UCSD **Université Luminy**

Quantum Learning and Quantum Perceptrons

Stage présenté devant

Université d'Aix-Marseille

pour obtenir le grade de

Master 2 - Physique Théorique et Mathématique

Présenté par

Jean-Christophe Gomez-Lavocat

Encadré par

David Meyer

Année 2009
Master 2 - Recherche

jcgomez-lavo@ec-marseille.fr
dmeyer@math.ucsd.edu

Abstract

French / Français

Ce document constitue le rapport de stage de seconde année de Master Recherche en Physique Théorique et Mathématique de l'université de Luminy. La thématique du stage dont ce rapport fait l'objet, s'articule autour de l'information quantique. Le stage en lui-même s'intitule ≪ Quantum Learning and Quantum Perceptron ≫, la traduction française pouvant être : ≪ Apprentissage Machine Quantique et Perceptron Quantique ≫.
Le document est divisé en trois parties. La première d'entre elles pose les bases de l'informatique classique et quantique. On y apprend les différences notables entre les deux paradigmes et on y donne les propriétés importantes. Quelques brefs exemples d'algorithmes quantiques sont rappelés. Dans la deuxième partie, les aspects classiques de l'apprentissage machine et des perceptrons sont introduits. Leurs limitations classiques qui constituent l'objet de cette étude est rappelée. Enfin, dans une troisième partie, nous introduisons la version quantique du perceptron et de l'apprentissage machine mise au point durant ce stage. À l'heure de la rédaction de ce rapport, les simulations n'étant pas terminées, nous décrivons les pistes que nous allons suivre dans la suite de ces recherches.
Mots Clés : Perceptron, Intelligence Artificielle, Information Quantique, Machine Learning, Qubit, Règle de Hebb, Apprentissage direct/passif, On-Line Learning

English / Anglais

This document has been written during an internship at *UCSD* for the Theoretical and Mathematical Physics MSc of Univeristé de Luminy. The theme of this thesis is Quantum Infomation and the subject is the following : "Quantum Learning and Quantum Perceptron".
This document is divided in three parts. First of all we present some basis of classical and quantum information. Differences and properties between the two paradigms are listed. Some examples of quantum algorithms are presented. In the second part the classical machine learning model is described and the perceptron is defined. The computational limitations of this classical model are detailed in the case of discrete perceptrons. Finally, the quantum version developped during this internship is introduced. Some simulations are still to be run, but the different possible improvements are detailed.
Keywords : Perceptron, Artificial Intelligence, Quantum Information, Machine Learning, Qubit, Hebbian Learning, Direct/Passive Learning, On-Line Learning

Table des matières

Chapitre 1

Quantum Computation

The field of quantum computation is not really new, but recent discoveries allowed it to have been really explored in the last decades. The main idea of *quantum computation* or *quantum information* is to process information and to run algorithms on quantum systems. In this report we will give a brief summary about the history of classical and quantum computation. Nevertheless, we will not go too in depth on this subject, and let the reader find out more details in the bibliography. After that section, we will give the definition of *Qubits* and some of their properties. In the end, we will introduce some of the most famous quantum algorithms, but the detailed review will be left to the attention of the curious reader [20].

1.1 History

1.1.1 The Origin of Computing Science

The origin of modern computing science could be attributed to *Turing* and his *Turing Machine* and also to the *Church-Turing Thesis*. In a remarkable paper written in 1936 [27], Alan Turing developed in detail an abstract notion of what we could nowadays call a programmable computer. This model is called a Turing Machine.

These developments have been performed in order to answer to the *Entscheidungsproblem* (the "decision problem") : Hilbert's tenth question of 1900. The exact problem was :

Find an algorithm to determine whether a given polynomial Diophantine equation with integer coefficients has an integer solution.

After a remark from *Heinrich Behmann*, Hilbert made his questions quite precise. First, was mathematics *complete* ? Second, was mathematics *consistent* ? And thirdly, was mathematics *decidable* ? The first two questions were answered in 1930 by *Kurt Gödel*. The third one was answered negatively by the *Church-Turing Thesis* (Church [4] - Turing [27]).

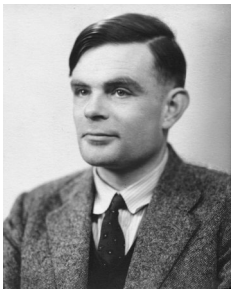

FIGURE 1.1 – Alan Turing

In this report we will not give more details about Turing machines. The curious reader could find more information about the theory of computing science in [26].
Not long after Turing's paper was published the first computers were developed. The developments were such that *Gordon Moore* gave in 1965 his famous law about the computer's power. The *Moore's law* is the following :

The computer power will double for constant cost roughly once every two years.

Nevertheless, the size of current transistors are beginning to run up against fundamental difficulties of size. Quantum effects will soon become predominant in the functioning of electronic devices. Many people believe that the Moore's law will probably become obsolete.

One possible solution is to move to a different computing paradigm. The reader should easily guess that *quantum computation* is the new paradigm researchers are trying to build. The advantages gained by a quantum computer will be understood in the following sections, after having introduced the notion of *computational complexity*.

The fact that quantum computing could be more efficient than the current classical computing *for some problems* is strengthened by recent discoveries in the field of quantum algorithms and also by this modified version of the Church-Turing Thesis :

Any algorithmic process can be simulated efficiently using a probabilistic Turing machine.

By saying that, we assume that a model of computation (that can be probabilistic) exists, which can simulate efficiently, a given algorithm

1.1.2 The emergence of Quantum Computation

In 1982, *Feynman* pointed out the difficulty of simulating quantum systems on classical computers [8]. He then suggested the use of computers based on quantum physics laws and properties. This application was the first evocation of a quantum computer and at the same time an improvement of classical results (the simulation of quantum systems is slow in classical computation).

In 1985 *David Deutsch* [6] attempted to define a computational device that would be capable of efficiently simulating an arbitrary physical system. He was naturally led by the laws of quantum mechanics to design this device (which is a quantum equivalent of Turing machines). Even if we still are not sure if the *Universal Quantum Computer* is sufficient enough to simulate an arbitrary system, it led to the model of quantum computers we currently use.

FIGURE 1.2 – Richard Feynam

FIGURE 1.3 – David Deutsch

These two ideas led to some breathtaking results, both theoretically and experimentally. The most famous theoretical results are due to *Shor* and *Grover* in the 1990's (see the section "quantum algorithms" below).

Since then, a handful of quantum computers have been built. The first, a 2-qubit quantum computer in 1998 [3], could perform trivial calculations before losing coherence after a few nanoseconds. In 2000, teams successfully built both a 4-qubit and a 7-qubit quantum computer. Research on the subject is still very active, although some physicists and engineers express concerns over the difficulties involved in upscaling these experiments to full-scale computing systems. Still, the success of these initial steps show that the fundamental theory could be tested.

1.2 Quantum Bits

1.2.1 Definition

In classical computation, the main component is the *bit* . It can take two different states : 0 and 1. Many bits are used to carry information by encoding it. In quantum information, the smallest piece of information is called a quantum-bit or *Qubit* . These Qubits also have two different basic states $|0\rangle$ and $|1\rangle$, called *pure states* .

The main difference between a classical and a quantum bit, is that a qubit can be in a state other than $|0\rangle$ or $|1\rangle$. It is also possible to form *linear combinations* of states, called *superpositions of states* .

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.1}$$

The numbers α and β are complex numbers. One of the usual ways to see a Qubit is to consider it as a vector of $\mathbb{C}^2$. We call the special cases $|0\rangle$ and $|1\rangle$: *pure states* . As qubits are quantum objects one cannot measure one, and get simultaneously the value of the coefficients. Instead, if we try to measure the quantum state $|\psi\rangle$ we will get the result $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. As they represent probabilities, we have : $|\alpha|^2 + |\beta|^2 = 1$.

This fact allows Qubits to be completely different from classical bits. Being in a superposition is not 'usual', and the information they can carry does not have a form we are used to. For instance, the qubit :

$$|+\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$$

Will return the qubit $|0\rangle$ with the same probability as $|1\rangle$. This could be compared to a perfect coin that you toss, and whose result you observe. At each measurement you would have only $|0\rangle$ or $|1\rangle$. But if you perform a huge amount of measurements, then you will guess the propability distribution of the qubit. This is the main difference between a bit and a qubit.

1.2.2 The Block Sphere

Because qubits are normalized, we can rewrite 1.1 and add a phase to characterize the different probability of the two pure states :

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

The first overall phase factor ($e^{i\gamma}$) could be ignored. So we would just write :

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle \tag{1.2}$$

The two parameters (θ and $\psi \in \mathbb{R}$) describes a unit sphere often called the *Bloch Sphere* . The Bloch sphere is a geometric representation of qubit states as points on the surface of a unit sphere. Many operations on single qubits that are commonly used in quantum information processing can be neatly described within the Bloch sphere picture.

We can notice that two opposite points on the Bloch sphere are orthogonal. This representation is often useful to represent some quantum operations on qubits (a generalization for multiple qubits is possible).

