task, this because while talking about SNN and their optimization on FPGA, the dimensionality of the design space could get out of hand pretty easily. In this chapter, more attention will be paid to what are the parameters of interest in our research, for convenience they will be divided into three macro-section.

7.1.1 Architectural Parameters

One of the most tricky parameter to tune is for sure the anatomy of the model itself. Choose the right number of hidden layer and for each the right number of neurons could lead to an expensive and extensive search. In past decades this decision was made by researchers, but nowadays it is unthinkable to manually decide what is the best configuration, moreover some solution could not be even explored and some could not be exploited enough. For this reason we decided to include in the process of AutoML also this kind of variables, letting our algorithm to decide instead of us.

Along with the number of layers and neurons, it is interesting to know how a particular architecture reacts to a given test. To be more precise, given a dataset sometimes could be more effective a to use a single hidden layer instead of multiple layers, or can be tuned the dropout of the weights between layers or again could be evaluated how the network reacts to the presence or absence of recursion between the neuron or the layer in its entirety. All this kind of choices form the macro area of the architectural parameters.

7.1.2 Neuron Models

Even if it is only a single parameter, there is the need to trait the type of the neuron model to use as a family of parameters in its own right. As we have seen before, there are plenty of model that can be used to emulate the biological neuron. From the simplest one to the most biologically plausible, it is difficult to choose the right model that fits our use case scenario. Often we would like to try different models, with the exploration of a certain type of models we will maybe find a particular behaviour of the neuron that leads to optimal results.

Moreover, while targeting hardware implementations of the neuron models, we are interested in reducing the power consumption or the area on the memory occupied by a chosen model. This translates in the need of trying different plateau of different neurons each with $1 \longrightarrow n$ parameters to tune (such as the decay rate seen in chapter 4).

The type of neuron and its parameters composes the second family of parameters that we will take into account.

7.1.3 Training Parameters

As we have seen before, SNN can not be trained as usual neural networks due to the impossibility of using backpropagation. In order to mimic the back-prop algorithm, we have to use a differentiable surrogate model. Here, again, we have a multitude of models between the right choice could be. After this, due to the intrinsic nature of SNNs, different questions arise: should the threshold potential be trainable? Which number of time steps leads to best processing of the input dataset? Should the membrane potential decay rate be trainable? Which works better? And, if present, what instead for the synaptic current decay rate?

On top of the previous choices, then come the ones related to the classic neural network training: which parameters to use for the optimizer? Which loss performs the best for the given task?

It seems now clear the amount of computational power needed to evaluate all these parameters. For this reason, considering such an extensive design space, conventional research methods are not enough in our case.

Chapter 8

Bayesian Optimization

There exists a plateau of different optimization techniques that can be used during the design space exploration of a neural network, having different pros and cons each. Between these there is one capable of resolve complex engineering problems, with a number of parameters not so small while keeping the convergence speed high: it is the Bayesian Optimization.

With respect to our use case, as we have previously seen in the last chapter, we have to deal with a not so small search space. Conventional search methods such as grid search or random search could require more evaluations than is practically feasible, for this reason BO becomes a valuable searching method. Let us take into account a practical example in order to highlight the difference between a standard method like grid search and BO, consider an experiment having 4 tunable parameters:

Country List		
Num. of Parameters	Grid Search	BO
Neuron Type	4	4
Alpha	5	5
Beta	5	5
Layers Configuration	18	18
Total evaluation	1800	$\simeq 20$

Table 8.1: Example of Bayesian setup

On top of the mere difference in absolute terms between the total numbers of evaluations, if we consider that a single evaluation can take between 5 to 60 minutes, Bayesian Optimization can save up to 74 days just by looking to a simple case like this. It becomes pretty clear that BO is applicable to problems such as [53]:

- Hyperparameter optimization fro ML,
- Tuning design parameters and rule-of-thumb heuristics for hardware design.

These are some of the uses of BO, and they were chosen since are particularly relatable to our use case.

8.1 How does BO work?

BO is an iterative algorithm that allows to find the optimal parameters configuration through a relatively small number of iterations by refining an initial surrogate model. It is an adaptive approach that updates the parameters to optimize after the observations of previous evaluations.

8.1.1 Gaussian Processes

Everything starts with the build of the surrogate model through Gaussian processes (GPs). GPs are stochastic processes in which every collection of some random variables has a multivariate normal distribution. This is equivalent to say that, for every finite set $(X_{t_1}, ..., X_{t_n})$ of indices $(t_1, ..., t_n)$, each linear combination associated to it has a univariate normal (Gaussian) distribution.

It comes particularly useful while talking about Bayesian Optimization because it can be used to compute the prior probability distribution [54]. GPs are simple and powerful methods to impose assumptions over some unknown function through the usage of a probability distribution [53]. In this way it is possible to create families of functions characterized by:

- A mean function, i.e. the average of all functions
- A covariance or kernel function that has the task to explain how functions belonging to the family can vary around the mean function, it carries information about the smoothness and the shape of the individual functions.

As we said, kernels are fundamental to determine the shape of the prior and posterior of GPs, this information is encoded in the assumption that similar datapoints return similar objective points [55]. They can be divided into: (i) stationary kernels, that rely only on the distance between two given datapoints, becoming invariant to translations in the input space; and (ii) non-stationary kernels that, on the contrary, varies with the values of the input data too. Different kernels can be chosen with respect to the type of data that need to be elaborated, let us dive deep into some of the main kernels.

Radial Basis Function

Between stationary kernels, one of the most used is the (RBF), also known as squared potential. Formally, the kernel can be written as:

$$k(x_i, x_j) = exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right) \tag{8.1}$$

where the length-scale parameter l, can be either a positive scalar or a positive vector (with number of dimensions equals to the same number of dimensions of the input x) based on the need of a isotropic kernel, i.e. a kernel invariant to rotations in the input space, or anisotropic kernel which varies, on the contrary, according to rotations in the input space. While $d(\cdot, \cdot)$, as we can imagine, indicates the Euclidean distance. The advantage of RBF is that it is infinitely differentiable, causing consequently that GPs with this kernel have derivatives of all orders, resulting in a very smooth prior (see 8.1).

Matérn Kernel

Matérn Kernel, along with the length-scale parameter l, introduces ν, which controls the smoothness of the resulting function. It is a generalization of the RBF, in fact when $\nu \longrightarrow \infty$, Matérn Kernel tends to RBF.

8.1.2 Acquisition functions

Along with the surrogate model, BO needs another element in order to work: an acquisition function. It is in charge of quantifying the value observed by a particular parameterization. Then, it is updated in order to find the best observable configuration, causing a new surrogate model to be fitted and re-starting the cycle. In this way, predictions and uncertainty are updated iteratively.

Another focal point of using an iterative method is trying to achieve the best possible result in the exploration vs. exploitation trade-off.

While choosing the right acquisition function to evaluate the loss of our tasks, we have different possibilities.

Probability of Improvement

Perhaps, the first acquisition function designed purposely for BO was the probability of improvement (PoI). Let us indicate with f' the minimum value observed for a function f. PoI evaluates f where it is more likely to obtain an improvement on this point. The utility function, i.e. the negative loss function, returns a unit reward when it is found a value smaller than f' and zero otherwise, formally it can be written as:

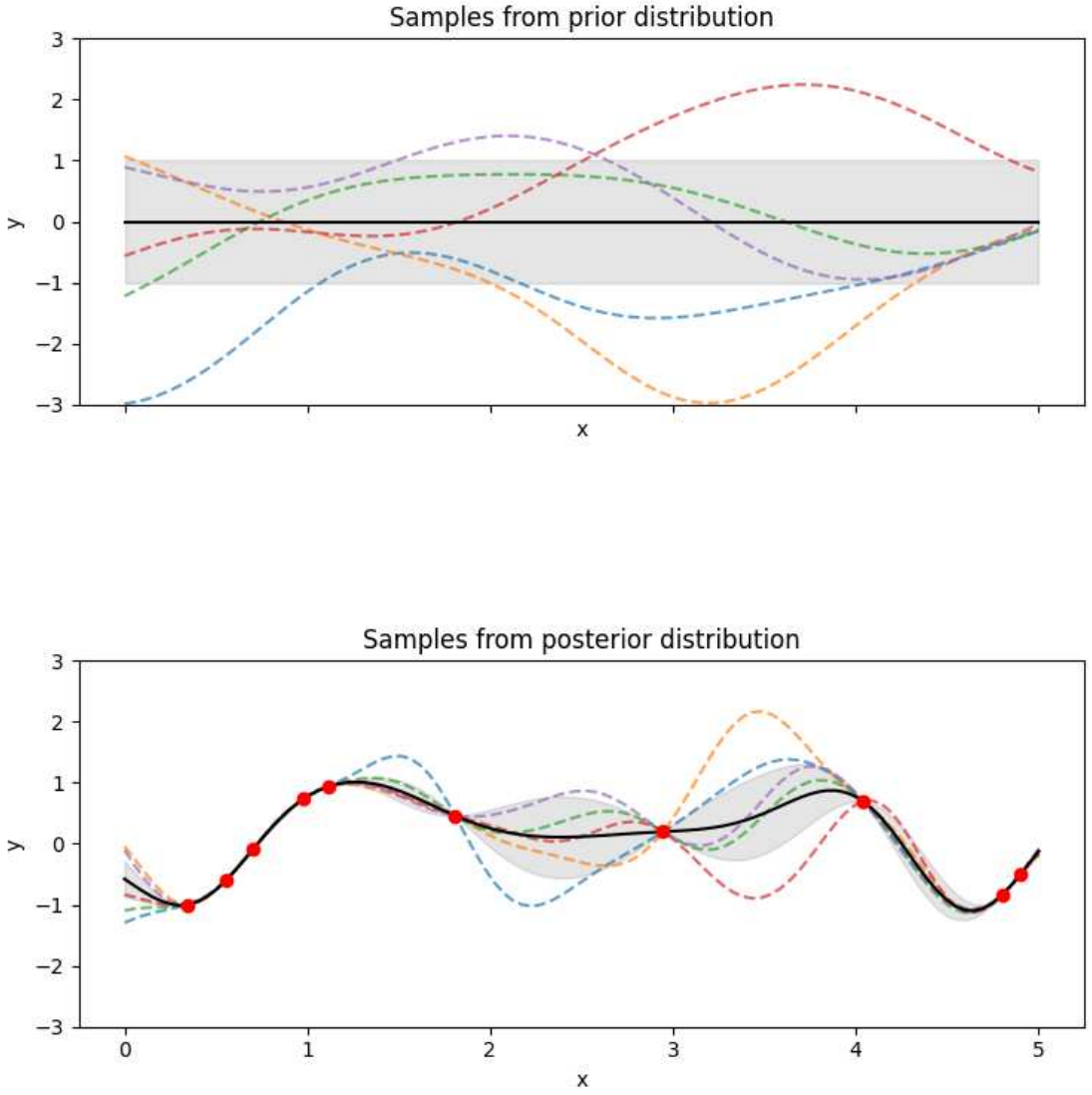


Figure 8.1: Prior and posterior representations of GPs with RBF created through the usage of *scikit-learn* library. In the image, with the dashed lines are represented the samples functions while with the black filled line the mean. The grey area indicates the confidence interval with respect to 1 standard deviation and with the red dots are indicated the observed point of the underlying function.(68.7%)[56]

$$u(x) = \begin{cases} 0, & \text{if } f(x) > f' \\ 1 & \text{if } f(x) \leq f' \end{cases} \tag{8.2}$$

