Universidad ORT Uruguay

Facultad de Ingeniería

Automatic sythesis of quantum circuits

With a quantum machine learning application

Entregado como requisito para la obtención del título de Master en Ingeniería

Carolina Allende Amen - 233383

Tutores

André Fonseca De Oliveira

Efrain Buksman Hollander

2023

.

Declaración de autoría

Yo, Carolina Allende Amen, declaro que el trabajo que se presenta en esa obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba el desarrollo de tesis del Master en Ingeniería;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mi;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Carolina Allende

01 de Marzo de 2023

To Julieta Carricondo

Acknowledgements

I would like to thank the best team mates I could have hoped for in this endeavour, Ignacio Aristimuño, Julieta Carricondo, Juan Michelini and Ariel Mordetzki for your relentless support and my colleagues, the awesome Marvik team, for your companionship

I would also like to thank Efrain Buksman, André Fonseca, Ilan Cohn and for your continued guidance thorughout these 4 years in the quantum realm

Abstract

English

This work introduces the use of a variational quantum circuit in order to perform unitary decomposition of a quantum algorithm. By use of classical optimization techniques and exploiting correlations and entanglement the variational quantum circuit is able to translate a wide range of quantum algorithms. A case study is also presented, where this decomposition was used to make a classifier over a quantum encoded version of a well known dataset.

Español

Este trabajo presenta un circuito variacional cuántico como forma de lograr la descomposición unitaria de un algoritmo cuántico. Usando técnicas de optimización clásicas y explotando propiedades cuánticas como el entrelazamiento y su relación con la correlación este circuito variacional puede traducir un gran número de algoritmos cuánticos. Además, se presenta un caso de estudio donde esta descomposición se utiliza para crear un clasificador para un dataset conocido, donde los datos clásicos se codifican en estados cuánticos.

Keywords

Unitary decomposition, Variational Quantum Circuits, Variational Quantum Classifier

Index

Introduction 9

Chapter 1: Theoretical overview

1.1 Postulates 11

1.2 First postulate: The qubit 12

1.3 Second postulate: How qubits evolve 14

1.4 Third postulate: How measurements affect qubits 16

1.5 Postulate 4: How qubits combine 18

1.6 Quantum entanglement 19

1.7 Density matrices 21

1.8 Partial trace operator 22

1.9 Universal quantum gates 23

1.10 Fidelity and correlation 24

1.10.1 Fidelity 24

1.10.2 Correlation 24

Chapter 2: Unitary decomposition

2.1 Introduction 26

2.2 Cartan KAK decomposition 26

2.3 Other approaches 27

2.4 Variational Quantum Circuits 28

2.4.1 Introduction 28

2.4.2 Data encoding: Amplitude encoding 19

2.4.3 Interpreting class labels 30

Chapter 3: Quantum circuit synthesis

3.1 Introduction 31

3.2 The basic ansatz 31

3.3 Correlation based learning 32

3.3.1.a Determining the number of C_{NOTs} 33

3.3.1.b Determining C_{NOT} placement 33

3.4 Single qubit matrices 36

3.4.1 Optimization 38

3.4.2 Hyperparameters 39

3.5 Training the whole network 40

Chapter 4: Use case and application

4.1 Use case: The Toffoli gate 42

4.2 Machine learning application: Hybrid classifier 44

4.2.1 Introduction 44

4.2.2 IRIS dataset: Preprocessing and feature encoding 44

4.2.3 Training the classifier and interpreting classifications 49

Chapter 5: Discussion

5.1 Discussion 53

5.1.1 Toffoli Gate 53

5.1.2 The hybrid classifier 53

5.2 Conclusions 54

References 55

Anex I: User manual

Auxiliary functions

Functions that deal with correlation

Functions that deal with unitary matrices

Functions that deal with blocks

Anex II: Code

Introduction

Quantum computing was born in the late 1980s and promises to solve problems which classical computing cannot tackle efficiently thanks to harnessing quantum properties [1]. Through the years a few algorithms have surfaces which exploit these quantum properties [2] [3] [4], but quantum computing is still very much in its infancy for a number of reasons, from needing temperatures near absolute zero to work to it being expensive and difficult to build suitable hardware.

Another of the main challenges in this area today is the difficulty faced when building new algorithms that solve standard classical problems given the programming paradigm is completely different from the classical one. Part of this problem is tackled in this work: unitary decomposition. The current state of the art still requires algorithms to be built from quantum gates, very much like digital electronics. Any feasible algorithm in this realm is given by a unitary matrix and the task of translating these matrices to a cascade of quantum universal gates is known as unitary decomposition. In order to be able to run a given algorithm, one needs to decompose it into gates and there are a number of possible solutions which are overviewed in this piece of research.

Using quantum machine learning and a set of universal quantum gates, we build a variational quantum circuit capable of approximating a wide range of unitary matrices which improves upon our past work in the field. This variational circuit learns on the one hand from classical optimization methods but also draws from quantum entanglement when fitting a given algorithm.

The obtained results are promising, with the circuit being able to decompose a wide range of algorithms including the Toffoli gate. Furthermore, a proof of concept is presented in which a variational quantum circuit is used to implement a hybrid classifier over classical data.

Chapter 1: Theoretical overview

1.1 Postulates

Quantum computing is based on four postulates.

- Postulate 1: Definition of a quantum bit, or qubit
- Postulate 2: How qubits evolve
- Postulate 3: How measurements affect qubits
- Postulate 4: How qubits combine

1.2 First postulate: The qubit

First postulate:

"Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the state space of the system. The system is completely described by its state vector, which is a unit vector in the system's state space." [5]

The basic classical unit used to represent data is the bit and classical memories are no more than arrays of them. The classical bit has it's quantum analog, the qubit, which can be seen as an interpretation of the spin of an electron which. The most common notation used in quantum computing is Dirac's notation, which is merely a different way to denote vectors in a Hilbert space, which are the state spaces of quantum systems. For instance, the one-qubit $|0\rangle$ state is given by:

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

While the one-qubit $|1\rangle$ state is given by:

$$|1\rangle = \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix} \tag{2}$$

These two states are one of many basis of the one-qubit state space, that is, any one qubit state can be written as a linear combination of these two vectors, which are known as superposition states. Superpositions are a core difference between quantum and classical computing, since:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{3}$$

where α, $\beta \in \mathscr{C}$are known as amplitudes, is a feasible quantum state.

Classical computing allows data to be determined in a precise manner, that is, data does not depend on who reads a memory. In quantum computing, generally, one cannot retrieve the exact quantum state in which a system is: when a measurement is performed superposition is destroyed and one of the basis states is produced with a certain probability. For example, in the case of $|\Psi\rangle$, $|0\rangle$ can be observed with a probability of $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Nevertheless, generally, it is impossible to know these amplitudes and hence, impossible to know in which precise state a qubit is. Since $|\alpha|^2$ and $|\beta|^2$ are probabilities, $|\Psi\rangle$ must be a unitary vector and satisfy:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{4}$$

Furthermore, there are notable operations in Hilbert spaces, inner and outer products, which can be denoted in Dirac's notation. The inner product remains unchanged from classical linear algebra, and is noted as $\langle\Psi|\Psi\rangle$, which yields the square of the norm of $|\Psi\rangle$. Meanwhile, the outer product of two vectors is given by $|\Psi\rangle\langle\Psi|$ and yields a matrix:

$$|\Psi\rangle\langle\Psi| = \begin{bmatrix} \alpha \\ \\ \beta \end{bmatrix} \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix} = \begin{bmatrix} |\alpha|^2 & \alpha\beta^* \\ \\ \beta\alpha^* & |\beta|^2 \end{bmatrix} \tag{5}$$

Each element (i, j) of the resulting matrix is obtained multiplying the $i - th$ component of the column vector by the conjugate of the $j - th$ element of the row vector.

1.3 Second postulate: How qubits evolve

The second postulate:

"The evolution of a ***closed*** *quantum system is described by a unitary transformation U. That is, the state $|\Psi\rangle$ $of the system at time t_1$ is related to the state $|\Psi'\rangle$ $of the system at time t_2$ by a unitary operator U which depends only on the times $t1$ and t_2* [6]

$$|\Psi'\rangle = \mathbf{U}\,|\Psi\rangle \tag{6}$$

Even though it is not straightforward to find the physical system associated to a specific unitary transformation of interest, the postulates of quantum mechanics guarantee the matrix exists and is unitary. It must be a unitary transformation to preserve the norm of quantum states, which can never exceed 1 since they result in a sum of probabilities. In the special case of one qubit systems, there are certain notable transformations known as the Pauli matrices which can be generalized to multiqubit systems. These matrices are given by equations 7 through 10.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{8}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{9}$$

$$Z = \begin{bmatrix} 1 & 0 \\ \\ 0 & -1 \end{bmatrix} \tag{10}$$

When applied to a generic superposition state:

$$I\,|\Psi\rangle = \begin{bmatrix} 1 & 0 \\ \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ \\ \beta \end{bmatrix} \tag{11}$$

Where I is the identity operator which does not modify the input amplitudes.

$$X\,|\Psi\rangle = \begin{bmatrix} 0 & 1 \\ \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \\ \alpha \end{bmatrix} \tag{12}$$

X is the quantum analog of the classical**NOT**, since it swaps input amplitudes between basis states.

$$Z\,|\Psi\rangle = \begin{bmatrix} 1 & 0 \\ \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ \\ -\beta \end{bmatrix} \tag{13}$$

Z is known as *Phase Flip*, since in introduces a relative phase around an axis.

$$Y\,|\Psi\rangle = \begin{bmatrix} 0 & -i \\ \\ i & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \\ \beta \end{bmatrix} = \begin{bmatrix} -i\beta \\ \\ i\alpha \end{bmatrix} \tag{14}$$

These four operators are known as the Pauli operators but there are several more. Another important operator which will later be useful in the following chapters: the C_{NOT} gate. This operator takes a two-qubit state as an input and swaps the second qubit if the first one is set to $|1\rangle$

$$C_{NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{15}$$

1.4 Third postulate: How measurements affect qubits

The third postulate:

"Quantum measurements are described by a collection M_m of measurement operators. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment." [7]

Let the state of a quantum system be $|\Psi\rangle$, then the probability of the outcome of a measurement being m is given by the inner product:

$$p(m) = \langle\Psi|M_m^\dagger M_m|\Psi\rangle \tag{16}$$

And the resulting state after measuring is given by:

$$\frac{M_m\,|\Psi\rangle}{\sqrt{\langle\Psi|M_m^\dagger M_m|\Psi\rangle}} \tag{17}$$

As aforementioned, the probability of a measurement yielding the result m is given by equation 17, and thus must satisfy

$$1 = \sum_m p(m) = \sum_m \langle\Psi|M_m^\dagger M_m|\Psi\rangle \tag{18}$$

And in order for equation 19 to hold, the operators M_m must satisfy:

$$\sum_m M_m^\dagger M_m = I \tag{19}$$

For further clarification, we will now analyse the possible outcomes of measuring state Ψ defined previously in equation 3. In this case there are only two basis states and therefore, the measurement operators are given by $M_0 = |0\rangle\langle 0|$ y $M_1 = |1\rangle\langle 1|$:

$$M_0 = \begin{bmatrix} 1 & 0 \\ \\ 0 & 0 \end{bmatrix} M_1 = \begin{bmatrix} 0 & 0 \\ \\ 0 & 1 \end{bmatrix} \tag{20}$$

Given this operators satisfy equation 20 and the result of squaring each matrix is the same matrix, the probability of measuring 0 noted $p(0)$ can be calculated:

$$p(0) = \langle\Psi|M_0^\dagger M_0|\Psi\rangle = \langle\Psi|M_0|\Psi\rangle \tag{21}$$

$$p(0) = |\alpha|^2 \tag{22}$$

And the resulting state:

$$\frac{M_0|\Psi\rangle}{|\alpha|} = \frac{\alpha|0\rangle}{|\alpha|} \tag{23}$$

The probability of getting 1 and the resulting state after measurement can be computed in an manner analog to the one above:

$$p(0) = |\alpha|^2 \tag{24}$$

$$p(1) = |\beta|^2 \tag{25}$$

As long as the quantum state is not measured it is made up of a superposition of basis states. After measuring the superposition is broken and the resulting state is one of the basis states of the measuring operator involved.