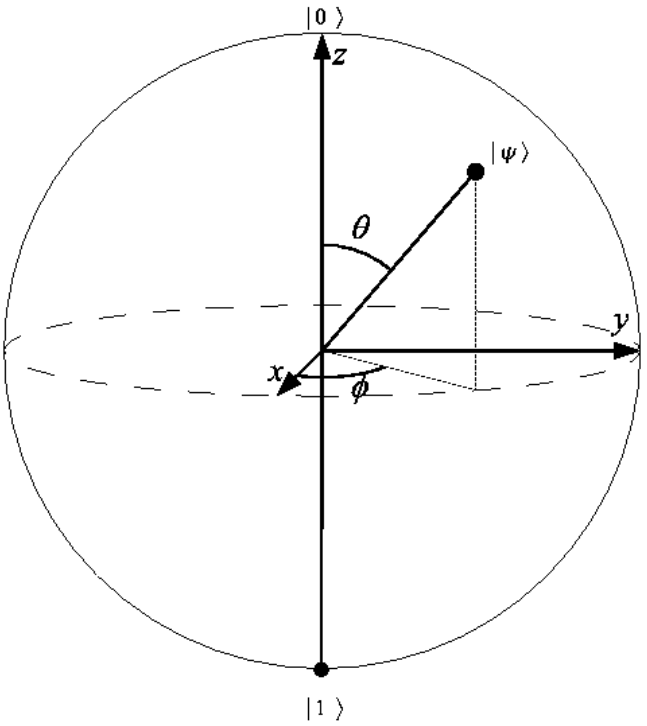

FIGURE 1.4 – The Bloch Sphere

1.2.3 Multiple Qubits

In classical computation, if we have two bits we could have four possibilities : 00,01,10,11. Thanks to the property of superposition, we could achieve the following quantum state for a 2-qubit system :

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle$$

Where $|00\rangle$,$|01\rangle$,$|10\rangle$,$|11\rangle$ are quantum pure states. In the previous relation, we still need to verify the normalization condition :

$$\sum_{i\in\{00,01,10,11\}} |\alpha_i|^2 = 1$$

In order to understand the meaning of $|00\rangle$ we have to make the notation precise. For two different qubits $|\psi\rangle$, $|\varphi\rangle$ we write :

$$|\psi\varphi\rangle = |\psi\rangle\,|\varphi\rangle = |\psi\rangle \otimes |\varphi\rangle \tag{1.3}$$

Where $\otimes$ is the tensor product.

If we measure only one of the two qubits (let's say the first one) the following probabilities apply : the first qubit will give 0 with probability $|\alpha_{00}|^2 + |\alpha_{01}|^2$. And the qubit is now in the following physical state, called *post-measurement state* .

$$|\psi'\rangle = \frac{\alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$$

Example One of the best known 2-qubit state is the *Bell state* , used for entanglement experiments and which have been made famous by a paper by Einstein, Podolsky, Rosen under the name *EPR effect* .

The Bell state is the following :

$$\left|\Psi^{+}\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

The Bell state has the property that upon measuring the first qubit, one obtains two possible results : 0 with probability 1/2, leaving the post-measurement state $|00\rangle$ with probability 1 ; and 1 with probability 1/2, leaving the post-measurement state $|11\rangle$ with probability 1. As a result, the measurement over the second qubit always gives the same result as the measurement of the first qubit. The two qubits the are measured are then *correlated* .

In general, if we consider a system of n qubits, the system is of the following form : $|x_1...x_n\rangle$ and there exists 2^n pure states. This is one of the reasons why simulating a quantum system is difficult classicaly. If $N = 500$ this number is greater than the number of assumed atoms in the universe. This is also the reason why simulating a quantum computer is possible, but really time and/or space consuming.

1.3 Quantum Computation

In this section we will introduce the concept of *quantum gates* and *quantum circuits*. We will see to what extent they carry and manipulate quantum information.

1.3.1 Introduction and notation

In classical computers, circuits consist of wires and *logical gates* . The wires carry information (the bits) and the logical gates apply some modifications on it. One of the two possible operation for a single classical bit is the **NOT** which changes the value of a bit to its opposite ($0 \to 1$ and $1 \to 0$). We usually use a particular notation for standard gates, and a graphic representation. In the graphic representation, the inputs are represented by lines coming for the left, the gates are either a block or a circle for particular operations, and outputs are lines going to the right.

$|0\rangle$ —[NOT]— $|1\rangle$

FIGURE 1.5 – The NOT gate

The *Pauli Matrices* also represent gates. We define them by :

$$\sigma_X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \text{—[}X\text{]—}$$

$$\sigma_Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = \text{—[}Y\text{]—}$$

$$\sigma_Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \text{—[}Z\text{]—}$$

The quantum gates must propagate a normalized qubit. It then has to be a *unitary operator* . A unitary operator A is defined by :

$$A^\dagger A = I \tag{1.4}$$

The importance of a unitary operator is due to their property of conserving the inner product, and then, the norm of the quantum state in Hilbert space.

1.3.2 Single Qubit Gates

The quantum analogous for the NOT must be a unitary operator, and must invert the two states $|0\rangle$ and $|1\rangle$. The solution is not complicated :

$$\text{Not} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = —\boxed{X}—$$

We can notice that this is the X-Pauli matrix. We have the following :

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Unlike the classical case, where you only have two possibilities ($b \to b$ or $b \to \bar{b}$), for the quantum gates, you can choose your gate in U(2), the group of unitary operators. Different behaviors can then occur. For instance, the Z gate leaves $|0\rangle$ unchanged, but flips the sign of $|1\rangle$:

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

The second important quantum gate is the *Hadamard gate* :

$$\text{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = —\boxed{H}—$$

We will let the reader verify the following action of the Hadarmard gate. If we define the following state $|+\rangle = \dfrac{|0\rangle + |1\rangle}{\sqrt{2}}$ and $|-\rangle = \dfrac{|0\rangle - |1\rangle}{\sqrt{2}}$, the Hadamard gate does the following :

$$\text{H}\,|0\rangle = |+\rangle$$
$$\text{H}\,|1\rangle = |-\rangle$$

Even if we can use any of the unitary matrix of U(2) an arbitrary quantum gate could be generated by only three universal gates. This result is true for any number n of inputs, but the result is more famous for $n = 2$.

Theorem 1. *Any unitary matrix U could be decomposed as :*

$$U = e^{i\alpha} \begin{bmatrix} e^{-i\beta/2} & 0 \\ 0 & e^{i\beta/2} \end{bmatrix} \begin{bmatrix} \cos\gamma/2 & -\sin\gamma/2 \\ \sin\gamma/2 & \cos\gamma/2 \end{bmatrix} \begin{bmatrix} e^{-i\delta/2} & 0 \\ 0 & e^{i\delta/2} \end{bmatrix}$$

1.3.3 Multiple Qubits Gates

In the classical information theory area, five classical 2-bits gates are the **AND**, **OR**, **XOR**, **NAND**, **NOR**. The NAND alone is known as an *universal gate* because you can perform every 2-bit function by a combination of NAND gates.

In quantum information processing, the prototypical multi-qubit quantum gates is the *Controlled-NOT* or *CNOT*. This gate has two input qubits, known as the *control qubit* and the *target qubit* . It is represented below. The top line represents the control qubit while the bottom one represents the target qubit.

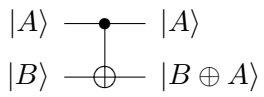

FIGURE 1.6 – The CNOT gate

$$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The CNOT is a quantum universal gate. If you use it with three inputs (2 control qubits and one target qubit), you can simulate any 2-bits logical gates.

We let the reader skim through [20] for more information about the basis of quantum circuits.

1.4 Quantum Algorithms

In the following section we recall some of the most famous quantum algorithms. Yet we will not go into details and explain the algorithm, it will go too far out of the focus of this report.

1.4.1 Computational Complexity

Computational complexity theory is a branch of the theory of computation in computer science that investigates the problems related to the resources required to run algorithms. In order to define it, three different concepts should be presented :

- Computability : Determine whether an algorithm exist that solves a given problem.
- Computational Complexity : For those problems that are computable, determine a coarse analysis of their time and space requirements. Such analyses may well ignore the differences between polynomials, and concentrate on the difference between polynomial and exponential behavior.
- Analysis of Algorithms : For those problems that can be solved in polynomial time, determine a more exact analysis of their time and space requirements

Given a specific computation problem, each of these three questions should be addressed. The first one will give people precision about a possible existing algorithm for the problem or not. The second question helps to classify problems in relation to their space or time complexity. Finally,the last one tells people which one of two distinct algorithms compute the problem in the most efficient way.

A wide theory is dedicated to the study of these three domains. Obviously, quantum algorithms are also subject to these models. They offered some good examples of speed improvements for classical algorithms : the best known example must be ths Shor's algorithm [25], which factorizes any integerof n digits in a time $O\left(n^3\right)$ whereas the best classical algorithm computes in time $Oe^{n^{1/3}(\log n)^{2/3}}$. In the same paper Shor gives an improvement for discrete logarithm.

The speedup could occur for any classical algorithm as it could also not exist. Quantum information is then trying to find out some answers about the improvement of classical known algorithms.

In another hand, quantum information could also help to find clues about the "P=NP Problem". The question of whether P = NP (can problems that can be solved in non-deterministic polynomial time also always be solved in deterministic polynomial time ?) is one of the most important open questions in theoretical computer science because of the wide implications of a solution. It is one of the seven one of the Millennium Prize Problems [1] by the Clay Institute, and a solution would be granted US$ 1,000,000.

