Imperial College London
Department of Computing

Learning Boolean Functions with Multi-Controlled X Gates

Viet Pham Ngoc

Supervised by Dr Herbert Wiklicky

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing and
the Diploma of Imperial College, August 2023

Declaration of Originality

I hereby declare that this thesis, the research it describes as well as the materials used for illustration and representation are the product of my own work. Where applicable, the works by other researchers have been clearly and appropriately referenced.

This thesis has not been submitted for any other purpose.

Copyright Declaration

'Do you guys just put the word quantum in front of everything?'
Scott Lang

'Cho mẹ. Đừng lo, con đi tới cùng'

Acknowledgements

First and foremost, I would like to extend my heartfelt gratitude to my family without whom I most certainly would not be the person I am today. Each person played their part in my upbringing and hence in this PhD. I would like to thank my father, for having instilled in me his love of science, my sister, for keeping my feet on the ground by constantly reminding me how much of a "gros caca" I am and finally my mother, for having been the constant, reliable, nourishing, loving and supportive (sometimes too loving and supportive) presence that only a mother can provide.

I would like to thank my supervisor Dr Herbert Wiklicky for his support and guidance throughout my PhD. Our weekly meetings will be missed.

A special thanks goes to Amani El-Kholy, without whom the life at Imperial would have been incommensurably more difficult.

I also would like to thank my flatmates: Sacha, then Sacha, Pierre and John followed by Sacha, Pierre and Manon then Sacha, Manon and Dominika and finally Sacha, Manon and Pierre again. They truly created a place that I can call home. A home within which we could host dinners with the friends we met along the way: Rodrigo, Miguel, Luca, Dominika, David, Mara, Benjamin, Anouck and their lovely daughter, Sofiane, Manon, Oriane, Borja and Raquel for those who stayed in London. John, Camille, Camille (not a typo) and Claire for those who left.

My final thanks go to Dominika for teaching me about parallelisation.

Abstract

As of late, both the fields of quantum computing and machine learning have experienced simultaneous developments. It is thus naturally that the interplay between these two fields is being investigated with the hope that they could benefit from one another. In this thesis, we explore one of the facets of this union called quantum machine learning. More accurately, throughout this thesis, the aim will be to learn Boolean functions using quantum circuits.

To do so, we first study a type of circuit, that we named tunable quantum neural network, exclusively made of multi-controlled $\mathbf{X}$ gates and we formally show that this type of circuit is able to express any Boolean function, provided that it is tuned correctly. We then devise a learning algorithm, that makes use of a specific quantum superposition to identify misclassified inputs. This algorithm intends to minimise the number of updates to the quantum circuit as it can be a costly operation. However, because of the large number of measurements required, it may not be practical.

To tackle this limitation and to guide our design of a learning algorithm that is indeed practical, we take advantage of the still ongoing field of quantum learning theory and design two other learning algorithms to be used in their respective framework. The first algorithm is used to train the network in the quantum probably approximately correct (QPAC) learning framework. By leveraging a quantum procedure called amplitude amplification, we show that this algorithm is efficient. The second algorithm also uses amplitude amplification but this time to train the network in the quantum exact learning framework with access to a uniform quantum example oracle. In both frameworks, we show that, in some cases, our algorithms perform better than what can be found in the literature.

Contents

List of Tables

List of Figures

Acronyms

ei error identification.

AA Amplitude Amplification.

ANF Algebraic Normal Form.

PAC Probably Approximately Correct.

QPAC Quantum Probably Approximately Correct.

TNN Tunable Quantum Neural Network.

VC dimension Vapnik-Chervonenkis dimension.

Glossary

$A \triangle B$ "The symmetric difference between two sets A and B defined as: $(A \setminus B) \cup (B \setminus A)$".

AND A Boolean binary operator that is True if and only if its two inputs are True.

Boolean A data type that takes two values. Generally True (or 1) or False (or 0).

Boolean function A function that takes as inputs Boolean variables and outputs a Boolean.

Complexity The complexity of an algorithm is the amount of resources required for it to run.

Confidence Interval A range of estimation for an unknown parameter, computed at a given confidence level.

Disjunctive Normal Form The representation of a Boolean function as an OR of ANDs.

Hamming weight The Hamming weight of a Boolean string is the number of elements of this string that are True (or 1).

Maximum likelihood estimation An estimation of the parameters of a probability distribution obtained by maximising the likelihood function.

OR A Boolean binary operator that is False if and only if its two inputs are False.

Oracle A black box that, in the case of this thesis, outputs a superposition of interest.

Permutation A permutation of a set is the rearrangement of the elements of this set.

Qiskit A software development kit for quantum computing developed by IBM.

XOR A Boolean binary operator that is True if and only if its two inputs have different values.

Chapter 1

Introduction

1.1 Motivation

Theorised in the early 1980's [27, 10], quantum computers promise to harness quantum phenomena, such as entanglement and superposition, to perform computations. This allows some quantum algorithms to perform the same task as their classical counterparts but with lower time complexity. This reduction in compute time is termed quantum speed-up and has been shown for numerous tasks, be it solving linear equations [37] or Monte-Carlo methods [52]. Until recently, these results were confined to theoretical results, but recent breakthroughs in quantum hardware have spurred the hope that they might one day become reality.

Simultaneously, the field of machine learning has experienced great advancements and success in many areas, spanning from natural language processing [79] to image recognition [39]. This rapid development has been driven by several factors. Chief among them is certainly the explosion of available data and the advancement of computation hardware. However, as the scale and complexity of the tasks increase, there is a possibility that this field may reach a limit [75]. It is thus necessary to explore more hardware-efficient algorithms [36] or novel hardware architectures altogether [58].

One proposed lead is the exploration of the interplay between quantum computing and machine

learning. This union of fields is known as quantum machine learning and covers different realities depending on the type of data and the type of algorithm being considered. In this thesis, our focus will be on performing machine learning tasks over classical data but with quantum algorithms. Many works have been done in that direction [69, 85, 60] with, for most of them, an emphasis on adapting existing classical methods to quantum computers [83, 74, 65]. As the goal is to exhibit a quantum speed-up, the performances of these methods have to be quantified. This is where the field of learning theory comes into play. This field is interested in studying the performances and limitations of learning algorithms and in doing so, it provides a rigorous framework to analyse such algorithms. Many learning models have been introduced, with two major models being: Probably Approximately Correct Learning (PAC-learning) [76] and Exact Learning [2]. As quantum machine learning algorithms emerged, so did the field of quantum learning theory [4]. The latter is predominantly populated with quantum counterparts to the classical models such as Quantum Probably Approximately Correct Learning (QPAC-Learning) [16].

While the field of quantum machine learning is promising, it is still hampered by the current state of quantum hardware which does not allow for practical implementations. With that in mind, we strived to develop a learning architecture made of simple gates and with low entanglement that can more easily be simulated on classical computers and potentially implemented on real hardware.

1.2 Objectives and Contributions

Given the above discussion, at the start of my PhD, my two objectives were:

1. Find a learning architecture made of simple gates and with low entanglement.
2. Study the architecture's resilience to noise.

But as my research went on, with the first objective completed, I decided to study the learning capacity of this architecture as it was something that had not been done before and so a new

objective came up:

3. Study the architecture in at least one learning theory framework.

With these objectives in mind, my contributions throughout my PhD are the followings:

1. In Chapter 3, I introduced a quantum circuit exclusively made of multi-controlled **X** gates acting on the same qubit. I then formally showed that this type of circuit is able to express any Boolean functions and named it tunable quantum neural network (TNN).

2. In Chapter 4, I designed an algorithm to train this architecture to learn any Boolean functions. This algorithm is based on a superposition of all the inputs where each of the inputs is associated with a specific amplitude. While successful, this algorithm required a number of samples that were impractical.

3. In Chapter 5, I turned to the QPAC framework to design an algorithm that works within this framework and proved that it is efficient. I then applied this algorithm to learn the class of parity functions. For this class of functions, the algorithm performs better, in some cases, than what can be found in the literature.

4. In Chapter 6, I studied this architecture in the exact learning framework and devised an algorithm that was then fine-tuned by training the architecture to learn generic functions and comparing the performances against a naive algorithm. This fine-tuned algorithm was then adapted to learn k-juntas. I found that under certain conditions, the algorithm exhibits better performances than the literature.

1.3 Publications

The following papers are the product of the research undertaken during my PhD:

1. **Tunable Quantum Neural Networks for Boolean Functions**
   Viet Pham Ngoc, Herbert Wiklicky. 2020.
   Presented as a poster at Quantum Techniques in Machine Learning (QTML) 2020.

2. **Tunable Quantum Neural Networks in the QPAC-Learning**

   Viet Pham Ngoc, David Tuckey, Herbert Wiklicky. 2022.

   Presented at Quantum Physics and Logic (QPL) 2022.

3. **Exactly Learning under the Uniform Distribution with Tunable Quantum Neural Networks and Amplitude Amplification**

   Viet Pham Ngoc, Herbert Wiklicky. 2023.

   Accepted as a poster at QTML 2023.

This thesis is based on these papers.

Chapter 2

Background

2.1 Introduction

This chapter intends to introduce the knowledge necessary for a reader to form their own understanding of the research presented in this thesis. While some knowledge of mathematics is required, no previous knowledge of quantum physics or quantum computing is necessary. This chapter will first deal with some notions of general algebra before delving into the mathematical framework for quantum computing. This will then lead to the presentation of the basics of this field. These two last parts are mostly inspired by [57], considered by many to be an essential textbook for anyone interested in the field of quantum computing. Concerning quantum computing, there exist many computing paradigms, among them are gate-based computing, adiabatic computing and measurement-based computing. In this thesis we will only focus on gate-based computing, hence we will only describe this model. We will also present some of the well-known quantum algorithms stemming from this model, amongst which we will detail the quantum amplitude amplification procedure.

2.2 Mathematical Background

2.2.1 Group

In Chapter 3, we will use the algebraic structure of groups that we define here.

Let G be a non-empty set, equipped with an inner operation $+ : G \times G \to G$. Then $(G, +)$ is said to be a group if this operation has the following properties:

- Associativity: $\forall a, b, c \in G,\ a + (b + c) = (a + c) + b$
- Neutral element: $\exists 0 \in G,\ \forall a \in G,\ 0 + a = a + 0 = a$
- Inverse element: $\forall a \in G,\ \exists b \in G,\ a + b = b + a = 0$
  This element is unique and is noted $-a$ for $a \in G$.

The definition of a group does not assume the commutativity of the inner operation, that is:

$$\forall a, b \in G,\ a + b = b + a \tag{2.1}$$

However, there exist groups for which this operation is commutative, in which case, these groups are called commutative groups or abelian groups.

Example 2.1:
Consider $(\mathcal{C}(\mathbb{R}), \circ)$ the set of continuous functions from $\mathbb{R}$ to $\mathbb{R}$, along with the composition operation. Then it is a group but it is not a commutative group. On the contrary, $(\mathbb{Z}, +)$ the set of the integers with the usual addition is an abelian group.

Subgroup generated by a set

Let $(G, +)$ be a group and S a subset of G. $\langle S \rangle$ is called the subgroup generated by S: it is the smallest subgroup of G containing S and is the subgroup made of combinations of a finite

number of elements of S or their inverse:

$$\langle S \rangle = \left\{ \sum_{i=1}^{m} (-1)^{\epsilon_i} s_i \mid m \in \mathbb{N}, \forall i,\ s_i \in S, \forall i,\ \epsilon_i \in \{0,1\} \right\} \tag{2.2}$$

Example 2.2:

Consider the group $(\mathbb{Z}, +)$, then $\mathbb{Z} = \langle 1 \rangle$.

Group morphism

Let $(G, +)$ and $(H, *)$ be two groups. A group morphism is a function $\phi : G \to H$ that preserves the structure of the two groups:

$$\forall a, b \in G,\ \phi(a + b) = \phi(a) * \phi(b) \tag{2.3}$$

A group morphism that is also bijective is called an isomorphism. If an isomorphism between two groups exists, then these two groups are said to be isomorphic in which case they can be considered as identical.

Galois field

Although this concept should have a section of its own, we will only briefly mention it at the start of Chapter 3 and Galois fields being related to groups we will define them here. Let F be a non empty set, together with the inner operations $+$ and $\times$, $(F, +, \times)$ is a field if:

- $(F, +)$ is a commutative group with neutral element 0
- $(F \setminus \{0\}, \times)$ is a commutative group with neutral element 1
- $\times$ is distributive over $+$: $\forall a, b, c \in F, a \times (b + c) = a \times b + a \times c$

A Galois field is then a finite field. If q is the cardinal of a Galois field, all finite fields of cardinal q are isomorphic to each other so the same denomination can be used: $\mathbf{F}_q$. The smallest Galois field is then $\mathbf{F}_2 = (\{0, 1\}, +, \times)$.

2.2.2 Finite Dimensional Hilbert Space

Here we present the concept of finite dimensional complex Hilbert spaces that form the mathematical framework for quantum mechanics and quantum computing. A complex vector space V is said to be a Hilbert space if it is equipped with an inner product and is a complete metric space with respect to the metric induced by this inner product. In the following, we will briefly present the notions that have been introduced in this definition.

Complex vector space

A non-empty set V is said to be a complex vector space if it is equipped with the two following operations:

1. Vector addition such that $(V, +)$ is a commutative group.

2. Scalar multiplication: $\mathbb{C} \times V \to V$ with the following properties:

   - Associativity:
   $$\forall \lambda, \mu \in \mathbb{C},\ \forall v \in V,\ \lambda(\mu v) = (\lambda\mu)v \tag{2.4}$$

   - Neutral element: if we denote 1, the identity element for the multiplication in $\mathbb{C}$ (that is the regular 1) then:
   $$\forall v \in V,\ 1v = v \tag{2.5}$$

   - Distributivity with regard to the vector addition:
   $$\forall \lambda \in \mathbb{C},\ \forall u, v \in V,\ \lambda(u + v) = \lambda u + \lambda v \tag{2.6}$$

   - Distributivity with regard to the scalar addition:
   $$\forall \lambda, \mu \in \mathbb{C},\ \forall v \in V,\ (\lambda + \mu)v = \lambda v + \mu v \tag{2.7}$$

A subset $S \subseteq V$ is said to be a spanning set for V if any vector of V can be expressed as a linear combination of vectors of S:

$$\forall v \in V,\ \exists u_1, \ldots, u_k \in S,\ \exists \lambda_1,\ \ldots \lambda_k \in \mathbb{C},\ v = \lambda_1 u_1 + \ldots + \lambda_k u_k \tag{2.8}$$

In addition, vectors $v_1, \ldots v_k \in V$ are said to be linearly independent if:

$$\forall \lambda_1, \ldots \lambda_k \in \mathbb{C},\ \lambda_1 v_1 + \ldots + \lambda_k v_k = 0 \iff \lambda_1 = 0, \ldots \lambda_k = 0 \tag{2.9}$$

A set of linearly independent vectors of V that is also spanning V is called a basis of V. It can be shown that, for any vector space V, there exists a basis and any two different bases of V have the same cardinal. This common cardinal is called the dimension of V. While infinite dimensional vector spaces exist, we will only consider finite dimensional vector spaces.

Example 2.3:
For $n \in \mathbb{N}$, let us consider $\mathbb{C}^n$. Then $\mathbb{C}^n$ is a complex vector space spanned by the linearly independent set $\{e_1, \ldots, e_n\}$ where e_i is the zero vector with 1 on the i-th coordinate. This set is called the canonical basis of $\mathbb{C}^n$ and the dimension of $\mathbb{C}^n$ is n.

Inner product

Let V be a complex vector space, an inner product is a function $\langle ., . \rangle : V \times V \to \mathbb{C}$ verifying the following properties:

- Conjugate symmetry:

$$\forall u, v \in V, \langle u, v \rangle = \langle v, u \rangle^* \tag{2.10}$$

- Linearity in the second argument[1]:

$$\forall \lambda \in \mathbb{C},\ \forall u, v, w \in V,\ \langle u, \lambda v + w \rangle = \lambda \langle u, v \rangle + \langle u, w \rangle \tag{2.11}$$

[1]In the mathematical literature, the inner product is defined as being linear in the first argument but in the scope of quantum physic it is linear in the second argument which is consistent with the Dirac notation.

- Positive-definiteness:

$$\forall v \in V,\ \langle v, v\rangle \geq 0 \text{ and } \langle v, v\rangle = 0 \iff v = 0 \tag{2.12}$$

Example 2.4:

In $\mathbb{C}^n$, the function defined by:

$$\langle (u_1, \ldots, u_n), (v_1, \ldots, v_n)\rangle = \sum_{i=1}^{n} u_i^* v_i \tag{2.13}$$

Is an inner product for $\mathbb{C}^n$

The inner product allows us to define orthogonality: two vectors $u, v \in V$ are said to be orthogonal if and only if:

$$\langle u, v\rangle = 0 \tag{2.14}$$

In addition the inner product induces a norm $\|.\|$, the canonical norm, defined by:

$$\forall v \in V,\ \|v\| = \sqrt{\langle v, v\rangle} \tag{2.15}$$

This norm, in turn, induces the metric $d : V \times V \to \mathbb{R}$ defined by:

$$\forall u, v \in V,\ d(u, v) = \|v - u\| \tag{2.16}$$

Hence, a complex vector space, equipped with an inner product is also a metric space.

Complete metric space

Let (M, d) a metric space. Then the sequence $(u_k) \in M^{\mathbb{N}}$ is said to be a Cauchy sequence if and only if:

$$\forall \epsilon > 0,\ \exists N \in \mathbb{N},\ \forall k, l > N,\ d(u_k, u_l) < \epsilon \tag{2.17}$$

(M, d) is then said to be a complete metric space if any Cauchy sequence in M also converges in M.

Regarding the finite-dimensional complex vector space V equipped with an inner product, we have seen that the inner product induces a metric d, making (V, d) a metric space. Being of finite dimensional, it can be shown that (V, d) is a complete metric space[2]. Hence any finite-dimensional vector space equipped with an inner product is a Hilbert space. In particular, for $n \in \mathbb{N}$, $\mathbb{C}^n$ is a Hilbert space. In fact, a complex vector space of dimension n being isomorphic to $\mathbb{C}^n$ we can limit ourselves to these spaces[3].

2.2.3 Matrix

Let V and W be two complex vector spaces. Then $f : V \to W$ is a linear map if:

$$\forall u, v \in V,\ \forall \lambda \in \mathbb{C},\ f(u + \lambda v) = f(u) + \lambda f(v) \tag{2.18}$$

Suppose now that V and W are of finite dimensional $n \in \mathbb{N}$ and $m \in \mathbb{N}$ respectively. Let $\{v_1, \ldots, v_n\}$ be a basis of V and $\{w_1, \ldots, w_m\}$ a basis of W. For $j \in [1, \ldots, n]$, as $f(v_j) \in W$, and $\{w_i\}_{i\in[1,\ldots m]}$ being a basis for W, there exist $a_{i,j} \in \mathbb{C}$ such that:

$$f(v_j) = \sum_{i=1}^{m} a_{i,j} w_i \tag{2.19}$$

Adopting the column vector representation, in the basis $\{w_i\}_{i\in[1,\ldots m]}$, we have:

$$f(v_j) = \begin{bmatrix} a_{1,j} \\ \vdots \\ a_{m,j} \end{bmatrix} = C_j \tag{2.20}$$

[2]This stems from the fact that, in the finite-dimensional case, the convergence of a vector is component-wise convergence and $\mathbb{C}$ is itself a complete metric space.

[3]Consider the linear map sending one basis of the space onto the canonical basis of C^n, then this map is an isomorphism.

$\{v_j\}_{j\in[1,\dots n]}$ being itself a basis of V, for $x \in V$, there exist $x_j \in \mathbb{C}$ such that $x = \sum_{j=1}^{n} x_j v_j$ and f being a linear map, it follows that:

$$f(x) = \sum_{j=1}^{n} x_j f(v_j) = \sum_{j=1}^{n} x_j C_j \tag{2.21}$$

Where the last equality has been written in the $\{w_i\}_{i\in[1,\dots m]}$ basis. If we collect the columns C_j into a matrix:

$$\mathbf{A} = [C_1 \dots C_n] \tag{2.22}$$

Then $\mathbf{A}$ is called the matrix representation of the linear map f in the bases $\{v_j\}_{j\in[1,\dots n]}$ and $\{w_i\}_{i\in[1,\dots m]}$. And we have:

$$f(x) = \sum_{j=1}^{n} x_j C_j = \mathbf{A}x \tag{2.23}$$

If $V = W$, then f is called an endomorphism. If f is invertible, it is called an isomorphism and its inverse is also an isomorphism. If f is an isomorphism, the spaces V and W are said to be isomorphic which implies that they have the same dimension. Often times, the matrix representation of a linear map $f : V \to W$ is given in the canonical bases of V and W respectively and the two are conflated with one another. From now on, we will only consider linear maps through matrices. The space of all the matrices with m rows, n columns and complex coefficients is denoted $\mathcal{M}_{m\times n}(\mathbb{C})$, and if $m = n$, the space of all complex square matrices is denoted $\mathcal{M}_n(\mathbb{C})$.

Hermitian conjugate

Let V and W be two Hilbert spaces of respective dimensions n and m. Let $\mathbf{A} : V \to W$ be a linear map, then it can be shown that there exists a unique linear map $\mathbf{A}^\dagger : W \to V$ verifying[4]:

$$\forall v \in V,\ \forall w \in W,\ \langle w, \mathbf{A}v\rangle_W = \langle \mathbf{A}^\dagger w, v\rangle_V \tag{2.24}$$

Where $A^\dagger$ is called the Hermitian conjugate or the adjoint and has the following properties:

[4]Once again the definition will differ from the one found in mathematical literature but it is consistent with the right linearity of the inner product as defined previously.

- $\forall \mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{C}),\ \left(\mathbf{A}^\dagger\right)^\dagger = \mathbf{A}$
- $\forall \mathbf{A}, \mathbf{B} \in \mathcal{M}_{m\times n}(\mathbb{C}),\ \forall \lambda \in \mathbb{C},\ (\mathbf{A} + \lambda\mathbf{B})^\dagger = \mathbf{A}^\dagger + \lambda^*\mathbf{B}^\dagger$

Now, if $W = V$, we have the additional properties:

- $\forall \mathbf{A}, \mathbf{B} \in \mathcal{M}_n(\mathbb{C}), (\mathbf{AB})^\dagger = \mathbf{B}^\dagger\mathbf{A}^\dagger$
- If $\mathbf{A}$ is invertible, so is $\mathbf{A}^\dagger$ and we have $\left(\mathbf{A}^\dagger\right)^{-1} = (\mathbf{A}^{-1})^\dagger$

If we consider the matrix representation, the Hermitian conjugate of $\mathbf{A}$ is simply the transposed complex conjugate of $\mathbf{A}$, that is:

$$\mathbf{A}^\dagger = (\mathbf{A}^*)^t \tag{2.25}$$

Linear forms

Let V be a Hilbert space. One type of linear maps of particular interest arises when considering the linear maps from V to $\mathbb{C}$. These linear maps are then called linear forms and the space of the linear forms from V to $\mathbb{C}$ is called the dual space of V, noted V^*. It can be shown that V^* is a vector space that is isomorphic to V. In fact, the inner product on V induces an isomorphism between V and V^*. Indeed, consider Φ defined as follows:

$$\Phi : \begin{cases} V \to V^* \\ v \mapsto \langle v, . \rangle \end{cases} \tag{2.26}$$

Then Φ is an isomorphism and for $v \in V$, $\Phi(v) \in V^*$ will be called the covector of v. In terms of matrix representation, it comes that:

$$\Phi(v) = (v^*)^t = v^\dagger \tag{2.27}$$

From now on, we will only consider matrices in $\mathcal{M}_n(\mathbb{C})$ for $n \in \mathbb{N}$. From the notion of adjoint stem two categories of square matrices that are fundamental in the field of quantum mechanics: hermitian matrices and unitary matrices.

Hermitian matrix

Let $n \in \mathbb{N}$ and $\mathbf{A} \in \mathcal{M}_n(\mathbb{C})$, then $\mathbf{A}$ is said to be hermitian if it is self adjoint, meaning that it is equal to its adjoint or formally:

$$\mathbf{A} = \mathbf{A}^\dagger = (\mathbf{A}^*)^t \tag{2.28}$$

Example 2.5:

Among the hermitian matrices, the Pauli matrices are particularly important. They can be denoted $\sigma_1, \sigma_2, \sigma_3$ or $\sigma_x, \sigma_y, \sigma_z$. The notation that will be used throughout this thesis is $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \mathbf{M}_2(\mathbb{C})$ defined by:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \ \mathbf{Y} = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}, \text{ and } \mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.29}$$

The properties of this set will be studied further throughout this chapter but it can already be established that they are involutory, meaning that they are their own inverse:

$$\mathbf{X} = \mathbf{X}^{-1}, \ \mathbf{Y} = \mathbf{Y}^{-1} \text{ and } \mathbf{Z} = \mathbf{Z}^{-1} \tag{2.30}$$

This can also be written:

$$\mathbf{X}^2 = \mathbf{Y}^2 = \mathbf{Z}^2 = \mathbf{I} \tag{2.31}$$

Where $\mathbf{I} \in \mathbf{M}_2(\mathbb{C})$ is the identity matrix. They are anti-commuting:

$$\mathbf{XY} = -\mathbf{YX}, \ \mathbf{XZ} = -\mathbf{ZX} \text{ and } \mathbf{YZ} = -\mathbf{ZY} \tag{2.32}$$

And we also have:

$$\mathbf{XY} = i\mathbf{Z}, \ \mathbf{YZ} = i\mathbf{X} \text{ and } \mathbf{ZX} = i\mathbf{Y} \tag{2.33}$$

Unitary matrix

Let $n \in \mathbb{N}$ and $\mathbf{A} \in \mathcal{M}_n(\mathbb{C})$, then $\mathbf{A}$ is said to be unitary if it is invertible and its inverse is its adjoint:

$$\mathbf{A}^{-1} = \mathbf{A}^{\dagger} \tag{2.34}$$

Example 2.6:

The Pauli matrices being hermitian and involutory, they are also unitary matrices. Other examples of unitary matrices are the rotation matrices $\mathbf{R}(\theta)$ defined, for $\theta \in \mathbb{R}$, by:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.35}$$

If $U(n)$ is the set of all the unitary matrices of size n, then $U(n)$ equipped with the matrix multiplication is a group called the unitary group. This means that:

- The identity matrix $\mathbf{I}$ is unitary.
- $\forall \mathbf{A}, \mathbf{B} \in U(n),\ \mathbf{AB} \in U(n)$
- $\forall \mathbf{A} \in U(n),\ \mathbf{A}^{-1} \in U(n)$

An interesting property of the unitary matrices is that they conserve the inner product:

$$\forall \mathbf{A} \in U(n),\ \forall u, v \in \mathbb{C}^n, \langle \mathbf{A}u, \mathbf{A}v \rangle = \langle u, v \rangle \tag{2.36}$$

Eigenvalues and eigenvectors

Let $n \in \mathbb{N}$ and $\mathbf{A} \in \mathcal{M}_n(\mathbb{C})$. $v \in \mathbb{C}^n$ is said to be an eigenvector of $\mathbf{A}$ associated with the eigenvalue $\lambda \in \mathbb{C}$ if:

$$v \neq 0 \text{ and } \mathbf{A}v = \lambda v \tag{2.37}$$

A matrix $\mathbf{A} \in \mathcal{M}_n(\mathbb{C})$ is said to be diagonalisable if there exists a set of vectors $\{v_1, \ldots, v_n\}$ that are both eigenvectors of $\mathbf{A}$, associated with the eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$ and form a basis of $\mathbb{C}^n$. $\mathbf{A}$ is said to be diagonalisable because in the basis $\{v_1, \ldots, v_n\}$, referred to as the eigenbasis, its representation is the diagonal matrix: $diag(\lambda_1, \ldots, \lambda_n)$. If in addition, $\{v_1, \ldots, v_n\}$ is an orthonormal basis, $\mathbf{A}$ is said to be unitarily diagonalisable. In this case, $\mathbf{A}$ can easily be decomposed as follows:

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i v_i v_i^\dagger \tag{2.38}$$

This leads to a strong characterisation called the spectral theorem. A matrix $\mathbf{A}$ is said to be normal if it commutes with its adjoint:

$$\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger\mathbf{A} \tag{2.39}$$

The spectral theorem then states that a matrix is normal if and only if it is unitarily diagonalisable.

Regarding the hermitian matrices, as they are their own adjoint, they are also normal, hence according to the spectral theorem, they are diagonalisable. There exist an additional result related to their eigenvalues: a matrix is hermitian if and only if it is normal and its eigenvalues are real.

On the same note, as the inverse of unitary matrices is their adjoint, they are normal. And concerning their eigenvalues: a matrix is unitary if and only if it is a normal matrix and its eigenvalues of modulus 1.

Example 2.7:

Coming back to the Pauli matrices $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$ introduced in Example 2.5. As seen previously they are hermitian so unitarily diagonalisable and their eigenvalues are real numbers. In addition, they are also unitary so their eigenvalues must have modulus 1. All in all, this means that their eigenvalues are in $\{-1, 1\}$. As they are not $\mathbf{I}$ or $-\mathbf{I}$ we can conclude that their eigenvalues

are exactly -1 and 1. Their eigenvectors being:

$$\mathbf{X}\frac{1}{\sqrt{2}}[1,1]^t = \frac{1}{\sqrt{2}}[1,1]^t \text{ and } \mathbf{X}\frac{1}{\sqrt{2}}[1,-1]^t = -\frac{1}{\sqrt{2}}[1,-1]^t \tag{2.40}$$

$$\mathbf{Y}\frac{1}{\sqrt{2}}[i,1]^t = \frac{1}{\sqrt{2}}[i,1]^t \text{ and } \mathbf{Y}\frac{1}{\sqrt{2}}[1,i]^t = -\frac{1}{\sqrt{2}}[1,i]^t \tag{2.41}$$

$$\mathbf{Z}[1,0]^t = [1,0]^t \text{ and } \mathbf{Z}[0,1]^t = -[0,1]^t \tag{2.42}$$

2.2.4 Tensor Product

Let $n, m \in \mathbb{N}$, the tensor product of $\mathbb{C}^n$ and $\mathbb{C}^m$, denoted $\mathbb{C}^n \otimes \mathbb{C}^m$, is a vector space arising from the bilinear map $\otimes$ defined by:

$$\begin{cases} \mathbb{C}^n \times \mathbb{C}^m \to \mathbb{C}^n \otimes \mathbb{C}^m \\ (u,v) \mapsto u \otimes v \end{cases} \tag{2.43}$$

If $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_m\}$ are basis of $\mathbb{C}^n$ and $\mathbb{C}^m$, then $\{a_i \otimes b_j \mid 1 \leq i \leq n,\ 1 \leq j \leq m\}$ is a basis of $\mathbb{C}^n \otimes \mathbb{C}^m$ and for $u = \lambda_1 a_1 + \ldots \lambda_n a_n \in \mathbb{C}^n$ and $v = \mu_1 b_1 + \ldots \mu_m a_m \in \mathbb{C}^m$ we have:

$$u \otimes v = \sum_{i,j} \lambda_i \mu_j a_i \otimes b_j \tag{2.44}$$

In addition, $\mathbb{C}^n \otimes \mathbb{C}^m$ is equipped with an inner product defined by:

$$\forall u_1, u_2 \in \mathbb{C}^n,\ \forall v_1, v_2 \in \mathbb{C}^m, \langle u_1 \otimes v_1, u_2 \otimes v_2 \rangle = \langle u_1, u_2 \rangle \langle v_1, v_2 \rangle \tag{2.45}$$

So $\mathbb{C}^n \otimes \mathbb{C}^m$ is a Hilbert space of dimension nm, hence it is isomorphic to $\mathbb{C}^{nm}$.

Now let $\mathbf{A} \in \mathcal{M}_n(\mathbb{C})$ and $\mathbf{B} \in \mathcal{M}_m(\mathbb{C})$, then their tensor product $\mathbf{A} \otimes \mathbf{B}$, also known as Kronecker product, is defined by:

$$\forall u \in \mathbb{C}^n,\ \forall v \in \mathbb{C}^m,\ (\mathbf{A} \otimes \mathbf{B})(u \otimes v) = (\mathbf{A}u) \otimes (\mathbf{B}v) \tag{2.46}$$

In terms of matrix coefficients, if $\mathbf{A} = [a_{i,j}]_{1\leq i,j\leq n}$, then:

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & \cdots & a_{n,n}\mathbf{B} \end{bmatrix} \tag{2.47}$$

So $\mathbf{A}\otimes\mathbf{B} \in \mathcal{M}_{nm}(\mathbb{C})$.

Example 2.8:

Consider $\mathbb{C}^2$ with its canonical basis $\{e_1 = [1,0]^t, e_2 = [0,1]\}$, then:

$$e_1\otimes e_1 = \begin{bmatrix} 1\begin{bmatrix}1\\0\end{bmatrix} \\ 0\begin{bmatrix}1\\0\end{bmatrix} \end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix},\ e_1\otimes e_2 = \begin{bmatrix} 1\begin{bmatrix}0\\1\end{bmatrix} \\ 0\begin{bmatrix}0\\1\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

$$e_2\otimes e_1 = \begin{bmatrix} 0\begin{bmatrix}1\\0\end{bmatrix} \\ 1\begin{bmatrix}1\\0\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix},\ e_2\otimes e_2 = \begin{bmatrix} 0\begin{bmatrix}0\\1\end{bmatrix} \\ 1\begin{bmatrix}0\\1\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

And $\{e_1\otimes e_1, e_1\otimes e_2, e_2\otimes e_1, e_2\otimes e_2\}$ is a basis of $\mathbb{C}^2\otimes\mathbb{C}^2$ which is itself isomorphic to $\mathbb{C}^4$.

Now if we consider the Pauli matrices introduced in Example 2.5. We have, for example:

$$\mathbf{X}\otimes\mathbf{Y} = \begin{bmatrix} 0\mathbf{Y} & 1\mathbf{Y} \\ 1\mathbf{Y} & 0\mathbf{Y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & i \\ 0 & 0 & -i & 0 \\ 0 & i & 0 & 0 \\ -i & 0 & 0 & 0 \end{bmatrix} \tag{2.48}$$

And:

$$\mathbf{Y} \otimes \mathbf{X} = \begin{bmatrix} 0\mathbf{X} & i\mathbf{X} \\ -i\mathbf{X} & 0\mathbf{X} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{bmatrix} \tag{2.49}$$

Notice that the tensor product is not commutative as $\mathbf{X} \otimes \mathbf{Y} \neq \mathbf{Y} \otimes \mathbf{X}$.