It follows that the PoI acquisition function is the expected utility as a function

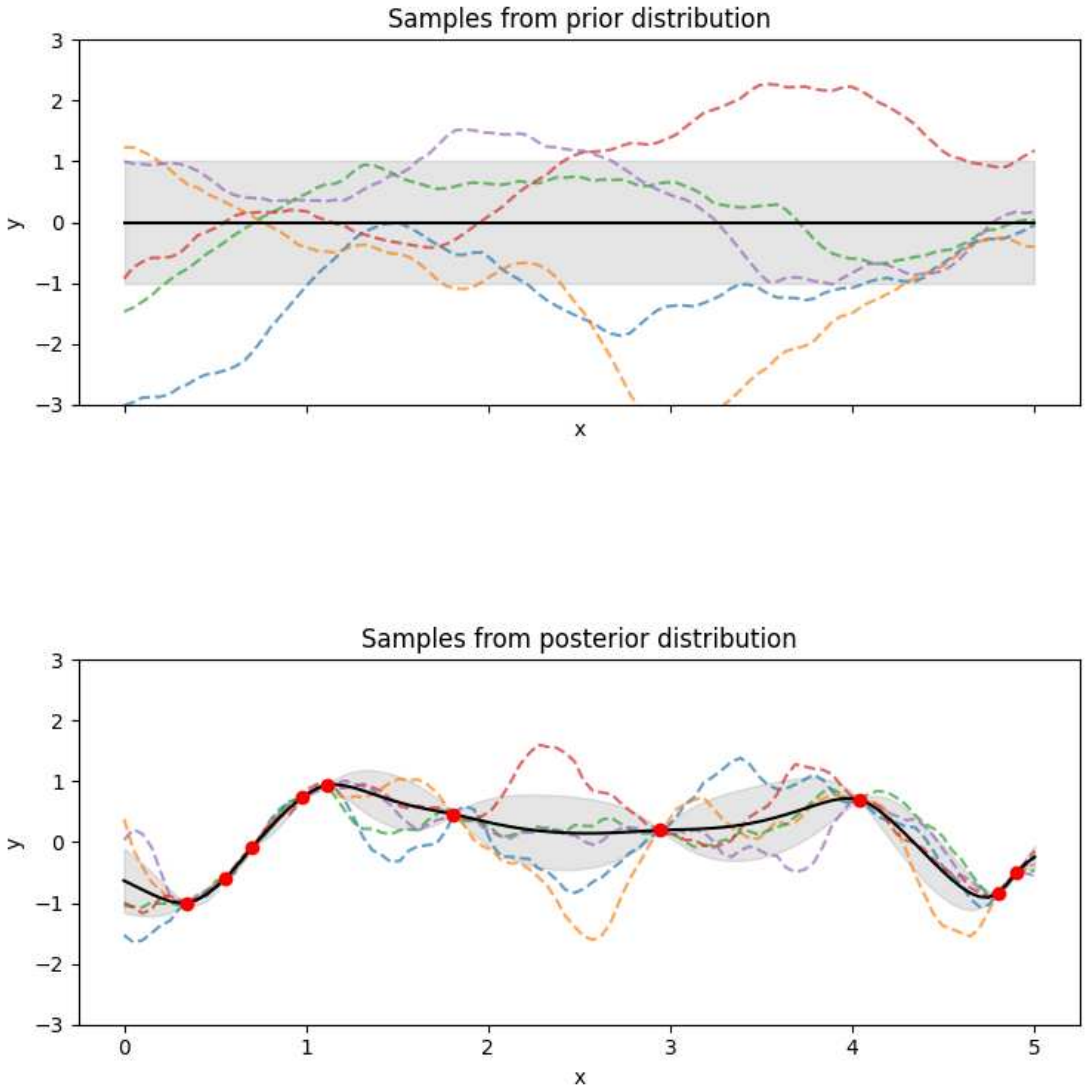


Figure 8.2: Prior and posterior representations of GPs with Matérn kernel created through the usage of *scikit-learn* library. Legend equal to the previous image. [56]

of x [57]:

$$\begin{aligned} a_{PI}(x) &= E[u(x)|x, D] \\ &= \int_{-infty}^{f'} N(f; \mu(x), \sigma^2(x))df \\ &= \Phi(f'; \mu(x), \sigma^2(x)) \end{aligned} \tag{8.3}$$

The point with the maximum expected probability of improvement is chosen.

Expected Improvement

Sometimes we would like to use a loss function associated to PoI but this leads to some tricky implementations, since the reward gained does not vary with respect to the size of improvement.

But there is another acquisition function capable of accounting for the size of improvement: expected improvement. In this situation, the utility function becomes:

$$u(x) = max(0, f' - f(x)) \tag{8.4}$$

i.e. the reward associated to EI is equal to the distance between f' and $f(x)$ (the improvement) if a new minimum is found and zero otherwise. The correlated acquisition function can be written as:

$$\begin{aligned} a_{EI}(x) &= E[u(x)|x, D] \\ &= \int_{-infty}^{f'} (f' - f) N(f; \mu(x), \sigma^2(x)) df \\ &= (f' - \mu(x))\Phi(f'; \mu(x), \sigma^2(x)) + \sigma^2(x) N(f'; \mu(x), \sigma^2(x)) \end{aligned} \tag{8.5}$$

As we can see, EI acquisition function is composed by two components: the first one can be increased by reducing the mean, while the second one can be increased by increasing the variance. This is one of the more tangible mathematical implementations of the trade-off between exploration and exploitation.

Chapter 9

Implementation Choices

9.1 Why Python?

Python is an exceptional object-oriented, interpreted and interactive programming language [58]. The power of Python resides into its dynamic approach to writing code. Differently from Java or C it is not highly typed, instead its variables are fluid and again dynamic. The low complexity, its remarkable power and its very clear syntax made of it a standard in the developer's community. Furthermore, it is open-source.

But why it is so popular in machine learning tasks? Due to its versatility, in the last decade a considerable amount of ML libraries spread out, between these, TensorFlow and PyTorch [59, 60] are by far the two most used ones, in fact the latter will be used also in this thesis work due to its compatibility with most SNN module. The reasoning of why Python was the favorite choice for ML applications back in the days is not only because of its high-level abstraction and efficiency, there were other players competing in this game. While trying to optimize their products, Yahoo and Google had to choose the best performing language to simplify the developers life. The first went with the most famous language at that time Perl while the latter choose the most innovative one: Python. We all know who won between the two competitors, causing an heavy amount of investments in Python, leading a shift in the machine learning community towards this language.

9.2 Why Adaptive eXperimentation?

AX or Adaptive eXperimentation is a product developed and owned by Meta ®. Ax is an optimization platform able to perform any kind of experiment such as A/B Tests, simulation and machine learning related experiments. When dealing with discrete search spaces, Ax exploit the power of multi-armed bandit optimization

while Bayesian Optimization is used in its continuous counterpart, this allows Ax to be modular and adaptive to any situation.

It has unique capabilities [53]:

- **Support for noisy functions**: in A/B tests and simulations, it can be observed a considerable amount of noise while using reinforcement learning agents, this is leverage by Ax through the use of SOTA algorithms that have better performance with respect to traditional BO
- **Customization**: by dividing this product into three different usable Application Programming Interfaces (APIs), the developer made possible to offer different level of customization, spacing from an high-level abstraction API that is ready to use to a fully customizable API that can be used in the hardest experiments
- **Multi-modal experimentation**: Ax gives first-class support while working with offline and online data in order to run and combine data from different types of experiments
- **Multi-objective optimization**: one of the greatest strengths of Ax is for sure the possibility to run multi-objective optimizations that support constraints on objectives, pretty common in real-world scenarios.

From these capabilities, it is straightforward the decision to use Ax has a powerful framework on which to base SpikeExplorer, because the customization and multi-objective optimization will be useful to us.

9.2.1 Ax's APIs

As we mentioned before, Ax is composed by three different APIs. Let's dive into the principal concepts that characterize each one [53].

Loop API

While dealing with the simplest experiments, the Loop API is an easy choice. With this interface we can setup, launch and evaluate the results of an experiment without knowing anything of the underlying structure. It was thought with a high-level of abstraction in order to offer BO and multi-armed bandit even to who does not know the mathematics behind them. In this way, someone like a manager can taste easily the power of Ax, using it out-of-the-box and having (semi) instantaneously some results on which base his thesis.

Service API

Much more popular and powerful is Service API: even if it does not unlock the full potential of Ax, it can be use to leverage most of the hyperparameter optimization functionalities without knowing what happens under the hood [53]. In this lightweight service, trials can be evaluated in parallel keeping the data available in an asynchronous fashion.

Developer API

The most customizable interface is the Developer API, great in the research field. It is recommended for who need to run A/B tests, it is the most powerful between the three with the drawback to have a complex learning curve, with the need of deeply understand the structures and capabilities of Ax in its entirety. It is characterized by the presence of a scheduler capable of run configurable experiments with closed-loop optimization without any human intervention.
For our use case, the Service API is more than sufficient.

Chapter 10

SpikeExplorer Architecture

After a large explanation of the problem and components needed to solve it, now it is finally the moment to take into account the architecture of our solution, *SpikeExplorer*. It is structurally divided into three different modules that co-operate in order to compute the best configurations for the specific use case. Along with its main framework, there is a standalone module to make a visual representation of the best performing architectures found during the experiment. This chapter will discuss about the first three while the next will be used while talking about the metrics and results. Figure 10.1 shows a graphical representation of SpikeExplorer.

From a theoretical point of view, the solution was divided into different logic modules even if they are not in the same portion of code and classes in the project, thus simplifying the explanation of SpikeExplorer. Given an input, the DSE Engine interprets the objectives and the search space of the experiment, preparing the Network Evaluator (NE) trial by trial. At each iteration of the NE, a new model is generated and a new arm (a combination of parameters picked from the design space by Ax) is chosen. After the train and evaluate function, for each objective, a result is returned, these observations will allow the update of the surrogate model with the consecutive choice of new points that need to be observed.

