UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Quantum algorithm for the implementation of a perceptron

Relatore	Laureando
Prof. Simone Montangero	Riccardo Bon

Anno Accademico 2018/2019

Abstract

Quantum information and quantum computers have enabled us to write algorithms, - quantum algorithms - that in some cases display a speedup with respect to their classical counterpart. As machine learning is becoming of upmost importance not only for our every-day-life and for scientific research, it is natural to inquire if we could use methods from quantum information to speed up machine learning's processes. In this Thesis, we review a model of a quantum perceptron, a key component for modern neural networks, and simulate its time evolution to characterize behavior.

Contents

Introduction

The idea of a 'quantum computer' dates back to 1982 [1] when Richard Feynman proposed to use a computer based on the laws of Quantum Mechanics, i.e. the quantum computer, to simulate quantum systems as this task is very hard to perform using classical computers. Feynman, in fact, believed that such a machine, being based on Quantum Mechanics' laws, would be much more suitable for the task. Researchers who started testing this idea in the 1990s, found that it was indeed possible to achieve a quantum simulation of a quantum system. To this day this remains one of the main fields of possible application of Quantum Information.

Another field of application would be 'Quantum Algorithms', i.e. algorithms written to run on quantum computers. Their relevance is due to the possibility, thanks to them, of tackling what would be difficult[1] problems in classical information, all thanks to properties derived directly from Quantum Mechanics (such as the fact that pure states are length-one vectors in a Hilbert space) some of which will be recalled in Chapter 1.

Quantum Algorithms that exhibit a gain in the time of computation, thanks to quantum information, are said to experience a 'quantum speedup'. To show how these quantum speedups are obtained, let us consider the example of the quantum search algorithm. This enables us to search in an unsorted database of N entries in time proportional to $\sqrt{N}$, rather than in time proportional to N (what we can do classically) [2]. Although the speedup might not seem that impressive in this case, in many others, among which we mention the important (as it is at the base of many others quantum algorithms) Quantum Fourier Transform (QFT) case, the speedup is exponential [3, 2]. There are, however, some problems with the quantum speedup. Although it is true that the QFT exhibit an exponential speedup compered to its classical counterpart, i.e. the Fast Fourier Transform (FFT), the process of readout of the output (needed if we want to extract any information, as will be recalled in Chapter 1) of the algorithm introduces a slow down that cancels the speed of the QFT, resulting in a time of execution of the same order of the one of FFT. Not only that, also the process of inputting data in the quantum computer might be very slow. A possible solution to this latter problem is using what is known as 'quantum random access memory' (qRAM for short) [4]. These problems hinder not only the QFT algorithm, but many others. Still, there are quantum algorithms, such as Shor's one, that offer an exponential advantage compared to their classical version.

The field we are focusing on in this Thesis, is Quantum Machine Learning. Since learning is the process of using past experiences to gather knowledge, the goal with machine learning is to program a machine (a computer) to learn from input data. We then aim to use the knowledge just acquired, to exploit (efficiently) tasks otherwise impossible as spam filters for emails and pattern recognition [5].

Consider the problem of the spam email filter. If we wanted to write a program to do just so, we would need a database containing emails we regard as spam. The database will be gradually filled up by emails that the user marks as spam. Every time a new email arrives, the program will sequentially scan the database looking for an identical email; if there is one, the new email is marked as spam, if there is not an identical email, the program does nothing. We remark that it takes a one word difference for the program to fail. On the other hand, the machine learning approach is to train the computer to recognize frequently-used words in the emails marked as spam by the user. If a new email contains too many of these words, it is moved to the spam section.

Thanks to its ability at recognizing patterns in data, machine learning is very effective when it comes

[1]Here difficult is intended as algorithmic complexity, i.e. the computational resources needed to solve the problem scale as a super polynomial with the dimension of the problem.

to analyzing data in, for example, high energy physics where, quoting [6], "The process to determine the particle characteristics depends on track reconstruction and their fitting, this process is governed by pattern recognition methods performed on offline data and no doubt machine learning techniques contribute a lot.". Because of this, quantum machine learning is an intriguing new subject, possibly exploiting a speed up in the calculation. Not only that, as pointed out in [2], classical machine learning methods recognize the patterns that they produce. So, if quantum processor could generate patterns difficult for classical ones, perhaps they could also identify patterns difficult for classical algorithms.

What we are most interested in in this Thesis are quantum perceptrons. In fact what we are going to review and perform a classical simulation of the model of a quantum perceptron proposed in [7]. The Thesis is organized as follows:

Chapter 1 We present the elements of quantum information that are necessary to comprehend what will be presented in Chapter 2. In particular, we are going to briefly introduce the concepts of qubit, gate and multiple qubits gates. The last section of the chapter is dedicated to the description of Hypergraph states. In particular, we will review the procedure needed to produce them, as we will make use of it in Chapter 2.

Chapter 2 After a brief description of a classical perceptron, we will build step-by-step the quantum algorithm proposed in [7] to implement a perceptron on a quantum computer.

Chapter 3 After a concise introduction of the Cirq library, we will show, again by following [7], how to apply the quantum algorithm from Chapter 2 to pattern recognition. We will then present the results we obtained by simulating the quantum algorithm on a classical computer using the tools provided by the aforementioned Cirq library, using both 2 and 6 qubits.

Chapter 1

Quantum Information Theory

In this chapter we give a brief recall to the basic notions of quantum information which will be exploited in Chapter 2: the qubit, single and multiple quantum gates.

In the last section we define Hypergraph states, describing what they are, and how to generate them using the gates previously introduced.

1.1 Qubit

The fundamental entity of both classical computation and classical information is the bit. The classical bit is typically represented by a binary digit, namely a number which can either be 0 or 1. We refer to the value of these two numbers as 'states' in the sense that a bit whose value is 0 (1) will be in state 0 (1). We would like to stress that a bit is always either in state 0 or state 1 and, to know its state, we need to measure it. This is the key process that enables us to access the information stored in the bit: if the state of the bit is unknown to us, once we measure it, the information in our possession increases. Finally, the state of a classical bit is not influenced by a measurement, namely it does not change due to the measurement.

The same role in quantum information and quantum computation is played by the so-called 'qubit' [3]. In the remaining of this section, we will give a mathematical description of a qubit, independently of any possible physical realization. By doing so, what we will say can be generalized to any quantum computer such as the ones using the polarization states of a photon, either vertical of horizontal, or the two states of an electron in an atom. For both more examples and further detail, we redirect to [3].

Coming back to our general description of a qubit, its state can be described, according to rules of Quantum Mechanics (QM), and using Dirac notation, as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \quad \text{with} \quad \alpha, \beta \in \mathbb{C} \text{ s.t. } |\alpha|^2 + |\beta|^2 = 1. \tag{1.1}$$

In (1.1), the kets $|0\rangle$ and $|1\rangle$ form what is known as the 'computational basis' of the Hilbert space[1] $\mathbb{C}^2$, i.e. the Hilbert space of a qubit.

Since the (pure) state of a qubit is represented by a norm-one vector in $\mathbb{C}^2$, it can always be written in terms of the linear combination of other states, where the only constraint comes from the norm-one condition. This property, known as 'superposition', is one of the crucial differences between qubits and bits. Indeed, while the state of a bit is always, as already stated, either 0 or 1, we can clearly see from (1.1) that $|\psi\rangle$, which is a legitimate qubit-state, is neither $|0\rangle$ nor $|1\rangle$, but a superposition of the two.

As for the bit, if we want to access the information stored in the qubit, we need to measure it. According to QM's rules, once we measure the qubit in a given basis, we either find result 0, with probability $|\alpha|^2$, or result 1, with probability $|\beta|^2$. We remark that it is from this interpretation of $|\alpha|^2$ and $|\beta|^2$ as outcome probabilities that we can justify the requirement of normalization. Furthermore,

[1]A Hilbert space is a (real or complex) vector space endowed with an inner product, that induces a norm function, with respect to which the space is complete.