The tensor product also holds the following properties:

- $\forall \mathbf{A}, \mathbf{C} \in \mathcal{M}_n(\mathbb{C}),\ \forall \mathbf{B}, \mathbf{D} \in \mathcal{M}_m(\mathbb{C}),\ (\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD})$
- $\forall \mathbf{A} \in \mathcal{M}_n(\mathbb{C}),\ \forall \mathbf{B} \in \mathcal{M}_m(\mathbb{C}), (\mathbf{A} \otimes \mathbf{B})^\dagger = \mathbf{A}^\dagger \otimes \mathbf{B}^\dagger$
- For $\mathbf{A} \in \mathcal{M}_n(\mathbb{C})$ and $\mathbf{B} \in \mathcal{M}_m(\mathbb{C})$, $\mathbf{A} \otimes \mathbf{B}$ is invertible if and only if $\mathbf{A}$ and $\mathbf{B}$ are invertible, in which case: $(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$

From these properties, it comes that a tensor product of hermitian matrices is a hermitian matrix and the tensor product of unitary matrices is also a unitary matrix.

2.2.5 Dirac Notation

This section is intended to provide a link between the fields of mathematics and quantum mechanics by introducing the Dirac notation. This notation is widely used in quantum mechanics to denote vectors and covectors using respectively the ket and the bra.

Let $n \in \mathbb{N}$, a vector $v \in \mathbb{C}^n$ will be denoted with the ket notation:

$$|v\rangle \tag{2.50}$$

The covector of v will then be denoted using the bra notation:

$$\langle v| \tag{2.51}$$

And we thus have the relation:

$$\langle v| = |v\rangle^\dagger \tag{2.52}$$

The Dirac notation offers some useful notational shortcuts. Among them, for $u, v \in \mathbb{C}^n$, the inner product between u and v is written:

$$\langle u|v\rangle \tag{2.53}$$

This notation explains why the inner product has been defined as linear in the second argument and anti-linear in the first argument. If $\mathbf{A} \in \mathcal{M}_n(\mathbb{C})$ has eigenvalue λ, then one can denote $|\lambda\rangle$ to refer to an eigenvector corresponding to λ, that is:

$$\mathbf{A}\,|\lambda\rangle = \lambda\,|\lambda\rangle \tag{2.54}$$

So if $\mathbf{A}$ is unitarily diagonalisable with eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$, $\mathbf{A}$ can be decomposed as:

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i\,|\lambda_i\rangle\,\langle\lambda_i| \tag{2.55}$$

Finally, the Dirac notation allows for more compact writing of tensor products. For $n, m \in \mathbb{N}$, if $u \in \mathbb{C}^n$ and $v \in \mathbb{C}^m$, the tensor product of u and v can be written:

$$|u\rangle \otimes |v\rangle = |u\rangle\,|v\rangle = |u, v\rangle = |uv\rangle \tag{2.56}$$

2.3 Quantum Computing Background

2.3.1 The Postulates of Quantum Mechanics

Here we will state the four postulates of quantum mechanics as can be found in [57]. These postulates map the physical reality to the mathematical framework used to describe it.

First postulate

The state space of any closed physical system is a complex Hilbert space. The system is completely described by its state vector which is a unit vector. This means that if $\mathbb{C}^n$ is the state space of the system, then its state can be described by $|\psi\rangle \in \mathbb{C}^n$ such that:

$$\| |\psi\rangle \| = \sqrt{\langle\psi|\psi\rangle} = 1 \tag{2.57}$$

Example 2.9:

Suppose the state space to be $\mathbb{C}^2$, then by denoting $\{|0\rangle, |1\rangle\}$ its canonical basis, a state vector can be any vector $a|0\rangle + b|1\rangle$ such that:

$$a, b \in \mathbb{C} \text{ and } \sqrt{|a|^2 + |b|^2} = 1 \tag{2.58}$$

Second postulate

The evolution in time of a closed system is described by a unitary transformation. Suppose that the system evolves from the state $|\psi_1\rangle \in \mathbb{C}^n$ to the state $|\psi_2\rangle \in \mathbb{C}^n$. Then there exists $\mathbf{U} \in U(n)$ such that:

$$|\psi_2\rangle = \mathbf{U}|\psi_1\rangle \tag{2.59}$$

Example 2.10:

The Pauli matrices introduced in Example 2.5, being also unitary matrices, can be used to describe the evolution of a physical system evolving in $\mathbb{C}^2$. Another widely used transformation is the Hadamard transformation $\mathbf{H}$:

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.60}$$

Third postulate

Quantum measurements are described by a set $\{\mathbf{M}_m\}$ of measurement operators. Those are linear maps acting on the state space and verifying the completeness equation:

$$\sum_m \mathbf{M}_m^\dagger \mathbf{M}_m = \mathbf{I} \tag{2.61}$$

The index m here refers to one of the possible outcomes of the measurement. When applying the measurement to the state $|\psi\rangle$, the probability $p(m)$ of measuring m is given by:

$$p(m) = \langle\psi| \mathbf{M}_m^\dagger \mathbf{M}_m |\psi\rangle \tag{2.62}$$

Immediately after measuring m, the system will collapse to the state $|\psi_m\rangle$:

$$|\psi_m\rangle = \frac{\mathbf{M}_m |\psi\rangle}{\sqrt{\langle\psi| \mathbf{M}_m^\dagger \mathbf{M}_m |\psi\rangle}} \tag{2.63}$$

Example 2.11:

One important type of measurement is the projective measurement. A projective measurement is described by an observable that is a hermitian matrix $\mathbf{M}$. $\mathbf{M}$ being hermitian, we know that it is unitarily diagonalisable and its eigenvalues are real numbers. This means that $\mathbf{M}$ can be decomposed as:

$$\mathbf{M} = \sum_{\lambda \in Sp(\mathbf{M})} \lambda \mathbf{P}_\lambda \tag{2.64}$$

Where $Sp(\mathbf{M})$ is the set of the distinct eigenvalues of $\mathbf{M}$ and for $\lambda \in Sp(\mathbf{M})$, $\mathbf{P}_\lambda$ is the orthogonal projector onto the eigenspace associated to λ. $\mathbf{M}$ being unitarily diagonalisable, we have $\mathbf{P}_\lambda = \mathbf{P}_\lambda^2 = \mathbf{P}_\lambda^\dagger$ and

$$\sum_{\lambda \in Sp(\lambda)} \mathbf{P}_\lambda = \mathbf{I} \tag{2.65}$$

So:

$$\sum_{\lambda \in Sp(\lambda)} \mathbf{P}_\lambda^\dagger \mathbf{P}_\lambda = \mathbf{I} \tag{2.66}$$

Meaning that it is valid to perform measurements in the eigenbasis of a hermitian matrix.

Fourth postulate

The state space of a composite system is the tensor product of the component systems. This means that if the composite system is made of s systems that are in the respective states $|\psi_1\rangle, \ldots, |\psi_s\rangle$, then the overall system is in state:

$$|\psi_1\rangle \otimes \ldots \otimes |\psi_s\rangle = |\psi_1, \ldots, \psi_s\rangle \tag{2.67}$$

Example 2.12:

From this description of composite systems arises the intriguing concept of entanglement that is unique to quantum mechanics. A composite system is said to be in an entangled state if its state vector can not be decomposed as a tensor product of the states of its component systems. Prime and fundamental examples of such entangled states are the Bell states. Consider $\mathbb{C}^2 \otimes \mathbb{C}^2$, where, as in Example 2.9, the canonical basis of $\mathbb{C}^2$ is denoted $\{|0\rangle, |1\rangle\}$, then the four Bell states are defined by:

$$\left|\Phi^{\pm}\right\rangle = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}} \text{ and } \left|\Psi^{\pm}\right\rangle = \frac{|01\rangle \pm |10\rangle}{\sqrt{2}} \tag{2.68}$$

It can be shown that none of these states can be decomposed as a tensor product of two vector states in $\mathbb{C}^2$. Hence those states are said to be entangled states.

2.3.2 Quantum Computing Framework

The qubit

The most elementary system in the theory of quantum computing is the qubit. This is a two-dimensional system hence, according to the first postulate, its state space is $\mathbb{C}^2$ and we denote the canonical basis:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.69}$$

This basis is also called the computational basis. A qubit can then be described by the state vector:

$$a\,|0\rangle + b\,|1\rangle \text{ where } a, b \in \mathbb{C} \text{ and } |a|^2 + |b|^2 = 1 \tag{2.70}$$

If $a \neq 0$ and $b \neq 0$, then the qubit is said to be in superposition of the states $|0\rangle$ and $|1\rangle$ and performing a projective measurement using the observable $\mathbf{Z}$, the possible outcomes will be:

$$1 \text{ with probability } |a|^2 \text{ or } -1 \text{ with probability } |b|^2 \tag{2.71}$$

So after measurement, the superposition will collapse to the state

$$|0\rangle \text{ with probability } |a|^2 \text{ or } |1\rangle \text{ with probability } |b|^2 \tag{2.72}$$

Another well-known one qubit basis is given by the states $|+\rangle$ and $|-\rangle$ defined by:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \text{ and } |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.73}$$

Multiple qubits system

Suppose now that we are working with a system comprising of n qubits, then according to the fourth postulate, the system's state space is $(\mathbb{C}^2)^{\otimes n}$ that is isomorphic to $\mathbb{C}^{2^n}$. The computational basis of this system is then:

$$\{|x\rangle \mid x \in \{0,1\}^n\} \tag{2.74}$$

And the state of the system can be described by any state vector of the form:

$$\sum_{x\in\{0,1\}^n} a_x\,|x\rangle \text{ where } \forall x \in \{0,1\}^n,\ a_x \in \mathbb{C} \text{ and } \sum_{x\in\{0,1\}^n} |a_x|^2 = 1 \tag{2.75}$$

Quantum gates

Now that the computation units have been introduced, the question of what can be done with them arises. Although there exist other models for quantum computation, we will only focus on gate-based quantum computing. The second postulate states that any transformation to the system must be a unitary transformation. From now on, we will call a unitary transformation a quantum gate as they can be seen as the quantum version of the classical logic gates. The simplest of these quantum gates are the single qubit gates, acting on one qubit. These gates are elements of $U(2)$ and the Pauli matrices as well as the Hadamard transformation are single qubit gates. Other possible single qubit gates are the rotation gates $\mathbf{R}_X(\theta)$, $\mathbf{R}_Y(\theta)$ and $\mathbf{R}_Z(\theta)$ defined for $\theta \in \mathbb{R}$ by:

$$\mathbf{R}_x(\theta) = e^{-\frac{i\theta\mathbf{X}}{2}} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \tag{2.76}$$

$$\mathbf{R}_y(\theta) = e^{-\frac{i\theta\mathbf{Y}}{2}} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \tag{2.77}$$

$$\mathbf{R}_z(\theta) = e^{-\frac{i\theta\mathbf{Z}}{2}} = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix} \tag{2.78}$$

Notice that the $\mathbf{R}_y$ gate is the usual rotation matrix as introduced in Example 2.6. Actually, it can be shown [57, Theorem 4.1] that up to a phase factor, any single qubit gate can be decomposed as a product of some of these rotation gates.

Regarding systems with multiple qubits, suppose the state space is $\mathbb{C}^{2^n}$, we could apply single qubit gates $\mathbf{U}_i \in U(2)$ to each of the composite systems, resulting in the overall gate:

$$\mathbf{U} = \mathbf{U}_1 \otimes \ldots \otimes \mathbf{U}_n \in U(2^n) \tag{2.79}$$

However, in that case, each qubit would evolve independently from the others. It would thus be more interesting if we could apply gates that, in one way, make qubits interact with each other. One such gate is the controlled NOT gate, denoted **CNOT** or **CX** that takes as inputs

two qubits, one control qubit and one target qubit, and will apply the **X** gate on the target qubit only if the control qubit is in the state $|1\rangle$. In a two qubits system where the first qubit is the control one and the second is the target qubit, the **CNOT** gate is given by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.80}$$

In fact, it is possible to take any single qubit gate and make a controlled version of it. There also exist gates that are controlled by more than one qubit. For example, the Toffoli gate is a **X** gate that is controlled by two qubits. While there exist many other multi-qubits gates that are other than controlled gates, the work of this thesis revolves around **X** gates controlled by multiple qubits.

Quantum circuits

A quantum circuit is a graphical representation of the transformation applied to a system of multiple qubits that is akin to classical logic circuits. In a quantum circuit, a qubit is represented by a wire and a gate is represented by a block covering the qubits it is acting on. If a gate is a controlled gate, a "switch" linked to the control qubits is added to the gate. Finally, measurements are represented with a meter symbol, if not specified, the measurements will usually be done in the computational basis or equivalently in the **Z** basis. In Figure 2.1 are represented some gates commonly found in the quantum computing literature as well as the measurement representation. A quantum circuit is then built by assembling these different

(a) Hadamard gate (b) CNOT gate (c) Toffoli gate (d) Measurement

Figure 2.1: Common quantum circuit components

components. Starting from the left with the initial state, usually, all the qubits are initialised in the $|0\rangle$ state, the quantum gates are added to the circuit in the desired configuration and finally, the quantum circuits are usually terminated by measurements so that a result can be measured. In Figure 2.2 is depicted a quantum circuit that creates the Bell state $|\Phi^+\rangle$, starting from the state $|00\rangle$. Right before the measurement, the system is indeed in state $|\Phi^+\rangle$. The

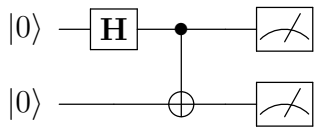


Figure 2.2: Quantum circuit creating the Bell state $|\Phi^+\rangle$ and performing measurement

measurements performed at the end of this circuit will then yield $\{+1, +1\}$ or $\{-1, -1\}$ with probability $\frac{1}{2}$ each. Actually as $|\Phi^+\rangle$ is an entangled state, it would have been sufficient to just measure one of the qubits.

2.3.3 Quantum Algorithms

Quantum algorithms are procedures that in one way or another involve quantum circuits and most importantly leverage quantum phenomena such as superposition and entanglement. By doing so, some of the quantum algorithms exhibit a speed-up when compared to their classical counterparts. In the following we list and slightly detail some of the most famous quantum algorithms.

Certainly among the pantheon of quantum algorithms is Shor's algorithm [73]. By using superposition, entanglement and quantum Fourier transform, this algorithm can find a prime factor of a number. While for large numbers this is a notoriously hard task for classical computers, with the best-known complexity being sub-exponential, Shor's algorithm can theoretically perform the same task with a polylogarithmic complexity.

Other quantum algorithms that may be more closely related to the work of this thesis are oracle identification algorithms. While the tasks solved by these algorithms can differ, they

all involve a Boolean function in which a characteristic has to be identified. For example, the Deutsch-Jozsa algorithm [24] will differentiate between a balanced function and a constant one. If n is the number of variables of the function, then classically, the worst case requires $2^{n-1}+1$ different evaluations of the function. On the other hand, by using a superposition of all the possible inputs, entangled with their respective image by the Boolean function, the quantum algorithm can determine the nature of the function in just one query to function oracle. In a similar way, the Bernstein-Vazirani algorithm [11] will find the secret string generating the Boolean function via dot product. If n is the number of variables, it does so with only one query to the oracle instead of n for the classical algorithm.

Finally, the other archetypal example of a quantum algorithm is Grover's algorithm [34] which is a search algorithm. Given an unstructured database of N elements, a classical algorithm will require approximately N operations to find a marked element within this database. Once again by using a superposition of all the elements of the database, entangled with their respective mark, Grover's algorithm will find the marked element after $\sqrt{N}$ application of a specific operator.

2.3.4 Quantum Amplitude Amplification

Throughout this thesis, we will use a procedure named quantum amplitude amplification or amplitude amplification for short. This procedure, introduced by [15], can be seen as a generalisation of the previously mentioned Grover's algorithm [34] as it works for any configuration of superposition where Grover's algorithm only works with the uniform superposition. As its name suggests, amplitude amplification allows to amplify the amplitude of states that are of interest so that in the end, the probability of measuring one of them is close to 1. While the basis used in [15] is the computational basis, the version presented here works with any orthonormal basis.

Let $n \in \mathbb{N}$, $\mathcal{A} = \{|\alpha_k\rangle\}$ and $\mathcal{B} = \{|\beta_k\rangle\}$ orthonormal basis of the Hilbert space $\mathbb{C}^{2^n}$ and suppose

we have $\mathbf{U} \in U(2^n)$ such that :

$$\mathbf{U} |\alpha_0\rangle = |\psi\rangle = \sum_{x \in \mathcal{B}} b_x |x\rangle \tag{2.81}$$

Now let $G \subseteq \{|\beta_k\rangle\}$ be the set of the good states, that is, the states we are interested in. $|\psi\rangle$ can be rewritten:

$$|\psi\rangle = \sum_{x \in G} b_x |x\rangle + \sum_{x \notin G} b_x |x\rangle \tag{2.82}$$

Let $\theta \in \left[0, \frac{\pi}{2}\right]$ such that:

$$\sin(\theta) = \sqrt{\sum_{x \in G} |b_x|^2} \tag{2.83}$$

Then $|\psi\rangle$ can be rewritten:

$$|\psi\rangle = \sin(\theta) |\psi_G\rangle + \cos(\theta) |\psi_{\not G}\rangle \tag{2.84}$$

Where And $|\psi_G\rangle$ and $|\psi_{\not G}\rangle$ are the normalised states:

$$|\psi_G\rangle = \frac{1}{\sin(\theta)} \sum_{x \in G} b_x |x\rangle \tag{2.85}$$

And:

$$|\psi_{\not G}\rangle = \frac{1}{\cos(\theta)} \sum_{x \notin G} b_x |x\rangle \tag{2.86}$$

Then when measuring in the $\mathcal{B}$ basis, the probability of measuring a state in G is equal to $\sin^2(\theta)$. In order to perform amplitude amplification, two more operators are needed, namely $\mathcal{X}_G$, defined in the $\mathcal{B}$ basis by:

$$\mathcal{X}_G = \mathbf{I} - 2 \sum_{x \in G} |x\rangle \langle x| \tag{2.87}$$

So for $|x\rangle \in \mathcal{B}$, $\mathcal{X}_G$ will act as follows:

$$\mathcal{X}_G |x\rangle = \begin{cases} -|x\rangle \text{ if } |x\rangle \in G \\ \\ |x\rangle \text{ else} \end{cases} \tag{2.88}$$

And $\mathcal{X}_{\alpha_0}$ defined in the $\mathcal{A}$ basis by:

$$\mathcal{X}_{\alpha_0} = \mathbf{I} - 2\left|\alpha_0\right\rangle\left\langle\alpha_0\right| \tag{2.89}$$

Similarly to $\mathcal{X}_G$, $\mathcal{X}_{\alpha_0}$ has the following action on the $\mathcal{A}$ basis:

$$\mathcal{X}_{\alpha_0}\left|x\right\rangle = \begin{cases} -\left|x\right\rangle \text{ if } \left|x\right\rangle = \left|\alpha_0\right\rangle \\ \left|x\right\rangle \text{ else} \end{cases} \tag{2.90}$$

These operators allow to define the diffusion operator $\mathbf{Q}$:

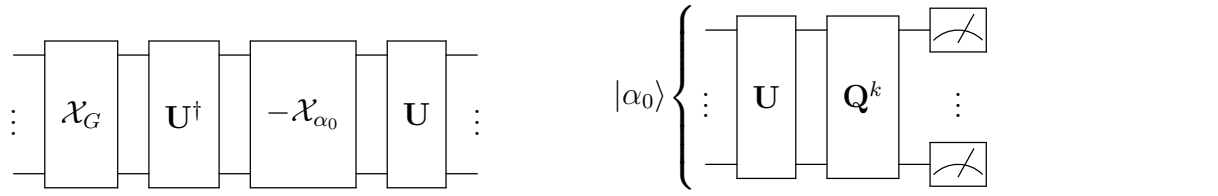
$$\mathbf{Q} = -\mathbf{U}\mathcal{X}_{\alpha_0}\mathbf{U}^\dagger\mathcal{X}_G \tag{2.91}$$

The quantum circuit for $\mathbf{Q}$ is depicted in Figure 2.3a

As shown in [15], for $k \in \mathbb{N}$:

$$\mathbf{Q}^k\left|\psi\right\rangle = \sin((2k+1)\theta)\left|\psi_G\right\rangle + \cos((2k+1)\theta)\left|\psi_{\not G}\right\rangle \tag{2.92}$$

So by taking $k_0 \in \mathbb{N}$ such that $(2k_0+1)\theta \approx \frac{\pi}{2}$ it is possible to measure a good state with high probability. The quantum circuit for the whole amplitude amplification procedure is represented in Figure 2.3b



(a) Quantum circuit for $\mathbf{Q}$ (b) Quantum circuit for amplitude amplification

Figure 2.3: Quantum circuits for the diffusion operator (Figure 2.3a) and amplitude amplification (Figure 2.3b)

Chapter 3

Tunable Neural Networks

3.1 Introduction

In this chapter, we introduce the architecture that we named tunable quantum neural networks. This construction relies on the correspondence, introduced by [88] and most recently by [14], between the algebraic normal form of a Boolean function and a quantum circuit made of multi-controlled **X** gates acting on a single ancillary qubit. We prove that this correspondence is correct and unique by building a group isomorphism between the set of Boolean functions and the set of multi-controlled **X** gates of this type. While this construction is quite simple, to the best of our knowledge, such a proof did not exist. This allows us to introduce a quantum circuit ansatz whose gates can be tuned in order to exactly learn any Boolean functions.

This circuit can be likened to a neural network in the sense that each gate can only perform a simple operation on the input data as well as the output of the previous gate. The circuit taken as a whole is then able to compute complex Boolean functions. The gates of this circuit can then be tuned in order to learn, either exactly or approximately, any Boolean function. This training process will be the subject of the following chapters where this architecture will be studied in different learning frameworks.

3.2 The Algebraic Normal Form of Boolean Functions

Let $\mathbf{F}_2 = (\{0,1\}, \oplus, .)$ be the smallest Galois field that will be identified with the set of Boolean values and denoted $\mathbb{B}$. In this case, $\oplus$ denotes the logical operator **XOR** and . denotes the logical operator **AND**. For $n \in \mathbb{N}$, $\mathbb{B}^{\mathbb{B}^n}$ is the set of the functions from $\mathbb{B}^n$ to $\mathbb{B}$, also known as Boolean functions.

Definition 3.1:

Let $n \in \mathbb{N}$. For $f, g \in \mathbb{B}^{\mathbb{B}^n}$ we define $f \oplus g \in \mathbb{B}^{\mathbb{B}^n}$ by:

$$f \oplus g : x \mapsto f(x) \oplus g(x) \tag{3.1}$$

From this definition of the **XOR** operator over two Boolean functions, we have the following:

Lemma 3.2:

For $n \in \mathbb{N}$, $\left(\mathbb{B}^{\mathbb{B}^n}, \oplus\right)$ is a finite abelian group where the identity element is the constant function $\mathbf{0}$ and where each element is its own inverse.

Proof:

Let $f, g \in \mathbb{B}^{\mathbb{B}^n}$:

Let $x \in \mathbb{B}^n$, by definition of $\oplus$, we have: $(f \oplus g)(x) = f(x) \oplus g(x) \in \mathbb{B}$ so: $f \oplus g \in \mathbb{B}^{\mathbb{B}^n}$.

Let $f, g, h \in \mathbb{B}^{\mathbb{B}^n}$:

Let $x \in \mathbb{B}^n$, then $((f \oplus g) \oplus h)(x) = (f \oplus g)(x) \oplus h(x) = (f(x) \oplus g(x)) \oplus h(x) = f(x) \oplus (g(x) \oplus h(x))$.

So $((f \oplus g) \oplus h)(x) = f(x) \oplus (g \oplus h)(x) = (f \oplus (g \oplus h))(x)$ hence: $(f \oplus g) \oplus h = f \oplus (g \oplus h)$

Let $f \in \mathbb{B}^{\mathbb{B}^n}$, and $\mathbf{0} \in \mathbb{B}^{\mathbb{B}^n}$ the constant function:

Let $x \in \mathbb{B}^n$, $(f \oplus \mathbf{0})(x) = f(x) \oplus \mathbf{0}(x) = f(x) \oplus 0 = f(x) = 0 \oplus f(x) = \mathbf{0}(x) \oplus f(x) = (\mathbf{0} \oplus f)(x)$

so $f \oplus \mathbf{0} = \mathbf{0} \oplus f$ and $\mathbf{0}$ is the identity element.

Let $f \in \mathbb{B}^{\mathbb{B}^n}$: Let $x \in \mathbb{B}^n$, $(f \oplus f)(x) = f(x) \oplus f(x) = 0 = \mathbf{0}(x)$ so $f \oplus f = \mathbf{0}$ and f is its own inverse. ■

Definition 3.3:

Let $n \in \mathbb{N}$ and $u = u_0 \dots u_{n-1} \in \mathbb{B}^n$. We denote:

$$1_u = \{i \in [0, \dots, n-1] \mid u_i = 1\} \tag{3.2}$$

1_u is the set of the indices for which a component of u is equal to 1

Definition 3.4:

Let $n \in \mathbb{N}$. For $x \in \mathbb{B}^n$ and $u \in \mathbb{B}^n \setminus \{0\}$. We define x^u as:

$$x^u = \prod_{i \in 1_u} x_i \tag{3.3}$$

And for $x \in \mathbb{B}^n$, we denote:

$$x^0 = 1 \tag{3.4}$$

This allows us to define the following function:

Definition 3.5:

Let $n \in \mathbb{N}$ and $u \in \mathbb{B}^n$. We denote $m_u \in \mathbb{B}^{\mathbb{B}^n}$ the function defined by:

$$m_u : x \mapsto x^u \tag{3.5}$$

m_u is then called a monomial.

Definition 3.6:

Let $n \in \mathbb{N}$ and $\mathcal{M}_n = \{m_u \mid u \in \mathbb{B}^n\}$ be the set of all the monomials in $\mathbb{B}^{\mathbb{B}^n}$.

We denote by $\mathcal{A}_n$ the subgroup of $\left(\mathbb{B}^{\mathbb{B}^n}, \oplus\right)$ generated by $\mathcal{M}_n$.

Theorem 3.7:

Let $n \in \mathbb{N}$ and $\mathcal{A}_n$ the subgroup of $\left(\mathbb{B}^{\mathbb{B}^n}, \oplus\right)$ as defined in Definition 3.6, then:

$$\mathcal{A}_n = \mathbb{B}^{\mathbb{B}^n} \tag{3.6}$$

Proof:

Let $n \in \mathbb{N}$. The commutativity of $\oplus$ and the fact that for $u \in \mathbb{B}^n$, we have $m_u = m_u^{-1}$ lead to:

$$\mathcal{A}_n = \left\{ \bigoplus_{u \in \mathbb{B}^n} \alpha_u m_u \mid \forall u \in \mathbb{B}^n, \alpha_u \in \{0,1\} \right\} \tag{3.7}$$

As $\mathcal{A}_n$ is a subgroup of $\mathbb{B}^{\mathbb{B}^n}$ we have $\mathcal{A}_n \subseteq \mathbb{B}^{\mathbb{B}^n}$.

Now take $\{\alpha_u\}_{u \in \mathbb{B}^n}, \{\beta_u\}_{u \in \mathbb{B}^n} \in \{0,1\}^{2^n}$ such that $\{\alpha_u\}_{u \in \mathbb{B}^n} \neq \{\beta_u\}_{u \in \mathbb{B}^n}$ and denote

$$D = \{u \in \mathbb{B}^n \mid \alpha_u \neq \beta_u\} \tag{3.8}$$

And suppose:

$$\bigoplus_{u \in \mathbb{B}^n} \alpha_u m_u = \bigoplus_{u \in \mathbb{B}^n} \beta_u m_u \tag{3.9}$$

Then:

$$\bigoplus_{u \in \mathbb{B}^n} \alpha_u m_u \oplus \bigoplus_{u \in \mathbb{B}^n} \beta_u m_u = \bigoplus_{u \in \mathbb{B}^n} (\alpha_u \oplus \beta_u) m_u = \bigoplus_{u \in D} m_u = \mathbf{0} \tag{3.10}$$

In Equation 3.10, the terms α_u and β_u taking value in $\{0,1\}$ have voluntarily been conflated with their Boolean values to express the fact that m_u is an involution. Let $u^0 \in D$ with minimal Hamming weight, that is u^0 is such that $|1_{u^0}|$ is the smallest among all $u \in D$. Let $u \in D \setminus \{u^0\}$ then there exist $i \in 1_u$ such that $i \notin 1_{u^0}$, meaning $u_i^0 = 0$, which leads to $m_u(u^0) = 0$. As $m_{u^0}(u^0) = 1$ we have:

$$\bigoplus_{u \in D} m_u(u^0) = 1 \neq 0 \tag{3.11}$$

So $\bigoplus_{u \in D} m_u \neq \mathbf{0}$ hence:

$$\bigoplus_{u \in \mathbb{B}^n} \alpha_u m_u \neq \bigoplus_{u \in \mathbb{B}^n} \beta_u m_u \tag{3.12}$$

From what precedes, we conclude that $|\mathcal{A}_n| = 2^{2^n} = \left|\mathbb{B}^{\mathbb{B}^n}\right|$ and thus: $\mathcal{A}_n = \mathbb{B}^{\mathbb{B}^n}$ ■

In essence, Theorem 3.7 states that any $f \in \mathbb{B}^{\mathbb{B}^n}$ can uniquely be written under the form:

$$f = \bigoplus_{u \in \mathbb{B}^n} \alpha_u^f m_u \tag{3.13}$$

This polynomial representation is called the algebraic normal form (ANF). The ANF is of

interest as it renders explicit the relation between the inputs of a Boolean function and the corresponding output using only two simple Boolean operators: **XOR** and **AND**. While there exist other polynomial representations such as the disjunctive normal form, the ANF is particularly suited to being used in the context of quantum computing as it can easily be translated into a quantum circuit as will be shown later. In Table 3.1 are gathered some functions of $\mathbb{B}^{\mathbb{B}^2}$ along with their algebraic normal form and the corresponding coefficients.

function	$\{\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11}\}$	ANF
0	$\{0,0,0,0\}$	0
1	$\{1,0,0,0\}$	1
AND(x_0x_1)	$\{0,0,0,1\}$	$x_0.x_1$
XOR(x_0x_1)	$\{0,1,1,0\}$	$x_0 \oplus x_1$
OR(x_0x_1)	$\{0,1,1,1\}$	$x_0 \oplus x_1 \oplus x_0.x_1$
NOT(x_0)	$\{1,0,1,0\}$	$1 \oplus x_0$

Table 3.1: Some functions of $\mathbb{B}^{\mathbb{B}^2}$ with their respective ANF

For the sake of completeness, we present a way to construct the algebraic normal form of a Boolean function. There are several different methods but here we present the method of indeterminate coefficients [45] through an example.

Example 3.8:

Suppose we wish to express the ANF of the function **OR** $\in \mathbb{B}^{\mathbb{B}^2}$ defined by Table 3.2.

x	00	01	10	11
OR(x)	0	1	1	1

Table 3.2: Truth table of **OR**

The outputs of f can be gathered in the vector $\mathbf{v}^f = (0,1,1,1)^T$ and we are looking for the vector $\mathbf{c}^f = (\alpha^f_{00}, \alpha^f_{01}, \alpha^f_{10}, \alpha^f_{11})$ such that $\mathbf{M}\mathbf{c}^f = \mathbf{v}^f$ where $\mathbf{M}$ is the matrix constructed as follow: $\mathbf{M}_{x,u} = m_u(x)$. In our case:

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{3.14}$$

It can be shown that $\mathbf{M} = \mathbf{M}^{-1}$ in $\mathbb{B}^{4\times4}$, so we have $\mathbf{c}^f = \mathbf{M}\mathbf{v}^f$ that is $\mathbf{c}^f = (0,1,1,1)^T$. This leads to the ANF: $\mathbf{OR}(x_0x_1) = x_0 \oplus x_1 \oplus x_0x_1$ which is matching the one given in Table 3.1.

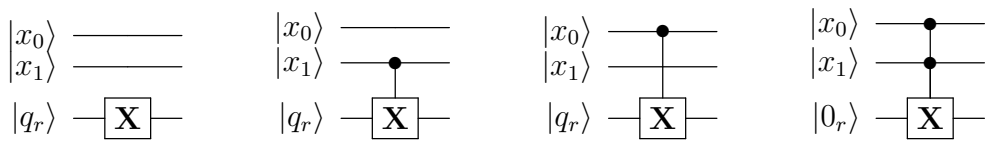
In this section, for $n \in \mathbb{N}$, we have equipped the set of the Boolean functions from $\mathbb{B}^n$ to $\mathbb{B}$ with the $\oplus$ operator and shown that it has a group structure. This allowed us to show that any function of $\mathbb{B}^{\mathbb{B}^n}$ can uniquely be expressed under a polynomial form that only involves the **AND** and **XOR** logical operator. From this preliminary work, we will derive a way of expressing any function of $\mathbb{B}^{\mathbb{B}^n}$ into a quantum circuit of $n+1$ qubits using exclusively multi-controlled $\mathbf{X}$ gates.

3.3 From Algebraic Normal Form to Quantum Circuit

Let $n \in \mathbb{N}$, and consider a quantum circuit operating on $n+1$ qubits separated in two registers $|x_0 \dots x_{n-1}\rangle\,|q_r\rangle$ where the first register of n qubits is the input register and the second register of one qubit is the read-out register.

Definition 3.9:
Let $u \in \mathbb{B}^n$, we denote $\mathbf{C}_u$ the multi-controlled $\mathbf{X}$ gates that is acting on the read-out qubit $|q_r\rangle$ and controlled by the qubits $\{|x_i\rangle \mid i \in 1_u\}$ where 1_u is as in Definition 3.3.

In Figure 3.1 are represented all the possible gates for the case $n = 2$. We now show some



(a) $\mathbf{C}_{00}$ (b) $\mathbf{C}_{01}$ (c) $\mathbf{C}_{10}$ (d) $\mathbf{C}_{11}$

Figure 3.1: The set of the possible $\mathbf{C}_u$ gates for $n = 2$

properties pertaining to this type of gate. These properties will result in a way of easily expressing any Boolean function through a quantum circuit composed exclusively of gates of the type defined in Definition 3.9.

Proposition 3.10:

Let $n \in \mathbb{N}$ and $u \in \mathbb{B}^n$ then:

$$\forall x \in \mathbb{B}^n, q_r \in \mathbb{B}, \mathbf{C}_u |x\rangle |q_r\rangle = |x\rangle |q_r \oplus m_u(x)\rangle \tag{3.15}$$

Where m_u is defined in Definition 3.5

Proof:

Let $n \in \mathbb{N}$ and $u \in \mathbb{B}^n$. Take $\mathbf{C}_u$ the quantum gate as in Definition 3.9.