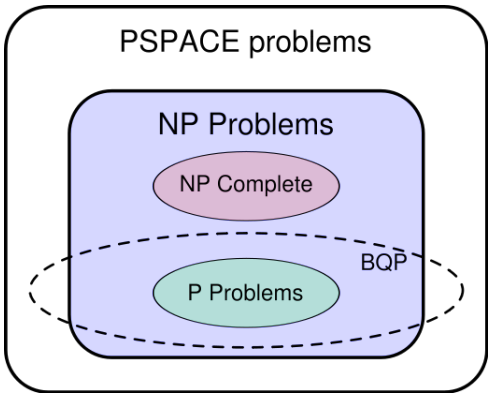

FIGURE 1.7 – Suspected Diagram of Complexity Classes
Source : *http ://en.wikipedia.org*

Without going in depth, the class of problems that can be efficiently solved by quantum computers is called *BQP* , for *bounded error, quantum, polynomial time*. Quantum computers only run probabilistic algorithms, so BQP on quantum computers is the counterpart of *BPP* on classical computers. It is defined as the set of problems solvable with a polynomial-time algorithm. BQP is suspected to be disjoint from NP-complete and a strict superset of P, but that is not known.

Both integer factorization and discrete log are in BQP. Both of these problems are *NP problems* suspected to be outside BPP, and hence outside P. Both are suspected to not be *NP-complete* .

But for the moment, only speedup in classical algorithms is known. In the following section we present some of them.

1.4.2 Examples of Quantum Algorithms

The Deutsch-Jozsa algorithm

This algorithm uses a single function evaluation to solve the problem of distinguishing between a constant and a balanced (returns 1 for half the domain and 0 for the other half) boolean function of d variables. The naive classical algorithm checks every possible values, so its complexity is $O(2^d)$. The *Deutsch-Jozsa quantum algorithm* [7] produces an answer that is always correct with just 1 evaluation of f.

First, do Hadamard transformations on n 0s, forming all possible inputs, and a single 1, which will be the answer qubit :

$$|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$$

$$|\psi_1\rangle = H |\psi_0\rangle = \sum_{x\in\{0,1\}^n} \frac{|x\rangle}{\sqrt{2^n}} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

1. see : http://www.claymath.org/millennium

Next, run the function once using $U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$. The effect is to XOR the result with the answer qubit.

$$|\psi_2\rangle = \sum_{x \in \{0,1\}^n} \frac{(-1)^{f(x)} |x\rangle}{\sqrt{2^n}} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

Finally, do Hadamards on the n inputs again, and measure the answer qubit. If it is 0, the function is constant, otherwise the function is balanced.

$$|\psi_3\rangle = \sum_{z \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} \frac{(-1)^{x.z+f(x)} |z\rangle}{2^n} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

where $x.z = x_0 z_0 \otimes ... \otimes x_{n-1} z_{n-1}$.

Finally we examine the probability of measuring $|0\rangle^{\otimes n}$.

Shor's Factoring Algorithm

Shor's Algorithm for factoring [25] is the most powerful and famous quantum algorithm. As we previously said its complexity is $O(n^3)$, but the basis of this algorithm, *Quantum Fourier Transformation* , helps for lots of quantum algorithms. Shor's algorithm consists of two parts :

- A reduction of the factoring problem to the *Problem of order-finding* (finding the period of a particular function), which can be done on a classical computer.
- A quantum algorithm to solve the order-finding problem.

The second part finds the period using the quantum Fourier transform, and is responsible for the quantum speedup.

Grover's Search algorithm

Grover's search algorithm [11] is a quantum algorithm for searching an unsorted database with N entries in $O(N^{1/2})$ time and using $O(\log(N))$ storage space. Classically, searching an unsorted database requires a linear search, which is O(N) in time.

Grover's algorithm can also be used for estimating the mean and median of a set of numbers, and for solving the collision problem. In addition, it can be used to solve NP-complete problems by performing exhaustive searches over the set of possible solutions.

Chapitre 2

The Standard perceptron

In this section we present the standard notion of perceptron. We will give its basic definition, some classical properties and limitations. As a lot of different definitions are available, we will mainly focus on the properties of the *discrete perceptron*.

2.1 Introduction to the general structure

2.1.1 Overview of the Biological Neural Network [1]

The biological neural network is the main component of living animals' brain. It is been known to give living animals their cognitive functions and *intelligence*. In general it is composed of groups of chemically connected *neurons* . A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called *synapses* , are usually formed of *axons* and *dendrites* . Apart from the electrical signaling, there are other forms of signaling that arise from neurotransmitter diffusion, which have an effect on electrical signaling. As such, neural networks are extremely complex.

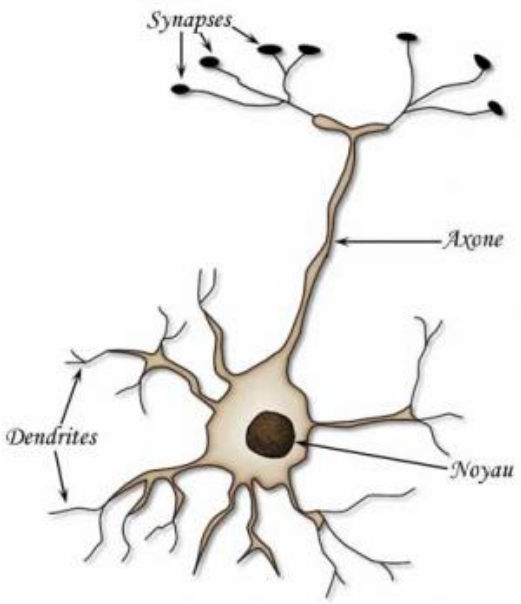

FIGURE 2.1 – Biological Neuron
Source : *http ://www.vieartificielle.com*

1. Source : http://en.wikipedia.org/wiki/Neuron
But another useful ressource is : http://www.blackwellpublishing.com/intropsych/pdf/chapter3.pdf

Neurons communicate with one another via synapses. There are several stimuli that can activate a neuron leading to electrical activity, including pressure, stretch, chemical transmitters, and changes of the electric potential across the cell membrane. Stimuli cause specific ion-channels within the cell membrane to open, leading to a flow of ions through the cell membrane, changing the membrane potential.

In a chemical synapse, the process of synaptic transmission is as follows : when an action potential reaches the axon terminal, it opens voltage-gated calcium channels, allowing calcium ions to enter the terminal. Calcium causes synaptic vesicles filled with neurotransmitter molecules to fuse with the membrane, releasing their contents into the synaptic cleft. The neurotransmitters diffuse across the synaptic cleft and activate receptors on the postsynaptic neuron.

The human brain contains a huge numbers of such connections. Each of the 10^{11} neurons has on average 7 000 synaptic connections to other neurons. The previous description put forward the mechanism of action of neurons. Information is trasnmitted from a neuron to the other, and neurons are stimuli activated. Since the 1950's people have tried to simulate these neurons with a simple mathematic model. The resulting theory is called *artificial neural network*.

2.1.2 Overview of the Artificial Neural Network

Artificial intelligence tries to simulate some properties of neural networks by copying the same basic structure. Artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control. The structure of an *artificial neuron* is quite the same as a biological one.

Basically an artificial neuron is a mathematical function conceived as a model of biological neurons. The artificial neuron receives one or more inputs (representing the one or more dendrites) and sums them to produce an output (synapse). Usually the sums of each node are weighted, and the sum is passed through a non-linear function known as an *activation function* or transfer function. The activation function is characterized by its shape and a *threshold* value.

Many artificial neurons can be used to construct an artificial neural network. Many neurons can be joined to form a layer. According to the number n of layers needed to perform a given function, people call it a *n-layer neural network* .

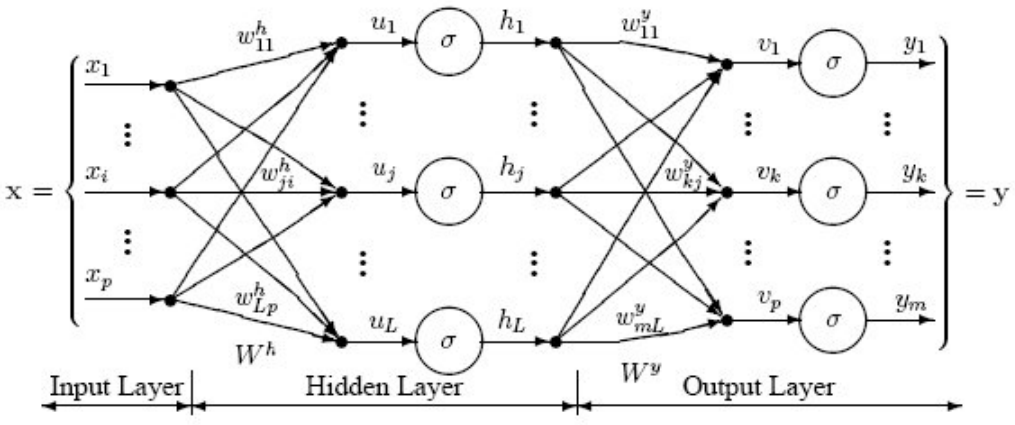

FIGURE 2.2 – Artificial Neural Network with Hidden Layers
Source : *http ://www.dtreg.com*

Feed-forward networks have the following characteristics :

1. Neurons are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called *hidden layers* .

2. Each neuron in one layer is connected to every neuron on the next layer. Hence information is constantly "fed forward" from one layer to the next, and this explains why these networks are called feed-forward networks.
3. There are no connections among neurons in the same layer.