1.5 Fourth postulate: How qubits combine

The fourth postulate, where $\otimes$ notes outer product:

" The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number i is prepared in the state $|\Psi_i\rangle$*, then the joint state of the total system is* $|\Psi_1\rangle \otimes |\Psi_2\rangle ... \otimes |\Psi_n\rangle$*"* [8]

Given the example two qubit state $|\Psi\rangle = |\psi\rangle \otimes |\phi\rangle$ y $|\phi\rangle = |\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, the state space described by them $|\psi\rangle \otimes |\phi\rangle$ would simply be:

$$|\psi\rangle \otimes |\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{26}$$

$$|\psi\rangle \otimes |\phi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \tag{27}$$

1.6 Quantum entanglement

Quantum entanglement is the physical phenomena which can be observed when a system of particles interact in such a manner that the quantum state of each single particle cannot be described independently from the quantum states of the other particles in the system, not even if they are separated a long distance. It's nature is purely quantum since this phenomena is not present in classical physics, and to illustrate this quantum property, quantum teleportation will be analyzed. Quantum teleportation is a technique used in order to send quantum states long distances without sending the information directly through a quantum channel which would link the state sent to its sender.

Two scientists, Rosalind and Marie, worked together and created a pair of entangled qubits but now live in opposite sides of the world and each took one of the entangled pair with them. Years later, Rosalind wants to send a quantum state $|\Psi\rangle$ but does not know the mathematical formulation of the state she needs to send and can only send information through a classical channel. Rosalind cannot determine the state $|\Psi\rangle$ given that she only has a single copy of it and even if she could, describing it would take an infinite amount of classical information. But she has her half of the entangled state which she can have interact with the state she wants to send and then measure both qubits.

After measuring, Rosalind gets one of four possible results: 00, 01, 10 o 11. She then can send that information to Marie via a classical channel and depending on this information Marie operates over her pair of the entangled pair in order to get $|\Psi\rangle$. Figure 1 illustrates the full process.

State $|\beta_{00}\rangle$ is the original entangled pair, where Rosalind keeps the first qubit and Marie the second. Let $|\Psi\rangle{=}\alpha\,|0\rangle{+}\beta\,|1\rangle$:

$$|\Psi_0\rangle = |\Psi\rangle \otimes |\beta_{00}\rangle \tag{28}$$

$$|\Psi_0\rangle = \frac{1}{\sqrt{2}}[\alpha\,|0\rangle\,(|00\rangle + |11\rangle) + \beta\,|1\rangle\,(|00\rangle + |11\rangle)] \tag{29}$$

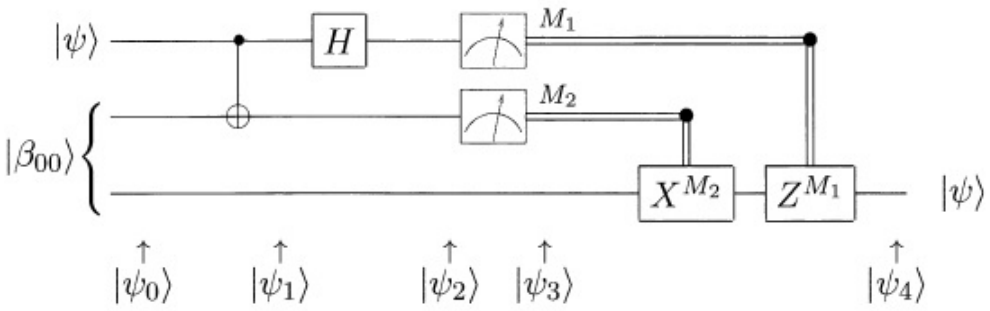

Figure 1: Quantum teleportation [9]

$|\Psi_1\rangle$, after the C_{NOT} gate:

$$|\Psi_1\rangle = \frac{1}{\sqrt{2}}[\alpha\,|0\rangle\,(|00\rangle + |11\rangle) + \beta\,|1\rangle\,(|10\rangle + |01\rangle)] \tag{30}$$

Analogly, the Hadamard get affects Rosalind's qubit in the entangled pair

$$|\Psi_2\rangle = \frac{1}{\sqrt{2}}[\alpha(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + \beta(|0\rangle - |1\rangle)(|10\rangle + |01\rangle)] \tag{31}$$

Rewriting $|\Psi_2\rangle$

$$|\Psi_2\rangle = \frac{1}{\sqrt{2}}[|00\rangle\,(\alpha\,|0\rangle + \beta\,|1\rangle) + |01\rangle\,(\alpha\,|1\rangle + \beta\,|0\rangle) \tag{32}$$

$$+ |10\rangle\,(\alpha\,|0\rangle - \beta\,|1\rangle) + |11\rangle\,(\alpha\,|1\rangle - \beta\,|0\rangle)]$$

Measuring Rosalind's qubit of the entangled pair yields:

$$|\Psi_3(00)\rangle \Rightarrow [\alpha\,|0\rangle + \beta\,|1\rangle] \tag{33}$$

$$|\Psi_3(01)\rangle \Rightarrow [\alpha\,|1\rangle + \beta\,|0\rangle] \tag{34}$$

$$|\Psi_3(10)\rangle \Rightarrow [\alpha\,|0\rangle - \beta\,|1\rangle] \tag{35}$$

$$|\Psi_3(11)\rangle \Rightarrow [\alpha\,|1\rangle - \beta\,|0\rangle] \tag{36}$$

Where $|\Psi_3(ii)\rangle$ is the result obtained after measuring $|\Psi_3\rangle$.

Rosalind lets Marie know the result of the measurement and Marie operates over her qubit. If the measurement yields $|00\rangle$, then Marie already has state $|\Psi\rangle$. On the other hand, if it yields $|01\rangle$ Marie must apply an X gate to obtain state $|\Psi\rangle$, the other cases can be solved in a similar manner and Marie can get the state Rosalind meant to send in the first place.

1.7 Density matrices

So far only vector notation of quantum states has been introduced in section 1.2. Unfortunately, this representation is not applicable to all quantum states given that in some cases one does not know full quantum system involved, rather than that, one can only know a subspace of the system. This is the case with open quantum systems that interact with the environment, and the states that describe these systems are denominated mixed states and can be described by the density matrix operator. The density matrix is an operator in a Hilbert space given by:

$$\rho = \sum_i p_j \left|\Psi_j\right\rangle \left\langle\Psi_j\right| \tag{37}$$

$$0 \leq p_j \leq 1 \tag{38}$$

$$\sum_j p_j = 1 \tag{39}$$

p_j is the element of a matrix that weighs the contribution of the state $|\Psi_j\rangle$ in the state of the complete system. In this manner a statistical mixture of pure states can be represented.

The expected value of observable A over t this system is given by 40 y 41 where Tr is the partial trace operator which will be explained in the following section.

$$< A >= \sum_j p_j \left\langle\Psi_j\right|A\left|\Psi_j\right\rangle = \sum_j p_j Tr(\left|\Psi_j\right\rangle \left\langle\Psi_j\right| A) = \sum_j Tr(p_j \left|\Psi_j\right\rangle \left\langle\Psi_j\right| A) \tag{40}$$

$$< A >= Tr(\sum_j p_j \left|\Psi_j\right\rangle \left\langle\Psi_j\right| A) = Tr(\rho A) \tag{41}$$

Density matrices satisfy:

- $Tr(\rho)$=1
- ρ=$\rho^{\dagger}$
- $\langle \Psi \rho \Psi \rangle > 0 \ \forall \ \Psi$

When the state of a quantum system can be written in vector form it is known as a pure state and its density matrix can be easily calculated: $\rho_{\Psi} = |\Psi\rangle \langle \Psi|$

1.8 Partial trace operator

The partial trace operator maps density matrices ρ_{AB} in the composite space $\mathscr{H}_A \otimes \mathscr{H}_B$ to density matrices ρ_A in $\mathscr{H}_A$ and ρ_B in $\mathscr{H}_B$.

$$Tr_B : S \otimes T \rightarrow Tr(T)S \tag{42}$$

For any S in $\mathscr{H}_A$ and T in $\mathscr{H}_B$, and let $[|a_i\rangle]$ be a base $\mathscr{H}_A$ and $[|b_i\rangle]$ a base of $\mathscr{H}_B$. Any matrix ρ_{AB} in $\mathscr{H}_A \otimes \mathscr{H}_B$ can be decomposed as $\rho_{AB} = \sum_{ijkl} c_{ijkl} |a_i\rangle \langle a_j| \otimes |b_k\rangle \langle b_l|$ and its partial trace with respect to B:

$$Tr_B(\rho_{AB}) = \sum_{ijkl} c_{ijkl} |a_i\rangle \langle a_j| \otimes \langle b_k| |b_l\rangle \tag{43}$$

Which is a matrix ρ_A en $\mathscr{H}_A$.
A generic two qubit state can be written as a combination of the basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$:

$$\rho_{AB} = \rho_{11} |00\rangle \langle 00| + \rho_{12} |00\rangle \langle 01| + \rho_{13} |00\rangle \langle 10| + ... + \rho_{44} |11\rangle \langle 11| \tag{44}$$

The partial trace with respect to B is known as reduced matrix ρ_A of the subspace $\mathscr{H}_A$ y and can be calculated (43):

$$\rho_A = (\rho_{11} + \rho_{22}) |0\rangle \langle 0| + (\rho_{13} + \rho_{24}) |0\rangle \langle 1| + (\rho_{31} + \rho_{42}) |1\rangle \langle 0| + (\rho_{33} + \rho_{44}) |1\rangle \langle 1| \tag{45}$$

1.9 Universal quantum gates

In classical computation any binary operation can be written as a finite linear combination of the logic gates AND, OR and NOT. Furthermore, any binary operation can be executed using only $NAND$ gates since the other aforementioned gates can be computed using $NAND$ gates. The sets AND, OR, NOT and $NAND$ are sets of universal gates in classical computing, given that any other operation desired can be made by them.

In a similar manner, there are universal sets of quantum gates. An important theoretical conclusion is the Solovay-Kitaev theorem [10]. This theorem proves that given a set $\mathscr{S}$ of universal quantum gates any quantum operation given by a finite number m of quantum gates can by aproximate with an error ε by a quantum circuit with order $O(m \log(\frac{m}{\varepsilon}))$ made solely of elements of $\mathscr{S}$. As a result, quantum computers need only implement a finite number of quantum gates in order to be able to implement any unitary transformation.

Particularly, the set made up by arbitrary one qubit gates and the C_{NOT} gate given by equation (15) is a universal quantum set of gates [11] and will be further discussed in Chapter 3.

1.10 Fidelity and correlation

1.10.1 Fidelity

A key part of researching in quantum computing consists of comparing an imperfect experimental result against the theoretical result the experiment is mean to yield. In order to do so a number of different distances can be computed, but the chosen one for this piece of research stems from quantum fidelity.

Let ρ and γ be two quantum states, the fidelity between them is given by:

$$Fid(\rho, \gamma) \coloneqq Tr\sqrt{\rho^{\frac{1}{2}}\gamma\rho^{\frac{1}{2}}} \tag{46}$$

And the distance which stems from it is then given by:

$$D_{Fid}(\rho, \gamma) \coloneqq \arccos\left(Fid(\rho, \gamma)\right) \tag{47}$$

Moreover, fidelity satisfies:

- $0 \geq Fid(\rho, \gamma) \geq 1$
- $Fid(\rho, \gamma) = Fid(\gamma, \rho)$,
- If ρ (o γ) is a pure state, $\rho = |\psi\rangle \langle\psi|$; $Fid(\rho, \gamma) = \sqrt{\langle\psi| \gamma |\psi\rangle}$

1.10.2 Correlation

On top of the information readily available from the fidelity between states more information can be obtained computing correlations. Correlation refers to the measure in which two variables are related and the quantum analog to classical correlation is particularly useful since, when computed for pure quantum states, it reveals the presence of quantum entanglement.