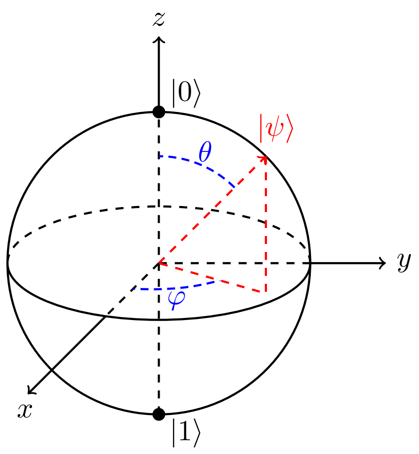


Figure 1.1: Bloch sphere for a qubit $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle$

this interpretation means that by repeating many times the same measurement with the same (known) input state, we find that the probabilities of obtaining either 0 or 1 are equal, respectively, to $|\alpha|^2$ and $|\beta|^2$. Concerning the state after the measurement, according to QM, it is different, in principle, from the one we had before. Indeed, suppose we measure a qubit, initially in state $|\psi\rangle$, and we obtain result 0 (1); then the state after the measurement occurred is $|0\rangle$ ($|1\rangle$). This process is known as 'collapse' of the wave function.

Given the restrictions on α and β, we may rewrite $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ as:

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right) \tag{1.2}$$

where $\theta, \varphi, \gamma \in \mathbb{R}$. As we know from QM, we can neglect the global phase $e^{i\gamma}$ as it is not observable, obtaining:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\,. \tag{1.3}$$

The angles, θ and φ can be interpreted as spherical coordinates on the so called 'Bloch sphere', depicted in Figure 1.1.

Though θ and φ are real numbers, and thus in theory we could store an unlimited amount of data using just them and one qubit, as we already recalled, once we measure a qubit, we only find either 0 or 1, with no trace left of the two angles. Despite that, since, according to QM, the time evolution of an isolated quantum system is unitary, if we do not measure the qubit, the evolution does keep track of both of θ and φ. This is one key aspect to keep in mind when writing quantum algorithms in order to make them efficient.

1.1.1 Multiple qubit system

We move now to consider a system with more than just one qubit. Although we will restrict to the case of 2 qubits, what we will say can easily be generalized to the case of N qubits. We will make use of the concept of tensor product, represented by $\otimes$, since, as we know from QM, the phase space of a two qubits system is the tensor product of the two single qubit's Hilbert spaces. The computational basis for the new Hilbert space $\mathbb{C}^2 \otimes \mathbb{C}^2$ is $\{\,|00\rangle, |01\rangle, |10\rangle, |11\rangle\,\}$, where $|00\rangle \equiv |0\rangle \otimes |0\rangle \equiv |0\rangle|0\rangle$. The most general state, expressed in this basis is $|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$ with $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$.

Given two qubits, we can now choose whether to measure just one or both of them. If we choose to measure both the qubits, the possible results are similar to the case of the single qubit case, namely we will get output $x \in \{\,00, 01, 10, 11\,\}$ with outcome probability $|\alpha_x|^2$, leaving the qubit in state $|x\rangle$. If, however, we decide to measure just one of the two qubits, say the first one, and we get result 0, whose outcome probability is $|\alpha_{00}|^2 + |\alpha_{01}|^2$, the state after the measurement occurred must be $\frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$.

1.2 Quantum gates

Quantum gates are linear unitary operators that enable us to describe the manipulation of a set of isolated qubits. The unitarity condition, that is the only constraint on linear operators to be considered "good" gates, derives from the necessity of $|\psi'\rangle = U|\psi\rangle$, where U is just a generic gate, to be the state vector of a qubit, namely to have norm equal to 1.

As what we are going to do with gates is to construct circuits to perform algorithms, we present here the series of conventions that are used to draw such circuits. A circuit is supposed to be read from left-to-right, each wire drawn represents a wire in the quantum computer; gates are generally drawn as rectangles. The initial state, if nothing is said otherwise, is supposed to be all $|0\rangle$s.

1.2.1 Single qubit gates

We begin by introducing the gate performing the analogue of the classical NOT gate, which is the only non-trivial classical single bit gate. Such gate, which will be indicated by either NOT or X, acts on the computational basis as follows: $\mathrm{NOT}\,|0\rangle = |1\rangle$ and $\mathrm{NOT}\,|1\rangle = |0\rangle$. The action on linear combinations of these two elements of the basis, such as on $|\psi\rangle$ in (1.1), can be calculated by exploiting the linearity of the gate.

Using the vector-space nature of $\mathbb{C}^2$, we can express the computational basis, and therefore the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, as follows

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix} \Rightarrow |\psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix}. \tag{1.4}$$

Using this notation, the NOT gate is represented by the following 2×2 matrix

$$X = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}. \tag{1.5}$$

Where we used the letter X because the matrix in (1.5) is the same as σ_x, one of the Pauli matrices[2]. It is worth noting that the matrix representation is not unique to the X gate; the action of every gate on a single qubit can be summarized by a 2×2 unitary matrix acting on a two complex-components vector.

Three other fundamental single-qubit gates are the Z and Y gates and, last but certainly not least, the so called 'Hadamard' gate which will be referred to as H. Their explicit matrix form, in the computational basis, is the following

$$Z = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} \quad Y = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix} \quad H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}. \tag{1.6}$$

The Z gate acts as follows on the elements of the computational basis $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$, whereas the Hadamard gate's action is

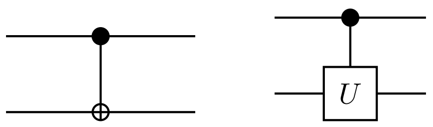
$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \equiv |+\rangle \quad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \equiv |-\rangle. \tag{1.7}$$

The main reason for its importance, is what happens to a state of N-qubits when we apply to each an Hadamard; let's begin with just two qubits. What we are trying to evaluate is

$$\begin{aligned} H^{\otimes 2}|0\rangle_1|0\rangle_2 = H|0\rangle_1 H|0\rangle_2 &= \left(\frac{|0\rangle_1 + |1\rangle_1}{\sqrt{2}}\right)\left(\frac{|0\rangle_2 + |1\rangle_2}{\sqrt{2}}\right) \\ &= \frac{1}{2}\big(|0\rangle_1|0\rangle_2 + |0\rangle_1|1\rangle_2 + |1\rangle_1|0\rangle_2 + |1\rangle_1|1\rangle_2\big). \end{aligned} \tag{1.8}$$

Thanks to (1.8), we see that applying $H^{\otimes 2}$ to $|0\rangle|0\rangle$ ($\equiv |00\rangle$) yields a superposition of the states of the computational basis. Before generalizing what we just said to the case of N-qubits, let us introduce a

[2]The Pauli matrices are: $\sigma_x = \left(\begin{smallmatrix}0 & 1\\1 & 0\end{smallmatrix}\right), \sigma_y = \left(\begin{smallmatrix}0 & -i\\i & 0\end{smallmatrix}\right)$ and $\sigma_z = \left(\begin{smallmatrix}1 & 0\\0 & -1\end{smallmatrix}\right)$.