10.1 DSE Engine

This module acts as an orchestrator with the ability to interconnect each module. Here, it is possible to control the scope of the experimentation through the usage of a input configuration file in JavaScript Object Notation (JSON) format. Through this file is possible to set global variables (such as experiment name, number of trials, number of epochs, dataset and other default values used for parameters not of interest during the run), objectives (as we will see later in this chapter, they are constrained to accuracy, area, power consumption and latency) and parameters to

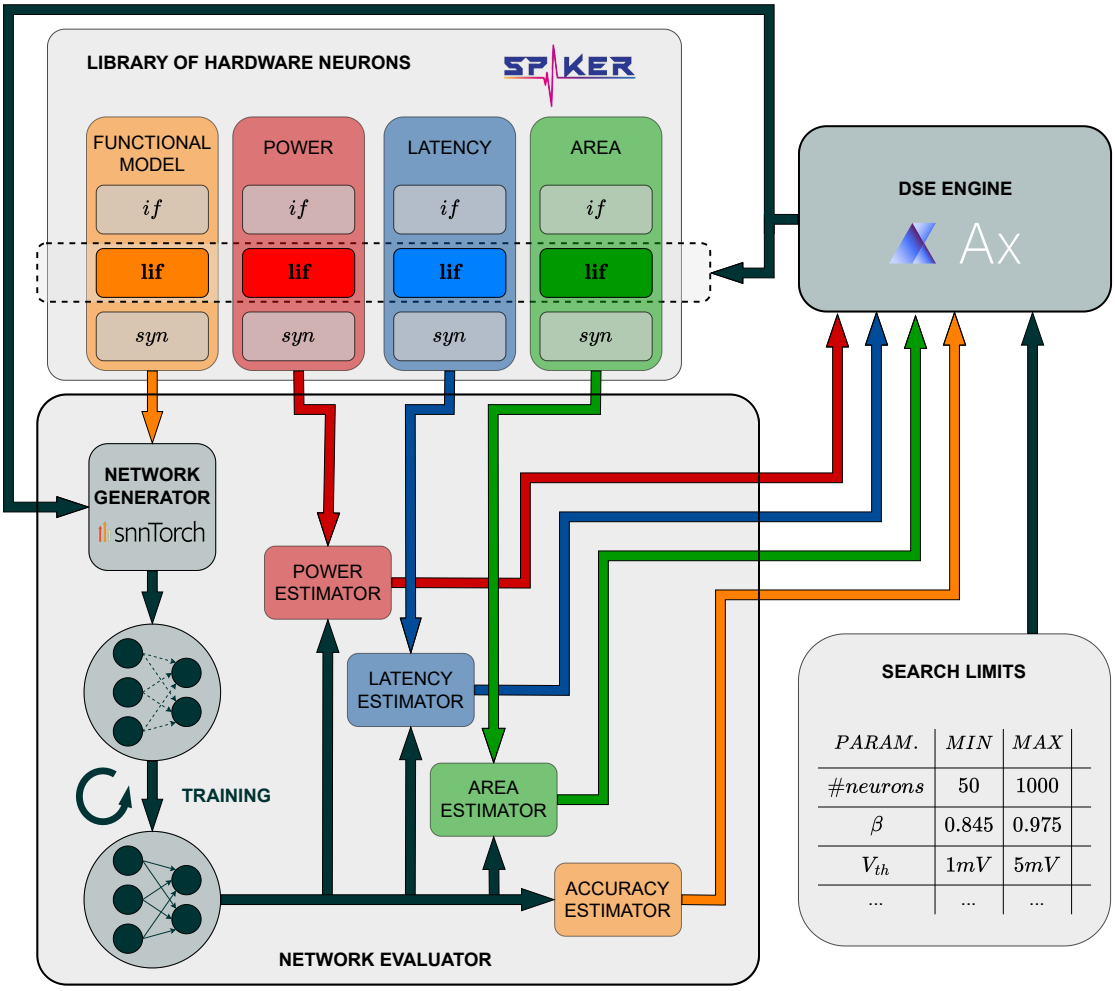


Figure 10.1: An high-level design of SpikeExplorer

search. Table 10.1 shows all the possible configurable variables of the experiment, the column "Required" can assume values Y, for variables that must be in the configuration, and N, for those variables that have a default value, usually the one used in the python libraries invoked.

As it has already been said, finding the best architecture usable on FPGA could be daunting. In order to simplify and automatize the process, SpikeExplorer relies on Ax Platform to implement the core of the DSE engine. It automatically identifies which Bayesian algorithm to use both for the Gaussian Process and the activation function, it is also prepared to discrete search space through the usage of a Bandit Optimizer.

List of configurable variables			
Variable Name	Type	Required	Description
name	string	Y	Name of the experiment
dataset	string	Y	Dataset on which run the experiment
overwrite_existing_experiment	boolean	Y	Ax parameter
is_test	boolean	Y	Ax parameter
num_trials	int	N	Number of trials, required if not present in parameters_to_search
batch_size	int	N	Batch size in which the dataset is split, required if not present in parameters_to_search
num_epochs	int	N	Epochs number, required if not present in parameters_to_search
num_steps	int	N	Time steps considered in the spiking neuron, required if not present in parameters_to_search
weight_memory_occupation	int	N	Required only if area is in the objectives
neurons_occupation	obj	N	Required only if area is in the objectives
objectives	list[obj]	Y	The list of objectives, has dimensionality $1 \longrightarrow 4$
parameters_to_search	list[obj]	Y	The list of parameters to search, has dimensionality $1 \longrightarrow n$
neuron_consumption	obj	N	It contains active and passive consumption values

Table 10.1

10.2 Network Evaluator

The next module to be invoked is the Network Evaluator, it is composed of three sub-modules, each having a specific task, they work together in order to complete the evaluation passage needed to continue with the iterative approach of the Bayesian optimization.

10.2.1 Network Generator

The network generator (NG) has a simple purpose, given a set of inputs formed by the parameters chosen by Ax, it is able to create a new model, dynamically choosing the layers and the neuron type and adapting the forward passage to it. The latter will invoke the functions needed to compute the metrics related to the forward passage only, i.e. power consumption since it depends on the membrane potential at each time steps, as we will see later on this chapter.

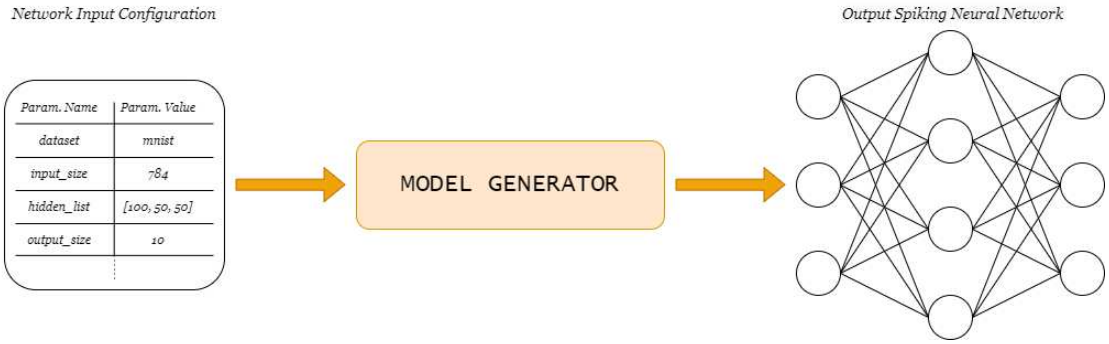


Figure 10.2: Representation of Network Generator

The generated SNN is ready to be passed to the train and evaluation function. Figure 10.2 presents a brief representation of the NG. The NG was designed to be maintainable and easy to update with future features. In the appendix it can be found the full set of parameters that can be accepted by the NG.

10.2.2 Train-Evaluate Function

It is pretty self-explanatory by the name, the *train-evaluate function* has to:

- Setup all the variable and use the relative fallback value if a parameter is not specified,
- Load the dataset
- Create the model
- Instantiate training optimizer (e.g. Adam) and loss functions
- Manage the training loop
- Manage the evaluation passage
- Compute and return objective metrics to Ax

Some items on this list, at the eye of an expert developer may seems useless and redundant to replicate at each trial but there is a reason behind this choice and it resides in the goal of SpikeExplorer: the tool aims to achieve maximum generalization during the exploration of the best design. Since Ax needs a function that acts as a black box (given an arm, a set of resulting metrics is returned, what happens in the between does not matter) for its functioning, we had to integrate expensive passages such as the dataset load and the model creation into this function. In this way, some performances were traded in favor of a much higher degree of freedom.

10.2.3 Metrics

SpikeExplorer has been set up for the management of up to 4 objectives at the same time, they are: accuracy, latency, area and power consumption. We decided to focus on these four since they are a common way of comparing different architectures in the scientific community. More objectives than four objectives could be optimized at the same time but this would need an update on the code base in order to prepare the algorithm for the collection of the custom metric.

Accuracy

While considering metrics that express the quality of the results obtained with a model, there are a multitude of formulas to choose from. Before considering the main metrics used in ML, lets make some clarifications. For convenience, since we are dealing with multi-class classification, lets consider with *positive* the correct predictions belonging to a class and with *negative* the others. Moreover, while considering precision and recall, there are two possible approaches to adopt: the first one considers the metric with respect to a single class while the second computes and averages between the obtained partial metrics.

- **Accuracy**: it represents the ratio between the number of correctly predicted results and the total points evaluated. It is an overall measure of the quality of a model. The way to calculate it does not vary between binary and multi-class, moreover it has the advantage to be straightforward and easy to understand, with the cons to not understand how good is a prediction with respect to a single class.

- **Precision**: precision measures how the model is able to correctly identify instances belonging to a particular class. It is computed as:

$$Precision_{classA} = \frac{TruePositive_{classA}}{TruePositive_{classA} + FalsePositive_{classA}} \tag{10.1}$$

- **Recall**: recall measures instead how the model is able to correctly identify all instances belonging to a particular class. The formula is similar to the one seen before:

$$Recall_{classA} = \frac{TruePositive_{classA}}{TruePositive_{classA} + FalseNegative_{classA}} \tag{10.2}$$

For convenience, since Ax needs metrics easy to understand, accuracy has been chosen as the quality metric of the model.

Latency

Latency is not as straightforward as accuracy to compute since SpikeExplorer runs on conventional computers while the resulting models will be deployed on neuromorphic hardware, we can deal with this problem in two different ways depending on whatever we choose to address the structural differences between the machines or not.

In the first case, the most simple, it could be considering the resulting delta computed as the difference between the timestamp at the start of the inference and the one at the end. This is pretty much immediate to implement on a normal CPU. Instead, while working with GPU an ulterior passage of warm-up is needed in order to have precise metrics.

The second case is not trivial as the previous one. There exist two possible latency values: one refers to the presence of at least one input spike with a consecutive scan of all inputs done by the neuron, this is the high value, while the low value simply refers to the absence of spikes, omitting the scanning process. In this way, the latency value is strongly coupled to the hardware implementation of the network.

Alternatively, there is a third path i.e. proceeding in a similar manner to what is done with power consumption.

Area

The area is quite difficult to correctly estimate since SpikeExplorer does not include the search of different quantization values. In order to properly estimate the area, it has been done a proportion with respect to different neuron models (*if lif* and *syn*) and architecture (*FF* and *recurrent*). The resulting *equivalent Look Up Tables* (LUTs) were computed by synthesizing a set of reference SNN architectures for each model obtaining an average of 200 LUTs for *if*, 300 LUTs for *lif* and 500 for *syn*. In order to consider also the architecture choices in the calculus, the set of available neuron was increased to 6: *if, rif, lif, rlif, syn and rsyn* (where *r* indicates the

presence of feedback both inter-neuron and intra-neuron); treating the recurrent counterpart architecture as a whole new neuron model.

SpikeExplorer can compute the raw data of the area, usable in the conversation, as follows:

$$\begin{aligned} Area = & weight\_quantization * \sum_{x=0}^{n-1} num\_weights_x * num\_weights_{x+1} \\ & + neuron\_model\_quantization * num\_neurons \end{aligned} \tag{10.3}$$

where x iterates through the network layers, while *weight_quantization* and *neuron_model_quantization* are values given by the user using the input configuration.

Power Consumption

Also for the power consumption, it is not as easy as we would have liked it to be. Power generally varies according to the activity level of the neurons in clock-driven architectures [61] and even more in event-driven alternatives. As it was done previously with the area, in SpikeExplorer it is possible to pass consumption values determined empirically on FPGA for each state of the neuron. But which are these states? They can be divided mainly into three different phase in which the neuron can be at each time step, they are:

- Active state: a spike is received as input causing the increasing of the membrane potential, the latter reaches the threshold potential values firing a new spike; the membrane potential is reset.

- Semi-active state: a spike is received as input, again with a consecutive increase of the membrane potential but this time the threshold value is not reached, no spike is fired. The neuron is in the *integrate* phase, the membrane potential instead decays through time.

- Passive state: no spikes is received and, consecutively, no spikes is fired. The neuron is in *leaky* phase, also in this case, if present, the membrane potential instead decays through time.