The quantum equivalent to mutual correlation *Quantum Mutual Information* (MI) is given by:

$$I(\rho_{AB}) = S(\rho_{AB} \| \rho_A \otimes \rho_B) = S(\rho_A) + S(\rho_B) - S(\rho_{AB}) \tag{48}$$

where ρ_A y ρ_B are the reduced matrices and $S(\rho)$ is the quantum Von Neumann entropy [[9]].

Another correlation measure that will be used in this piece of research is the *Cumulative Correlation Measure*(CCM)[12] which acumulates information on the internal correlation

of the substances of the system, defined iteratively as:

$$C(\rho) = \min_{k}([2^{N-2}S(\rho\|\rho_{Ak} \otimes \rho_{Bk}) + |C(\rho_{Ak}) + C(\rho_{Bk})]) \tag{49}$$

Where k refers to an element (ρ_{Ak},ρ_{Bk}) of all possible bipatritions of the state ρ dividing the state into the product of subspaces until single qubits are reached. Although generally the CCM computes both classical and quantum correlation given the cases of study are made up soley of pure states the result obtained is only quantum correlation.

Chapter 2: Unitary decomposition

2.1 Introduction

At the current stage of quantum computers, that is, Noisy Intermediate Scale Quantum Computers (NISQ), any algorithm that one might desire to run on a quantum hardware needs to be implemented using quantum gates, similarly to low level programming in classical digital electronics. However, several algorithms are given by a unitary matrix that needs to be broken down into smaller blocks, that is, single and two qubit gates. This process is called unitary decomposition, and this chapter introduces an overview of the current state of the art methods to decompose an arbitrary matrix into quantum gates.

2.2 Cartan KAK Decomposition

Cartan KAK decomposition, also known as Kaneja-Glaser decomposition, is an algebraic method borrowed from Lie algebra to tackle the decomposition of a unitary matrix. Cartan KAK1 is the special case we will analyze in this overview, that is the assertion that [13]:

Given any $U \in SU(4)$, one can find A_0, A_1, B_0, $B_1 \in SU(2)$ and $\vec{k} \in R^3$ so that:

$$U = A_0 \otimes A_1 e^{i\vec{k}\cdot\vec{\Sigma}} B_0 \otimes B_1 \tag{50}$$

Where $SU(N)$ is the special unitary group of degree N (where $N = 2^n$ and n is the number of qubits) the Lie group of $N \times N$ unitary matrices whose determinant is equal to 1 and $\vec{\Sigma}$ is an operator independent of U. Cartan KAK1 maps a 4×4 matrix, a two qubit operation, to four 2×2 matrices, one qubit operations and a single non local operation that is given by the term $e^{i\vec{k}\cdot\vec{\Sigma}}$. The non local term introduces operations capable of entangling pairs, and generally refers to a controlled rotation along an axis. In summary, Cartan KAK is a generalization of SVD decomposition for a Lie group and, though powerful, requires a series of algebraic steps including simultaneous SVD decompositions of several matrices

in order to compute A_0, A_1, B_0, B_1.

However, in order to run an algorithm in a quantum computer simply using Cartan KAK1 decomposition is not enough, since at low level the available set of gates is generally reduced to single qubit rotations $R_x(\theta), R_y(\phi), Rz(\alpha)$ and CNOTs. After runing KAK1, one would need to map the non local operation to two CNOTs and a series of single qubit rotations process known as demultiplexing [14]. Furthermore, one would also need to translate matrices A_0, A_1, B_0, B_1 into a series of $R_x(\theta), R_y(\phi), Rz(\alpha)$ gates, and only at this point would one be able to run the algorithm on a quantum computer.

2.3 Other approaches

There are a series of other analytical solutions promptly available throughout literature that are the currently being used. Cosine Decomposition and Quantum Shannon Decomposition are similar approaches, all of which still need further decomposition like in Cartan's case [15], [16]. Furthermore, decomposing larger matrices by this partition methods comes at an increasingly high computational cost since these algorithms need to be applied recursively. These approaches all produce an exact and deterministic decomposition of the desired algorithm.

However, other approaches offer an approximate decomposition. Classical machine learning has been successfully applied to a wide range of problems and unitary decomposition is no exception. Using genetic algorithms which draw from natural selection, some authors have accomplished unitary decomposition by turning on and off a series of non parametric quantum gates [17] [18].

Lastly, this problem can also be tackled using Quantum Machine Learning (QML), a novel interdisciplinary research area at the intersection of quantum mechanics and AI. QML is an umbrella term that refers to any algorithm that learns from data, be it classical or quantum, using classical or quantum optimization methods. Quantum neural networks

are given by variational quantum circuits which are particularly relevant to this study since they have been used to solve this problem [19] and it is the approach taken in this work.

2.4 Variational Quantum Circuits

2.4.1 Introduction

Variational quantum circuits are quantum algorithms made up of gates with free parameters, with three key components:

- A fixed initial state, which can be chosen at will but is generally chosen to be $|0\rangle$
- A quantum circuit with gates that depend on parameters, U(θ)
- Measurement M of at least one of the qubits at the output of the circuit

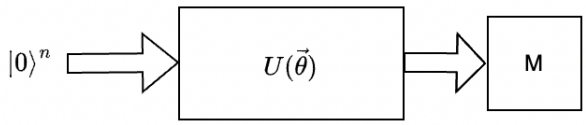

Figure 2: Variational Quantum Circuit

By changing the parameters in $\vec{\theta}$, the output of the circuit can be modified. By comparing the output of the circuit to a target set of states using some sort of error measure, for example the fidelity between these states, the parameters could be tuned iteratively using classical machine learning techniques, such as the gradient descent [20] until the error is minimized.

This idea could be used to produce an approximate decomposition of any quantum algorithm one might desire, provided that the set of gates used to define the basic block, also known as the ansatz, of the variational quantum circuit form a universal set of quantum gates. Similarly to classical neural networks, the power of variational quantum circuits lays in stacking a series of ansatz which are then tuned using gradient descent.

2.4.2 Data encoding: Amplitude encoding

Variational quantum circuits bare many similarities with feedforward neural networks and can likewise be used to solve classification problems [21] [22]. The main issue when training and using quantum classifiers is how to feed classical data into a quantum classifier, what the optimal way to encode classical data into qubits is. This is still very much an open question, given different encodings can give place to different loading order (among others, the number of qubits used) which is critical given quantum machine learning promises a speed up in runtime, provided data encoding can be done in logarithmic or linear time [23]. One popular encoding scheme is amplitude encoding which, depending on the data, can potentially be done in logarithmic time.

To illustrate how this encoding works, let's say we are dealing with a two feature f_1, f_2 problem, where feature refers to information on the data provided, i.e. if the problem was classifying dog and cats, features may refer to the length of their snouts or tails, or whether they have pointy ears. f_1, f_2 can be encoded in a two qubit quantum state:

$$|\Psi_1\rangle = f_1 |0\rangle + \sqrt{1 - f_1^2} |1\rangle \tag{51}$$

$$|\Psi_2\rangle = f_2 |0\rangle + \sqrt{1 - f_2^2} |1\rangle \tag{52}$$

$$|\Psi\rangle = |\Psi_1\rangle \otimes |\Psi_2\rangle \tag{53}$$

2.4.3 Interpreting class labels

Another open question in the variational classifiers realm is how to interpret class labels. Each class is mapped to a quantum state, and the classification is given by the output of

the measurement(s) applied at the end of the circuit, but there are several ways to assign a label. In the case of binary classification, the simplest way to assign a class label is to repeat the experiment a number of times and measure the qubit in which the data is encoded. Given a single qubit is used, the resulting measurement will be either 0 or 1, but the probabilities for each will be computed. Simply thresholding this probability is a way to assign a class to an unlabeled point.

Chapter 3: Quantum circuit synthesis

3.1 Introduction

In this chapter we introduce the quantum variational circuit implemented, discussing design criteria and improvements over past configurations. Here network and variational quantum circuits are used as synonims.

3.2 The basic ansatz

In order to build our variational quantum circuit, the first decision taken was the set of universal quantum gates to use. We opted for one qubit U 3 parameter gates and C_{NOT} since these where the most basic available gates on IBMQ, the cloud platform used to access IBM's quantum computers. To avoid possible confusions between U, the unitary matrix of an arbitrary quantum algorithm and U, the single qubit gate, we will refer to the latter as U_3 throughout this chapter. The expression for this gate is given by:

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{i\lambda+i\phi}\cos\frac{\theta}{2} \end{bmatrix} \tag{54}$$

U_3 is a parametrized one qubit gate capable of becoming any one qubit operation possible. For example, the X gate presented in chapter 1 can be obtained simply by choosing $\theta = \pi$, $\lambda = \pi, \phi = 0$, and the Z gate by choosing $\theta = 0$, $\lambda + \phi = \pi$.

The basic ansatz of our network is made up of an input layer of U_3 gates, one for each input qubit to the system, possibly at least one C_{NOT} gate and a second layer of output U_3 gates. A schematic for a two qubit ansatz is presented in figure 3

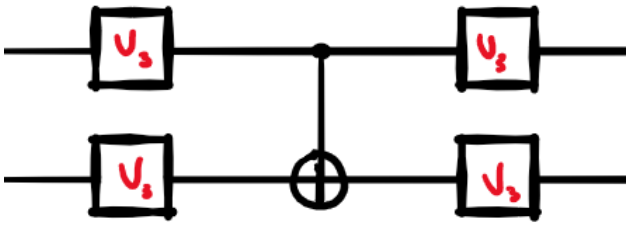

Figure 3: A viable two qubit ansatz

The parameters to be tuned in our variational quantum circuit are the parameters λ, ϕ and θ associated to all the U_3 in the network. Our model needs to decide how many and where to place C_{NOT} gates, which are not placed by tuning parameters and thus fall outside of the scope of a classical optimization. We introduce correlation based learning to this end.

3.3 Correlation based learning

Correlation is a measure of dependency between random variables. In this case, the variables are the results of the measurements of each of the qubits in the input and output sets. Two types of correlations are calculated for each set: Mutual Information (MI) is computed between all pairs of qubits and the Cumulative Correlation Measure (CCM) is calculated for the whole sets. Given the training set is made up solely of pure states, both correlations imply presence (or absence) of quantum entaglement. The only gate in the ansatz capable of producing entanglement is the C_{NOT} gate and studying where the changes in correlation take place, one can guess where to put these gates.

3.3.1 Determining the number of C_{NOTs}

In order to learn quantum algorithms of n qubits more than a single C_{NOT} is usually necessary. Furthermore, if the mininum number of necessary C_{NOTS} can be estimated,

one is also estimating the minimum number of required ansatz given each can only have $n-1$ of these gates.

By virtue of the bipartitions computed, CCM can estimate the minimum number of necessary C_{NOTs}. Esentially, CCM computes the correlations between the subspaces of a given quantum state, which means it takes an n qubit state and brakes it down progressively until it is reduced to n one qubit states, and in the processes accumulates the correlation counts found in all computed subspaces. Given this count, the minimum number of C_{NOTs} is known [12].

3.3.2 Determining C_{NOT} placement

Once the minimum number of required C_{NOTs} is known, we tackle how to place them. For example, in a three qubits system there are three possible placements of these gates (six counting flipping target and control qubits) as seen in figure 4 and as the number of qubits grows, simply trying all possible combinations to see which one minimizes the cost is not viable.