The first artificial neuron was the Threshold Logic Unit (TLU) first proposed by *Warren McCulloch* and *Walter Pitts* in 1943 [18]. As a transfer function, it employed a threshold, which is equivalent to using the Heaviside step function. Initially, only a simple model was considered, with binary inputs and outputs, some restrictions on the possible weights, and a more flexible threshold value. Since the beginning it has already been noticed that *any boolean function could be implemented by networks of such devices*, what is easily seen from the fact that one can implement the AND and OR functions, and use them in the disjunctive or the conjunctive normal form.

2.2 The Perceptron

One important and pioneering artificial neural network that used the linear threshold function was the *perceptron* , developed by *Frank Rosenblatt* in 1958 [23].

2.2.1 General definition

The perceptron of size N is defined by a set of *synaptic weights* $\{w_i\}_{1 \leqslant i \leqslant n}$, and a *threshold* b. It takes a vector of size N, $\vec{x}$ as input, and outputs $f(x)$. The values of w_i, x_i, b can be taken in any real set.

$$f(x) = \begin{cases} 0 & \text{if } \vec{w}.\vec{x} < \text{b} \\ 1 & \textit{otherwise} \end{cases} \tag{2.1}$$

So the basic action of a perceptron is to attribute a weight to every input x_i , and see if the resulting quantity is greater (outputs a 1) or less (outputs a 0) than the threshold b.

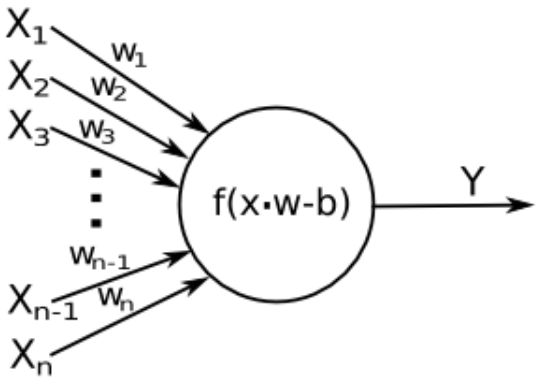

FIGURE 2.3 – Perceptron with threshold b

Example Given a perceptron of size 2 with the following weights and threshold : $w_1 = 2$, $w_0 = -0.4$, $b = 10$, we would have the following inputs/outputs :

x_1	x_2	$f(\vec{x})$
0	0	0
4	0	0
10	2	1
6	10	0

Nevertheless, people usually use the perceptron in a neural network, so the output of one perceptron could be the input of a perceptron on the following layer. The value of the input will then be restricted to $\{0,1\}^{\mathbb{N}}$. Moreover, people include most of the time the threshold to an additional input line $x_0 = 1$ whose weight is always $-b$. As the weights are still real ($w_i \in \mathbb{R}$) we call that perceptron : a *continuous perceptron* . The perceptron's function then become :

$$f(x) = \begin{cases} 0 & \text{if } \vec{w}.\vec{x} - \text{b} < 0 \\ 1 & otherwise \end{cases} \tag{2.2}$$

And the perceptron would be represented by :

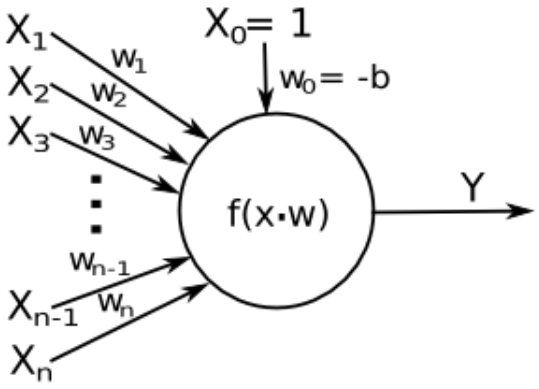

FIGURE 2.4 – Perceptron without threshold

2.2.2 Properties

Since their discovery, these continuous perceptrons have been studied very broadly in literature [32]. In this section we will give only one basic property of these perceptrons : the universality of representation for every mapping from $\{0,1\}^k \to \{0,1\}^p$, $k, p \in \mathbb{N}$. This result will allow us to introduce the advantage of studying quantum perceptron.

2.2.3 Function linearly separable

Let us first introduce the concept of objects *linearly separable* .

Definition 1:
Two sets A and B of points in an n-dimensional space are called absolutely linearly separable if $n+1$ real numbers $w_1, ..., w_{n+1}$ exist such that
every point $(x_1, x_2, ..., x_n) \in$ A satisfies :

$$\sum_{i=1}^{n} w_i x_i > w_{n+1}$$

every point $(x_1, x_2, ..., x_n) \in$ B satisfies :

$$\sum_{i=1}^{n} w_i x_i < w_{n+1}$$

Geometrically it is easy to see that these two sets A and B in a $n-space$ would be linearly-separable only if there exists a $(n-1)$-hyperplane that separates the two sets. In the 2-dimension space, this would be a line separating two set of points. This will allow us to prove one property of the classical perceptron below.

Given the previous definition it is obvious to see that a continuous perceptron could compute every *linearly separable function* (a function that defines two linearly separable spaces) given we adapt its weights in the correct way.

Example The function **AND** is a linearly-separable function of the $2-space$ for the set of points $\{(0,0),(0,1),(1,0)\}$ and $\{(1,1)\}$. The following perceptron can be used to compute this function : $w_1 = 1$, $w_2 = 1$, $b = 1,5$,

x_1	x_2	$f(\vec{x})$
0	0	0
0	1	0
1	0	0
1	1	1

The linear function that separates the correct set from the wrong set is :

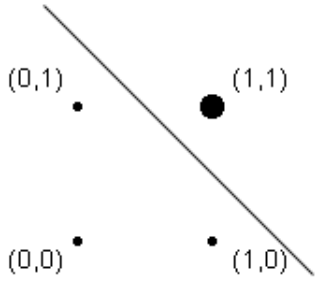

FIGURE 2.5 – A linear separation for the AND function

Most of the time it's more easy to take the input set as $\{-1;1\}$ instead of $\{0;1\}$. We will get the same result for the function **AND** by adjusting the threshold : $b = 0$.

2.2.4 Performing every mapping of $\{0;1\}$

But, if one tries all the different functions of $\{0,1\}^2 \rightarrow \{0,1\}^2$, one would discover that some of them are not computable with a perceptron. That is the case of the **XOR**. This function is not linearly separable.

So people could wonder about the interest of a single continuous perceptron if it does not suceed in performing a simple function as the **XOR** ($\oplus$). The point is that, as already mentionned, people can associate, in different following layers, different perceptrons. And we have seen that the perceptron could perform the **AND** ($\wedge$)and the **OR** ($\vee$) (both linearly separable).

One of the success of Boolean algebra is that you can write every single function, with any number of variables, as a *conjunctive normal form* . A statement is in conjunctive normal form if it is a conjunction (sequence of ANDs) consisting of one or more conjuncts, each of which is a disjunction (OR) of one or more literals. [2].

2. For an interesting ressource, see : http://www10.wolframalpha.com/input/?i=boolean+algebra

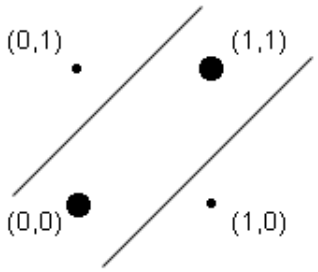

FIGURE 2.6 – The XOR function is not linearly separable

Example | $x \oplus y = (\bar{x} \vee \bar{y}) \wedge (x \vee y)$

Then, we see that we could use a three-layer perceptron to perform the XOR function.

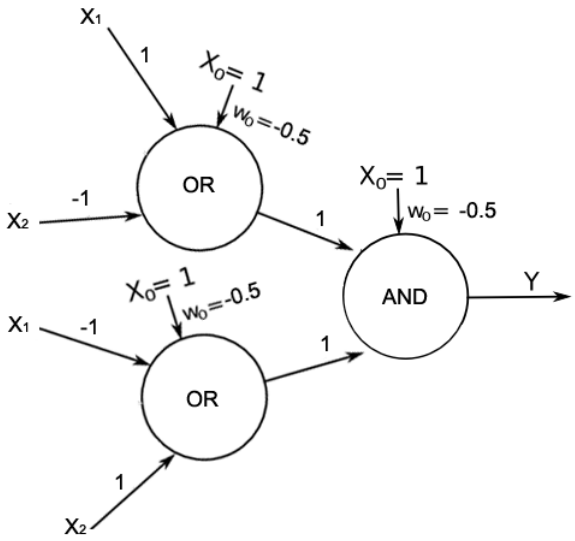

FIGURE 2.7 – The layered version of the XOR

Therefore we have the following theorem :

Theorem 2. *Any mapping of* $\{0,1\}^k \to \{0,1\}^p$ $k, p \in \mathbb{N}$ *can be performed by a i-layer perceptron* ($i \in \mathbb{N}$).