(a) CNOT's circuit (b) C-U circuit

Figure 1.2: Circuital representation of some gates.

rather useful notation. Recalling the conversion algorithm from binary to decimal base and vice versa, we could write

$$\frac{1}{2}\big(|00\rangle + |01\rangle + |10\rangle + |11\rangle\big) = \frac{1}{2}\sum_{x=0}^{3} |x\rangle\,. \tag{1.9}$$

Now the generalization of (1.8) to the case of N-qubits is trivial

$$H^{\otimes N}\,|\underbrace{0\ldots0}_{N\text{ times}}\rangle \equiv H^{\otimes N}\,|0\rangle^{\otimes N} = \frac{1}{\sqrt{2^N}}\sum_{x=0}^{2^N-1} |x\rangle\,. \tag{1.10}$$

1.2.2 Multiple qubits gate

One of the most relevant classes of multiple-qubit gates is that of the controlled gates; an example is the controlled-not gate (CNOT) which, in its simplest form, is a two-qubit gate. The two bits on which the CNOT operates are known as 'controlled' qubit and 'target' qubit respectively. The CNOT applies a NOT gate to the target qubit if the controlled one is in state $|1\rangle$, while does nothing if the state is $|0\rangle$. Its matrix form is

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{1.11}$$

the basis being

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \tag{1.12}$$

Another form to express it is[3] $\text{CNOT} = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|$. Its circuital representation is shown in Figure 1.2a.

Given a gate U, it is possible to define its controlled-version C-U whose action is similar to that of the CNOT, the difference being the application of U, instead of the NOT, if the state of the controlled qubit is $|1\rangle$. The circuital representation is depicted in Figure 1.2b. One example of this kind of gate is the controlled-Z gate $C^N Z$, where N is the number of the involved qubits; this gate acts on the state in which all qubits are in state $|1\rangle$, changing the sign of its coefficient. That is,

$$\alpha C^N Z\,|1\ldots1\rangle = -\alpha\,|1\ldots1\rangle\,. \tag{1.13}$$

1.3 Quantum Hypergraph States

Since we will need them in the next chapter, we are going to briefly present Hypergraph states and show how we can generate them. For further details, see [8]. We are going to proceed by steps: starting from the simplest case of Graph states, moving on to k-uniform Hypergraph states, to finally arrive at Hypergraph states.

[3]We recall, in the case of dimension 2 for convenience, that $|0\rangle\langle 0| = (1\ 0)\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix}$

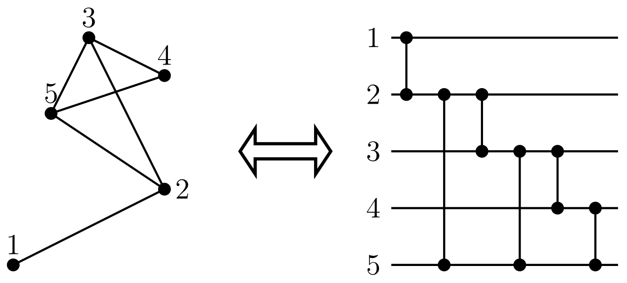


Figure 1.3: Diagram of the graph $g_2 = \{\,5, E\,\}$ (left side) with the corresponding graph-state-generating-circuit (right side). For each qubit starting from state $|+\rangle$ there is a vertex, and the vertical line with two dots connecting two qubits is a C^2Z gate. Here $E = \{\,(1,2),(2,3),(3,4),(4,5),(3,5),(2,5)\,\}$ and, as you can see, the number of elements of E is equal to 6, namely the number of edges in the graph or, in the Graph state, the number of C^2Z gates.

Let us begin by giving an informal definition of what a mathematical graph is. A mathematical graph $g_2 = \{\,V, E\,\}$ is a set of a fixed number of 'vertices' V and a set of 'edges' E, where each edge in E connects exactly, namely no more, no less, 2 vertices in V; i.e. $E \subseteq V \times V$. The subscript "2" underlines this property of the edges. For example, $g_2 = \{\,5, E\,\}$ with $E = \{(1,2),(2,3),(3,4),\,(4,5),(3,5),(2,5)\}$ (see the left side of Figure 1.3) is a graph with 5 vertices and 6 edges, each connecting 2 vertices.

The formalism of graphs can be used to define a class of quantum states called graph states. In our analogy, vertices are represented by qubits in the state $|+\rangle$, see (1.7), while edges by C^2Z gates, see (1.13), where the superscript "2" plays the same role of the subscript one in the mathematical graph. A Graph state is usually depicted by $|g_2\rangle$ and obtained as follows

$$|g_2\rangle = \prod_{\{\,i_1,i_2\,\}\in E} C^2Z_{i_1,i_2}\,|+\rangle^{\otimes n} \tag{1.14}$$

where $\{\,i_1, i_2\,\} \in E$ means that vertex i_1 and i_2 in the graph are connected by an edge, namely the C^2Z gate will be applied to the qubits corresponding to the vertices i_1 and i_2. In order to clarify, recalling the example $g_2 = \{\,5, E\,\}$, its corresponding Graph state would be

$$|g_2\rangle = C^2Z_{i_1,i_2}C^2Z_{i_2,i_3}C^2Z_{i_2,i_5}C^2Z_{i_3,i_5}C^2Z_{i_3,i_4}C^2Z_{i_4,i_5}\,|+\rangle_{i_1}|+\rangle_{i_2}|+\rangle_{i_3}|+\rangle_{i_4}|+\rangle_{i_5}\,.$$

See the right side of Figure 1.3 for its circuit. The order in which we apply the $C^2Z_{i_1,i_2}$ gates is irrelevant as they commute.

We now move to k-uniform Hypergraphs. A k-uniform Hypergraph is indicated with $g_k = \{\,V, E\,\}$, where V is the same as before, while now $E \subseteq V \times \cdots \times V$, namely, it is the set of 'hyperedges', that are nothing but edges connecting more than 2 vertices. In our case, being this a k-uniform Hypergraph, each hyperedge connects exactly k vertices from V. The quantum analogue, known as Hypergraph states, are then obtained as follows

$$|g_k\rangle = \prod_{\{\,i_1,\ldots,i_k\,\}\in E} C^kZ_{i_1\ldots i_k}\,|+\rangle^{\otimes n} \tag{1.15}$$

where now C^kZ acts on the k states that correspond to the vertices connected by a hyperedge; these are indicated by $\{\,i_1,\ldots,i_k\,\} \in E$.

Finally, Hypergraphs are denoted by $g_{\leq n} = \{V, E\}$, where V is the same set of vertices. In this case, contrary to the one of k-uniform Hypergraphs, the hyperedges can connect any number k of vertices up to n, i.e. $1 \leq k \leq n$, with n being the number of vertices in our Hypergraph. The Hypergraph states are then denoted by

$$|g_{\leq n}\rangle = \prod_{k=1}^{n}\prod_{\{\,i_1,\ldots,i_k\,\}\in E} C^kZ_{i_1\ldots i_k}\,|+\rangle^{\otimes n}\,. \tag{1.16}$$

We remark two things regarding the product on k from (1.16): first, note that if we remove it, we get back (1.15); this is just because Hypergraphs, and so Hypergraph states, can be seen as various

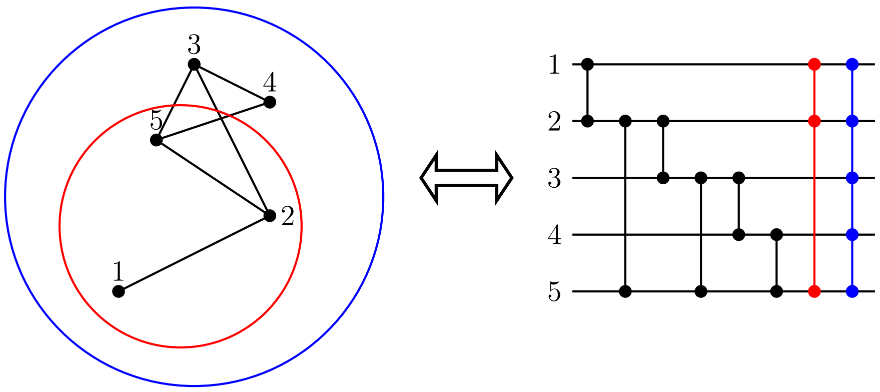


Figure 1.4: On the left we see the diagram corresponding to the Hypergraph $g_{\leq 5} = \{5, E\}$, while on the right side we have the circuit used to generate the corresponding Hypergraph state. Concerning the left side, we see that in this particular example, we have 5 vertices and $E = \{(1,2),(2,3),(3,4),(4,5),(3,5),(2,5),(1,2,5),(1,2,3,4,5)\}$; in particular we thus have 8 hyperedges. Because of this, on the right side we have 5 qubits, all starting in state $|+\rangle$, and 8 C^kZ gates, 6 with $k = 2$, 1 with $k = 3$ (the one drawn in red) and, finally, one with $k = 5$ (the one drawn in blue).