Now let $x = x_0 \dots x_{n-1} \in \mathbb{B}^n, q_r \in \mathbb{B}$, $\mathbf{C}_u$ being the $\mathbf{X}$ gates controlled by the qubits $|x_i\rangle$ such that $i \in 1_u$, it will switch the value of the read-out qubit if and only if all the control qubits are in the state $|1\rangle$. This behaviour can be written as:

$$\mathbf{C}_u |x\rangle |q_r\rangle = |x\rangle \left| q_r \oplus \prod_{i \in 1_u} x_i \right\rangle = |x\rangle |q_r \oplus x^u\rangle = |x\rangle |q_r \oplus m_u(x)\rangle \tag{3.16}$$

■

Property 3.10 shows that a single gate $\mathbf{C}_u$ is able to express the monomial m_u on the read-out qubit. We now show that by combining multiple gates of this form we can express any Boolean function.

Definition 3.11:

Let $n \in \mathbb{N}$, we denote $\mathcal{C}_n = \{\mathbf{C}_u \mid u \in \mathbb{B}^n\}$ and $\mathcal{A}_n^Q$ the subgroup of the unitary group $U(2^{n+1})$ that is generated by $\mathcal{C}_n$.

Proposition 3.12:

Let $n \in \mathbb{N}$, and $\mathcal{A}_n^Q$ as in Definition 3.11, then $\mathcal{A}_n^Q$ is a finite commutative group where each element is its own inverse.

Proof:

Let $n \in \mathbb{N}$ and $u, v \in \mathbb{B}^n$.

Now let $x \in \mathbb{B}^n$ and $q_r \in \mathbb{B}$ then by using Property 3.10, we have:

$$\mathbf{C}_u \mathbf{C}_v |x\rangle |q_r\rangle = \mathbf{C}_u |x\rangle |q_r \oplus m_v(x)\rangle = |x\rangle |q_r \oplus m_v(x) \oplus m_u(x)\rangle \tag{3.17}$$

But $m_v(x) \oplus m_u(x) = m_u(x) \oplus m_v(x)$ so:

$$\mathbf{C}_u\mathbf{C}_v |x\rangle |q_r\rangle = |x\rangle |q_r \oplus m_u(x) \oplus m_v(x)\rangle = \mathbf{C}_v |x\rangle |q_r \oplus m_u(x)\rangle = \mathbf{C}_v\mathbf{C}_u |x\rangle |q_r\rangle \tag{3.18}$$

Hence: $\mathbf{C}_u\mathbf{C}_v = \mathbf{C}_v\mathbf{C}_u$

In addition:

$$\mathbf{C}_u\mathbf{C}_u |x\rangle |q_r\rangle = |x\rangle |q_r \oplus m_u(x) \oplus m_u(x)\rangle = |x\rangle |q_r \oplus 0\rangle = |x\rangle |q_r\rangle \tag{3.19}$$

So: $\mathbf{C}_u = \mathbf{C}_u^{-1}$ All these go to show that $\mathcal{A}_n^Q$ can be described as:

$$\mathcal{A}_n^Q = \left\{ \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u} \mid \forall u \in \mathbb{B}^n, \alpha_u \in \{0,1\} \right\} \tag{3.20}$$

So $\mathcal{A}_n^Q$ is finite and $\left|\mathcal{A}_n^Q\right| \leq 2^{2^n}$.

Let $\mathbf{A}, \mathbf{B} \in \mathcal{A}_n^Q$, then there exist $\{\alpha_u\}_{u\in\mathbb{B}^n} \in \{0,1\}^{2^n}$ and $\{\beta_u\}_{u\in\mathbb{B}^n} \in \{0,1\}^{2^n}$ such that $\mathbf{A} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u}$ and $\mathbf{B} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\beta_u}$.

The commutativity of the gates $\mathbf{C}_u$ leads to

$$\mathbf{AB} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u} \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\beta_u} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\beta_u} \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u} = \mathbf{BA} \tag{3.21}$$

And:

$$\mathbf{A}^2 = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u} \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u\oplus\alpha_u} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{0} = \mathbf{I} \tag{3.22}$$

So $\mathbf{A} = \mathbf{A}^{-1}$. ■

Now that we have established that for $n \in \mathbb{N}$, $\mathbb{B}^{\mathbb{B}^n}$ and the set $\mathcal{A}_n^Q$ have a similar group structure, we introduce a correspondence between these two groups that will make evident how a Boolean function can be expressed by such a quantum circuit.

Definition 3.13:

Let $n \in \mathbb{N}$, we introduce $\Phi : \mathbb{B}^{\mathbb{B}^n} \to \mathcal{A}_n^Q$ the group morphism such that:

$$\forall u \in \mathbb{B}^n, \Phi(m_u) = \mathbf{C}_u \tag{3.23}$$

Lemma 3.14:

Let $n \in \mathbb{B}^n$ and $f \in \mathbb{B}^{\mathbb{B}^n}$, then:

$$\forall x \in \mathbb{B}^n, q_r \in \mathbb{B}, \Phi(f)\left|x\right\rangle\left|q_r\right\rangle = \left|x\right\rangle\left|q_r \oplus f(x)\right\rangle \tag{3.24}$$

Proof:

Let $n \in \mathbb{N}$ and $f \in \mathbb{B}^{\mathbb{B}^n}$, from Theorem 3.7, f has an algebraic normal form. So there exist $\{\alpha_u\}_{u\in\mathbb{B}^n} \in \{0,1\}^{2^n}$ such that:

$$f = \bigoplus_{u\in\mathbb{B}^n} \alpha_u m_u \tag{3.25}$$

Φ being a group morphism, we have:

$$\Phi(f) = \Phi\left(\bigoplus_{u\in\mathbb{B}^n} \alpha_u m_u\right) = \prod_{u\in\mathbb{B}^n} \Phi(m_u)^{\alpha_u} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u} \tag{3.26}$$

So for $x \in \mathbb{B}^n$ and $q_r \in \mathbb{B}$, thanks to Property 3.10:

$$\Phi(f)\left|x\right\rangle\left|q_r\right\rangle = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u}\left|x\right\rangle\left|q_r\right\rangle = \left|x\right\rangle\left|q_r \oplus \bigoplus_{u\in\mathbb{B}^n} \alpha_u m_u(x)\right\rangle = \left|x\right\rangle\left|q_r \oplus f(x)\right\rangle \tag{3.27}$$

■

We can now show that for $n \in \mathbb{N}$, $\mathbb{B}^{\mathbb{B}^n}$ and $\mathcal{A}_n^Q$ are in fact equivalent:

Theorem 3.15:

Let $n \in \mathbb{N}$ and $\Phi : \mathbb{B}^{\mathbb{B}^n} \to \mathcal{A}_n^Q$ the group morphism introduced in Definition 3.13.

Φ is a group isomorphism.

Proof:

Let $n \in \mathbb{N}$.

Let $f, g \in \mathbb{B}^{\mathbb{B}^n}$ such that $\Phi(f) = \Phi(g)$. Let $x \in \mathbb{B}^n$, then from Lemma 3.14 we have:

$$\Phi(f)\left|x\right\rangle\left|0\right\rangle = \left|x\right\rangle\left|f(x)\right\rangle \text{ and } \Phi(g)\left|x\right\rangle\left|0\right\rangle = \left|x\right\rangle\left|g(x)\right\rangle \tag{3.28}$$

But $\Phi(f) = \Phi(g)$ so $\left|x\right\rangle\left|f(x)\right\rangle = \left|x\right\rangle\left|g(x)\right\rangle$, that is $f(x) = g(x)$.

Hence $f = g$.

Let $\mathbf{A} \in \mathcal{A}_n^Q$, then there exists $\{\alpha_u\}_{u\in\mathbb{B}^n} \in \{0,1\}^{2^n}$ such that $\mathbf{A} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u}$. Let us denote $f = \bigoplus_{u\in\mathbb{B}^n} \alpha_u m_u$, then $f \in \mathbb{B}^{\mathbb{B}^n}$ and we have:

$$\Phi(f) = \Phi\left(\bigoplus_{u\in\mathbb{B}^n} \alpha_u m_u\right) = \prod_{u\in\mathbb{B}^n} \Phi(m_u)^{\alpha_u} = \prod_{u\in\mathbb{B}^n} \mathbf{C}_u^{\alpha_u} = \mathbf{A} \tag{3.29}$$

So all in all, Φ is an isomorphism between $\left(\mathbb{B}^{\mathbb{B}^n}, \oplus\right)$ and $\left(\mathcal{A}_n^Q, .\right)$ ■

Theorem 3.15 ensures that for $n \in \mathbb{N}$, if we take $f \in \mathbb{B}^{\mathbb{B}^n}$, a Boolean function, then it is possible to construct a quantum circuit that will express this function on the read out qubit. To do so we only need to decompose f into its algebraic normal form, transform each monomial of the ANF into its corresponding multi-controlled $\mathbf{X}$ gate and add them sequentially to the quantum circuit. Theorem 3.15 also ensure that the resulting circuit is unique, up to permutations on the order of the gates. We illustrate the construction of such quantum circuits in Examples 3.16 and 3.17

Example 3.16:

Let us build the quantum circuit expressing the function $\mathbf{OR} \in \mathbb{B}^{\mathbb{B}^2}$ from Example 3.8, its truth table being given in Table 3.2. From Example 3.8, we know that:

$$\mathbf{OR} = m_{01} \oplus m_{10} \oplus m_{11} \tag{3.30}$$

Applying Φ, we get:

$$\Phi(\mathbf{OR}) = \mathbf{C}_{01}\mathbf{C}_{10}\mathbf{C}_{11} \tag{3.31}$$

That is, the quantum circuit corresponding to **OR** is given in Figure 3.2

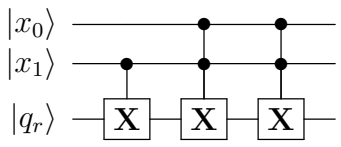


Figure 3.2: Quantum circuit corresponding to **OR**

Example 3.17:

Here we will construct the circuit corresponding to the function $f \in \mathbb{B}^{\mathbb{B}^3}$ for which the values are given in Table 3.3.

x	000	001	010	011	100	101	110	111
$f(x)$	1	0	1	0	0	0	1	1

Table 3.3: Truth table of f

We apply the method of indeterminate coefficients to find the algebraic normal form of f:

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \text{ and } \mathbf{c}^f = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{3.32}$$

So $\mathbf{v}^f = (1, 1, 0, 0, 1, 1, 1, 0)^T$ that is:

$$f = m_{000} \oplus m_{001} \oplus m_{100} \oplus m_{101} \oplus m_{110} \tag{3.33}$$

Applying Φ to get the quantum circuit, we have:

$$\Phi(f) = \mathbf{C}_{000}\mathbf{C}_{001}\mathbf{C}_{100}\mathbf{C}_{101}\mathbf{C}_{110} \tag{3.34}$$

Which is represented in Figure 3.3

In this section, for $n \in \mathbb{N}$, we have shown that the set of the Boolean functions $\mathbb{B}^{\mathbb{B}^n}$ is equivalent to the set of quantum circuits $\mathcal{A}_n^Q$. This set contains the quantum circuits of $n+1$ qubits made of $\mathbf{X}$ gates controlled by subsets of the n first qubits and acting on the read-out qubit. This equivalence is such that for $f \in \mathbb{B}^{\mathbb{B}^n}$, there exists a unique corresponding quantum circuit and this circuit is able to express f.

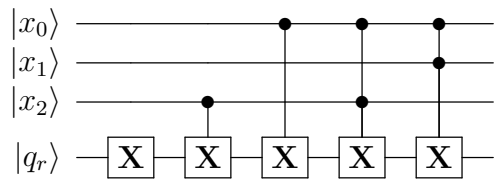


Figure 3.3: Quantum circuit corresponding to f

In this thesis, we only focus on multi-controlled **X** gates to perform the computations. This type of gates represents an abstraction in the sense that they are not considered to be basic gates but they can nonetheless be implemented with basic gates such as Toffoli gates and some additional ancillary qubits. The reason we chose to work with this abstraction that are multi-controlled **X** gates is that, in our opinion, they are the best and most direct way to visualise the equivalence between the ANF of a function and the quantum circuit expressing this function.

From this equivalence, we introduce a type of quantum circuit that we called tunable quantum neural network. This type of circuit will be the subject of the studies conducted in this thesis.

3.4 Tunable Quantum Neural Network

Let $n \in \mathbb{N}$ and suppose we wish to express a Boolean function of $\mathbb{B}^{\mathbb{B}^n}$ with a quantum circuit. From Section 3.3, we know this can be achieved with a circuit of $n+1$ qubit, made of **X** gates controlled by subsets of the n first qubits and acting on the read-out qubit. We formalise this approach in the form of a circuit ansatz that we called tunable quantum neural network (TNN). This ansatz is built from multi-controlled gates acting on the read-out qubit, the action being switchable between the identity and the **X** gate.

Formally, we consider the quantum circuit of $n+1$ qubits $|x_0 \dots x_{n-1}\rangle |q_r\rangle$ and we define the following tunable gate:

Definition 3.18:

Let $n \in \mathbb{N}$ and $u \in \mathbb{B}^n$. The tunable gate $\mathbf{G}_u(\mathbf{A}_u)$ is the gate controlled by the set of qubits:

$$\{|x_i\rangle \mid i \in 1_u\} \tag{3.35}$$

Where 1_u is defined in Definition 3.3. It is acting on the read-out qubit $|q_r\rangle$ and its action, $\mathbf{A}_u$, on this qubit can be switched between the identity $\mathbf{I}$ or the NOT $\mathbf{X}$ gate.

A tunable quantum neural network $\mathbf{T}$ is then defined by:

$$\mathbf{T} = \prod_{u \in \mathbb{B}^n} \mathbf{G}_u(\mathbf{A}_u) \tag{3.36}$$

For $u \in \mathbb{B}^n$, $\mathbf{G}_u(\mathbf{I}) = \mathbf{I}$ or $\mathbf{G}_u(\mathbf{X}) = \mathbf{C}_u$ so from Property 3.12 it follows that the order with which the gates $\mathbf{G}_u$ are added to the circuit does not change the overall result. Additionally, as:

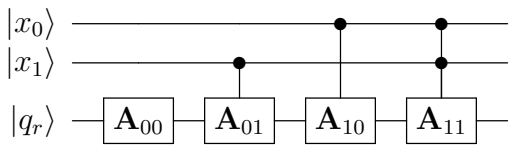
$$\left\{ \prod_{u \in \mathbb{B}^n} \mathbf{G}_u(\mathbf{A}_u) \mid \forall u \in \mathbb{B}^n, \mathbf{A}_u \in \{\mathbf{I}, \mathbf{X}\} \right\} = \left\{ \prod_{u \in \mathbb{B}^n} \mathbf{G}_u \mid \forall u \in \mathbb{B}^n, \mathbf{G}_u \in \{\mathbf{I}, \mathbf{C}_u\} \right\} = \mathcal{A}_n^Q \tag{3.37}$$

We are assured that any Boolean function of $\mathbb{B}^{\mathbb{B}^n}$ can be expressed by a TNN of $n+1$ qubits.

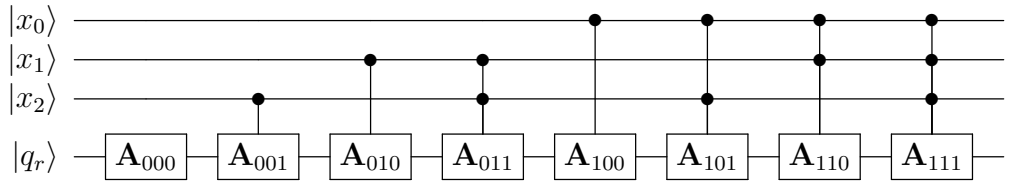
We chose to call this ansatz tunable quantum neural network as similarly to the classical case, when taken in isolation, each gate is performing a simple computation on its inputs but when put into a network, the sequence of gates is able to compute any Boolean function no matter how complex it can be, it suffices to tune it accordingly. Figure 3.4 depicts TNN in the case $n = 2$ and $n = 3$ while in Figure are represented networks tuned to express the Boolean functions from Examples 3.16 and 3.17.

3.5 Conclusion

In this chapter, we have introduced a quantum circuit ansatz made of $\mathbf{X}$ gates controlled by subsets of the input qubits and acting on a common ancillary qubit. We have then shown that the set of circuits of this type is isomorphic to the set of Boolean functions. This means

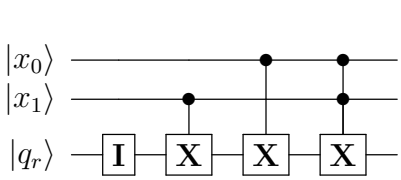


(a) Tunable neural network for $n = 2$

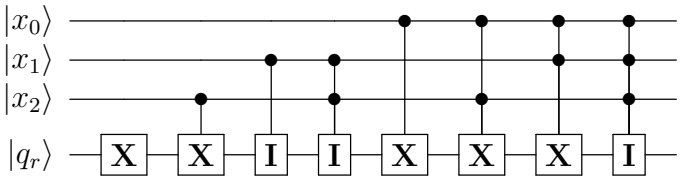


(b) Tunable neural network for $n = 3$

Figure 3.4: Tunable neural networks for $n = 2$ and $n = 3$



(a) Tuned network for Example 3.16



(b) Tuned network for Example 3.17

Figure 3.5: Tuned networks for Example 3.16 and 3.17

that, provided it is properly tuned, this ansatz is able to express any Boolean function. In the following chapters, we will study different ways to perform this tuning process.

Chapter 4

Learning Boolean Functions with a Specific Superposition

4.1 Introduction

In this chapter, we introduce an algorithm to train a tunable quantum neural network to learn a Boolean function. While [87] introduced an algorithm to construct a correctly tuned network, it required access to the whole truth table of the target function and involved the use of positively and negatively controlled $\mathbf{X}$ gates that are then transformed into regular multi-controlled $\mathbf{X}$ gates. In contrast, our algorithm is more in line with the proceedings of machine learning tasks. In our setup, the network is given access to an oracle that provides a superposition of the inputs, entangled with their respective image by the function. The goal is to progressively tune the network so that the function it is expressing converges to the target function. When the algorithm stops, the network should express a function that is point wise equal to the target function. To do so, we present the algorithm and prove that it will correctly stop, moreover, we show that the algorithm will stop after at most $n+1$ updates when n is the number of variables. We present a version of this algorithm when the oracle produces a superposition with specific amplitudes that allows for a better identification of the misclassified inputs. Finally we also implement it for $n = 3$ and $n = 4$ and verify that the number of updates to the network is

indeed at most $n+1$.

4.2 Learning Algorithm

4.2.1 Presentation of the Algorithm

Let $n \in \mathbb{N}$ and $f \in \mathbb{B}^{\mathbb{B}^n}$, in this chapter we introduce an algorithm resulting in a correctly tuned neural network expressing f. We first outline the general structure of this algorithm and then go further into the implementation details. This algorithm focuses on reducing the number of updates to the circuit during the training phase. For $f \in \mathbb{B}^{\mathbb{B}^n}$, suppose we are given an oracle $\mathbf{O}(f)$ that outputs $|\psi\rangle$:

$$|\psi\rangle = \sum_{x \in \mathbb{B}^n} a_x |x\rangle |f(x)\rangle \tag{4.1}$$

Where $a_x \in \mathbb{C}$ for $x \in \mathbb{B}^n$. Let $\mathbf{T}(h)$ be a tunable neural network such that it expresses the function $h \in \mathbb{B}^{\mathbb{B}^n}$ in its current state, then:

$$\mathbf{T}(h) |\psi\rangle = \sum_{x \in \mathbb{B}^n} a_x |x\rangle |f(x) \oplus h(x)\rangle = \sum_{f(x)=h(x)} a_x |x\rangle |0\rangle + \sum_{f(x)\neq h(x)} a_x |x\rangle |1\rangle \tag{4.2}$$

So the inputs for which their output by the tunable neural network is different from the target function are such that the read-out qubit is in state $|1\rangle$. Now suppose that we have a way of identifying such inputs in this superposition, that is:

Definition 4.1:

Let ei (error identification) be the operation such that:

$$ei\left[\sum_{x \in \mathbb{B}^n} b_x |x\rangle |r_x\rangle\right] = \{x \in \mathbb{B}^n \mid b_x \neq 0, r_x = 1\} \tag{4.3}$$

Then the tuning algorithm would work as follow: starting from $\mathbf{T}^{(0)}$ the tunable network with all the gates initialised with the identity gate $\mathbf{I}$, for $k \in \mathbb{N}$, let $\mathbf{T}^{(k)}$ be the state of the network following the k-th update. We denote $E^{(k)} = ei\left[\mathbf{T}^{(k)} |\psi\rangle\right]$ the set of the misclassified inputs by

$\mathbf{T}^{(k)}$, then an update step can be described as:

1. Determine $E^{(k)}$
2. For $u \in E^{(k)}$ switch the action of the corresponding gate $\mathbf{G}_u$

Once the update step is completed, the resulting network is denoted $\mathbf{T}^{(k+1)}$. Switching the action of the gate $\mathbf{G}_u$ can be performed by multiplying it by $\mathbf{C}_u$ due to the fact that $\mathbf{C}_u = \mathbf{C}_u^{-1}$. The algorithm terminates when the tunable network reaches a state $\mathbf{T}^{(h)}$ such that $E^{(h)} = \varnothing$. The learning algorithm is outlined in Algorithm 1.

Algorithm 1: Learning algorithm

Input: The superposition $|\psi\rangle$ created by the oracle
Output: A correctly tuned network $\mathbf{T}$
for $u \in \mathbb{B}^n$ **do**
  $\mathbf{G}_u \leftarrow \mathbf{I}$;
end
$\mathbf{T} \leftarrow \prod_{u \in \mathbb{B}^n} \mathbf{G}_u$;
$E \leftarrow ei\left[\mathbf{T}\,|\psi\rangle\right]$;
while $E \neq \varnothing$ **do**
  for $u \in E$ **do**
    $\mathbf{G}_u \leftarrow \mathbf{G}_u \mathbf{C}_u$;
  end
  $\mathbf{T} \leftarrow \prod_{u \in \mathbb{B}^n} \mathbf{G}_u$;
  $E \leftarrow ei(f)\left[\mathbf{T}\,|\psi\rangle\right]$;
end

Example 4.2:

Let us run this algorithm to learn the function introduced in Example 3.17, the values of which are gathered in Table 3.3. Suppose the oracle $\mathbf{O}(f)$ produces $|\psi\rangle = \sum_{x \in \mathbb{B}^n} a_x\,|x\rangle\,|f(x)\rangle$ and we assume further that for $x \in \mathbb{B}^n, a_x \neq 0$.

As stated in Algorithm 1, we start with all the gates initialised to $\mathbf{I}$, hence $\mathbf{T}^{(0)} = \mathbf{I}$ and the function $h^{(0)}$ expressed by the network is $h^{(0)} = 0$. The effect of the network on the inputs of the oracle's output is given in Table 4.1.

This leads to $E^{(0)} = \{000, 010, 110, 111\}$. The update step then yields:

$$\mathbf{T}^{(1)} = \mathbf{C}_{000}\mathbf{C}_{010}\mathbf{C}_{110}\mathbf{C}_{111} \tag{4.4}$$

x	000	001	010	011	100	101	110	111
$h^{(0)}(x)$	0	0	0	0	0	0	0	0
$(f \oplus h^{(0)})(x)$	1	0	1	0	0	0	1	1

Table 4.1: Truth table of $h^{(0)}$ and $f \oplus h^{(0)}$

In this new state, the network now expresses the function $h^{(1)}$. We have gathered the values of this function as well as those of $f \oplus h^{(1)}$ in Table 4.2:

x	000	001	010	011	100	101	110	111
$h^{(1)}(x)$	1	1	0	0	1	1	1	0
$(f \oplus h^{(1)})(x)$	0	1	1	0	1	1	0	1

Table 4.2: Truth table of $h^{(1)}$ and $f \oplus h^{(1)}$

From this state of the network, we thus have $E^{(1)} = \{001, 010, 100, 101, 111\}$, leading to:

$$\mathbf{T}^{(2)} = \mathbf{T}^{(1)}\mathbf{C}_{001}\mathbf{C}_{010}\mathbf{C}_{100}\mathbf{C}_{101}\mathbf{C}_{111} \tag{4.5}$$

$$= \mathbf{C}_{000}\mathbf{C}_{001}(\mathbf{C}_{010}\mathbf{C}_{010})\mathbf{C}_{100}\mathbf{C}_{101}\mathbf{C}_{110}(\mathbf{C}_{111}\mathbf{C}_{111}) \tag{4.6}$$

$$= \mathbf{C}_{000}\mathbf{C}_{001}\mathbf{C}_{100}\mathbf{C}_{101}\mathbf{C}_{110} \tag{4.7}$$

Continuing, we have $E^{(2)} = \varnothing$ so the algorithm terminates and we have found a network that correctly expresses the target function as it matches its corresponding quantum circuit as shown in Figure 3.3.

4.2.2 Proof of Termination and Correctness

We now properly define an oracle $\mathbf{O}(f)$ mentioned earlier. When queried, this oracle outputs the type of superposition that we will need:

Definition 4.3:

Let $n \in \mathbb{N}$ and $f \in \mathbb{B}^{\mathbb{B}^n}$, we define $\mathbf{O}(f)$ to be the oracle such that a call to it provides the superposition:

$$|\psi\rangle = \sum_{x \in \mathbb{B}^n} a_x |x\rangle |f(x)\rangle \tag{4.8}$$

Where $a_x \neq 0$ for $x \in \mathbb{B}^n$.

We show here that when provided with such a superposition, the algorithm will terminate after at most n+1 update steps and the resulting network is properly tuned to express f.

Lemma 4.4:

Let $n \in \mathbb{N}$ and $x \in \mathbb{B}^n$, we denote by $\mathcal{T}_x$ the set of the controlled gates that can be triggered by $|x\rangle$ and $w_H(x) = |1_x|$ the Hamming weight of x. Then:

$$\forall x \in \mathbb{B}^n,\ \mathcal{T}_x \subseteq \{\mathbf{G}_x\} \cup \{\mathbf{G}_u \mid w_H(u) < w_H(x)\} \tag{4.9}$$

Proof:

Let $x \in \mathbb{B}^n$. For $u \in \mathbb{B}^n$, by Definition 3.18, the gate $\mathbf{G}_u$ is controlled by the set of qubits $\{|x_i\rangle \mid i \in 1_u\}$ so x triggers $\mathbf{G}_u$ if and only if $x_i = 1$ for $i \in 1_u$ that is to say if and only if $1_u \subseteq 1_x$. So either $1_u = 1_x$ and $u = x$ or $1_u \subset 1_x$ and in this case we have $w_H(u) < w_H(x)$. This leads to $\mathcal{T}_x \subseteq \{\mathbf{G}_x\} \cup \{\mathbf{G}_u \mid w_H(u) < w_H(x)\}$. ■

From Lemma 4.4 we naturally have:

Corollary 4.5:

Let $n \in \mathbb{N}$, $x \in \mathbb{B}^n$, $q_r \in \mathbb{B}$ and $\mathbf{T} = \prod_{u \in \mathbb{B}^n} \mathbf{G}_u$ a tunable neural network. Then:

$$\mathbf{T}\,|x\rangle\,|q_r\rangle = \mathbf{G}_x \prod_{w_H(u) < w_H(x)} \mathbf{G}_u\,|x\rangle\,|q_r\rangle \tag{4.10}$$

This result will be useful to show the following:

Lemma 4.6:

Let $k \in \mathbb{N}^*$. Suppose that we are tuning the network according to Algorithm 1, then following the k-th update, the gates controlled by at most k-1 qubits will not be updated anymore.

Proof:

We show this result by induction over k.

Let $n \in \mathbb{N}$ and $k \in \mathbb{N}^*$.

Suppose that in the general case, the oracle produces the superposition

$$|\psi\rangle = \sum_{x \in \mathbb{B}^n} a_x\,|x\rangle\,|q_x\rangle \tag{4.11}$$

We denote $\mathbf{T}^{(k)} = \prod_{u\in\mathbb{B}^n} \mathbf{G}_u^{(k)}$ the state of the network after the k-th update.

For $k = 1$:

Before the first update, the network is in the state $\mathbf{T}^{(0)}$. From Corollary 4.5 we have:

$$\mathbf{T}^{(0)} |0\rangle |q_0\rangle = \mathbf{G}_0^{(0)} |0\rangle |q_0\rangle = |0\rangle |q_0'\rangle \tag{4.12}$$

From the update rule, if $0 \in E^{(0)}$, meaning $|q_0'\rangle = |1\rangle$ and $a_x \neq 0$, we switch the action of $\mathbf{G}_0^{(0)}$. If not, we keep it the same. Either way, let us denote $\mathbf{G}_0$ the resulting gate and we have:

$$\mathbf{T}^{(1)} = \prod_{u\neq 0} \mathbf{G}_u^{(1)} \mathbf{G}_0 \tag{4.13}$$

With

$$\mathbf{T}^{(1)} |0\rangle |q_0\rangle = \mathbf{G}_0 |0\rangle |q_0\rangle = |0\rangle |0\rangle \tag{4.14}$$

This means that $0 \notin E^{(1)}$ and so $\mathbf{G}_0$ will not be updated further. Let $\mathbf{T}^{(k)} = \prod_{u\in\mathbb{B}^n} \mathbf{G}_u^{(k)}$ the state of the network following the k-th update. From the induction hypothesis, we have:

$$\mathbf{T}^{(k)} = \prod_{w_H(u)\geq k} \mathbf{G}_u^{(k)} \prod_{w_H(u)<k} \mathbf{G}_u \tag{4.15}$$

Let $x \in \mathbb{B}^n$ such that $w_H(x) = k$, from Corollary 4.5 we have:

$$\mathbf{T}^{(k)} |x\rangle |q_x\rangle = \mathbf{G}_x^{(k)} \prod_{w_H(u)<k} \mathbf{G}_u |x\rangle |q_x\rangle = |x\rangle |q_x'\rangle \tag{4.16}$$

Once again, if $x \in E^{(k)}$, i.e. $|q_x'\rangle = |1\rangle$ and $a_x \neq 0$, then the action of $\mathbf{G}_x^{(k)}$ is switched, otherwise it remains unchanged. Either way, we denote $\mathbf{G}_x$ the resulting gate and we have:

$$\mathbf{T}^{(k+1)} |x\rangle |q_x\rangle = \mathbf{G}_x \prod_{w_H(u)<k} \mathbf{G}_u |x\rangle |q_x\rangle = |x\rangle |0\rangle \tag{4.17}$$

This leads to $x \notin E^{(k+1)}$ meaning that $\mathbf{G}_x$ will not be updated at the $k+1$-th update nor any further updates. ■

From Lemma 4.6 stems the following:

Corollary 4.7:

Let $n \in \mathbb{N}$ and $k \leq n$. Let $E^{(k)}$ be the set of misclassified inputs after the k-th update, then:

$$E^{(k)} \subseteq \{x \in \mathbb{B}^n \mid w_H(x) \geq k\} \tag{4.18}$$

Proof:

Let $n \in \mathbb{N}$ and $k \geq n$. Suppose there exist $x \in \mathbb{B}^n$ such that $w_H(x) = s < k$ and $x \in E^{(k)}$. Then according to the update rule, $\mathbf{G}_x$ will be updated at the $k+1$-th update. This is in contradiction with Lemma 4.6 as $\mathbf{G}_x$ is controlled by $s \leq k-1$ qubits. ■

We recall that the tuning algorithm terminates when it reaches a state $h \in \mathbb{N}$ such that $E^{(h)} = \varnothing$. The previous results lead to the following:

Theorem 4.8:

Let $n \in \mathbb{N}$. The training of a TNN of n inputs will terminate after at most $n+1$ updates.

Proof:

This is a direct consequence of Corollary 4.7. After the $n+1$-th update the set of misclassified inputs $E^{(n+1)}$ is such that

$$E^{(n+1)} \subseteq \{x \in \mathbb{B}^n \mid w_H(x) \geq n+1\} \tag{4.19}$$

But $\{x \in \mathbb{B}^n \mid w_H(x) \geq n+1\} = \varnothing$ so $E^{(n+1)} = \varnothing$ and the algorithm terminates. ■

Finally, the following ensures that when provided with a superposition of all the possible inputs, the training will result in a correctly tuned network which will be a translation of the algebraic normal form of the target function.

Theorem 4.9:

Let $n \in \mathbb{N}$ and $f \in \mathbb{B}^{\mathbb{B}^n}$ be the target function. Suppose we are training a tunable neural network on the superposition produced by the oracle $\mathbf{O}(f)$ as in Definition 4.3:

$$|\psi\rangle = \sum_{x \in \mathbb{B}^n} a_x |x\rangle |f(x)\rangle \tag{4.20}$$

With $a_x \neq 0$ for $x \in \mathbb{B}^n$. Then when the training terminates, the network is a translation of the algebraic normal form of f.

Proof:

Let $n \in \mathbb{N}$, $f \in \mathbb{B}^{\mathbb{B}^n}$ the target function and $|\psi\rangle$ the superposition as in (4.20). Suppose further that the algorithm terminated and that the network is in its final state $\mathbf{T}$. By design, the algorithm stops when the set of misclassified inputs E is empty. Because we suppose that $a_x \neq 0$ for $x \in \mathbb{B}^n$, this means:

$$\forall x \in \mathbb{B}^n,\ \mathbf{T}\,|x\rangle\,|f(x)\rangle = |x\rangle\,|0\rangle \tag{4.21}$$

By construction:

$$\mathbf{T} = \prod_{u \in \mathbb{B}^n} \mathbf{C}_u^{\alpha_u} \tag{4.22}$$

Where for $u \in \mathbb{B}^n, \alpha_u \in \{0,1\}$, so $\mathbf{T} \in \mathcal{A}_n^Q$. Let Φ the isomorphism introduced in Definition 3.13 let us denote

$$\tilde{f} = \Phi^{-1}(\mathbf{T}) \tag{4.23}$$

Then for $x \in \mathbb{B}^n$ and $q_x \in \mathbb{B}$, we have

$$\mathbf{T}\,|x\rangle\,|q_x\rangle = |x\rangle\left|q_x \oplus \tilde{f}(x)\right\rangle \tag{4.24}$$

Together with (4.21), this means that for $x \in \mathbb{B}^n$:

$$|x\rangle\left|f(x) \oplus \tilde{f}(x)\right\rangle = |x\rangle\,|0\rangle \tag{4.25}$$

Hence

$$\forall x \in \mathbb{B}^n,\ \tilde{f}(x) = f(x) \tag{4.26}$$

That is $\tilde{f} = f$ and in its final state, the network correctly expresses the target function. ■

Here we have proven that the tuning algorithm terminates and when it does, the network is correctly tuned to express the target function on the read-out qubit. Nevertheless, this is all conditioned on the fact that we are able to perform the operation denoted *ei* that identifies all

the inputs present in the superposition that have been erroneously classified by the network. We will not present two ways this operation can be achieved.