From the definition 1 we could modify the theorem 2 and go a little further :

Theorem 3. *Any mapping of* $\{0,1\}^k \to \{0,1\}^p$ $k, p \in \mathbb{N}$ *can be performed by a i-layer perceptron* ($i \in \mathbb{N}$) *with a 0 threshold.*

Démonstration. One first has to decompose the intial function to get its conjunctive normal form. After that, for all couple of conjunction or disjunction you had a layer. Another proof, more complicated can be found in [5]. A concrete realization can be found in [16]. □

But as the number of inputs increases, the number of layers increases too. Quantum computation allowed us to have some improvement on that kind of result. Indeed, [15, 30, 33] showed that a quantum system can perform the XOR function thanks to quantum properties.

2.2.5 The Discrete Perceptron

In the previous section we only studied continuous valued weights. But as the precision of modern computer is still limited to discrete values (even for floating-point numbers) the behaviour of *discrete perceptrons* has been investigated. In the next two sections we define the two discrete perceptrons we have been led to study.

The Ising Perceptron

We call *Ising perceptron* a perceptron whose inputs, output, weights are taken in $\{-1, +1\}$. As we previously stated, we do not need to define a threshold because for any function performing a mapping $\{0,1\}^k \to \{0,1\}^p,\ \ k, p \in \mathbb{N}$ one can find such a perceptron with a 0 threshold.

The Clipped Perceptron

The *Clipped perceptron* is a discrete perceptron defined by a *synaptic depth* $L \in \mathbb{N}$ and by a continuous perceptron J (whose weights are $\{J_i\}_{1 \leqslant i \leqslant N}$). The clipped perceptron's weights W_i can take values in $\{0, \pm\frac{1}{L}, ..., \pm 1\}$. In order to associate the continuous perceptron to the clipped one, we use the following *clipping process* : We first define limit values λ_l, which are arranged in increasing order. The limit values divide the continuous region of the precursor (the continuous perceptron) weight vector components to $2L+1$ intervals. The clipping process is such that J_i is mapped onto $\frac{l}{L}$.

$$w_i = \sum_{l=-L}^{L} \frac{l}{L} \left(\theta(\lambda_{l+1} - J_i) - \theta(\lambda_l - J_i)\right), \tag{2.3}$$

where θ is the Heaviside function.

Chapitre 3

The Learning algorithm

In the previous parts we have defined the perceptron and we saw that it could be defined only by its weights. The applications in which it is used often require a lot of inputs, and thus, programming a perceptron is a bit slow if one wants to do it manually. People therefore use a process similar to human learning and called *perceptron learning* . The main idea is that one gives the perceptron a set of inputs and the expected answers, and the perceptron tries to adapt its weights to approximate the assumed function.

3.1 The Classical Algorithm

3.1.1 General Case

Notation

We call N the *size of a perceptron* . When a perceptron tries to learn it uses a set of examples $(\vec{x}, b) \in (\mathbb{Z}/2\mathbb{Z})^N \times (\mathbb{Z}/2\mathbb{Z})$. We call Ω the set of all examples that the perceptron can study.

The perceptron only knows the set of input/answers given to him through Ω. People often call *teacher* the ideal perceptron that could give the correct answers to the *student* who is learning. The weights of the teacher are the ones we want the student to have learnt at the end of the process. In the following we will use the upper script ^{T} for the teacher and ^{S} for the student.

If we study a student perceptron of *size* N we call $\vec{w^S}$ its weights and $F^S(\vec{x})$ the answer of the perceptron to the input $\vec{x}$, a vector of size N. In the same way we have $\vec{w^T}$ and F^T.

In order to know if a student has learnt more or less something, we introduce the concept of *overlap* .

$$R = \frac{w^T.w^S}{N} \tag{3.1}$$

The letter R stands for *Recognition* from the first definition of this vector given by Vallet for a general case [28]. The closer the student's and the teacher's weights are, the more R tends to 1. Another definition, based on the number of correct answer is possible when the teacher is unknown (most of the case). In these cases, R is just the probability that the student gives the good answer. Once again, $R \to 1$ when the student learns.

We call *perfect learning* the situation $R = 1$.

Goal of the Learning

As we previously said, if we try to *create a perceptron by hand* we would hardly be able to apportion all the weights by ourselves. As soon as $N = 5$ it is almost impossible, and it is not rare than in the usual applications, people study hundreds of perceptrons with at least 50 input bits. People then found a solution that adds a new perspective to computation. The perceptron's ability to learn allows

people to avoid choosing arbitrarily the weights, and also give a extra feature to the perceptron : *generalization* .

Indeed, the set of examples Ω could be different than $(\mathbb{Z}/2\mathbb{Z})^N \times (\mathbb{Z}/2\mathbb{Z})$, but the perceptron could have to answer to answer to inputs $x \notin \Omega$. We could actually define another function, the *generalization function* [31] G, which would be the equivalent of R over the set of inputs that the perceptron could encounter. Obviously $G \leqslant R$. But in this report we will consider that the generalisation set is equal to Ω. We would just have to take care of R.

The learning will then be a situation in which we initiate arbitrary weights, and use something to update them according to the examples that they receive. Time steps will be noted μ. The updating process will be characterized by the function f.

The goal of the learning process is to update the student's weights in order for R to tend to 1.

3.1.2 Continuous Learning

Most of the research papers about perceptron learning deal with continuous perceptron. Two main categories of continuous learning are available : batch (off-line) learning (the most famous algorithm is Hebbian rule) and on-line learning (the most famous algorithm is gradient descent).

In the continuous learning, most of the time, the update depends on the *learning rate* $\eta \in \mathbb{R}$. Taking $\eta \ll 1$ allows to update weights with infinitesimal values.

In the *on-line learning* the weights are updated each time a training example from the training set Ω is presented, such that the error on this example is reduced.

Example The *gradient descent learning* is of the form :

$$\vec{w^S}^{\mu+1} = \vec{w}^{\mu} + \frac{\eta}{N} f\left(F^T(\vec{x}), F^S(\vec{x})\right) \tag{3.2}$$

In *batch learning* , the total error on all examples in the training set is accumulated before a a gradient descent weight update is made. For a given training set and starting weights, batch learning entirely deterministic. On-line learning, in the other hand, is a stochastic process due to the random choice of training example for each update.

Example In this example we will present the *hebbian learning* . This model has been presented in 1949 by the psychologist *Hebb* [12]. The hebbian learning is of the form :

$$\vec{w}^{Hebb} = \sum_{x \in \Omega} \vec{x} F^T(\vec{x}) \tag{3.3}$$

A lot of papers have been published on the subject, and we let the reader read the following reference book [5].

3.1.3 Discrete Learning

In discrete learning the perceptron's weights are discrete, so there cannot be infinitesimal updating of them. The updating step will then only modify weights in a discrete way : there exists no learning rate ν.

But, exactly like as the previous part, we will keep the definition of the overlap and its meaning (if $|R|$ tends to 1, then the student and the teacher almost agree on a given set of inputs/answers, if not, then they disagree). We will still try to increase the overlap during the learning process.

In order to achieve this result, different strategies are available. In most cases, the point is to try to imitate the teacher. Some procedures take account of the student answers, some procedures have memory about the previous choices, some update weights at random ...

Training by Clipping : using a Continuous Intermediate

The clipped version of a perceptron has been first introduced by *Van den Broeck* and *Bouten* in 1993 [29]. Strangely this kind of perceptron has not been deeply studied [2, 24, 29].

The strategy in that case is to use a continuous intermediate perceptron. The continuous perceptron will learn in a standard way, whereas the clipped perceptron will take discrete values based on the continuous' ones. As shown in [21] it could increase the speed of the learning.

Nevertheless, the most interesting result for our study has been published in [22]. In this paper it is shown that the behaviour of such a clipped perceptron is approximately the same as the continuous one if the number of values L that the weights can take is proportional to $\sqrt{N}$. We will not really give details about this result because the case of clipped perceptrons is not the one that interested us, but one will notice that the important theorem we will try to improve is quite similar.

Training without a Continuous Intermediate

More naturally we could define a discrete perceptron that learns without a continuous precursor. As the clipped perceptron, this kind of perceptron has not been studied much. The reason was probably that people used to study perceptrons with a formalism that allow only study of continuous ones. The first paper that tries to tackle the problem is dated 1993 [9].

Next we give the learning rule given in [14]. This model is the one that we tried to transform to a quantum one.

We denote by $^{\mu}$ the values of the variables at time step μ. In the following we will often use $\vec{w}$ and $\vec{w'}$ instead of $\vec{w^{\mu}}$, $\vec{w^{\mu+1}}$ (respectively) when no confusion is possible. The function f_u is the updating function : $(\{0,...,L\}^N \times (\mathbb{Z}/2\mathbb{Z})^{2+N}) \to \{0,...,L\}^N$

$$\vec{w'} = f(\vec{w}, F^T, F^S, \vec{x})$$
$$\vec{w^{\mu+1}} = f(\vec{w^{\mu}}, F^T, F^S, \vec{x})$$

In this part we will study the most simple law that could be used for discrete perceptron. We have first to remember that the weights are discrete numbers taken in the set $1,...,L$ $L \in \mathbb{N}$. The updating rule is the following :

$$w_i = \begin{cases} w_i & +x_i F^T(\vec{x}) \quad \text{if} \quad w_i + x_i F^T(\vec{x}) \in \{1,...,L\} \\ w_i & \text{otherwise} \end{cases} \tag{3.4}$$

At each step the weights $\vec{w}$ are updated and increase or decrease to L or 0. Some obvious questions then arise :

- Does that process converge (in term of clipping weights) and lead to learning ?
- What are the effect of each parameters ?
- How fast does it learn ?

