Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Quantum Convolutional Neural Networks for High Energy Physics

Tesis presentada al

Colegio de Física

como requisito parcial para la obtención del grado de

Licenciado en Física

por

Luis Roberto Cervantes Guevara

Asesorado por

Dra María Isabel Pedraza Morales

Puebla Pue.

September 5, 2022

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Quantum Convolutional Neural Networks for High Energy Physics

Tesis presentada al

Colegio de Física

como requisito parcial para la obtención del grado de

Licenciado en Física

por

Luis Roberto Cervantes Guevara

Asesorado por

Dra María Isabel Pedraza Morales

Puebla Pue.

September 5, 2022

Título: Quantum Convolutional Neural Networks for High Energy Physics

Estudiante: Luis Roberto Cervantes Guevara

COMITÉ

Dr. Humberto Antonio Salazar Ibargüen

Presidente

María del Carmen Heras Sánchez

Secretario

Enrique Varela Carlos

Vocal

Dr. Jorge Velázquez Castro

Vocal

Dra María Isabel Pedraza Morales

Asesor

Acknowledgments

To all my professors throughout my academic life, especially those who taught me during my undergraduate studies. I believe that teaching is one of the noblest endeavors, and I value very much the patience with which all my professors have taught me. I would not be what I am today without their efforts and kind advice.

To my advisor, PhD. Isabel Pedraza, without whose support this work would not have been possible.

Contents

Chapter 1

Introduction

Artificial Intelligence is a tool that is increasingly becoming an integral element of scientific research. High Energy Physics, whose experiments produce some of the largest amounts of data in science, is no exception. For this reason, the objective of this thesis is to develop a pipeline for classifying backgrounds and signals particle jets using Machine Learning (ML) techniques, specifically convolutional neural networks (CNN).

Particle jets have proven to be a very powerful tool for studying particle collisions at accelerators such as CMS and ATLAS, at the LHC, where the constituents of these events hadronize or decay so quickly that they are very difficult to detect. Jets are objects that seek to retrieve information about these particles by encapsulating the energy depositions that were indeed sensed by the detector. So having an algorithm that can reliably tell us which particle generated a given jet is not a straightforward assignment, even more so taking into account that many approaches can be taken to this problem depending on the way in which we believe it is most convenient to arrange our data.

Given a set of jets represented by their respective constituents (or pseudo vectors) characterized by momentum quadrivectors, an image approach is proposed, where each jet is assigned a calorimeter image that can be analyzed by a convolutional neural network. That done, in order to establish a simple connection with quantum computing, a new model integrating a quantum neural layer is proposed. This is done via a parameterized quantum circuit, which is implemented thanks to the Qiskit framework for quantum computing in Python.

Chapters 2 and 3 develop some of the theory underlying artificial neural networks and quantum computing, respectively. Chapter 4 describes the geometry of particle detectors, the role of jets in proton-proton collisions, and serves as a preamble to introduce the dataset with which we work in Chapter 5. The penultimate chapter presents an end-to-end machine learning pipeline for classification of calorimeter jets images, in which we also introduce a hybrid classical-quantum ML algorithm that can solve the same problem with similar accuracy.

1.1 General Objective

To illustrate an implementation of quantum computing in data analysis in High Energy Physics, by developing a classical-quantum classical-quantum algorithm for the classification of jets produced in proton-proton collisions in particle particle accelerators.

1.2 Specific objectives

1. From simulated data of jets produced by proton-proton collisions, develop a set of calorimetric images where each image represents a jet.
2. To train a classical convolutional neural network in Python that learns to distinguish between a signal and a background in a set of calorimetric images.
3. Create a hybrid neural network, adding to a convolutional classical neural network architecture a layer that simulates a parameterized quantum circuit using the Qiskit (Quantum Information Software Kit for Quantum Computation) library from IBM.

Chapter 2

Artificial Neural Networks

2.1 Artificial intelligence

Artificial intelligence, machine learning and deep learning are three different concepts that are sometimes used interchangeably. Kaplan and Haenlein define artificial intelligence as *a system's ability to interpret external data correctly, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation* [1]. This is a very broad definition, since artificial intelligence is considered to be any autonomous mechanism capable of performing a set of tasks without having been given precise and explicit instructions to do so. Self-driving cars can be considered as examples of artificial intelligence since they are not programmed with all the infinite possibilities that can occur during a specific route, nor with the series of specific decisions (right turn, left turn, accelerate, brake) that they should make for each of these possibilities. They perform the decision-making process autonomously, just as a human being would.

2.2 Machine learning

Machine learning is a subset of artificial intelligence and it can be defined as a technique of analyze data, learn from that data and then apply what it have learned to make an informed decision [citation]. The way it does this is through trial-and-error, adjusting parameters at each iteration with the objective of improving performance (the accuracy of the outputs) based on experience.

To better understand this, it is worth mentioning how machine learning algorithms are classified: supervised, unsupervised and reinforcement learning.

2.2.1 Supervised learning

Supervised learning automates decision-making processes by generalizing known examples. In this configuration, the user provides the algorithm with pairs of desired inputs and outputs, and the algorithm finds a way to produce the desired output given a new input. In particular, the algorithm may create an output for an input that it has never seen before without the help of a human.

This implementation is used in the identification of objects given at least two classes of them. For example, distinguishing between pictures of dogs or cats, or determining whether an email is spam or not. A machine learning algorithm intended to perform this task must be provided, in addition to the training data set, with its respective labels that determine to which class each object belongs.

2.2.2 Unsupervised learning

In unsupervised learning the training data does not include labels and the algorithm tries to classify or decode the information on its own. This is done by finding patterns in the data that even humans cannot distinguish, at least in an efficient way.

An algorithm that exemplifies the goal of unsupervised learning very well is known as clustering, aimed at grouping sets of data that share certain characteristics. This algorithm is used to group Twitter users, for example, who exhibit similar behaviors in order to show them personalized advertisements.

Another very interesting implementation is through generative neural networks which, as their name suggests, have the task of producing an object (image, text, sound) that is similar to the data with which it was trained. This type of neural networks are mainly used in the generation of synthetic videos or images.

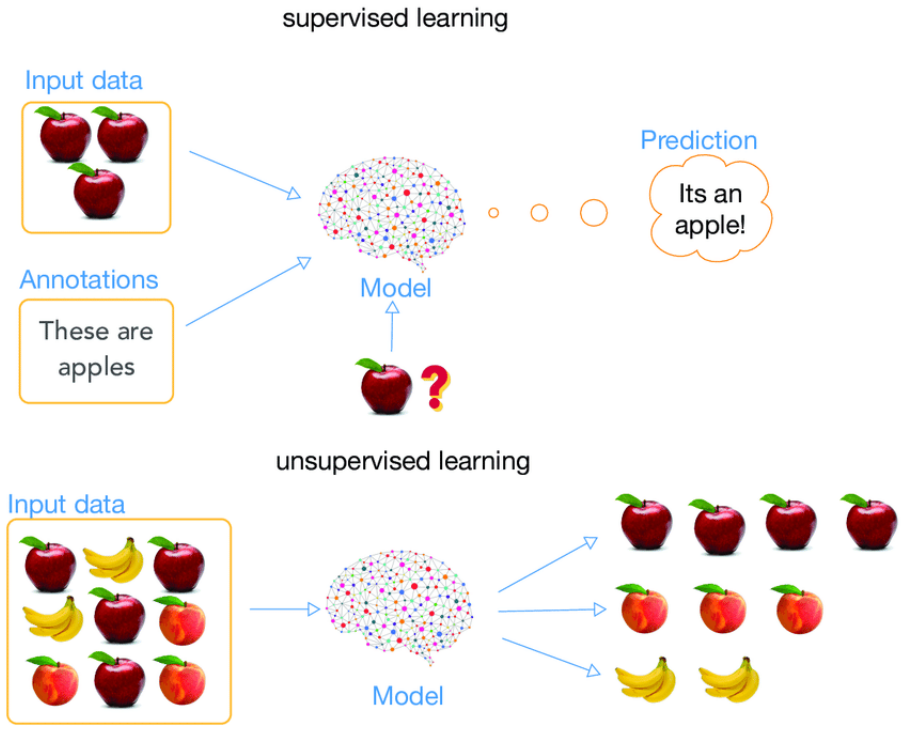


Figure 2.1: Difference between supervised and unsupervised learning. Figure taken from reference [2]

2.2.3 Reinforcement Learning

In the reinforcement learning, our system is an "autonomous agent" that has to explore an unknown "space" and determine the actions to carry out through trial and error. It learns by itself by obtaining rewards and penalties until it finds the optimal way to travel a path, solve a puzzle or display a behavior. This type of algorithm is used to find the best possible strategy to maximize a company's profits.

2.3 Deep learning

Deep learning is a subset of machine learning that deals with algorithms inspired by the structure and function of the human brain. Deep learning algorithms can work with an enormous amount of both structured and unstructured data. Deep learning's core concept lies in artificial neural networks, which enable machines to make decisions [1].

The major difference between deep learning vs machine learning is the way data is presented to the machine. Machine learning algorithms usually require structured data, whereas deep learning networks work on multiple layers of artificial neural networks[cite]. This mainly means that these structures are made up of objects, called neurons, which receive a number of inputs that determines their state: whether they are "activated" or not. Subsequently, the output of this neuron (whether it is activated or not) is one of the inputs that other neurons use to determine whether they are activated or not. This process is computed through the so-called activation functions.

2.3.1 Neural Networks

When we talk about an artificial neuron we refer to an abstract object that stores a number. This number is known as the activation value, which is the output value of the neuron. This value is obtained either by direct assignment (if it is a neuron of the first layer of the network) or through an activation function, whose variables are the activation values of the preceding neurons. In the specific case of Deep learning, the aforementioned algorithmic structures allow models that are composed of multiple processing layers of artificial neurons to learn data representations with multiple levels of abstraction that perform a series of linear and nonlinear transformations such that they generate an output close enough to the expected value.

To understand this more precisely and to define what our neural network learns, it is necessary to introduce the foundation of artificial neurons: the perceptron.

2.3.2 Perceptron

Perceptrons were developed in the 1950s and 1960s by neuroscientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts[3]. A perceptron takes several binary input values x_1, x_2, x_3... and produces a single binary output. Rosenblatt proposed a simple rule to calculate the output, he introduced weights w_1, w_2 being real numbers expressing the importance of the respective input. The output of the neuron, 0 or 1, is determined if the weighted sum $\sum_j x_j w_j$ is greater or less than some threshold value. Like the weights, the threshold is a real number that is a parameter of the neuron. In other words:

$$output = \begin{cases} 0 & if \quad \sum_j x_j w_j < b' \\ 1 & if \quad \sum_j x_j w_j > b' \end{cases} \tag{2.1}$$

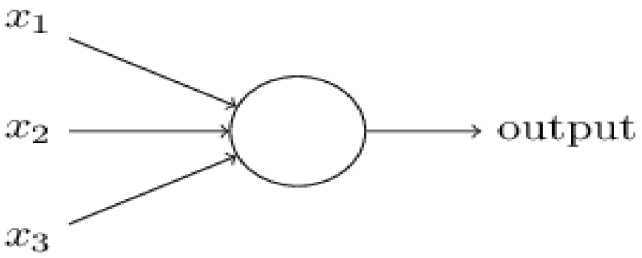


Figure 2.2: Representation of a three-input perceptron[4].

To further clarify the idea of weights and threshold value, we have the following example. Suppose you have been invited to a party on the weekend, but there are several factors that condition whether you are going to go or not. Let's say there are three:

1. Is there anyone to accompany me to the party?
2. Will the weather be good?
3. Is there public transportation to take me to the site?

The answer to each of these questions is 0 if no and 1 if yes. These values are the inputs received by our perceptron. However, each person is different and it may be that for you it is not as important to have someone accompanying you as it is to have good weather. So, the answer to the second question has more weight for you than the other two. In numbers, it can be said that the answers to the first and last question have a weight (importance) of 2, $w_1 = w_3 = 2$, meanwhile the answer to the second one has a weight of 6, $w_2 = 6$. Now let's say that you have decided to go to the party if the sum of the inputs multiplied by their respective weights is greater than a fixed value, the threshold, which you have decided to be 3. The perceptron has thus been transformed into a decision-making model, in which you go to the party if $\sum_1^3 x_j w_j \geq 3$ and you do not otherwise.

We can rewrite Eq. 1 as follows:

$$output = \begin{cases} 0 \quad if \quad \vec{x} \cdot \vec{w} + b \leq 0 \\ \\ 1 \quad if \quad \vec{x} \cdot \vec{w} + b > 0 \end{cases} \tag{2.2}$$

b is what is known as the perceptron bias, which as can be seen is closely related to the threshold.

Bias can be thought as a measure of how easy it is to get the perceptron to emit a 1, or to put it in biological terms, it is a measure of how easy it is to get the perceptron to fire. For a perceptron with a really large bias, it is extremely easy to generate a 1. But if the bias is very negative, then it is difficult for the perceptron to generate a 1. Obviously, introducing the bias is only a small change in how we change in how we describe perceptrons, but we will see later that it leads to further simplifications of notation.

We can connect several perceptrons in different layers, in such a way that the activation values of one layer are the input values of the next one. In this way, we are building what is known as a multilayer perceptron. Also, each perceptron (neuron) in this network has a particular bias value b associated with it, which determine whether it is activated or not according to Eq. 2.2.

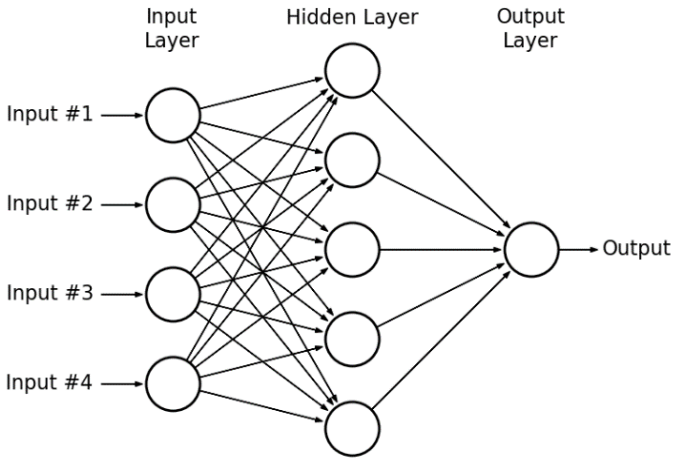


Figure 2.3: Representation of a multilayer perceptron. Image taken from Ref. [4]

In the figure, the edges connecting each neuron from one layer to the next represent the weights that each connection has. Thus, the same activation value affects each of the neurons in the next layer of the network in a different way. Mathematically, we can represent the weighted connections between two layers with a matrix W of dimensions nxm, where n represents the number of neurons of the subsequent layer and m those of the previous layer. The reason for this is that it is easier to represent the calculation of the activation values of each of the neurons of the second layer. Let's see it. Let $\vec{x}$ be the vector of all the values of the neurons of the first layer (in the case of Fig. 2.3, four), W be a matrix of dimensions 5×4 and $\vec{b}$ be the vector containing the five biases of the neurons from the hidden layer. Then, a vector $\vec{y'}$ containing the pre-outputs of the second layer can be obtained as follows:

$$W\vec{x} + \vec{b} = \vec{y'} \tag{2.3}$$

"Pre" stands for the need to yet apply the decision rule described in Eq. 2.2 to the inputs of the vector $\vec{y'}$ in order to obtain final values 0s and 1s. Here is where perceptrons present their big problem, which is described below.

2.3.3 Sigmoid neuron

Suppose we have a perceptron network that we would like to use to learn how to solve some problem. For example, the inputs to the network might be the raw pixel data of a scanned, handwritten image of a digit. And we would like the network to learn the weights and biases so that the network output correctly classifies a digit. To see how the learning might work, suppose we make a small change to some weight (or bias) in the network. What we would like is for this small change in weight to cause only a small corresponding change in the network output. As we will see next, this property will make learning possible. Schematically, what we want is represented in the Fig. 2.4

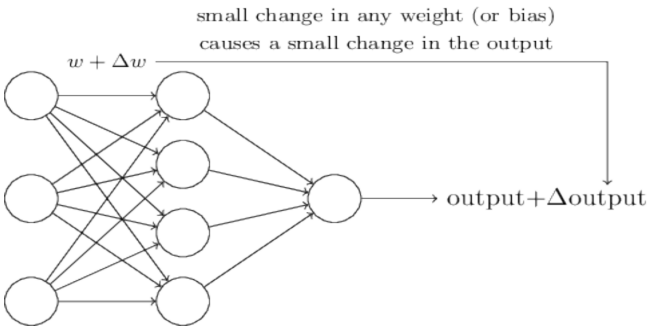


Figure 2.4: A small change in a weight implies a small change in output.

If it were true that a small change in a weight (or bias) causes only a small change in the output, then we could use this fact to modify the weights and biases to make our network behave more the way we want it to. This process of modifying weights and biases in order to obtain the desired output is known as learning. For example, suppose a network mistakenly classifies an image as an "4" when it should be a "7". We could figure out how to make a small change in the weights and biases so that the network gets a little closer to classifying the image as a "7". And then we would repeat this, changing the weights and biases over and over for each number from our dataset to produce a better result.

The problem is that this is not possible when a perceptron network is used. In fact, a small change in the weights or bias of any individual perceptron in the network can sometimes cause the output of that perceptron to change completely from 0 to 1. That change can cause the behavior of the rest of the network to change completely in a very complicated way. We can overcome this problem by introducing a new type of artificial neuron called a sigmoid neuron. Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and biases cause only a small change in their output, since it is not binary. That is the crucial fact that allows a network of sigmoid neurons to learn. Sigmoid neurons are represented in the same way as the perceptrons schematized in Fig. 2.2

Like a perceptron, sigmoid neurons have inputs x_1, x_2, x_3... but instead of these being only values 0 and 1, they can be all values from 0 to 1. Also, sigmoid neurons have weights for each input w_1, w_2, w_3... and a general bias b; however the output of the new form are not only values 0 and 1, instead they are of the form $\sigma(\vec{x} \cdot \vec{w} + b)$ where σ is called the sigmoid function defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

Therefore, the output of a sigmoid neuron is given by:

$$output = \frac{1}{1 + exp\left(-\sum_j w_j x_j - b\right)} \tag{2.5}$$

The graph of this function is shown in Fig. 2.5

By using the real σ function we obtain a smoothed perceptron. The smoothness of the function σ is a crucial fact of this whole development. The smoothness of σ means that small changes Δw_j in the weights and Δb in the bias produce a small change $\Delta output$ in the output of the neuron. In fact, calculus tells us that output is well approximated by:

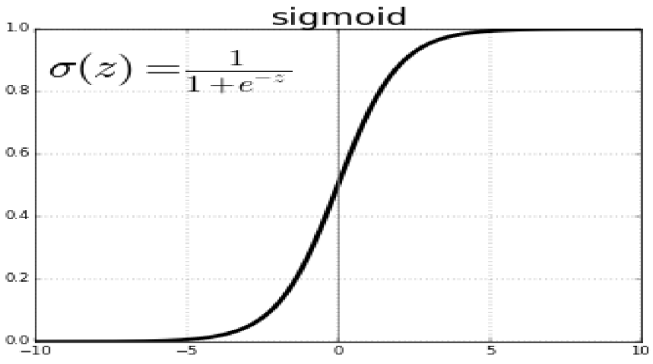


Figure 2.5: Sigmoid neuron

$$\Delta output \approx \sum_j \frac{\partial \Delta output}{\partial w_j} \Delta w_j + \sum_j \frac{\partial \Delta output}{\partial b} \Delta b \tag{2.6}$$

The major difference between perceptrons and sigmoid neurons is that sigmoid neurons do not only output 0 or 1. They can output any real number between 0 and 1. This can be useful, for example, if we want to use the output value to represent the average pixel intensity in an input image to a neural network. But sometimes it can be a nuisance. Suppose we want the output of the network to indicate "the input image is a 7" or "the input image is not a 7". Naturally, it would be easier to do this if the output were a 0 or a 1, as in a perceptron. But in practice we can establish a convention for dealing with this, for example, deciding to interpret any output of at least 0.5 as indicating a "7", and any output less than 0.5 as indicating "not a 7".