4.3 Identifying Errors via Maximum Likelihood Estimation

As previously, suppose that we want to train a tunable network to express a Boolean function while given access to an oracle $\mathbf{O}(f)$ as in Definition 4.3. Suppose further that in its current state, the network is expressing the function $h \in \mathbb{B}^{\mathbb{B}^n}$. Let us denote $\mathbf{T}(h)$ this state, then we have:

$$\mathbf{T}(h)\,|\psi\rangle = \sum_{f(x)=h(x)} a_x\,|x\rangle\,|0\rangle + \sum_{f(x)\neq h(x)} a_x\,|x\rangle\,|1\rangle \tag{4.27}$$

Then the probability P_1 of measuring $|1\rangle$ on the read-out qubit is:

$$P_1 = \sum_{f(x)\neq h(x)} |a_x|^2 \tag{4.28}$$

This means that by using a suitable amplitude encoding and by estimating P_1 it is possible to retrieve the inputs that have been misclassified by the network hence realising the operation ei.

4.3.1 Building a Suitable Amplitude Encoding

Suppose we can accurately enough estimate P_1. Here we can limit ourselves to real amplitudes and we want a set of amplitudes $\{a_x \in \mathbb{R} \mid x \in \mathbb{B}^n\}$ so that there exists a unique subset $S \subseteq \mathbb{B}^n$ such that:

$$P_1 = \sum_{x\in S} a_x^2 \tag{4.29}$$

This goal can be achieved by leveraging the uniqueness of the binary decomposition. For $x \in \mathbb{B}^n$, we denote $\bar{x} \in \mathbb{N}$ its conversion in the decimal system and we consider the superposition:

$$|\psi\rangle = \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{\bar{x}}} \, |x\rangle \, |f(x)\rangle \tag{4.30}$$

The uniqueness of the binary decomposition yields:

$$\forall S, S' \subseteq \mathbb{B}^n, S \neq S' \iff \frac{1}{2^{2^n}-1} \sum_{x \in S} 2^{\bar{x}} \neq \frac{1}{2^{2^n}-1} \sum_{x \in S'} 2^{\bar{x}} \tag{4.31}$$

And for $p \in [0, 1]$ we have:

$$(2^{2^n}-1)p \in \mathbb{N} \Rightarrow \exists! S \subseteq \mathbb{B}^n, \sum_{x \in S} 2^{\bar{x}} = (2^{2^n}-1)p \tag{4.32}$$

Which is what we are looking for. However, we recall that when running the tuning algorithm, the gates controlled by the least number of qubits are tuned to their definitive value early in the process. We thus want to reflect this behaviour in the superposition by granting a larger amplitude to inputs with a small Hamming weight. Later on, this superposition can then be switched with another where inputs with a large Hamming weight have a larger amplitude to finish the tuning process.

Definition 4.10:
Let $n \in \mathbb{N}$ and for $h \in [\![0, n]\!]$ we denote $W_h = \{x \in \mathbb{B}^n \mid w_H(x) = h\}$ and for $x \in W_h$, we introduce $\omega_h(x)$ its rank in W_h, induced by the natural order $\leq$ on $\mathbb{N}$ and applied to $\bar{x}$.

Examples of this set of functions are given in Figure 4.1 for the case $n = 4$.

We now define a function σ, the goal of which is to introduce an order relation in the Boolean strings that takes into account their Hamming weight. While there exist other ways of defining this order relation, we chose this one as it takes into account both the Hamming weight as well as the natural order on $\mathbb{N}$.

Definition 4.11:

x	$\omega_0(x)$
0000	1

(a) Values of ω_0

x	$\omega_1(x)$
0001	1
0010	2
0100	3
1000	4

(b) Values of ω_1

x	$\omega_2(x)$
0011	1
0101	2
0110	3
1001	4
1010	5
1100	6

(c) Values of ω_2

x	$\omega_3(x)$
0111	1
1011	2
1101	3
1110	4

(d) Values of ω_3

x	$\omega_4(x)$
1111	1

(e) Values of ω_4

Figure 4.1: The set of functions ω_h for $n = 4$

Let $n \in \mathbb{N}$, we define the function $\sigma : \mathbb{B}^n \to [\![0, 2^n - 1]\!]$ by:

$$\begin{cases} \sigma(0) = 0 \\ \sigma(x) = \max_{y \in W_{w_H(x)-1}} \sigma(y) + \omega_{w_H(x)}(x) \end{cases} \tag{4.33}$$

The values of $\sigma(x)$ for $x \in \mathbb{B}^4$ are given in Table 4.3.

x	$\sigma(x)$	x	$\sigma(x)$
0000	0	1000	4
0001	1	1001	8
0010	2	1010	9
0011	5	1011	12
0100	3	1100	10
0101	6	1101	13
0110	7	1110	14
0111	11	1111	15

Table 4.3: Values of $\sigma(x)$ for $x \in \mathbb{B}^4$

Using the function σ allows us to introduce the followings:

Definition 4.12:

Let $n \in \mathbb{N}$ and $f \in \mathbb{B}^{\mathbb{B}^n}$, we define $\mathbf{O}_{\downarrow}(f)$ and $\mathbf{O}_{\uparrow}(f)$ the oracles respectively producing the superpositions

$$|\psi_{\downarrow}\rangle = \frac{1}{\sqrt{2^{2^n} - 1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n - 1 - \sigma(x)}} \, |x\rangle \, |f(x)\rangle \tag{4.34}$$

And

$$|\psi_{\uparrow}\rangle = \frac{1}{\sqrt{2^{2^n} - 1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{\sigma(x)}} \, |x\rangle \, |f(x)\rangle \tag{4.35}$$

By using these superpositions we can mitigate some of the imprecision arising from the estimation of P_1. By first working with $|\psi_\downarrow\rangle$ we are more likely to identify the inputs with small Hamming weights that have been misclassified by the network. This, along with Lemma 4.6 ensures that the gates controlled by a small number of qubits are correctly tuned early in the tuning process. Once these gates have been set correctly, we can switch to using $|\psi_\uparrow\rangle$ in order to tune the gates controlled by a larger number of qubits.

4.3.2 Estimating P_1 and Identifying the Errors

As previously, for $n \in \mathbb{N}$, suppose the target function to be $f \in \mathbb{B}^{\mathbb{B}^n}$ and in its current state, the TNN is expressing the function $h \in \mathbb{B}^{\mathbb{B}^n}$ and is noted $\mathbf{T}(h)$, then as in (4.27) we have:

$$\mathbf{T}(h)\,|\psi\rangle = \sum_{f(x)=h(x)} a_x\,|x\rangle\,|0\rangle + \sum_{f(x)\neq h(x)} a_x\,|x\rangle\,|1\rangle$$

Where $|\psi\rangle$ is given in (4.8). Thus measuring the read-out qubit in the **Z**-basis can be modelled as a Bernoulli process with the probability of measuring 1 being as in (4.28):

$$P_1 = \sum_{f(x)\neq h(x)} |a_x|^2$$

Lemma 4.13:

Suppose we are measuring the read-out qubit s times. Out of this measurements, the state $|1\rangle$ is measured N_1 times, then the maximum likelihood estimation $\tilde{P_1}$ of P_1 is:

$$\tilde{P_1} = \frac{N_1}{s} \tag{4.36}$$

And for $\epsilon > 0$, taking

$$s = \frac{1}{\epsilon^2} \tag{4.37}$$

Will result in an estimation $\tilde{P_1}$ such that $P(|P_1 - \tilde{P_1}| \leq \epsilon) \geq 0.95$

Proof:

These are quite standard results in probability and statistics but for the sake of completeness

we give a proof.

For this experiment, the likelihood $L(p)$ is given by:

$$L(p) = p^{N_1}(1-p)^{s-N_1} \tag{4.38}$$

Differentiating the likelihood yields a maximum reached for

$$p = \frac{N_1}{s} = \tilde{P}_1 \tag{4.39}$$

The 95% confidence interval for this experiment is given by [80]:

$$\left[\tilde{P}_1 - z_{0.975}\sqrt{\frac{\tilde{P}_1(1-\tilde{P}_1)}{s}}, \tilde{P}_1 + z_{0.975}\sqrt{\frac{\tilde{P}_1(1-\tilde{P}_1)}{s}}\right] \tag{4.40}$$

Now let $\epsilon > 0$, we want:

$$\left|\tilde{P}_1 - P_1\right| \leq z_{0.975}\sqrt{\frac{\tilde{P}_1(1-\tilde{P}_1)}{s}} \leq \epsilon \tag{4.41}$$

That is

$$s \geq \frac{z_{0.975}^2 \tilde{P}_1(1-\tilde{P}_1)}{\epsilon^2} \tag{4.42}$$

But

$$\tilde{P}_1(1-\tilde{P}_1) \leq \frac{1}{4} \text{ and } z_{0.975} = 1.96 < 2 \tag{4.43}$$

So

$$\frac{z_{0.975}^2 \tilde{P}_1(1-\tilde{P}_1)}{\epsilon^2} < \frac{1}{\epsilon^2} \tag{4.44}$$

Thus by taking:

$$s = \frac{1}{\epsilon^2} \tag{4.45}$$

We do have

$$P(|P_1 - \tilde{P}_1| \leq \epsilon) \geq 0.95 \tag{4.46}$$

■

Recall that we work with the superposition $|\psi_\downarrow\rangle$ to first identify the misclassified inputs of low

Hamming weight and once those have been correctly classified, we switch to $|\psi_\uparrow\rangle$. Because $|\psi_\downarrow\rangle$ is defined such that the inputs of low Hamming weight have the largest amplitude, they will contribute more to the estimation of the probability P_1. This means that we only need to estimate P_1 accurately enough to identify the inputs that have the largest contribution. We set this accuracy so that half of the inputs in $|\psi_\downarrow\rangle$, the ones with the largest amplitude, can correctly be identified. Once they have been identified, we can switch the input superposition to $|\psi_\uparrow\rangle$.

Theorem 4.14:

Let $n \in \mathbb{N}$. In order to correctly tune the network to express $f \in \mathbb{B}^{\mathbb{B}^n}$ with high confidence, s calls to the oracles $\mathbf{O}_\downarrow(f)$ or $\mathbf{O}_\uparrow(f)$ have to be made at each update step where:

$$s = O\left(2^{2^n}\right) \tag{4.47}$$

Proof:

Because we are working either $|\psi_\downarrow\rangle$ and $|\psi_\uparrow\rangle$, as said previously, it suffices to identify up to half of the inputs present in the superpositions, namely the ones with the largest amplitude. This means that we want:

$$\epsilon = \frac{2^{2^{n-1}}}{2^{2^n} - 1} \approx \frac{1}{2^{2^{n-1}}} \tag{4.48}$$

From Lemma 4.13 it follows:

$$s = \frac{1}{\epsilon^2} = 2^{2^n} \tag{4.49}$$

■

While this sampling complexity can be considered as very large, the procedure at hand is akin to trying to reconstruct a quantum state from measurement, also known as quantum tomography. In which case, the sampling complexity is also in $O(\frac{1}{\epsilon^2})$ [35].

Now that P_1 has been estimated accurately enough, it remains to identify the misclassified inputs using this estimation. This can be done following Algorithm 2. As we go through the s measurements, the misclassified inputs are collected via the measurement on the first n qubits. The estimation of P_1 is finally computed and used to reconstruct another set of misclassified

inputs that are then added to the one obtained from collecting the measurements. This two-fold method allows for a more complete error identification than solely relying on the estimation of P_1. To reconstruct the set of misclassified inputs that contributed to the estimation of P_1 we use the following:

Proposition 4.15:

Let $n \in \mathbb{N}$ and $\sigma : \mathbb{B}^n \to [\![0, 2^n - 1]\!]$ as in Definition .Suppose that we obtained an estimation $\tilde{P_1}$ of the probability P_1. Then there exists a unique set $E_2 \subseteq \mathbb{B}^n$ such that:

$$\sum_{x \in E_2} 2^{2^n - 1 - \sigma(x)} = \lfloor (2^{2^n} - 1)\tilde{P_1} \rfloor \tag{4.50}$$

Proof:

Let $n \in \mathbb{N}$. Because $0 \leq \tilde{P_1} \leq 1$, we have that:

$$\lfloor (2^{2^n} - 1)\tilde{P_1} \rfloor \in [\![0, 2^{2^n} - 1]\!] \tag{4.51}$$

Because of the uniqueness of the binary representation, we are assured that there exists a unique set $S_p \subseteq [\![0, 2^n - 1]\!]$ such that:

$$\sum_{p \in S_p} 2^{2^n - 1 - p} = \lfloor (2^{2^n} - 1)\tilde{P_1} \rfloor \tag{4.52}$$

Now because σ is a bijection between $\mathbb{B}^n$ and $[\![0, 2^n]\!]$, there exists a unique subset $E_2 \subseteq \mathbb{B}^n$ such that:

$$E_2 = \sigma^{-1}(S_p) \tag{4.53}$$

And we have:

$$\sum_{x \in E_2} 2^{2^n - 1 - \sigma(x)} = \lfloor (2^{2^n} - 1)\tilde{P_1} \rfloor \tag{4.54}$$

■

Algorithm 2: Error identification

Input: M the list of the s measurements
Output: E the list of misclassified inputs
$N \leftarrow 0$;
$E_1 \leftarrow []$;
for $m \in M$ **do**
  if *the last bit of* m *is 1* **then**
    N += 1;
    append the first n bits of m to E_1;
  end
end
$P_1 \leftarrow \frac{N}{s}$;
if *working with* $|\psi_\downarrow\rangle$ **then**
  Find E_2 such that $\sum_{x \in E_2} 2^{2^n - 1 - \sigma(x)} = \lfloor (2^{2^n} - 1) P_1 \rfloor$
end
else
  Find E_2 such that $\sum_{x \in E_2} 2^{\sigma(x)} = \lfloor (2^{2^n} - 1) P_1 \rfloor$
end
$E \leftarrow E_1 \cup E_2$

4.3.3 Implementation

Here we will present a way of implementing the oracle on a quantum computer. Let $n \in \mathbb{N}$ and $f \in \mathbb{B}^{\mathbb{B}^n}$ be the target function. Following the requirements of a quantum system, we will consider the oracles $\mathbf{O}_\downarrow(f)$ and $\mathbf{O}_\uparrow(f)$ to be unitary matrices, elements of $U(2^{n+1})$, such that:

$$\mathbf{O}_\downarrow(f)\,|0\rangle = |\psi_\downarrow\rangle \text{ and } \mathbf{O}_\uparrow(f)\,|0\rangle = |\psi_\uparrow\rangle \tag{4.55}$$

In the following we will focus on $\mathbf{O}_\downarrow(f)$ but the construction of $\mathbf{O}_\uparrow(f)$ is similar. The oracle has been implemented in three stages, as shown in Figure 4.2, where the unitary gate $\mathbf{U}_\downarrow \in U(2^n)$ generates the amplitudes, $\mathbf{U}_\mu \in U(2^n)$ operates a permutation over the states and $\mathbf{U}_f \in U(2^{n+1})$ entangles the read-out qubit to the input register according to the target function f.

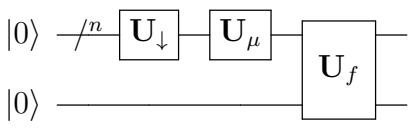


Figure 4.2: Construction of $\mathbf{O}_\downarrow(f)$

We recall that $\mathbf{O}_{\downarrow}(f)$ is such that:

$$\mathbf{O}_{\downarrow}(f)\,|0\rangle = |\psi_{\downarrow}\rangle = \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x\in\mathbb{B}^n} \sqrt{2^{2^n-1-\sigma(x)}}\,|x\rangle\,|f(x)\rangle \tag{4.56}$$

And we show a way to construct the aforementioned unitary gates so that the initial state $|0\rangle$ is transformed into $|\psi_{\downarrow}\rangle$.

Generation of the amplitudes

The first stage in implementing the oracle consists in generating the right amplitudes. To do so, we use the rotation operator $\mathbf{R}_y(\theta)$ defined by:

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \tag{4.57}$$

We define the following angles:

Definition 4.16:

Let $k \in \mathbb{N}$, we denote $\theta_k \in [0, \pi/2]$ with:

$$\theta_k = \arccos\left(\sqrt{\frac{2^{2^k}}{2^{2^k}+1}}\right) \tag{4.58}$$

Then

$$\mathbf{R}_y(2\theta_k) = \frac{1}{\sqrt{2^{2^k}+1}} \begin{bmatrix} \sqrt{2^{2^k}} & -1 \\ 1 & \sqrt{2^{2^k}} \end{bmatrix} \tag{4.59}$$

We can now show the following:

Proposition 4.17:

Let $n \in \mathbb{N}$, we define $\mathbf{U}_{\downarrow} \in U(2^n)$ by:

$$\mathbf{U}_{\downarrow} = \mathbf{R}_y(2\theta_{n-1}) \otimes \mathbf{R}_y(2\theta_{n-2}) \otimes \ldots \otimes \mathbf{R}_y(2\theta_0) \tag{4.60}$$

Where for $k \in [0, \dots, n-1]$, $\mathbf{R}_y(2\theta_k)$ is as in Definition 4.16. Then:

$$\mathbf{U}_\downarrow |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\bar{x}}} |x\rangle \tag{4.61}$$

Proof:

We show this by induction on $n \in \mathbb{N}^*$.

For $n = 1$:

$$\mathbf{U}_\downarrow = \mathbf{R}_y(2\theta_0) = \frac{1}{\sqrt{3}} \begin{bmatrix} \sqrt{2} & -1 \\ 1 & \sqrt{2} \end{bmatrix} \in U(2) \tag{4.62}$$

Hence:

$$\mathbf{U}_\downarrow |0\rangle = \frac{1}{\sqrt{3}} \left(\sqrt{2} |0\rangle + |1\rangle \right) \tag{4.63}$$

Now for $n > 1$, let us denote $\mathbf{U} = \mathbf{R}_y(2\theta_{n-1}) \otimes \mathbf{R}_y(2\theta_{n-2}) \otimes \ldots \otimes \mathbf{R}_y(2\theta_0) \in U(2^n)$ and suppose that:

$$\mathbf{U} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\bar{x}}} |x\rangle \tag{4.64}$$

Then by looking at the columns of $\mathbf{U} = [\mathbf{u}_1 | \ldots | \mathbf{u}_{2^n}]$, we have:

$$\mathbf{u}_1 = \frac{1}{\sqrt{2^{2^n}-1}} \begin{bmatrix} \sqrt{2^{2^n-1}} \\ \vdots \\ 1 \end{bmatrix} \tag{4.65}$$

We now denote $\mathbf{U}_\downarrow = \mathbf{R}_y(2\theta_n) \otimes \mathbf{U} \in U(2^{n+1})$, then the first column of $\mathbf{U}_\downarrow$ is:

$$\frac{1}{\sqrt{2^{2^n}+1}} \begin{bmatrix} \sqrt{2^{2^n}} \mathbf{u}_1 \\ \mathbf{u}_1 \end{bmatrix} = \frac{1}{\sqrt{(2^{2^n}+1)(2^{2^n}-1)}} \begin{bmatrix} \sqrt{2^{2^n} 2^{2^n-1}} \\ \vdots \\ \sqrt{2^{2^n}} \\ \sqrt{2^{2^n-1}} \\ \vdots \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2^{2^{n+1}}-1}} \begin{bmatrix} \sqrt{2^{2^{n+1}-1}} \\ \vdots \\ 1 \end{bmatrix} \tag{4.66}$$

Thus we do have:

$$\mathbf{U}_{\downarrow} |0\rangle^{\otimes n+1} = \frac{1}{\sqrt{2^{2^{n+1}}-1}} \sum_{x \in \mathbb{B}^{n+1}} \sqrt{2^{2^{n+1}-1-\bar{x}}} |x\rangle \tag{4.67}$$

■

So we have found a way of creating the amplitudes that we are interested in, we now have to group the inputs by Hamming weight so that the ones with the smallest weight have the largest amplitudes in the superposition. This can be achieved by permuting the states within the superposition obtained after the step that has just been described.

Permutation over the states

Let $n \in \mathbb{N}$, now that the right amplitudes have been generated, we have to find a unitary $\mathbf{U}_{\mu}$ such that:

$$\mathbf{U}_{\mu}\mathbf{U}_{\downarrow} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\sigma(x)}} |x\rangle \tag{4.68}$$

Where $\sigma : \mathbb{B}^n \to \mathbb{N}$ has been defined in Definition 4.11. Or equivalently:

$$\mathbf{U}_{\mu}\mathbf{U}_{\downarrow} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\bar{x}}} \left|\sigma^{-1}(\bar{x})\right\rangle \tag{4.69}$$

We thus see that to reach the desired superposition, a permutation over the states has to be performed, specifically:

Definition 4.18:

Let $n \in \mathbb{N}$. $S_{\mathbb{B}^n}$ is the symmetric group on $\mathbb{B}^n$ (the group of the permutations on $\mathbb{B}^n$) and $\sigma : \mathbb{B}^n \to [\![0, 2^n - 1]\!]$ as defined in Definition 4.11. We introduce $\mu \in S_{\mathbb{B}^n}$ the permutation defined by:

$$\mu(x) = \sigma^{-1}(\bar{x}) \tag{4.70}$$

Example 4.19:

The values of $\mu(x)$ for $x \in \mathbb{B}^4$ are given in Table 4.4, with the values of σ taken from Table 4.3.

x	$\mu(x)$	x	$\mu(x)$
0000	0000	1000	1001
0001	0001	1001	1010
0010	0010	1010	1100
0011	0100	1011	0111
0100	1000	1100	1011
0101	0011	1101	1101
0110	0101	1110	1110
0111	0110	1111	1111

Table 4.4: Values of $\mu(x)$ for $x \in \mathbb{B}^4$

In order to construct $\mathbf{U}_\mu$ we will use this well-known result [21]:

Proposition 4.20:

Let X be a finite set and S_X the symmetric group on X. For $x, y \in X$, we denote (x, y) the permutation such that:

$$\begin{cases} (x,y)(x) = y \\ (x,y)(y) = x \\ (x,y)(z) = z \text{ if } z \notin \{x, y\} \end{cases} \tag{4.71}$$

Then (x, y) is called a transposition and S_X is generated by the transpositions.

Property 4.20 means that any permutation can be decomposed as a product of transpositions and there exists an algorithm yielding this decomposition as shown in Algorithm 3. In this algorithm, the notion of minimum is used as a way to choose one non-fixed point so it can be taken with respect to any order relation on the set X.

Algorithm 3: Permutation decomposition into a product of transposition

Input: υ a permutation on X
Output: Π the decomposition in product of transposition
$\Pi \leftarrow Id$;
while $\upsilon \neq Id$ **do**
  $x_0 \leftarrow \min_{x \in X}\{\upsilon(x) \neq x\}$;
  $\Pi \leftarrow \Pi \circ (x_0, \upsilon(x_0))$;
  $\upsilon \leftarrow (x_0, \upsilon(x_0)) \circ \upsilon$
end

Example 4.21:

Applying Algorithm 3 to the permutation $\mu \in S_{B^4}$ as introduced in Example 4.19 results in:

$$\mu = (3,4)(4,8)(5,8)(6,8)(7,8)(8,9)(9,10)(10,12)(11,12) \tag{4.72}$$

Where the used order relation on $\mathbb{B}^4$ is the one derived from $\mathbb{N}$. For the sake of presentation $\bar{x} \in \mathbb{N}$ has been used instead of $x \in \mathbb{B}^4$ in Equation 4.72.

Now it happens that a transposition can easily be translated into a unitary gate using only multi-controlled **X** gates:

Proposition 4.22:

Let $n \in \mathbb{N}$ and $x, y \in \mathbb{B}^n$. We denote τ the transposition (x, y), then there exists a unitary gate $\mathbf{U}_\tau \in U(2^n)$ made of multi-controlled **X** gates such that:

$$\begin{cases} \mathbf{U}_\tau |x\rangle = |y\rangle \\ \mathbf{U}_\tau |y\rangle = |x\rangle \\ \mathbf{U}_\tau |z\rangle = |z\rangle \text{ if } z \notin \{x, y\} \end{cases} \tag{4.73}$$

Proof:

Let $n \in \mathbb{N}$, $x, y \in \mathbb{B}^n$ and $\tau = (x, y)$ the transposition. We can build $\mathbf{U}_\tau$ by using a Gray code [57] as follow:

1. Build a chain $c_0, \ldots, c_t$ with $c_0 = x$ and $c_t = y$ where for $k \in [\![0, t-1]\!]$, c_k and c_{k+1} differ by exactly one bit.

2. The transition from c_k to c_{k+1} is represented by a X gate acting on the bit that is differently controlled by the state of the other bits that are similar.

3. Once y is reached, complete with the symmetric of the obtained circuit with regard to the last gate.

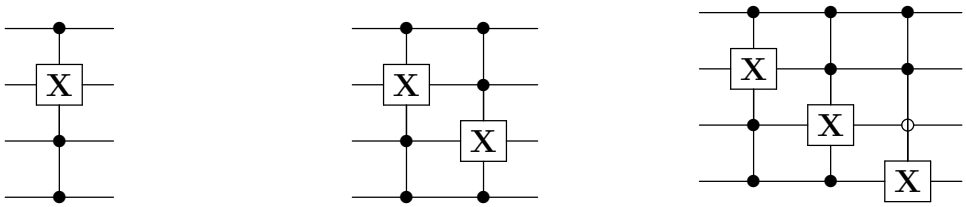
■

Example 4.23:

Suppose we want to construct the gate performing the transposition $\tau = (1011, 1100)$. One Gray code is:

$$1011 \to 1111 \to 1101 \to 1100 \tag{4.74}$$

The second step to build the gate performing $(1011, 1100)$ is shown in Figure 4.3 where Figure 4.3a represents the transition $1011 \to 1111$, in Figure 4.3b, the second transition $(1111 \to 1101)$ is added to the first and in Figure 4.3c, the last transition $(1101 \to 1100)$ is added. Finally, the circuit is completed with symmetry to result in $\mathbf{U}_\tau$ as shown in Figure 4.4.



(a) First transition (b) Second transition (c) Final transition

Figure 4.3: Second step to build the gate performing τ

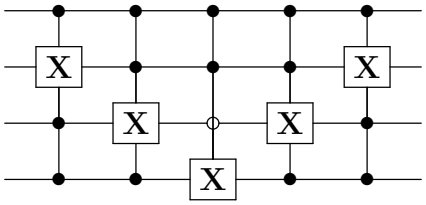


Figure 4.4: $\mathbf{U}_\tau$, the gate performing the transposition τ

We now show:

Proposition 4.24:

Let $n \in \mathbb{N}$ and $\upsilon \in S_{\mathbb{B}^n}$ a permutation on $\mathbb{B}^n$, then there exists a unitary gate $\mathbf{U}_\upsilon \in U(2^n)$ such that:

$$\forall x \in \mathbb{B}^n, \mathbf{U}_\upsilon |x\rangle = |\upsilon(x)\rangle \tag{4.75}$$

Proof:

Let $n \in \mathbb{N}$ and $\upsilon \in S_{\mathbb{B}^n}$. From Property 4.20, there exist $\tau_1, \ldots, \tau_k \in S_{\mathbb{B}^n}$ transpositions such that:

$$\upsilon = \tau_k \ldots \tau_1 \tag{4.76}$$

Now according to Property 4.22, for $i \in [\![1, k]\!]$, τ_i can be represented by a gate $\mathbf{U}_{\tau_i} \in U(2^n)$, then by denoting:

$$\mathbf{U}_\upsilon = \mathbf{U}_{\tau_k} \ldots \mathbf{U}_{\tau_1} \tag{4.77}$$

We are assured to have for $x \in \mathbb{B}^n$:

$$\mathbf{U}_\upsilon |x\rangle = |\upsilon(x)\rangle \tag{4.78}$$

■

Thanks to these result, we can finally build $\mathbf{U}_\mu$ where μ is defined in Definition 4.18. Therefore, for $n \in \mathbb{N}$, we have:

$$\mathbf{U}_\mu \mathbf{U}_\downarrow |0\rangle^{\otimes n} = \mathbf{U}_\mu \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\bar{x}}} |x\rangle \tag{4.79}$$

$$= \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\bar{x}}} |\mu(x)\rangle \tag{4.80}$$

$$= \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\bar{x}}} \left|\sigma^{-1}(\bar{x})\right\rangle \tag{4.81}$$

$$= \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\sigma(x)}} |x\rangle \tag{4.82}$$

Which is what we were looking for. All that is left is to entangle the read-out qubit with the input register according to the target function.

Entangling the read-out qubit

Let $n \in \mathbb{N}$, we now need a unitary gate $\mathbf{U}_f \in U(2^{n+1})$ such that:

$$\mathbf{U}_f \mathbf{U}_\mu \mathbf{U}_\downarrow |0\rangle^{\otimes n} |0\rangle = \frac{1}{\sqrt{2^{2^n}-1}} \sum_{x \in \mathbb{B}^n} \sqrt{2^{2^n-1-\sigma(x)}} |x\rangle |f(x)\rangle \tag{4.83}$$

This can be achieved with a gate $\mathbf{U}_f$ such that for $x \in \mathbb{B}^n$:

$$\mathbf{U}_f |x\rangle |0\rangle = |x\rangle |f(x)\rangle \tag{4.84}$$

While this may seem redundant as well as defeating the purpose of this work, we recall that we are presenting a way to implement the oracle $\mathbf{O}_\downarrow(f)$ on the quantum computer so that it transforms the state $|0\rangle$ into $|\psi_\downarrow\rangle$. Once the construction is finished, the whole can be abstracted into a black box producing the superposition $|\psi_\downarrow\rangle$.

There are many ways to construct this unitary, depending on the programming language used. For example, in Qiskit there exists a method[1] that takes as argument the truth table of a Boolean function and returns a gate performing as in (4.84). For this reason, we will not delve further into this matter.

For $n \in \mathbb{N}$ and $f \in \mathbb{N}$, we have thus shown a way to implement on a quantum computer the oracle $\mathbf{O}_\downarrow(f)$. To implement $\mathbf{O}_\uparrow(f)$ it suffices to swap the arccos in (4.58) with arcsin.

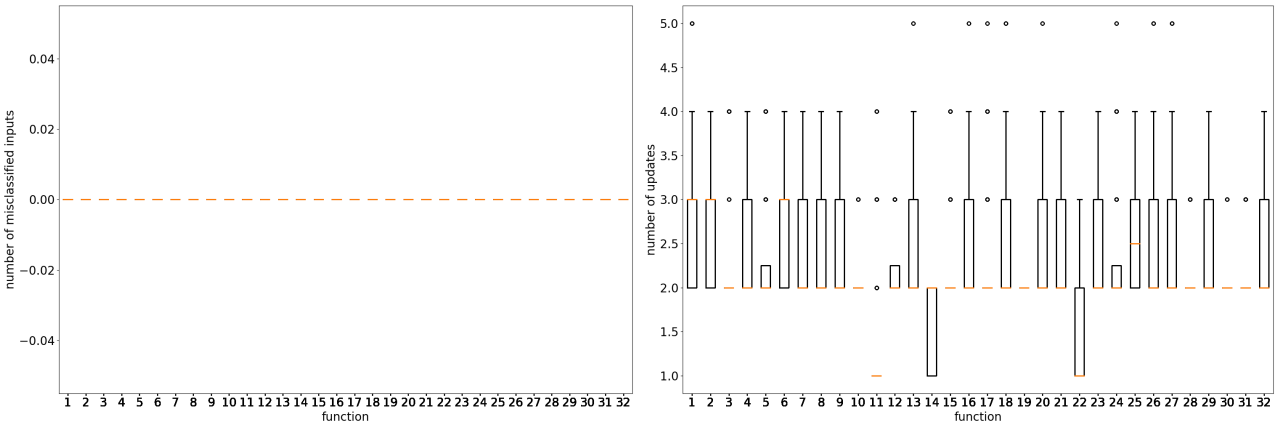
4.3.4 Experimental Results

We have implemented this algorithm and tested it for $n = 3$ and $n = 4$. The reason we did not test for larger dimensions is that for $n \geq 5$, the number of samples exceeded the simulator's maximum sample number.

Both for $n = 3$ and $n = 4$, we have trained the network over 32 target functions. To account for the randomness of the process, each training has been repeated 100 times. The statistics

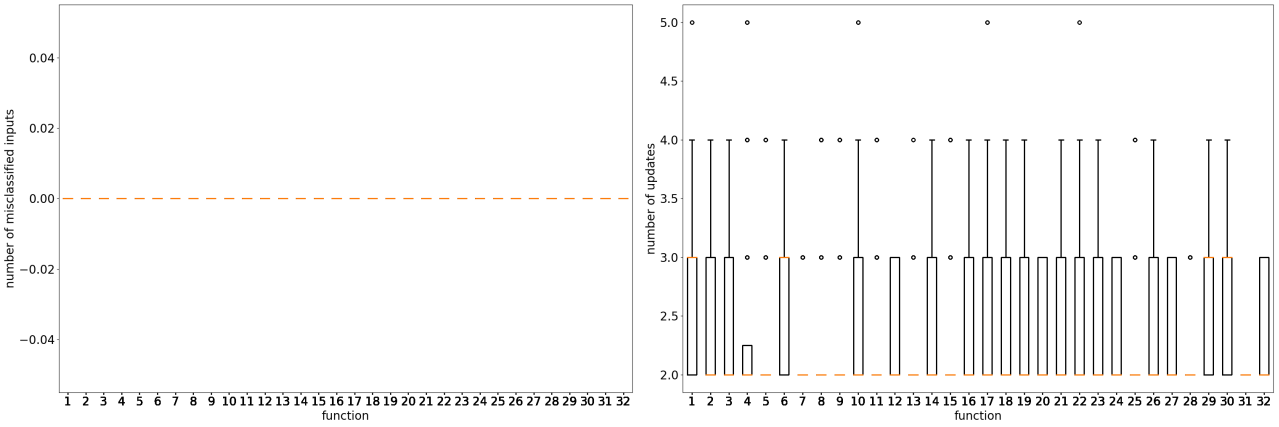
[1] qiskit.org/documentation/stable/0.26/stubs/qiskit.aqua.components.oracles.TruthTableOracle.html

obtained from these experiments are represented in Figures 4.5 and 4.6.