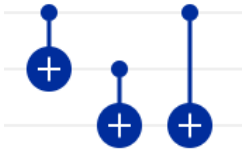

Figure 4: CNOT configuration of a three qubits system

With the purpose of automating this process, an auxiliary input set is built combining $|+\rangle$ states with $|0\rangle$ states. These states are combined starting by the subset made up of one $|+\rangle$ and $(n-1)$ $|0\rangle$, then two $|+\rangle$ and $(n-2)$ $|0\rangle$ and so on until the states with $(n-1)$ $|+\rangle$ and a single $|0\rangle$ are generated, where n is the number of qubits of the system. Where $|+\rangle$ is given by:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{55}$$

In order to show this set's virtues, the two qubit case will be studied. In this case, the auxiliary set is composed only of two states:

$$|\Psi_{aux}\rangle_1 = |+\rangle \otimes |0\rangle \tag{56}$$

$$|\Psi_{aux}\rangle_2 = |0\rangle \otimes |+\rangle \tag{57}$$

Furthermore, a CNOT in a two qubit system can only be placed in two different positions. If the target is the second qubit and the control the first and this auxiliary set is run through it:

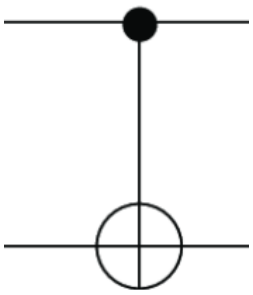

Figure 5: CNOT in the first position

When $|\Psi_{aux}\rangle_1$ goes through:

$$|\Psi_{out}\rangle_1 = |+\rangle \otimes |1\rangle \tag{58}$$

Given that there is a $|1\rangle$ in the control qubit, it is activated and it changes the second qubit. Moreover, when the second state in the auxiliary set goes through:

$$|\Psi_{out}\rangle_2 = |0\rangle \otimes |+\rangle \tag{59}$$

Given there is no $|1\rangle$ in the control qubit, the target remains unaltered.

Analogly, for the C_{NOT} in the opposite position:

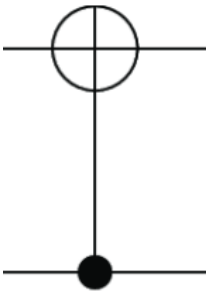

Figure 6: CNOT in the opposite position

For the first state int he auxiliary set:

$$|\Psi_{out}\rangle_1 = |+\rangle \otimes |0\rangle \tag{60}$$

Given there are no $|1\rangle$ states in the control qubit, the target remains unchanged. For the second set in the auxiliary set, the control is activated and the target changed.

$$|\Psi_{out}\rangle_2 = |1\rangle \otimes |+\rangle \tag{61}$$

Results are presented in the table below.

	First position	Opposite position
$\|\Psi_{out}\rangle_1$	$\|+\rangle \otimes \|1\rangle$	$\|+\rangle \otimes \|0\rangle$
$\|\Psi_{out}\rangle_2$	$\|0\rangle \otimes \|+\rangle$	$\|1\rangle \otimes \|+\rangle$

Table 1: Output for each input state in the set for each C_{NOT} configuration

The outputs of different configurations of C_{NOTs} to this auxiliary input set is never the same for any two given configurations. This analysis allows us to place these gates in the different ansatz in the network.

It is worth noting that the auxiliary set grows exponentially with the number of qubits in the system, the three qubit set is given by:

- $|+,0,0\rangle$
- $|0,+,0\rangle$
- $|0,0,+\rangle$
- $|+,+,0\rangle$
- $|0,+,+\rangle$
- $|+,0,+\rangle$

And for an n qubit system, the number of states in the auxiliary set is given by expression (62).

$$\#\{\Psi_{out}\} = \sum_{j=1}^{n-1} C_j^n = 2^n - 2 \tag{62}$$

3.4 Single qubit matrices

3.4.1 Optimization

Given the continuous parameters to be tuned are the one associated to U_3 gates and each gate has 3, θ, ϕ, λ, for each ansatz used in the system the number of tunable parameters is given by expression (63) where ω is the name given to the vector made up of all tunable parameters and n is the number of qubits in the system.

$$\#\{\omega\} = 2(3^{n-1}) \tag{63}$$

The optimization run for these parameters is simply steepest gradient descent, a classical optimization algorithm widely used in classical machine learning. The cost or loss function is given by the fidelity between the output of the network and the target states.

Even though this is a multivariable optimization with multiple possible local minima, the target global minimum is known give we desire the fidelity between output and target to be 1, which would mean the output is equal to the target.

Now let the output of A_i, the i-th ansatz in the network, be $A_{i_{out}}$ and let the target be A_T. The loss is then given by equation (64)

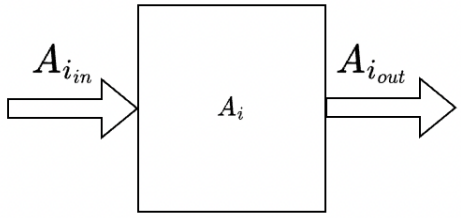

Figure 7: i-th ansatz in the network

$$J = Fid(A_{i_{out}}, T) \tag{64}$$

Given J is actually a vector of the size of the training set, one must choose a value (mean, max, min) from the set in order to optimize. By default the network is adapted according to the mean of J, but can be modified to use the minimum value of J provided one is aware

it might bring along discontinuations or jumps in the optimization evolution, a problem that using the mean does not have.

Once the criterion has been set, the gradient of J with respect to the parameters is calculated and then said parameters are updated by a rule combining their past value, the gradient and the learning rate of the system, a parameter that sets the learning pace of the network. Then the fidelity between target and the new outputs is computed: if fidelity increases, then these new parameters are returned in the ansatz, if not, the original ansatzs is returned.

3.4.2 Hyperparameters

The network is then optimized as a whole following certain stop criteria.

- A maximum number of iterations is reached, which can be set as an input to the optimization. In this case, the result might not be optimal and the fidelity might not be high enough
- A target minimum of mean output fidelity is reached, which can also be set by the user at the begining of the optimization. If this criterion is met, the optimization finishes because the circuit obtained accurately decomposes the target matrix.
- Output fidelity stalls and does not grow above a small tolerance for the last k iterations, with $k = 10$ by default.

The maximum number of iterations, the minimum output fidelity and the minimum tolerable change in fidelity are all hyperparameters that can be set for each optimization. Furthermore, there is another set of hyperparameters which deal with possible convergence issues:

- Variable step size: If fidelity increases between two consecutive iterations, the step size of the optimization grows according to a growing rate provided by the user. If, in the other hand, fidelity decreases between two consecutive steps then the step size

is reduced according to another rate provided by the user. Both changes saturate: the step grows no more than 10 times the initial step and is not decreased beyond 0.1 times the initial step. This range was found to work well empirically.

- Momentum: The learning rule of the network is not only given by the gradient multiplied by the learning rate, but also includes a term that factors in the change in the parameters given in the previous iteration. The learning rule of the network is given by equation (65) where γ is the momentum and the expression has been written for parameter λ_i but can be generalized for all tunable parameters.

$$\Delta_{\lambda_i} = \eta \frac{\partial J}{\partial \lambda_i} + \gamma \Delta_{\lambda_i}^{t-1} \tag{65}$$

  Momentum is by default set to 0.1 but can be changed to suit other problems.

Both of these last two parameters aim to prevent the optimization from getting caught and oscillate around local minima.

3.5 Training a whole network

The learning rules for both C_{NOTs} and U_3 matrices was introduced, as well as how a single ansatz is optimized. In general, more than one ansatz will be necessary to decompose a given algorithm. This section deals with the optimization of a full network, a cascade of ansatz as can be seen in figure 8.

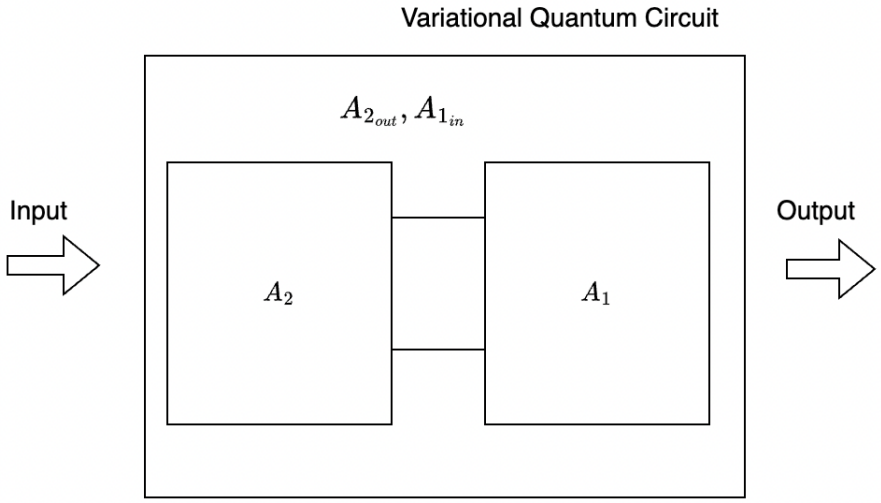

Figure 8: A two ansatz variational quantum circuit

To explain the manner in which the ansatz are added and trained, a two ansatz circuit will be used. The conclusions apply to any m size circuit. Let $Input$ and $Output$ be the training set for the whole network and let A_1 and A_2 be the composing blocks. The first ansatz added to the network is in fact the last one in the chain, numbered A_1. This block is trained with $Input$ and $Output$ until a stopping condition is reached, but, by itself, it cannot reach the target minimum fidelity. A_1 is then freezed and a blank ansatz A_2 is added. A_2 is then trained using $Input$ and the set $A_{2_{out}}$, which is calculated using the matrix associated to $A_1(U_{A_1})$ so that $A_{2_{out}} = U_{A_1}^{\dagger} Output$. Generalizing, for A_{i+1}:

$$A_{i+1_{in}} = Input \tag{66}$$

$$A_{i+1_{out}} = U_{A_1}^{\dagger} A_{i_{out}} \tag{67}$$

This manner introduces a notion of backwards propagation of knowledge throughout the

network. The training loop in fact switches between training single blocks and the whole network together. To further exemplify, for this circuit the training loop would be:

- Add A_1, train it to its maximum potential and freeze it. Calculate $A_{2_{out}}$
- Add A_2, train it *by it self* using $Input$ and $A_{2_{out}}$
- If at this point the target fidelity is not reached, train both ansatz together using $Input$ and $Output$

This mixed training scheme was developed to cut down on processing times given that as the network grows, training all parameters together becomes increasingly slower. This mixed scheme allows the optimization of the whole network to begin at a point significantly closer to optimal than just adding an empty block and training them all together.

Lastly, a single last hyperparameter is added: the maximum number of tolerated blocks. The optimization can then end before this maximum number is reached if the output fidelity reaches its target, or will be terminated when this maximum is achieved and the user will then have to set a new larger number of maximum blocks and run the optimization again.

Chapter 4: Use cases

4.1 The Toffoli gate

The motivation behind this research was to develop an algorithm capable of decomposing the Toffoli gate into a cascade of single qubit operations and C_{NOT} gates. The Toffoli gate can be thought of as a generalization of the C_{NOT} for 3 qubits, where both of the control qubits need to be activated in order to act over the target. This gate is notable because the generalized Toffoli gate, which can be created from regular Toffoli gates, is widely used in many algorithms.

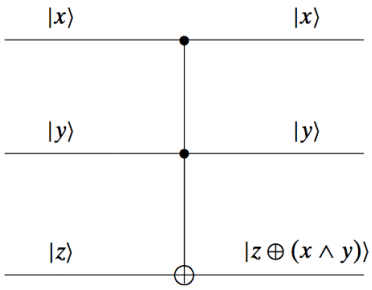

Figure 9: Toffoli Gate

The minimum necessary number of CNOTs required to decompose a Toffoli gate has been found to be 6 [24] so the optimization was allowed to run for a maximum number of 10 blocks to allow room for error. Given the numerical algorithm's statistical nature and the strong dependence on the initial conditions of the network, different runs of this optimization method resulted in different decompositions of this gate, with the most favourable outcome being the one presented below in Figure 10. In said figure, the blue curve is the mean output fidelity for the entire train set. Furthermore, the red curve is a low pass filtered version of the blue curve to clean oscillations in data.