k-uniform Hypergraphs where k varies from 1 to m, where m is the maximum number of vertices connected by a hyperedge. Second, note that the aforementioned product goes from 1 to n even though we can build a Hypergraph state in which any hyperedge connects m vertices with $m < n$. We can do this because for k from $m + 1$ to n, there will not be any hyperedge, meaning no more C^kZ will be applied, thus leaving the Hypergraph state unchanged. See Figure 1.4 for an example of both the diagram of a Hypergraph, as well as the circuit generating the corresponding Hypergraph state.

Hypergraph states are relevant to us because it is possible to prove that every state of N qubits of the form

$$|\varphi\rangle = \frac{1}{\sqrt{2^N}} \sum_{x}^{2^N-1} (-1)^{f(x)} |x\rangle \quad \text{with} \quad f(x) : \{0,1\}^N \to \{0,1\}$$

is an Hypergraph state, thus, to generate it, we can use the gates in (1.16).

Chapter 2

Quantum perceptron

In this chapter we are presenting the quantum algorithm we are going to use to implement a perceptron on a quantum computer; we will follow [7]. Before presenting the quantum algorithm we will introduce what a perceptron is.

2.1 Perceptron

The perceptron, i.e. the first model of a classifier [9], was proposed by Rosenblatt in 1957 [10]. Its basic component is an artificial neuron whose simplest (and first) model was proposed in 1943 by McCulloch and Pitts [11]. The neuron compares the input, stored in the components of what is known as 'input vector' $\boldsymbol{i}$, with the weights stored in the components of the 'weight vector' $\boldsymbol{w}$. It then outputs a yes/no kind-of answer (respectively 1/0) using the basic scheme of operation that is depicted in Figure 2.1: it computes $\boldsymbol{i} \cdot \boldsymbol{w} + b$ (this how the comparison is performed) where $\boldsymbol{i}$ is the real input vector (1), $\boldsymbol{w}$ is the weight vector (2) whose components are also real but predetermined via a training procedure and, finally, b is the bias term[1](3).

$$\boldsymbol{i} \to \sum_{j=0}^{m-1} i_j w_j + b \tag{2.1}$$

The output of the neuron is $\operatorname{sign}(\boldsymbol{i} \cdot \boldsymbol{w} + b)$ (4) where $\operatorname{sign}(x)$ is the sign function.

Since a single neuron is able to only give yes/no as an answer, it is not very powerful in terms of computation, for this reason, Rosenblatt's model contains multiple neurons. They are organized in layers, the input of layer $l+1$ being the output of layer l; the final layer, i.e. the one whose output is presented to the user, is a single-neuron-layer. To determine the correct weight-vector's components, Rosenblatt proposed to fix the weights of all the layers but the last one; then, determine the weights of the last neuron through training. If the number of layers is n, this process is equivalent to considering as input, rather than the vectors $\boldsymbol{i}_j \in \mathcal{I}$, the vectors $\boldsymbol{z}_i \in \mathcal{Z}$, where $\mathcal{Z}$ is the space to which the output of the neurons in layer $n-1$ belongs to. So, starting from the training set $(\boldsymbol{i}_1, y_1), \ldots, (\boldsymbol{i}_m, y_m)$, where y_j is the label of $\boldsymbol{i}_j$, namely the value we aim to get as output from the perceptron if we feed it with input $\boldsymbol{i}_j$, the transformed input will be $(\boldsymbol{z}_1, y_1), \ldots, (\boldsymbol{z}_m, y_m)$. Now, at each step k in the training procedure, an element of this latter sequence is fed into the perceptron. Supposing of having completed the training algorithm up to the element $(\boldsymbol{z}_k, y_k)$ in the training sequence, and denoting by $\boldsymbol{w}_k$ the weight-vector at step k, the training algorithm is the following:

1. If the element $(\boldsymbol{z}_{k+1}, y_{k+1})$ in the training sequence is classified correctly, i.e. if

   $$y_k(\boldsymbol{w}_k \cdot \boldsymbol{z}_{k+1} + b) > 0,$$

   then $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k$;

2. If the classification is not correct, i.e. if

   $$y_{k+1}(\boldsymbol{w}_k \cdot \boldsymbol{z}_{k+1} + b) < 0,$$

[1]The bias can always be merged in the scalar product by setting $i_m = 1$ and $w_m = b$

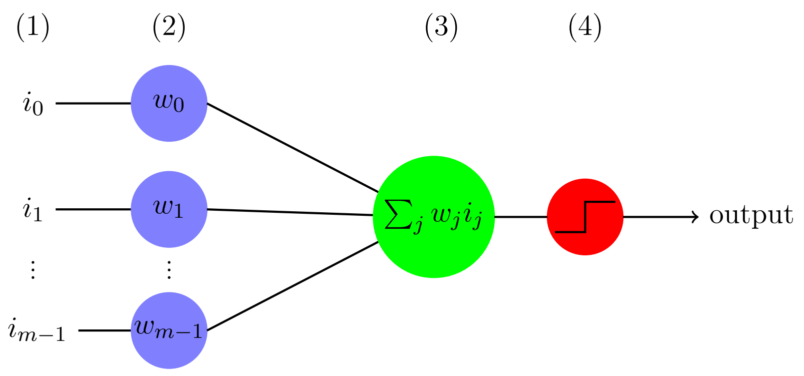


Figure 2.1: Scheme of the neuron model by McCulloch and Pitts. $i_0, i_1, \ldots, i_{m-1}$ are the components of the input vector (1), while $w_0, w_1, \ldots, w_{m-1}$ are the ones of the weight vector (2). The neuron works by confronting the input and weight vector via the scalar product (3) outputting the answer via the sign function (4), represented by the graph in the red circle.

then $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_{k+1}\boldsymbol{z}_{k+1}$;

3. In the first step $\boldsymbol{w}_1 = 0$.

2.2 Quantum perceptron

In our quest to finding a quantum algorithm for implementing a perceptron, we will restrict ourselves to the case where both input and weight vectors' components are either ± 1, i.e. $i_j, w_j \in \{-1, 1\}$. Moreover, we will examine the case of a single neuron where the output is either 1 or 0 (yes/no answers). First we write both the input and weight vector quantum counterparts. To do this, we recall that, as the components of weight and input vectors can only be $w_j, i_j \in \{-1, 1\}$, and the number of components in the vector representation of a N qubits system is 2^N, we can write

$$|\psi_i\rangle = \frac{1}{\sqrt{2^N}} \sum_{j=0}^{2^N-1} i_j |j\rangle \quad |\psi_w\rangle = \frac{1}{\sqrt{2^N}} \sum_{j=0}^{2^N-1} i_w |j\rangle \quad \text{with} \quad i_j, w_j \in \{-1, 1\}, \tag{2.2}$$

where $|\psi_i\rangle$ serves the role of the input vector, while $|\psi_w\rangle$ serves the role of the weight vector. By using this technique, we exploit the exponential advantage in the memory required to store the input and weight vectors since we just require N qubits to store a number of bits equal to $m = 2^N$. From now on, if not stated otherwise, we will always consider $m \equiv 2^N$.