In practice, SpikeExplorer, during the evaluation, finds positive or negative variations in the membrane potential in order to determine the state of the neuron at each time step.

10.3 Library of Neuron Models

SpikeExplorer supports both ready-to-use neurons provided by snnTorch and custom neurons. The latter need to follow the same structure as the previous neurons in order to guarantee the correct functioning of the tool. The NG automatically recognises if the neuron model passed is stock or custom. Between the stock options, it is possible to choose: *Integrate-and-Fire*, *Leaky Integrate-and-Fire*, *Recurrent Leaky Integrate-and-Fire*, *Synaptic*, *Recurrent Synaptic*, *Lapicque*, *Alpha*, *Spiking LSTM* and *Spiking Convolutional LSTM*.

Chapter 11

Datasets

During the experimentation, it was decided to use three different dataset in order to evaluate Spike Explorer. The choice fell on:

- MNIST: a standard in every classification work, probably the best known dataset. Both the normal dataset and the one encoded into spikes using rate-coding counterpart were taken into account [62, 63].
- SHD: the Spiking Heidelberg Dataset, born with the intent of proposing an efficient way for comparing the computational performance of spiking neural networks, both on software applications and neuromorphic hardware implementations of snn [64].
- DVS: the Dynamic Vision Sensor dataset was created through the usage of a particular camera, the dataset is biologically inspired, in fact DVS transmits data only when a pixel detects a change, unlike traditional frame-based cameras which sample every pixel at a fixed frame rate [65]

11.1 MNIST

11.1.1 MNIST's Origin

The Modified National Institute of Standards and Technology (MNIST) database, was born as an improvement of the NIST since, as we can learn from [66], the last one is divided into two subset SD-3 and SD-1 (SD stands for Special Database) the first one for training purpose and the second one for the test part, but there was a major problem, SD-3 had too precise and clear handwritten digits compared to SD-1. The reasoning behind this difference dwells into the different place where the digits were collected (the first one among the Census Bureau Employees and the second one from high-school students). Since it is crucial that the results need

to be independent of the choice between training and test set, it follows the need to mix the NIST.

11.1.2 MNIST Structure

The dataset is composed by 70,000 examples, 60,000 for the training part and 10,000 for the test part. It is a subset of a much greater database the *NIST Special Database 19* composed by over 800,000 images with hand checked classifications [67]. Both training and test set are drawn in equal proportion (50/50) from the SD-3 and SD-1. The 60,000 samples from the training set contain digits from approximately 250 writers, paying attention to maintain disjoint the sets of writers between training and test set, in order to preserve the independence.

11.1.3 File format for the MNIST database

The files in the dataset are not in any standard image format, in fact they use the *IDX* file format purposely created to store vectors and multidimensional matrices of various numerical types. It is a simple format, for example for the training part it is constructed as follow for the labels:

```
[offset]    [type]           [value]            [description]
0000        32-bit integer   0x00000801 (2049)  magic number
0004        32-bit integer   60000              number of items
0008        unsigned byte    ??                 label
0009        unsigned byte    ??                 label
........
```

with labels values that vary between 0 to 9. While the image file is constructed as:

```
[offset]    [type]           [value]            [description]
0000        32-bit integer   0x00000803 (2051)  magic number
0004        32-bit integer   60000              number of images
0008        32-bit integer   28                 number of rows
0012        32-bit integer   28                 number of columns
0016        unsigned byte    ??                 pixel
0017        unsigned byte    ??                 pixel
........
```

pixels are organized row-wise in an image of 28x28, assuming values between 0 to 255 (0 the background and 255 for the foreground).

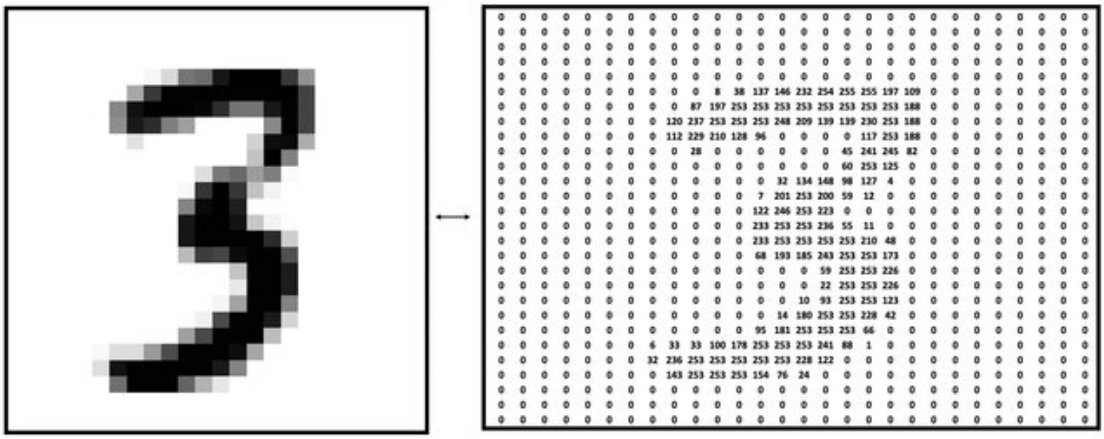


Figure 11.1: Representation of value three in the MNIST dataset and its equivalent matrix [68].

11.1.4 Encoding MNIST

Again, from the previous chapters we know that the *global currency* of the brain are *spikes* [5]. Every information coming from outside (i.e. images, smells or sounds) is encoded in brain-readable data. While talking about SNNs the same can be done with input and output data but even if it is not mandatory while taking into account the input dataset it is strongly recommended in order to exploit how SNNs extract additional meanings from temporal data. Three different encoding techniques can be used to do so: rate encoding, latency encoding, Delta modulation.

Rate Encoding

From the excellent Nobel prize-winning research done by Hubel and Wiesel, we know that for brighter input or favourable orientation of light impacting our photoreceptor cells corresponds to a higher firing rate. Equally speaking an image can be rate encoded according to the value assumed point-wise by pixels:

- Bright pixels will be encoded into a high-frequency firing rate:
- Dark pixels will be encoded into low-frequency firing rate.

The tricky part resides into the evaluation of the firing rate of a neuron, one can count the action potentials generated by the application of an input stimulus, this could be the simpler method except for the fact that the dynamics of a neuron vary over time. Alternatively, for a short temporal window Δt the spikes count can be restricted to [0,1] limiting the total number of possible outcomes to 2, spike

present and spike absent. In this way by diving the trial average by the duration of the interval one can come up with the time-dependent firing rate.

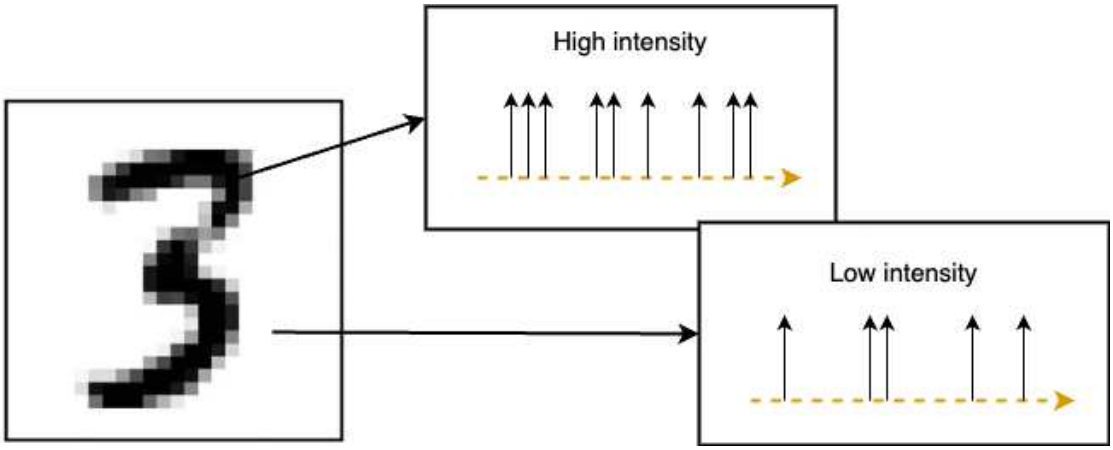


Figure 11.2: Here it can be seen the previous image from the MNIST and how it is associated an high intensity firing rate to high valued pixel and a low intensity firing rate to low valued pixel.

In order to transform the MNIST dataset into a *rate-coded MNIST*, it is possible to use the *spikegen* module that came within *snnTorch*:

```
from snntorch import spikegen

def rate_encode_mnist(data, dataset_name, time_steps):
    if dataset_name == "mnist":
        input_spikes = spikegen.rate(data, num_steps=time_steps)
```

Latency Encoding

If it is needed to put more emphasis on the timing of a spike, it can be used the latency or *temporal* encoding. Spikes ceases to be consequential, now what matters is when the spike arrives. Keeping the same analogy that we have done in the previous encoding technique, a bright pixel is coded as an early spike while the dark pixel corresponds to a late one, or even not spike at all. The single spike, in this situation, carries much more information than the spikes train previously seen.

In order to transform the MNIST dataset into a latency-coded MNIST, it is possible to use the spikegen module that came within snnTorch:

```
from snntorch import spikegen

def rate_encode_mnist(data, dataset_name, time_steps):
    if dataset_name == "mnist":
        input_spikes = spikegen.latency(data, num_steps=time_steps)
```

Delta Modulation

As it is suggested by the name, delta modulation take into account that the neurons thrive on change. This type of encoding could be implemented in various way but a common approach is to consider a positive difference greater than a given threshold, from this criteria comes a second name for this technique: threshold crossing.

In order to delta modulate the MNIST dataset, it is possible to use the spikegen module that came within snnTorch:

```
from snntorch import spikegen

# no change in size, only in the elements of the array
def rate_encode_mnist(data, dataset_name, time_steps):
    if dataset_name == "mnist":
        input_spikes = spikegen.delta(data, num_steps=time_steps)
```

11.2 SHD

Previous encoding techniques are effective but sub-optimal, in order to fully exploit the power of spiking neural networks it is convenient to use dataset *natively* formed by spikes. The first neuromorphic dataset that will be used in this thesis work is the Spiking Heidelberg Dataset. It is a relatively new dataset born in 2022 after the spreading of different spiking neural networks' training algorithms for both conventional and neuromorphic hardware, the need to clarify a common method to evaluate various architecture is more necessary than ever.

11.2.1 Dataset's structure

The Heidelberg Digits (HD) dataset consists in approximately 10,000 high-quality samples of spoken digits from 0 to 9 both in English and German language [64], a total of 12 speaker both female and male alternate in the samples having different age, each recorded 40 digits sequences in either English and German. The total

number of the so obtained digits is 10,420. in 11.3 it can be seen some highlights regarding the dataset distribution.