2.3.4 Hidden Layers

Let's say we have the following neural network:

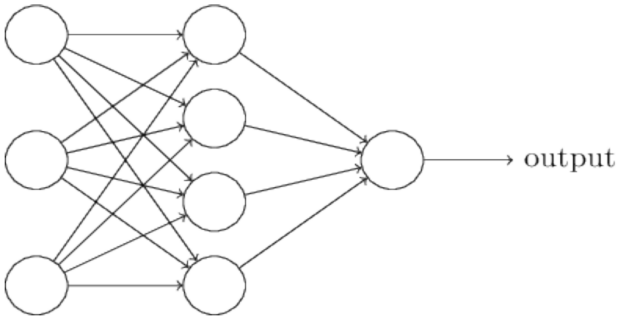


Figure 2.6: A simple neural network [5].

The leftmost layer in this network is called the input layer and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons or, in this case, a single output neuron. The middle layer is called the hidden layer, since the neurons in this layer are neither inputs nor outputs. The term *hidden* means nothing more than "neither an input nor

an output" and, in fact, this is where deep learning gets its name from. The above network has a single hidden layer, but some networks have multiple hidden layers. For example, the four-layer network shown in Fig. 2.7 has two hidden layers.

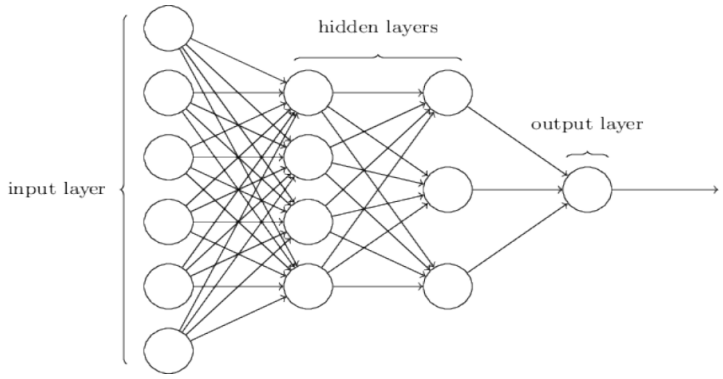


Figure 2.7: A four-layer neural network. [5].

For historical reasons, these multilayered networks are sometimes referred to as multilayered perceptrons or MLPs, even though they consist of sigmoid neurons, not perceptrons.

While the design of the input and output layers of a neural network is usually straightforward, the design of the hidden layers can be quite an art. In particular, it is not possible to summarize the process of designing hidden layers with a few basic rules. Instead, neural network researchers have developed many design heuristics for hidden layers that help people get the behavior they want from their networks.

2.3.5 Cost Function and Gradient Descent

In previous sections we studied the design of a neural network, but we did not discuss how it learns. That is, how it modifies the weights and biases so that the output of the network, $y(x)$ approximates to the real expected values for all x training inputs. For this purpose, a so-called loss function is defined below:

$$C(w,b) = \frac{1}{2n} \sum_{x} ||y(x) - a||^2 \tag{2.7}$$

The value of this function grows as we move further away from the expected outputs. w denotes the collection of all the weights in the network, b all the biases, n is the total number of training inputs, a is the vector of network outputs when x is the input, and the sum is over all the training inputs, x. The output a depends on x, w and b, but to keep the notation simple this dependence is not explicitly stated. The notation $|v|$ simply denotes the usual length function for a vector v.

We call C the quadratic cost function; it is sometimes also referred to as the mean square error or simply MSE. By inspecting the form of the quadratic cost function, we see that $C(w,b)$ is non-negative, since all the terms of the sum are non-negative. Moreover, the cost $C(w,b)$ becomes small, i.e., $C(w,b) \approx 0$, precisely when $y(x)$ is approximately equal to the output, a, for all training inputs, x.

We know that our neural network is better trained the smaller the value of the cost function. Thus, the purpose of training a neural network is to minimize C, and for this we use two fundamental algorithms in deep learning: Gradient Descent and Backpropagation.

Suppose we try to minimize some function $C(v)$, which can be any function of several variables ($v = v_1, v_2, ...$). It might help us to visualize this by taking C as a function of two variables which we call v_1 and v_2.

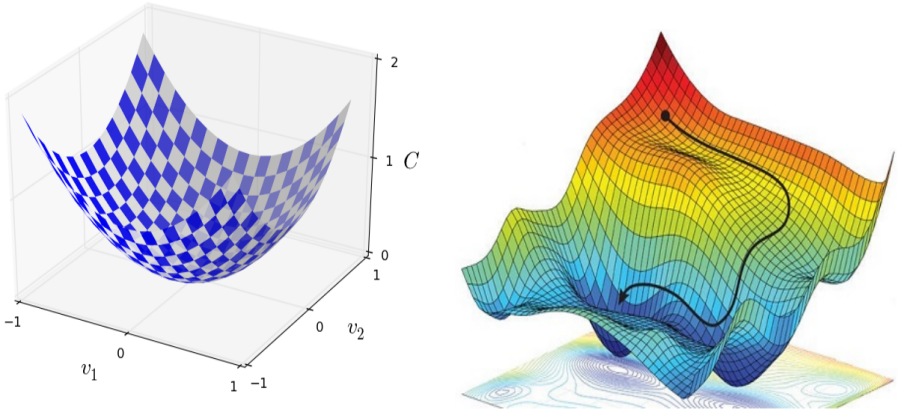


Figure 2.8: Visualization of the gradient descent algorithm. Images taken from [5]

.

The task is to find where C reaches its global minimum. Fig. 2.8 (left) is a fairly simple function to exemplify, however C can be a very complicated function of multiple variables as Fig. 2.8 (right).

One way to attack the problem is to use calculus to try to find the minimum analytically. By these means, we calculate derivatives and then try to use them to find places where C is an extreme. With a little luck, that might work when C is a function of only one or a few variables, but it gets complicated when we have many more variables. And for neural networks, it is wanted to have many more variables: the largest neural networks have cost functions that depend on billions of weights and biases in an extremely complicated way.

Since calculus does not help us in this problem we have to attack it in another way. We start by thinking of our function as a kind of valley and visualizing a ball rolling down the side of our valley, rolling to the bottom. What we want to find is the trajectory that the ball will follow, since, naturally, this will be the fastest to reach the bottom.

This can be thought of as the ball constantly finds the direction of the steepest slope, and advances a minuscule amount until it finds a steeper direction. This direction is mathematically given by the negative gradient of the point where it meets, so if we say that the initial position of the ball is x_0, and that the minimum distance it moves is α, the ball will reach the bottom of the valley, x_b, if it iterates the following operation:

$$\vec{x_b} = \vec{x_0} - \alpha \nabla f(\vec{x_0}) \tag{2.8}$$

Where alpha is known as the learning rate. Thus, an approximation of the minimum of a given cost function can be found numerically by iterating Eq. 2.8 .

2.3.6 Convolutional Neural Networks

A common task where machine learning is used is in image recognition, for instance, if we want an algorithm to determine whether a image is, let's say, a panda or not. Humans have the ability to recognize animals or objects because we have the capacity to remember patterns. That is, from the optical information collected by our eyes from the photographs in 2.9, our brain is able to recognize that, although with different luminosity and angle of exposure, the two pairs of eyes in the photos belong to the same kind of animal. Likewise with their ears, horns, spots, etc.; this recognition works even if there are variations in the objects.

Figure 2.9: A color image consists of three matrices, one for each color in the RGB (red, green and blue) model, where the input of each matrix indicates the intensity of each of these colors. Images taken from [6]

How can we recreate this ability on a computer? The most efficient way is to do it using convolutional neural networks [7] [8], which their function is to identify patterns in the features of an object. First of all, an image for a computer is nothing more than a matrix where the intensity of each pixel is represented by the value of the corresponding input in the matrix. This value ranges from 0 to 255, but we can normalize it to values from 0 to 1.

Intuitively we can think that the first thing an image recognition algorithm must do is to be able to distinguish, from a subset of pixels, the lines, edges and textures (and their distribution) that identify a specific feature of the image. Generally, as will be seen below, this is achieved by committing a neural network layer for each level of abstraction in the features of an image [7]. This means that while the first layer is in charge of distinguishing, for example, between edge lines, the second layer must group these results to identify noses, ears and eyes, and the last layer combines this information to distinguish between a head in the image (see Fig. 2.10).

One way to train a neural network is by reshaping the nxn matrix into a $(n^2)x(1)$ vector, and feed the first layer with that input. However, doing that we are losing all the image spatial information, and that represents an extra cost for our algorithm. So, the alternative is the so-called convolution operation. By means of this operation, each of the features of an image is represented by a matrix (or filter) smaller than the image, so that the original matrix can be "traversed" by overlapping and multiplying its entries with those of the filter, adding these values and thus creating a feature map (see Fig 2.11). When the filter matches a section of the image that is similar to the feature it wants to identify, the "activation value" of that filter will be higher.

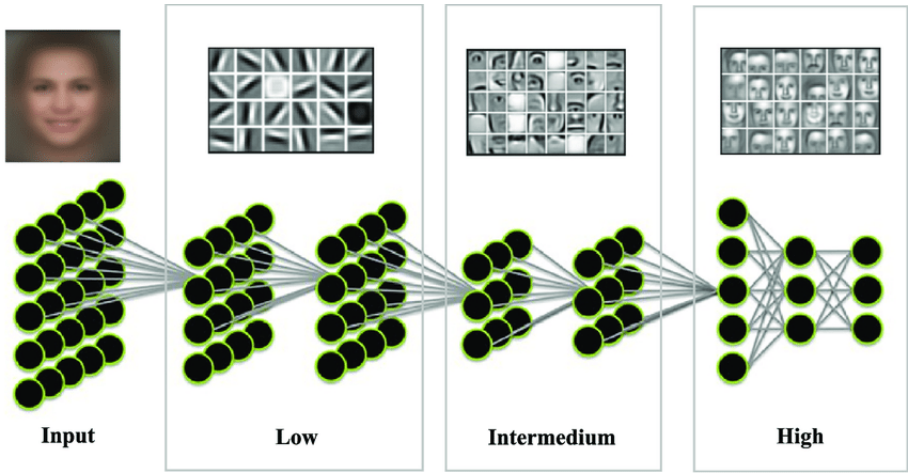


Figure 2.10: Intuitive idea of the abstraction levels of a CNN in face recognition.[9]

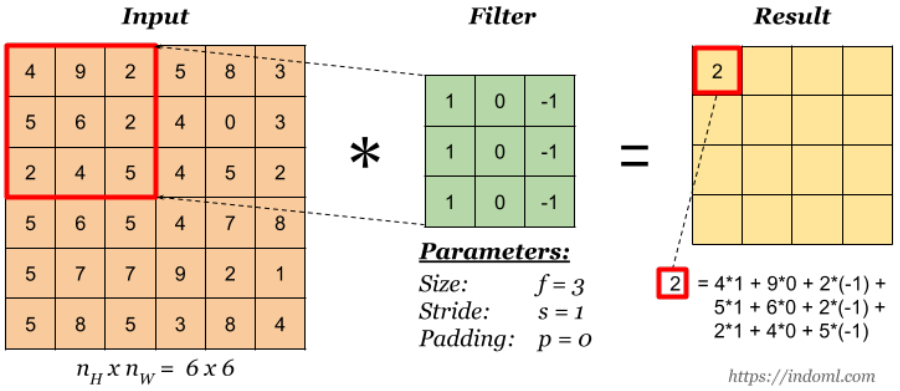


Figure 2.11: In a neural network, the outputs of this operation are the inputs of the next layer of neurons, or an additional function called MaxPooling (or AveragePooling) is usually applied to them.

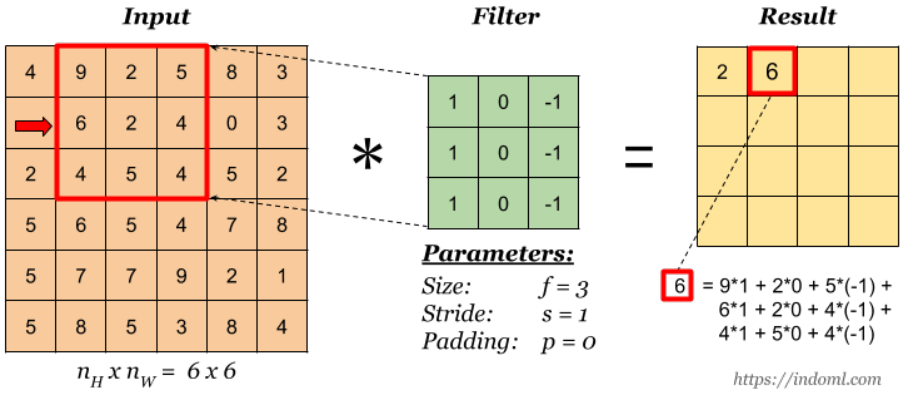


Figure 2.12: Images taken from [10]

Convolutional layers in a neural network are groups of neurons specialized in two operations: convolution and pooling. The main difference between a dense layer and a convolutional layer is that the former learns global patterns from the entire input space, while the latter learns local patterns using small two-dimensional windows (subset of pixels), known as a kernel. This is an advantage for convolutional networks, because once a feature is learned at one position in an image, it will be able to recognize it at any other position in other images.

Each neuron in a convolutional layer is not connected to all neurons in the previous layer, as is the case in densely connected layers. Instead, the first convolutional neuron receives the scalar product between the first subset of image pixels and the filter (kernel). This first subset is usually located in the upper left corner of the image (see Fig. 2.11) and is then «swept» until it covers the entire image (see 2.12). With reference to the above images, while the input data layer has 36 neurons, the first hidden layer (result) has 16 (4x4) after the first convolution operation.

An important aspect to note is that the kernel entries (or weights) do not change as the image is swept, so a single kernel is only able to identify a specific pattern in the image. So, if you want to identify several patterns in an input, you will need to use one kernel and one convolutional layer for each, operating in parallel (i.e., they are not connected to each other). This can be visualized in Fig. 2.13.

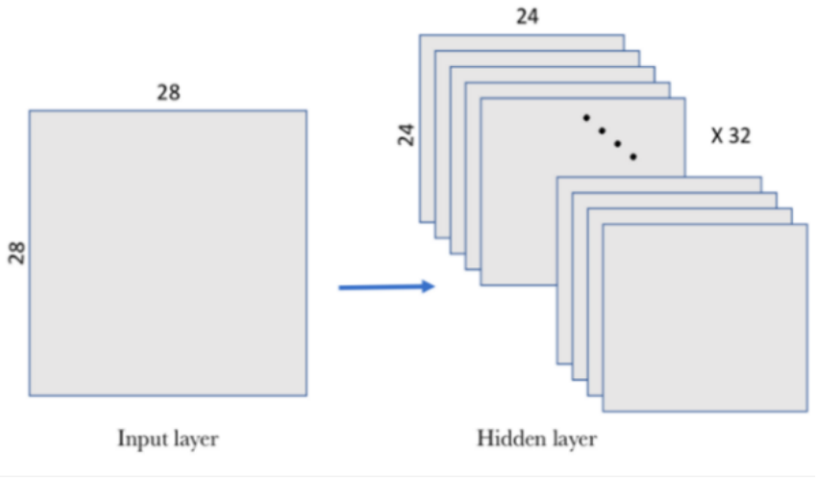


Figure 2.13: In this example 32 kernels were assigned to identify 32 features on the 28x28 image [11].

On the other hand, in the example of Fig. 2.11 and Fig. 2.12, it can be seen that the kernel travels over the input matrix with an advance of one pixel; however, this does not have to be the case. The step value can be set using the «stride» parameter. Similarly, in convolutional networks a padding technique can be applied to add zero-valued pixels around the image so that information about the image boundaries is included.

To conclude this introduction to CNNs, it is worth mentioning a type of layer that generally accompanies convolutional layers: pooling layers. A pooling layers create a condensed version of the outputs obtained by the previous layer. This is done by choosing the size of a kernel that sweeps the feature matrix obtained from the previous convolution operations, then condenses that information into a single value. The latter can be done in several ways. The most usual are taking the largest value among the kernel inputs (MaxPooling) or calculating the average of them (AveragePooling).

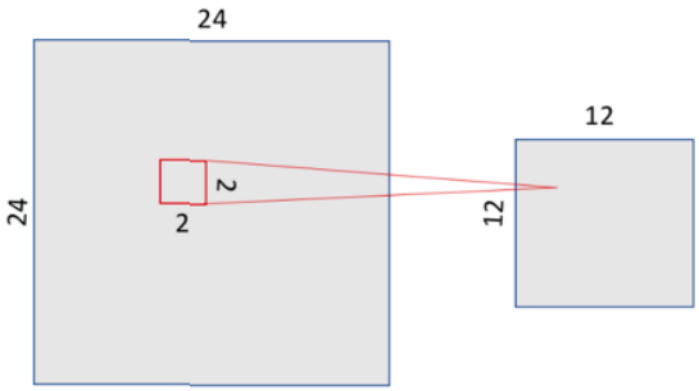


Figure 2.14: In this example, a 2x2 kernel is applied to a 24x24 matrix, and for each of those four pixels an unique value is returned in the 12x12 feature matrix.

Chapter 3

Quantum Computing

3.1 Two-level systems: qubits

A *quantum bit*, or *qubit* for short, is a *mathematical object* used to describe a *state* [12]. A state can be 0 or 1 (akin to a classical bit), or a *superposition* of these. For example, the 0 state of a qubit may represent the ground state of an atom, while the 1 state may represent the excited one. Also, these *basis states* can be used to represent the spin orientation of the atoms used in the Stern-Gerlach experiment, as described in subsection 3.1.4.

The bra-ket Dirac notation leads us to express the basis states of a qubit as $|0\rangle$ and $|1\rangle$. Consequently, any qubit state can be described as a *linear combination* of $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{3.1}$$

Where α and β are complex numbers called amplitudes. $|0\rangle$ and $|1\rangle$ form an orthonormal basis for this complex vector space, and they are known as *computational basis states*. Furthermore, since $|0\rangle$ and $|1\rangle$ are orthogonal, $\alpha = \langle 0|\psi\rangle$ and $\beta = \langle 1|\psi\rangle$.

Although the laws of Quantum Mechanics allow a qubit to be in a superposition of $|0\rangle$ and $|1\rangle$, when we measure this qubit we obtain one single value: $|0\rangle$ with a probability of $|\alpha|^2$ or $|1\rangle$ with a probability of $|\beta|^2$. It follows that $|\alpha|^2 + |\beta|^2 = 1$, i.e. a qubit's state is normalized to 1. Therefore, geometrically, we can visualize a qubit's state as a unit vector in a two-dimensional complex vector space, which we introduce as Bloch's Sphere below.

3.1.1 Bloch's sphere

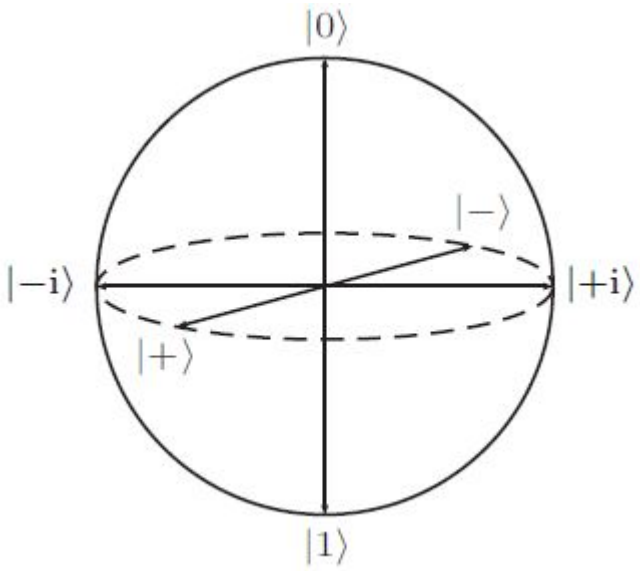


Figure 3.1: The Bloch's Sphere and three pairs of common used basis. Image taken from [13]

A two-level quantum system has an infinite number of bases, consisting of two orthogonal states. It is easy to see that, given any physical state, say $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, we can form a computational basis by obtaining the state orthogonal to it, i.e. $|\psi_\perp\rangle = \alpha^*\,|0\rangle - \beta^*\,|1\rangle$. This way, $\langle\psi|\psi_\perp\rangle = 0$, $\langle\psi|\psi\rangle = 1$ and $\langle\psi_\perp|\psi_\perp\rangle = 1$. In the Bloch Sphere, each pair of basis states can be visualised as two opposite poles, as seen in Fig. 3.1. Physically, kets $|0\rangle$ and $|1\rangle$ can represent any pair of quantum states of a particle. For example, its spin up $|\uparrow\rangle$ and spin down $|\downarrow\rangle$; or the ground $|g\rangle$ and excited $|e\rangle$ state of an atom.