Having established the form of $|\psi_i\rangle$, we just need a gate, lets call it U_i, which takes as input the usual starting-state for any quantum algorithm, i.e. $|0\rangle^{\otimes N}$, and outputs $|\psi_i\rangle$. In principle, any unitary matrix with the vector $\boldsymbol{i}$ in the first column is adequate for this. We are not going to propose any explicit form for U_i for the moment as this is will be done in Section 2.3. A possible alternative to this step in the future, as for now it represent a difficult technological problem [2], might come from the execution of a call to a quantum RAM (qRAM) [4] where the $|\psi_i\rangle$ was previously stored. Since, however, this is not the main concern of this Thesis, we will not go into detail in explaining what a qRAM really is.

The next step in our algorithm is to compute the inner product between $|\psi_i\rangle$ and $|\psi_w\rangle$. We can do this efficiently by recalling that $\langle\psi_i| U^\dagger U |\psi_w\rangle = \langle\psi_i|\psi_w\rangle$ for any U unitary. Choosing the gate U_w in such a way that

$$U_w |\psi_w\rangle = |1\rangle^{\otimes N} = |m-1\rangle, \tag{2.3}$$

the computation of the scalar product becomes trivial. Once again, this task can be performed by any unitary matrix having $\boldsymbol{w}$ in the last row, and, as before, we will examine the particular form of U_w in Section 2.3. Knowing how U_w acts on $|\psi_w\rangle$, to determine the result of the said inner product, we just need to compute

$$U_w |\psi_i\rangle = \sum_{j=0}^{2^N-1} c_j |j\rangle \equiv |\varphi_{i,w}\rangle \quad \text{with} \quad \sum_{j=0}^{2^N-1} |c_j|^2 = 1, \tag{2.4}$$

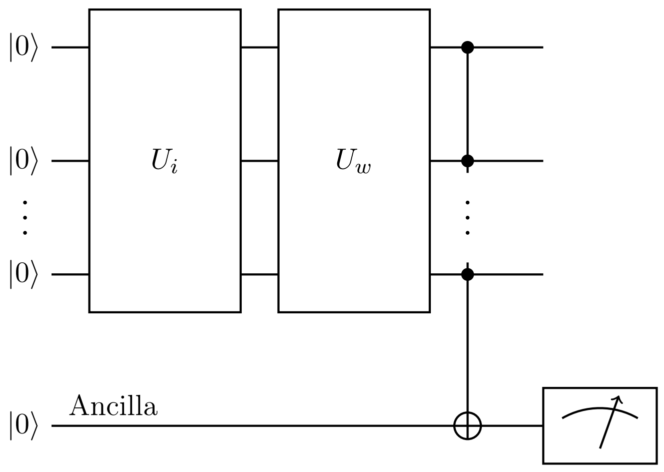


Figure 2.2: Scheme of the circuit for the quantum perceptron model. U_i is the gate generating the state $|\psi_i\rangle$ such that its components (representing the state in the computational basis) are the same of $\boldsymbol{i}$; while U_w is the gate computing the inner product $\langle\psi_i|\psi_w\rangle$, where $|\psi_w\rangle$ is the vector with the same components as $\boldsymbol{w}$. The ancilla qubit is needed to measure the value of this scalar product.

where the coefficients c_j depend on the particular form of U_w. Exploiting the unitariety of U_w, we can write

$$\langle\psi_i|\psi_w\rangle = \langle\psi_i| U_w^\dagger U_w |\psi_w\rangle = \langle\varphi_{i,w}|m-1\rangle = c_{m-1}. \tag{2.5}$$

We remark that having $c_{m-1} = 1$ corresponds to $|\psi_i\rangle = |\psi_w\rangle$, namely having the same state both in input and in output. Computing now the same inner product directly from (2.2), we get

$$\langle\psi_i \,|\, \psi_w\rangle = \frac{1}{2^N}\sum_{k,j}^{2^N-1} i_k w_j \langle j \,|\, k\rangle = \frac{1}{2^N}\sum_{k,j}^{2^N-1} i_k w_j \delta_{k,j} = \frac{1}{m}\boldsymbol{i}\cdot\boldsymbol{w}. \tag{2.6}$$

Comparing (2.5) with (2.6), we se that $\boldsymbol{w}\cdot\boldsymbol{i}$, namely exactly what the classical perceptron computes in order to produce its output, is equal to $m\langle\psi_w|\psi_i\rangle$, that is $\boldsymbol{w}\cdot\boldsymbol{i} = m\langle\psi_w|\psi_i\rangle$. To get the value of the inner product on the right-hand side of the latter formula, we use an ancilla qubit and a multi-controlled NOT gate with all the qubits on which U_i and U_w act upon, working as controlled qubits, and the ancilla working as the target one; see Figure 2.2. This yields

$$\mathrm{CNOT}\big(|\varphi_{i,w}\rangle\otimes|0\rangle_a\big) = \mathrm{CNOT}\left(\sum_{j=0}^{m-1} c_j|j\rangle\otimes|0\rangle_a\right) = \sum_{j=0}^{m-2} c_j|j\rangle\otimes|0\rangle_a + c_{m-1}|m-1\rangle\otimes|1\rangle_a \tag{2.7}$$

where the qubit with subscript a is the ancilla one.

Thanks to this, by measuring the ancilla qubit, we are now able to estimate the value of c_{m-1} or, better, $|c_{m-1}|^2$ since we know that we find output $|1\rangle$ (what we refer-to as 'activated perceptron') with probability $|c_{m-1}|^2$. Moreover, as the probability of obtaining an active perceptron depends on the squared-modulo of c_{m-1}, the anti-parallel vectors of $\boldsymbol{i}$ and $\boldsymbol{w}$ give us the same result; this is a trivial consequence of the invariance of a quantum state under a global phase factor.

The measurement also introduces the non-linearity required to output functions in the classical perceptron model. This is particularly relevant when it comes to neural networks, where, if the output function of every perceptron were to be linear, all of the layers of perceptrons constituting the network would be equal to a single perceptron, as the combination of multiple linear transformation remains a linear transformation.

2.3 Implementation of the gates

We are going to consider two particular methods for the implementation of U_i and U_w: the "brute force" method and a method based on quantum Hypergraph states. Lets start with the brute force one.

We will omit the ancilla-qubit from this as it does not affect what we are about to say. As we are starting from the state $|0\rangle^{\otimes N}$, in order to prepare $|\psi_i\rangle$, we first need to achieve an equal superposition of all the elements of the computational basis (ESECB), namely we need something similar to what we have done in (1.10). We then need to introduce the minus signs in the right places in order to reproduce a given vector $\boldsymbol{i}$. For this, we use multiple signs flip gates, where the action on $|j'\rangle$ of a sign flip gate $\text{SF}_{N,j}$ is

$$\text{SF}_{N,j}\,|j'\rangle \begin{cases} |j'\rangle & \text{if } j \neq j' \\ -\,|j'\rangle & \text{if } j = j' \end{cases}. \tag{2.8}$$

We already know that, for any number of qubits N such that $m = 2^N$, $C^N Z$ corresponds to $\text{SF}_{N,m-1}$, see (1.13), whereas a single Z gate corresponds to $\text{SF}_{1,1}$. In order to explain how to obtain all the others $\text{SF}_{N,j}$, allow us to consider an example. The method we will use can then be generalized to the case of N qubits. Suppose we are in the case with $N = 3$ qubits and, starting from

$$\frac{1}{\sqrt{8}}\sum_{k=0}^{7} |k\rangle\,, \tag{2.9}$$

we would like to only flip the sign of $|6\rangle \equiv |110\rangle$, leaving the others signs unchanged. We cannot use $C^3 Z$ as this would flip the sign of $|7\rangle \equiv |111\rangle$, nor can we use Z applied to one of the first two qubits, as this would add a phase to all the other states where the qubit to which we have applied Z is in state $|1\rangle$. What we do instead is:

1. Use $\mathbb{1} \otimes \mathbb{1} \otimes \text{NOT}$ to flip the third qubit's state thus transforming $|6\rangle \equiv |110\rangle$ into $|7\rangle \equiv |111\rangle$;
2. Apply the $C^3 Z$ to transform $|111\rangle$ into $-\,|111\rangle$, while leaving all the other states unchanged;
3. Finally, reapply $\mathbb{1} \otimes \mathbb{1} \otimes \text{NOT}$ to transform $-\,|111\rangle$ into $-\,|110\rangle \equiv -\,|6\rangle$.