(a) Final number of misclassified inputs (b) Number of necessary updates

Figure 4.5: Results of the training runs for $n = 3$



(a) Final number of misclassified inputs (b) Number of necessary updates

Figure 4.6: Results of the training runs for $n = 4$

To represent the statistics, we have used box plots where the red line represents the median, and the lower and upper limits of the box represent the first and third quartiles respectfully. In addition, the circles indicate outliers. Looking at the final number of misclassified inputs for $n = 3$ and $n = 4$ (Figures 4.5a and 4.6a) we can see that all of the training runs resulted in a correctly tuned network, expressing exactly the target function. Regarding the number of updates, we also notice that for $n = 4$ (Figure 4.6b) all of the learning algorithms never took more than $n + 1 = 5$ updates of the network. For $n = 3$, some of the training runs took more than $n + 1 = 4$ update steps but these can be regarded as outliers. These punctual behaviours can be explained by the fact that whatever the oracle used, $\mathbf{O}_{\downarrow}$ or $\mathbf{O}_{\uparrow}$, the inputs

with a Hamming weight of approximately $n/2$ will have a small assigned amplitude. This means that they might not always be detected, hence it might take more than one update to accurately correct them. Despite these outliers, the algorithm performed as expected, resulting in a correctly tuned network, with the training process taking no more than $n+1$ update steps (for the vast majority).

4.4 Conclusion

In this chapter, we have devised an algorithm to train a tunable neural network so that it learns to express a target Boolean function. By using a quantum oracle providing a quantum superposition of all the inputs associated with a specific amplitude, we show that the training stage requires at most $n+1$ updates to the network to successfully learn the target function. However, this comes at the cost of a significant sampling complexity, in the order of 2^{2^n}. We implemented this algorithm and tested it for small dimensions. The experiments show that this approach works but the high sample complexity proved to be extremely limiting. For that reason, we turned to the field of learning theory to provide a guide in the design of an efficient training algorithm for the tunable quantum neural network.

Chapter 5

TNN in the QPAC-Learning Framework

5.1 Introduction

Introduced by [76], probably approximately correct (PAC) learning provides a mathematical framework to analyse classical machine learning techniques. This framework revolves around the existence of an oracle that provides samples drawn from the instance space. These samples are then used as examples for an algorithm to learn a target concept. The efficiency of this learning algorithm can then be characterised by its sampling complexity. Given the momentum gained by quantum techniques for machine learning, it seems natural for a quantum equivalence of PAC-learning to be introduced. This is exactly what has been done in [16] with QPAC-learning. In this framework, instead of providing samples from the instance space, the oracle generates a superposition of all the examples. Similarly to PAC-learning, QPAC-learning can be used to evaluate the learning efficiency of quantum algorithms.

In this chapter we are using this framework to study a learning algorithm based on quantum amplitude amplification [15]. We invoke this fundamental quantum procedure both to compare the error rate to a threshold as well as increase the probability of measuring the errors produced by the learner. These measured errors are then used to tune the learner in order to decrease

the error rate. In our case, this learner is a tunable quantum neural network as introduced in Chapter 3. We implemented this approach and used it to learn the concept class of parity functions. The results of these experiments show that this architecture is an efficient learner for this class with a sample complexity that is lower in some cases than what can be found in the literature.

5.2 Quantum Probably Approximately Correct

QPAC-Learning has been introduced in [16] and is the quantum version of the work presented in [76]. This framework allows for a mathematical analysis of learners by quantifying the number of samples that are needed to learn.

Definition 5.1:

Let $n \in \mathbb{N}$, a concept is simply a Boolean function $c \in \mathbb{B}^{\mathbb{B}^n}$ and we call concept class $\mathcal{C}$, a subset:

$$\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n} \tag{5.1}$$

During the learning process, the learner has access to an example oracle defined as follows:

Definition 5.2:

Let $n \in \mathbb{N}$, $c \in \mathbb{B}^{\mathbb{B}^n}$ a concept and $D : \mathbb{B}^n \to [0, 1]$ a probability distribution over $\mathbb{B}^n$. Then the example oracle $\mathbf{EX}(c, D)$ will output the superposition $|\Psi(c, D)\rangle$ such that:

$$|\Psi(c, D)\rangle = \sum_{x \in \mathbb{B}^n} \sqrt{D(x)}\, |x\rangle\, |c(x)\rangle \tag{5.2}$$

Herein lies the difference between the quantum and the classical version of PAC-Learning as in the classical version, a call to the oracle will return an example $(x, c(x))$ where x is drawn from $\mathbb{B}^n$ according to D. The following definitions are similar to the classical case:

Definition 5.3:

Let $n \in \mathbb{N}$, $c \in \mathbb{B}^{\mathbb{B}^n}$ and $D : \mathbb{B}^n \to [0, 1]$ as defined in Definition 5.2. We denote by $P_{x \sim D}$ the

sampling probability from the distribution D. Suppose that the learner outputs an hypothesis $h \in \mathbb{B}^{\mathbb{B}^n}$, then the error $err_D(c, h)$ is:

$$err_D(c, h) = P_{x \sim D}(h(x) \neq c(x)) \tag{5.3}$$

The aim for the learner is then to output a hypothesis h such that the error is small enough (approximately correct) and to do so with high probability (probably correct). This allows us to properly define what a learner is in the QPAC-Learning framework:

Definition 5.4:
Let $n \in \mathbb{N}$, $\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n}$ be a concept class, $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$. Suppose that for every distribution D over $\mathbb{B}^n$ and every target concept $c \in \mathcal{C}$, and given access to the oracle $\mathbf{EX}(c, D)$, an algorithm outputs and hypothesis $h \in \mathbb{B}^{\mathbb{B}^n}$ such that:

$$P(err_D(c, h) < \epsilon) > 1 - \delta \tag{5.4}$$

Then the algorithm is said to be an (ϵ, δ)-learner for $\mathcal{C}$.

Note that in Definition 5.4, the hypothesis is taken is $\mathbb{B}^{\mathbb{B}^n}$ but it is also possible to require $h \in \mathcal{C}$ which leads to two types of learners:

Definition 5.5:
Let $\mathcal{C}$ be a concept class. Suppose the learner outputs h:

- If we are assured that $h \in \mathcal{C}$, then the learner is said to be a proper learner
- Otherwise it is an improper learner

As stated previously, this framework allows for a mathematical analysis of learning algorithms by studying their sample complexity:

Definition 5.6:
Let $\mathcal{C}$ a concept class and L an (ϵ, δ)-learner for $\mathcal{C}$. The sample complexity of L is the maximum

number of calls to the example oracle $\mathbf{EX}(c, D)$ needed to output a hypothesis h verifying the conditions of (5.4), taken over all target concepts $c \in \mathcal{C}$ and distributions D.

This notion of sample complexity then leads to the idea of an efficient learner:

Definition 5.7:

Let $\mathcal{C}$ a concept class and L an (ϵ, δ)-learner for $\mathcal{C}$. L is said to be an efficient learner if its sample complexity is polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$

5.3 TNN in the QPAC-Learning Framework

It turns out that tunable networks are particularly well suited to be studied in the QPAC-learning framework. Let $n \in \mathbb{N}$, $\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n}$ a concept class and $c \in \mathcal{C}$. Let D be a probability distribution over $\mathbb{B}^n$ and $\mathbf{EX}(c, D)$ the oracle producing $|\Psi(c, D)\rangle$ as in (5.2). Now let $\mathbf{T}(h)$ a tunable network that is expressing $h \in \mathbb{B}^{\mathbb{B}^n}$ in its current state, then we have:

$$\mathbf{T}(h)\,|\Psi(c, D)\rangle = \sum_{x \in \mathbb{B}^n} \sqrt{D(x)}\,|x\rangle\,|h(x) \oplus c(x)\rangle \tag{5.5}$$

$$= \sum_{h(x)=c(x)} \sqrt{D(x)}\,|x\rangle\,|0\rangle + \sum_{h(x)\neq c(x)} \sqrt{D(x)}\,|x\rangle\,|1\rangle \tag{5.6}$$

Definition 5.8:

Let $\theta_{err} \in \left[0, \frac{\pi}{2}\right]$ such that:

$$\sin(\theta_{err}) = \sqrt{\sum_{h(x)\neq c(x)} D(x)} \tag{5.7}$$

Then we have:

$$\mathbf{T}(h)\,|\Psi(c, D)\rangle = \sin(\theta_{err})\,|\phi_{err}\rangle\,|1\rangle + \cos(\theta_{err})\,|\phi_{\not{err}}\rangle\,|0\rangle \tag{5.8}$$

Where $|\phi_{\not{err}}\rangle$ and $|\phi_{err}\rangle$ are the normalised states proportional to $\sum_{h(x)=c(x)} \sqrt{D(x)}\,|x\rangle$ and $\sum_{h(x)\neq c(x)} \sqrt{D(x)}\,|x\rangle$ respectively. Then by denoting P_1 the probability of measuring the read-

out qubit in state $|1\rangle$, we have:

$$P_1 = \sin^2(\theta_{err}) = \sum_{h(x)\neq c(x)} D(x) = err_D(c,h) \tag{5.9}$$

To use tunable neural networks in the QPAC-learning framework we thus need a mean to estimate $\sin^2(\theta_e)$ and compare it to $0 < \epsilon < 1/2$. If the error is smaller than ϵ, output the current hypothesis. Otherwise, tune the network so that it expresses another hypothesis h' and repeat. The proposed algorithm does just that through the use of amplitude amplification.

5.4 Tuning Algorithm

5.4.1 Preliminary Results

At its core, the algorithm aims at amplifying the probability of measuring an erroneous input in order to increase the chance of correcting it. By doing so, it turns out that it is also possible to compare this probability to a given value ϵ.

Definition 5.9:
Let $0 < \epsilon < \frac{1}{2}$, then we denote $\theta_\epsilon \in \left]0, \frac{\pi}{4}\right[$ such that:

$$\sin^2(\theta_\epsilon) = \epsilon \tag{5.10}$$

Additionally, we define $m_{\max} \in \mathbb{N}$ by:

$$m_{\max} = \arg\min_{m\in\mathbb{N}} (2m+1)\theta_\epsilon \geq \frac{\pi}{4} \tag{5.11}$$

Because sin is increasing on $\left[0, \frac{\pi}{2}\right]$, comparing $err_D(c,h)$ to ϵ is equivalent to comparing θ_{err} to θ_ϵ. To do so, we introduce the following:

Lemma 5.10:

Let $\theta_\epsilon \in \left]0, \frac{\pi}{4}\right[$ and $m_{\max} \in \mathbb{N}$ as in Definition 5.9 and let $\theta \in \left[0, \frac{\pi}{2}\right]$.

If for all $m \leq m_{\max}$, we have $\sin^2\left((2m+1)\theta\right) < \frac{1}{2}$ then $\theta < \theta_\epsilon$

Proof:

We show this by contraposition.

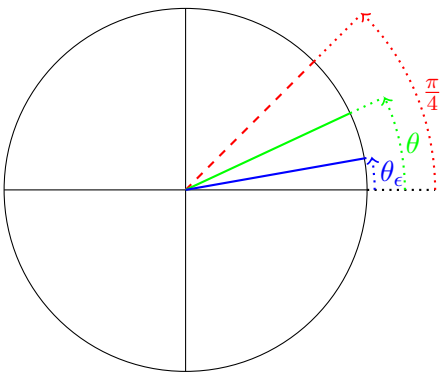
Let $\theta \geq \theta_\epsilon$.

Suppose $\theta \in \left[\frac{\pi}{4}, \frac{\pi}{2}\right]$, then taking $m = 0 \leq m_{\max}$, we have $\sin^2(\theta) \geq \frac{1}{2}$.

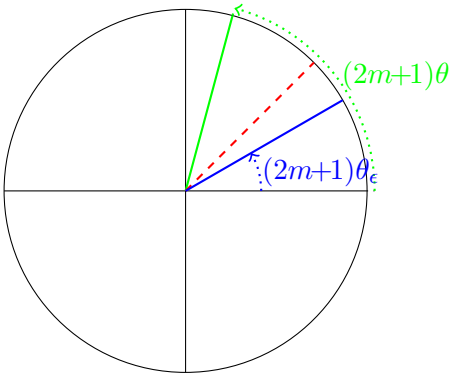
Now suppose that $\theta \in \left[0, \frac{\pi}{4}\right[$. By definition, we have $m_{\max} = \left\lceil \frac{1}{2}\left(\frac{\pi}{4\theta_\epsilon} - 1\right)\right\rceil$ and we denote $m = \left\lceil \frac{1}{2}\left(\frac{\pi}{4\theta} - 1\right)\right\rceil$, then $m \leq m_{\max}$. Let $0 \leq \alpha < 1$ such that $m = \frac{1}{2}\left(\frac{\pi}{4\theta} - 1\right) + \alpha$. Then we have $(2m+1)\theta = \frac{\pi}{4} + 2\alpha\theta$ and $\frac{\pi}{4} \leq (2m+1)\theta \leq \frac{3\pi}{4}$. This leads to $\sin^2((2m+1)\theta) \geq \frac{1}{2}$.

Hence Lemma 5.10. ■

This result is illustrated in Figures 5.1 and 5.2.

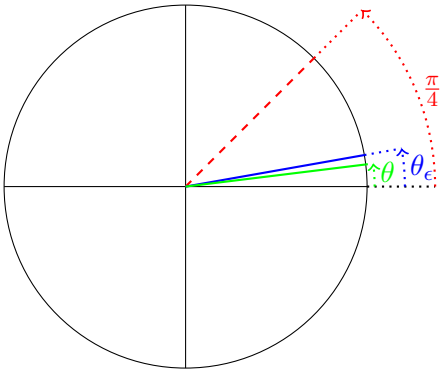


(a) Initial state.

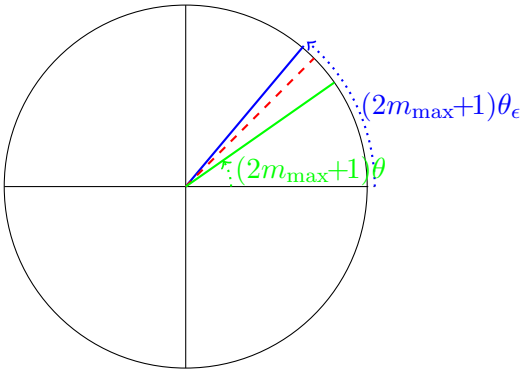


(b) After $m \leq m_{\max}$ steps of amplitude amplification.

Figure 5.1: Case where $\theta \geq \theta_\epsilon$.



(a) Initial state.



(b) After $m_{\max}$ steps of amplitude amplification.

Figure 5.2: Case where $\theta < \theta_\epsilon$.

5.4.2 Description of the Algorithm

The tuning algorithm works as follows. After each update of the tunable network, the error is compared to ϵ thanks to Lemma 5.10: after $m_0 < m_{\max}$ steps of amplification, the resulting state is measured N times and the number S of measurements for which the ancillary qubit is in state $|1\rangle$ is counted.
If $S > \frac{N}{2}$, then the error is greater than ϵ and the network is updated using the measurements on the first n qubits. If on the other hand $S \leq \frac{N}{2}$, the process is repeated with $m_0 + 1 \leq m_{\max}$ steps of amplification.
The algorithm stops when the network reaches a state such that even after $m_{\max}$ steps of amplification we have $S \leq \frac{N}{2}$. It is thus necessary to choose N such that when the algorithm stops the error is most probably lower than ϵ.

To avoid the angle $(2m_{\max}+1)\theta_\epsilon$ from overshooting $\frac{\pi}{4}$ by too much, we have chosen to limit ϵ to $\left]0, \frac{1}{10}\right[$. This limitation can be achieved without loss of generality by scaling the problem by a factor of $\frac{1}{5}$. This can be done by introducing a second ancillary qubit and applying a controlled rotation gate:

Definition 5.11:
We define **CR** the controlled rotation gate such that:

$$\mathbf{CR}\,|10\rangle = \frac{2}{\sqrt{5}}\,|10\rangle + \frac{1}{\sqrt{5}}\,|11\rangle \tag{5.12}$$

The states of interest are then the ones for which the two qubits ancillary register is in state $|11\rangle$. This procedure has two consequences: for $\epsilon \in \left]0, \frac{1}{2}\right[$, we are effectively working with $\epsilon' = \frac{\epsilon}{5} \in \left]0, \frac{1}{10}\right[$ as required. Additionally, the effective error is itself limited to $\left[0, \frac{1}{5}\right]$ meaning that we have $\theta_{err} \in \left[0, \arcsin\left(\frac{1}{\sqrt{5}}\right)\right]$. This means that $\theta_{err} \neq \frac{\pi}{4}$ which is a stationary point of the amplitude amplification procedure thus ensuring that the error will always be amplified.

Using the notations for quantum amplitude amplification introduced in Section 2.3.4, we can define the following operators:

Definition 5.12:

Let $n \in \mathbb{N}$, $h \in \mathbb{B}^{\mathbb{B}^n}$ and $\mathbf{T}(h)$ a tunable network that, in its current state, is expressing h, then:

$$\mathbf{U}(h) = (\mathbf{I}^{\otimes n} \otimes \mathbf{CR})(\mathbf{T}(h) \otimes \mathbf{I}) \tag{5.13}$$

Now for $c \in \mathbb{B}^{\mathbb{B}^n}$ and D a distribution over $\mathbb{B}^n$, we denote:

$$\mathcal{X}_{\alpha_0}(c, D) = \mathbf{I} - 2\,|\Psi(c, D)\rangle\,\langle\Psi(c, D)| \tag{5.14}$$

And finally:

$$\mathcal{X}_G = \mathbf{I}^{\otimes n} \otimes \mathbf{CZ} \tag{5.15}$$

Where **CZ** is the **Z** gate controlled by the first ancillary qubit and acting on the second ancillary qubit.

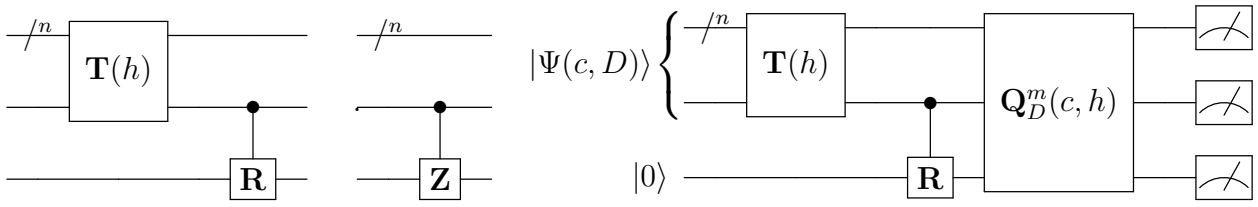
From these components, the diffusion operator $\mathbf{Q}_D(c, h)$ can be build:

Definition 5.13:

Let $n \in \mathbb{N}$, $c, h \in \mathbb{B}^{\mathbb{B}^n}$ and D a distribution over $\mathbb{B}^n$, then the diffusion operator is defined by:

$$\mathbf{Q}_D(c, h) = -\mathbf{U}(h)\mathcal{X}_{\alpha_0}(c, D)\mathbf{U}(h)^\dagger\mathcal{X}_G \tag{5.16}$$

The quantum circuits corresponding to $\mathcal{U}(h)$ and $\mathcal{X}_G$ are depicted respectively in Figure 5.3a and 5.3b while the circuit used to tune the network is shown in Figure 5.3c.



(a) $\mathbf{U}(h)$ (b) $\mathcal{X}_G$ (c) Circuit used to tune the network

Figure 5.3: Quantum circuits of some components of the diffusion operator (Figures 5.3a and 5.3b) and overall circuit used to tune the network (Figure 5.3c).

Finally, the tuning algorithm is shown in Algorithm 4. In Algorithm 4 the instruction "Update **TNN**" is given without further specification as the update strategy depends on the concept class being learnt. The update strategy used to learn the class of concepts introduced in Section 5.6 is specified in Algorithm 5. The requirement for the number of samples N will be justified in Section 5.5.1.

Algorithm 4: Tuning Algorithm for QPAC-learning

Data: $0 < \epsilon < \frac{1}{2}$, $0 < \delta < \frac{1}{2}$, $|\Psi(c, D)\rangle$
Result: **TNN** expressing h^* such that $P(err_D(h^*, c) < \epsilon) > 1 - \delta$
$N \leftarrow 2\left(\left\lfloor \frac{1}{\pi\delta^2} \right\rfloor //2\right) + 2$;
$m_{\max} \leftarrow$ smallest integer such that $(2m_{\max} + 1)\arcsin\left(\sqrt{\frac{\epsilon}{5}}\right) \geq \frac{\pi}{4}$;
$\mathbf{T} \leftarrow \mathbf{I}$;
$m \leftarrow -1$;
$s \leftarrow 0$;
$errors \leftarrow []$;
$corrects \leftarrow []$;
while $m < m_{max}$ ***or*** $s > \frac{N}{2}$ **do**
  $m \leftarrow m + 1$;
  $s \leftarrow 0$;
  $\mathbf{U} \leftarrow \left(\mathbf{I}^{\otimes n} \otimes \mathbf{R}\right)\left((\mathbf{T} \otimes \mathbf{I}\right)$;
  $|\Phi\rangle \leftarrow \mathbf{U}\,|\Psi(c, D)\rangle$;
  $\mathbf{Q} \leftarrow -\mathbf{U}\mathcal{X}_{\alpha_0}\mathbf{U}^{-1}\mathcal{X}_G$;
  for $1 \leq k \leq N$ **do**
    Measure $\mathbf{Q}^m\,|\Phi\rangle$;
    if *1 is measured on the first ancillary qubit* **then**
      append the string of the first n qubits to $errors$;
      if *1 is measured on the second ancillary qubit* **then**
        $s \leftarrow s + 1$;
      end
    end
    else
      append the string of the first n qubits to $corrects$;
    end
  end
  if $s > \frac{N}{2}$ **then**
    Update $\mathbf{T}$;
    $m \leftarrow -1$;
    $errors \leftarrow []$;
    $corrects \leftarrow []$;
  end
end

5.5 Proof and Analysis of the Algorithm

5.5.1 Proof for the Number of Samples

Let N be the number of measurements performed during a measurement round and S be the number of these measurements that return a state of interest. As stated in Algorithm 4, the tuning stops when even after $m_{\max}$ rounds of amplitude amplification we have:

$$S \leq \frac{N}{2} \tag{5.17}$$

This event will only occur if for all $m < m_{\max}$, we also have $S \leq \frac{N}{2}$ following m rounds of amplification.

Prior to the diffusion operator, the system is in the state $|\Phi\rangle$ such that:

$$|\Phi\rangle = \sin(\theta_{err}) |\phi_{err}\rangle |11\rangle + \cos(\theta_{err}) |\phi_{\perp}\rangle \tag{5.18}$$

Where $|\phi_{err}\rangle$ is the superposition of the states that are misclassified by the network and $\langle\phi_{\perp}|\phi_{err}, 11\rangle = 0$. After $m_{\max}$ rounds of amplitude amplification we have:

$$\mathbf{Q}^{m_{\max}} |\psi\rangle = \sin((2m_{\max}+1)\theta_{err}) |\phi_{err}\rangle |11\rangle + \cos((2m_{\max}+1)\theta_{err}) |\phi_{\perp}\rangle \tag{5.19}$$

Let us denote $p = \sin^2((2m_{\max}+1)\theta_{err})$, then p is the probability of measuring a state of interest, marked with the ancillary qubits in state $|11\rangle$, and we wish to compare p to $\frac{1}{2}$ in order to apply Lemma 5.10. This boils down to estimating $P\left(p < \frac{1}{2} | S < \frac{N}{2}\right)$. First, we need:

Lemma 5.14:

Let $N, k \in \mathbb{N}$ with $k \leq N$ and $a \in [0,1]$, then:

$$\int_0^a \theta^k (1-\theta)^{N-k} d\theta = \frac{k!(N-k)!}{(N+1)!} - (1-a)^{N-k+1} \sum_{i=0}^{k} \frac{k!(N-k)!}{(k-i)!(N-k+i+1)!} a^{k-i}(1-a)^i \tag{5.20}$$

Proof:

Let $N \in \mathbb{N}$ and $a \in [0,1]$ we show the result by induction over $0 \leq k \leq N$.

For $k = 0$:

$$\int_0^a (1-\theta)^N d\theta = \left[-\frac{1}{N+1}(1-\theta)^{N+1}\right]_0^a = \frac{1}{N+1} - \frac{1}{N+1}(1-a)^{N+1} \tag{5.21}$$

Suppose the result verified for $0 \leq k < N$, by integration by parts:

$$\int_0^a \theta^{k+1}(1-\theta)^{N-(k+1)} d\theta = \left[-\frac{\theta^{k+1}(1-\theta)^{N-k}}{N-k}\right]_0^a + \frac{k+1}{N-k}\int_0^a \theta^k (1-\theta)^{N-k} d\theta \tag{5.22}$$

$$= -\tfrac{a^{k+1}(1-a)^{N-k}}{N-k} + \tfrac{k+1}{N-k}\left[\tfrac{k!(N-k)!}{(N+1)!} - (1-a)^{N-k+1}\textstyle\sum_{i=0}^{k} \tfrac{k!(N-k)!}{(k-i)!(N-k+i+1)!} a^{k-i}(1-a)^i\right] \tag{5.23}$$

$$= \tfrac{(k+1)!(N-(k+1))!}{(N+1)!} - \left[\tfrac{a^{k+1}(1-a)^{N-k}}{N-k} + (1-a)^{N-k+1}\textstyle\sum_{i=0}^{k} \tfrac{(k+1)!(N-k-1)!}{(k-i)!(N-k+i+1)!} a^{k-i}(1-a)^i\right] \tag{5.24}$$

But:

$$\frac{a^{k+1}(1-a)^{N-k}}{N-k} + (1-a)^{N-k+1}\sum_{i=0}^{k} \frac{(k+1)!(N-k-1)!}{(k-i)!(N-k+i+1)!} a^{k-i}(1-a)^i \tag{5.25}$$

$$= (1-a)^{N-k}\left[\frac{a^{k+1}}{N-k} + \sum_{i=0}^{k} \frac{(k+1)!(N-k-1)!}{(k-i)!(N-k+i+1)!} a^{k-i}(1-a)^{i+1}\right] \tag{5.26}$$

$$= (1-a)^{N-(k+1)+1}\left[\tfrac{a^{k+1}}{N-k} + \textstyle\sum_{i=1}^{k+1} \tfrac{(k+1)!(N-(k+1))!}{((k+1)-i)!(N-(k+1)+i+1)!} a^{(k+1)-i}(1-a)^i\right] \tag{5.27}$$

$$= (1-a)^{N-(k+1)+1}\sum_{i=0}^{k+1} \frac{(k+1)!(N-(k+1))!}{((k+1)-i)!(N-(k+1)+i+1)!} a^{(k+1)-i}(1-a)^i \tag{5.28}$$

Combining (5.24) with (5.28), we get the result for $k+1$. ■

Lemma 5.14 will be used to show:

Proposition 5.15:

Let S be the number of states of interest that have been measured among the N measurements, then $S \sim B(N, p)$. When placing a uniform marginal distribution on p and assuming that N is even, we have:

$$P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) = \frac{1}{2} + \left(\frac{1}{2}\right)^{N+1} \frac{N+1}{N+2}\left(2^N - \binom{N}{N/2}\right) \tag{5.29}$$

Proof:

From Bayes Theorem:

$$P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) = \frac{P(S \leq N/2 \mid p < 1/2)P(p < 1/2)}{P(S \leq N/2)} \tag{5.30}$$

Placing a uniform distribution on p yields:

$$P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) = \frac{\sum_{k=0}^{N/2} \binom{N}{k} \int_0^{\frac{1}{2}} \theta^k (1-\theta)^{N-k} d\theta}{\sum_{k=0}^{N/2} \binom{N}{k} \int_0^1 \theta^k (1-\theta)^{N-k} d\theta} \tag{5.31}$$

But according to Lemma 5.14:

$$\int_0^1 \theta^k (1-\theta)^{N-k} d\theta = \frac{k!(N-k)!}{(N+1)!} \tag{5.32}$$

So:

$$\sum_{k=0}^{N/2} \binom{N}{k} \int_0^1 \theta^k (1-\theta)^{N-k} d\theta = \sum_{k=0}^{N/2} \binom{N}{k} \frac{k!(N-k)!}{(N+1)!} \tag{5.33}$$

$$= \frac{N+2}{2(N+1)} \tag{5.34}$$

Similarly, according to Lemma 5.14, we have:

$$\int_0^{\frac{1}{2}} \theta^k (1-\theta)^{N-k} d\theta = \frac{k!(N-k)!}{(N+1)!} - \left(\frac{1}{2}\right)^{N+1} \sum_{i=0}^{k} \frac{k!(N-k)!}{(k-i)!(N-k+i+1)!} \tag{5.35}$$

So:

$$\sum_{k=0}^{N/2} \binom{N}{k} \int_0^{\frac{1}{2}} \theta^k (1-\theta)^{N-k} d\theta = \frac{N+2}{2(N+1)} - \left(\frac{1}{2}\right)^{N+1} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \frac{N!}{(k-i)!(N-k+i+1)!} \tag{5.36}$$

Putting (5.30), (5.34) and (5.36) together yields:

$$P(p < 1/2 \mid S \leq N/2) = 1 - \left(\frac{1}{2}\right)^N \frac{N+1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \frac{N!}{(k-i)!(N-k+i+1)!} \tag{5.37}$$

$$= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \frac{(N+1)!}{(k-i)!(N-k+i+1)!} \tag{5.38}$$

$$= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \binom{N+1}{k-i} \tag{5.39}$$

$$= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \binom{N+1}{i} \tag{5.40}$$

$$= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \binom{N+1}{k} \left(\frac{N}{2} + 1 - k\right) \tag{5.41}$$

Now:

$$\sum_{k=0}^{N/2} \binom{N+1}{k} \left(\frac{N}{2} + 1 - k\right) = \frac{N+2}{2} \sum_{k=0}^{N/2} \binom{N+1}{k} - \sum_{k=0}^{N/2} \binom{N+1}{k} k \tag{5.42}$$

As N is even:

$$\sum_{k=0}^{N/2} \binom{N+1}{k} = 2^N \tag{5.43}$$

And for $k > 0$, we have:

$$\binom{N+1}{k} k = \binom{N}{k-1} (N+1) \tag{5.44}$$

Plugging (5.44) and (5.43) into (5.42) leads to:

$$\sum_{k=0}^{N/2} \binom{N+1}{k} \left(\frac{N}{2} + 1 - k\right) = 2^{N-1}(N+2) - (N+1) \sum_{k=0}^{\frac{N}{2}-1} \binom{N}{k} \tag{5.45}$$

Replacing (5.45) in (5.41), gives:

$$P(p < 1/2 \mid S \leq N/2) = 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \left(2^{N-1}(N+2) - (N+1) \sum_{k=0}^{\frac{N}{2}-1} \binom{N}{k}\right) \tag{5.46}$$

$$= \frac{1}{2} + \left(\frac{1}{2}\right)^N \frac{N+1}{N+2} \sum_{k=0}^{\frac{N}{2}-1} \binom{N}{k} \tag{5.47}$$

N being even:

$$\sum_{k=0}^{\frac{N}{2}-1}\binom{N}{k} = \sum_{k=0}^{\frac{N}{2}-1}\binom{N}{k} = \frac{1}{2}\left(2^N - \binom{N}{N/2}\right) \tag{5.48}$$

Finally, plugging (5.48) into (5.47), we have:

$$P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) = \frac{1}{2} + \left(\frac{1}{2}\right)^{N+1}\frac{N+1}{N+2}\left(2^N - \binom{N}{N/2}\right)$$

■

We can quickly check that this equality is sound by using Stirling's approximation:

$$\binom{n}{k} \sim \sqrt{\frac{n}{2\pi k(n-k)}}\frac{n^n}{k^k(n-k)^{n-k}} \tag{5.49}$$

Hence:

$$\binom{N}{N/2} \sim 2^N\sqrt{\frac{2}{\pi N}} \tag{5.50}$$

So:

$$\lim_{N\to\infty} P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) = 1 \tag{5.51}$$

Thanks to Property 5.15, we can finally determine the number of measurements needed to ensure that the tuning algorithm successfully stops with high probability.

Theorem 5.16:

Let $0 < \delta < \frac{1}{2}$, by taking:

$$N = 2\left(\left\lfloor\frac{1}{\pi\delta^2}\right\rfloor //2\right) + 2 \tag{5.52}$$

We have $P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) > 1 - \delta$.

Proof:

As seen in Property 5.15, we have:

$$P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) = \frac{1}{2} + \left(\frac{1}{2}\right)^{N+1}\frac{N+1}{N+2}\left(2^N - \binom{N}{N/2}\right) \tag{5.53}$$

But [20]:

$$\binom{N}{N/2} \leq \sqrt{2}\frac{2^N}{\sqrt{\pi N}} \tag{5.54}$$

So:

$$P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) \geq \frac{1}{2} + \frac{N+1}{2(N+2)}\left(1 - \frac{\sqrt{2}}{\sqrt{\pi N}}\right) \tag{5.55}$$

Because $\frac{N+1}{N+2} \sim 1$, we can look for a lower bound of the form $1 - \frac{\alpha}{\sqrt{N}}$. For example:

$$P\left(p < \frac{1}{2} \mid S < \frac{N}{2}\right) > 1 - \frac{1}{\sqrt{\pi N}} \tag{5.56}$$

Now take $0 < \delta < \frac{1}{2}$, we want N even such that:

$$\frac{1}{\sqrt{\pi N}} < \delta \Rightarrow N > \frac{1}{\pi\delta^2} \tag{5.57}$$

In particular, we can set:

$$N = 2\left(\left\lfloor \frac{1}{\pi\delta^2} \right\rfloor //2\right) + 2$$

■

We have shown that when the algorithm stops, we have:

$$P\left(\sin^2((2m_{\max}+1)\theta_{err}) < \frac{1}{2}\right) > 1 - \delta \tag{5.58}$$

This, with Lemma 5.10 leads to $P(\theta_{err} < \theta_\epsilon) > 1 - \delta$, hence:

$$P(err_D(c,h) < \epsilon) > 1 - \delta \tag{5.59}$$

5.5.2 Analysis of the Algorithm

Let $n \in \mathbb{N}$, $\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n}$ a concept class and $c \in \mathcal{C}$ the target concept. Moreover, let D be a distribution over $\mathbb{B}^n$ and $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$. In Subsection 5.5.1 we have shown that during a measurement phase, the algorithm will make $\Theta\left(\frac{1}{\delta^2}\right)$ calls to the example oracle $\mathbf{EX}(c, D)$. Now during an update step, there will be at most $m_{\max}$ of these measurement phases where $m_{\max}$ is, as in Definition 5.9, such that $(2m_{\max}+1)\theta_\epsilon \approx \frac{\pi}{4}$ with $\theta_\epsilon = \arcsin\left(\sqrt{\frac{\epsilon}{5}}\right)$. As $0 < \epsilon < \frac{1}{2}$, we have $\theta_\epsilon \approx \sqrt{\frac{\epsilon}{5}}$, hence $m_{\max} = \Theta\left(\frac{1}{\sqrt{\epsilon}}\right)$. In total, an update step will thus

make $\Theta\left(\frac{1}{\sqrt{\epsilon}}\frac{1}{\delta^2}\right)$ calls to $\mathbf{EX}(c, D)$. As we will see in Section 5.6, it is possible to reduce the dependence in $\frac{1}{\epsilon}$ from square root to logarithm. The number of steps will then depend on the concept class being learnt but whatever the concept class, the sample complexity of this algorithm is polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$, so the proposed algorithm is an efficient learner.