Recalling some properties of complex numbers, we can write the amplitudes of a quantum state described by equation 3.1 as a function of a γ angle and two phases, i.e. $\alpha = e^{i\phi_1} cos\gamma$ and $\beta = e^{i\phi_2} sin\gamma$, in such a way that the normalisation relation is satisfied:

$$|\alpha|^2 + |\beta|^2 = |e^{i\phi_1} cos\gamma|^2 + |e^{i\phi_2} sin\gamma|^2$$
$$= |e^{i\phi_1}|^2 cos^2\gamma + |e^{i\phi_2}|^2 sin^2\gamma = sin^2\gamma + cos^2\gamma = 1$$

Therefore, equation 3.1 can be written as:

$$|\psi\rangle = e^{i\phi_1} cos\gamma\,|0\rangle + e^{i\phi_2} sin\gamma\,|1\rangle = e^{i\phi_1}(cos\gamma\,|0\rangle + e^{i(\phi_2-\phi_1)} sin\gamma\,|1\rangle)$$

However, a phase applied to a quantum state has no observable effects [12], so we can ignore the $e^{i\phi_1}$ term in the above equation, so that:

$$|\psi\rangle = cos\gamma\,|0\rangle + e^{i\phi} sin\gamma\,|1\rangle \tag{3.2}$$

The angles ϕ and γ of equation 3.2, which describe the state of a two-level quantum system, parameterise the Bloch Sphere. Thus, any well-determined quantum system (pure state) can be represented *on* the surface of this unitary sphere. In the following sections, we will see that those

quantum states that cannot be represented as a linear combination of two bases (mixed states) lie *inside* the sphere.

Having defined our base 0 and 1, there are two other base pairs that are quite common and can be written in terms of this first one. They are shown in Fig. 3.1 and are defined as:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{3.3}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{3.4}$$

$$|i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle) \tag{3.5}$$

$$|-i\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle) \tag{3.6}$$

3.1.2 Dirac Notation

The Dirac notation is used to denote qubits states. Recall that this notation is used to represent elements in a Hilbert space, where their inner product, (x, y) is denoted as $\langle x|y\rangle$ and $x, y \in \mathbb{C}^n$. However, as for the moment we are only interested in two-level systems, the state of a qubit is represented as an element of the set $\mathbb{C}^2$, i.e. $|\psi\rangle \in \mathbb{C}^2$. In this way:

$$|0\rangle \doteq \begin{pmatrix}1\\0\end{pmatrix} \qquad |1\rangle \doteq \begin{pmatrix}0\\1\end{pmatrix}$$

And, remembering Eq. 3.1:

$$|\psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix} \qquad \langle\psi| = \begin{pmatrix}\alpha^* & \beta^*\end{pmatrix}$$

It follows that the inner product of two states can be written as:

$$\langle\psi_1|\psi_2\rangle = \begin{pmatrix}\alpha_1^* & \beta_1^*\end{pmatrix}\begin{pmatrix}\alpha_2\\\beta_2\end{pmatrix} = \alpha_1^*\alpha_2 + \beta_1^*\beta_2 \tag{3.7}$$

The state of two qubits together is an element of $\mathbb{C}^4$, since it is the tensor product of these two states, i.e. $|\psi_1\rangle \otimes |\psi_2\rangle$ (or $|\psi_1\rangle\,|\psi_2\rangle$ for short), and can be written as follows in the computational basis $|0\rangle$ and $|1\rangle$:

$$\begin{aligned}|\psi_1\rangle\,|\psi_2\rangle &= (\alpha_1\,|0\rangle + \beta_1\,|1\rangle)(\alpha_2\,|0\rangle + \beta_2\,|1\rangle)\\ &= \alpha_1\alpha_2\,|0\rangle\,|0\rangle + \alpha_1\beta_2\,|0\rangle\,|1\rangle + \beta_1\alpha_2\,|1\rangle\,|0\rangle + \beta_1\beta_2\,|1\rangle\,|1\rangle\end{aligned}$$

Which is more usually expressed as:

$$|\psi_1\rangle\,|\psi_2\rangle = \alpha_1\alpha_2\,|00\rangle + \alpha_1\beta_2\,|01\rangle + \beta_1\alpha_2\,|10\rangle + \beta_1\beta_2\,|11\rangle \tag{3.8}$$

Where:

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \qquad |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \qquad |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \qquad |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Leading us to another common way to write Eq. 3.8:

$$|\psi_1\rangle |\psi_2\rangle = \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix} \tag{3.9}$$

From the above, it is easy to realise that, as qubits are added to the system (a quantum computer, for example), the information that can be stored (or the number of coefficients) doubles. In other words, for a system of n qubits, 2^n coefficients are needed to describe it.

We have defined the inner product of two qubit states (Eq. 3.7) as an operation that takes us from a space $\mathbb{C}^2$ to $\mathbb{C}$ (i.e. $\mathbb{C}^2 \times \mathbb{C}^2 \to \mathbb{C}$). Now let's define the outer product of $|\psi_1\rangle$ and $|\psi_2\rangle$ as an operation that takes us from a space $\mathbb{C}^2$ to one of linear operators, i.e. $\mathbb{C}^2 \times \mathbb{C}^2 \to \mathbb{O}(\mathbb{C}^2)$. Namely:

$$|\psi_1\rangle \langle\psi_2| = \begin{pmatrix} \alpha_1 \\ \beta_1 \end{pmatrix} \begin{pmatrix} \alpha_2^* & \beta_2^* \end{pmatrix} = \begin{pmatrix} \alpha_1\alpha_2^* & \alpha_1\beta_2^* \\ \beta_1\alpha_2^* & \beta_1\beta_2^* \end{pmatrix}$$

3.2 Quantum gates and circuits

3.2.1 Single qubit gates

A quantum gate is an operation that transforms the state of a qubit $|\psi\rangle$. Since $|\psi\rangle$ is a linear combination of two quantum states, a quantum gate is an unitary operator ($O^\dagger O = \mathbb{1}$), which is usually denoted by capital letters and can be described by a matrix nxn, where n is the number of bases in our system.

$$|\psi_2\rangle = O |\psi\rangle$$

A quantum operator must be a linear operation, i.e.:

$$O(\alpha |0\rangle + \beta |1\rangle) = \alpha O |0\rangle + \beta O |1\rangle$$

Therefore, quantum gates are linear maps that keep the total probability equal to 1 [14]. As we are working on a two-level system, the these gates for a single qubit must operate on the space $\mathbb{C}^2$. In other words, they must be 2x2 matrices.

The importance of O being a unitary operator lies in the fact that we are working with normalised vectors. Namely, if:

$$\langle\psi_1|\psi_1\rangle = 1$$

and

$$|\psi_2\rangle = O |\psi_1\rangle \Leftrightarrow \langle\psi_1| O^\dagger = \langle\psi_2|$$

Where $O^\dagger$ is the hermitian adjoint of O. It must be fulfilled that:

$$\langle\psi_2|\psi_2\rangle = 1$$
$$\Leftrightarrow (\langle\psi_1| O^\dagger)(O |\psi_1\rangle)$$
$$\Leftrightarrow \langle\psi_1| \mathbb{1} |\psi_1\rangle = 1$$

Examples of single-qubit quantum gates

1. The identity operator $\mathbb{1}$ is an operator that applied to a state $|\psi\rangle$ returns the same state, and can be represented as the unitary matrix: $\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

$$\mathbb{1} |\psi\rangle = |\psi\rangle$$

2. Pauli gates, which can be represented with the Pauli matrices:

   (a) Pauli-X gate:

   $$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

   This gate behaves like a classical NOT gate applied on a computational basis.

   $$X|0\rangle = |1\rangle \qquad X|1\rangle = |0\rangle$$

   This can be easily seen by performing the corresponding matrix operations.

   $$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

   On the Bloch sphere, the X-gate has the effect of rotating $|\psi\rangle$ 180^{o} about the x-axis.

   (b) Pauli-Y gate:

   $$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

   Applied to the computational basis:

   $$Y|0\rangle = i|1\rangle \qquad Y|1\rangle = -i|0\rangle$$

   On the Bloch sphere, the X-gate has the effect of rotating $|\psi\rangle$ 180^{o} about the y-axis.

   (c) Pauli-Z gate:

   $$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

   Applied to the computational basis:

   $$Z|0\rangle = |0\rangle \qquad Z|1\rangle = -|1\rangle$$

   On the Bloch sphere, the X-gate has the effect of rotating $|\psi\rangle$ 180^{o} about the Z-axis.

   Y and Z gates do not have an equivalent in classical computing.

   It is easy to see that implementing any of the Pauli gates twice on a qubit will have no effect on it (a 360° rotation on the Bloch sphere is applied). This means that these operators are idempotent, i.e. $X^2 = X$, $Y^2 = Y$ and $Z^2 = Z$.

3. The Hadamard gate operating on a base state creates a superposition between the two base states, with equal probability for each. It is represented by the following matrix:

   $$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

   And, recalling Eq. 3.3 and 3.4, their effect on the computational basis is:

   $$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$$

   $$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$$

   Also can be shown that:

   $$H|+\rangle = |0\rangle \qquad H|-\rangle = |1\rangle$$

   and remembering Eq. 3.5 and 3.6:

   $$H|i\rangle = |-i\rangle \qquad H|-1\rangle = |i\rangle$$

   This gate represents a 180^{o} rotation about the $x - z$ axis, therefore it is also idempotent ($H^2 = H$).

4. The Phase gate S adds a phase i to a quantum state, i.e. it applies a 90° rotation about the z-axis. This gate can be seen as the square root of the Z, i.e. $S^2 = Z$.

$$S|0\rangle = |0\rangle \qquad S|1\rangle = i|1\rangle$$

5. The T-gate (or $\pi/8$ gate) is the square root of the S-gate ($T^2 = S$ and $T^4 = Z$). This gate rotates one qubit 45^{o} about the z-axis.

$$T|0\rangle = |0\rangle \qquad T|1\rangle = e^{i4/\pi}|1\rangle$$

On the Bloch Sphere, one-qubit quantum gates are rotations along an axis. This can be proved by checking two facts that characterise rotations on the Bloch sphere: they are norm-preserving and they are linear maps [cite]. Moreover, it can be shown that any rotation around an axis $\hat{n} = n_x\hat{x} + n_y\hat{y} + n_z\hat{z}$ can be written as a particular case of the general quantum gate [14]:

$$U = e^{i\gamma}\left[cos\left(\frac{\theta}{2}\right)I - isin\left(\frac{\theta}{2}\right)(n_xX + n_yY + n_zZ)\right] \tag{3.10}$$

Where $\hat{n}$ is an unit vector ($n_x^2 + n_y^2 + n_z^2 = 1$). For example, the Hadamard gate can be written from the above equation with $\theta = \pi$ and $\hat{n} = \left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right)$:

$$\begin{aligned} U_H &= e^{i\gamma}\left[cos\left(\frac{\pi}{2}\right)I - isin\left(\frac{\pi}{2}\right)\left(\frac{1}{\sqrt{2}}X + 0Y + \frac{1}{\sqrt{2}}Z\right)\right] \\ &= -ie^{i\gamma}\frac{1}{\sqrt{2}}(X + Z) \end{aligned}$$

Utilising it over $|0\rangle$:

$$\begin{aligned} U_H|0\rangle &= -ie^{i\gamma}\frac{1}{\sqrt{2}}(X + Z)|0\rangle \\ &= -ie^{i\gamma}\frac{1}{\sqrt{2}}(X|0\rangle + Z|0\rangle) \\ &= -ie^{i\gamma}\frac{1}{\sqrt{2}}(|1\rangle + |0\rangle) \\ &= -ie^{i\gamma}|+\rangle = |+\rangle \end{aligned}$$

We can drop the phase $-ie^{i\gamma}$ since it has no effect on the global state. A similar development proves that $U_H|1\rangle = |-\rangle$.

3.2.2 Quantum Circuits

The application of one or more quantum gates on a qubit forms what is known as a quantum circuit, and is generally visualised as shown below.

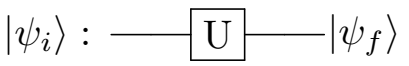


Figure 3.2: The line connecting the initial state of the qubit $|\psi_i\rangle$ to its final state $|\psi_f\rangle$, passing through the U-gate, is known as the qubit's lifeline.

Several quantum gates can be used consecutively on a qubit. For example, having $HXZH\,|0\rangle$, a Hadamard gate acts on the qubit $|0\rangle$, then a Pauli-Z gate, then a T gate and finally an H gate again. The action of the gates on the qubit occurs from right to left, but in the circuit diagram, the gates act from left to right following the qubit's lifeline.

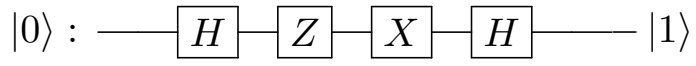


Figure 3.3: $HXZH\,|0\rangle$ operation

The final state seen in the diagram above is obtained by the following operations:

$$\begin{aligned} HXZH\,|0\rangle &= \frac{1}{\sqrt{2}} HXZ(|0\rangle + |1\rangle) \\ &= \frac{1}{\sqrt{2}} HX(|0\rangle - |1\rangle) \\ &= \frac{1}{\sqrt{2}} H(|1\rangle - |0\rangle) \\ &= \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) - \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \right) \\ &= -|1\rangle \end{aligned}$$

In Fig. 3.2, it is usually implied that a state measurement is conducted at the end of the operation, but sometimes it can also be denoted by a meter:

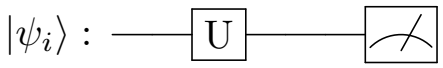


Figure 3.4:

3.3 Multi-qubit gates

3.3.1 Separable gates

The gates described above can only be used for a single qubit. To represent the action of several gates on a system of two or more qubits, it is necessary to perform the tensor product between the gates and then apply it to the state of the qubits. Two one-qubit gates, A and B, operating on two qubits, $|0\rangle$ and $|1\rangle$, is mathematically described as:

$$A \otimes B\,|01\rangle = A\,|0\rangle \otimes B\,|1\rangle$$

If we set $A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix}$ and $B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$, the above equation in its matrix form would look like:

$$\begin{pmatrix} A_1 \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} & A_2 \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \\ A_3 \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} & A_4 \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

We can write the effect of n equal gates U on n qubits in the same state (e.g. $|0\rangle$) as $U^{\otimes n}|0\rangle^{\otimes n} = U_n|0\rangle^{\otimes n}$. Gates that can be written in this way are known as separable gates. For U operating on two qubits, U is separable if $U = U_1 \otimes U_2$, and can be visualised as:

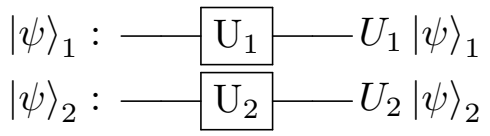


Figure 3.5: Gates U_1 and U_2 applied to qubits $|\psi\rangle_1$ and $|\psi\rangle_2$

Which is equivalent to:

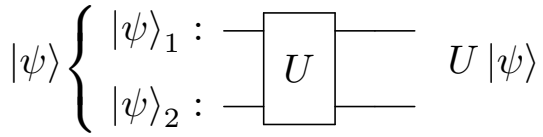


Figure 3.6: The gates U_1 and U_2 expressed as a single separable gate U, where $|\psi\rangle = |\psi\rangle_1 \otimes |\psi\rangle_2$ (or just $|\psi\rangle_1 |\psi\rangle_2$). Moreover, the resulting state $U|\psi\rangle$, in turn, is also separable as $(U_1|\psi\rangle_1) \otimes (U_2|\psi\rangle_2)$

3.3.2 Non-separable gates

The first sample of a two-qubit gate that is not divisible is the so-called *Controlled NOT*. The first qubit of the input of this operation is named the control qubit, while the second one is the target qubit, and for the computational basis it is portrayed as:

$$C_{NOT}|ab\rangle = |a \;\; (b \oplus a)\rangle \tag{3.11}$$

Explicity:

$$\begin{aligned} C_{NOT}|00\rangle &= |00\rangle \\ C_{NOT}|01\rangle &= |01\rangle \\ C_{NOT}|10\rangle &= |11\rangle \\ C_{NOT}|11\rangle &= |10\rangle \end{aligned}$$

The matrix form of this gate can be obtained as pursues.

$$\begin{aligned} C_{NOT} &= \begin{pmatrix} \langle 00|C_{NOT}|00\rangle & \langle 00|C_{NOT}|01\rangle & \langle 00|C_{NOT}|10\rangle & \langle 00|C_{NOT}|11\rangle \\ \langle 01|C_{NOT}|00\rangle & \langle 01|C_{NOT}|01\rangle & \langle 01|C_{NOT}|10\rangle & \langle 01|C_{NOT}|11\rangle \\ \langle 10|C_{NOT}|00\rangle & \langle 10|C_{NOT}|01\rangle & \langle 10|C_{NOT}|10\rangle & \langle 10|C_{NOT}|11\rangle \\ \langle 11|C_{NOT}|00\rangle & \langle 11|C_{NOT}|01\rangle & \langle 11|C_{NOT}|10\rangle & \langle 11|C_{NOT}|11\rangle \end{pmatrix} \\ &= \begin{pmatrix} \langle 00|00\rangle & \langle 00|01\rangle & \langle 00|11\rangle & \langle 00|10\rangle \\ \langle 01|00\rangle & \langle 01|01\rangle & \langle 01|11\rangle & \langle 01|10\rangle \\ \langle 10|00\rangle & \langle 10|01\rangle & \langle 10|11\rangle & \langle 10|10\rangle \\ \langle 11|00\rangle & \langle 11|01\rangle & \langle 11|11\rangle & \langle 11|10\rangle \end{pmatrix} \end{aligned} \tag{3.12}$$

$$C_{NOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.13}$$

And the quantum circuit that illustrates it is:

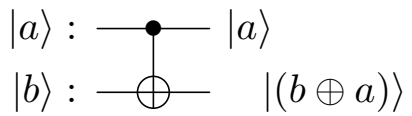


Figure 3.7: Controlled NOT gate

As will be seen later, just as the Hadamard gate can generate superposition, the C_{NOT} gate can create entanglement between two qubits.

On the other hand, this operation can be seen as a "copier" of a state of the control qubit, if we set the target qubit to be $|0\rangle$. If we have a qubit $a \in \{0, 1\}$ and another one initialized to $|0\rangle$, operation $C_{NOT} |a\rangle |0\rangle = |a\rangle |0 + a\rangle = |a\rangle |a\rangle$ copies the state a to the second qubit. Something worth noting is that this feature only works for the states in the computational basis since no quantum gate is capable of replicating an arbitrary quantum state. This result is easily demonstrable by contradiction and it is known as the No-Cloning Theorem, which can be written as:

$$\nexists U : U |\psi\rangle |0\rangle = |\psi\rangle |\psi\rangle \tag{3.14}$$

With $\psi \notin \{0, 1\}$.

It can be noted that the C_{NOT} gate is very similar to the X gate, so much so that the former can be written in terms of the latter.

$$C_{NOT} = |0\rangle \langle 0| \otimes \mathbb{1} + |1\rangle \langle 1| \otimes X$$

Where $|0\rangle \langle 0|$ and $|1\rangle \langle 1|$ are the basis' projectors.

This example is helpful to generalize the concept of quantum controlled gates, which are gates that act on a target qubit only if the control qubit (or qubits) are activated. They can be described mathematically as:

$$C_U = |0\rangle \langle 0| \otimes \mathbb{1} + |1\rangle \langle 1| \otimes U$$

If $U = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$, the matrix form of C_U will be:

$$C_U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & \gamma & \delta \end{pmatrix} \tag{3.15}$$

And can be displayed as:

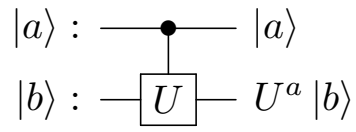


Figure 3.8: Controlled U gate. Where $U^0 = \mathbb{1}$ and $U^1 = U$

For control qubits in superposition, the controlled gates act on *all states* of the superposition, taking advantage of the linearity property of quantum operators. For instance, let's see the circuit below (Fig. 3.9).