This procedure works because $\text{NOT} \equiv \text{NOT}^{-1}$, and $C^3 Z$ only changes the amplitude of the state $|111\rangle$, leaving the others unchanged. Generalizing this idea to the case of N qubits, we get

$$\text{SF}_{N,j} = O_j\; C^N Z\; O_j \quad \text{with} \quad O_j = \bigotimes_{l=0}^{N-1} (\text{NOT}_l)^{1-j_l}, \tag{2.10}$$

where j_l indicates the l-th digit in the binary representation of the number j, and $(\text{NOT})^0 = \mathbb{1}$. For example if $j = 4$ and $N = 3$ then $j_0 = 0$, $j_1 = 0$ and $j_2 = 1$ as 100_2 converts to[2] 4_{10}.

Now, we need to find the form of gate U_w. Keeping in mind that U_w must be such that $U_w\,|\psi_w\rangle = |1\rangle^{\otimes N}$, if we find a gate exploiting the transformation from $|\psi_w\rangle$ into the balanced superposition $|\psi_0\rangle \left(= H^{\otimes N}\,|0\rangle^{\otimes N}\right)$, we could then apply $H^{\otimes N}$ to $|\psi_0\rangle$ to get $|0\rangle^{\otimes N}$, and finally $\text{NOT}^{\otimes N}$ to achieve $|1\rangle^{\otimes N}$. Using a scheme to clarify what we said we obtain

$$|\psi_w\rangle \longrightarrow |\psi_0\rangle \xrightarrow{H^{\otimes N}} |0\rangle^{\otimes N} \xrightarrow{\text{NOT}^{\otimes N}} |1\rangle^{\otimes N}\,. \tag{2.11}$$

Basically we need to invert the process $|\psi_0\rangle \to |\psi_w\rangle$. Since this process makes use of the sign flip gates from (2.10), and all these gates are the inverse of themselves, we can just use the appropriate sign flips taking $|\psi_0\rangle$ to $|\psi_w\rangle$, and this will also compute the change from $|\psi_w\rangle$ to $|\psi_0\rangle$.

A more efficient strategy consists in using a Hypergraph-states-generating-subroutine (HSGS), as $|\psi_{i,w}\rangle$ coincide with this class of states; see Section 1.3. After the initial $H^{\otimes N}$ needed to set all of the qubit's states to the state $|+\rangle$, we have to introduce the appropriate minus signs to reproduce the input/weight vector (in the following we will only refer to the input vector). To achieve this task, we first check whether in $|\psi_i\rangle$ there are any components with just one qubit in state $|1\rangle$, i.e. states like $|0\ldots010\ldots0\rangle$, requiring the minus sign. If so, we apply a Z gate to the designed qubit. We repeat the process until there are no-more of such components. Now, for $k = 2,\ldots,N$, we apply a

[2] In this the subscripts 2 and 10 are used to indicate the basis in which the numbers are expressed; 2 meaning binary while 10 decimal

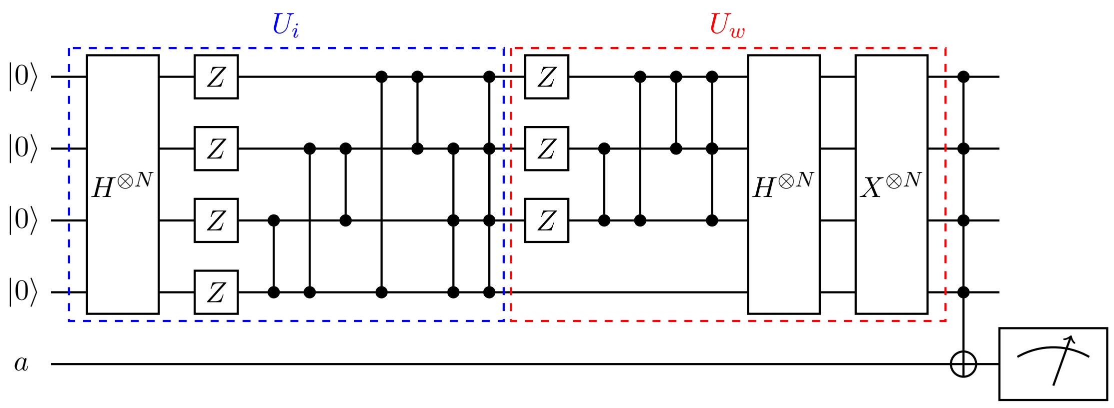


Figure 2.3: An example of the circuit for a perceptron in the case of $N = 4$ qubits. The input vector has components $i_0 = i_{10} = -1$ while $i_j = 1$ for $j \neq 0, 10$; on the other hand, the weight vector has components $w_0 = w_1 = -1$ and $w_j = 1$ for $j = 2, \dots, 15$. We employed the HSGS for the building of both U_i and U_w. We recall that C^kZ gates are depicted as vertical lines with black dots, while the X gate is just another name for the NOT one.

similar procedure, the difference being the application of C^kZ instead of Z, for any component in the expression of $|\psi_i\rangle$ with k qubits in state $|1\rangle$ for which the corresponding component i_j of vector $\boldsymbol{i}$ is equal to -1. What we are saying is: once we fixed the value of k, we check if any of the states in $|\psi_i\rangle$ with exactly k qubits (which may not be adjacent) in state $|1\rangle$ need a minus sign and this was not already placed by one of previous steps. If so, we apply C^kZ gate to the qubits in need.

In all of the steps of this process, we might introduce some undesired minus signs that we need to get rid of. In order to achieve that, we simply apply the necessary C^kZ gates, where k is now the number of qubits in state $|1\rangle$ in the components of $|\psi_i\rangle$ which exhibit the undesired minus sign. As C^kZ acts as the identity on the components with a number of qubits in state $|1\rangle$ smaller than k, once we reach $k = N$, we obtain the desired $|\psi_i\rangle$. The same process can be applied to the building of U_w.

Let us consider a simple example to clarify the procedure we examined. Suppose we are in the case of $N = 2$ qubits, and that we aim to reproduce vector $\boldsymbol{i} = (1, 1, -1, 1)$ starting from $|\psi_i\rangle = \frac{1}{\sqrt{4}} \sum_{j=0}^{3} |j\rangle$. Following the procedure, we act as follows:

1. We check whether the states $|01\rangle$ or $|10\rangle$ need a minus sign by looking at the components i_j for $j = 2^0, 2^1$ of vector $\boldsymbol{i}$. Since $i_2 = -1$, and this component in the vector-representation of the state $|\psi_i\rangle$ corresponds to the state $|10\rangle$, see (1.12), we apply the gate $Z \otimes \mathbb{1}$ to $|\psi_i\rangle$. This yields (omitting the normalization factor)

$$(Z \otimes \mathbb{1}) |\psi_i\rangle = |00\rangle + |01\rangle - |10\rangle - |11\rangle \equiv |\psi_i'\rangle \, .$$

As the coefficient multiplying $|01\rangle$ is 1 and $i_1 = 1$, we move on to considering the states with both qubits in state $|1\rangle$;

2. Due to the previous step, we see that the coefficient of the state $|11\rangle$ is -1. Because of this, if it was $i_3 = -1$, we would not need to act in any way on $|\psi_i'\rangle$ and the procedure would terminate here. However, since we have $i_3 = 1$, we need to change the state $|11\rangle$'s coefficient from -1 to 1. To achieve this, we apply a C^2Z gate to $|\psi_i'\rangle$ obtaining

$$C^2Z |\psi_i'\rangle = |00\rangle + |01\rangle - |10\rangle + |11\rangle \, .$$

As we can see, the procedure has produced the correct input state for the algorithm.