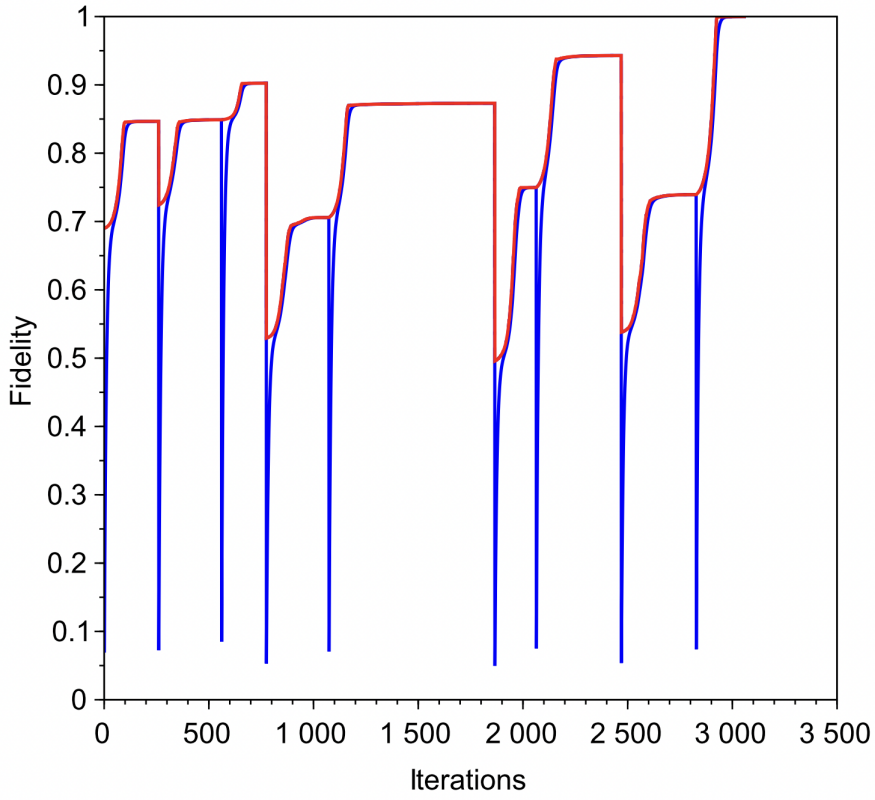

Figure 10: Evolution of the fidelity while learning a Toffoli gate

This experiment resulted in a 5 block decomposition of the Toffoli gate with 10 CNOTs, which is above the theoretical minimum found above, the final quantum circuit is presented below.

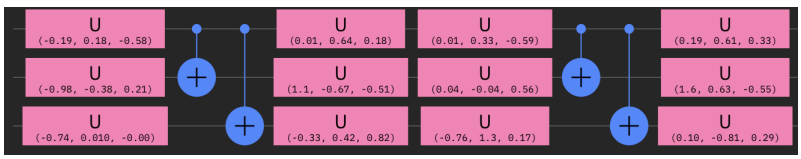

Figure 11: Toffoli decomposition: Blocks 1 and 2

Figure 12: Toffoli decomposition: Blocks 3 and 4

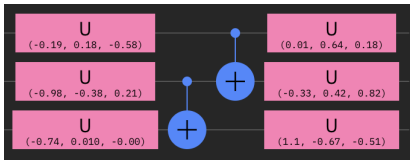

Figure 13: Toffoli decomposition: Block 5

4.2 Machine learning application: Hybrid classifier

4.2.1 Introduction

Given the network can potentially learn any unitary matrix, the proposed problem we decided to tackle was a multiclass classification. A multiclass classification problem refers to classifying a set of data into three or more classes according a certain set of features. This problem is tipically tackled by machine learning and support vector machines or can also be tackled by deep learning techniques, using classical neural networks. Our aim is to build a quantum classifier not to prove it better than its classical counter part, but to show that quantum classification is possible.

4.2.2 Iris dataset: Preprocessing and feature encoding

Iris [25] is a publicly available dataset contains 150 samples of 3 different types of iris flower species: Setosa, Versicolor and Virginica. The set includes 4 different features for sample: sepal length, sepal width, petal length and petal width. This is a very simple classification problem and it is one of the datasets most widely used in educational applications today.

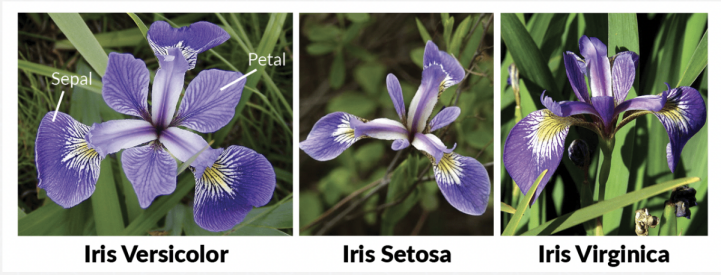

Figure 14: Iris classification. Image taken from [25]

A graph for each feature value and for each sample is presented below.

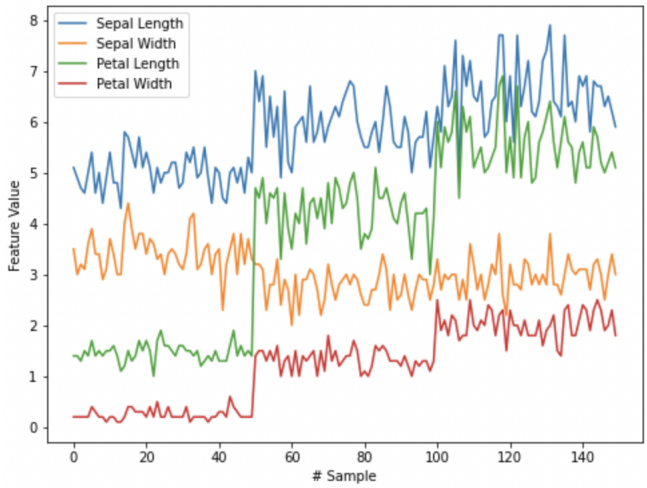

Figure 15: Features by sample

Examining the graph above it becomes clear that certain features carry the equivalent information and some could be omitted so the feature correlation matrix was analyzed. This matrix is calculated computing the pairwise correlation between features.

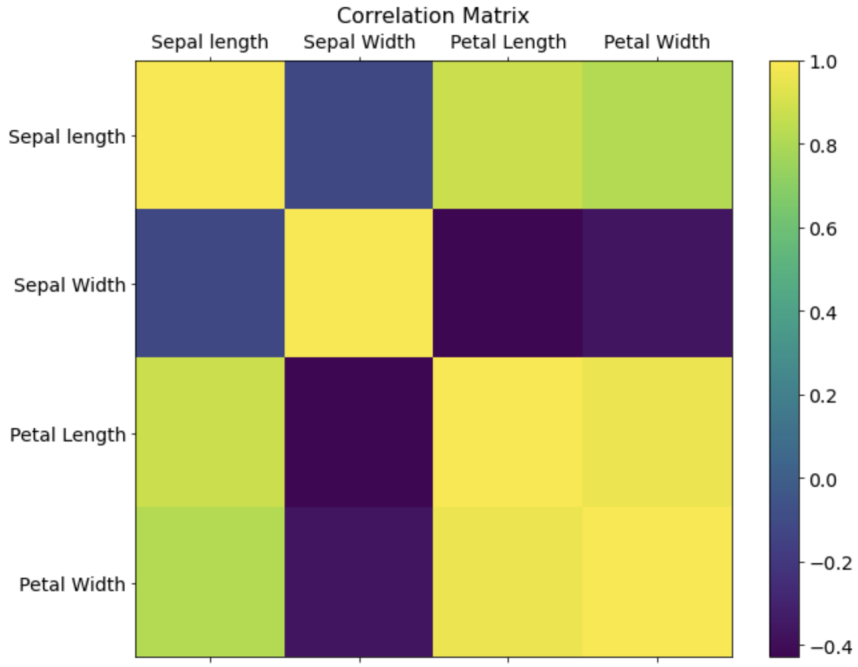

Figure 16: Feature correlation matrix

The feature correlation matrix confirms the hypothesis: sepal length is strongly correlated to petal length and width and thus two features can be dropped, we chose to drop sepal length and sepal width. Further analyzing the two remaining features and the ground truth of the classification:

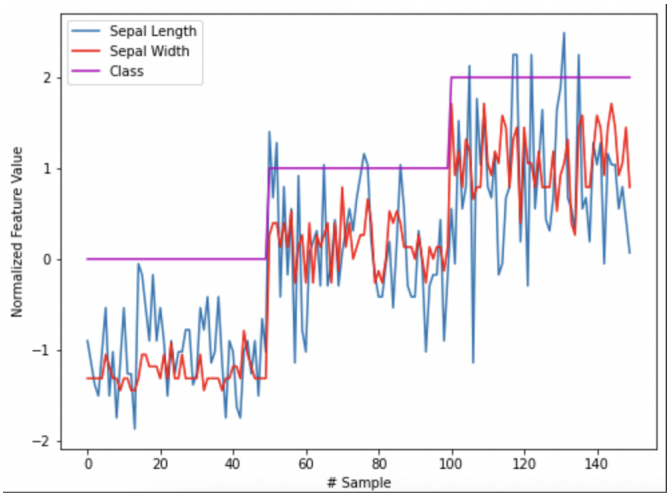

Figure 17: Selected features and class ground truth

We decided to encode these two features using the amplitudes of two separate qubits, but in order to do so the feature values needed to be bounded to the [0, 1] interval. To normalize these features two sigmoid functions where introduced, one for each feature, which are in charge of parting the values in two separate regions. The sigmoid acts simply as a filter, taking an inflexion point c and a slope a as parameters and the features x as input and filtering according to equation 68.

$$S(x) = \frac{1}{1 + e^{-a(x-c)}}$$

c was simply chosen to be the midpoint between the center of a feature for two consecutive classes, and a was calculated as the slope of the straight line that joins the centers of each feature value of each class. Features before and after filtering can be observed below, where the inflexion point c is marked as a red diamond in each graph.

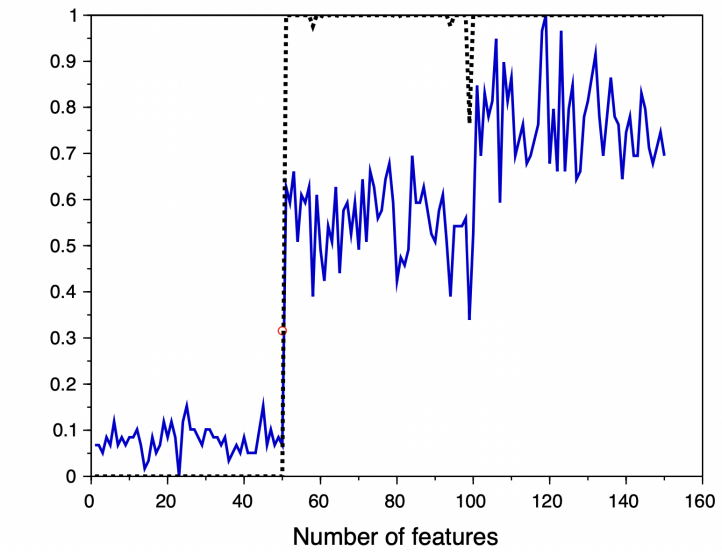

Figure 18: Selected features and class ground truth

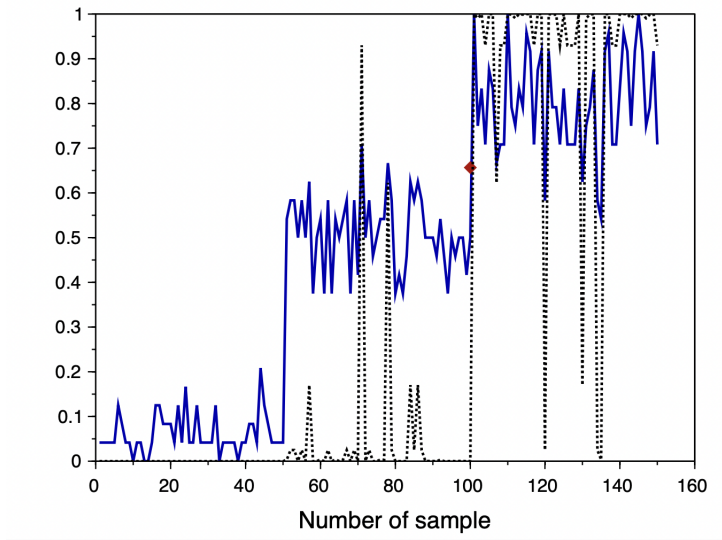

Figure 19: Selected features and class ground truth

Analyzing the filtered graphs it becomes clear that this filtering technique works far better for the first feature that for the second one, where variability in the second feature below

and above the inflexion point leads to a noisier output signal.