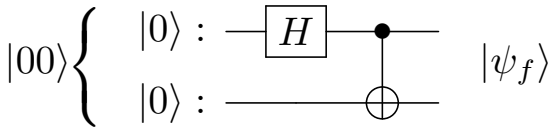


Figure 3.9: Superposition on a controlled gate

$|\psi_f\rangle$ can be obtained as follows:

$$\begin{aligned} |00\rangle \to (H\,|0\rangle)\,|0\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}}\,|0\rangle \\ \to C_{NOT}^{1\to 2}\left(\frac{|00\rangle + |10\rangle}{\sqrt{2}}\right) &= \frac{C_{NOT}\,|00\rangle + C_{NOT}\,|10\rangle}{\sqrt{2}} \\ &= \frac{|00\rangle + |11\rangle}{\sqrt{2}} = |\psi_f\rangle \end{aligned} \tag{3.16}$$

Where the super-index $1 \to 2$ denotes the control and target qubits. It is noteworthy to mention that in this example it is no longer possible to express $|\psi_f\rangle$ as the product of two other states, as we had in the beginning, because the controlled gates are responsible for creating entanglement between the qubits. Thus, we can make a distinction between two classes of quantum states:

1. Separable states:
$$|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle$$

2. Non-separable or entangled states:
$$|\psi\rangle \neq |\phi_1\rangle \otimes |\phi_2\rangle$$
With $|\phi_1\rangle$ and $|\phi_2\rangle$ any others quantum states.

The circuit depicted in Fig. 3.9 is the simplest example of how to generate entangled states from a separable basis. The states resulting from this circuit (e.g., $|\psi_f\rangle$ in Eq. 3.3.2) are given their own name: the Bell States:

$$|\beta_{00}\rangle = |\phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

$$|\beta_{10}\rangle = |\phi^-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

$$|\beta_{01}\rangle = |\psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$

$$|\beta_{11}\rangle = \left|\psi^-\right\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

Where the subscripts of β denote the states from which it comes.

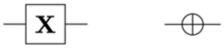
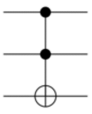
Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
π/8 (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Figure 3.10: Some common used quantum gates and their matricial representation. Image taken from [15].

3.3.3 Quantum parallelism

We can generalize the definition of gate C_{NOT}, presented in Eq. 3.11, in a new gate, let's say U_f, so that, instead of adding the control qubit modulo 2 to the target qubit, we add a function of the former:

$$U_f\,|ab\rangle = |a(b \oplus f(a))\rangle$$

Where $f : \{0,1\} \to \{0,1\}$, and the quantum circuit associated with U_f is usually drawn as

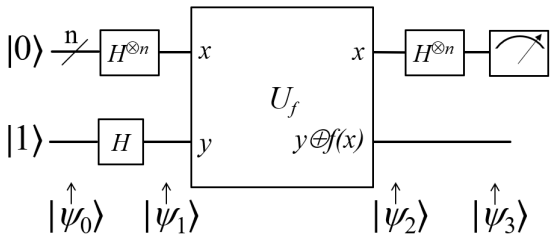
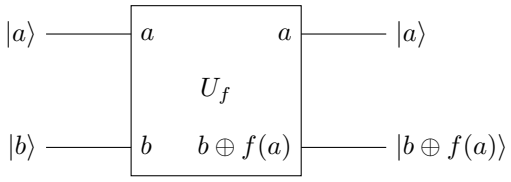


Figure 3.11: Deutsch–Jozsa algorithm circuit. Image taken from [12].



With $a, b \in \{0, 1\}$, it turns out that this circuit allows us to simultaneously evaluate two values of the function f, if the control qubit is in a superposition of the basis states. For instance, let's see what happens if we have the initial state $|00\rangle$:

$$U_f(H \otimes \mathbb{1}) |00\rangle = U_f \left(\frac{|00\rangle + |10\rangle}{\sqrt{2}} \right)$$

$$= \left(\frac{U_f |00\rangle + U_f |10\rangle}{\sqrt{2}} \right)$$

$$= \frac{|0\rangle |f(0)\rangle + |1\rangle |f(1)\rangle}{\sqrt{2}} \tag{3.17}$$

The quantum state 3.17 contains the outputs from both $f(0)$ and $f(1)$, which is as if the quantum computer has evaluated the function for two arguments with a single operation. This feature is known as quantum parallelism. Moreover, this process can be extended for a larger number of qubits by applying n Hadamard gates to n qubits (operation known as the Hadamard transform).

Although our quantum algorithm has already computed $f(0)$ and $f(1)$, we can not know their values because Eq. 3.17 is a superposed state. It is necessary to measure it to obtain information about f, which causes the collapse of the superposition and gives us only either $|0\rangle |f(0)\rangle$ or $|1\rangle |f(1)\rangle$; the same as if we had evaluated the function on a classical computer.

Quantum parallelism by itself does not give quantum computers any advantage over their classical counterparts. Nevertheless, this feature can be used to build more sophisticated algorithms that work with states in superposition but whose final output is well defined in a single eigenstate, causing their measurement to have 100% reliability.

An example of these algorithms is the Deutsch–Jozsa algorithm, which uses quantum parallelism to determine whether a function $f : \{0, 1\}^n \to \{0, 1\}$ is alternating ($f(0) \neq f(1)$) or constant ($f(0) = f(1)$). We will not go into details about this algorithm, but its associated circuit can be seen in Fig. 3.11.

3.3.4 Universal quantum gates

A set of quantum gates that allows creating any other quantum gate using them is known as a set of universal quantum gates.

Without going into an exhaustive mathematical development, it can be said that a set of universal quantum gates must meet some requirements. Some of them are: to generate superposed states, through a Hadamard gate for example; to create entanglement, something that can be achieved with a C_{not} gate; and to produce quantum states with complex amplitudes, as do the T and S gates [14].

Two sets of universal quantum gates are:

1. $\{C_{NOT}$, and all one-qubit gates$\}$
2. {Toffoli Gate, H, $S\}$

Where the Toffoli gate (or CC_{NOT} gate) is the generalization of the C_{NOT} gate for three qubits:

$$CC_{NOT}\left|a\right\rangle\left|b\right\rangle\left|c\right\rangle = \left|a\right\rangle\left|b\right\rangle\left|c \oplus ab\right\rangle \tag{3.18}$$

For example, using the first set, the inverse C_{NOT} ($C^{2\to1}_{NOT}\left|ab\right\rangle = \left|a \oplus b\right\rangle\left|b\right\rangle$) gate is tantamount to applying one C_{NOT} gate and four H gates as follows:

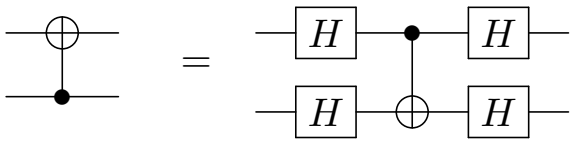


Figure 3.12: Inverse C_{NOT} gate

Equivalence between two sets of quantum gates can be demonstrated either by performing matrix multiplication of each of them and observing that the resulting matrix is the same (tedious), or by evaluating their effect on the complete basis on which they are operating. For out inverse C_{NOT} example:

$$H^{\oplus2}\left|ab\right\rangle = \frac{\left|00\right\rangle + (-1)^b\left|01\right\rangle + (-1)^a\left|10\right\rangle + (-1)^{a+b}\left|11\right\rangle}{2}$$

$$\to C_{NOT}H^{\oplus2}\left|ab\right\rangle = \frac{\left|00\right\rangle + (-1)^b\left|01\right\rangle + (-1)^a\left|11\right\rangle + (-1)^{a+b}\left|10\right\rangle}{2}$$

$$\to H^{\oplus2}C_{NOT}H^{\oplus2}\left|ab\right\rangle = \frac{H\left|0\right\rangle}{\sqrt{2}}\left(H\frac{\left|0\right\rangle + (-1)^b\left|1\right\rangle}{\sqrt{2}}\right) + \frac{H\left|1\right\rangle}{\sqrt{2}}\left(H\frac{\left|0\right\rangle + (-1)^b\left|1\right\rangle}{\sqrt{2}}\right)(-1)^{a+b}$$

$$\to H^{\oplus2}C_{NOT}H^{\oplus2}\left|ab\right\rangle = H\frac{\left|0\right\rangle + (-1)^{a+b}\left|1\right\rangle}{\sqrt{2}}\left(H\frac{\left|0\right\rangle + (-1)^b\left|1\right\rangle}{\sqrt{2}}\right)$$

$$\to H^{\oplus2}C_{NOT}H^{\oplus2}\left|ab\right\rangle = \left|a \oplus b\right\rangle\left|b\right\rangle = C^{2\to1}_{NOT}\left|ab\right\rangle$$

What is illustrated on Fig. 3.12.

3.4 Quantum Fourier Transform

3.4.1 Discrete Fourier Transform

The Fourier Transform decomposes continuous functions from the time domain to the frequency domain. However, when computing or working with real data, it is necessary to approximate the Fourier transform on discrete vectors of data [16]. One way to do this is to consider $f(x)$, rather than as a smooth continuous function of x, as a vector $\vec{x}$ with N entries, where each of them is a sample from our data set.

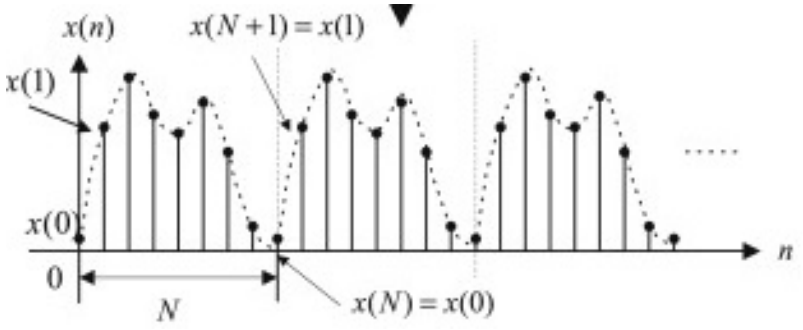


Figure 3.13: Discretization of a function for the DFT.

Thus, having a vector $\vec{x} = \begin{pmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ x_{N-1} \end{pmatrix}$, its discrete Fourier transform is defined as:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{i\frac{2\pi jk}{N}} \tag{3.19}$$

Or, if we set $F_{kj} = \frac{e^{i\frac{2\pi jk}{N}}}{\sqrt{N}}$:

$$y_k = \sum_{j=0}^{N-1} F_{kj} x_j \tag{3.20}$$

Or, in its matrix form:

$$\vec{y} = F\vec{x} \tag{3.21}$$

Where F is a unitary matrix.

Quantum Fourier Transform

It has been shown that the quantum Fourier transform offers some advantages over its classical equivalent, mainly that it requires fewer stages/gateways to complete the same result.If we have an arbitrary state $|\psi\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$, where N is the number of basis states, the QFT $|\psi\rangle$ is set as:

$$|\psi\rangle = \sum_{j=0}^{N-1} x_j |j\rangle \xrightarrow{F} |\phi\rangle = \sum_{k=0}^{N-1} y_k |k\rangle \tag{3.22}$$

Where y_k is given by Eq. 3.19.

Or, put another way:

$$|\psi\rangle = \sum_{j=0}^{N-1} x_j |j\rangle \xrightarrow{F} |\phi\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} x_j e^{i\frac{2\pi jk}{N}} |k\rangle \tag{3.23}$$

This transformation acting on the basis states leads us to:

$$|j\rangle \xrightarrow{F} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{i\frac{2\pi jk}{N}} |k\rangle \tag{3.24}$$

So, for our two-level computational basis ($N = 2$ and $j \in \{0, 1\}$):

$$|0\rangle \xrightarrow{F} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad |0\rangle \xrightarrow{F} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{3.25}$$

This result is not accidental, since it is easy to see that the action of the QFT on $|\psi\rangle$ is the equivalent of implementing a F operator on that state. F can be computed using the same expression as for the DFT. Also, it will be N dimension, where $N = 2^n$ (for our two energy levels systems) and n is the total number of qubits.

$$F_{kj} = \frac{e^{i\frac{2\pi jk}{N}}}{\sqrt{N}}$$

So, for one qubit ($N = 2$), the QFT gate will look like $F_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = H$, which explains the results in Eq. 3.25.

The Quantum Fourier transform transforms a state from the computational basis to the Fourier basis. That is, if in the computational basis numbers can be stored in binary notation using the basis states $|0\rangle$ and $|1\rangle$, in the Fourier basis they are stored using rotations around the z-axis (phase $e^{i\frac{2\pi jk}{N}}$ in 3.23).

With this in mind, let's see what QFT would look like on a computationalbasis $|j\rangle = |j_1 j_2 ... j_n\rangle$ for an arbitrary number of qubits ($n \neq 1$), where j_h are the digits of k in binary notation (not a product).

$$F|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{i\frac{2\pi jk}{2^n}} |k\rangle$$

If k is also expressed in binary notation $k = k_1 k_2 ... k_n$, then $k/2^n = \sum_{h=1}^{n} k_h/2^h$. Therefore, the above equation will look like:

$$F|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i(\sum_{h=1}^{n} k_h/2^h)j} |k_1 ... k_n\rangle$$

$$= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \prod_{h=1}^{n} e^{2\pi i \frac{jk_h}{2^h}} |k_1 ... k_n\rangle$$

Furthermore, we can expand $\sum_k^{N-1} = \sum_{k_1=0}^{1} k_1 \sum_{k_2=0}^{1} k_2 ... \sum_{k_n=0}^{1} k_n$ in our binary notation, which leads us to an expression on terms of $|0\rangle$ and $|1\rangle$:

$$F\,|j\rangle = \frac{1}{\sqrt{N}} \bigotimes_{h=1}^{n} \left(|0\rangle + e^{2\pi i \frac{j}{2^h}}\,|1\rangle\right) \tag{3.26}$$

$$= \frac{1}{\sqrt{N}} \left(|0\rangle + e^{\frac{2\pi i}{2}j}\,|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2}j}\,|1\rangle\right) \otimes ... \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}j}\,|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n}j}\,|1\rangle\right)$$

From Eq. 3.26 we can see, for example, that the state $|6\rangle = |0110\rangle$ on the Bloch Sphere looks like in Fig. 3.14.

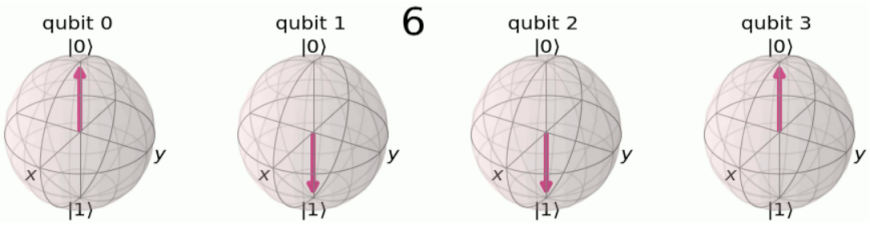


Figure 3.14: Representation of $|6\rangle$ with four qubits. Image taken from [17].

Encoding $\left|\tilde{6}\right\rangle$ on the Fourier basis with four qubits is:

$$F\,|6\rangle = \frac{1}{4} \left(|0\rangle + e^{2\pi i \frac{6}{2}}\,|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i \frac{6}{4}}\,|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i \frac{6}{8}}\,|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i \frac{6}{16}}\,|1\rangle\right)$$

This means that the qubit 3 (the most significant qubit) does three full turns from its initial position (state $|+\rangle$). Qubit 2 does $\frac{3}{2}$ fulls turns; qubit 1 does $\frac{3}{4}$; and qubit 0, $\frac{3}{8}$. This can be visualized in Fig. 3.15.

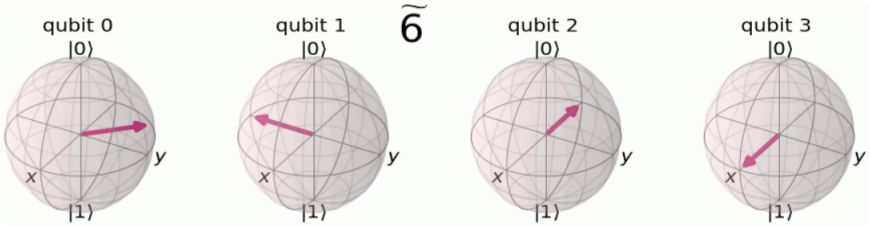


Figure 3.15: Representation of $\left|\tilde{6}\right\rangle$ with four qubits. Image taken from [17].

If we consider a unitary transformation R_k as

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$$

with a little observation in equation 3, we can notice that the quantum circuit that performs a QFT for a number n of qubits on a $|j\rangle = |j_1 j_2 ... j_n\rangle$ is:

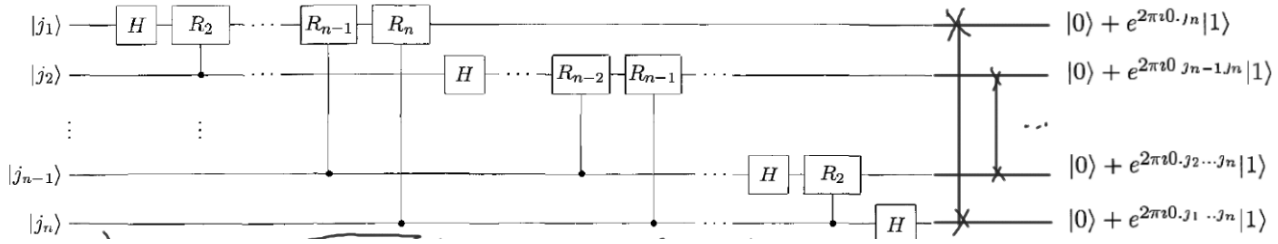


Figure 3.16: Image partially taken from [12].

For $n = 3$, Fig. 3.16 becomes:

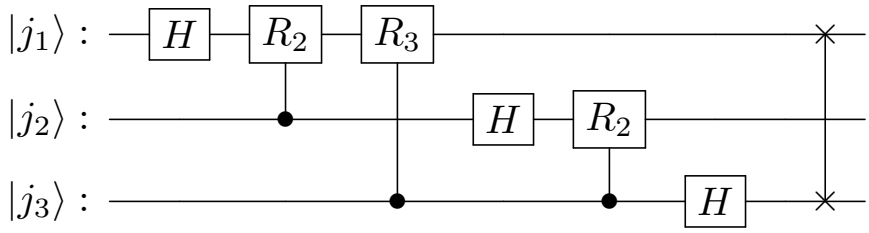


Figure 3.17: Quantum Fourier Transform circuit for three qubits.

The quantum Fourier transform is used in other more interesting algorithms, such as phase estimation and Shor's algorithm. These topics are not covered in this material, but a lot of literature can be found in references [12] [14] [17].

3.5 Density Operator and Schmidt decomposition

3.5.1 Mixed states and pure states

Throughout the discussion in this section, quantum systems have been presented as state vectors 3.1. However, not every two-level quantum system can be represented in this way (the Stern-Gerlach experiment with different bases, for example). A more general mathematical object is required: the density operator.

Consider an ensemble of N identical particles, but in different states not necessarily orthogonal to each other:

$$\{|\psi_i\rangle\}_{i=1}^{N} \qquad \langle\psi_i|\psi_j\rangle$$

For each particle, let's denote the expected value of an observable A as $\langle\psi_i| A |\psi_i\rangle = \langle A_i\rangle$. The expected value of all the ensemble is:

$$[A] = \sum_{i=1}^{N} \frac{1}{N} \langle A_i\rangle = \sum_{i=1}^{N} \frac{1}{N} \langle\psi_i| A |\psi_i\rangle \tag{3.27}$$

Inserting a completeness relation:

$$[A] = \sum_{i=1}^{N} \frac{1}{N} \langle A_i \rangle = \sum_{i=1}^{N} \sum_{j=1}^{m} \frac{1}{N} \langle \psi_i | A | j \rangle \langle j | \psi_i \rangle$$

Rearranging:

$$[A] = \sum_{i=1}^{N} \sum_{j=1}^{m} \frac{1}{N} \langle j | \psi_i \rangle \langle \psi_i | A | j \rangle = \sum_{j=1}^{m} \langle j | \left(\sum_{i=1}^{N} \frac{1}{N} | \psi_i \rangle \langle \psi_i | \right) A | j \rangle \tag{3.28}$$

Note that all the information regarding the particle ensemble is contained in the object in parentheses, which happens to be in the form of an operator. We will call this object the *density operator* ρ:

$$\rho = \sum_{i=1}^{N} \frac{1}{N} | \psi_i \rangle \langle \psi_i | \tag{3.29}$$

Thus, Eq. 3.28 can be written as:

$$[A] = \sum_{j=1}^{m} \langle j | \rho A | j \rangle$$

Which it turns out to be:

$$[A] = Tr\{\rho A\} \tag{3.30}$$

Where $Tr\{\rho A\}$ stands for the trace of ρA.