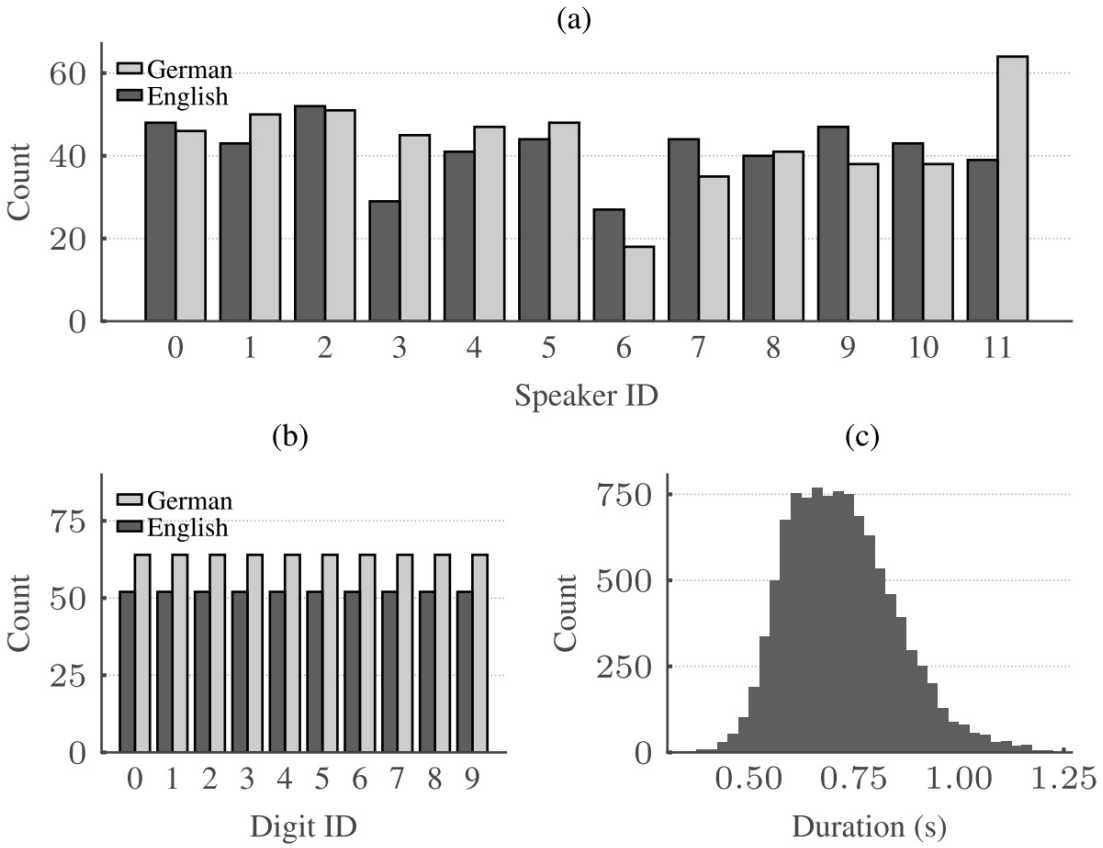


Figure 11.3: "The Heidelberg Digits HD have a balanced class count and variable temporal duration. The HD consist of 10420 recordings of spoken digits ranging from zero to nine in English and German language. (a) Histogram of per-speaker digit counts. Variable numbers of digits are available for each speaker and each language. (b) Histogram of per-class digit counts. The data set is balanced in terms of digits within each language. (c) Histogram of audio recording duration. The HD audio recordings were cut for minimal duration to keep computation time at bay." [64]

11.2.2 Sampling procedure

The samples were recorded in an isolated room in the Heidelberg University Hospital using two microphones Audio Technica Pro 37 and a Beyer Dynamic M201 TG, they were recorded at 48 kHz and 24-bit precision, successively a manual selection

was done, then the WAVE format was transformed into FLAC (Free Loseless Audio Codec) to preserve the audio quality. In order to explore the generalization capabilities of a network, the dataset was accurately divided into training and test part by recording the second one with two different microphone to avoid the over-fitting of the models to the mechanical characteristics of the microphones themselves.

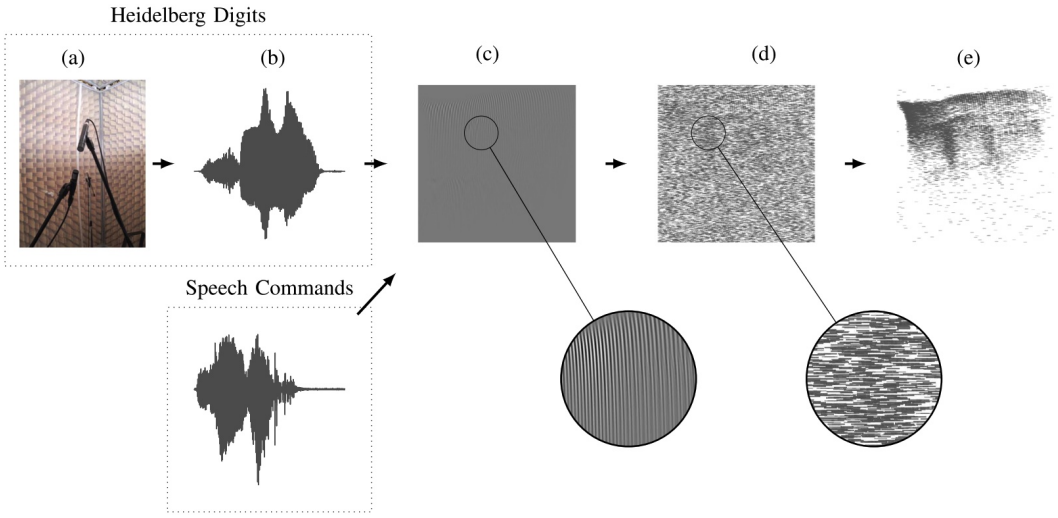


Figure 11.4: "Processing pipeline for the HD and the SC data set. (a) HD are recorded in a sound-shielded room. (b) Afterward, the resulting audio files are cut and mastered. (c) HD as well as the SC are fed through a hydrodynamic BM model. (d) BM decompositions are converted to phase-coded spikes by use of a transmitter-pool based HC model. (e) Phase-locking is increased by combining multiple spike trains of hair cells at the same position of the basilar membrane in a single bushy cell." [64]

11.2.3 Spike conversion

The audio samples were then converted into spikes through the usage of LAUSCHER (Flexible Auditory Spike Conversion Chain), an artificial model of the inner ear and parts of the ascending auditory pathway. This model emulates what seems to happen in the biological auditory system.

11.3 DVS-Gesture

DVS (Dynamic Vision Sensor) dataset born as a spiking alternative to the much more common frame-image datasets. Through the exploitation of event-based camera it is possible to rely on a more efficient way of elaborate and analyse real-time video input. The DVS hardware is biologically inspired in a sense that it is capable of recognising only when pixels change with the consecutively emission of spikes. The regular frame-based cameras are less efficient since they have to save frame-by-frame each pixels, but there is a problem with the *event-driven images*: with standard synchronous processor it could not be possible to benefit from the input coming from this sensors. Luckily, in the last years become much more frequent obtain new and powerful neuromorphic hardware capable of working with highly-parallelised spiking neurons.

The reasoning behind the development of this event-driven capture process resides again in the perfection of human brain, in fact this is a biologically inspired paradigm like the one that everyday we put into action with our eyes, even if biological synapses are much more slower that silicon transistor [65], they are capable of solving complex visions problems faster and in a much more efficient way, leveraging on a parallel, distributed and event-based computation.

11.3.1 EVS Technology

The Event-based Vision Sensors technology was created to emulate the functioning of human eyes and how they react to the light, for this reason they are often called neuromorphic sensors. The power of this technology resides into the fact that high-speed data output can be elaborated with extremely low latency through the limitation of incoming data to luminance variance and the combination of information about coordinate and time [69].

In 11.5 we can see how an event-generated wave of a pixel is captured by the camera and successively elaborated into spikes. The function represents how light intensity varies through time, when captured it is time sliced and for each positive variation corresponds a positive spike, equally speaking for the negative counterpart. The spike is emitted when the delta between two different time interval reaches a certain pre-defined threshold.

11.3.2 DVS-Gesture Structure

The DVS128 hand gesture dataset includes events from both the camera in 11.6 and a webcam that captures frame-based images. In order to create this dataset, they were collected 122 trials from 29 different subjects for a total of approximately 1,300 samples. A output we have 11 possible classes of gestures, both with hand

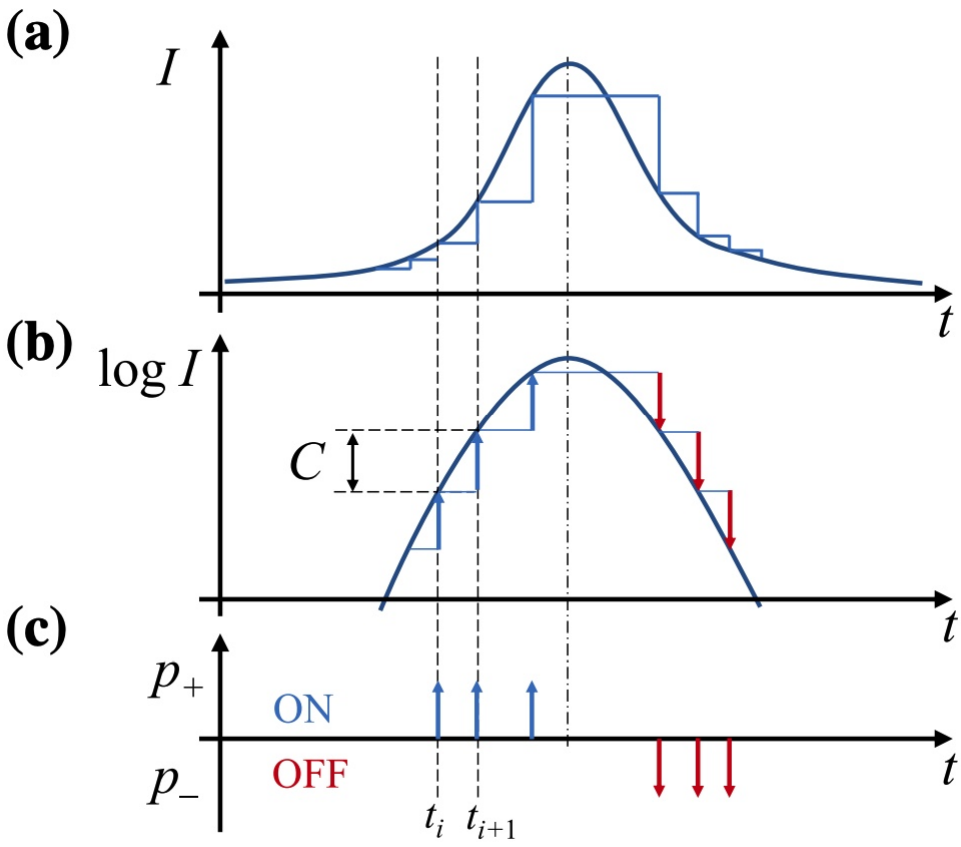


Figure 11.5: "Illustration of the event generation process for a single pixel of the event sensor: a time-varying intensity signal seen by the photodiode of the pixel, b log intensity signal processed by the contrast detection unit of the pixel, c asynchronous event stream produced by the pixel after reaching a certain contrast threshold C." [70]

and arm. The samples were collected under three different lightning condition: LED light, natural light and fluorescent light. A trial was composed by a methodic pattern: the subject, stood against a stationary background, had to perform all the 11 gestures under the same lightning condition. The sample duration is around six seconds. To the training set were allocated a total of 23 subjects while in the test set were placed the remaining 6 subjects.