Further optimizations could be useful in this step, for example, optimizing c and a parameters for each sigmoid function in order for the filtering to be optimal. This task was put aside for the time being given that the main focus of this application is to prove if the quantum circuit synthesis algorithm can be applied to a classical problem.

With these filtered feature values the two qubit encoding state was put together:

$$|\Psi_1\rangle = f_1 |0\rangle + \sqrt{1 - f_1^2} |1\rangle$$

$$|\Psi_2\rangle = f_2 |0\rangle + \sqrt{1 - f_2^2} |1\rangle$$

$$|\Psi\rangle = |\Psi_1\rangle \otimes |\Psi_2\rangle \tag{68}$$

4.2.3 Training the classifier and interpreting classifications

Two optimizations were run in order to find the circuit that correctly classified the flowers: first, one optimization what could discover the mapping matrix and second, one that could decompose said matrix into its corresponding quantum circuit.

Finding a matrix that mapped the input states to the corresponding classes meant running an optimization in search of unitary and hermitian mapping. Given the input was already a two qubit state and a single ancilla qubit was added to improve expressivity, the matrix in question was an 8 x 8 matrix and with the necessary constraints this meant a 64 parameter optimization. Each output class, numbered 1 through 3, was mapped to a quantum state and the distance between the output state and the target was computed and minimized iteratively.

Class 1	Class 2	Class 3
$\|100\rangle$	$\|010\rangle$	$\|001\rangle$

Table 2: Class mapping

With the mapping matrix U, we then proceeded to decompose it into quantum gates in the same manner as was explained in chapter 3. Once the model was trained it could be used to run inference on new data, the full pipeline is presented below.

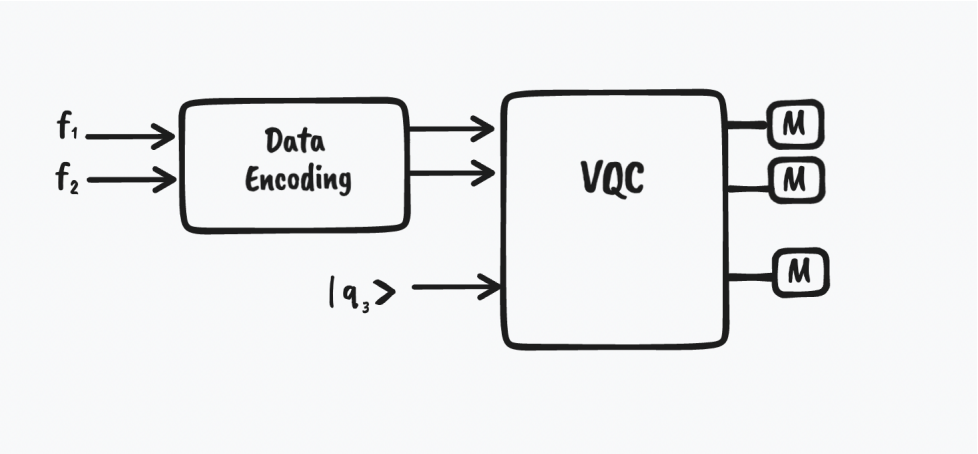

Figure 20: Inference pipeline

Measuring the output of the variational quantum classifier (VQC) and counting occurrences for each of the possible outcomes where the correct class bit was activated, the classification results were obtained. The output state could be reconstructed running a quantum state tomography, but we steered clear of it since it is a computationally expensive algorithm. The results for each of the three classes are presented below, where the first two bars correspond to the classification over the training set and the last two to the classification over the test set. In each case, the bar on the left (blue for training, red for test) corresponds to the classification done using the mapping matrix U and the bar on the right corresponds to the classification done with the quantum circuit (green for train, sky blue for test).

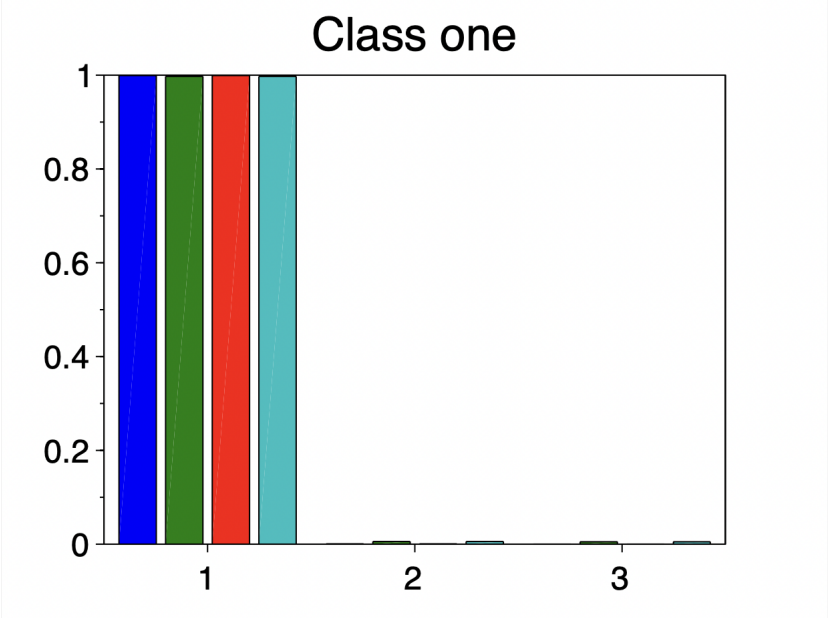

Figure 21: Accuracy over class 1

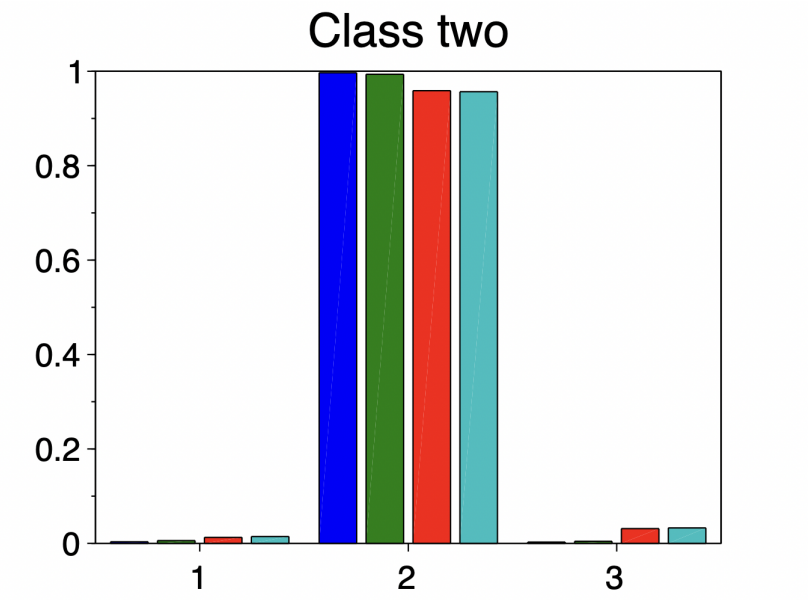

Figure 22: Accuracy over class 2

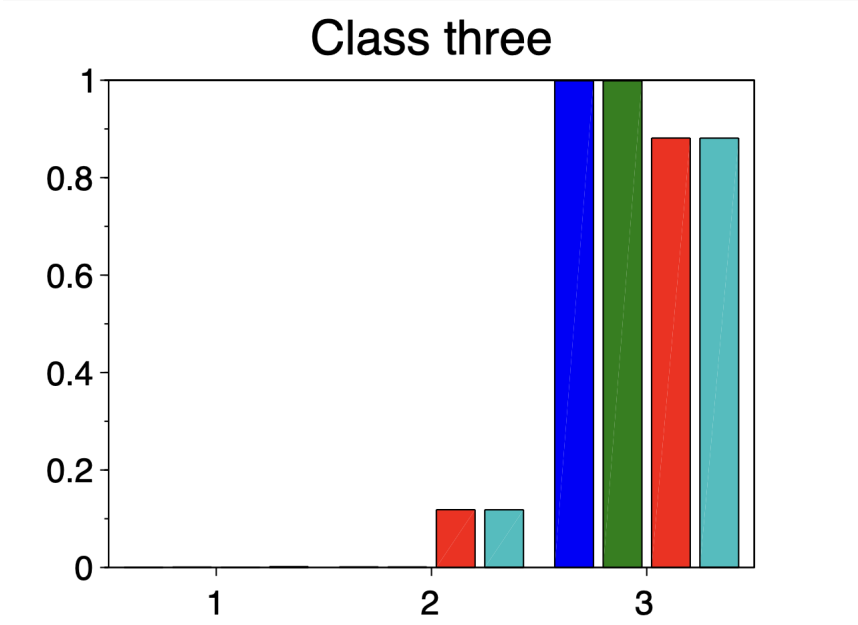

Figure 23: Accuracy over class 3

The corresponding accuracies for each class over the test set using the quantum circuit are presented in Table 3.

Class 1	Class 2	Class 3
0.997	0.957	0.881

Table 3: Accuracy per class

Chapter 5: Discussion and conclusions

5.1 Toffoli gate decomposition

The mapping of the Toffoli gate was quite successful in the sense that the network was able to correctly create a decomposition of the gate, albeit, with more CNOT gates than the theoretical minumin required. This is still a major improvement over our previous work [26], where we were not able to produce an accurate decomposition of the Toffoli gate. Given that the statistical nature of our algorithm means that it will produce different decompositions each time it is run, it is possible that running the algorithm several times for this gate could potentially produce a version of this gate with fewer blocks. In sum, one could run an optimization over the optimization to find the optimal decomposition for this gate.

5.2 The hybrid classifier

The goal of this application was to prove that the quantum circuit synthesis algorithm would be able to perform favourably in a classical classification problem. Figures (21) through (23) promptly show this hypothesis was verified, with an over all accuracy of 0.94. Although the overall metric is favourable, performance over classes 2 and 3 is poorer than over class 1, which probably has to do with the data encoding. Furthermore, as was mentioned in chapter 4, filtered feature 2 is far noisier than feature 1 which contributes to making classes 2 and 3 difficult to tell apart. Performance could probably be improved optimizing the position of the inflexion points and slopes of the filtering sigmoid functions.

Furthermore, the issue of the optimal way to map classical data into quantum states is still an open question. We acknowledge the mapping used in this piece of research is not

the optimal mapping, and we are looking into improving this part in further work. We are exploring Quantum Random Access Codes (QRACs) [27] as a means of improving our encoding of data.

The goal of this piece of research was to improve on an automatic method to decompose an arbitrary quantum operation into a cascade of universal quantum gates, work which had firstly been tackled in my undergraduate thesis [26]. A hybrid deep learning-like algorithm was created, which tunes the parameters associated with quantum gates iteratively following the gradient descent method that can solve, for example, a classification problem.

References

[1] Arute, F., Arya, K., Babbush, R. et al. "*Quantum supremacy using a programmable superconducting processor*", Nature, vol. 574, 2019, pp. 505–510.

[2] Shor P. W. et al. "*Quantum Error Correction and Orthogonal Geometry*". Physical Review Letters vol.78 no.3, 1997, pp.405–408

[3] Shor, P. "*Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*", Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, 1994.

[4] Grover, L. " *A fast quantum mechanical algorithm for database search*" Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), 1996, pp. 212-219.

[5] Nielsen, M., Chuang, I. "*Quantum computation and Quantum Information*", Cambridge University Press, 2000, pp. 80.

[6] Nielsen, M., Chuang, I. "*Quantum computation and Quantum Information*", Cambridge University Press, 2000, pp.. 81.

[7] Nielsen, M., Chuang, I. "*Quantum computation and Quantum Information*", Cambridge University Press, 2000, pp. 85.

[8] Nielsen, M., Chuang, I. "*Quantum computation and Quantum Information*", Cambridge University Press, 2000, pp. 94.

[9] Nielsen, M., Chuang, I. "*Quantum computation and Quantum Information*", Cambridge University Press, 2000, pp 109.

[10] Dawson, C., Nielsen, M., "*The Solovay-Kitaev Algorithm*", Quantum Information and Computation, Vol. 0, No. 0, 2005.