5.6 Learning Parity Functions

5.6.1 Description of the Concept Class

In this section, we will focus on learning the class of parity functions:

Definition 5.17:

Let $n \in \mathbb{N}$ and $s \in \mathbb{B}^n$, we define the parity function $p_s \in \mathbb{B}^{\mathbb{B}^n}$ by:

$$p_s : x \mapsto s.x = s_0.x_0 \oplus \ldots \oplus s_{n-1}.x_{n-1} \tag{5.60}$$

And the class of the parity functions $\mathcal{C}_p$ is:

$$\mathcal{C}_p = \{p_s \mid s \in \mathbb{B}^n\} \tag{5.61}$$

An important quantity that characterises a concept class is the Vapnik-Chervonenkis dimension (VC dimension) which is defined as follows:

Definition 5.18:

Let $n \in N$ and $\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n}$ be a concept class. Let $d \in \mathbb{N}$ and $(b_0, \ldots, b_{d-1})$ where $b_i \in \mathbb{B}^n$, for $0 \leq i \leq d-1$. Then the set $(b_0, \ldots, b_{d-1})$ is said to be shattered by $\mathcal{C}$ if:

$$\{(c(b_0), \ldots, c(b_{d-1})) \mid c \in \mathcal{C}\} = \mathbb{B}^d \tag{5.62}$$

The VC dimension of $\mathcal{C}$ is then the maximum d such that there exists a set $(b_0, \ldots, b_{d-1})$ that is shattered by $\mathcal{C}$.

The VC dimension of a concept class quantifies the expressivity of the class. We can then show the following:

Proposition 5.19:

Let $n \in \mathbb{N}$, the VC dimension of $\mathcal{C}_p = \{p_s \mid s \in \mathbb{B}^n\}$ is n.
Proof:

This is a rather known proof but out of completeness, we include it here.

Let $n \in \mathbb{N}$. Then $|\mathcal{C}_p| = |\mathbb{B}^n| = 2^n$.

Now let $d \in \mathbb{N}$ and $b_0, \ldots, b_{d-1} \in \mathbb{B}^n$, then:

$$|\{(c(b_0), \ldots, c(b_{d-1})) \mid c \in \mathcal{C}_p\}| \leq |\mathcal{C}_p| = 2^n \tag{5.63}$$

In particular, for $0 \leq i \leq n-1$, let us take b_i such that $1_{b_i} = \{i\}$. Then for $s \in \mathbb{B}^n$, we have:

$$(p_s(b_0), \ldots, p_s(b_{n-1})) = s \tag{5.64}$$

So:

$$\mathbb{B}^n \subseteq \{(c(b_0), \ldots, c(b_{n-1})) \mid c \in \mathcal{C}_p\} \tag{5.65}$$

But $|\mathbb{B}^n| = 2^n$ so together with (5.63), it follows that:

$$\{(c(b_0), \ldots, c(b_{n-1})) \mid c \in \mathcal{C}_p\} = \mathbb{B}^n \tag{5.66}$$

Hence the VC dimension of $\mathcal{C}_p$ is n. ■

5.6.2 Implementation

Any concept from this class can easily be expressed by a tunable network: for each non-zero bit of s it suffices to apply the $\mathbf{X}$ gate controlled by the corresponding qubit. In order to learn a parity function, we are applying the update strategy shown in Algorithm 5.

Let $s \in \mathbb{B}^n$ and $p_s \in \mathcal{C}_p$ be the target concept. The update strategy stems from the equivalence between the two following facts:

- There exist a misclassified input $x_e \in \mathbb{B}^n$ and a correctly classified input $x_c \in \mathbb{B}^n$ such that $|w_H(x_e) - w_H(x_c)| = 1$ and either $1_{x_e} \subset 1_{x_c}$ or $1_{x_c} \subset 1_{x_e}$.

- $1_{x_e} \triangle 1_{x_c} \subseteq 1_s$. Where $1_{x_e} \triangle 1_{x_c}$ is the symmetric difference between these two sets.

Given this observation, the update strategy consists in:

1. Group the measured misclassified inputs by increasing Hamming weight and do the same with the measured correctly classified inputs.

2. For each of the misclassified inputs of Hamming weight i, x_e, check if there is a correctly classified input of weight $i+1$, x_c such that $1_{x_e} \subset 1_{x_c}$. If this is the case, add the gate controlled by the qubit $1_{x_c} \setminus 1_{x_e}$. Do the same with the correctly classified inputs of weight i and the misclassified inputs of weight $i+1$.

These steps alone would have sufficed to tune the network but in order to reduce the number of update steps we added some additional steps. Indeed, because the gates to be updated are each controlled by a single qubit that is in 1_s, they will not be updated anymore and thus we can predict the effect of the updated network on the current measurements:

3. For each of the single control qubit q of the list of gates to be updated, if there exist a misclassified input of weight i, x_c such that $q \in 1_{x_c}$, then remove x_c from the list of misclassified inputs and add x such that $1_x = 1_{x_c} \setminus \{q\}$ to the list of correctly classified inputs of weight $i-1$. Do the same with the correctly classified inputs.

One final observation we can make is that the measured correctly classified inputs of weight 1 x_c are such that $1_{x_c} = \{q\}$ with $q \notin 1_s$. And hence we can add the following step:

4. For each of the correctly classified inputs of weight 1, do the same as 3. but without swapping from misclassified to correctly classified and vice-versa.

5. Restart from 5.6.2. until no more gates to be updated can be found.

Because the gates to be updated are all controlled by a single qubit, we are assured that the final hypothesis will be a parity function, hence our algorithm is a proper learner.

Algorithm 5: Update strategy for parity functions

Data: *errors* and *corrects* the lists of measured inputs that are respectively misclassified and correctly classified
Result: *gates* the list of gates to be updated
Gather the measurements in *errors* by group of same Hamming weight so that $errors[i]$ is the list of misclassified inputs of Hamming weight i;
Do the same with *corrects*;
$gates \leftarrow errors[1]$;
while *A gate can be added to gates* **do**
  for $1 \leq i \leq n-1$ **do**
    for *e in* $errors[i]$ **do**
      for *c in* $corrects[i+1]$ **do**
        if $e \oplus c$ *has Hamming weight 1* **then**
          Add $e \oplus c$ to *gates* if it is not already in;
        end
      end
    end
    Do the same with *corrects* and *errors* switched;
  end
  for $g \in gates$ **do**
    for $1 \leq j \leq n$ **do**
      for $e \in errors[i]$ **do**
        if $1_g \subseteq 1_e$ **then**
          Remove e from $errors[i]$;
          Add $e \oplus g$ to $corrects[i-1]$;
        end
      end
      Do the same with $corrects[i]$;
    end
  end
  $not\_significant \leftarrow corrects[1]$;
  for $n_s \in not\_significant$ **do**
    Do the same as above but pass from $corrects[i]$ to $corrects[i-1]$ and from $errors[i]$ to $errors[i-1]$
  end
end
Return *gates*;

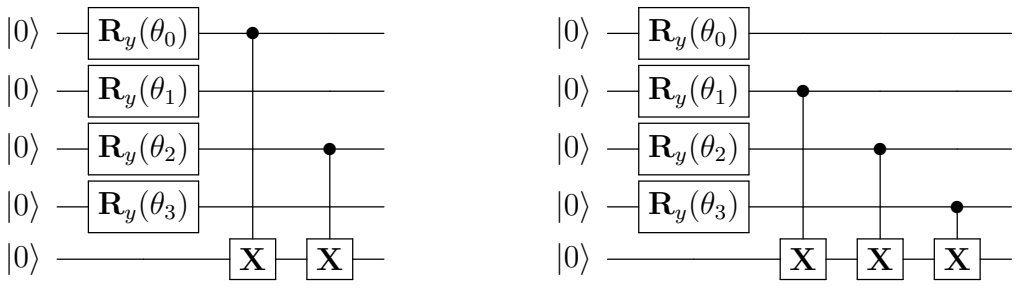
To completely specify the algorithm's query complexity, it remains to determine the VC-dimension dependant part of the complexity. It is clear that when δ is small enough, the number of samples during an update phase will be large enough so that the variety of samples allows for the network to be correctly tuned after one update. We are now interested in the other side of the spectrum: when δ is large and hence the number of samples is small. The number of necessary updates will then be a function of both n and ϵ, in which case we will use the result from [4] to state:

Proposition 5.20:

Let $n \in \mathbb{N}$ and $\mathcal{C}_p \subseteq \mathbb{B}^{\mathbb{B}^n}$ be the class of parity functions of n variables. Let $0 < \epsilon < \frac{1}{4}$, then the VC-dimension dependant part of the query complexity is in:

$$\Omega\left(\frac{n-1}{\epsilon}\right) \tag{5.67}$$

This approach has been implemented[1] for different values of $n \geq 4$. For a given n, the probability distributions over $\mathbb{B}^n$ have been created randomly by applying to each qubit a $\mathbf{R}_y$ rotation gate with an angle randomly chosen in $[0, \pi]$. The controlled $\mathbf{X}$ gates corresponding to the target concept are then added to the circuit. These two blocks taken together are constituting the oracle $\mathbf{EX}(c, D)$. The construction of such oracles for $n = 4$ is illustrated in Figure 5.4. The TNN has then been trained to learn the concepts of the class $\mathcal{C}_p$ according to the training algorithm and the update strategy introduced in this chapter (Algorithm 4 and Algorithm 5).



(a) Oracle for p_{1010}.

(b) Oracle for p_{0111}.

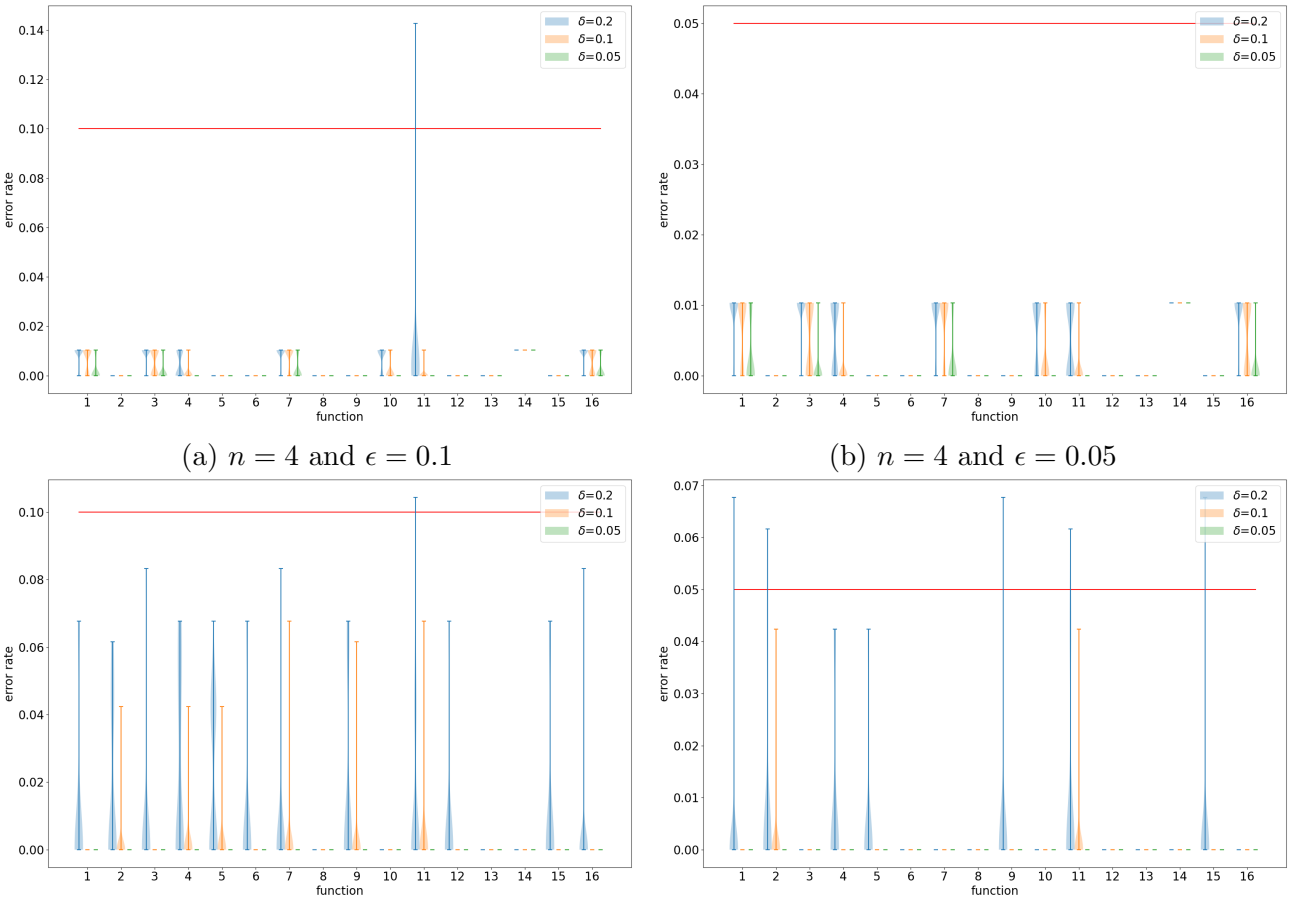
Figure 5.4: Implementation of the oracle for $n = 4$ and different target concepts.

[1]The code can be found in the following repository: https://github.com/vietphamngoc/QPAC

For the different $n \geq 4$, the experiments have been performed for 16 randomly selected concepts from this class. In all cases, different values of ϵ and δ have been used. Once the training has been completed, the error is evaluated using the Qiskit statevector simulator by running the circuit composed of the oracle and the tuned network and getting the amplitude of all the inputs for which the network's output is wrong and hence the final error probability. To account for the overall randomness of the process, each training has been performed 50 times.

5.6.3 Experimental Results

In the first set of experiments, we wished to verify that when the algorithm stops, the training target has indeed been reached. To do so, we trained the network for $n = 4$ and 8 and for different values of ϵ and δ. The results of these experiments are depicted in Figure 5.5 using violin plots. These types of plot represent the occurrence frequency of the values within the



(a) $n = 4$ and $\epsilon = 0.1$

(b) $n = 4$ and $\epsilon = 0.05$

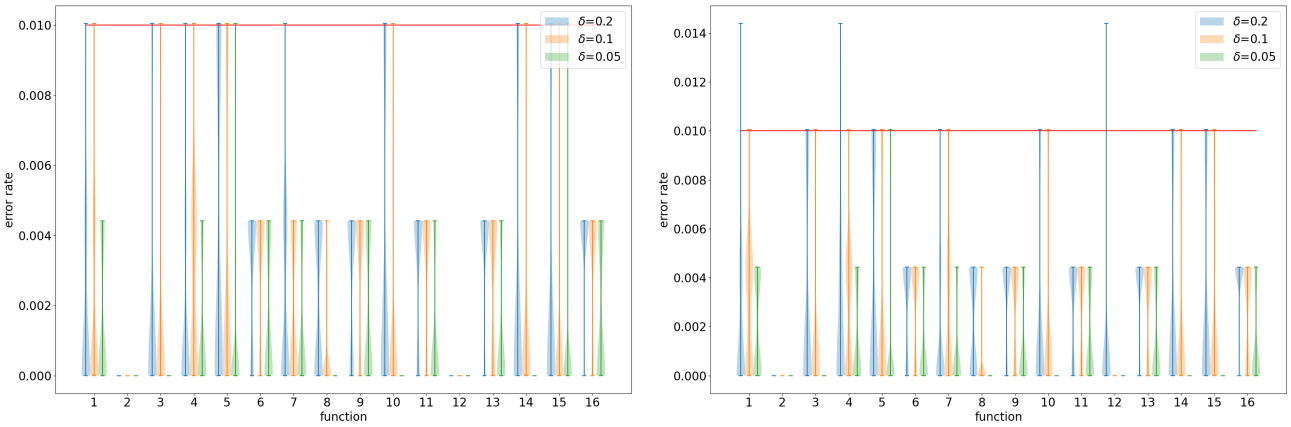
(c) $n = 8$ and $\epsilon = 0.1$

(d) $n = 8$ and $\epsilon = 0.05$

Figure 5.5: Final error rates for different values of n, ϵ and δ. The red line represents ϵ.

data through the width of the violins. From Figure 5.5, we can conclude that the training algorithm works correctly. Indeed, for each of the n, ϵ configurations, the final error rate is for the most part below ϵ with the occasional training ending with an error rate greater than ϵ only occurring for $\delta = 0.2$.

In order to speed up the training process, we investigated the possibility of incrementing m, the number of amplification rounds, not by 1 but with powers of 2. To test this possibility, we trained the network for $n = 6$, $\epsilon = 0.01$ and $\delta = 0.2, 0.1$ and 0.05 with, on one side, an increment by 1 and, on the other side, an increment with powers of 2. The outcomes of these experiments are gathered in Figure 5.6. Given that the final error rate are comparable, we chose to keep



(a) m incremented by 1 (b) m incremented with powers of 2

Figure 5.6: Final error rates for $n = 6$, $\epsilon = 0.01$

this increment schedule. This means that the number of samples during an update phase is in:

$$\Theta\left(\log\left(\frac{1}{\epsilon}\right)\frac{1}{\delta^2}\right) \tag{5.68}$$

We continued the verification of the performances of the algorithm by confirming that the training target can be reached independently of the probability distribution. To do so, we trained the network for $n = 6$, $\epsilon = 0.1$ and $\delta = 0.1$ (with m incremented with powers of 2). Figure 5.7 depicts the error rate resulting from the training of the network with these given parameters and 3 randomly generated probability distributions over $\mathbb{B}^6$. Figure 5.7 shows that regardless of the distribution, the training will result in a network tuned such that the final error rate is most probably less than ϵ.

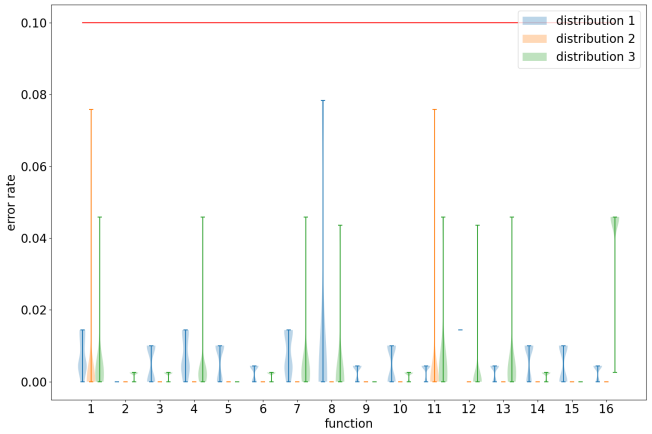


Figure 5.7: Final error rate for $n = 6$, $\epsilon = 0.1$ and $\delta = 0.1$ obtained with 3 different probability distributions

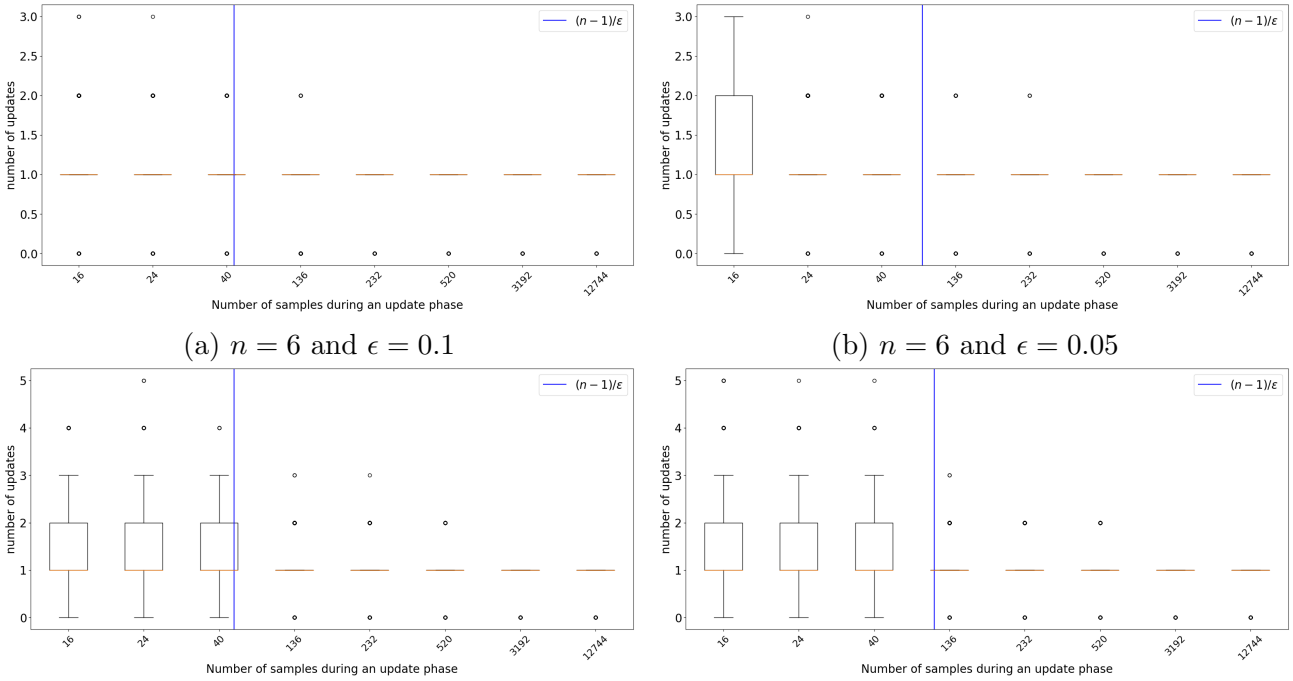
Through the experiments that we have conducted, we have established that our training algorithm works as intended with, for δ small enough, a query complexity in $\Theta\left(\log\left(\frac{1}{\epsilon}\right)\frac{1}{\delta^2}\right)$. Taken together with Property 5.20, it follows that the query complexity of our algorithm to learn parity functions is:

$$\Omega\left(\frac{n-1}{\epsilon} + \log\left(\frac{1}{\epsilon}\right)\frac{1}{\delta^2}\right) \tag{5.69}$$

To verify this, we conducted a final set of experiments wherein we trained the network for $n \in \{6, 7\}$, $\epsilon \in \{0.1, 0.05\}$ and different values for δ. The results are shown in Figure 5.8. The number of updates is plotted against the number of samples during an update phase, calculated from the values of ϵ and δ. The data comes from the aggregation of the training for 16 different functions for each dimension, trained 50 times each, with the repetitions being evenly split between two different probability distributions over $\mathbb{B}^6$ and $\mathbb{B}^7$. The results are depicted using boxes where the red line represents the median and the lower and upper limits of the box represent the first and third quartiles respectively. The whiskers point to the maximum and minimum values while the rounds identify outliers. Figure 5.8 shows that when the number of samples during an update phase is larger than $\frac{n-1}{\epsilon}$, the training only requires one update, which validates the sample complexity given in Equation 5.69.

In [7] the authors proposed the following complexity:

$$\Omega\left(\frac{n-1}{\epsilon} + \frac{1}{\epsilon}\log\left(\frac{1}{\delta}\right)\right) \tag{5.70}$$



(a) $n = 6$ and $\epsilon = 0.1$

(b) $n = 6$ and $\epsilon = 0.05$

(c) $n = 7$ and $\epsilon = 0.1$

(d) $n = 7$ and $\epsilon = 0.05$

Figure 5.8: Number of updates necessary to train the network for $n \in \{6, 7\}$, $\epsilon \in \{0.1, 0.05\}$ and different numbers of samples, calculated from ϵ and δ

Were the VC-dimension independent part has been obtained by considering two distinct concepts and placing a distribution depending on ϵ. The complexity then stems from the number of samples needed to distinguish two states. While our complexity in δ is exponentially worst, we have an exponential gain in terms of ϵ. So for applications where a small ϵ is required, our algorithm will require fewer samples.

5.7 Conclusion

In this chapter, we have designed an algorithm to train a tunable quantum neural network in the QPAC-learning framework. We have shown that when the algorithm stops, the network expresses a hypothesis that is ϵ-close to the target concept with high probability, i.e. the network has learnt the target concept. We have implemented this algorithm and tested it on the class of parity functions. The results of the experiments show that our algorithm works correctly and is efficient with a complexity of $\Omega\left(\frac{n-1}{\epsilon} + \log\left(\frac{1}{\epsilon}\right)\frac{1}{\delta^2}\right)$. When comparing to the literature, we found that our algorithm will perform better for applications where ϵ needs to

be small.

Chapter 6

Exact Learning with a Quantum Example Oracle

6.1 Introduction

Introduced in [2], the model of exact learning is, together with PAC learning, one of the main frameworks of learning theory. Initially, the learner was given access to two different oracles: the membership oracle and the equivalence oracle. Suppose that c is the target concept. Given x, a call to the membership oracle will return $c(x)$ and given the learner's current hypothesis h, the equivalence oracle will output a randomly chosen $x \in \mathbb{B}^n$ such that $h(x) \neq c(x)$ should such x exists The goal is then to produce a hypothesis h^* that is pointwise equal to c. Given the impracticality of the equivalence oracle it has become common practice to only work with the membership oracle [16, 25] or forgo these two oracles altogether and use another kind, generally a random example oracle. In this case, the preferred flavour is the uniformly distributed examples [70, 54, 38]. These last two research directions naturally gave rise to quantum generalisations of these oracles [4, 8].

In this chapter, we will study the performances of the tunable quantum neural network as introduced in Chapter 3 in the quantum exact learning framework when the learner is given access to the quantum version of the uniform example oracle. To do so, we devised a training

algorithm that leverages amplitude amplification and fine-tuned it on the task of learning generic Boolean functions. We show that in this case, it performs better than a naive algorithm that does not use amplitude amplification. Finally, we adapted this algorithm to learn the class of k-juntas and found that it requires $O(n^2 2^k)$ examples. This query complexity is, in some cases, lower than what can be found in the literature.

6.2 Quantum Exact Learning

As said previously, quantum exact learning is the extension of the exact learning framework to quantum oracles. We define the quantum membership oracle [4, 3, 71] and the uniform quantum example oracle [8] and formalise the training target in the quantum exact learning framework. The definitions for concept class, concept and sample complexity remain the same as in Section 5.2.

Definition 6.1:

Let $n \in \mathbb{N}$ and suppose that the target concept is $c \in \mathbb{B}^{\mathbb{B}^n}$. Then the quantum membership oracle $\mathbf{MO}(c)$ will have the following action:

$$\forall x \in \mathbb{B}^n, \forall b \in \mathbb{B},\ \mathbf{MO}(c)\,|x, b\rangle = |x, b \oplus c(x)\rangle \tag{6.1}$$

On the other hand, the uniform quantum example oracle is defined as:

Definition 6.2:

Let $n \in \mathbb{N}$ and $c \in \mathbb{B}^{\mathbb{B}^n}$ be the target concept, then a query to the quantum example oracle $\mathbf{EX}(c)$ will return the superposition $|\psi(c)\rangle$ such that:

$$|\psi(c)\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{B}^n} |x, c(x)\rangle \tag{6.2}$$

By using either one of these oracles, the goal of the learner in the exact learning framework is:

Definition 6.3:

Let $n \in \mathbb{N}$ and $\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n}$ be a concept class. An algorithm is said to be an exact learner for $\mathcal{C}$ if for every $c \in \mathcal{C}$, it outputs a hypothesis $h \in \mathbb{B}^{\mathbb{B}^n}$ such that, with high probability:

$$\forall x \in \mathbb{B}^{\mathbb{B}^n}, h(x) = c(x) \tag{6.3}$$

This notion of high probability is to be defined. While some authors use a threshold of 2/3 [4], we chose to use 0.95.

In this chapter, we aim at performing exact learning with a learner having access to the uniform quantum example oracle.

6.3 Learning Algorithm

6.3.1 Preliminary Analysis

Let $n \in \mathbb{N}$ and $c \in \mathbb{B}^{\mathbb{B}^n}$. As stated previously, we have access to the oracle $\mathbf{EX}(c)$, a query to which will produce the superposition $|\psi(c)\rangle$ given in Equation 6.2.

Now suppose that, in its current state (denoted $\mathbf{T}(h)$), the network is expressing $h \in \mathbb{B}^{\mathbb{B}^n}$ then:

$$\mathbf{T}(h)\,|\psi(c)\rangle = \frac{1}{\sqrt{2^n}} \sum_{h(x)=c(x)} |x\rangle\,|0\rangle + \frac{1}{\sqrt{2^n}} \sum_{h(x)\neq c(x)} |x\rangle\,|1\rangle \tag{6.4}$$

So from Equation 6.4, it follows that the number of misclassified inputs and the probability of measuring the read-out qubit in state $|1\rangle$ are directly proportional:

Proposition 6.4:

Let $E(h) = \{x \in \mathbb{B}^n \mid h(x) \neq c(x)\}$ be the set of misclassified inputs and P_1, the probability of measuring the read-out qubit in state $|1\rangle$, then:

$$P_1 = \frac{|E(h)|}{2^n} \tag{6.5}$$

In particular, we have:

$$\min_{P_1>0} P_1 = \frac{1}{2^n} \tag{6.6}$$

The idea is thus to use amplitude amplification (AA) to boost this probability and hence measure as many misclassified inputs as possible before updating the network. Recall that amplitude amplification is performed thanks to the diffusion operator. This operator itself is composed of unitaries that were introduced in Section 2.3.4. In this setup, the diffusion operator is as follows:

Definition 6.5:

Let $n \in \mathbb{N}$ and $c \in \mathbb{B}^{\mathbb{B}^n}$ be the target concept. Suppose that the network is expressing the function $h \in \mathbb{B}^{\mathbb{B}^n}$ in its current state, denoted $\mathbf{T}(h)$, we set:

$$\mathcal{X}_{\alpha_0}(c) = \mathbf{I} - 2\,|\psi(c)\rangle\,\langle\psi(c)| \tag{6.7}$$

$$\mathbf{U}(h) = \mathbf{T}(h) \tag{6.8}$$

And

$$\mathcal{X}_G = \mathbf{I}^{\otimes n} \otimes \mathbf{Z} \tag{6.9}$$

This allows us to define the diffusion operator:

$$\mathbf{Q}(c,h) = -\mathbf{U}(h)\mathcal{X}_{\alpha_0}(c)\mathbf{U}^\dagger(h)\mathcal{X}_G \tag{6.10}$$

Because the minimum non-zero probability of measuring a misclassified input is known, we can also determine the maximum number of iterations of the diffusion operator needed to appropriately amplify the amplitudes of the inputs of interest:

Definition 6.6:

Let $n \in \mathbb{N}$, we define:

$$\theta_{\min} = \arcsin\left(\frac{1}{\sqrt{2^n}}\right) \tag{6.11}$$

And:

$$m_{\max} = \arg\min_{k\in\mathbb{N}} \left|(2k+1)\theta_{\min} - \frac{\pi}{2}\right| \tag{6.12}$$

We now show the following property:

Proposition 6.7:

Let P_1 be a non-zero probability of measuring the read-out qubit in state $|1\rangle$ in this setup. Let us denote $P_1^{(m)}$ this same probability but after m rounds of amplitude amplification. Then there exists $m_{1/2} \leq m_{\max}$ such that:

$$P_1^{(m_{1/2})} \geq \frac{1}{2} \tag{6.13}$$

Proof:

Let $\theta = \arcsin\left(\sqrt{P_1}\right)$, then because P_1 is non-zero, we have:

$$\frac{1}{2^n} \leq P_1 \leq 1 \tag{6.14}$$

Thus:

$$\theta_{\min} \leq \theta \leq \frac{\pi}{2} \tag{6.15}$$

If $\frac{\pi}{4} \leq \theta \leq \frac{\pi}{2}$, then we can take $m_{1/2} = 0$. Otherwise, there exist $m > 0$ such that:

$$\frac{\pi}{4(2m+1)} \leq \theta < \frac{\pi}{4(2m-1)} \tag{6.16}$$

So:

$$\frac{\pi}{4} \leq (2m+1)\theta < \frac{\pi}{4} + \frac{\pi}{2(2m-1)} \leq \frac{3\pi}{4} \tag{6.17}$$

Hence after m rounds of amplitude amplification, we have:

$$\frac{1}{2} \leq \sin^2((2m+1)\theta) = P_1^{(m)} \tag{6.18}$$

Let us denote this m, $m_{1/2}$, then because $P_1 \geq \frac{1}{2^n}$, we also have $m_{1/2} \leq m_{\max}$ ■

The general idea for the algorithm is the following:

1. E is the set of misclassified inputs, initialised with the empty set

2. For $m \in [0, m_{\max}]$, perform m rounds of AA:

   - Perform s measurements
     - If 1 is measured on the read-out qubit, add the measurement of the n first qubits to E the set of misclassified inputs

3. Update the TNN according to E and restart from 1.

4. The algorithm stops if E remains empty when 3. is reached

In order to ensure that the algorithm correctly terminates 95% of the time, we give a condition on the number of measurements s.