Figure 11.6: In the figure can be seen the hardware setup used to record the DVS dataset. There is a iniLabs DVS128 camera (128x128 pixel Dynamic Vision Sensor) capable to generate events when a pixel value changes magnitude by a user-tunable threshold. In addition to the spatial coordinates, the event encodes the timestamp indicating when the spike was registered. [65]

Chapter 12

Results

12.1 Experimental Set-up

After the creation of the tool, there is the need to test it. As it was seen previously, there were targeted three different dataset, summarizing we have:

- **MNIST**: a hand-written digit images dataset on a grey-scale with a total of 784 inputs (28x28), 10 total output classes.
- **DVS128**: video recordings of hand and arm gestures done through the usage of neuromorphic sensors. A total inputs size of 16384 (128x128) and 11 output classes.
- **SHD**: audio recordings of number in English and German, the audio was then converted into spikes using 700 channels. 20 total output classes.

As mentioned before, SpikeExplorer has a wide range of configurable parameters, we have chosen a sub-set of the most significant parameters, they can be seen in table 12.1. Some clarifications need to be made: it has been decided to set the synaptic current and membrane potential decay rates correspondingly to $\alpha = 0.9$ and $\beta = 0.82$, obtained through some warm-up executions; furthermore, to limit the search time, the number of trials done in the specific runs of the SHD was reduced to 15 in order to avoid starvation. That being said, theoretically the research supports up to 20 parameters [71], limit dictated by the nature of Bayesian Optimization.

After some warm-up runs to check the presence of bugs in the tool, it has been decided to start a complete run of 25 trials on each dataset, with the right constraint on the objectives, the convergence is ensured after about 16 trials, after that Ax will fine tune the parameters in order to get the best results. Once completed, we decided to go further by re-run SpikeExplorer two more time for another 25

Table 12.1: Experimental set-up

	MNIST		**SHD**		**DVS128**	
	min	max	min	max	min	max
Learning rate	10^{-4}	$1.2 \cdot 10^{-4}$	10^{-4}	$1.2 \cdot 10^{-4}$	10^{-4}	$1.2 \cdot 10^{-4}$
Adam β_{optim}	0.9	0.999	0.9	0.999	0.9	0.999
# layers	1	3	1	3	1	3
Model	lif, syn, rlif, rsyn		lif, syn, rlif, rsyn		lif, syn, rlif, rsyn	
Reset	subtractive		subtractive		subtractive	
Time steps	10, 25, 50		10, 25, 50		10, 25, 50	
# neurons/layers	200, 100, 50		200, 100, 50		200, 100, 50	
Search iterations	25		15, 25		25	
Training epochs	50		100		50	

trials for MNIST and DVS and 15 for SHD: the first experiment is focused on the exploitation of the best performing neuron model while the latter redirects the attention on how the consumption and performances vary as the number of layers vary while keeping the same number of neurons in the architecture.

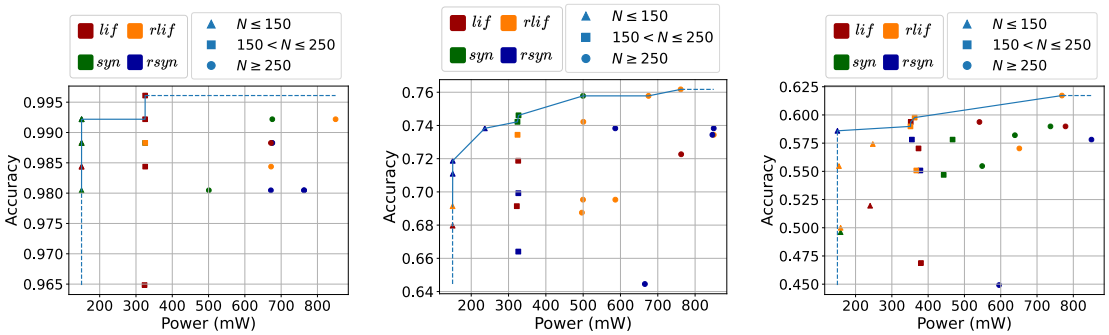
Finally, once satisfied by the exploration, one of the so obtained architecture is synthesized in order to do some comparisons with the SotA FPGA accelerators for SNNs.

12.2 Complete Exploration

All the explorations were done on a workstation with an AMD Ryzen 9 7950X 16-Core Processor, a GPU Nvidia RTX A4000 and 64 GB of ram memory. On average, without any leakage, the explorations for MNIST and DVS took around 5 hours, while 16 hours for SHD. Remember that the large amount of time is took by the training phase, Ax need to perform up to 25 training in that amount of time, taking from 13 to 40 minutes per training. This without saying that standard technologies are not optimized to compute recurrent passage like neuromorphic devices, causing an increment in time during training while evaluating the recurrent neuron models.

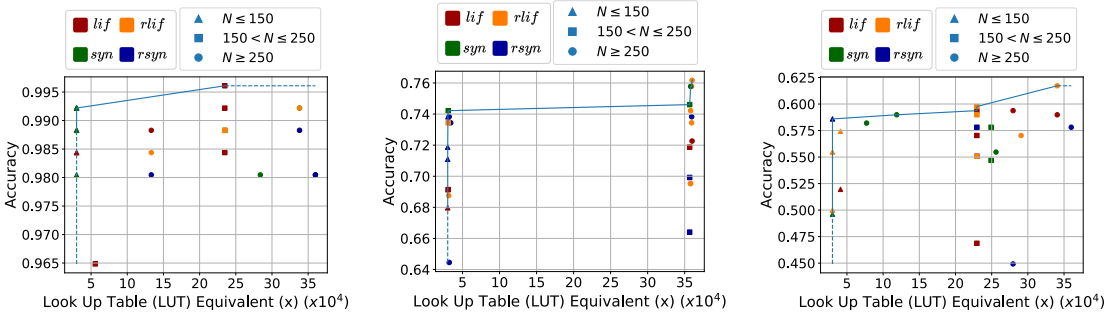
After these preliminary thoughts, it has to be addressed the correlation between two of the metrics while used at the same time: due to their nature, area and consumption are strictly correlated. The results are represented in figure 12.1 as the Pareto Frontier between accuracy and power consumption, and in 12.2.

As we can see in 12.5, from the complete run it was found that the best performing neuron model was LIF for the MNIST dataset, reaching an accuracy of 99.61%, considerably lower are instead the accuracies of DVS and SHD, respectively



(a) MNIST acc-power **(b)** DVS acc-power **(c)** SHD acc-power

Figure 12.1: Pareto frontiers between accuracy and power for the complete exploration



(a) MNIST accuracy-area **(b)** DVS accuracy-area **(c)** SHD accuracy-area

Figure 12.2: Pareto frontiers between accuracy and area for the complete exploration

76.17% and 61.70%. But this, instead, is not something to worry about, in fact our scope is not merely to reach the best accuracy but create a tool that is capable of finding the best set of objectives for *a specific scenario*. What is surely more interesting is the similarities between the results of SHD and DVS: both scored best while using RLIF, so it could be assumed to be necessary some kind of recursion to collect more information from time steps and it is precisely on the latter that we can see how SpikeExplorer has chosen more time steps than before, in fact, as we will see later on this chapter, usually more time steps are required in order to achieve better performances at the price of much higher power consumption. Moreover, SpikeExplorer seems to prioritize architectures with a larger number of neurons for these two dataset (over 450 neurons).

One another interesting thing is the formation of recurrent pattern along the figures 12.2b, 12.2c in which it seems that two cluster are formed among the trials,

Type	Layers	Time Steps	Acc.	Power (mW)
LIF	200-10	10	99.61%	310
RLIF	200-100-200-10	10	99.22%	860
SYN	200-200-10	25	99.22%	680
RSYN	100-10	25	99.22%	140

Table 12.2: Best architectures for each neuron type on MNIST

Type	Layers	Time Steps	Acc.	Power (mW)
LIF	200-200-50-11	50	72.27%	500
RLIF	200-200-50-11	25	76.17%	760
SYN	200-100-11	50	75.78%	500
RSYN	100-200-50-11	50	73.83%	590

Table 12.3: Best architectures for each neuron type on DVS

Type	Layers	Time Steps	Acc.	Power (mW)
LIF	200-20	50	59.41%	360
RLIF	200-200-20	50	61.70%	760
SYN	100=100-200-20	10	58.98%	720
RSYN	100-20	50	58.59%	140

Table 12.4: Best architectures for each neuron type on SHD

Table 12.5: Complete table

one with a greater focus on bigger architecture and the other on smaller ones. Highly focused researches on these clusters are needed in order to individuate the reasoning behind this behavior.

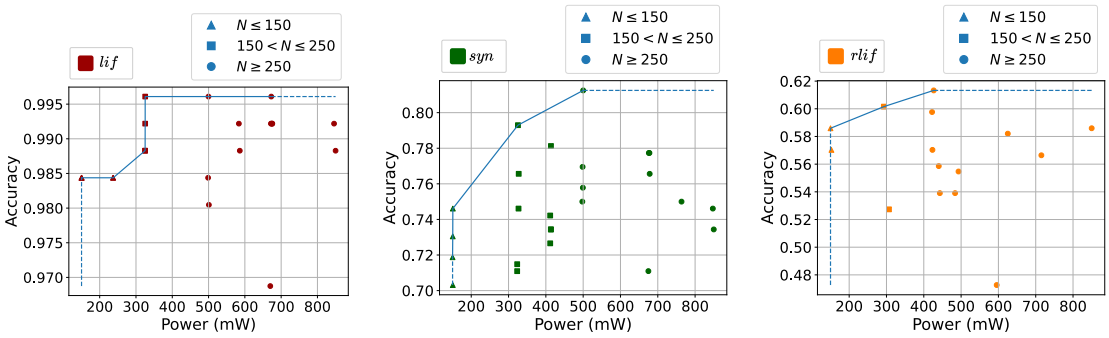
Last but not least, it is correct to point out how *quantized* are the accuracies while talking about the MNIST dataset, the most credible hypothesis lies in the fact that, unlike other dataset, MNIST underwent a procedure of *rate encoding* in order to exploit the intrinsic nature of SNNs. More research on this need to be made.

12.3 Specific Explorations

In the first batch of experiments, it was decided to let SpikeExplorer *explore* the design space without any prior knowledge on which are the best constraints to use while doing this kind of research, full freedom was given to the tool. Now, lets fix some variables in order to evaluate the *exploitation* capabilities of SpikeExplorer.

12.3.1 Fixed neurons model's type

The first experiment in this category of explorations, is trying to exploit as much as possible the neuron model that for each dataset performed the best previously, some characteristic results were found. Also here, as we can expect highest accuracy are reached for most of the part by bigger architecture but the Pareto Frontiers in 12.3 are mostly made up of smaller architecture with constrained power, in fact is interesting to see how the tool achieved the retention of the accuracy while lowering the power consumption by choosing a smaller architecture. What catches the eyes, is for sure the ability of SpikeExplorer to increment the maximum achievable accuracy from 76.17% up to 81.6% for the DVS Gesture.