Finally, we remark that, because of the invariance of the output $|c_{m-1}|^2$ of the quantum algorithm under a global -1 factor, i.e. using $|\psi_i\rangle$ instead of $-|\psi_i\rangle$ does not affect the final output of the algorithm, the number of independent signs in $|\psi_i\rangle$ is 2^{N-1} rather than 2^N. The same can be said regarding $|\psi_w\rangle$.

Chapter 3

Numerical simulation

In this chapter, after a concise presentation of the Cirq library, we are going to present the results we obtained by running our algorithm using the said software. We reproduce some of the results shown in [7] and then we will apply the algorithm to an original case.

3.1 Cirq library

Quoting [12], "Cirq is a software library for writing, manipulating, and optimizing quantum circuits and then running them against quantum computers and simulators. Cirq attempts to expose the details of hardware, instead of abstracting them away, because, in the Noisy Intermediate-Scale Quantum (NISQ) regime, these details determine whether or not it is possible to execute a circuit at all.".

Considering a given fixed number of qubits, circuits in Cirq are made of `Moments`, namely a collection of `Operations` all occurring at the same abstract time slice. An `Operation` consist in applying a gate to a certain subset of the said qubits; see Figure 3.1 for a scheme describing this structure.

Regarding the qubits, Cirq offers two kinds of geometrical structures: `GridQubits` and `LineQubits`. In the former, the qubits are organized in a 2d lattice, whereas in the latter they are grouped in a 1d lattice; one arrangement has to be chosen over the other depending on the particular problem we are studying. We will adopt the `LineQubits`.

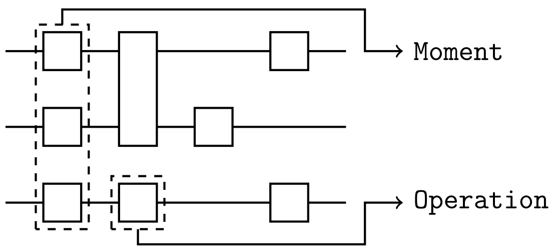


Figure 3.1: Cirq typical circuit structure. `Operations` are actions operating on subset of qubits, the most frequent operations are gates; `Moments` are a collection of operations all happening at the same abstract time slice.

3.2 Numerical simulation

We are going to start with two qubits to test the algorithm in a simple scenario. As stated in Section 2.2, having 2 qubits enables us to represent vectors with $2^2 = 4$ components and, since each component can either be 1 or -1, we have 2^4 different inputs to analyze and to compare with as many different weights.

To help visualizing all these different inputs and weights, we will represent them by using images: each image is made of $2 \times 2 = 4$ pixels (one for each bit) and each one can be either black or white, depending on the value of the corresponding bit. The conversion-procedure for going from vector $\boldsymbol{i}$ ($\boldsymbol{w}$) to the corresponding image is the following:

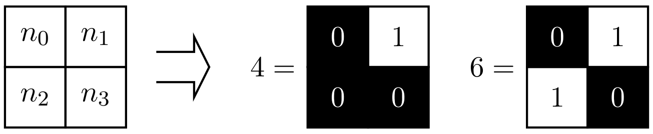


Figure 3.2: Images obtained from input/weight vectors. Here, to find the input vector $\boldsymbol{i}$ corresponding to the number k_i, we first compute the binary conversion, namely $(k_i)_2 = \{ n_0, n_1, n_2, n_3 \}$ and then exploit the relation $i_j = (-1)^{n_j}$ to find the components of the vector. The numbers inside the pixels are there just to help comprehending the procedure bringing from k_i to the image.

1. We label each pixel from left to right and from top to bottom with a number n_k whose value is either 0 or 1. As we show in Figure 3.2, we represent the bits whose value is 0 with a back pixel and those with value 1 with a white pixel;
2. Since at each pixel corresponds a component of $\boldsymbol{i}$ ($\boldsymbol{w}$), we impose that $i_j = (-1)^{n_j}$ ($w_j = (-1)^{n_j}$);
3. We assign to each input (weight) vector a number, let's call it k_i (k_w), which is the decimal conversion of the string $\{ n_0, n_1, n_2, n_3 \}$.

For instance, in the case of $k_i = 6$, we have $\{ n_0, n_1, n_2, n_3 \} = \{ 0, 1, 1, 0 \}$ thus, recalling that $i_j = (-1)^{n_j}$, we obtain $\boldsymbol{i} = (1, -1, -1, 1)$.

We now describe the algorithm we wrote using the Cirq library's tools and that we used to produce the outputs we will later present. The inputs provided by the user are k_i, k_w and the number of qubits N. The program then converts k_i and k_w into the respective vectors using the procedure described above. Once this is completed, the program generates U_i and U_w exploiting the HSGS from the end of Section 2.3. Finally, the program applies the required CNOT gate and the measurement on the ancilla qubit. The output of the program is the number of times, out of the n repetitions, that 1 and 0 come out. In the following we fixed $n = 10000$.

In order to test the algorithm, we ran it for each pair (k_i, k_w) for $k_i, k_w = 0, \ldots, 15$; the results are shown in Figure 3.3. Looking at the histogram, we can see that for any value of $\boldsymbol{w}$, the perceptron is able to identify (namely those for which we have output $|c_{m-1}|^2 = 1$) the vectors $\boldsymbol{i}$ such that[1] $\boldsymbol{i} = \pm\boldsymbol{w}$. In all the other cases, the output of the perceptron is either in the range $]0.24, 0.26[$, or equal to 0. We remark that for all possible combinations, the output of the perceptron $|c_{m-1}|^2$ is in agreement with (2.6).

Furthermore, from the histogram we clearly see that our algorithm is symmetric under the swap of the input and the weight vectors. Moreover, the appearance of the diagonal going from $(0, 14)$ to $(14, 0)$ is related to symmetry under a global -1 factor multiplying either the weight or the input vector[1].

In order to test the quantum algorithm with a higher number of qubits, we set $N = 6$ and check the capability of one neuron to recognize the letters of the alphabet. To do this, we used[2] 8×8 pixels images to portray the letters, and run the algorithm for any possible combination of letters. As before, we organized the results we obtained in a histogram, see Figure 3.4. On one axis of the histogram there are the letters (in alphabetical order) corresponding to the number k_i, while on the other axis there are the ones corresponding to the number k_w still ordered in alphabetical order.

The results confirm what we expected: the neuron is activated ($|c_{m-1}|^2 = 1$) for input and weight vectors corresponding to identical letters, while it remains (almost) perfectly inactive ($|c_{m-1}|^2 = 0$) in the majority of the other cases. There are, however, instances in which the neuron does mix up the letters with $|c_{m-1}|^2$ going from ~ 0.1 to 0.87. This is the case for letters having similar representations in the 8×8 pixels images, such as the pair (G, O). Due to this, if we aim to know whether the letter we provided to the algorithm is or is not the same as the one represented by the weight vector, we cannot state that the neuron is successful because of the non-neglectable outputs for letters that are not the same. If, however, we want to identify a letter given to us, we can compare the results obtained by iterating over all the letters in the alphabet represented by weights, then chose the vector

[1]The $\pm$, as pointed out in Section 2.1, is due to the output of the perceptron being equal $|c_{m-1}|^2 = 1$. We remark that going from $\boldsymbol{i}$ to vector $-\boldsymbol{i}$ corresponds to inverting black and white pixels in the image representation of the said vector. The same applies for the weight vector.