Furthermore, if we name p_i the probability of each state $|\psi_i\rangle$, the density operator of the whole quantum system can be stated as:

$$\rho = \sum_{i=1}^{N} p_i | \psi_i \rangle \langle \psi_i | \tag{3.31}$$

Which naturally fulfills that $\sum_{i=1}^{N} p_i = 1$.

With this new tool, we can distinguish between two classes of quantum systems: a system whose state $|\psi\rangle$is well defined is said to be in a *pure state*, and its density operator is $|\psi\rangle = |\psi\rangle \langle \psi|$. If the state can not be expressed in this way, it is said to be a *mixed state*, since it is made up of a mixture of several pure states, described by Eq. 3.31.

The trace of any density matrix, whether of a pure or mixed state, must always be 1. However, if we square the density matrix of a pure state we obtain: $\rho^2 = |\psi\rangle \langle \psi | \psi \rangle \langle \psi | = |\psi\rangle \langle \psi|$, which means that ρ is idempotent. This fact leads us to establish a criterion to distinguish quantum states:

1. If $Tr\{\rho^2\} = 1$, ρ describes a pure state.
2. If $Tr\{\rho^2\} < 1$, ρ describes a mixed state.

$P(\rho) = Tr\{\rho^2\} \leq Tr\{\rho\}$ is a measure known as *purity*, which, as the name implies, shows how pure a state is.

A density matrix must fulfill the following three properties:

1. Must preserve probability
$$Tr\{\rho\} = 1$$
Since:
$$Tr\{\rho\} = Tr\{\sum_i p_i \left|\psi_i\right\rangle \left\langle\psi_i\right|\} = \sum_i p_i Tr\{\left|\psi_i\right\rangle \left\langle\psi_i\right|\} = \sum_i p_i = 1$$

2. $\rho = \rho^\dagger$ (ρ is hermitian).
Since: $[A] = Tr\{\rho A\} \in \mathbb{R}$, as A is an observable.

3. All its eigenvalues are non-negative: $\rho \geq 0$.
Since:
$$\left\langle\varphi\right| \rho \left|\varphi\right\rangle = \sum_i p_i \left\langle\varphi|\psi_i\right\rangle \left\langle\psi_i|\varphi\right\rangle = \sum_i p_i |\left\langle\varphi|\psi_i\right\rangle|^2 \geq 0$$
Where $\left|\varphi\right\rangle$ is an arbitrary state.

With this properties in mind, as a quick note, a simple way to calculate purity without the need to compute ρ^2 is to remember that $Tr\{AB\} = \sum_{i=1}^{n}\sum_{j=0}^{n} A_{ij}B_{ji}$, with $A = B = \rho$.

The purity of a quantum system can be visualized on the Bloch sphere described in Fig. 3.18.

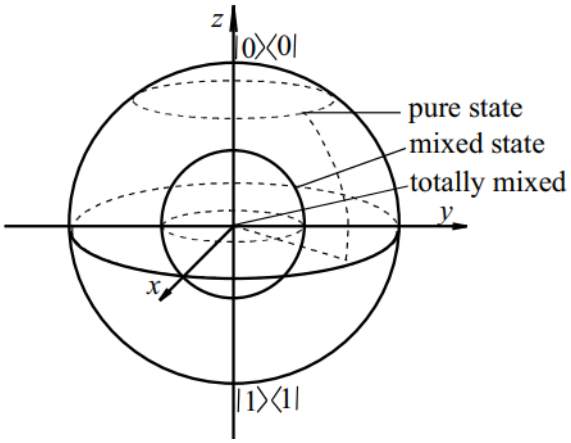


Figure 3.18: In this image, we can see that the north pole is located at $\left|0\right\rangle \left\langle 0\right|$ and the south pole at $\left|1\right\rangle \left\langle 1\right|$ [18].

3.5.2 The reduced density operator

The reduced density operator allows to study the subsystems, A and B, of a composite system, AB. This object is defined using the so-called partial trace over the joint density operator ρ_{AB} as follows:

$$\rho^A = Tr_B\{\rho_{AB}\} = Tr_B\{\left|a_1\right\rangle \left\langle a_2\right| \otimes \left|b_1\right\rangle \left\langle b_2\right|\} = \left|a_1\right\rangle \left\langle a_2\right| Tr\{\left|b_1\right\rangle \left\langle b_2\right|\}$$

Or:

$$\rho^B = Tr_A\{\rho_{AB}\} = Tr_A\{\left|a_1\right\rangle \left\langle a_2\right| \otimes \left|b_1\right\rangle \left\langle b_2\right|\} = Tr\{\left|a_1\right\rangle \left\langle a_2\right|\} \left|b_1\right\rangle \left\langle b_2\right|$$

Where $|a_1\rangle$ and $\langle a_2|$ are any two vectors in the state space of A, just as $|b_1\rangle$ and $|b_2\rangle$ are from B. The reduced density matrix ρ^A describes the reduced state of subsystem A without considering the degrees of freedom of subsystem B, but it does consider the effects of their coupling.

3.5.3 The Schmidt decomposition

The Schmidt decomposition is applied in bipartite systems (composed of two parts, A and B) and establish that a general state with the form:

$$|\Psi\rangle_{AB} = \sum_{j=1}^{N}\sum_{k=1}^{M} C_{jk}\,|j\rangle_A\,|k\rangle_B \qquad\qquad \langle j|j'\rangle = \delta_{j,j'}, \langle k|k'\rangle = \delta_{k,k'}$$

can be decomposed as:

$$|\Psi\rangle = \sum_{j=1}^{N} \sqrt{\lambda_j}\,|\psi_j\rangle_A\,|\phi_j\rangle_B \tag{3.32}$$

Where $|\psi_j\rangle_A$ and $|\phi_j\rangle_B$ are orthonormal states for system A and system B, respectively. λ_i are known as Schmidt coefficients, which must be non-negative.

The Schmidt decomposition is a consequence of the singular value decomposition seen in linear algebra ($C = V^\dagger \Lambda U$).

3.6 Quantum Machine Learning

Aimeur, Brassard and Gambs [19] distinguish four approaches to understand the term *quantum machine learning*. These are based on two parameters: whether the data sets are classical (C) or quantum (Q) and whether the device with which the data are to be analyzed is classical (C) or quantum (Q). How these four elements combine is illustrated in Fig. 3.19.

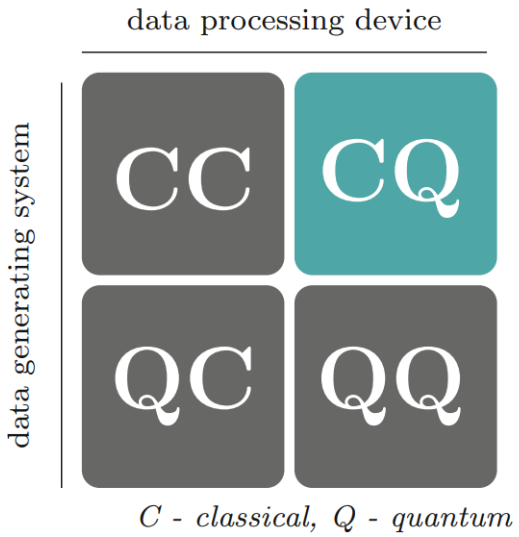


Figure 3.19: Quantum machine learning has several connotations. In the present thesis, we focus on CQ. Image taken from [20].

CC stands for the classical machine learning algorithms used to analyze, predict and generate data from huge classical data sets, using some of the procedures described in chapter 2. QC refers to processing quantum data using classical machine learning algorithms. This approach has proven to be very productive in qubit characterization, readout and control [21] [22] [23] [24].

QQ encompasses quantum computing algorithms that are used to describe quantum systems information. There are not yet many outstanding results in this area, but it will surely be, together with quantum simulations, one of the best ways we will have in the coming years to understand quantum phenomena.

Finally, the approach used in this work is CQ, where the aim is to use a quantum algorithm (hybrid quantum/classical, to be fair) to find insights on a classical dataset (energy depositions on a detector).

Chapter 4

High Energy Physics

4.1 Proton-Proton Collisions

Since the particles involved in accelerator experiments move at speeds close to the speed of light, their collisions are studied from relativistic dynamics. An important result of relativistic kinematics is that any quantity that can be expressed as the inner product of two four-vectors is Lorentz-invariant, or, in other words, can be calculated in any frame of reference obtaining the same result[25]. In collider physics, the position and momentum of a particle are represented by four-vectors. The former as (t, x, y, z) and the latter, which is named *four-momentum*, as (E, p_x, p_y, p_z), where p_x, p_y and p_z are the constituents of a particle´s momentum and E its energy.

The properties of the elementary particles outlined in the Standard Model (SM) are studied through high-energy collisions in particle accelerators, such as CERN's Large Hadron Collider (LHC). In these giant experiments, beams of particles are accelerated to near-light speeds and then collide at specific points where appropriate detectors are placed.

The main objective of the study of hadron-hadron collisions (particles formed by quarks held together by the strong nuclear force) is the search for new particles [26]. Specifically, the interaction between the partons (quarks or gluons) that make up a proton is called a proton-proton collision. These events are performed by accelerating two proton beams (obtained from the disintegration of hydrogen atoms) in opposite directions in an accelerator, obtaining energies E_1 and E_2. Thus, the energy of the center of mass of the collision can be calculated from the relation $E_{cm}^2 = (E_1 + E_2)^2$ [27]. The LHC's first run was performed at $E_{cm} = 7TeV$, the second one at $8TeV$ and recently it has been upgraded to produce 13.6TeV hadronic collisions. 13.6 trillion electronvolts (TeV) may sound a lot of energy, but actually, for perspective, one TeV is the equivalent of the energy produced by the flight of a mosquito. The point here is that a mosquito has trillions of trillions the mass of a proton. So it is no trivial matter to get a subatomic particle to reach the energy of 13 mosquitoes [28].

4.2 Particle Accelerators Coordinate System

A high-energy physics detector typically consists of at least three parts. Near the collision point, a tracker is placed, whose purpose is to determine the direction of the resulting particles by bending their trajectories with a magnetic field. Subsequently, there are two detectors: the electromagnetic and the hadronic calorimeters, ECAL and HCAL, respectively. They are devoted to measuring the particle energies. The former is where electrons and photons are detected, losing all their energy. After passing through the ECAL, Hadrons retain some of their momentum, which is fully

determined upon hitting the HCAL. Muons are the only charged particle that escapes these three layers, so their energy is measured by adding a fourth detector after HCAL. Furthermore, with this configuration, it is not possible to detect neutrinos at all, which contribute to the missing energy of the system.

To take advantage of the cylindrical geometry of hadron colliders, one uses a modified spherical coordinate system. The z-axis lies along the direction of the beams, placing the center of the reference frame at the point of collision (usually at the center of the detector). The ϕ angle is the azimuthal angle around the beam-axis. For a particle four-momentum $\vec{p} = (E, p_x, p_y, p_z)$, ϕ can be compute as:

$$\phi = tan^{-1}\left(\frac{p_y}{p_x}\right) \tag{4.1}$$

Where $-\pi \leq \phi \leq \pi$T. The magnitude of the so-called transverse momentum is given by:

$$p_t = \sqrt{p_x^2 + p_y^2} \tag{4.2}$$

This quantity is the component of momentum in the transverse plane $x - y$, and allows us to express $p_x = p_t cos\phi$ and $p_y = p_t sin\phi$. The transverse mass is another Lorentz invariant quantity and is defined as $m_t = \sqrt{m^2 + p_t^2}$.

Instead of the polar angle θ, another useful variable in accelerators physics is the rapidity y, which is, in terms of a particle momentum p:

$$y = \frac{1}{2} ln\left(\frac{E + p_z}{E - p_z}\right) \tag{4.3}$$

However, for particles with very high energy ($p >> m$), rapidity is hard to measure. Therefore, another quantity must be introduced, which can be derived from Eq. 4.3:

$$\begin{aligned} y &= \frac{1}{2} ln\left(\frac{E + p_z}{E - p_z}\right) = \frac{1}{2} \ln \frac{\sqrt{p^2 + m^2} + p\cos\theta}{\sqrt{p^2 + m^2} - p\cos\theta} && (4.4) \\ &= \frac{1}{2} \ln \frac{p + p\cos\theta}{p - p\cos\theta} = \ln \sqrt{\frac{1 + \cos\theta}{1 - \cos\theta}} && (4.5) \\ &= -\ln\tan\left(\frac{\theta}{2}\right) = \eta && (4.6) \end{aligned}$$

Where θ is the angle between the particle three-momentum and the positive direction of the z-axis (or the beam axis). η in Eq. 4.6 is termed as *pseudorapidity* of the particle. As can be seen from above, rapidity and pseudorapidity coincide for massless particles, but in general $|\eta| > |y|$.

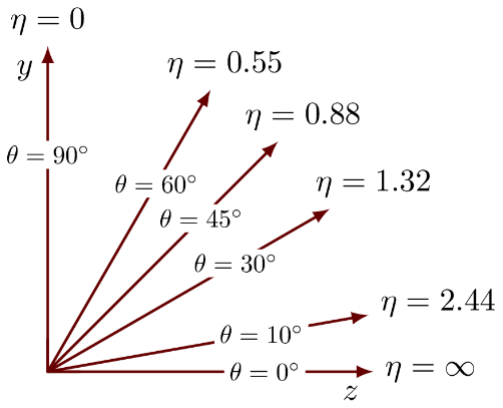


Figure 4.1: Pseudorapidity η for remarkable values of θ. Image taken from [29].

In conclusion, the position of a particle in a detector can be described by the azimuthal angle ϕ, which is measured in the anticlockwise direction from the x-y plane, and the pseudorapidity η (or rapidity y for massive particles). In the detectors at LHC, as the Compact Muon Solenoid (CMS), the x-axis is pointing to the center of the LHC ring, the y-axis is perpendicular to the plane of the ring and the z-axis, as mentioned above, lies along the protons beams. These coordinates are shown in Fig. 4.2.

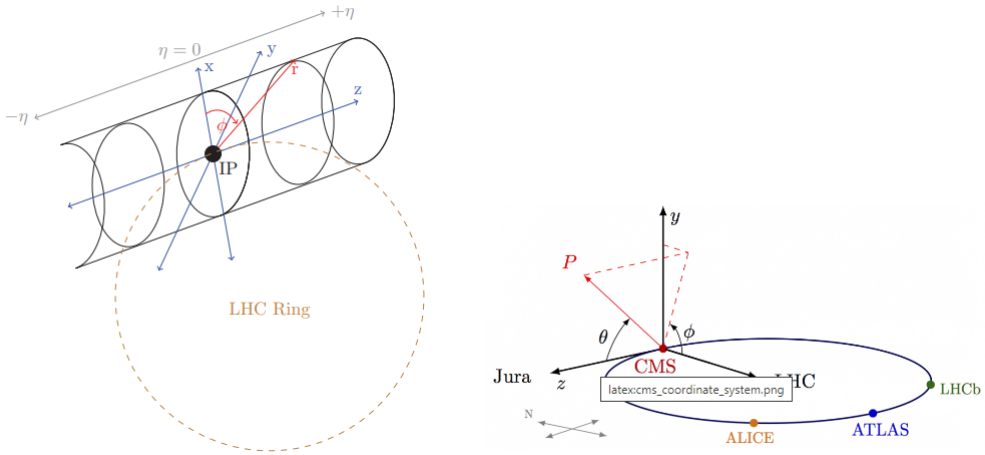


Figure 4.2: Schemes of the coordinate system used in LHC experiments. As can be seen in the left image, the pseudorapidity increases its (absolute) value as the particle goes away from the center of the beams interaction. Figures are taken from [25] and [29].

4.3 Jets

In high-energy hadronic collisions, quantum chromodynamics (QCD) radiation from colliding partons is very collimated to the incoming beams, a region where there are not detectors. It is therefore desirable to extend the rapidity range encompassed by the hadronic calorimeter specifically as much as possible. For example, the tracker in the ATLAS experiment is placed in a central region (with reference of the collision point) at $|\eta| \leq 2.5$ and the hadronic calorimeter extends up to $|\eta| \approx 5$

[30]. However, the cells of the HCAL do not have the same resolution as those of the tracker [26], making it difficult to spot individual particles. Also, only electrons, muons, photons and hadrons can be detected in a detector. This makes it necessary to create a new object that allows us to study in some way the particles that are produced by a collision before they decay or hadronize. Therefore, instead of looking for well-identified particles produced by the fragmentation of the proton beam, the objects that are measured in the HCAL are the so-called hadronic jets, which are collections (or streams) of energy depositions (or pseudo-particles) detected in the calorimeter. These hadron jets inherit the energy and momentum of the parton from which they originate [31].

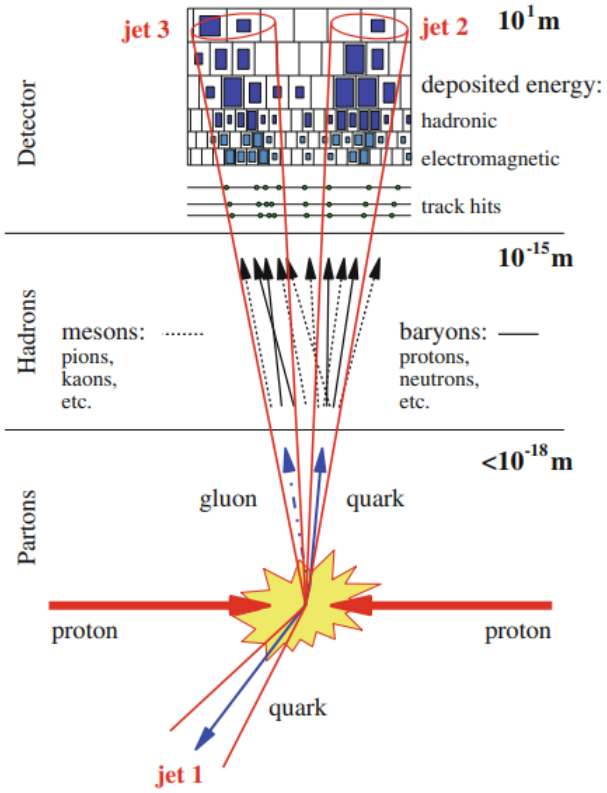


Figure 4.3: Illustration showing the hadronization of quarks and gluons and their subsequent sensing. Image taken from [31].

Although Fig. 4.3 is quite illustrative, it is far from what the actual classification of jets looks like. The events in Fig. 4.4 ilustrate electron-positron collisions at the ALEPH detector. How could we determine how many jets are in each event? How do we know which energy depositions belong to each jet? It is immediate to think that a mathematical tool for jet characterization is needed: the jet algorithms.

4.3.1 Jet Algorithms

Since 1977, when the first one was published by Sterman and Weinberg [33], a wide variety of jet algorithms have emerged [34], which can be classified into two classes:

1. Cone algorithms, which define conic geometric areas around the highest energy depositions in the calorimeters and a radius R, the so-called jet radius. These areas can be fixed cones [35], stable cones [36] or overlapping cones [37][38].

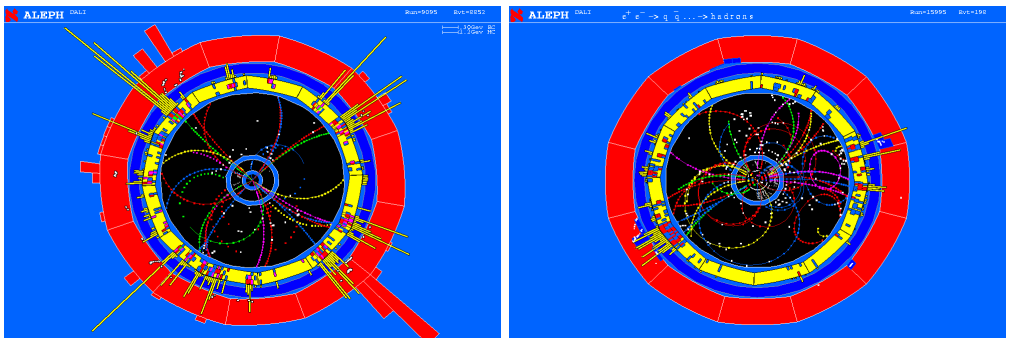


Figure 4.4: e^+e^- collisions in the Aleph experiment, a collaboration at the Large Electron-Positron Collider (LEP). Images taken from [32].