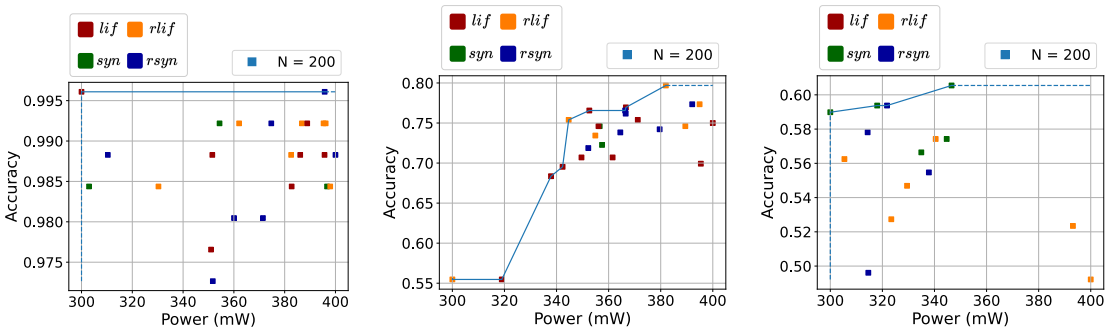
(a) MNIST acc-power **(b)** DVS acc-power **(c)** SHD acc-power

Figure 12.3: Pareto frontiers between accuracy and power for the partial exploration, neuron type kept fixed

12.3.2 Fixed number of neurons

The latter experiment of this batch consisted in setting the number of hidden neurons to 200 allowing SpikeExplorer to choose the better hidden layers configuration and neuron pair, choosing them respectively in $[(200), (100, 100), (100, 50, 50)]$ and $[\text{"}lif\text{"}, \text{"}rlif\text{"}, \text{"}syn\text{"}, \text{"}rsyn\text{"}]$. Here again, it can be witnessed the increment in accuracy for the DVS dataset, even if it is one percentage point lower. It is

interesting how different dataset seems to prefers a certain neuron model rather than other: for MNIST, the pareto frontier is dominated by the *lif* while for the SHD, the preference goes for *syn*. Instead the DVS presents a the possibility of choice the best trade-off between accuracy and power depending on the use cases.



(a) MNIST acc-power **(b)** DVS acc-power **(c)** SHD acc-power

Figure 12.4: Pareto frontiers between accuracy and power for the partial exploration, neurons number kept fixed

12.4 *State-of-the-Art* SNN architectures

Since a comparison between different DSE tools applied to SNN would be impossible due to its uniqueness, it was decided to compare the resulting SNN architectures to the SotA SNNs. After a complete exploration on MNIST, keeping constant the number of neuron in the hidden layer (fixed to 128, single layered), the resulting architecture was synthesized using [61]. In table 12.6 are reported the state of the art FPGA accelerators for SNN. Starting from a similar situation to the one seen in [61], SpikeExplorer was able to improve the overall latency, divided by more than 6 times, passing from $780\mu s$ to $120\mu s$, and the accuracy, which saw an increase of 1.95%. The architecture found was only 1.26% far from the SotA architecture [72] in terms of accuracy but reducing the amount of power needed from $0.477W$ to $0.18W$, almost a third of the SotA value. What we saw turns out to be the concrete proof we were looking for to affirm what it is capable of SpikeExplorer.

Table 12.6: Comparison of SpikeExplorer to state-of-the-art FPGA accelerators for SNN

Design	Han et al.[72]	Gupta et al. [73]	Li et al.[74]	SPIKER[75]	SPIKER+ [61]	This work
Year	2020	2020	2021	2022	2024	
f_{clk}**[MHz]**	200	100	100	100		
Neuron bw	16	24	16	16	6	
Weights bw	16	24	16	16	4	
Update	Event	Event	Hybrid	Clock		
Model	LIF	LIF[76]	LIF	LIF		
FPGA	XC7Z045	XC6VLX240T	XC7VX485	XC7Z020		
Avail. BRAM	545	416	2,060	140		
Used BRAM	40.5	162	N/R	45	18	
Avail. DSP	900	768	2,800	220		
Used DSP	0	64	N/R	0		
Avail. logic cells	655,800	452,160	485,760	159,600		
Used logic cells	12,690	79,468	N/R	55, 998	7,612	
Arch	1024-1024-10	784-16	200-100-10	400	128-10	
#syn	1,861,632	12,544	177,800	313,600	101,632	
T_{lat}**/img [ms]**	6.21	0.50	3.15	0.22	0.78	0.12
Power [W]	0.477	N/R	1.6	59.09	0.18	
E/img $[mJ]$	2.96	N/R	5.04	13	0.14	0.02
E/syn $[nJ]$	1.59	N/R	28	41	1.37	0.22
Accuracy	97.06%	N/R	92.93%	73.96%	93.85%	95.8%

Chapter 13

Conclusion

13.1 Some Considerations

This master's thesis work was aimed towards the presentation of SpikeExplorer, a tool capable of automatizing the design space exploration of spiking neural networks intended to be used on Field Programmable Gate Array. As we have seen in the previous chapter, the framework achieved remarkable results fulfilling the predetermined objectives.

Even if it was not supposed to achieve state-of-the-art accuracies for the tasks used as evaluation metrics, i.e. MNIST, DVS and SHD, SpikeExplorer managed to score exceptional performances on MNIST, 95.8% while keeping low latency, about μs and minimal power consumption (180mW). Even on the other tasks it demonstrated what it is capable of and we strongly believe that with some more research and experiments about them we could reach SotA accuracies also for them.

13.2 Future improvements

SpikeExplorer was thought as modular framework that can either exchange the parameters to research or even the DSE engine. In fact, it was foreseen, as a future development, the support of Convolutional Spiking Neural Network capable of filling the gap left by FF-FC in some task (just think about the input size of DVS and SHD, FF networks have not enough abstraction capabilities).

Moreover, we have planned to simplify the set-up phase of SpikeExplorer in order to ease the access to the tool, enabling an high level of abstraction while using it.

Furthermore, it is thinkable to add into the train-evaluate function also a simulation phase in which the architecture could be synthesized evaluating with more accuracy metrics such as power consumption or memory occupation.

Finally we can conclude by saying that SpikeExplorer, through its results, lays the foundation to an automatic approach to design space exploration of SNNs hoping to ease the adoption of neuromorphic, facilitating the optimization phase for resource-constrained edge applications.

Bibliography

[1] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002 (cit. on p. 1).

[2] Wikipedia contributors. *Biological neuron model — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-February-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Biological_neuron_model&oldid=1209231590 (cit. on pp. 2, 3, 28).

[3] 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: https://doi.org/10.1113/jphysiol.1952.sp004764. URL: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764 (cit. on pp. 2, 3, 26).

[4] Wikipedia contributors. *Spike-timing-dependent plasticity — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-February-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Spike-timing-dependent_plasticity&oldid=1200755346 (cit. on p. 2).

[5] Jason K. Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2023. arXiv: 2109.12894 [cs.NE] (cit. on pp. 3, 33–35, 64).

[6] S Finger. *SOrigins of neuroscience: A history of explorations into brain function*. Oxford University Press, 1994 (cit. on p. 3).

[7] Heba Ibrahim, A. Abdo, Ahmed M. El Kerdawy, and A. Sharaf Eldin. «Signal Detection in Pharmacovigilance: A Review of Informatics-driven Approaches for the Discovery of Drug-Drug Interaction Signals in Different Data Sources». In: *Artificial Intelligence in the Life Sciences* 1 (2021), p. 100005. ISSN: 2667-3185. DOI: https://doi.org/10.1016/j.ailsci.2021.100005. URL: https://www.sciencedirect.com/science/article/pii/S2667318521000052 (cit. on p. 3).

[8] Andrea Palazzi. *Deep Neural Network Lectures*. [Online; accessed 25-March-2024]. 2024. URL: https://github.com/ndrplz/machine_learning_lectures/blob/master/slides/deep_learning/deep_neural_networks/deep_neural_networks.pdf (cit. on p. 3).

[9] Chuan Li. *OpenAI's GPT-3 Language Model: A Technical Overview*. [Online; accessed 25-March-2024]. 2024. URL: https://lambdalabs.com/blog/demystifying-gpt-3 (cit. on pp. 5, 6).

[10] Wikipedia contributors. *History of artificial intelligence — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=History_of_artificial_intelligence&oldid=1215362966 (cit. on pp. 7, 11).

[11] Jef Akst. *Machine Learning 1951*. [Online; accessed 28-March-2024]. 2019. URL: https://www.the-scientist.com/machine--learning--1951-65792 (cit. on pp. 9, 10).

[12] GRACE SOLOMONOFF. *The Meeting of the Minds That Launched AI*. [Online; accessed 25-March-2024]. 2023. URL: https://spectrum.ieee.org/dartmouth-ai-workshop (cit. on p. 10).

[13] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books, 1962. URL: https://books.google.it/books?id=7FhRAAAAMAAJ (cit. on pp. 11, 16).

[14] Rockwell Anyoha. *The History of Artificial Intelligence*. [Online; accessed 25-March-2024]. 2017. URL: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/ (cit. on p. 11).

[15] Wikipedia contributors. *Activation function — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=1213222372 (cit. on p. 13).

[16] Varun Kumar Ojha, Ajith Abraham, and Václav Snásel. «Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research». In: *CoRR* abs/1705.05584 (2017). URL: http://arxiv.org/abs/1705.05584 (cit. on p. 14).

[17] Mathew Mithra Noel, Arindam Banerjee, Geraldine Bessie Amali D, and Venkataraman Muthiah-Nakarajan. *Alternate Loss Functions for Classification and Robust Regression Can Improve the Accuracy of Artificial Neural Networks*. 2023. arXiv: 2303.09935 [cs.NE] (cit. on p. 15).

[18] Wikipedia contributors. *Cross-entropy — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Cross-entropy&oldid=1208849293 (cit. on p. 16).

[19] Anqi Mao, Mehryar Mohri, and Yutao Zhong. *Cross-Entropy Loss Functions: Theoretical Analysis and Applications*. 2023. arXiv: 2304.07288 [cs.LG] (cit. on p. 16).

[20] Alexander Amini, Ava Soleimany, Sertac Karaman, and Daniela Rus. «Spatial Uncertainty Sampling for End-to-End Control». In: *CoRR* abs/1805.04829 (2018). URL: http://arxiv.org/abs/1805.04829 (cit. on p. 17).

[21] Juergen Schmidhuber. *Annotated History of Modern AI and Deep Learning*. 2022. arXiv: 2212.11279 [cs.NE] (cit. on p. 16).

[22] Seppo Linnainmaa. *Algoritmin kumulatiivinen pyöristysvirhe yksittäisten pyöristysvirheiden*. [Online; accessed 25-March-2024]. 1970. URL: URN:NBN:fi:hulib-202006173019;http://hdl.handle.net/10138/316565 (cit. on p. 17).

[23] A. Torcini. *Modelli semplici di neuroni*. [Online; accessed 26-March-2024]. 2007. URL: https://perso.u-cergy.fr/~atorcini/neuro07.pdf (cit. on p. 19).

[24] Bradley Voytek. *The measure of a whale*. [Online; accessed 26-March-2024]. 2013. URL: https://www.nature.com/scitable/blog/brain-metrics/the_measure_of_a_whale/ (cit. on p. 19).

[25] Encyclopedia Britannica. *neuronal membrane*. [Online; accessed 26-March-2024]. URL: https://www.britannica.com/science/nervous-system/The-neuronal-membrane (cit. on p. 21).

[26] Encyclopedia Britannica. *action potential*. [Online; accessed 26-March-2024]. 2024. URL: https://www.britannica.com/science/action-potential (cit. on pp. 22, 23).

[27] Feldman D. E. «The spike-timing dependence of plasticity». In: *Neuron* 75.4 (2012), pp. 556–571. DOI: 10.1016/j.neuron.2012.08.001 (cit. on p. 23).

[28] Wolfgang Maass. «Networks of spiking neurons: The third generation of neural network models». In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7 (cit. on p. 25).

[29] L.F Abbott. «Lapicque's introduction of the integrate-and-fire model neuron (1907)». In: *Brain Research Bulletin* 50.5 (1999), pp. 303–304. ISSN: 0361-9230. DOI: https://doi.org/10.1016/S0361-9230(99)00161-6 (cit. on pp. 25, 26, 29).