Definition 6.8:

Let $m \in [0, m_{\max}]$ and $s \in \mathbb{N}$, we define N_m the number of times the read-out qubit is measured in state $|1\rangle$ after m rounds of amplification and over s measurements. Then:

$$N_m \sim \mathrm{B}\left(s, P_1^{(m)}\right) \tag{6.19}$$

Using Definition 6.8, we can now estimate the probability for the algorithm to terminate incorrectly:

Proposition 6.9:

The probability that the algorithm stops when there are still misclassified inputs is given by:

$$P(N_0 = 0, \ldots, N_{m_{\max}} = 0 \mid P_1 > 0) \tag{6.20}$$

If $s \geq 5$, then

$$P(N_0 = 0, \ldots, N_{m_{\max}} = 0 \mid P_1 > 0) \leq 0.05 \tag{6.21}$$

Proof:

We have:

$$P(N_0 = 0, \ldots, N_{m_{\max}} = 0 \mid P_1 > 0) \leq P(N_{m_{1/2}} = 0 \mid P_1 > 0) \tag{6.22}$$

But

$$P(N_{m_{1/2}} = 0 \mid P_1 > 0) = (1 - P_1^{(m_{1/2})})^s \leq \frac{1}{2^s} \tag{6.23}$$

So by taking $s \geq 5$, we do have:

$$P(N_0 = 0, \ldots, N_{m_{\max}} = 0 \mid P_1 > 0) \leq 0.05 \tag{6.24}$$

■

Given this condition on s, we will now specify further the learning algorithm through the task of learning a generic Boolean function.

6.3.2 Learning a Generic Boolean Function

Let $n \in \mathbb{N}$, in this section, we aim at learning $c \in \mathbb{B}^{\mathbb{B}^n}$ without assuming any property or structure on c, that is we take the concept class $\mathcal{C}$ to be $\mathbb{B}^{\mathbb{B}^n}$. This task allows us to specify the learning algorithm to train a TNN in the exact learning framework. To do so we compare it to a naive approach. Without assuming anything about the target concept c, the naive way to correctly learn it, is to evaluate $c(x)$ for all $x \in \mathbb{B}^{\mathbb{B}^n}$. Given that we only have access to a uniform quantum example oracle and not a quantum membership oracle the expected number of queries to measure all the inputs is given by:

Theorem 6.10:

Let K be a set with $|K| = N$ and $\{k_i\}$ be uniformly sampled elements from K. Now let $s \in \mathbb{N}$ such that:

$$\{k_i\}_{1 \leq i \leq s} = K \tag{6.25}$$

This problem is known as the coupon collector's problem [28] and it can be shown that $\mathbb{E}(s)$, the expected value of s, is:

$$\mathbb{E}(s) = N \ln(N) \tag{6.26}$$

Naive algorithm

Using Theorem 6.10 we propose the naive learning algorithm described in Algorithm 6. With this algorithm, during each update phase, $\Theta(n2^n)$ queries to the oracle are done. Because after each update, we should have measured all of the misclassified inputs, Theorem 4.8 can be applied and the total number of updates is in $\Theta(n)$. The total number of queries to the example oracle can thus be evaluated to be in $\Theta(n^2 2^n)$.

Algorithm 6: Naive algorithm

```
Data: |ψ(c)⟩ and T the network with all gates initialised to I
Result: Tuned network T expressing c
E ← [0];
s ← ⌊2^n ln(2^n)⌋;
while E ≠ ∅ do
    E ← [];
    for 1 ≤ i ≤ s do
        Measure T |ψ(c)⟩;
        if 1 is measured on the ancillary qubit then
            Add the first n qubits to E;
        end
    end
    for u ∈ E do
        Update G_u in T;
    end
end
```

One property of this algorithm is that it puts an emphasis on the misclassified inputs. However within these $2^n \ln(2^n)$ measurements, a mix of misclassified and correctly classified inputs will be measured. By using amplitude amplification, the results of the measurements can then be focused on the inputs of interest. The idea is thus to adapt the number of samples to the number of amplification rounds in order to measure just what is necessary.

A first improvement

Let $N_{err} \in [1, 2^n]$ be the number of misclassified inputs and $\theta_{err} \in \left]0, \frac{\pi}{2}\right]$ defined by:

$$\theta_{err} = \arcsin\left(\sqrt{\frac{N_{err}}{2^n}}\right) = \arcsin\left(\sqrt{P_1}\right) \tag{6.27}$$

Then there exists $m_{err} \in \mathbb{N}$ such that:

$$\theta_{err} \in \left] \frac{\pi}{2(2m_{err}+3)}, \frac{\pi}{2(2m_{err}+1)} \right] \tag{6.28}$$

This yields:

$$N_{err} \in \left] \sin^2\left(\frac{\pi}{2(2m_{err}+3)}\right) 2^n, \sin^2\left(\frac{\pi}{2(2m_{err}+1)}\right) 2^n \right] \tag{6.29}$$

And after m_{err} rounds of amplification, the probability $P_1^{(m_{err})}$ of measuring the readout qubit in state $|1\rangle$ is now:

$$P_1^{(m_{err})} \in \left] \sin^2\left(\frac{\pi}{2} - \frac{\pi}{2m_{err}+3}\right), 1 \right] \tag{6.30}$$

For m sufficiently large enough, if $N_{err} \in \left] \sin^2\left(\frac{\pi}{2(2m_{err}+3)}\right) 2^n, \sin^2\left(\frac{\pi}{2(2m_{err}+1)}\right) 2^n \right]$, then after m rounds of amplification, the results of the measurements will mostly be misclassified inputs. Applying Theorem 6.10 and taking into account the condition on the number of samples from Property 6.9 we define the following:

Definition 6.11:

Let $n \in \mathbb{N}$ and $m_{\max}$ as in Definition 6.6. Then for $m \in [0 \dots m_{\max}]$, we denote:

$$N_m = \sin^2\left(\frac{\pi}{2(2m+3)}\right) 2^n \tag{6.31}$$

And

$$s_m = \max(5, N_m \ln(N_m)) \tag{6.32}$$

The total number of samples during an update phase is thus: $\sum_{m=0}^{m_{\max}} s_m$ which is to be compared to $2^n \ln(2^n)$

Proposition 6.12:

Let $n \in \mathbb{N}$, $m_{\max}$ as in Definition 6.6 and s_m defined as in Definition 6.11. Then for n large enough:

$$\sum_{k=0}^{m_{\max}} s_m < 2^n \ln(2^n) \tag{6.33}$$

Proof:

For the sake of clarity, for $m \in [0 \dots m_{\max}]$, we define:

$$u_m = \sin^2\left(\frac{\pi}{2(2m+3)}\right) \tag{6.34}$$

So that $N_m = u_m 2^n$. Instead of directly showing $\sum_{m=0}^{m_{\max}} s_m < 2^n \ln(2^n)$, we will instead show:

$$\sum_{m=0}^{m_{\max}} (N_m \ln(N_m) + 5) < 2^n \ln(2^n) \tag{6.35}$$

For the first sum:

$$\sum_{m=0}^{m_{\max}} N_m \ln(N_m) = \sum_{m=0}^{m_{\max}} \ln\left(N_m^{N_m}\right) = \ln\left(\prod_{m=0}^{m_{\max}} N_m^{N_m}\right) \tag{6.36}$$

And

$$\prod_{m=0}^{m_{\max}} N_m^{N_m} = \prod_{m=0}^{m_{\max}} (u_m 2^n)^{N_m} = \prod_{m=0}^{m_{\max}} u_m^{N_m} \prod_{m=0}^{m_{\max}} (2^n)^{N_m} = \prod_{m=0}^{m_{\max}} u_m^{N_m} \prod_{m=0}^{m_{\max}} (2^n)^{u_m 2^n} \tag{6.37}$$

$$= \prod_{m=0}^{m_{\max}} u_m^{N_m} \prod_{m=0}^{m_{\max}} ((2^n)^{2^n})^{u_m} \tag{6.38}$$

Because $u_m \leq 1$, we have $\prod_{m=0}^{m_{\max}} u_m^{N_m} \leq 1$, hence:

$$\prod_{m=0}^{m_{\max}} N_m^{N_m} \leq \prod_{m=0}^{m_{\max}} ((2^n)^{2^n})^{u_m} \tag{6.39}$$

But:

$$\prod_{m=0}^{m_{\max}} ((2^n)^{2^n})^{u_m} = ((2^n)^{2^n})^{\sum_{m=0}^{m_{\max}} u_m} \tag{6.40}$$

And:

$$\sum_{m=0}^{m_{\max}} u_m = \sum_{m=0}^{m_{\max}} \sin^2\left(\frac{\pi}{2(2m+3)}\right) \tag{6.41}$$

$$\leq \sum_{m=0}^{m_{\max}} \left(\frac{\pi}{2(2m+3)}\right)^2 \tag{6.42}$$

$$\leq \frac{\pi^2}{4} \sum_{m=1}^{m_{\max}+1} \frac{1}{(2m+1)^2} \tag{6.43}$$

$$\leq \frac{\pi^2}{4}\left(\sum_{m=1}^{+\infty} \frac{1}{m^2} - \sum_{m=1}^{+\infty} \frac{1}{(2m)^2} - 1\right) \tag{6.44}$$

$$\leq \frac{\pi^2}{4}\left(\frac{\pi^2}{6} - \frac{\pi^2}{24} - 1\right) \tag{6.45}$$

$$\leq 0.58 \tag{6.46}$$

This means:

$$\prod_{m=0}^{m_{\max}} N_m^{N_m} \leq ((2^n)^{2^n})^{0.58} \tag{6.47}$$

Hence:

$$\sum_{m=0}^{m_{\max}} N_m \ln(N_m) \leq 0.58 \times 2^n \ln(2^n) \tag{6.48}$$

Now for the second sum:

$$\sum_{m=0}^{m_{\max}} 5 = 5(m_{\max}+1) \tag{6.49}$$

But $m_{\max} \approx \sqrt{2^n}$ so for n large enough:

$$\sum_{m=0}^{m_{\max}} 5 \approx 5\sqrt{2^n} \lll 2^n \ln(2^n) \tag{6.50}$$

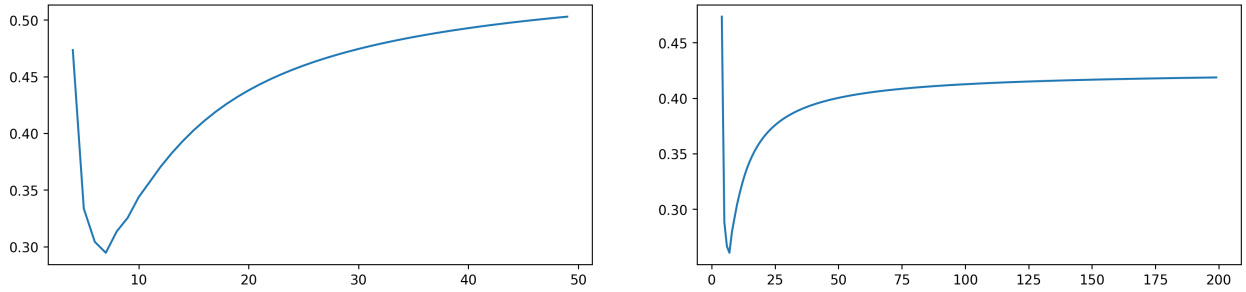
Its contribution is thus negligible in the overall sum and we have:

$$\sum_{m=0}^{m_{\max}} s_m < \sum_{m=0}^{m_{\max}} (N_m \ln(N_m) + 5) < 2^n \ln(2^n) \tag{6.51}$$

■

In order to have a better idea of the reduction in oracle queries introduced by this algorithm, we have plotted in Figure 6.1 the ratio, as a function of n for $n \geq 4$, between the total number

of samples and $2^n \ln(2^n)$. As shown in Figure 6.1a, for n large enough, the total number of



(a) k incremented by 1 (b) k incremented with powers of 2

Figure 6.1: Ratio $\sum_{m=0}^{m_{\max}} s_m / 2^n \ln(2^n)$ as a function of n for $n \geq 4$. In Figure 6.1a m has been incremented by 1 while in Figure 6.1b, m has been incremented with powers of 2

samples is approximately halved when compared to the naive algorithm. To reduce further this number and speed up the training algorithm, we decided to increment m, the number of amplification rounds, not by 1 but with powers of 2. The ratio of the total number of queries using this scheme, compared to the naive algorithm is given in Figure 6.1b. In this case, we can see that it has been divided by more than 2. For these reasons, the increment schedule for m will now be:

$$[0, 1, 2, 4, \ldots, m_{\max}] \tag{6.52}$$

The refined procedure is detailed in Algorithm 7. If m is the number of amplification rounds, the circuit used in Algorithm 7 is depicted in Figure 6.2.

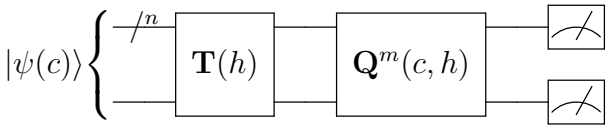


Figure 6.2: Quantum circuit for the refined algorithm

A further refinement

One issue with this algorithm resides in the fact that for $n \in \mathbb{N}$ and $0 \leq m \leq m_{\max}$, the range of misclassified inputs to be covered is:

$$\left]\sin^2\left(\frac{\pi}{2(2m+3)}\right)2^n, \sin^2\left(\frac{\pi}{2(2m+1)}\right)2^n\right] \tag{6.53}$$

Algorithm 7: Refined algorithm

Data: $|\psi(c)\rangle$ and $\mathbf{T}$ the network with all gates initialised to $\mathbf{I}$
Result: Tuned network $\mathbf{T}$ expressing c
$E \leftarrow [0]$;
$schedule \leftarrow [0, 1, 2, 4, \ldots, m_{\max}]$;
while $E \neq \varnothing$ **do**
  $E \leftarrow []$;
  for $m \in schedule$ **do**
    $N \leftarrow \sin^2\left(\frac{\pi}{2(2m+3)}\right) 2^n$;
    $S \leftarrow \max(5, N\ln(N))$;
    Perform m rounds of amplification;
    for $1 \leq i \leq S$ **do**
      Measure ;
      if *1 is measured on the ancillary qubit* **then**
        Add the first n qubits to E;
      end
    end
  end
  for $u \in E$ **do**
    Update $\mathbf{G}_u$ in $\mathbf{T}$;
  end
end

So as m increases, the range will decrease. This means that approximating the number of misclassified inputs with the lower bound becomes more accurate as m increases. However, for small m, this approximation is not as accurate. For example, for $m = 0$, the range given in Equation 6.53 becomes:

$$\left]\frac{2^n}{4}, 2^n\right] \tag{6.54}$$

This range can be refined by adding a second ancillary qubit, initialised in state $|0\rangle$ and a rotation gate acting on this qubit and controlled by the first ancillary qubit. We give a more accurate definition of this gate:

Definition 6.13:

Let $m_0 \in \mathbb{N}$, we define:

$$\theta_{m_0} = \frac{\pi}{2(2m_0 + 1)} \tag{6.55}$$

And we denote $\mathbf{CR}_{m_0}$ the controlled rotation (around the y-axis) gate such that:

$$\mathbf{CR}_{m_0} |10\rangle = \cos(\theta_{m_0}) |10\rangle + \sin(\theta_{m_0}) |11\rangle \tag{6.56}$$

Then if $c \in \mathbb{B}^{\mathbb{B}^n}$ is the target concept and the network is expressing $h \in \mathbb{B}^{\mathbb{B}^n}$ in its current state $\mathbf{T}(h)$, we have:

$$\mathbf{CR}_{m_0}\mathbf{T}(h) |\psi(c)\rangle |0\rangle = \sin(\theta_{m_0}) \frac{1}{\sqrt{2^n}} \sum_{h(x)\neq c(x)} |x\rangle |11\rangle + |\perp\rangle \tag{6.57}$$

So the inputs of interest are now marked with the two ancillary qubits being in $|11\rangle$ and the probability P_{11} of measuring such inputs is now such that:

$$P_{11} \in \left[0, \sin^2(\theta_{m_0})\right] \tag{6.58}$$

And in this case:

$$\min_{P_{11}>0} = \sin^2(\theta_{m_0}) \frac{1}{2^n} \tag{6.59}$$

We adapt Definitions 6.5 and 6.6 to take into account this modification:

Definition 6.14:

Let $n \in \mathbb{N}$, $c \in \mathbb{B}^{\mathbb{B}^n}$ be the target concept and $m_0 \in \mathbb{N}$. Suppose that the network is expressing the function $h \in \mathbb{B}^{\mathbb{B}^n}$ in its current state $\mathbf{T}(h)$. We denote:

$$\mathcal{X}_{\alpha_0}(c) = (\mathbf{I} - 2 |\psi(c)\rangle \langle\psi(c)|) \otimes \mathbf{I} \tag{6.60}$$

$$\mathbf{U}_{m_0}(h) = \mathbf{CR}_{m_0}\mathbf{T}(h) \tag{6.61}$$

And

$$\mathcal{X}_G = \mathbf{I}^{\otimes n+1} \otimes \mathbf{Z} \tag{6.62}$$

The diffusion operator $\mathbf{Q}_{m_0}(c, h)$ is then defined as:

$$\mathbf{Q}_{m_0}(c, h) = -\mathbf{U}_{m_0}(h)\mathcal{X}_{\alpha_0}(c)\mathbf{U}^{\dagger}_{m_0}(h)\mathcal{X}_G \tag{6.63}$$

We now define the quantities that are related to this modified process.

Definition 6.15:

Let $n \in \mathbb{N}$ and $m_0 \in \mathbb{N}$, we denote:

$$\theta_{\min,m_0} = \arcsin\left(\sin\left(\theta_{m_0}\right)\frac{1}{\sqrt{2^n}}\right) \tag{6.64}$$

And:

$$m_{\max,m_0} = \arg\min_{m\in\mathbb{N}}\left|(2m+1)\theta_{\min,m_0} - \frac{\pi}{2}\right| \tag{6.65}$$

Let $m_0 \in \mathbb{N}$. Now let $\theta_{err} \in \left]0, \frac{\pi}{2}\right]$ such that:

$$\theta_{err} = \arcsin\left(\sqrt{P_{11}}\right) \tag{6.66}$$

Then there exists $m \in \mathbb{N}$ such that:

$$\theta_{err} \in \left]\frac{\pi}{2(2m+3)}, \frac{\pi}{2(2m+1)}\right] \tag{6.67}$$

But according to Equation 6.58, we have $P_{11} \leq \sin^2\left(\theta_{m_0}\right)$ so it follows that:

$$m \geq m_0 \tag{6.68}$$

Now if N_{err} is the number of misclassified inputs, then Equation 6.57 yields:

$$N_{err} = \sin^2(\theta_{err})\frac{2^n}{\sin^2(\theta_{m_0})} \tag{6.69}$$

Hence:

$$N_{err} \in \left]\sin^2\left(\frac{\pi}{2(2m+3)}\right)\frac{2^n}{\sin^2(\theta_{m_0})}, \sin^2\left(\frac{\pi}{2(2m+1)}\right)\frac{2^n}{\sin^2(\theta_{m_0})}\right] \tag{6.70}$$

So for $m \geq m_0$, the ratio between the two bounds of this interval is:

$$1 \leq \frac{\sin^2\left(\frac{\pi}{2(2m+1)}\right)}{\sin^2\left(\frac{\pi}{2(2m+3)}\right)} \leq \frac{\sin^2\left(\frac{\pi}{2(2m_0+1)}\right)}{\sin^2\left(\frac{\pi}{2(2m_0+3)}\right)} \approx \left(\frac{m_0+3}{m_0+1}\right)^2 = \left(1+\frac{2}{m_0+1}\right)^2 \tag{6.71}$$

Where the approximation is valid for $m_0 > 0$. So as m_0 grows, it is possible for the bounds of the interval given in Equation 6.70 to become quite close. Moreover, after m rounds of amplitude amplification, we have:

$$P_{11}^{(m)} \in \left]\sin^2\left(\frac{2m+1}{2m+3}\frac{\pi}{2}\right), 1\right] \tag{6.72}$$

With:

$$\sin^2\left(\frac{2m+1}{2m+3}\frac{\pi}{2}\right) = \sin^2\left(\frac{\pi}{2} - \frac{\pi}{2m+3}\right) = \cos^2\left(\frac{\pi}{2m+3}\right) = 1 - \sin^2\left(\frac{\pi}{2m+3}\right) \tag{6.73}$$

As $m \geq m_0$, we have:

$$\sin^2\left(\frac{2m+1}{2m+3}\frac{\pi}{2}\right) \geq 1 - \sin^2\left(\frac{\pi}{2m_0+3}\right) \tag{6.74}$$

Here again, for m_0 large enough, it is possible for the lower bound to be very close to 1. This means that for m_0 large enough, whatever the number of misclassified inputs (or equivalently the probability P_{11}), as long as it is non-zero, it is possible to find a $0 \leq m \leq m_{\max,m_0}$ such that after m rounds of amplitude amplification, the measurement process will be a random process where the probability of measuring a misclassified input is close to 1. In addition, the number of such inputs can more accurately be approximated by the lower bound of the interval given in Equation 6.70. However, it does not suffice to take m_0 very large. Indeed, as $m_{\max,m_0}$ increases with m_0, so does the total number of samples. A good choice for this parameter is thus one that allows for the behaviour previously described while ensuring that the number of samples remains small enough.

Definition 6.16:

Let $n \in \mathbb{N}$, $m_0 \in \mathbb{N}$ and $m_{\max,m_0}$ as in Definition 6.15. For $m_0 \leq m \leq m_{\max,k_0}$ we define:

$$N_{m,m_0} = \sin^2\left(\frac{\pi}{2(2m+3)}\right)\frac{2^n}{\sin^2(\theta_{m_0})} \tag{6.75}$$

And

$$s_{m,m_0} = \max\left(N_{m,m_0}\ln(N_{m,m_0}), 5\right) \tag{6.76}$$

The total number of samples during an update phase is thus $\sum_{m=m_0}^{m_{\max}} s_{m,m_0}$ but in order to speed up the algorithm we chose to keep the schedule introduced in the first improvement hence:

Definition 6.17:

Let $n \in \mathbb{N}$, $m_0 \in \mathbb{N}$, $m_{\max,m_0}$ as in Definition 6.15 and s_{m,m_0} as in Definition 6.16, then the total number of samples S_{m_0}during an update phase is given by:

$$S_{m_0} = s_{m_0,m_0} + \sum_{m_0 < 2^p < m_{\max,m_0}} s_{m,m_0} + s_{m_{\max,m_0},m_0} \tag{6.77}$$

To guide the choice for a suitable m_0, the ratio $S_{m_0}/2^n \ln(2^n)$ has been plotted for different values of $n \in \mathbb{N}$ and $m_0 \in \mathbb{N}$ in Figure 6.3. From this figure, it is apparent that the values

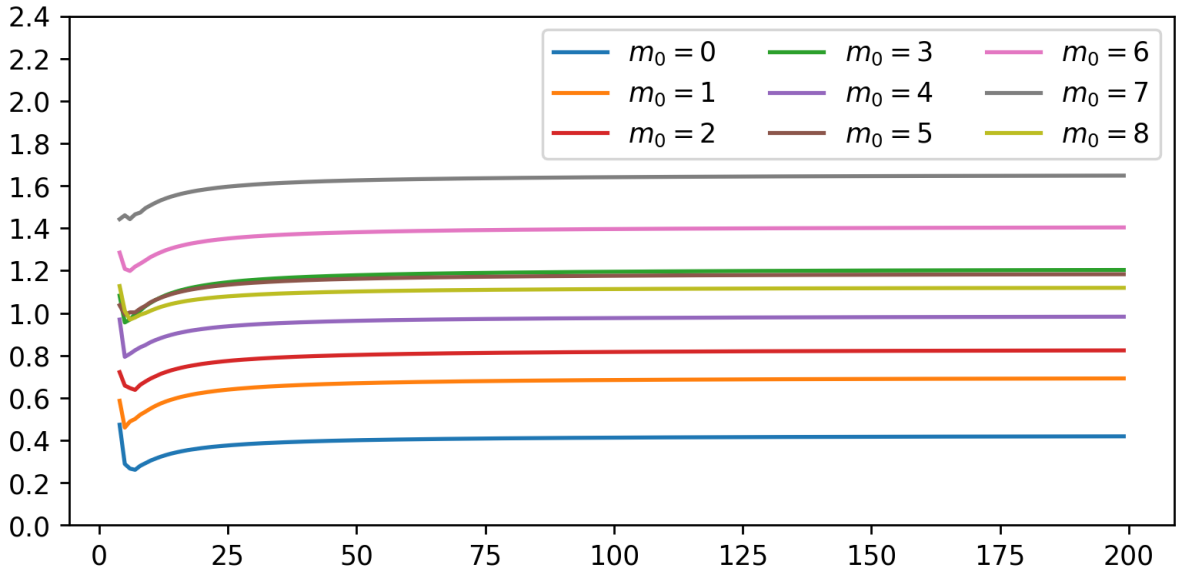


Figure 6.3: Ratio $S_{m_0}/2^n \ln(2^n)$ as a function of n for different values of m_0

$0 \leq m_0 \leq 4$ are suitable with $m_0 = 0$ corresponding to the first improvement introduced earlier. Algorithm 8 describes the training algorithm that will be used to train the network. Notice that when measuring, we still look at the first ancillary qubit. Indeed, because of the controlled rotation gate $\mathbf{CR}_{m_0}$, a misclassified input will result in the ancillary qubits being measured either in state $|10\rangle$ or $|11\rangle$. While the latter is used in the amplification process to refine it, the former still corresponds to a state of interest and it would be a waste to ignore it during the measurements.

For $m_0 \in \mathbb{N}$ and $m \in \mathbb{N}$, the circuit being measured is depicted in Figure 6.4.

Algorithm 8: Refined algorithm

Data: m_0, $|\psi(c)\rangle$ and $\mathbf{T}$ the network with all gates initialised to $\mathbf{I}$
Result: Tuned network $\mathbf{T}$ expressing c
$E \leftarrow [0]$;
$schedule \leftarrow [m_0, 2^{\lfloor \log_2 m_0 \rfloor + 1}, 2^{\lfloor \log_2 m_0 \rfloor + 2}, \ldots, m_{\max, m_0}]$;
while $E \neq \varnothing$ **do**
  $E \leftarrow []$;
  for $m \in schedule$ **do**
    $N \leftarrow \frac{\sin^2\left(\frac{\pi}{2(2k+3)}\right)}{\sin^2\left(\frac{\pi}{2(2k_0+1)}\right)} 2^n$;
    $S \leftarrow \max(5, N\ln(N))$;
    Perform m rounds of amplification;
    for $1 \leq i \leq S$ **do**
      Measure;
      if *1 is measured on the first ancillary qubit* **then**
        Add the first n qubits to E;
      end
    end
  end
  for $u \in E$ **do**
    Update $\mathbf{G}_u$ in $\mathbf{T}$;
  end
end

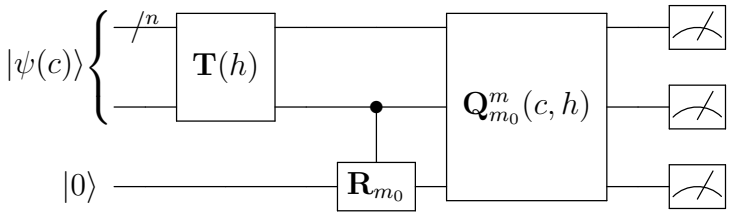
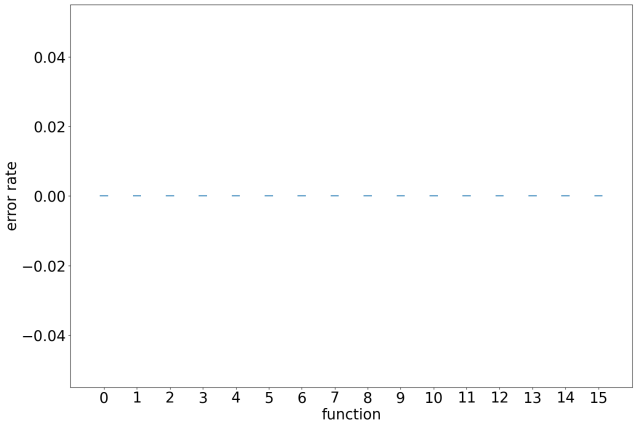


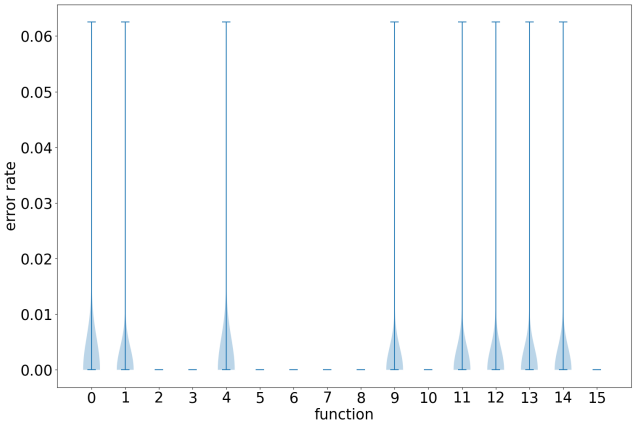
Figure 6.4: Quantum circuit for the improved algorithm

6.3.3 Implementation

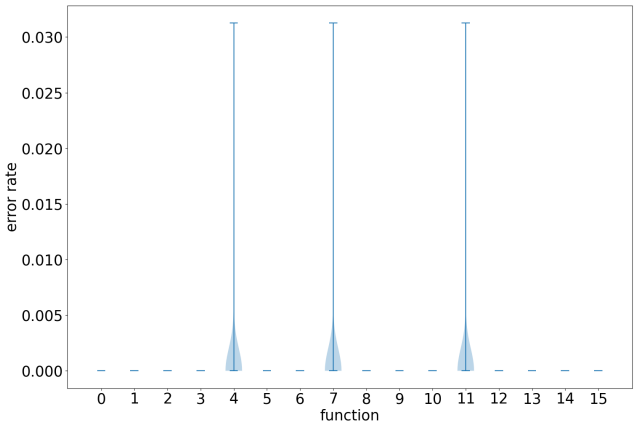
Let $n \in \mathbb{N}$. In order to specify the choice of m_0 we have implemented Algorithm 8 for $0 \leq m_0 \leq 4$ with the target being a generic Boolean function $c \in \mathbb{B}^{\mathbb{B}^n}$. In detail, for $4 \leq n \leq 8$, 16 target functions have been chosen randomly and for each target function, the network has been trained 50 times. Each time, the number of samples needed until the algorithm stops has been recorded as well as the final error rate. As a comparison, the naive algorithm has also been implemented and studied under the same regime. Concerning the error rate, when using amplitude amplification, whatever the values for n and m_0, the final error rate was consistently equal to 0, as shown in Figure 6.5a, indicating that the target function has indeed been exactly learnt. On the other hand, when training with the naive algorithm, some of the experiments for $n = 4$ to 6 failed to exactly learn the target function as depicted in Figures 6.5b to 6.5d.



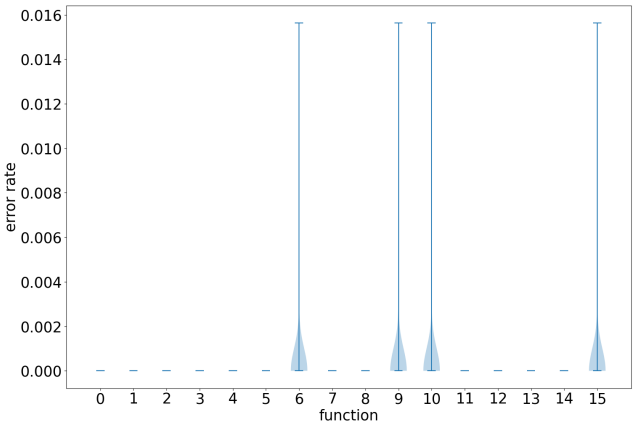
(a) Algorithm using amplitude amplification for $4 \leq n \leq 8$ and $0 \leq m_0 \leq 4$



(b) Naive algorithm for $n = 4$



(c) Naive algorithm for $n = 5$



(d) Naive algorithm for $n = 6$

Figure 6.5: Final error rate for different experiments

Hence, when only looking at the goal of exactly learning, the algorithm using amplitude amplification performs better than the naive algorithm and thus for the different values of m_0 that have been selected. To further confirm the advantage of the refined algorithm over the naive one, we put our focus on the number of samples required to learn the target functions. For each chosen m_0, the mean number of samples taken over all the experiments, i.e. all the 16 functions and the 50 runs by function, has been plotted against the dimension of the input space n for $4 \leq n \leq 8$ in Figure 6.6. As a point of comparison, the same metric has been plotted for the naive algorithm.

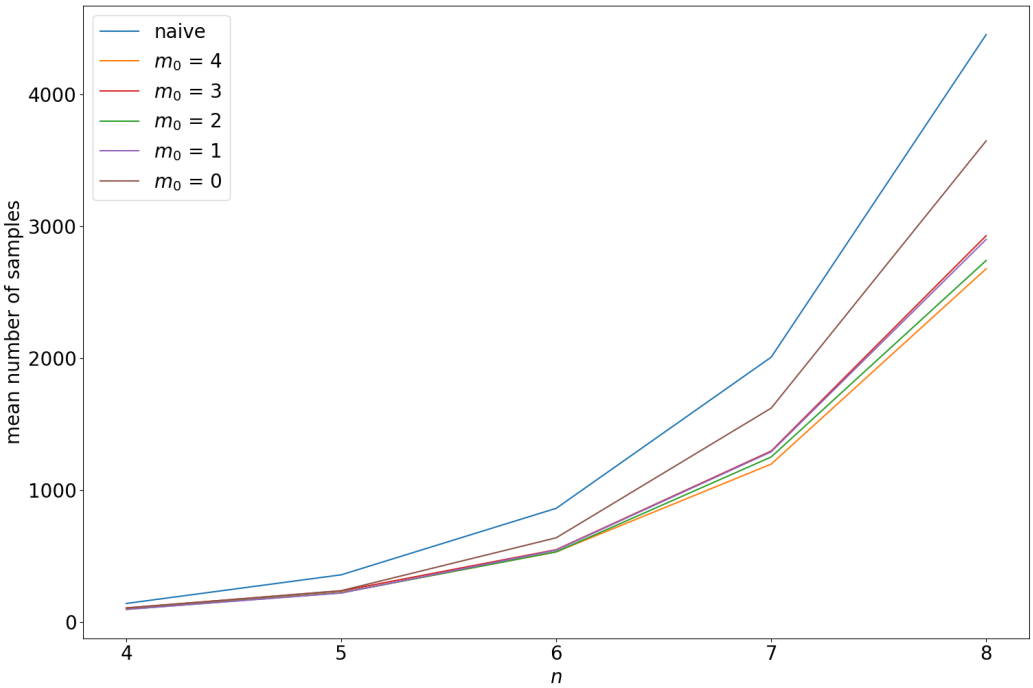


Figure 6.6: Mean number of samples taken over all the experiments (functions and runs) against the dimension n for different values of m_0 ($0 \leq m_0 \leq 4$) and the naive algorithm

What transpires from this comparison is that for the selected m_0, the number of necessary samples is considerably lower than for the naive algorithm, with the gap increasing as the input dimension increases. Two values for m_0 particularly stand out: $m_0 = 2$ and $m_0 = 4$ as they lead to the lowest number of samples with the plot for $m_0 = 4$ being below the one for $m_0 = 2$. While this trend seems to be contradictory with Figure 6.3, this can be explained by the fact that the number of updates when $m_0 = 4$ is lower than when $m_0 = 2$. Seeing that the difference in terms of sample number between these two choices is relatively small, it seems that $m_0 = 2$ offers the best compromise between minimising the number of samples and minimising

the running time of the training algorithm. In Section 6.4, a modified version of Algorithm 8 will be used with $m_0 = 2$ to learn k-juntas. The modifications will concern the number of queries during an update stage as well as the update strategy with regard to the measurement outcomes.