2. Sequential algorithms, which construct the jets by iteratively clustering the closest pairs of pseudo-particles. Some famous examples are the k_t algorithm [39] (Durham algorithm[40] for hadronic collisions), the Cambridge/Aachen algorithm [41], and the anti-k_t algorithm [42].

In this study we will not go into the mathematical description of each of these algorithms, since each of them is discussed in more detail in the references presented [1]. However, an illustration of how the clustering differs depending on the algorithm used can be seen in Fig. 4.5.

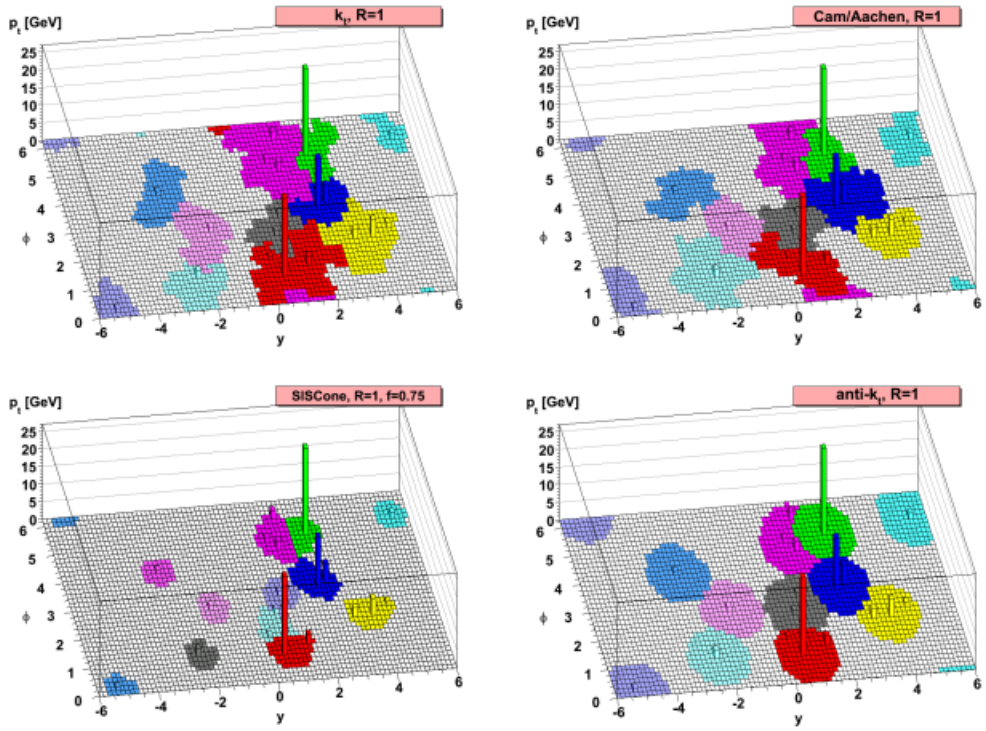


Figure 4.5: Comparison of the k_t, Cambridge/Aachen, SISCone, and anti-k_t algorithms. Cells with the same color are grouped in the same jet. Image taken from [44]

[1] A quick summary of each of them can also be found in [43].

4.4 Monte Carlo Simulations

The quantum nature of particle collisions introduces the concept of randomness in the behavior of the objects generated by these events. This is why Monte Carlo techniques are frequently used by experimental physicists to simulate these processes.

Some of the most commonly used software to generate Monte Carlos simulations are PYTHIA[45], MADGRAPH[46], Sherpa [47] and POWHEG [48]. In general, the process for the generation of events in these softwares is as follows. Firstly, for each colliding beam, only one proton's parton is considered to be involved in the jet scattering, and is probabilistically assigned a fraction of the total proton momentum via parton distribution functions (PDFs). Subsequently, by simulating the fragmentation, decay and hadronization of the initially produced particles, the final stable particles, each described by a momentum quadrivector, are obtained [49].

Chapter 5

Classifier algorithms

5.1 Data preprocessing

5.1.1 Data

The goal of this thesis is to create a simple straightforward algorithm which can classify signal and background jets based on calorimeter information. For this purpose, we use the public dataset that can be found in references [50] and [51]. The data consist of one million top signal jets and one million background jets produced using Pythia8 [52], with a center-of-mass energy of 14TeV. To simulate the collisions in the ATLAS experiment, the Delphes [53] software was used with the ATLAS detector card with a magnetic field of 2 T and a radius of 1.15m. The anti-k_t algorithm [42] was used to define the jets, with a jet radius $R = 0.8$ and requiring a minimum jet energy of 550GeV ($550GeV \leq p_{t,jet} \leq 650GeV$). In addition, each jet satisfies $|\eta| \leq 2$.

Each of the two million jets generated is made up of the leading 200 constituents (energy depositions) of the Delphes simulation, each characterized by a momentum quadrivector (E, p_x, p_y, p_z). Then a value of 0 or 1 is assigned whether the jet is a signal or a background, respectively. If there are not enough constituents for a jet, zeros are added to complete 200 four-momentum vectors.

	E_0	PX_0	PY_0	PZ_0	E_1	PX_1	PY_1	PZ_1	E_2	PX_2	...	E_199	PX_199	PY_199
0	218.364243	-172.341858	110.129105	-76.503624	153.661118	-111.320465	93.167969	-50.390713	76.708054	-56.523701	...	0.0	0.0	0.0
1	122.238762	26.738468	-91.613998	76.382225	121.227135	17.644758	-93.015450	75.715302	90.420105	21.377417	...	0.0	0.0	0.0
2	383.772308	-97.906456	79.640709	-362.426361	200.625992	-54.921326	37.994343	-189.184753	123.247223	-33.828953	...	0.0	0.0	0.0
3	132.492752	-77.763947	-87.322601	-62.304600	83.946594	-49.450481	-53.823605	-41.288010	28.072624	-19.964916	...	0.0	0.0	0.0
4	730.786987	-209.120010	-193.454315	-672.973877	225.477325	-75.363350	-66.226990	-201.926651	217.040192	-63.698189	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	107.598961	-86.493256	-37.625134	51.776459	83.556297	-69.174370	-35.723881	30.337543	69.006447	-53.915443	...	0.0	0.0	0.0
29996	40.411552	35.681664	15.854855	-10.418072	41.870693	28.990129	23.265043	-19.273426	33.280991	24.119259	...	0.0	0.0	0.0
29997	247.267456	18.934837	-119.225204	-215.796234	139.735626	9.669763	-66.933296	-122.280319	130.134888	9.838408	...	0.0	0.0	0.0
29998	230.381836	114.787895	91.257820	177.683823	177.950272	87.036385	67.774910	139.633560	139.115097	67.835197	...	0.0	0.0	0.0
29999	79.925987	10.867038	68.753288	-39.281746	66.523880	32.544441	39.659695	-42.348488	63.071362	12.241156	...	0.0	0.0	0.0

30000 rows × 806 columns

Figure 5.1: In this Dataframe we have 30,000 rows, which represent one jet each. The jets are made up of 200 quadrivectors $(E_i, p_{X,i}, p_{Y,i}, p_{Z,i})$, that is, 800 columns. In addition, each jet (row) is assigned a number 0 or 1, depending on whether it is a background or a signal (not displayed).

Due to limited computational resources, in the present study we only take 40000 jets to perform our analyses; 30000 for the training set, and the remaining for the test set. Fig. 5.1 shows the arrangement of the train data after being converted into a Pandas DatraFrame [54]. Also, we only consider the first 100 constituents of each jet, since only 200 of the 40000 rows had more than 100 non-zero quadrivectors.

We ensure that the amounts of jets corresponding to signals and those corresponding to background ware more or less the same, to avoid bias in subsequent Machine Learning tasks 5.2.

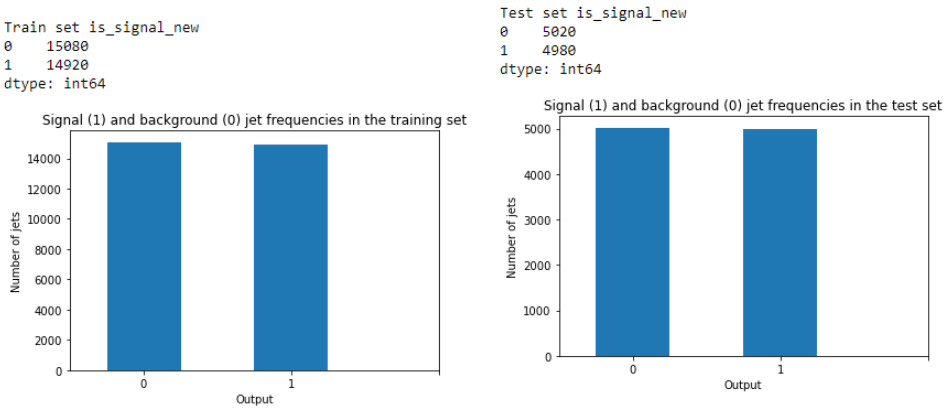


Figure 5.2: Signal and background frecuencies for training and test sets.

5.1.2 Data set construction

We can think of the data we have in Fig. 5.1 as a cloud of points in a three-dimensional space. The momentum components define the position of each pseudo-particle, further characterized by its energy (Fig. 5.3).

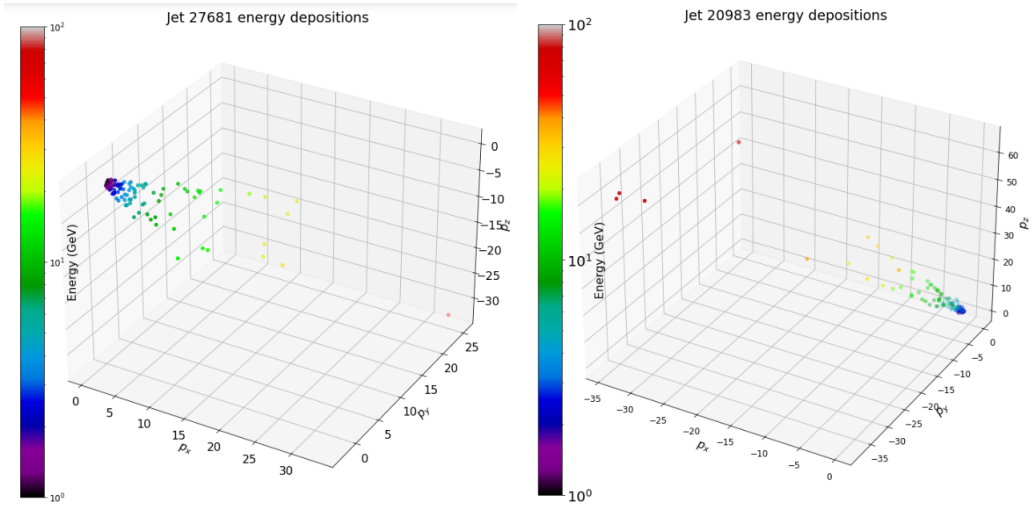


Figure 5.3: Jets can be represented as a cloud of points in a three-dimensional space, where each pseudo-particle defines a point in space and the energy is a feature of each one (color). The figure on the left illustrates a signal jet, and the one on the right a background.

Analyzing these data in this format can be somewhat complicated, although many resources can be found to begin working with them [55]. For the scope of this work, we manipulate these data sets to give them a more geometrical sense, since we know that they come —or are simulated — from an experiment with well-defined geometry (the ATLAS experiment).

This manipulation consists of using the equations discussed in Sec. 4.2 to compute the position of each energy deposition within the detector. That is, we used Eqs. 4.1 and 4.6 to obtain ϕ and η for each pseudo-particle, for every jet. To this end, it is worth remembering that $\theta = \arccos(pz/p)$ and $p = \sqrt{p_x^2 + p_y^2 + p_z^2}$. This done, we have new data where each jet is conformed by up to 100 three-vectors (E_i, η_i, ϕ_i), which can be visualized in Fig. 5.4. We will refer to this new data as the dataset $\eta - \phi$.

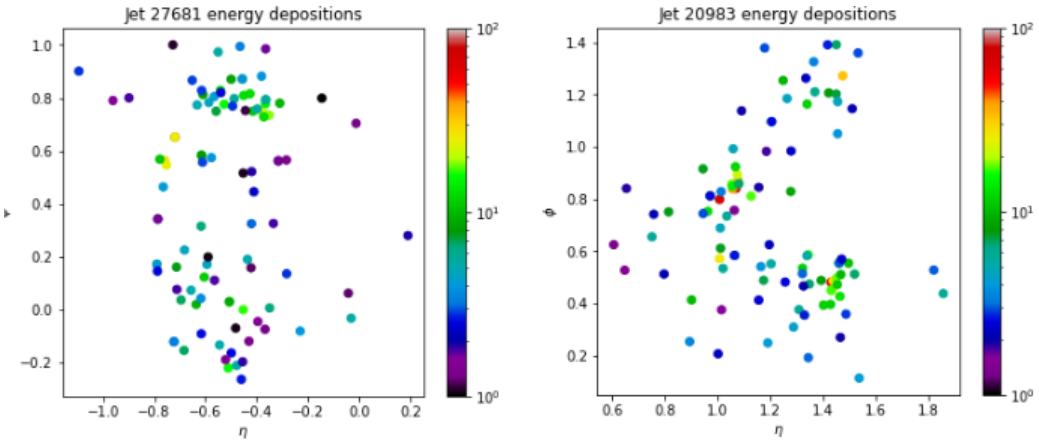


Figure 5.4: Distribution of jets 27681 (signal) and 20983 (background) in the ϕ-η plane. The color bar stands for the energy measured in GeV.

Graph Neural Networks approach

From this point on, two straightforward approaches can be taken in order to perform a Deep Learning analysis. The first and probably the most accurate is through Graph Neural Networks[56], where we want to represent each jet by means of a graph. A graph is a set of objects, called nodes, that are connected to each other by edges. Not necessarily all nodes are connected, but each of the edges is assigned a weight, depending on how strong the relationship between the nodes is. In addition, each node (or vertex) is assigned a feature vector, which differentiates it from other vertices and facilitates classification tasks. The tasks for which graph neural networks are used can be divided into two: node prediction, where the feature vector of an unknown node is predicted based on its connections to other nodes; and graph prediction, where an object is represented by a graph, to which a predicted label is to be assigned.

In this work, we do not delve further into graph neural networks, but briefly discuss how a dataset can be constructed under this approach, for comparison purposes. As mentioned above, each jet is mapped as a graph, where each energy deposition corresponds to a node, characterized by its energy. The edges correspond to the spatial relationship between each of the points; that is, how close or far apart two pseudo-particles are in the detector. Thus, all the nodes are connected to each other (see Fig. 5.5), and the edge weight between two nodes is given by $r_{ij} = \sqrt{(\eta_i - \eta_j)^2 + (\phi_i - \phi_j)^2}$.

In Python, a graph class can be created with the help of libraries such as Pytorch Geometric (PyG) [57] or Deep Graph Library (DGL) [58]. Then, a custom dataset can be created in Pytorch, so

that a graph neural network can iterate over each graph (as if it were any other object) and assign a label to it. Further readings of this implementation can be found in [59] [60] [61].

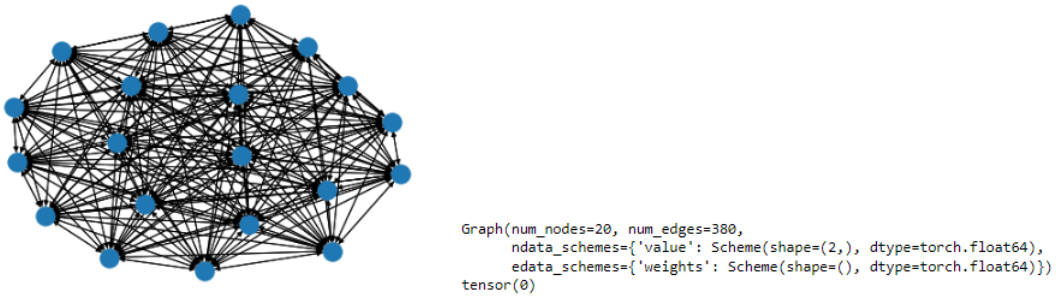


Figure 5.5: Graph generated via the DGL Python library, typifying a background jet with 20 constituents.

Calorimeter Images Approach

Although the method described above is probably one of the most accurate approaches to analyzing point-cloud data, it is quite expensive in terms of computational resources, and its implementation is more complicated. For this reason, we choose to use a simpler approach in the following, using image classification techniques.

The first thing to do is to generate *discrete* images from the $\eta - \phi$ dataset 5.4. We will refer to a discrete image as an ordered matrix of size nxn, where each entry corresponds to the value of a pixel in an nxn grid. In our case, we will scale a region of the $\eta - \phi$ plane of the detector into a $40x40$ pixel image, with $|\phi| \leq 100°$ and $|\eta| \leq 2$. These constrictions are considered because none of the jets in our data are outside of them.

This discretization procedure has a problem. Because in each jet there are energy depositions very close to each other, and our discrete images have finite pixels (only one energy value per pixel), not all points of the $\eta - \phi$ dataset can be displayed in the new data (which we will refer to as the calorimeter images dataset). However, there are —at least— two ways to alleviate this issue. One option is to increase the numbers of pixels, so that the cells are thinner and more points are included in the grid 5.6.

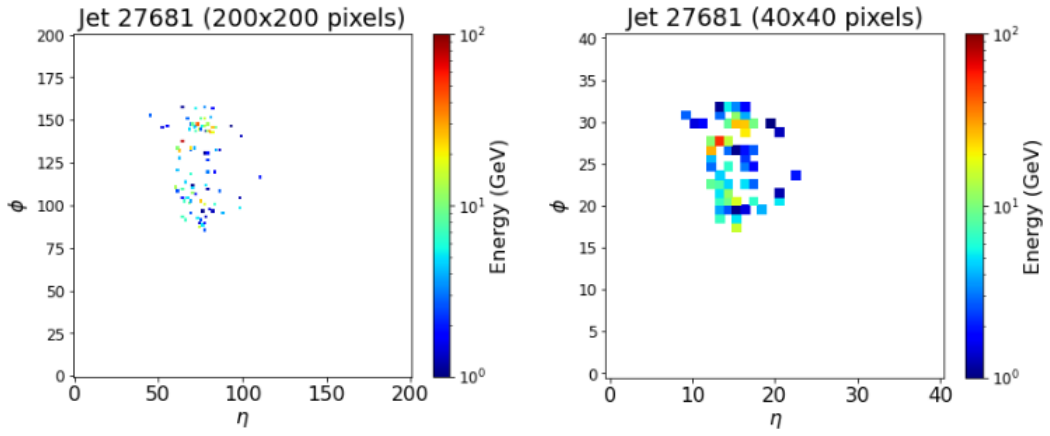


Figure 5.6: Comparison between calorimeter images with 200x200 and 40x40 pixels.

Although Fig. 5.6 (left) has better resolution and resembles more closely to Fig. 5.4 (left), in this work, for computational resource efficiency again, we opt for a second option: instead of choosing a single energy value for each pixel (of the most energetic particle), we group (sum) all the depositions of energies that fall in a single pixel (Fig. 5.7). Thus, each pixel in a $40x40$ calorimeter image depicts not only the energy of a single pseudo-particle, but also the energy of those nearby.

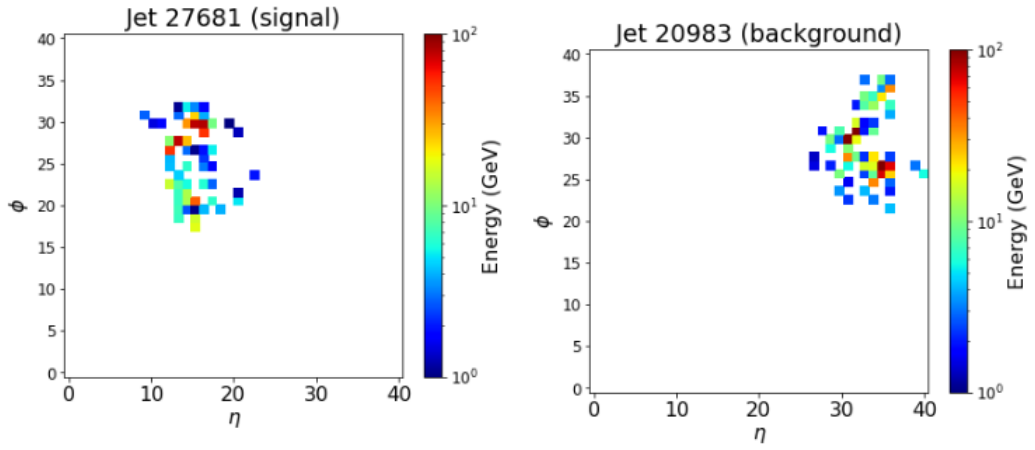


Figure 5.7: Discretized calorimeter images for a signal and a background. It is worth comparing the image on the left with the image on the right in Fig. 5.6 to note the difference in the amount of energy plotted. Each pixel on these images can be thought as a calorimeter tower of size $\Delta\eta \times \Delta\phi = 0.1 \times 5°$.