[11] Nielsen, M., Chuang, I. "*Quantum computation and Quantum Information*", Cambridge University Press, 2000, pp. 191-194.

[12] Fonseca, A., Buksman, E., López De Lacalle, J., "*Cumulative measure of correlation for multipartite quantum states*", International Journal Of Modern Physics B, vol. 28, 2013.

[13] Tucci, Robert R. "*An introduction to Cartan's KAK decomposition for QC programmers.*", 2005, [Online] Available: https://arxiv.org/pdf/quant-ph/0507171.pdf Accessed on: February 14th, 2022

[14] Krol, A. M. "*Unitary Decomposition Implemented in the OpenQL programming language for quantum computation*", 2019. [Online] Available: https://repository.tudelft.nl object OBJ download Accessed on: Februrary 15th, 2023.

[15] Krol, A.M., Sarkar, A., Ashraf, I., Al-Ars, Z., Bertels, K. "*Efficient Decomposition of Unitary Matrices in Quantum Circuit Compilers.* Appl. Sci. vol. 12, no. 759, 2022

[16] Li, Chi-Kwong, Rebecca Roberts, and Xiaoyan Yin. "*Decomposition of unitary matrices and quantum gates.*", International Journal of Quantum Information vol. 11, no.01, 2013.

[17] Miranda, Fernando T., Pedro Paulo Balbi, and Pedro Costa. "*Synthesis of quantum circuits with an island genetic algorithm*" , 2021, [Online] Available: https://arxiv.org/pdf/2106.03115.pdf Accessed on: December 12th, 2022.

[18] Bang, Jeongho Yoo, Seokwon. "A genetic-algorithm-based method to find unitary transformations for any desired quantum computation and application to a one-bit oracle decision problem." Journal- Korean Physical Society, vol. 65, no. 2001, 2015

[19] Y. Huang, X. Li, Y. Zhu, H. Lei, Q. Zhu et al., "*Learning unitary transformation by quantum machine learning model*", Computers, Materials Continua, vol. 68, no.1, pp. 789–803, 2021.

[20] Kathuria, A. "*Intro to optimization in deep learning: Gradient Descent*", 2018. [Online] Available: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/ Accessed on: January 30, 2023

[21] Maheshwari, D., Sierra-Sosa, D., Garcia-Zapirain, B. " *Variational Quantum Classifier for Binary Classification: Real vs Synthetic Dataset*," IEEE Access, vol. 10, pp. 3705-3715, 2022.

[22] Saxena, N., Akriti, N. "*Performance Evaluation of a Variational Quantum Classifier.*" 2022 IEEE 9th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON), pp.1-5, 2022.

[23] Weigold, M., Barzen, J., Leymann, F. and Salm, M.*"Data encoding patterns for quantum computing."* In Proceedings of the 27th Conference on Pattern Languages of Programs, pp. 1–11, 2022.

[24] Vivek V. Shende and Igor L. Markov. 2009. On the CNOT-cost of TOFFOLI gates. Quantum Information Computation, vol. 9, no. 5, pp. 461–486, 2009.

[25] The UCI Machine Learning Repository, Center for Machine Learning and Intelligent Systems [Online] Available: http://archive.ics.uci.edu/ml/datasets/Iris Accessed on: September 26, 2022

[26] Allende, C. "Redes neuronales cuánticas", 2021. [Online] Available: https://sisbibliotecas.ort.edu.uy/cgi-bin/koha/opac-retrieve-file.pl?id=844409f5c37504f3ebb95e6517a1f340 Accessed on: January 10, 2023

[27] Ambainis, A., Leung, D., Mancinska, L., and Ozols, M., "Quantum Random Access Codes with Shared Randomness", 2008, [Online] Available: https://arxiv.org/pdf/0810.2937.pdf Accessed on: January 30, 2023

Annex I: User Manual

Auxiliary functions

```
[comb_lst, cant_comb] = comb2qbs(nqbs)
```

It takes the number of qubits as input and returns a list of all possible two qubit combinations.

```
[sts, comb_lst, cant] = GeneraEstadosCeroMas(nqbs)
```

It generates the auxiliary set of states to find CNOT placements. It takes the number of qubits as input and returns the auxiliary set and the total number of generated states.

```
[sts, cant] = GeneraEstadosGiros(nqbs, cant_giros)
```

It generates states resulting from rotating qubits around the X and Y axis, taking the number of qubits and the number of desired rotations as input and returning a list with the desired rotations and the total number of generated states.

Correlation related functions

```
[max_ccm, ind_max, v_ccm] = CalculaCCM(sts)
```

It takes a list of vector states as input, calculates their CCM and returns them a vector v_ccm for each of the input states, as well as the maximum CCM found, max_ccm, and the index for which it occurs ind_max.

```
cant_CNots = CantidadCNots(val_ccm)
```

It takes a value of CCM as inpt and calculates the necessary number of CNOTs.

```
[med_corr, corr_res, corr_max, ind_max] = OutqbsCorr(sts, comb_lst)
```

Taking a list of vector states as input and a list of two by two combinations, this function calculates the mutual information for each combination and state and returns it in matrix form corr_res. It then calculates the mean correlation for each qubit and returns a vector med_corr and its maximum corr_max, as well as the index for which it ocurrs.

One qubit unitary matrix functions

```
U = qb_QAN_Unit3(parsU3)
```

It takes a list of order parameters θ, ϕ, λ as input and returns the one qubit unitary matrix asociated to said parameters.

```
[parsU] = qb_QAN_Inv_Unit3(U)
```

This function computes the inverse function to qb_QAN_Unit3.

```
[CNot_lst] = qb_QAN_CreateCNots(nqbs, comb_lst)
```

It takes the number of qubits and a list with two qubit combinations and returns a list with all possible CNOTs for the system.

Functions associated to cells

```
QC = qb_QAN_QCell_create(nqbs, sts_out, comb_lst, Cnot_lst)
```

This function takes the number of qubits, nqbs, the output of the function GeneraEstadosCeroMas sts_out, the list of two qubit combinations comb_lst and the output of qb_QAN_CreateCnots, Cnot_lst, as input and returns an object of type QCell where:

- QC(1): nqbs
- QC(2): ind_Cnot, the qubit pair where the CNOTs are to be placed

- QC(3): U_Cnot, the matrix asociated to the CNOTs
- QC(4): Parameters for the input layer of single qubit gates, of length 3nqbs
- QC(5): Parameters for the output layer of single qubit gates, of length 3nqbs
- QC(6): Matrix of the cell

```
[QC, UQC] = qb_QAN_QCell_U(QC)
```

It takes a QCell as input and returns the same block and its matrix.

```
[QC, inc_fid, fid_out, vec_grad] = qb_QAN_QCell_AdaptU1(QC, sts_in, sts_out,

vec_grad, parsU1, calc_mean)
```

This function takes a QCell as input, as well as an input set of states sts_in, an output one sts_out, as well as three extra optional input parameters: vec_grad is a four element vector whwre the first two values are used to estimate the gradient, the third to estimate the step and the last is the momentum. parsU1 is a $2(3^{nqbs})$ with an initial guess for the one qubit parameters and calc_mean is a flag parameter that indicates whether the optimization is to be run using the minimum or mean fidelity is metric. It outputs the QCell, the increment of fidelity in the step inc_fid, the output fidelity fid_out and the change to be applied to the parameters, vec_grad.

```
[QC_out, vec_fid, med_vec_fid] = qb_QAN_QCell_AdaptU(QC, sts_in, sts_out, pars_opt)
```

This function takes a QCell as input, a set of input states sts_in and their corresponding desired output sts_out, and a vector of parameters pars_opt where:

- pars_opt(1): Parameter to estimate gradient
- pars_opt(2): Parameter to estimate gradient
- pars_opt(3): Initial step of the optimization

- pars_opt(4): Initial momentum for the optimization
- pars_opt(5): Rate at which the step should increase
- pars_opt(6): Rate at which the step should decrease
- pars_opt(7): Maximum number of iterations
- pars_opt(8): Percentage of input states used for training
- pars_opt(9): Optimization stopping parameter: minimum desired output fidelity
- pars_opt(10): Optimization stopping parameter: minimum tolerated fidelity change
- pars_opt(11): Optimization stopping parameter: maximum difference between the fitered and unfiltered graphs of fidelity
- pars_opt(12): How much the filter filters

It returns a vector of output fidelities vec_fid, the mean of said vector med_vec_fid and the output Qcell, QC_out.

```
QN = qb_QAN_QNet_create(nqbs)
```

This function takes the number of qubits as input and returns an object of type QNet, QN, where:

- QN(1): nqbs
- QN(2): comb_lst
- QN(3): List of all CNOTs in de net
- QN(4): List of all QCells in the net
- QN(5): Matrix of the full QNet

```
QN = qb_QAN_QNet_insertCell(QC, sts_out)
```

This function takes a QCell, QC, as input and the set of states obtained using GeneraEstadosCeroMas, returning an object of type QNet, QN, with the input cell embedded.
.

```
[QN, net_fid, out_fid, med_fid, UQN] = qn_QAN_QNet_adapt(QN,

sts_in_cell, sts_out_cell,

sts_in_train, sts_out_train, pars_opt_cell, pars_opt_net)
```

This function takes as input QNet, the input and output sets for the first QCell , a set of input and output states for the first QCell sts_in_cell and sts_out_cell, a set of input and output states to train the whole network sts_in_train and pars_opt_net, where this last parameter refers to the maximum number of cells tolerated. It returns a net QN, the fidelity of said net net_fid, out_fid and med_fid are the vectors with the evolution of fidelity and mean fidelity in the adaptative process, and the matrix associated to the network, UQN.