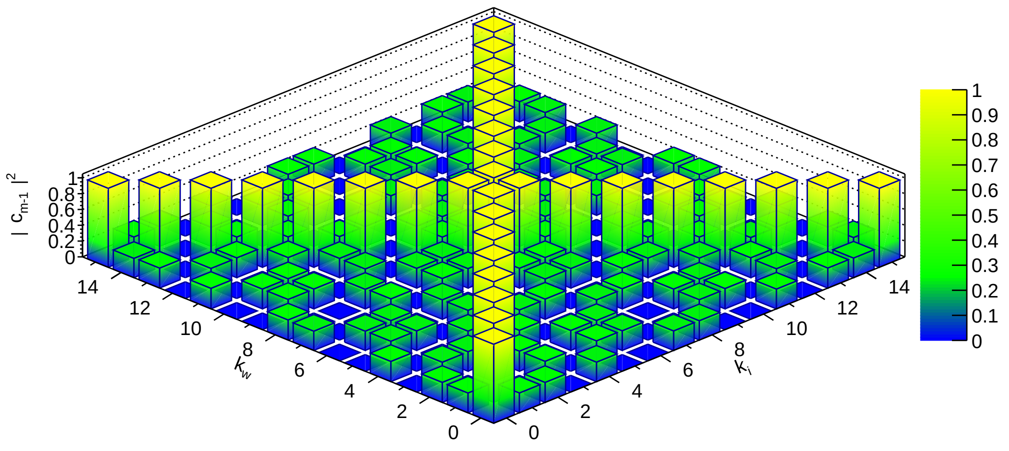


Figure 3.3: Histogram showing the output of the algorithm for all possible combinations of k_i and k_w using 2 qubits.

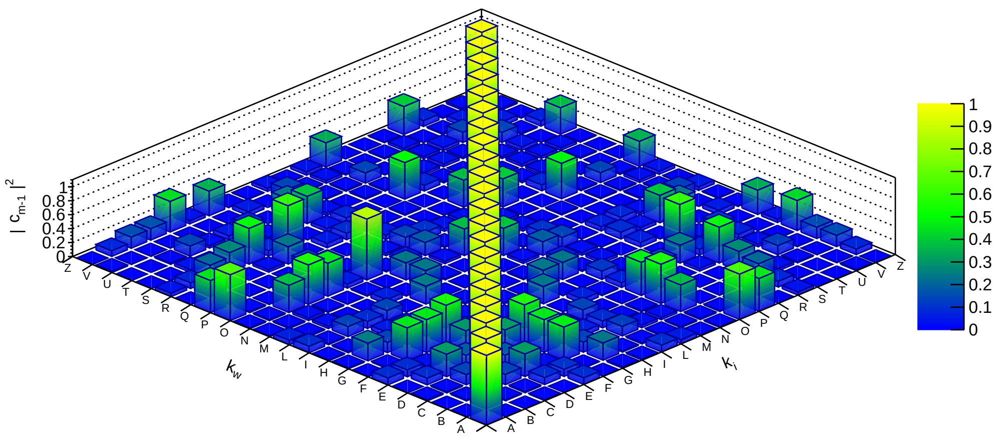


Figure 3.4: Histogram showing the output of the algorithm for all possible combinations of letters.

corresponding to the letter producing the highest result for $|c_{m-1}|^2$. Because of this, we can state that the neuron is indeed able to read a message.

[2]This is because the number of components $\boldsymbol{i}$ or $\boldsymbol{w}$ we can represent with 6 qubits is $2^6 = 64$ and $8^2 = 64$.

Conclusions

In this Thesis we have given a review of the most basic concepts of quantum information. With the idea of exploiting them in the quantum algorithm, we then moved on to the presentation of Hypergraph states, starting first from Graph states, to then move on to uniform Hypergraph states to finally arrive at the description of Hypergraph states.

After that, we reviewed the concept of a classical perceptron, and of a quantum algorithm to implement one on a quantum computer. We applied what we learned about Hypergraph states in the subroutine used to generate the initial states necessary for the algorithm to work. We showed that the algorithm guarantees an exponential advantage with respect to its classical counterpart by using N qubits to represent 2^N bits.

To test the algorithm, we used the tools provided by the Cirq library to perform a classical simulation of the aforementioned algorithm. We first started with 2 qubits; we represented the different input and weight vectors via images of 4 pixel, in order to demonstrate a pattern-recognition capability. We found that the algorithm works as expected, producing the correct outputs to be in perfect agreement with its classical version. We then increased the number of qubits to 6, and tested the neuron's capability to recognize letters and found once again that the neuron is successful.

Further development of what we have presented could be implementing an algorithm for multiple-layer neural networks: indeed, the single-neuron model used here, more specifically the McCulloch and Pitts neuron model, is not a powerful computational tool as we already know from classical machine learning. Neural networks, and more specifically feedforward ones, cope with the lack of computational power by organizing many neurons in layers, where the outputs of the neurons of layer l are the inputs of the ones of layer $l + 1$. The non-linearity in the output of the neurons, required for the correct working of neural networks, could be achieved in the quantum model by using two ancilla qubits: one to be measured, so to introduce the said non-linearity, the other one used to pass the output to the subsequent neuron.

In summary, we have discussed a quantum algorithm for implementing a classical perceptron on a quantum computer and tested it through a classical simulation using the Cirq library. We have shown that the algorithm here discussed guarantees an exponential advantage over the classical one as the number of the required bits is concerned, by representing 4 bits using just 2 qubits and 64 bits using just 6 qubits.

Bibliography

[1] Richard P. Feynman. “Simulating physics with computers”. In: *International Journal of Theoretical Physics* 21.6 (June 1982), pp. 467–488. ISSN: 1572-9575. DOI: 10.1007/BF02650179. URL: https://doi.org/10.1007/BF02650179.

[2] Jacob Biamonte et al. “Quantum machine learning”. In: *Nature* 549 (Sept. 2017), 195 EP -. URL: https://doi.org/10.1038/nature23474.

[3] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[4] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. “Quantum random access memory”. In: *Physical review letters* 100.16 (2008), p. 160501.

[5] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014. DOI: 10.1017/CBO9781107298019.

[6] Kapil Sharma. “Quantum machine learning in High-Energy Physics: The Future Prospects”. In: (May 2018). DOI: 10.13140/RG.2.2.34700.00642.

[7] Francesco Tacchino et al. “An artificial neuron implemented on an actual quantum processor”. In: *npj Quantum Information* 5 (Dec. 2019). DOI: 10.1038/s41534-019-0140-4.

[8] M. Rossi et al. “Quantum Hypergraph States”. In: *New Journal of Physics* 15 (Nov. 2012). DOI: 10.1088/1367-2630/15/11/113022.

[9] V. Vapnik. *The Nature of Statistical Learning Theory.* Information Science and Statistics. Springer New York, 1999. ISBN: 9780387987804. URL: https://books.google.it/books?id=sna9BaxVbj8C.

[10] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para.* Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. URL: https://books.google.it/books?id=P%5C_XGPgAACAAJ.

[11] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[12] The Cirq Developers. *Cirq documentation.* Release 0.5.0. Google. Apr. 2019. URL: https://buildmedia.readthedocs.org/media/pdf/cirq/stable/cirq.pdf.