As mentioned in Sec. 2.3.6, convolutional neural networks are designed to recognize patterns, regardless of the spatial distribution of these within the images. For this reason, we do not consider it necessary to center or make any further arrangements to the calorimeter image dataset that will be used to train our CNN. Nonetheless, for visualization purposes only, we allow ourselves to center each jet on the images in the training data set based on the pixel with the highest energy. Then, we average the intensity of each pixel for the 30000 jets to obtain the averaged images in Fig. 5.8.

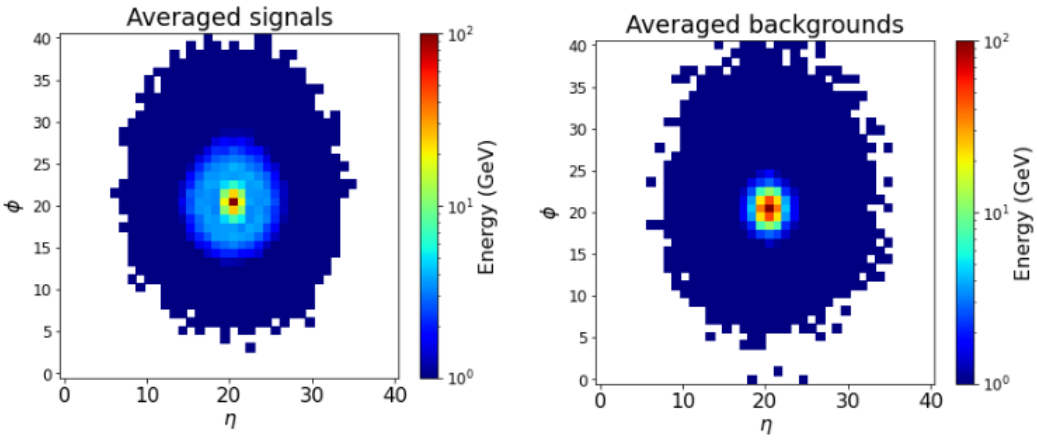


Figure 5.8: Signal and background images averaged over 30000 samples.

Interestingly, energy deposits in background jets appear to be more focal on a single point, whereas

signal jets have more than one focus of energy concentration. It is this type of behavior that we expect our neural network to be able to track in the next section.

5.2 Classic Convolutional Neural Network

Training our convolutional neural network is fairly straightforward once the above preprocessing has been done, but before doing so it is necessary to normalize (to unity) and randomize the data sets from the previous section.

5.2.1 Architecture

For the architecture of our neural network we chose the DeepTop network [62] with ZeroPadding, as shown in Fig. 5.9. The input layer is given by an image of 40×40 pixels in one channel. Firstly, we apply a convolutional layer of 8 feature maps with a 4×4 kernel followed by a second convolutional layer with the same characteristics. Next, we reduce our outputs by a 2×2 reduction factor using a max-pooling layer. Then we define another convolutional layer of 8 features maps and a 4×4 kernel. We flatten these feature maps and pass the outputs to three dense (fully connected) layers with 64 neurons each. The output layer consists of two neurons which compute the probability of the image of being a signal or background. We use the ReLU (Rectified Linear Unit) function as the activation function in all the layers, except on the last one, which uses the sigmoid function.

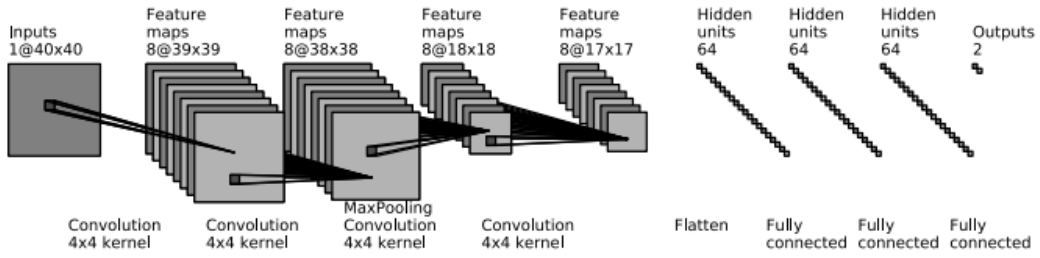


Figure 5.9: Architecture of our CNN based on the DeepTop CNN. Visualization generated via [63] and taken from [62]

The neural network is implemented in Python, using the library Keras [64] with TensorFlow [65] as backend. Information on the libraries versions used in this thesis can be found in Appendix B. We set the Adam optimizer with a learning rate of 0.001 and let the initial weights set by the Keras default settings.

Although in the original DeepTop network the mean-squared error (MSE) function is used as a loss function, we use the Binary Cross Entropy, as it is best suited for discrete classification tasks[5][66]. It computes the error (loss) between a model output $\hat{y}_i$ and the corresponding target value y_i by means of:

$$loss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i)) \tag{5.1}$$

The CNN is trained for 20 epochs and with a batch size of 10. For metrics, "accuracy" is chosen. To be precise, as mentioned in the Keras documentation [64], when the string *accuracy* is set as

a metric, the library automatically selects a metric depending on the loss function being used. Hence, since Binary Cross Entropy is our loss function, Keras selects Categorical Accuracy as our metric.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 37, 37, 8)         136

 conv2d_1 (Conv2D)           (None, 34, 34, 8)         1032

 max_pooling2d (MaxPooling2D  (None, 17, 17, 8)        0
 )

 conv2d_2 (Conv2D)           (None, 14, 14, 8)         1032

 flatten (Flatten)           (None, 1568)              0

 dense (Dense)               (None, 64)                100416

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 111,001
Trainable params: 111,001
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.10: Keras model summary

5.2.2 Results

With the above configuration, an accuracy of 88% was obtained on the training data set, while an accuracy of 84.7% was obtained on the test data. In Fig. 5.11 (left) we can visualize the decay of the loss function as the training progress. Also in Fig. 5.11 (right) the confusion matrix of the prediction of our model compared to the real labels of the test set is depicted.

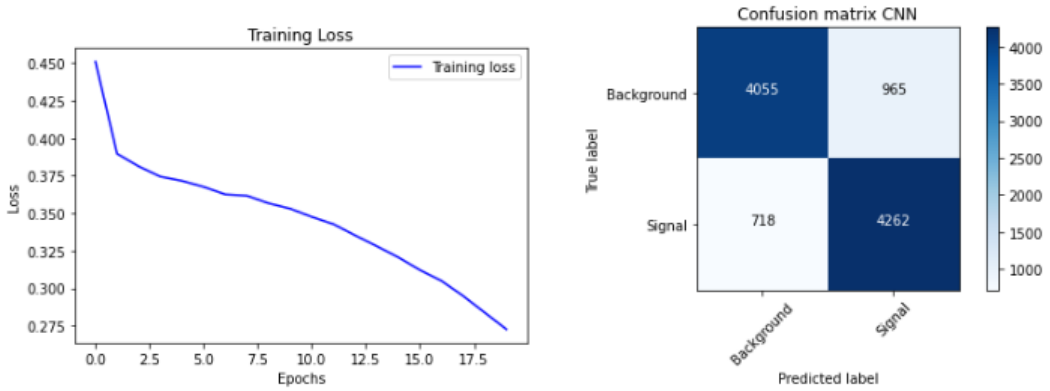


Figure 5.11: Loss function values over 20 training epochs and the confusion matrix from the predictions on the test set

Although as the epochs in training progress the cost function decreases, this accuracy is not reflected in the test set. This is why, after testing with different numbers of epochs, it was decided to set this number to 20 in order to avoid overfitting.

5.3 Hybrid Convolutional Neural Network

In this section we use IBM's Quantum Information Software Kit for Quantum Computation (Qiskit) [17] and Pytorch [67] libraries to build a classical-quantum algorithm that can predict the labels of our set of calorimeter images just as a neural network would. To emulate the quantum circuit we used the *Aer_simulator* class (backend) included in the Qiskit Aer package [68], which specializes in simulating noisy quantum circuits. The choice of Pytorch for this algorithm was not accidental, but is influenced by the features of this package that makes it easier to define custom non-sequential NN architectures by hand (within the forward and backward methods of the model class). This was of great importance to be able to include quantum layer, as well as its respective gradient for backpropagation.

5.3.1 Parametrized quantum circuit

The idea behind building a quantum-classical neural network is quite simple. Recalling that a quantum gate acting on a qubit represents a rotation of θ degrees, one can add a hidden quantum layer in a neural network architecture through a parameterized quantum circuit. This means that we can create a quantum circuit of a fixed number of qubits in a specified state (like $|0\rangle$) with n rotation gates $R(\theta_i)$ where the angle of rotation is a variable parameter θ_i. In other words, the input values of our quantum layer are the θ_i angles, which at same time are the output values of the previous hidden layer. Then, the σ_z measurement of each qubit (0 or 1) corresponds to the output values of our quantum layer, which can feed another classical or quantum hidden layer. A scheme of this configuration can be found in Fig. 5.12.

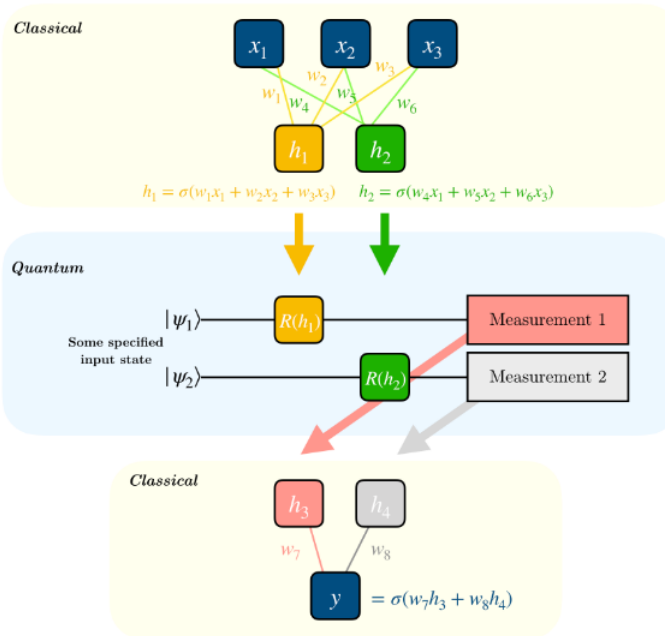


Figure 5.12: In this diagram, the parameters h_1 and h_2 are obtained from a dense neural network with three input and two output neurons. These angles parameterize two rotation gates $R(h_1)$ and $R(h_2)$ that are applied to two qubits, $|\psi_1\rangle$ and $|\psi_2\rangle$. Subsequently, a measurement is performed on each qubit and the resulting values are fed to a neural network with two input and one output neuron. Image taken from [17].

Let's explain it step by step. Consider a parameterized circuit of n qubits such that Hadamard gates are applied to all of them in order to create superposition, as shown in Fig. 5.13. Then a y-rotation is performed by a set of angles $\vec{\theta} = (\theta_1, \theta_2 ... \theta_{n-1}, \theta_n)$ on each qubit. Finally the state of each qubit is measured in the z-basis, which are stored on n auxiliary (classical) channels.

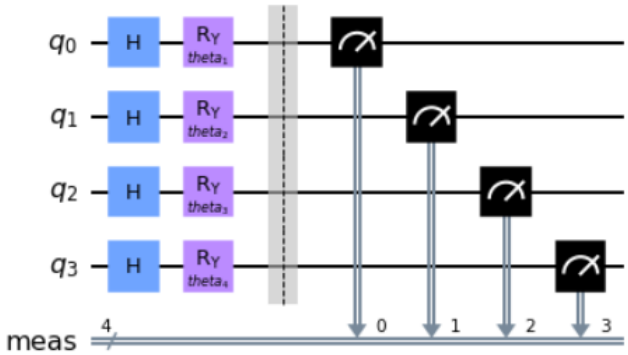


Figure 5.13: 4-qubit parameterized circuit.

As studied in Chapter 2, the measurement of a qubit results in one of two states, 0 or 1, only. Therefore, in the classical measurement channels of our circuit, we have a vector of n inputs, 0s or 1s. In other words, for a given vector of angles (inputs), after passing it to a quantum circuit (quantum layer), we obtain a vector with binary outputs.

We can repeat exactly the entire measurement process a finite number of times (shots), and record the number of times each qubit collapsed into one each of the bases. In the Qiskit framework, after having set up our quantum circuit, this procedure can be carried out as follows. First of all, we create a *job*, which is a module used to manage the 'runs' (a Aer_simulator function) [69] of a quantum object (*Qobj*) on a specific backend. In our case, the quantum object (our 'experiment' [70]) is an *assembly* [71] of the parameterized circuit, the number of shots and theta parameters $\vec{\theta}$. Next, we apply the *result()* and *get_counts()* methods consecutively to our *job*, which returns a dictionary for each qubit. In these, the keys are the states $|0\rangle$ and $|1\rangle$ and the values are the times that each key was measured. Then, by dividing each dictionary by the total number of shots, we will obtain the *a posteriori* probability of measuring each state in our experiment. Finally, we simplify our output by computing the expectation of these values, which is in the z-axis as:

$$\sigma_z = \sum_i z_i p(z_i) \tag{5.2}$$

Which turns out to be the probability of measuring the $|1\rangle$ state. We denote this quantity as $\sigma_{z,i}$, where i is the qubit to which this expectation pertains. In conclusion, after this procedure, a list of n expectations is obtained from a n-dimension $\vec{\theta}$ vector:

$$\begin{pmatrix} \theta_1 \\ \theta_2 \\ . \\ . \\ \theta_{n-1} \\ \theta_n \end{pmatrix} \longrightarrow \begin{matrix} parametrized \\ quantum \\ circuit(\vec{\theta}) \end{matrix} \longrightarrow \begin{pmatrix} \sigma_{z,1} \\ \sigma_{z,2} \\ . \\ . \\ \sigma_{z,n-1} \\ \sigma_{z,n} \end{pmatrix} \tag{5.3}$$

Where $p_{1,i} \in [0, 1]$.

The above process can be coded in a QuantumExperiment Class with two methods. The first, an __init__ that builds the quantum circuit from two parameters: the number of qubits and the backend (the simulator). The second, a 'run' method that receives the input $\vec{\theta}$ and the number of shots to execute the experiment and returns the expected values as a list or a *numpy.array*.

5.3.2 Backpropagation

To estimate the gradients of the parameters of a quantum circuit, the so-called parameter-shift rule is used [72]. In a nutshell, this method consists of considering our quantum circuit as a black box and taking the gradient of its parameters as the difference between its expectations when evaluating it at $\theta + s$ and $\theta - s$ (Eq. 5.4). s being a fixed shift value (sometimes named 'macroscopic shift'), which we set to $\frac{\pi}{2}$. Thus, these gradients can be used for a more complete backpropagation routine.

$$\Delta_{\vec{\theta}} \underset{Experiment}{Quantum} = \underset{Experiment}{Quantum}(\vec{\theta} + \vec{s}) - \underset{Experiment}{Quantum}(\vec{\theta} - \vec{s}) \quad , \quad \vec{s} = \begin{pmatrix} s \\ . \\ . \\ s \end{pmatrix} \tag{5.4}$$

The forward and backward passes necessary for the backpropagation can be integrated in a class (or more precisely, in a static method), where the forward passes call the QuantumExperiment class in order to obtain the expected values. These values are passed to the backward function to calculate the gradient as in Eq. 5.4, and returned for an optimizer such as Adam [73] or SGD [5] to minimize the cost function.

5.3.3 Architecture

The QuantumExperiment class and the backpropagation functions are nested in a *nn.Module* class called Hybrid layer, because it is specifically the classical-quantum layer that we will integrate in the architecture of our neural network. This layer, like any other layer, receives the outputs of the previous neural layer, as well as is configured with respect to the backend, quantum circuit and shift parameters. For the sake of simplicity, we only consider a quantum circuit with one qubit. So our quantum neuron layer can be reduced to Fig. 5.14.

Figure 5.14: A simple way to visualize our quantum neural layer.

After persistent trial and error, the architecture used in this neural network differed somewhat from the one used in the previous section. Our neural network was built in an *nn.module* class. There, the calorimeter images of dimensions 40×40 enter a convolutional layer of six feature maps and a kernel size of 5×5, to be subsequently reduced by a factor of two by a max-pooling layer. Again a convolutional layer with a 5×5 kernel is applied, but now with 16 feature maps, followed by max-pooling with the same factor as above. The generated outputs, of dimension $16 \times 7 \times 7$, are reshaped by a flattened layer so that they can now be processed by a linear layer of 784 neurons connected to another one of 64 nodes and a single output. This value parameterizes the one-qubit

quantum circuit 5.14 of the QuantumExperiment class. The activation of the convolutional layers is the ReLU function.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 6, 36, 36]             156
            Conv2d-2         [-1, 16, 14, 14]           2,416
            Linear-3                [-1, 64]         100,416
            Linear-4                 [-1, 1]              65
            Hybrid-5                 [-1, 1]               0
================================================================
Total params: 103,053
Trainable params: 103,053
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.08
Params size (MB): 0.39
Estimated Total Size (MB): 0.48
----------------------------------------------------------------
```

Figure 5.15: An overview of the neural network architecture (does not show the max-pooling layers).

5.3.4 Training task

After normalizing, the training and test datasets (numpy arrays) were converted into TensorDatasets, in order to create two DataLoader objects, where the data was shuffled. In the training loop, we use the Binary Cross Entropy again and the Adam optimizer with a default learning rate of 0.001. The model is trained only for six epochs.

5.3.5 Results

With the above configuration, we were able to achieve an accuracy of 86% in the training set and 83.8% in the test set. After six epochs, we noticed that the value of the loss function started to increase drastically, so it was decided to let the model train with only that number of iterations.

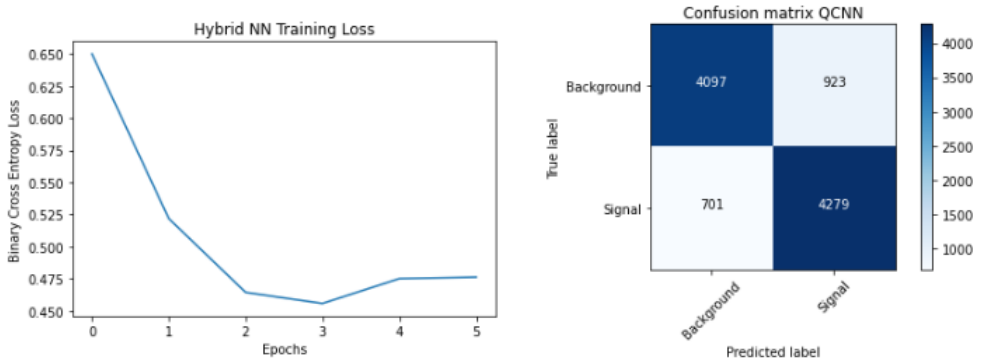


Figure 5.16: Binary Cross Entropy Loss over 6 training epochs, and the confusion matrix comparing the QCNN model versus the real labels.

5.4 Discussion and further considerations

Although neither of the two algorithms discussed above achieved an accuracy above 85% on the test sets, having reached nearly this standard is a good indicator that the preprocessing, proposed architectures, and model implementation are on track to create a more robust model.

In particular, there are several adaptations that can be further coupled to the models in seeking better accuracy. Regarding preprocessing, one could, in addition to increasing the sample size, search for a more optimal arrangement for the deployed jets. In this work, it was preferred not to train the models with images of jets centered with reference to their maximum energy point, mainly because by doing so some points on the edges were left out of the grids. However, it would be possible to agree on a "padding" criterion at the edges so that when the jet centroid is shifted, the image would enlarge its dimensions so that no energy deposition would be left out.

On the other hand, we could also add more channels to our input data, corresponding, for example, to measurements made by other detectors of the experiment, such as the ECal or the tracker. Thus, we would have calorimeter images of 3x40x40 dimensions that would better exploit the architecture of a convolutional neural network.

Regarding the hybrid classical-quantum model, in addition to trying the same changes mentioned above, more parameters could be added to our quantum layer. That is, extend the parameterized quantum circuit to more qubits, or even add more quantum layers between classical layers. Besides adding more qubits, one can change the design of the parameterized quantum circuit, adding more gates in each of the qubits in order to find a more precise configuration. Although, it is worth clarifying, this would require much more computational resources because, incidentally, the hybrid model, despite being simpler in terms of architecture, took much more computational time than the classical model.

It can also be noted that the parameter θ resulting from the next-to-last layer of our architecture is not scaled, for example, into a $[0, \pi]$, which make sense since we are representing rotations on the y-axis.

Finally, although our circuit took advantage of the superposition property of quantum systems (through Hadamard gates), no arrangement was made to create entanglement between qubits, one of the strongest features of quantum computing. Without entanglement, a quantum circuit is much easier to simulate on a classical computer, which would undermine the finding of any quantum advantage. Circuits with multi-qubit gates can be added to our architecture in search of any difference with the results presented here.

Chapter 6

Conclusions

The original idea of this thesis was to compare the performance of two Machine Learning Pipelines, one classical and one hybrid, in image jet classification tasks. Nevertheless, as the study progressed, we realized that this assignment was quite unfair. We were looking to compare a discipline that has been showing decades of success in a large number of fields, with hundreds of thousands of academics and professionals constantly improving their pipelines, with an area in quantum computation that until a few years ago, very few people conceived of. For this reason, the efforts in this work were directed toward making a guide that would allow a simple connection between the theory taught in quantum computing courses and the ML pipelines.

Although it is difficult to argue whether quantum computing will offer a better paradigm for Machine Learning, in recent years more and more people have become interested in the subject and have proposed very interesting approaches to solve some of the most important challenges of Quantum Machine Learning [74] [75] [76] [77].

Just as artificial intelligence has changed the way we relate to information, QML, coupled with other quantum technologies, could go deeper into the way we describe the world [78]. This is all the more encouraging if we embrace the epistemological premise that *not every physical phenomenon that occurs in the universe has to be mathematically describable.* Therefore, the challenge for many scientists in the coming decades will be to find new ways to explore questions about the world that the human intellect may not even be able to formulate. Machine learning and quantum technologies are good candidates to be valuable tools for this endeavor.

I hope that this work will serve as a basis for a more complex development in the subject among students who begin their study in quantum technologies, especially in Quantum Machine Learning, as I believe that there are many improvements that can be made to the algorithm presented here. Especially, I am very interested in the use of a parameterized quantum circuit in the creation of a hybrid graph neural network (QGNN).

One fact I would like to point out is that all this work was carried out on my personal laptop, with a rather limited processor and memory capacity. Although the execution times of the codes in this thesis were somewhat long (30 minutes for the classical training and about 2 hours for the hybrid), the success of this work was not determined by large computational resources, so practically any student with a computer and internet access can develop a similar algorithm. Moreover, the use of tools such as Google Colaboratory or access to the resources of the "Laboratorio Nacional de Supercómputo"[1] of our university (BUAP) can be of help for the development of much more complex algorithms.

[1] Laboratorio Nacional de Supercómputo del Sureste de México. https://lns.buap.mx/

Appendix A

Frameworks Versions and Scripts

All codes were developed in Python Jupyter Notebooks, which can be found in the GitHub repository of Ref. [79].

The versions of the libraries we worked with can be found in the following screenshot.