In 1997, Kinzel and Urbanczik [14] proved that in order to learn, L should be proportional to $\sqrt{N}$. (see proof in appendix A). They also gave the speed of that learning.

3.2 The Quantum Algorithm

We will now study the perceptron as a quantum gate. Some recent research papers have been published about a possible quantum learning algorithm. Most of them use the vector's property of superposition and use a superposed example input vector [1, 17]. Two recent papers give an interesting way of experimentally demonstrating the concept of quantum learning [10, 19].

In our study we will apply more directly concepts from classical computing science and use a classical update algorithm. Nevertheless, this algorithm will become quantum thanks to some tricks exposed below.

We will call the student's gate B_N : it has a double vectors input $|w, x\rangle$ and outputs the standard perceptron's answer. From a technical aspect, $|x\rangle$ is composed of N qubits and $|w\rangle$ is composed of N L-qubits with the length $L = 2^l$.

Example If $N = 2$ and $L' = 3$, the weights could be $w_0 = 010, w_1 = 110$. Then we would study $|w = 010110\rangle$ According to the first part of this paper we know that $|w_c\rangle = |01\rangle$ where w_{ci} is the projection on 0 or 1 according to $\theta\left(w_i - \frac{L}{2}\right)$

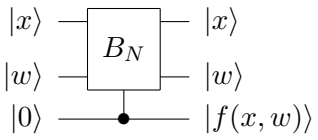

FIGURE 3.1 – The student's perceptron gate

In the same way, we define the teacher's gate T_N : it has a single vector input $|x\rangle$ and output the standard perceptron's answer.

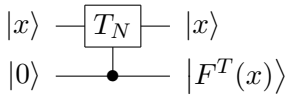

FIGURE 3.2 – The teacher's perceptron gate

We will now introduce the *learning matrix*, L_M. This matrix will take as inputs the weights of the student and update them. In order to properly update, it will need additional inputs according to the previous section. For the previous simple law we will just need the teacher answer and the example vector $|x\rangle$.

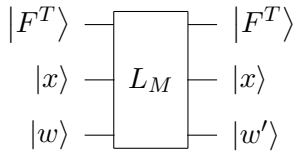

FIGURE 3.3 – The Learning Matrix gate

Nevertheless this matrix is the most simple object that one could design. It is obvious that the law could be adapted to a learning depending on a student's answer or some other extra bits (see the example of the buffer case bellow 3.2), as pictured in **??**.

According to the quantum computation requirements, the learning matrix must be unitary to describe a correct quantum gate. Obviously, one can note that this matrix associates the states $|F_T, x, w\rangle$ to $|F_T, x, w'\rangle$. If we use the previous updating function ($|w'\rangle = f_u(|w\rangle)$), we can see that the matrix looks like a permutation matrix apart from some exceptions.

Example $|F_T, x, w = L - 1\rangle \xrightarrow{f_u} |F_T, x, w' = L\rangle$
$|F_T, x, w = L\rangle \xrightarrow{f_u} |F_T, x, w' = L\rangle$
This two lines give the same updated weight, so the matrix is not a correct permutation.

We had then to find which modifications could give a permutation matrix. Two solutions can be found easily.

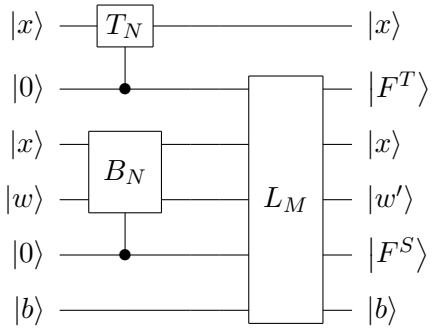

FIGURE 3.4 – The initial Learning Machine gate

Solution 1 :

$$\vec{w'} = w_i + x_i F_T(\vec{x}) \mod L \tag{3.5}$$

Solution 2 : We introduce *buffer bits* which will be markers for our algorithm. As soon as a weight reach a limit, the buffer takes the value 1. In the other cases it stays at 0. We denote them $\tilde{\mathbf{b}}$.

$$w_i = \begin{cases} w_i + x_i F_T(\vec{x}) & \text{if } w_i + x_i F_T(\vec{x}) \in \{1, ..., L\} \\ w_i & \text{otherwise} \end{cases} \tag{3.6}$$

$$b'_i = \begin{cases} 0 & \text{if } w_i + x_i F_T(\vec{x}) \in \{1, ..., L\} \\ 1 & \text{otherwise} \end{cases} \tag{3.7}$$

With these two solutions we get a permutation matrix that could be used as a quantum gate.

We will now give two simple examples about these two matrices. To understand them we just remind you that $|0\rangle = (1\ 0)$, $|1\rangle = (0\ 1)$ and $|xy\rangle = |x\rangle \otimes |y\rangle$

Example The swap case If $N = 1$ the structure of the input is $|F_T, x, w\rangle$ we get the following :

$$\begin{gathered} |0,0,0\rangle \to |0,0,1\rangle \\ |0,0,1\rangle \to |0,0,0\rangle \\ |0,1,0\rangle \to |0,1,1\rangle \\ ... \end{gathered}$$

The associated matrix (in the state space) is :

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

Example The buffer case If $N = 1$ the structure of the input is $|F_T, x, b, w\rangle$ we get the following :

$$|0,0,0,0\rangle \rightarrow |0,0,0,1\rangle$$
$$|0,0,0,1\rangle \rightarrow |0,0,1,1\rangle$$
$$...$$

The associated matrix (in the state space) is :

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

3.3 Simulation of the Quantum Version

3.3.1 Some preliminary reflexions

In order to compare the classical and quantum algorithm we need to simulate the quantum version. Some preliminary explanations are useful.

The difference between the classical and quantum version is due to the fact that qubits can be in quantum superposition of states. Instead of being only 0 or 1, people can use a combination of $|0\rangle$ and $|1\rangle$. In the classical case, the perceptron can receive only one bit at a time. In the quantum version, it can receive the combination of two (or more bits).

During a learning process, a perceptron uses at the same time : the example bits ($\vec{x}$), the teacher answer F_T and its own weights ($\vec{w}$). As the size of the example set is proportional to 2^N, the amount of different learning steps grows extremely fast with N. Moreover, the order of the example set is important. If you present examples in a different order, the result of the learning could be different.

By using the quantum property of superposition of states, you could send *at the same time* 2^N examples. You just have to send a normalised combination of all the possible examples in the appropriate state : $\sum_i \alpha_i |F_T(x_i), x_i, w_S\rangle$. The α_i could be adjust to give a greater or lesser importance to certain examples (**for instance, to examples that already agree with the student experience**).

The creation of the initial vector of examples is then an important step of the simulation. In the following we just used the most simple case (α_i is constant).

Another important fact to note is the following. The learning procedure is really simple as soon as one has the initial vector $\xi^{\mu=0}$ and the learning matrix (M_l). The update just rely on a matrix/vector multiplication : $\xi^{\mu+1} = M_l \xi^{\mu}$.

As both matrices are permutations, we have the following property : $M_l^s = I_s$ where s is the size of the matrix. Indeed, after s steps the permutation leads to the identity.

The learning process cannot then converge if we just apply successively the matrix to a given initial vector ! We need to add an extra step to make the process different from a real permutation. We only found it for the buffer case, so we will first study this case.

Note added : As the clearing of the buffer bits may not be unitary, and according to the results presented in a following sections, we would study an intermediate matrix between the swap-case matrix and the buffer-case matrix. This matrix will have $L = 3$, and the weights will be initialized at $w_i = 1$. For more details, read 3.4.

3.3.2 Description of the Simulation Algorithm

The simulation on a classical computer takes two arguments : the number of perceptron's inputs N and the precision of each weight L. According to the previous section, we will only study here the perceptron case.

The simulation is divided in three main parts :

1. generation of the matrix
2. generation of the example
3. the learning process

Generation of the Learning Matrix

Given N and L we associate every vector from $0, 1^{2^{1+N(2+L)}}$ to its associate update vector (see example 3.2). The procedure uses a simple *for loop* over all the possible vectors, and call a process *update_buffer*. This process transform the vector into the associated F_T, x, b, w and returns F_T, x, b, w' according to the following rule :

If $b_i = 0$

$$w_i' = \begin{cases} w_i + x_i F_T(\vec{x}) & \text{if } w_i + x_i F_T(\vec{x}) \in \{1, ..., L\} \\ w_i & \text{otherwise} \end{cases} \tag{3.8}$$

$$b_i' = \begin{cases} 0 & \text{if } w_i + x_i F_T(\vec{x}) \in \{1, ..., L\} \\ 1 & \text{otherwise} \end{cases} \tag{3.9}$$

If $b_i = 1$

$$w_i' = \begin{cases} w_i & \text{if } w_i + x_i F_T(\vec{x}) \in \{1, ..., L\} \\ w_i \bmod L & \text{otherwise} \end{cases} \tag{3.10}$$

$$b_i' = \begin{cases} 1 & \text{if } w_i + x_i F_T(\vec{x}) \in \{1, ..., L\} \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

After that we have the permutation vector corresponding to the states space. For instance, for $L = N = 1$ we have : (1,3,2,0,6,4,5,7,10,8,9,11,13,15,14,12). This vector will allow us to construct the **learning matrix**.