6.4 Learning Positive k-Juntas

6.4.1 Description of the Concept Class and the Update Algorithm

Definition 6.18:

Let $n \in \mathbb{N}^*$ and $k < n$. A Boolean function $c \in \mathbb{B}^{\mathbb{B}^n}$ is said to be a k-junta if its output only depends on at most k of its input variables, these variables are then called the relevant variables. Let $\rho_c \subset [0 \dots n-1]$ be the set of the relevant variables of c, then:

$$|\rho_c| \leq k \tag{6.78}$$

Additionally, we define a positive k-junta, $c \in \mathbb{B}^{\mathbb{B}^n}$, as being a k-junta such that:

$$c(0) = 0 \tag{6.79}$$

And we denote $\mathcal{J}_k^+$ the class of the positive k-juntas.

The fact that a function is a positive k-junta will impose a restriction on its algebraic normal form:

Proposition 6.19:

Let $n \in \mathbb{N}^*$ and $c \in \mathbb{B}^{\mathbb{B}^n}$ a positive k-junta. Then there exist $u_1^{(c)}, \dots, u_p^{(c)} \in \mathbb{B}^n \setminus \{0\}$ with $p < 2^k$ such that:

$$\forall i \in [1, p], 1_{u_i^c} \subseteq \rho_c \tag{6.80}$$

And:

$$c = \bigoplus_{i=1}^{p} m_{u_i^c} \tag{6.81}$$

We will call $u_1^{(c)}, \ldots, u_p^{(c)}$ the principals of c.

Proof:

Let $n \in \mathbb{N}^*$, $k < n$ and $c \in \mathcal{J}_k^+$. For $x \in \mathbb{B}^n$, we denote $x_{|\rho_c}$ the restriction of x to the relevant variables of c. As c is a k-junta, there exist $c' \in \mathbb{B}^{\mathbb{B}^{|\rho_c|}}$ such that:

$$\forall x \in \mathbb{B}^n, c(x) = c'(x_{|\rho_c}) \tag{6.82}$$

As $c' \in \mathbb{B}^{\mathbb{B}^{|\rho_c|}}$, it has an ANF, i.e. there exist $u_1^{(c')}, \ldots, u_p^{(c')} \in \mathbb{B}^{|\rho_c|}$ such that:

$$c' = \bigoplus_{i=1}^{p} m_{u_i^{c'}} \tag{6.83}$$

Because $c(0) = c'(0) = 0$, we also have:

$$u_1^{(c')}, \ldots, u_p^{(c')} \in \mathbb{B}^{|\rho_c|} \setminus \{0\} \tag{6.84}$$

Now to extend c' to c, for $i \in [1 \ldots p]$ we simply extend $u_i^{c'} \in \mathbb{B}^{|\rho_c|}$ to $u_i^c \in \mathbb{B}^n$ by padding with zeros where necessary. This ensures that $1_{u_i^c} \subseteq \rho_c$, $u_1^{(c)}, \ldots, u_p^{(c)} \in \mathbb{B}^n \setminus \{0\}$ and:

$$c = \bigoplus_{i=1}^{p} m_{u_i^c} \tag{6.85}$$

■

To understand the effect of the principals of a k-junta, we will introduce the notion of filter, derived from set theory:

Definition 6.20:

Let $n \in \mathbb{N}$ and $u \in \mathbb{B}^n$. We denote by ${}_{\uparrow}u$ the filter generated by u defined by:

$${}_{\uparrow}u = \{v \in \mathbb{B}^n \mid 1_u \subseteq 1_v\} \tag{6.86}$$

By using the filters generated by the principals of a positive k-junta c, a characterisation of the inputs $v \in \mathbb{B}^n$ such that $c(v) = 1$ can be done.

Proposition 6.21:

Let $n \in \mathbb{N}^*$, and $c \in \mathcal{J}_k^+$. Let $u_1^{(c)}, \ldots, u_p^{(c)}$ be the principals of c, then for $v \in \mathbb{B}^n$:
$c(v) = 1$ if and only if there exists an odd number of principals, $u_i^{(c)}$, such that v is an element of ${}_{\uparrow}u_i^{(c)}$.

Proof:

This result stems from the ANF of the function. ■

In order to show the effects of updating the network, we introduce the following:

Proposition 6.22:

Let $n \in \mathbb{N}^*$ and $c \in \mathcal{J}_k^+$. Let $v \in \mathbb{B}^{\mathbb{B}^n}$ such that $c(v) = 1$, then:

$$\forall w \in {}_{\uparrow}v, c(w) \oplus m_v(w) = 1 \oplus c(w) \tag{6.87}$$

And:

$$\forall w \notin {}_{\uparrow}v, c(w) \oplus m_v(w) = c(w) \tag{6.88}$$

Proof:

This is a consequence of Property 6.21. ■

Let $v \in \mathbb{B}^n$ such that $c(v) = 1$. Then according to Property 6.21:

$$v \in \bigcup_{i=1}^{p} {}_{\uparrow}u_i^{(c)} \tag{6.89}$$

And because ${}_{\uparrow}v \subset \bigcup_{i=1}^{p} {}_{\uparrow}u_i^{(c)}$, it follows that any misclassified input is an element of the filter generated by at least one principal of the target function. By updating the network with gates controlled by misclassified inputs we thus ensure that at any time during the training process, the network will express a hypothesis $h \in \mathbb{B}^{\mathbb{B}^n}$ such that:

$$h = \bigoplus_{u \in G} m_u \text{ where } G \subseteq \bigcup_{i=1}^{p} {}_{\uparrow}u_i^{(c)} \tag{6.90}$$

So the goal of the tuning algorithm is to gradually descend to the principals of the target function by adding to the network gates controlled by inputs of progressively lower Hamming weight. Once a gate controlled by v is added, all the gates controlled by inputs in $_{\uparrow}v$ can be trimmed from the network.

To facilitate this process, it would be advantageous to measure misclassified inputs that are close, in terms of Hamming weight, to the principals of the target function. Indeed, by doing so, the network could be updated with gates that are closer to the principals, thus cutting down the number of update steps. This can be achieved by using, once more, amplitude amplification. The target function being a k-junta, we know that its principals have a Hamming weight of at most k. So by using AA to focus on the inputs with that property, this descent process can be facilitated. Formally, let us define the following:

Definition 6.23:

Let $n \in \mathbb{N}^*$ and $k < n$. We define $\mathcal{X}_{\leq k} \in U(2^n)$ such that:

$$\forall x \in \mathbb{B}^n, \mathcal{X}_{\leq k} |x\rangle = \begin{cases} -|x\rangle & \text{if } w_H(x) \leq k \\ |x\rangle & \text{else} \end{cases} \tag{6.91}$$

Additionally, we define $\mathcal{X}_\psi \in U(2^{n+1})$ by:

$$\mathcal{X}_\psi = 2|\psi(c)\rangle\langle\psi(c)| - \mathbf{I} \tag{6.92}$$

The diffusion operator for the inputs with Hamming weight of at most k is then given by:

$$\mathcal{X}_\psi(\mathcal{X}_{\leq k} \otimes \mathbf{I}) \tag{6.93}$$

Because we know exactly how many of these inputs are in the superposition, we can determine the number of iterations of this diffusion operator are needed to appropriately amplify these inputs:

Definition 6.24:

Let $n \in \mathbb{N}^*$ and $k < n$. Let $N_{\leq k}$ be the number of inputs with Hamming weight of at most k, then:

$$N_{\leq k} = \sum_{j=0}^{k} \binom{n}{j} \tag{6.94}$$

This allows us to define $p_{\leq k}$, the number of iterations for the diffusion operator defined in Definition 6.23:

$$p_{\leq k} = \arg\min_{p \in \mathbb{N}} \left| (2p+1) \arcsin\left(\sqrt{\frac{N_{\leq k}}{2^n}}\right) - \frac{\pi}{2} \right| \tag{6.95}$$

We can now abstract the amplification process into a unitary:

Definition 6.25:

Let $n \in \mathbb{N}^*$ and $k < n$. Let $\mathcal{X}_{\leq k}$ and $\mathcal{X}_\psi$ be the gates defined in Definition 6.23 and $p_{\leq k}$ as in Definition 6.24, then the operator $\mathbf{A}_{\leq k} \in U(2^{n+1})$ defined by:

$$\mathbf{A}_{\leq k} = [\mathcal{X}_\psi(\mathcal{X}_{\leq k} \otimes \mathbf{I})]^{p_{\leq k}} \tag{6.96}$$

Will appropriately amplify the amplitudes of the inputs with Hamming weight of at most k.

Before continuing, we must briefly redefine the diffusion operator introduced in Definition 6.14 for it to be compatible with this additional procedure.

Definition 6.26:

Let $n \in \mathbb{N}^*$ and $k < n$ and $\mathbf{A}_{\leq k}$ as defined in Definition 6.25. We denote:

$$|\psi_{\leq k}(c)\rangle = \mathbf{A}_{\leq k} |\psi(c)\rangle \tag{6.97}$$

From this, we redefine $\mathcal{X}_{\alpha_0}$ as introduced in Definition 6.14 by:

$$\mathcal{X}_{\alpha_0} = (\mathbf{I} - 2 |\psi_{\leq k}(c)\rangle\langle\psi_{\leq k}(c)|) \otimes \mathbf{I} \tag{6.98}$$

The other components remaining the same (with $m_0 = 2$) we then have:

$$\mathbf{Q} = -\mathbf{U}(h)\mathcal{X}_{\alpha_0}(c)\mathbf{U}(h)^\dagger \mathbf{X}_G \tag{6.99}$$

Where $\mathbf{U}(h) = \mathbf{CR}_2\mathbf{T}(h)$.

The circuit used in the learning process is depicted in Figure 6.7.

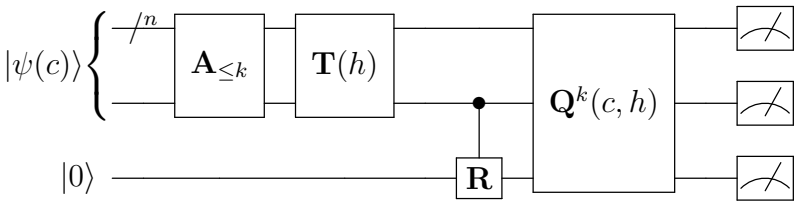


Figure 6.7: Quantum circuit to learn k-juntas

An update phase will thus unfold as follows (to facilitate the explanation, a gate and its controlling input will be conflated). To take advantage of the filter structure, the measurements are separated into correctly classified and misclassified inputs. For each group, the inputs are sorted by increasing Hamming weight and treated in this order. For each of the misclassified inputs, the number of gates to be added, of which the misclassified input is an element of the filter, is counted. If this number is even, this input is added to the list of gates to be updated and all the gates that are in the current network and are also in this input's filter are to be removed. This update process is also performed with the correctly classified inputs but in this case, the number has to be odd in order to add the input to the list of gates to be updated. The update process described above is detailed in Algorithm 9.

One quantity that remains to be determined is the number of queries to the quantum example oracle during such an update phase. While it would have been possible to train the network using the sampling schedule introduced in Section 6.3.2, this approach is best used when no characteristic is known of the target concept. In the case of k-juntas, it seems natural for the query complexity to be a function of k. As a result, when learning k-juntas, we chose to perform 2^k measurements after each amplification phase leading to:

Algorithm 9: Update algorithm for k-juntas

Data: *measurements*, the results of the measurements and *actives*, the list of the network's active gates
Result: *to_update* the list of the network's gates to be updated
to_update $\leftarrow$ [];
errors $\leftarrow$ a list of the measured misclassified inputs where $errors[l]$ is itself the list of the inputs with Hamming weight l;
corrects $\leftarrow$ the same as *errors* but with the correctly classified inputs;
for $0 \leq l \leq n$ **do**
  for $error \in errors[l]$ **do**
    $count \leftarrow 0$;
    for $upd \in to\_update$ **do**
      if $error \in {}_{\uparrow}upd$ **then**
        $count \leftarrow count + 1$;
      end
    end
    if *count is even* **then**
      Add *error* to *to_update*;
      Add the gates in *actives* that are also in ${}_{\uparrow}error$;
    end
  end
  for $correct \in corrects[l]$ **do**
    $count \leftarrow 0$;
    for $upd \in to\_update$ **do**
      if $correct \in {}_{\uparrow}upd$ **then**
        $count \leftarrow count + 1$;
      end
    end
    if *count is odd* **then**
      Add *correct* to *to_update*;
      Add the gates in *actives* that are also in ${}_{\uparrow}correct$;
    end
  end
end

Proposition 6.27:

Let $n \in \mathbb{N}^*$ and $k < n$. Suppose that the concept class is $\mathcal{J}_k^+$. If the number of queries after each amplification phase is 2^k, then during an update phase, the number of queries to the quantum example oracle is in:

$$\Theta(n2^k) \tag{6.100}$$

This leads to the sample complexity of the whole algorithm. Thanks to Property 6.27 we have already established that one update phase will perform $\Theta(n2^k)$ calls to the example oracle. To determine the number of updates, we make the following observation. Let $u_i^{(c)}$ be one of the principals of the target function $c \in \mathcal{J}_k^+$. Suppose further that we update the network with a gate controlled by $v \in {}_\uparrow u_i^{(c)}$. We are assured that during a further update step, we will correct an input $w \in {}_\uparrow u_i^{(c)}$ such that $v \in {}_\uparrow w$. Either because w was measured randomly or because all of the elements of ${}_\uparrow v$ have been corrected beforehand, thus ensuring that w will be measured. However, thanks to the pre-amplification that focuses on the inputs with Hamming weight of at most k, the first event is most likely to happen. Once the gate controlled by w has been updated, the same process will repeat until we reach $u_i^{(c)}$. All in all, we have that the number of updates is in $O(n)$, and hence the query complexity is in:

$$O(n^2 2^k) \tag{6.101}$$

6.4.2 Experimental Results

We have implemented[1] this learning algorithm for $n \in [5 \dots 8]$ and $k \in [2 \dots n-1]$. For each pair of n and k, the network has been trained to learn 16 randomly created k-junta. Each training has been repeated 25 times. The final error rates for these experiments were all similar to what is reported in Figure 6.8. This figure has been plotted using violin plots, hence it shows that all of the training successfully stopped with a correctly tuned network. We now focus on the number of update steps required to reach these results.

[1]The code can be found here: https://github.com/vietphamngoc/exact_AA

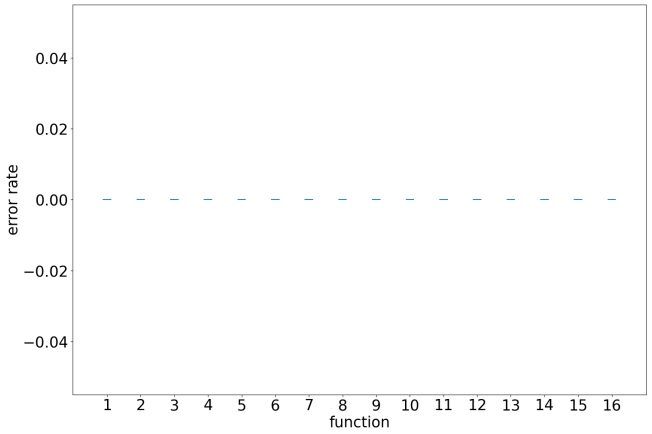
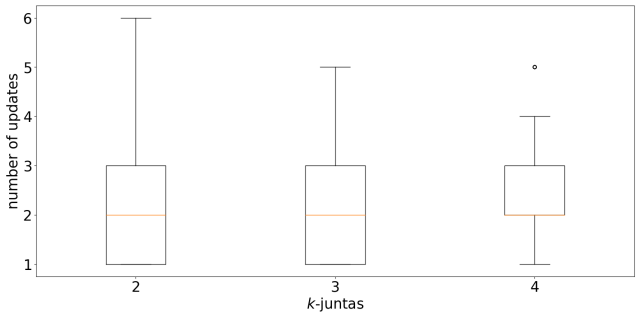
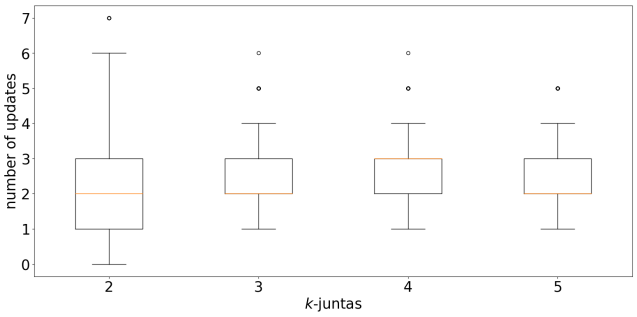


Figure 6.8: Final error rate for all the experiments for each pair (n, k)

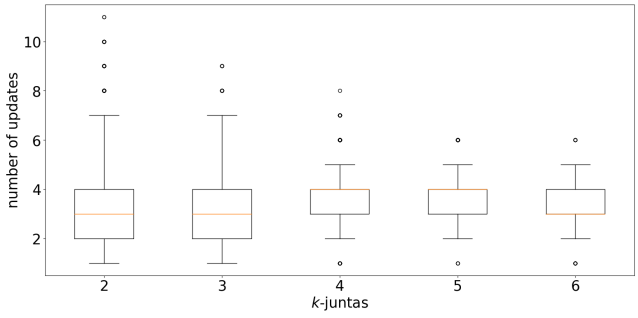
To have an overview of this metric, for a given n and a given k, we have aggregated all of the training runs for all of the functions. For a given n, this set of data has then been plotted against k using box plots. This way we can visualise the median, first and third quartiles, as depicted by the red line, the lower bound, and the upper bound of the box respectively. The maximum and minimum are indicated by the whiskers and the circles represent outliers. All of these are shown in Figure 6.9. From these results, we can see that for a given n, the number of



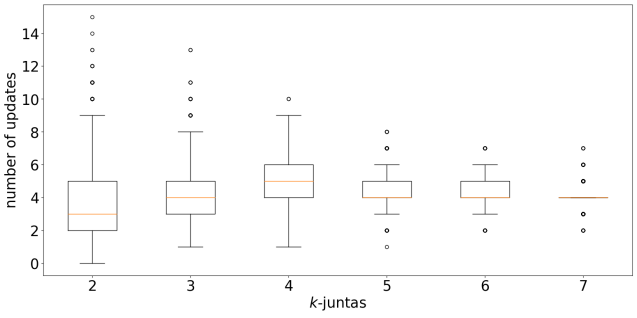
(a) Number of updates to learn k-juntas for $n = 5$ and $k \in [2 \dots 4]$



(b) Number of updates to learn k-juntas for $n = 6$ and $k \in [2 \dots 5]$



(c) Number of updates to learn k-juntas for $n = 7$ and $k \in [2 \dots 6]$



(d) Number of updates to learn k-juntas for $n = 8$ and $k \in [2 \dots 7]$

Figure 6.9: Number of updates for different pairs (n, k)

updates is indeed in the order of n. But more interestingly, it slightly decreases as k increases. This can be explained by the fact that during an update phase, the update algorithm has access to more samples, hence the descent to the principals of the target function is quicker. Another reason comes from the fact that for the algorithm to find a principal when k is small, it will potentially have to go "deeper" as the Hamming weight of a principal is at most k.

However, our upper bound of n still holds. From these experiments, we have verified that the total sample complexity is in $O(n^2 2^k)$. In [8], the task was also to learn k-juntas while given access to a uniform quantum example oracle albeit in the QPAC-learning framework. The complexity of their algorithm was then $O\left(\frac{k}{\epsilon}\log(k)\right)$. Assuming that this algorithm can be applied in the exact learning framework by taking $\epsilon = \frac{1}{2^n}$, we end up with a complexity of $O(2^n k \log(k))$. So in the case where $k \ll n$, our algorithm will perform better.

6.5 Conclusion

In this chapter, we have devised an algorithm to train a tunable quantum neural network in the exact learning framework with access to a uniform quantum example oracle. We refined it by employing it to learn generic Boolean functions and by comparing it to a naive algorithm. We then adapted this algorithm to learn the class of k-juntas. Following the implementation of this approach, we found that the query complexity is in $O(n^2 2^k)$. This complexity is lower than what can be found in the literature in the case where $k \ll n$.

Chapter 7

Conclusion

7.1 Summary of Thesis Achievements

As this PhD comes to an end, this section represents an opportunity to reflect on what has been accomplished during this time.

In Chapter 3, we have introduced a learning architecture that is simple in the sense that it is exclusively made of $\mathbf{X}$ gates that are controlled by subsets of the input qubits and acting on the ancillary qubit. This architecture presents the particularity that the entanglement only occurs between the set of input qubits and the ancillary qubit. We then provided a formal proof that this architecture can express any Boolean function. We called it a tunable quantum neural network as, if tuned correctly, this architecture can learn any Boolean functions.

In that optic, in Chapter 4, we devised an algorithm that channels the correspondence between quantum amplitude and measurement probability to identify misclassified inputs. This scheme allows for the network to be correctly tuned in at most $n+1$ updates when learning a function of n variables but at the cost of 2^{2^n} samples per update. We implemented this approach and while it worked as intended, the high number of samples became quickly limiting.

To tackle this limitation, in Chapter 5, we designed an algorithm that leverages the quantum amplitude amplification procedure to boost the probability of measuring misclassified inputs.

This algorithm was designed to train the TNN in the QPAC-learning framework. We tested the algorithm by implementing it and trained the network to learn the class of parity functions. The experiments confirmed that the query complexity was in $\Omega\left(\frac{n-1}{\epsilon} + \log\left(\frac{1}{\epsilon}\right)\frac{1}{\delta^2}\right)$. This represents an exponential slowdown in terms of $\frac{1}{\delta}$ but an exponential speedup in terms of $\frac{1}{\epsilon}$ when compared to the literature. These results indicate that this algorithm performs better for applications where a small ϵ is required.

Finally, in Chapter 6, we conducted further investigations regarding the network's capacity by studying it in the exact learning framework with access to a uniform quantum example oracle. Here again, the amplitude amplification procedure was used. In order to completely parameterise the algorithm, we compared it against a naive algorithm on the task of learning a generic Boolean function. We then adapted it to learn the class of k-juntas and implemented this approach. From the experiments, we concluded that the sampling complexity was in $O\left(n^2 2^k\right)$. This means that compared to the literature, this algorithm performs better in the case where $k \ll n$.

7.2 Limitations and Future Work

While this research can be of interest, we note that our approach does suffer from some limitations, which we describe in the following paragraphs.

The first one concerns the tunable quantum neural network itself. Indeed, while it was interesting to study and surprising that such a simple architecture could span the richness of the Boolean functions, it may be restricted to theoretical studies, with limited practical use.

The second limitation relates to the $\frac{1}{\delta}$ dependent part of the complexity of the algorithm in the QPAC-learning framework. From the experimental results, it seems that a reduction in the dependence in $\frac{1}{\delta}$ is possible. Moreover, in our study, we only trained the network to learn a single concepts class and it would have been interesting to verify the network's performance on other classes.

Similarly, this last limitation also applies to our work on the network in the Exact learning framework. Finally, the justification for the number of updates lacks a formal proof.

Considering these limitations, we propose the following as possible future work:

1. Find a tighter bound in terms of $\frac{1}{\delta}$ for the QPAC-learning algorithm and test it on other concept classes.

2. Give a more formal proof for the number of updates for the Exact learning algorithm and also test it on other concept classes. Indeed, the argument invoking the filters of the target function's principals together with the use of a prior amplitude amplification motivated our conjecture that the number of updates would be in $O(n)$. While we verified this conjecture with experiments, we did not manage to provide a formal proof for the number of updates.

3. Study this architecture in the presence of classification noise.

Bibliography

[1] S. Aaronson and P. Rall. Quantum approximate counting, simplified. *Symposium on Simplicity in Algorithms*, page 24–32, Jan 2020.

[2] D. Angluin. Queries and concept learning. *Machine Learning*, 2, 1988.

[3] S. Arunachalam, S. Chakraborty, T. L., M. Paraashar, and R. de Wolf. Two new results about quantum exact learning. *Quantum*, 5, 2021.

[4] S. Arunachalam and R. de Wolf. Guest column: A survey of quantum learning theory. *ACM SIGACT News*, 48(2):41–67, 2017.

[5] S. Arunachalam, V. Gheorghiu, T. Jochym-O'Connor, M. Mosca, and P. V. Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 12 2015.

[6] S. Arunachalam, A. B. Grilo, and H. Yuen. Quantum statistical query learning, 2020.

[7] S. Arunachalam and R. D. Wolf. Optimal quantum sample complexity of learning algorithms. *Journal of Machine Learning Research*, 19, 2018.

[8] A. Atıcı and R. A. Servedio. Quantum algorithms for learning and testing juntas. *Quantum Information Processing*, 6, 2007.

[9] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical review. A, Atomic, molecular, and optical physics*, 52(5):3457–3467, 1995.

[10] P. Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22, 1980.

[11] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26, 1997.

[12] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd. Quantum machine learning. *Nature*, 549:195–202, 9 2017.

[13] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36, 1989.

[14] Y. I. Bogdanov, N. A. Bogdanova, D. V. Fastovets, and V. F. Lukichev. Representation of boolean functions in terms of quantum computation. In V. F. Lukichev and K. V. Rudenko, editors, *International Conference on Micro- and Nano-Electronics 2018*, volume 11022. International Society for Optics and Photonics; SPIE, 2019.

[15] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. *Quantum Amplitude Amplification and Estimation*, volume 305 of *Quantum computation and information*, pages 53–74. Amer. Math. Soc., Providence, RI, 2002.

[16] N. Bshouty and J. Jackson. Learning dnf over the uniform distribution using a quantum example oracle. *SIAM Journal on Computing*, 28(3):1136–1153, 1998.

[17] N. H. Bshouty, J. C. Jackson, and C. Tamon. Exploring learnability between exact and pac. *Journal of Computer and System Sciences*, 70, 2005.

[18] C. Y. Chen. An exact quantum polynomial-time algorithm for solving k-junta problem with one uncomplemented product. *International Journal of Theoretical Physics*, 61, 2022.

[19] J.-J. Climent, F. J. García, and . V. Requena. The degree of a boolean function and some algebraic properties of its support. In *WIT Transactions on Information and Communication Technologies*, volume 45, 2013.

[20] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, Ltd, 2005.

[21] H. S. M. Coxeter and W. O. J. Moser. *Generators and Relations for Discrete Groups*. Springer, 1972.

[22] P. L. Dallaire-Demers and N. Killoran. Quantum generative adversarial networks. *Physical Review A*, 7 2018.

[23] D. Dervovic, M. Herbster, P. Mountney, S. Severini, N. Usher, and L. Wossnig. Quantum linear systems algorithms: a primer. *arXiv e-print*, 2 2018.

[24] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439, 1992.

[25] N. Ding. On exactly learning disjunctions and dnfs without equivalence queries. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11653 LNCS, 2019.

[26] V. Dunjko and H. J. Briegel. Machine learning & artificial intelligence in the quantum domain. *arXiv e-print*, 9 2017.

[27] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21, 1982.

[28] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39, 1992.

[29] D. F. Floess, E. Andersson, and M. Hillery. Quantum algorithms for testing boolean functions. *Electronic Proceedings in Theoretical Computer Science*, 26, 2010.

[30] V. Giovannetti, S. Lloyd, and L. Maccone. Architectures for a quantum random access memory. *Physical Review A*, 12 2008.

[31] V. Giovannetti, S. Lloyd, and L. Maccone. Quantum random access memory. *Physical Review Letters*, 100:160501, 4 2008.

[32] A. B. Grilo, I. Kerenidis, and T. Zijlstra. Learning-with-errors problem is easy with quantum samples. *Physical Review A*, 99, 2019.

[33] D. Grinko, J. Gacon, C. Zoufal, and S. Woerner. Iterative quantum amplitude estimation. *npj Quantum Information*, 7, 2021.

[34] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[35] J. Haah, A. W. Harrow, Z. Ji, X. Wu, and N. Yu. Sample-optimal tomography of quantum states. *IEEE Transactions on Information Theory*, 63(9):5628–5641, 2017.

[36] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[37] A. W. Harrow, A. H., and S. Lloyd. Quantum algorithm for solving linear systems of equations. *Physical Review Letters*, 11 2008.

[38] I. Haviv and O. Regev. The list-decoding size of fourier-sparse boolean functions. *ACM Transactions on Computation Theory*, 8, 2016.

[39] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December, 2016.

[40] W. Hu. Comparison of two quantum nearest neighbor classifiers on ibm's quantum simulator. *Natural Science*, 10:87–98, 2018.

[41] W. Huggins, P. Patil, B. Mitchell, K. B. Whaley, and E. M. Stoudenmire. Towards quantum machine learning with tensor networks. *Quantum Science and Technology*, 4(2):024001, jan 2019.

[42] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing cnot-based quantum circuits. In *Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324)*, pages 419–424, 2002.

[43] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45, 1998.

[44] I. Kerenidis, J. Landman, and A. Prakash. Quantum algorithms for deep convolutional neural networks. *arXiv e-print*, 11 2019.

[45] S. Kurgalin and S. Borzunov. *The Discrete Math Workbook*. Springer International Publishing, 2018.

[46] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.

[47] Y. Levine, O. Sharir, N. Cohen, and A. Shashua. Quantum entanglement in deep learning architectures. *Physical Review Letters*, 122, 2 2019.

[48] J. Li, X. Yang, X. Peng, and C. Sun. Hybrid quantum-classical approach to quantum optimal control. *Physical Review Letters*, 8 2016.

[49] S. Lloyd, M. Mohseni, and P. Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arViv e-print*, 2013.

[50] S. Lloyd, M. Mohseni, and P. Rebentrost. Quantum principal component analysis. *Nature Physics*, 10:631–633, 2014.

[51] S. Lloyd and C. Weedbrook. Quantum generative adversarial learning. *Physical Review Letters*, 7 2018.

[52] A. Montanaro. Quantum speedup of monte carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471, 2015.

[53] S. Morita and H. Nishimori. Mathematical foundation of quantum annealing. *arXiv e-print*, 6 2008.

[54] E. Mossel, R. O'Donnell, and R. A. Servedio. Learning juntas. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 2003.

[55] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[56] K. Nakaji. Faster amplitude estimation. *arXiv e-print*, 2020.

[57] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[58] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. G. H. Ong, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh. Can fpgas beat gpus in accelerating next-generation deep neural networks? *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[59] L. I. Panchi and Y. Zhao. Model and algorithm of sequence-based quantum-inspired neural networks. *Chinese Journal of Electronics*, 27:9–18, 1 2018.

[60] M. Panella and G. Martinelli. Neural networks with quantum architecture and quantum learning. *International Journal of Circuit Theory and Applications*, 39:61–77, 1 2011.

[61] A. Peruzzo, J. McClean, P. Shadbolt, M. H. Yung, X. Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien. A variational eigenvalue solver on a quantum processor. *Nature Communications*, 4 2013.

[62] V. Pham Ngoc, D. Tuckey, and H. Wiklicky. Tunable quantum neural networks in the qpac-learning framework. *arXiv e-print*, 2023.

[63] V. Pham Ngoc and H. Wiklicky. Tunable quantum neural networks for boolean functions. *arXiv e-print*, 2020.

[64] P. Rebentrost, T. R. Bromley, C. Weedbrook, and S. Lloyd. Quantum hopfield neural network. *Physical Review A*, 98(4), 2018.

[65] P. Rebentrost, M. Mohseni, and S. Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 9 2014.

[66] Y. Ruan, X. Xue, H. Liu, J. Tan, and X. Li. Quantum algorithm for k-nearest neighbors classification based on the metric of hamming distance. *International Journal of Theoretical Physics*, 56:3496–3507, 11 2017.

[67] J. E. Savage. *Models of Computation*. Addison-Wesley, Reading, Mass. [u.a.], reprinted with corr. edition, 2000.

[68] M. Schuld, I. Sinayskiy, and F. Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[69] M. Schuld, I. Sinayskiy, and F. Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56:172–185, 4 2015.

[70] L. Sellie. Exact learning of random dnf over the uniform distribution. *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2009.

[71] R. A. Servedio and S. J. Gortler. Equivalences and separations between quantum and classical learnability. *SIAM Journal on Computing*, 33, 2004.

[72] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, 2003.

[73] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 1994.

[74] F. Tacchino, C. Macchiavello, D. Gerace, and D. Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):1–8, 2019.

[75] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso. The computational limits of deep learning. *arXiv e-print*, 2022.

[76] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[77] V. N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10, 1999.

[78] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16, 1971.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December, 2017.

[80] S. Wallis. Binomial confidence intervals and contingency tests: Mathematical fundamentals and the evaluation of alternative methods. *Journal of Quantitative Linguistics*, 20(3):178–208, 2013.

[81] K. H. Wan, O. Dahlsten, H. Kristjánsson, R. G., and M. S. Kim. Quantum generalisation of feedforward neural networks. *npj Quantum Information*, 3, 12 2017.

[82] C. R. Wie. Simpler quantum counting. *Quantum Information and Computation*, 19, 2019.

[83] N. Wiebe, A. Kapoor, and K. Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv e-print*, 1 2014.

[84] N. Wiebe, A. Kapoor, and K. M. Svore. Quantum deep learning. *arXiv e-print*, 12 2014.

[85] P. Wittek. *Quantum Machine Learning: What Quantum Computing Means to Data Mining*. Elsevier, 2014.

[86] S. Yoo, J. Bang, C. Lee, and J. Lee. A quantum speedup in machine learning: Finding a n-bit boolean function for a classification. *New Journal of Physics*, 10 2014.

[87] A. Younes and J. Miller. Automated method for building cnot based quantum circuits for boolean functions. *arXiv e-prints*, pages quant–ph/0304099, 2003.

[88] A. Younes and J. F. Miller. Representation of boolean quantum circuits as reed-muller expansions. *International Journal of Electronics*, 91(7):431–444, 2004.

Index