```
python: 3.9.12
qiskit: 0.21.1
numpy: 1.21.5
matplotlib: 3.5.1
pandas: 1.4.2
keras: 2.9.0
pytorch: 1.12.1+cpu
torchvision: 0.13.1+cpu
seaborn: 0.11.2
sklearn: 1.0.2
```

Bibliography

[1] Andreas Kaplan. *Artificial Intelligence, Business and Civilization: Our Fate Made in Machines*. New York: Routledge, 2022.

[2] Yan Ma et al. "Background Augmentation Generative Adversarial Networks (BAGANs): Effective Data Generation Based on GAN-Augmented 3D Synthesizing". In: *Symmetry* 10.12 (2018). ISSN: 2073-8994. DOI: 10.3390/sym10120734. URL: https://www.mdpi.com/2073-8994/10/12/734.

[3] Frank Rosenblatt. *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 1958.

[4] Hassan Hassan et al. "Assessment of Artificial Neural Network for Bathymetry Estimation Using High Resolution Satellite Imagery in Shallow Lakes: Case Study El Burrullus Lake". In: *International Water Technology Journal* 5 (Dec. 2015).

[5] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[6] Rob Picheta. "Los pandas gigantes ya no están en peligro de extinción, gracias a los esfuerzos de conservación, dice China". In: *CNN* (Dec. 2021). DOI: https://cnnespanol.cnn.com/2021/07/09/pandas-gigantes-ya-no-peligro-extincion-conservacion-china-trax/.

[7] Alexander Amini. *MIT Introduction to Deep Learning 6.S191: Lecture 3*. MIT Open CourseWare, 2020.

[8] Sandro Skansi. *Introduction to Deep Learning. From Logical Calculus to Artificial Intelligence*. Springer, 2018.

[9] Xiang Li et al. "Predicting the effective mechanical property of heterogeneous materials by image based modeling and deep learning". In: *Computer Methods in Applied Mechanics and Engineering* 347 (Apr. 2019). DOI: 10.1016/j.cma.2019.01.005.

[10] Vedant Kuma. "Convolutional Neural Networks". In: (Aug. 2020). DOI: https://towardsdatascience.com/convolutional-neural-networks-f62dd896a856.

[11] Jordi Torres. *Deep Learning: Introducción práctica con Keras*. 1st ed. Colección Watch This Space. Barcelona, Spain: Lulu Press, Inc, 2018.

[12] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011. ISBN: 9781107002173.

[13] Kevin Garapo, Mhlambululi Mafu, and Francesco Petruccione. "Intercept-resend attack on six-state quantum key distribution over collective-rotation noise channels". In: *Chinese Physics B* 25.7 (July 2016), p. 070303. DOI: 10.1088/1674-1056/25/7/070303. URL: https://doi.org/10.1088/1674-1056/25/7/070303.

[14] Thomas G. Wong. *Introduction to Classical and Quantum Computing*. Rooted Grove, 2022. ISBN: 979-8-9855931-0-5.

[15] Rxtreme. *Common quantum logic gates by name (including abbreviation), circuit form(s) and the corresponding unitary matrices.* URL: https://en.wikipedia.org/wiki/Quantum_logic_gate#/media/File:Quantum_Logic_Gates.png.

[16] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

[17] MD SAJID ANIS et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[18] Cong Shuang. "The analysis of two-level quantum system states and control in the Bloch ball". In: *2008 27th Chinese Control Conference*. 2008, pp. 618–622. DOI: 10.1109/CHICC.2008.4604907.

[19] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. "Machine Learning in a Quantum World". In: *Advances in Artificial Intelligence*. Ed. by Luc Lamontagne and Mario Marchand. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 431–442. ISBN: 978-3-540-34630-2.

[20] Francesco Petruccione Maria Schuld. *Supervised Learning with Quantum Computers*. Springer Cham, 2018. DOI: https://doi.org/10.1007/978-3-319-96424-9.

[21] Élie Genois et al. "Quantum-Tailored Machine-Learning Characterization of a Superconducting Qubit". In: *PRX Quantum* 2 (4 Dec. 2021), p. 040355. DOI: 10.1103/PRXQuantum.2.040355. URL: https://link.aps.org/doi/10.1103/PRXQuantum.2.040355.

[22] Muhammad Usman et al. "Framework for atomic-level characterisation of quantum computer arrays by machine learning". In: *npj Computational Materials* 6.1 (Mar. 2020), p. 19. ISSN: 2057-3960. DOI: 10.1038/s41524-020-0282-0. URL: https://doi.org/10.1038/s41524-020-0282-0.

[23] Easwar Magesan et al. "Machine Learning for Discriminating Quantum Measurement Trajectories and Improving Readout". In: *Phys. Rev. Lett.* 114 (20 May 2015), p. 200501. DOI: 10.1103/PhysRevLett.114.200501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.114.200501.

[24] Murphy Yuezhen Niu et al. "Universal quantum control through deep reinforcement learning". In: *npj Quantum Information* 5.1 (Apr. 2019), p. 33. ISSN: 2056-6387. DOI: 10.1038/s41534-019-0141-3. URL: https://doi.org/10.1038/s41534-019-0141-3.

[25] Deepak Kar. *Experimental Particle Physics*. 2053-2563. IOP Publishing, 2019. ISBN: 978-0-7503-2112-9. DOI: 10.1088/2053-2563/ab1be6. URL: https://dx.doi.org/10.1088/2053-2563/ab1be6.

[26] Andrea Banfi. *Hadronic Jets*. 2053-2571. Morgan and Claypool Publishers, 2016. ISBN: 978-1-6817-4073-7. DOI: 10.1088/978-1-6817-4073-7. URL: https://dx.doi.org/10.1088/978-1-6817-4073-7.

[27] Werner Herr and B Muratori. "Concept of luminosity". In: (2006). DOI: 10.5170/CERN-2006-002.361. URL: https://cds.cern.ch/record/941318.

[28] R. Carreras and G. Hentsch. *Quand l'énergie devient matière...: un coup d'oeil sur le monde des particules*. Laboratoire européen de physique des particules, 1986. URL: https://books.google.com.mx/books?id=7h1rnQEACAAJ.

[29] I. Neutelings. *CMS Wiki Pages, How to draw diagrams in LaTeX with TikZ*. URL: https://wiki.physik.uzh.ch/cms/latex:tikz.

[30] Coll ATLAS et al. *The simulation principle and performance of the ATLAS fast calorimeter simulation FastCaloSim*. Tech. rep. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-PHYS-PUB-2010-013. Geneva: CERN, Oct. 2010. URL: https://cds.cern.ch/record/1300517.

[31] Thomas Schoerner-Sadenius, ed. *The Large Hadron Collider: harvest of Run 1*. http://inspirehep.net/record/1373790?ln=en. Cham: Springer, 2015, 1 online resource (xxvi, 532 pages) : illustrations. ISBN: 9783319150017. DOI: 10.1007/978-3-319-15001-7. URL: https://bib-pubdb1.desy.de/record/220418.

[32] Hans Drevermann. *DALI: The Aleph Offline Event Display*. URL: https://aleph.web.cern.ch/aleph/dali/.

[33] George Sterman and Steven Weinberg. "Jets from Quantum Chromodynamics". In: *Phys. Rev. Lett.* 39 (23 Dec. 1977), pp. 1436–1439. DOI: 10.1103/PhysRevLett.39.1436. URL: https://link.aps.org/doi/10.1103/PhysRevLett.39.1436.

[34] B. Flaugher and K. Meier. *A compilation of jet finding algorithms.* Tech. rep. FNAL/C–90/248-E. United States, 1990, p. 14. URL: http://inis.iaea.org/search/search.aspx?orig_q=RN:22033120.

[35] G. Arnison et al. "Hadronic jet production at the CERN proton-antiproton collider". In: *Physics Letters B* 132.1 (1983), pp. 214–222. ISSN: 0370-2693. DOI: https://doi.org/10.1016/0370-2693(83)90254-X. URL: https://www.sciencedirect.com/science/article/pii/037026938390254X.

[36] Gavin P Salam and Grégory Soyez. "A practical seedless infrared-safe cone jet algorithm". In: *Journal of High Energy Physics* 2007.05 (May 2007), pp. 086–086. DOI: 10.1088/1126-6708/2007/05/086. URL: https://doi.org/10.1088%5C%2F1126-6708%5C%2F2007%5C%2F05%5C%2F086.

[37] TeV4LHC QCD Working Group et al. *Tevatron-for-LHC Report of the QCD Working Group.* 2006. DOI: 10.48550/ARXIV.HEP-PH/0610012. URL: https://arxiv.org/abs/hep-ph/0610012.

[38] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. "The catchment area of jets". In: *Journal of High Energy Physics* 2008.04 (Apr. 2008), pp. 005–005. DOI: 10.1088/1126-6708/2008/04/005. URL: https://doi.org/10.1088/1126-6708/2008/04/005.

[39] S. Catani et al. "Longitudinally-invariant k-clustering algorithms for hadron-hadron collisions". In: *Nuclear Physics B* 406.1 (1993), pp. 187–224. ISSN: 0550-3213. DOI: https://doi.org/10.1016/0550-3213(93)90166-M. URL: https://www.sciencedirect.com/science/article/pii/055032139390166M.

[40] S. Catani et al. "New clustering algorithm for multijet cross sections in e+e annihilation". In: *Physics Letters B* 269.3 (1991), pp. 432–438. ISSN: 0370-2693. DOI: https://doi.org/10.1016/0370-2693(91)90196-W. URL: https://www.sciencedirect.com/science/article/pii/037026939190196W.

[41] "A Cambridge-Aachen (C-A) based Jet Algorithm for boosted top-jet tagging". In: (July 2009).

[42] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. "The catchment area of jets". In: *Journal of High Energy Physics* 2008.04 (Apr. 2008), pp. 005–005. DOI: 10.1088/1126-6708/2008/04/005. URL: https://doi.org/10.1088/1126-6708/2008/04/005.

[43] Ryan Atkin. "Review of jet reconstruction algorithms". In: *Journal of Physics: Conference Series* 645 (Oct. 2015), p. 012008. DOI: 10.1088/1742-6596/645/1/012008. URL: https://doi.org/10.1088/1742-6596/645/1/012008.

[44] K. Rabbertz. *Jet Physics at the LHC: The Strong Force beyond the TeV Scale.* Springer Tracts in Modern Physics. Springer International Publishing, 2016. ISBN: 9783319421131. URL: https://books.google.com.mx/books?id=6E3YjwEACAAJ.

[45] Torbjörn Sjöstrand, Stephen Mrenna, and Peter Skands. "PYTHIA 6.4 physics and manual". In: *Journal of High Energy Physics* 2006.05 (May 2006), pp. 026–026. DOI: 10.1088/1126-6708/2006/05/026. URL: https://doi.org/10.1088/1126-6708/2006/05/026.

[46] Johan Alwall et al. "MadGraph 5: going beyond". In: *Journal of High Energy Physics* 2011.6 (June 2011), p. 128. ISSN: 1029-8479. DOI: 10.1007/JHEP06(2011)128. URL: https://doi.org/10.1007/JHEP06(2011)128.

[47] Enrico Bothmann et al. "Event Generation with Sherpa 2.2". In: *SciPost Phys.* 7 (3 2019), p. 34. DOI: 10.21468/SciPostPhys.7.3.034. URL: https://scipost.org/10.21468/SciPostPhys.7.3.034.

[48] C. Oleari. "The POWHEG BOX". In: *Nuclear Physics B - Proceedings Supplements* 205-206 (Aug. 2010), pp. 36–41. DOI: 10.1016/j.nuclphysbps.2010.08.016. URL: https://doi.org/10.1016%5C%2Fj.nuclphysbps.2010.08.016.

[49] Diana León Silverio. "Search for Monotop Production in Events with One Lepton, Missing Transverse Energy, and Jets". Thesis. Facultad de Ciencias Físico Matemáticas. Benemérita Universidad Autónoma de Puebla, Nov. 2018.

[50] Anja Butter et al. "Deep-learned Top Tagging with a Lorentz Layer". In: *SciPost Physics* 5.3 (Sept. 2018). DOI: 10.21468/scipostphys.5.3.028. URL: https://doi.org/10.21468%5C%2Fscipostphys.5.3.028.

[51] Gregor Kasieczka et al. "The Machine Learning landscape of top taggers". In: *SciPost Physics* 7.1 (July 2019). DOI: 10.21468/scipostphys.7.1.014. URL: https://doi.org/10.21468%5C%2Fscipostphys.7.1.014.

[52] Torbjörn Sjöstrand et al. "An introduction to PYTHIA 8.2". In: *Computer Physics Communications* 191 (June 2015), pp. 159–177. DOI: 10.1016/j.cpc.2015.01.024. URL: https://doi.org/10.1016%5C%2Fj.cpc.2015.01.024.

[53] et. al. J. de Favereau. "DELPHES 3: a modular framework for fast simulation of a generic collider experiment". In: *Journal of High Energy Physics* 2014.2 (Feb. 2014). DOI: 10.1007/jhep02(2014)057. URL: https://doi.org/10.1007%5C%2Fjhep02%5C%282014%5C%29057.

[54] Wes McKinney. *pandas : powerful Python data analysis toolkit*. 2011. URL: http://pandas.sourceforge.net/.

[55] S. Liu et al. *3D Point Cloud Analysis: Traditional, Deep Learning, and Explainable Machine Learning Methods*. Springer International Publishing, 2021. ISBN: 9783030891794. URL: https://books.google.com.mx/books?id=KkCjzgEACAAJ.

[56] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

[57] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. DOI: 10.48550/ARXIV.1903.02428. URL: https://arxiv.org/abs/1903.02428.

[58] Minjie Wang et al. *Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks*. 2019. DOI: 10.48550/ARXIV.1909.01315. URL: https://arxiv.org/abs/1909.01315.

[59] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. "Graph neural networks in particle physics". In: *Machine Learning: Science and Technology* 2.2 (Jan. 2021), p. 021001. DOI: 10.1088/2632-2153/abbf9a. URL: https://doi.org/10.1088%2F2632-2153%2Fabbf9a.

[60] Murat Abdughani et al. "Probing stop pair production at the LHC with graph neural networks". In: *Journal of High Energy Physics* 2019.8 (Aug. 2019). DOI: 10.1007/jhep08(2019)055. URL: https://doi.org/10.1007%5C%2Fjhep08%5C%282019%5C%29055.

[61] Xiangyang Ju et al. "Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors". In: *33rd Annual Conference on Neural Information Processing Systems*. Mar. 2020. arXiv: 2003.11603 [physics.ins-det].

[62] Gregor Kasieczka et al. "Deep-learning top taggers or the end of QCD?" In: *Journal of High Energy Physics* 2017.5 (May 2017). DOI: 10.1007/jhep05(2017)006. URL: https://doi.org/10.1007%2Fjhep05%282017%29006.

[63] G. W. Ding. *Draw ConvNet*. 2018. URL: https://github.com/gwding/draw_convnet.

[64] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[65] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[66] Sebastian Macaluso and David Shih. "Pulling out all the tops with computer vision and deep learning". In: *Journal of High Energy Physics* 2018.10 (Oct. 2018). DOI: 10.1007/jhep10(2018)121. URL: https://doi.org/10.1007%2Fjhep10%282018%29121.

[67] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[68] MD SAJID ANIS et al. *Qiskit Aer*. 2022. URL: https://github.com/Qiskit/qiskit-aer.
[69] Md Sajid Anis et.al. *Simulators*. 2022. URL: https://qiskit.org/documentation/tutorials/simulators/1_aer_provider.html.
[70] Md Sajid Anis et.al. *Qiskit: Basic Syntaxis*. 2022. URL: https://qiskit.org/textbook/ch-appendix/qiskit.html.
[71] Md Sajid Anis et.al. *assemble*. 2022. URL: https://qiskit.org/documentation/stable/0.19/stubs/qiskit.compiler.assemble.html.
[72] Gavin E. Crooks. *Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition*. 2019. DOI: 10.48550/ARXIV.1905.13311. URL: https://arxiv.org/abs/1905.13311.
[73] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.
[74] Matthias C. Caro et al. *Generalization in quantum machine learning from few training data*. 2021. DOI: 10.48550/ARXIV.2111.05292. URL: https://arxiv.org/abs/2111.05292.
[75] Martin Larocca et al. *Group-Invariant Quantum Machine Learning*. 2022. DOI: 10.48550/ARXIV.2205.02261. URL: https://arxiv.org/abs/2205.02261.
[76] Satoshi Morita and Hidetoshi Nishimori. "Mathematical foundation of quantum annealing". In: *Journal of Mathematical Physics* 49.12 (2008), p. 125210.
[77] Arthur Pesah et al. "Absence of Barren Plateaus in Quantum Convolutional Neural Networks". In: *Physical Review X* 11.4 (Oct. 2021). DOI: 10.1103/physrevx.11.041011. URL: https://doi.org/10.1103%2Fphysrevx.11.041011.
[78] Hsin-Yuan Huang et al. "Quantum advantage in learning from experiments". In: *Science* 376.6598 (June 2022), pp. 1182–1186. DOI: 10.1126/science.abn7293. URL: https://doi.org/10.1126%2Fscience.abn7293.
[79] *Thesis repo*. URL: https://github.com/Luis518518/Thesis.git.