Generation of the Example

As we are studying the most simple case we will construct the example ξ containing the same repartition between every input/answer given by the teacher. We use a process *teacher* which has its own fixed weights and which take as input a vector $\vec{x}$ and returns $F_T(\vec{x})$. We compute all the possible outputs possible for $\vec{x} \in (\mathbb{Z}/2\mathbb{Z})^N$.

Our process then computes : *initiate_vector*(N,L) $\rightarrow \sum_{x^k \in (\mathbb{Z}/2\mathbb{Z})^N} \left| F_T(x^k), x^k, \tilde{0}, \tilde{0} \right\rangle$. We have chosen to initiate the student weights at 0, but that's arbitrary.

We finally normalize the vector in order to have a correct state that we can transpose to the state space.

The Learning Process

We now have the learning matrix M_l and the initial vector ξ. The learning process is the following :

1. multiplication of $M_l \xi$
2. resetting of the buffer bits

After these two steps we have the updated Qubit.

3.4 Results and Improvements

3.4.1 Result of the basic simulation

If we start with the weights initiated at $\frac{L}{2}$ the learning matrix outputs a result that converges after L steps. So, if $L = 1$, it will give the correct weights with a probability of $P(N)$ after only one step.

Theorem 4. *If $N = 2k$ $k \in \mathbb{N}$ then the probability $P(N)$ of getting a correct answer at the stationary state is :*

$$P(N) = \sum_{i=0}^{k} \binom{2k-1}{i} + \sum_{i=k}^{2k} \binom{2k-1}{i} \tag{3.12}$$

If $N = 2k+1$ $k \in \mathbb{N}$ then the probability $P(N)$ of getting a correct answer at the stationary state is :

$$P(N) = \sum_{i=0}^{k-1} \binom{2k}{i} + \sum_{i=k+1}^{2k} \binom{2k}{i} \tag{3.13}$$

Démonstration. Without loss of generality we can assume that $w_i^T = 1 \forall i$, and we can just focus on x_0 for getting the value of w_0^S. As the probability distribution is the same for the other weights there is no difference.

If N = 2k :

If $x_0 = 0$ there is $\sum_{i=0}^{k} \binom{2k-1}{i}$ distributions of the x-coordinates to have $F^T(\vec{x}) = 0$ (and then $w_0^S = 1$ after the update).

If $x_0 = 1$ there is $\sum_{i=k}^{2k} \binom{2k-1}{i}$ distributions of the x-coordinates to have $F^T(\vec{x}) = 1$ (and then $w_0^S = 1$ after the update).

If $\mathbf{N = 2k + 1}$:

If $x_0 = 0$ there is $\sum_{i=0}^{k-1} \binom{2k}{i}$ distributions of the x-coordinates to have $F^T(\vec{x}) = 0$) (and then $w_0^S = 1$ after the update).

If $x_0 = 1$ there is $\sum_{i=k+1}^{2k} \binom{2k}{i}$ distributions of the x-coordinates to have $F^T(\vec{x}) = 1$) (and then $w_0^S = 1$ after the update).

□

Therefore, if $N \to \infty$ the result is not really interesting because the limit of the distribution leads to $\frac{1}{2}$. We only have found the classical result.

3.4.2 Improvements

But fortunately, this result helps us to understand how to improve the process, and to find some ways to improve the result.

Changing the Learning Rule

As the final state is immediately reached if $L = 1$, and because the clearing of the buffer bits may not be unitary (some piece of information is lost during the process, forbidding it to be described by a unitary matrix) the most obvious improvement deals with the matrix/learning rule. In order to improve it while remaining simple, we will choose to use the swap rule, with $L = 3$ and only one learning step. The weight will be initialized at $w_i = 1$ $(L/2)$. Then after one step, the stationary state will be reach, without experiencing any swap event nor having to clear buffer bits.

Active Learning

As we have seen in the proof, we know which are the vectors $\vec{x}$ that give us a good learning result and which are the ones that give us a wrong result.

This process is called *active learning* . To our knowledge there exist no results about the speed of an active discrete Hebbian learning process. By our matrix representation and using the proof 3.4.1 we see that, if one changes the updating weights in the opposite direction for the wrong cases, then we will get a perfect learning in only one step. Yet this modification imply that we know the teacher before the learning.

Some improvements can be made in that direction but we are more confident in the following one : modification of the weight's phase.

Complex Amplitudes

There is another improvement that could be made with the concern of the value in each update. We could keep a discrete quantum perceptron, but update it with phase-modified values. So, for instance : $|\varphi\rangle \to |e^{i\alpha}\varphi\rangle$. The behaviour has no equivalent in the classical case.

The way this modification can happen follows the previous idea of active learning. The more the coordinates x_i are the same as the updated weight the less the phase will be important. In the opposite, if most of the coordinates are the same, the phase will tends to 0.

The phase could be the following : $\varphi_j = \frac{2\pi \,\mathrm{Card}(x_i \neq x_j)}{N}$ and $w'_j = e^{i\varphi_j}(w_i + x_i F_T(\vec{x}))$

Quantum Random Walk

Another ideas could be to use the difference between classical and quantum random walks. The behaviour of quantum random walks has been known for improving some algorithms [13].

Our next step will be to use one of these properties to reduce the size of the learning matrix.

Inputs modifications

Instead of inputing pure states we will try to input entangled states, and see if we can learn with less elements in the example set Ω. As the property is linked to the number of examples that the teacher could output, this is this element that we will try to target.

Annexe A

Proof of the synaptic's depth theorem - Kinzel and Urbanczik

A.1 Introduction

A.1.1 Notations

All the definitions are taken according to their paper. We only give some precision here about the calculations.

The teacher B^T is an Ising perceptron with binary couplings : $B_i \in \{-1, 1\}$. We call ξ the input, and $\sigma_B(\xi)$ the classification bit (the teacher's answer. $\sigma_B(\xi) \in \{-1, 1\}$. The student J is a discrete perceptron with synaptic depth L : $J_i \in \{1, ..., L\}$.

The clipping process is the following : $\tilde{J}_i = \text{sign}(J_i - L/2)$.

The overlap R is given by : $R = \dfrac{B^T.\tilde{J}}{N}$

A.1.2 Hebbian learning

In this chapter we are studying one of the most basic learning rule, the *Hebbian learning rule* :

$$J_i' = \begin{cases} J_i + \xi_i \sigma_B(\xi) & if J_i + \xi_i \sigma_B(\xi) \in \{1, ..., L\} \\ J_i & else \end{cases} \tag{A.1}$$

A.1.3 Result

The first result we want to show is the following :

If L is fixed and N tends to ∞ then the stationnary overlap R^s between the teacher and the clipped student will tends to 0 :

Theorem 5. $R^s \xrightarrow[N\to\infty]{} 0$

The second result deals with the minimum synaptic depths that perceptron must have to gain a non-zero overlap :

Theorem 6. *If* $L = \lambda\sqrt{N}$ *then* $R^s \xrightarrow[N\to\infty]{} 1 - \dfrac{1}{1 + e^{\lambda\sqrt{8/\pi}}}$.

A.2 Proof of the theorems

A.2.1 Random Walk

According to the Hebbian rule A.1 the increments $\xi_i \sigma_B(\xi)$ are not independent over the sites i and their covariances do decay as $\frac{1}{N}$. So, for large N we study a *biaised random walk* whose biais is :

$$< \xi_i \sigma_b(\xi) >_\xi = B_i \sqrt{2} \sqrt{\frac{1}{\pi N}} \tag{A.2}$$

We then study the random walk given by the reflecting barrier : g_i and r_i, $g_i = \frac{1}{2} + \frac{1}{\sqrt{2\pi N}}$ and $g_i + r_i = 1$.

Indeed, if we use the same notation as *Kinzel* we have $p_l^i(t)$ the probability that $J_i = l$ at time t. We then have :

$$\begin{aligned} p_1^i(t+1) &= r_i p_1^i(t) + r_i p_2^i(t) \\ p_l^i(t+1) &= g_i p_{l-1}^i(t) + r_i p_{l+1}^i(t) \\ p_L^i(t+1) &= g_i p_{L-1}^i(t) + g_i p_L^i(t) \end{aligned}$$

The stationnary solution of that random walk is :

$$p_l{}^{is} = \left(\frac{g_i}{r}\right)^{l-1} \left(\frac{g_i}{r_i} - 1\right) \left(\left(\frac{g_i}{r_i}\right)^L - 1\right)^{-1} \tag{A.3}$$

A.2.2 Fixed L

We can now compute the expected value of the student : $\left\langle \ \tilde{J}_i \ \right\rangle = \sum_{l=1}^{L} l p_l{}^{is}$. By replacing p_l^i by its value (A.3) we get :

$$< \tilde{J}_i >= \sum_{l=1}^{L} l p_l{}^{is} \xrightarrow[N \to \infty]{} \frac{L}{2} + \frac{1}{2} \tag{A.4}$$

As this value is exactly the middle of the interval $[1; L]$ we have prooved the first theorem.