[30] Wikipedia contributors. *Hodgkin-Huxley model — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-February-2024]. 2023. URL: https://en.wikipedia.org/w/indexFNagumo.php?title=Hodgkin%E2%80%93Huxley_model&oldid=1179847158 (cit. on pp. 26–28).

[31] Wikipedia contributors. *Hodgkin-Huxley circuit*. [Online; accessed 25-March-2024]. 2024. URL: https://commons.wikimedia.org/wiki/File:Hodgkin-Huxley.svg (cit. on p. 27).

[32] D Mishra, A Yadav, S Ray, and PK Kalra. «Exploring biological neuron models». In: *Directions, The Research Magazine of IIT Kanpur* 7.3 (2006), pp. 13–22 (cit. on p. 28).

[33] Wikipedia contributors. *FitzHugh-Nagumo model — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-February-2024]. 2023. URL: https://en.wikipedia.org/w/index.php?title=FitzHugh%E2%80%93Nagumo_model&oldid=1161091448 (cit. on p. 28).

[34] Catherine E. Morris and Harold Lecar. «Voltage oscillations in the barnacle giant muscle fiber.» In: *Biophysical journal* 35 1 (1981), pp. 193–213. URL: https://api.semanticscholar.org/CorpusID:25868925 (cit. on pp. 28, 29).

[35] Richard FitzHugh. «Impulses and Physiological States in Theoretical Models of Nerve Membrane». In: *Biophysical Journal* 1.6 (1961), pp. 445–466. ISSN: 0006-3495. DOI: https://doi.org/10.1016/S0006-3495(61)86902-6. URL: https://www.sciencedirect.com/science/article/pii/S0006349561869026 (cit. on p. 29).

[36] J. Nagumo, S. Arimoto, and S. Yoshizawa. «An Active Pulse Transmission Line Simulating Nerve Axon». In: *Proceedings of the IRE* 50.10 (1962), pp. 2061–2070. DOI: 10.1109/JRPROC.1962.288235 (cit. on p. 29).

[37] Wikipedia contributors. *Morris-Lecar model — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-February-2024]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Morris%E2%80%93Lecar_model&oldid=1125226824 (cit. on p. 29).

[38] Alain Destexhe, Zachary Mainen, and Terrence Sejnowski. «Methods in Neural Modelling: From Ions to Networks». In: Jan. 1998, pp. 1–25 (cit. on p. 30).

[39] Eugene M. Izhikevich. «Simple Model of Spiking Neurons». In: *IEEE Transactions on Neural Networks* 14 (2003), pp. 1569–1572. URL: https://www.izhikevich.org/publications/spikes.htm (cit. on p. 30).

[40] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. «Spiking Neural Networks and Their Applications: A Review». In: *Brain Sciences* 12.7 (2022). ISSN: 2076-3425. DOI: 10.3390/brainsci12070863. URL: https://www.mdpi.com/2076-3425/12/7/863 (cit. on p. 31).

[41] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. The MIT Press, July 2006. DOI: 10.7551/mitpress/2526.001.0001. URL: https://doi.org/10.7551/mitpress/2526.001.0001 (cit. on p. 31).

[42] Michael H. Chase and Francisco R. Morales. «Chapter 12 - Control of Motoneurons during Sleep». In: *Principles and Practice of Sleep Medicine (Fourth Edition)*. Ed. by Meir H. Kryger, Thomas Roth, and William C. Dement. Fourth Edition. Philadelphia: W.B. Saunders, 2005, pp. 154–168. ISBN: 978-0-7216-0797-9. DOI: https://doi.org/10.1016/B0-72-160797-7/50019-7. URL: https://www.sciencedirect.com/science/article/pii/B0721607977500197 (cit. on p. 31).

[43] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. «Supervised learning in spiking neural networks: A review of algorithms and evaluations». In: *Neural Networks* 125 (2020), pp. 258–280. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.02.011. URL: https://www.sciencedirect.com/science/article/pii/S0893608020300563 (cit. on pp. 32, 33).

[44] Warren S. McCulloch and Walter Pitts. «A Logical Calculus of the Ideas Immanent in Nervous Activity». In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/bf02478259 (cit. on p. 32).

[45] Romain Brette and Wulfram Gerstner. «Adaptive Exponential Integrate-And-Fire Model As An Effective Description Of Neuronal Activity». In: *Journal of neurophysiology* 94 (Dec. 2005), pp. 3637–42. DOI: 10.1152/jn.00686.2005 (cit. on p. 33).

[46] Wikipedia contributors. *Exploration-exploitation dilemma — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Exploration-exploitation_dilemma&oldid=1214467934 (cit. on p. 36).

[47] Andy D. Pimentel. «Methodologies for Design Space Exploration». In: *Handbook of Computer Architecture*. Ed. by Anupam Chattopadhyay. Singapore: Springer Nature Singapore, 2022, pp. 1–31. ISBN: 978-981-15-6401-7. DOI: 10.1007/978-981-15-6401-7_23-1. URL: https://doi.org/10.1007/978-981-15-6401-7_23-1 (cit. on pp. 37–39).

[48] Wikipedia contributors. *Pareto front — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-March-2024]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Pareto_front&oldid=1184190980 (cit. on p. 38).

[49] Wikipedia contributors. *Pareto efficiency — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Pareto_efficiency&oldid=1213179097 (cit. on p. 38).

[50] Andy D. Pimentel. «Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration». In: *IEEE Design & Test* 34.1 (2017), pp. 77–90. DOI: 10.1109/MDAT.2016.2626445 (cit. on p. 39).

[51] Matthias Gries. «Methods for evaluating and covering the design space during early design development». In: *Integration* 38.2 (2004), pp. 131–183. ISSN: 0167-9260. DOI: https://doi.org/10.1016/j.vlsi.2004.06.001. URL: https://www.sciencedirect.com/science/article/pii/S016792600400032X (cit. on p. 39).

[52] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. *Neural Architecture Search: A Survey*. 2019. arXiv: 1808.05377 [stat.ML] (cit. on p. 41).

[53] Meta Platforms, Inc. *Documentation for Ax Platform*. [Online; accessed 10-March-2024]. 2024. URL: https://ax.dev/docs/why-ax.html (cit. on pp. 45, 46, 52, 53).

[54] Wang B. Chen Z. and Gorban A.N. «Multivariate Gaussian and Student-t process regression for multi-output prediction». In: *Neural Comput & Applic* 32 (2020), pp. 3005–3028. DOI: 10.1007/s00521-019-04687-8. URL: https://doi.org/10.1007/s00521-019-04687-8 (cit. on p. 46).

[55] scikit-learn developers. *Kernels for Gaussian Processes*. [Online; accessed 17-March-2024]. 2024. URL: https://scikit-learn.org/stable/modules/gaussian_process.html (cit. on p. 46).

[56] scikit-learn developers. *Illustration of Kernels for Gaussian Processes*. [Online; accessed 17-March-2024]. 2024. URL: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_prior_posterior.html (cit. on pp. 48, 49).

[57] Roman Garnett. *Lectures on Bayesian Methods in Machine Learning*. [Online; accessed 17-March-2024]. 2015. URL: https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf (cit. on p. 49).

[58] Python Wiki. *Official Python Wiki Page*. [Online; accessed 10-March-2024]. 2024. URL: https://wiki.python.org/moin/FrontPage (cit. on p. 51).

[59] Google. *TensorFlow*. [Online; accessed 27-March-2024]. 2024. URL: https://www.tensorflow.org/ (cit. on p. 51).

[60] The Linux Foundation. *PyTorch*. [Online; accessed 27-March-2024]. 2024. URL: https://pytorch.org/ (cit. on p. 51).

[61] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. *Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge*. arXiv:2401.01141 [cs]. Jan. 2024. DOI: 10.48550/arXiv.2401.01141. URL: http://arxiv.org/abs/2401.01141 (cit. on pp. 60, 77, 78).

[62] Li Deng. «The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]». In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477 (cit. on p. 62).

[63] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791 (cit. on p. 62).

[64] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. «The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks». In: *IEEE Transactions on Neural Networks and Learning Systems* 33.7 (2022), pp. 2744–2757. DOI: 10.1109/TNNLS.2020.3044364 (cit. on pp. 62, 66–68).

[65] Arnon Amir et al. «A Low Power, Fully Event-Based Gesture Recognition System». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7388–7397. DOI: 10.1109/CVPR.2017.781 (cit. on pp. 62, 69, 71).

[66] LeCun Y., Cortes C., Burges C. *The MNIST database of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (cit. on p. 62).

[67] *NIST Special Database 19*. 1995. DOI: http://doi.org/10.18434/T4H01C. URL: https://www.nist.gov/srd/nist-special-database-19 (cit. on p. 63).

[68] Soha Boroojerdi and George Rudolph. «Handwritten Multi-Digit Recognition With Machine Learning». In: May 2022, pp. 1–6. DOI: 10.1109/IETC54973.2022.9796722 (cit. on p. 64).

[69] SONY. *Event-based Vision Sensor(EVS)Technology*. URL: https://www.sony-semicon.com/en/technology/industry/evs.html (cit. on p. 69).

[70] Klinner Willert E. Christian and Joachim. «Event-based imaging velocimetry: an assessment of event-based cameras for the measurement of fluid flows». In: *Experiments in Fluids* 63 (2022). DOI: 10.1007/s00348-022-03441-6. URL: https://doi.org/10.1007/s00348-022-03441-6 (cit. on p. 70).

[71] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. arXiv: 1807.02811 [stat.ML] (cit. on p. 72).

[72] Jianhui Han, Zhaolin Li, Weimin Zheng, and Youhui Zhang. «Hardware implementation of spiking neural networks on FPGA». In: *Tsinghua Science and Technology* 25.4 (Aug. 2020). Conference Name: Tsinghua Science and Technology, pp. 479–486. ISSN: 1007-0214. DOI: 10.26599/TST.2019.9010019. URL: https://ieeexplore.ieee.org/document/8954866 (visited on 12/06/2023) (cit. on pp. 77, 78).

[73] Shikhar Gupta, Arpan Vyas, and Gaurav Trivedi. «FPGA Implementation of Simplified Spiking Neural Network». In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. Nov. 2020, pp. 1–4. DOI: 10.1109/ICECS49266.2020.9294790. URL: https://ieeexplore.ieee.org/abstract/document/9294790 (cit. on p. 78).

[74] Sixu Li, Zhaomin Zhang, Ruixin Mao, Jianbiao Xiao, Liang Chang, and Jun Zhou. «A Fast and Energy-Efficient SNN Processor With Adaptive Clock/Event-Driven Computation Scheme and Online Learning». In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.4 (Apr. 2021). Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers, pp. 1543–1552. ISSN: 1558-0806. DOI: 10.1109/TCSI.2021.3052885 (cit. on p. 78).

[75] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. «Spiker: an FPGA-optimized Hardware accelerator for Spiking Neural Networks». In: *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. ISSN: 2159-3477. July 2022, pp. 14–19. DOI: 10.1109/ISVLSI54635.2022.00016 (cit. on p. 78).

[76] Taras Iakymchuk, Alfredo Rosado-Muñoz, Juan F. Guerrero-Martínez, Manuel Bataller-Mompeán, and Jose V. Francés-Víllora. «Simplified spiking neural network architecture and STDP learning algorithm applied to image classification». In: *EURASIP Journal on Image and Video Processing* 2015.1 (Feb. 2015), p. 4. ISSN: 1687-5281. DOI: 10.1186/s13640-015-0059-4. URL: https://doi.org/10.1186/s13640-015-0059-4 (visited on 12/12/2023) (cit. on p. 78).