A.2.3 L proportionnal to $\sqrt{N}$

For this proof, we need to give the exact value of the stationnary : $R^s = \frac{B^T \tilde{J}^s}{N}$:

$$R^s = \frac{1}{N} \sum_{i=1}^{N} B_i \text{sign} \left(\frac{\left(L \frac{1/2+1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}}{1/2-1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}} - L - 1 \right) \left(\frac{1/2+1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}}{1/2-1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}} \right)^L + 1}{\left(\frac{1/2+1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}}{1/2-1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}} - 1 \right) \left(\left(\frac{1/2+1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}}{1/2-1/2 \frac{B_i\sqrt{2}}{\sqrt{\pi L^{\beta-1}}}} \right)^L - 1 \right)} - \frac{L}{2} \right) \tag{A.5}$$

If we write :

$$\begin{aligned} p_{++} &= P(J_i > 1 | B_i = 1) \\ p_{+-} &= P(J_i < 1 | B_i = 1) \\ p_{-+} &= P(J_i > 1 | B_i = -1) \\ p_{--} &= P(J_i < 1 | B_i = -1) \end{aligned}$$

We get : $RN = \dfrac{(p_{++} + p_{--} - p_{+-} - p_{-+})\, N}{4}$ According to the previous section, one can rewrite the p's as :

$$\begin{aligned} p_{++} &= \sum_{k=L/2+1}^{L} p_k^- \\ p_{+-} &= \sum_{k=1}^{L/2} p_k^+ \\ p_{-+} &= \sum_{k=L/2+1}^{L} p_k^- \\ p_{--} &= \sum_{k=1}^{L/2} p_k^- \end{aligned}$$

Where ${p_l}^+ = \left(\dfrac{g_+}{r}\right)^{l-1} \left(\dfrac{g_+}{r_+} - 1\right) \left(\left(\dfrac{g_+}{r_+}\right)^L - 1\right)^{-1}$ and the plus indice $_+$ indicates that B_i=1. In the same way, the minus indice $_-$ indicates you have to take B_i=-1.

The overlap could be simplified : $R = 1 - (p_{+-} + p_{-+})$. After some straightforward calculations one gets :

$$p_{+-} + p_{-+} = \frac{x^{1/2\,L} y^L - x^{1/2\,L} - 2\, y^L + 1 + y^L x^L - y^{1/2\,L} x^L + y^{1/2\,L}}{(x^L - 1)\,(y^L - 1)}$$

where $x = \dfrac{g_+}{r_+}$ and $x = \dfrac{g_-}{r_-}$. As $y = 1/x$, one gets finally :

$$\frac{x^{\frac{1}{2}\ L} - 1}{x^L - 1} + \frac{\left(\frac{1}{x}\right)^L - \left(\frac{1}{x}\right)^{\frac{1}{2}\ L}}{\left(\frac{1}{x}\right)^L - 1}$$

If we replace x by its real value, and if we use $L = \lambda N^{1/\alpha}$, we get the following (monstruous)

expression :

$$p_{+-}+p_{-+}=\frac{\left(\left(\frac{\left(1/2-1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}\right)}{1/2+1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}}\right)^{L}-\left(\frac{\left(1/2-1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}\right)}{1/2+1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}}\right)^{1/2\,L}\right)}{\left(\frac{\left(1/2-1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}\right)}{1/2+1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}}\right)^{L}-1}$$

$$+\frac{\left(\frac{\left(1/2+1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}\right)}{1/2-1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}}\right)^{1/2\,L}-1}{\left(\frac{\left(1/2+1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}\right)}{1/2-1/2\sqrt{2}\frac{1}{\sqrt{\pi}}\left(\left(\frac{L}{\lambda}\right)^{1/2\,\alpha-1}\right)^{-1}}\right)^{L}-1}$$

Fortunately the limit is easy to discover when $\xrightarrow[N\to\infty]{}$. We get the following result :

Value of α	$\alpha<\frac{1}{2}$	$\alpha=\frac{1}{2}$	$\alpha>\frac{1}{2}$
$\xrightarrow[L\to\infty]{\lim}(p_{+-}+p_{-+})$	1	$\frac{2}{1+e^{\lambda\sqrt{\frac{2}{\pi}}}}$	0

Bibliographie

[1] Jeongho Bang, James Lim, M. S. Kim, and Jinhyoung Lee. Quantum learning machine, 2008. URL http://www.citebase.org/abstract?id=oai:arXiv.org:0803.2976.

[2] D. Bolle and GM Shim. Nonlinear hebbian training of the perceptron. *Network : Computation in Neural Systems*, 6(4) :619–633, 1995.

[3] I.L. Chuang, L.M.K. Vandersypen, X. Zhou, D.W. Leung, and S. Lloyd. Experimental realization of a quantum algorithm. *Nature*, 393 :143–146, 1998. doi : http://dx.doi.org/10.1038/30181.

[4] A. Church. A note on the entscheidungsproblem. *Journal of Symbolic Logic*, 1(1) :40–41, 1936.

[5] ACC Coolen, R. Kühn, and P. Sollich. *Theory of neural information processing systems*. Oxford University Press, USA, 2005.

[6] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, pages 97–117, 1985. URL http://physics.princeton.edu/~mcdonald/examples/QM/deutsch_prsl_a400_97_85.pdf.

[7] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings : Mathematical and Physical Sciences*, pages 553–558, 1992.

[8] R.P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6) :467–488, 1982.

[9] JF Fontanari and R. Meir. The statistical mechanics of the ising perceptron. *Journal of Physics A : Mathematical and General*, 26 :1077–1089, 1993.

[10] S. Gammelmark and K. Molmer. Quantum learning by measurement and feedback. *New Journal of Physics*, 11(3) :033017–+, March 2009. doi : 10.1088/1367-2630/11/3/033017.

[11] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79 :325, 1997. URL doi:10.1103/PhysRevLett.79.325.

[12] D.O. Hebb. *The organisation of behaviour*. 1949.

[13] Julia Kempe. Quantum random walks hit exponentially faster. *Probability Theory and Related Fields*, 133 :215, 2005. URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0205083.

[14] W Kinzel and R Urbanczik. On-line learning in a discrete state space. *Journal of Physics A : Mathematical and General*, 31(1) :L27–L30, 1998. URL http://stacks.iop.org/0305-4470/31/L27.

[15] M. Lewenstein. Quantum perceptrons. *Journal of Modern Optics*, 41(12) :2491–2501, 1994. ISSN 0950-0340. URL http://www.informaworld.com/10.1080/09500349414552331.

[16] R. Liu and Z.H. Chai. Realization of boolean functions by perceptron algorithm. pages 704–705, 1990.

[17] Daniel Manzano, Marcin Pawlowski, and Caslav Brukner. The speed of quantum and classical learning for performing the k-th root of not, 2009. URL http://www.citebase.org/abstract?id=oai:arXiv.org:0904.4571.

[18] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4) :115–133, December 1943. URL http://dx.doi.org/10.1007/BF02478259.

[19] Rodion Neigovzen, Jorge L. Neves, Rudolf Sollacher, and Steffen J. Glaser. Quantum pattern recognition with liquid-state nuclear magnetic resonance. *Physical Review A (Atomic, Molecular, and Optical Physics)*, 79(4) :042321, 2009. doi : 10.1103/PhysRevA.79.042321. URL http://link.aps.org/abstract/PRA/v79/e042321.

[20] MA Nielsen and IL Chang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[21] M. Rosen-Zvi. On-line learning in the ising perceptron. *Journal of Physics A : Mathematical and General*, 33(41) :7277–7287, 2000.

[22] M. Rosen-Zvi and I. Kanter. Training a perceptron in a discrete weight space. *Pre*, 64(4) : 046109–+, October 2001. doi : 10.1103/PhysRevE.64.046109.

[23] F. Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386–408, 1958. URL http://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

[24] J. Schietse, M. Bouten, and C. Van den Broeck. Training binary perceptrons by clipping. *Europhysics letters*, 32 :279–279, 1995.

[25] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J.SCI.STATIST.COMPUT.*, 26 :1484, 1997. URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/9508027.

[26] M. Sipser. *Introduction to the Theory of Computation*, volume 1. ACM New York, NY, USA, 1996.

[27] A.M. Turing. On computable numbers : With an application to the entscheidungsproblem. 1936.

[28] F. Vallet. The hebb rule for learning linearly separable boolean functions : Learning and generalization. *Europhys. Lett*, 8(8) :747–751, 1989.

[29] C. Van den Broeck and M. Bouten. Clipped-hebbian training of the perceptron. *EPL (Europhysics Letters)*, 22 :223–229, 1993.

[30] Dan Ventura and Tony Martinez. An artificial neuron with quantum mechanical properties. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 482–485, 1997.

[31] TLH Watkin. Optimal learning with a neural network. *Europhysics Letters*, 21 :871–871, 1993.

[32] B. Widrow and M.A. Lehr. 30 years of adaptive neural networks : Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9) :1415–1442, 1990.

[33] Rigui Zhou, Ling Qin, and Nan Jiang. Quantum perceptron network, 2006. URL http://dx.doi.org/10.1007/11840817_68.

Index