Anex II: Code

```
function [comb_lst, cant_comb] = comb2qbs(nqbs)
    comb_lst = [];
    for i=1:(nqbs-1)
        for j=(i+1):nqbs
            temp = [i j];
            comb_lst = [comb_lst; temp];
        end
    end
    cant_comb = size(comb_lst, 'r');
endfunction

function [sts, comb_lst, cant] = GeneraEstadosCeroMas(nqbs)

comb_lst = qb_ut_bmatrix(nqbs);
st_0 = qb_st_state(0);
st_M = qb_st_state([0 1], 1, [1 1]);

sts = [];

for i=2:size(comb_lst, 'r')
    aux_st = 1;
    aux_comb = comb_lst(i, :);
    for j=1:nqbs
        if aux_comb(j)==0 then
            aux_st = qb_ut_tensorial(aux_st, st_0);
        else
            aux_st = qb_ut_tensorial(aux_st, st_M);
        end
    end
```

```
    sts = [sts aux_st];
end

aux_sts =qb_st_state([0 2^nqbs-1],nqbs,[1 1]);
sts = [aux_sts sts];

cant = size(sts, 'c');
endfunction

function [sts, cant] = GeneraEstadosGiros(nqbs, cant_giros)

    st0 = qb_st_state(0, nqbs);

    sts = [];
    for j=(%pi/((cant_giros)*4)):(%pi/((cant_giros)*4)):(%pi/4)
        auxM = qb_cmp_Xa(nqbs, j);
        auxst = auxM*st0;
        sts = [sts auxst];
    end

    for j=(%pi/((cant_giros)*4)):(%pi/((cant_giros)*4)):(%pi/4)
        auxM = qb_cmp_Ya(nqbs, j);
        auxst = auxM*st0;
        sts = [sts auxst];
    end

    cant = size(sts, 'c');
endfunction


function [max_ccm, ind_max, v_ccm] = CalculaCCM(sts)
```

```
    [tp, cant, cqubits] = qb_st_type(sts);
    v_ccm = zeros(1:cant);
    for i=1:cant
        v_ccm(i) = qb_corr_CCM(sts(:,i));
    end
    max_ccm = max(v_ccm);
    ind_max = min(find(v_ccm == max_ccm));
endfunction

function cant_CNots = CantidadCNots(val_ccm)
    d = 0.5
    cant_qbs = 2:10;
    fact_d = [0 1 2 4 6 9 12 18 24];
    tabla = 2.^(cant_qbs-2) + fact_d*d;

    if (val_ccm == 0) then
        cant_CNots = 0;
    elseif (val_ccm <= tabla(1)) then
        cant_CNots = 1;
    elseif (val_ccm <= tabla(2)) then
        cant_CNots = 2;
    elseif (val_ccm <= tabla(3)) then
        cant_CNots = 3;
    elseif (val_ccm <= tabla(4)) then
        cant_CNots = 4;
    elseif (val_ccm <= tabla(5)) then
        cant_CNots = 5;
    else
        cant_CNots = 10;
    end
```

```
endfunction

function [med_corr, corr_res, corr_max, ind_max] = OutqbsCorr(sts, comb_lst)
    [lhs rhs]=argn();
    [tp_st, cant_sts, nqbs] = qb_st_type(sts);
    if (rhs < 2) then
        [comb_lst, cant_comb] = comb2qbs(nqbs);
    end
    cant_comb = size(comb_lst, 'r');
    corr_res = zeros(cant_comb,cant_sts);
    for i=1:cant_comb
        inds_comb = comb_lst(i,:);
        for j=1:cant_sts
            aux_rho1 = qb_st_vec2dens(sts(:,j));
            aux_st_2qbs = qb_st_partialtrace(aux_rho1, inds_comb);
            aux_st_1qbsA = qb_st_partialtrace(aux_st_2qbs, 1);
            aux_st_1qbsB = qb_st_partialtrace(aux_st_2qbs, 2);
            [res, mi] = qb_corr_DistMI(aux_st_2qbs, aux_st_1qbsA, aux_st_1qbsB)
            corr_res(i,j) = mi;
        end
    end
    med_corr = mean(corr_res, 'c');
    if (lhs > 1) then
        ind_max = min(find(med_corr == max(med_corr)));
        corr_max = med_corr(ind_max);
    end
endfunction

function U = qb_QAN_Unit3(parsU3)
    th = parsU3(1);
    ph  = parsU3(2);
```

```
    lamb = parsU3(3);
    U = [cos(th/2) -exp(%i*lamb)*sin(th/2); exp(%i*ph)*sin(th/2) exp(%i*(lamb+ph))*cos(
    U = clean(U);
endfunction;

function  [parsU] = qb_QAN_Inv_Unit3(U)
    th = clean(2*acos(U(1,1)));
    if (abs(sin(th/2))<%eps) then
        th = 0;
        aux = U(2,2);
        aux = atan(imag(aux), real(aux));
        ph = aux/2;
        lamb = aux/2;
    else
        aux = U(2,1)/sin(th/2);
        ph = atan(imag(aux), real(aux));
        aux = U(1,2)/(-sin(th/2));
        lamb = atan(imag(aux), real(aux));
    end
    parsU = [th ph lamb];
endfunction;



function [CNot_lst] = qb_QAN_CreateCNots(nqbs, comb_lst)

    [lhs rhs]=argn();
    if (rhs < 2) then
        [comb_lst, cant_comb] = comb2qbs(nqbs);
    else
        cant_comb = size(comb_lst, 'r');
```

```
    end
    CNot_lst = list();
    auxCnot = qb_cmp_CNOT_k(1);


    if (nqbs > 2) then
        auxCnot = qb_ut_tensorial(auxCnot, qb_cmp_I(nqbs-2));
    end

    for i=1:cant_comb
        inds = comb_lst(i,:);
        auxMS = qb_ut_cambia_filas(auxCnot, 2, inds(2));
        auxMS = qb_ut_cambia_filas(auxMS, 1, inds(1));
        CNot_lst($+1) = auxMS;
    end
endfunction




function QC = qb_QAN_QCell_create(nqbs, sts_out, comb_lst, Cnot_lst)


    printf('\n Cell_create');
    [lhs rhs]=argn();
    QC = list(nqbs);

    [max_ccm ind_max v_ccm] = CalculaCCM(sts_out);
    printf('\n CCM: %f', max_ccm);
    cant_Cnots = CantidadCNots(max_ccm);
```

```
if (cant_Cnots > 0) then
    s=rand()*(cant_Cnots-1);
    cant_Cnots = round(s)+1;
end



printf('\n cant_Cnots: %i', cant_Cnots);
U_Cnot = qb_cmp_I(nqbs);
if (cant_Cnots == 0) then
    ind_Cnot = [0 0];
else
    if (rhs < 3) then
      [comb_lst, cant_comb] = comb2qbs(nqbs);
    else
        cant_comb = size(comb_lst, 'r');
    end
    if (rhs < 4) then
      Cnot_lst = qb_QAN_CreateCNots(nqbs, comb_lst);
    end
    med_corr = OutqbsCorr(sts_out, comb_lst);
    [s, k]=gsort(med_corr);
    k = k(1:cant_Cnots);
    ind_Cnot = comb_lst(k(1:cant_Cnots), :);


    for i=cant_Cnots:-1:1
        U_Cnot = Cnot_lst(k(i))*U_Cnot;
    end
end
QC($+1) = ind_Cnot;
QC($+1) = U_Cnot;
QC($+1) = zeros(1,(3*nqbs));
```

```
    QC($+1) = zeros(1,(3*nqbs));
    QC($+1) = %nan;
endfunction

function [QC, UQC] = qb_QAN_QCell_U(QC)

    nqbs = QC(1);
    Ui = 1;
    pars_in = QC(4);
    for i=1:nqbs
        aux_pars = pars_in([(3*(i-1)+1):(3*i)]);
        Ui = qb_ut_tensorial(Ui, qb_QAN_Unit3(aux_pars));
    end
    Uo = 1;
    pars_in = QC(5);
    for i=1:nqbs
        aux_pars = pars_in([(3*(i-1)+1):(3*i)]);
        Uo = qb_ut_tensorial(Uo, qb_QAN_Unit3(aux_pars));
    end
    UQC = Uo*QC(3)*Ui;
    QC(6) = UQC;
endfunction

function [QC, inc_fid, fid_out, vec_grad] = qb_QAN_QCell_AdaptU1(QC, sts_in, sts_out, v
    [lhs rhs]=argn();
    nqbs = QC(1);
    if (rhs < 4) then
        vec_grad = zeros(1:(2*3*nqbs));
    end

    if (rhs < 5) then
```

```
    parsU1 = [1e-9 1e-5 1e-3 0];
end

if (rhs < 6) then
    calc_mean =%F;
end

dfs = parsU1(1);
dfp = parsU1(2);
f_p = parsU1(3);
f_m = parsU1(4);

if isnan(QC(6)) then
    [QC, UQC] = qb_QAN_QCell_U(QC);
else
    UQC = QC(6);
end
sts_calc = UQC*sts_in;


vec_pars = [QC(4) QC(5)];
cant_pars = length(vec_pars);

cant = size(sts_out, 'c');
vfid = zeros(1:cant);
for i=1:cant
    temp1 = qb_st_partialtrace(sts_out(:,i), [1:2]);
    temp1 = qb_st_vec2dens(temp1);
    temp2 = qb_st_partialtrace(sts_calc(:,i), [1:2]);
    temp2 = qb_st_vec2dens(temp2);
    [res, fid] = qb_corr_DistFid(temp1, temp2);
```

```
    vfid(i) = fid;
end

fid_in = mean(vfid);

vec_fid = zeros(1:cant_pars);

for i = 1:cant_pars
    aux_QC = QC;
    aux_pars = zeros(vec_pars);
    aux_pars(i) = dfs+abs(dfp*vec_pars(i));
    aux_pars = vec_pars + aux_pars;
    aux_QC(4) = aux_pars(1:(cant_pars/2));
    aux_QC(5) = aux_pars((cant_pars/2)+1:$);
    [aux_QC, UQC] = qb_QAN_QCell_U(aux_QC);

    sts_calc = UQC*sts_in;

    cant = size(sts_calc, 'c');
    vfid = zeros(1:cant);
    for j=1:cant
        temp1 = qb_st_partialtrace(sts_out(:,j), [1:2]);
        temp1 = qb_st_vec2dens(temp1);
        temp2 = qb_st_partialtrace(sts_calc(:,j), [1:2]);
        temp2 = qb_st_vec2dens(temp2);
        [res, fid] = qb_corr_DistFid(temp1, temp2);
        vfid(i) = fid;
    end

    vec_fid(i) = mean(vfid);
end
```

```
dif_fid = vec_fid - fid_in*ones(vec_fid);
dif_fid = dif_fid ./ ((dfs+abs(dfp*vec_pars)) +%eps);
aux_vec_grad = dif_fid;
aux_vec_grad = aux_vec_grad/(norm(aux_vec_grad)+%eps);

vec_grad = f_m*vec_grad + (1-f_m)*aux_vec_grad;
vec_grad = vec_grad/(norm(vec_grad)+%eps);
aux_pars = vec_pars + f_p*vec_grad;



aux_QC = QC;
aux_QC(4) = aux_pars(1:(cant_pars/2));
aux_QC(5) = aux_pars((cant_pars/2)+1:$);
[aux_QC, UQC] = qb_QAN_QCell_U(aux_QC);



sts_calc = UQC*sts_in;



cant = size(sts_out, 'c');
vfid = zeros(1:cant);
for i=1:cant
    temp1 = qb_st_partialtrace(sts_out(:,i), [1:2]);
    temp1 = qb_st_vec2dens(temp1);
    temp2 = qb_st_partialtrace(sts_calc(:,i), [1:2]);
    temp2 = qb_st_vec2dens(temp2);
    [res, fid] = qb_corr_DistFid(temp1, temp2);
    vfid(i) = fid;
end
```

```
    fid_out = mean(vfid);

    if (fid_out < fid_in) then
        fid_out = fid_in;
    else
        QC = aux_QC;
    end
     inc_fid = fid_out - fid_in;
endfunction

function [QC_out, vec_fid, med_vec_fid] =
    qb_QAN_QCell_AdaptU(QC, sts_in, sts_out, pars_opt)

    printf('\n Cell_Adapt');
    [lhs rhs]=argn();
    cant_sts = size(sts_in, 'c');

    if (rhs < 4) then
        pars_opt = [1e-9 1e-5 1e-3 0 1 1 5000 1 0.99 1e-10 1e-6 0.9];
    end

    dfs = pars_opt(1);
    dfp = pars_opt(2);
    fp_ini = pars_opt(3);
    fm = pars_opt(4);
    inc_fp = pars_opt(5);
    dec_fp = pars_opt(6);
    cant_max_it = pars_opt(7);
    porc_train = pars_opt(8);
    min_fid = pars_opt(9);
```

```
deriv_min = pars_opt(10);
diff_fid = pars_opt(11);
fid_filt = pars_opt(12);



med_fid = 0;
change_fid = 1;
cant_it = 1;
count_print = 0;
delta_print = round(cant_max_it/250);
vec_fid = zeros(1:cant_max_it);
med_vec_fid = zeros(1:cant_max_it);
vec_paso = zeros(1:cant_max_it);
fid_out = 1000;
fp = fp_ini;

QC_out = QC;
QC_out = qb_QAN_QCell_U(QC);
nqbs = QC_out(1);
v_g = zeros(1:(2*3*nqbs));
Nosalir = %T;
tic();
while Nosalir

    pars_U1 = [dfs dfp fp fm];
    cant_entrena = round(porc_train*cant_sts);
    ind_sts = round((cant_sts-1)*rand(1:cant_entrena))+1;
    if (cant_it == 10*floor(cant_it/10)) then
        cant_media =%F;
    else
        cant_media = %T;
```

```
    end
    [QC_out, inc_fid, fid_out, vec_grad] =
        qb_QAN_QCell_AdaptU1(QC_out, sts_in, sts_out, v_g, pars_U1, cant_media);

    vec_paso(cant_it) = fp;
    vec_fid(cant_it) = fid_out;
    med_fid = fid_filt*med_fid + (1-fid_filt)*fid_out;
    med_vec_fid(cant_it) = med_fid;


    if (cant_it > 1) then
        if ((vec_fid(cant_it) > vec_fid(cant_it-1)) & (fp < 100*fp_ini)) then
            fp = inc_fp*fp;
        end
        if ((vec_fid(cant_it) <= vec_fid(cant_it-1)) & (fp > 0.1*fp_ini)) then
            fp = dec_fp*fp;
        end
    end


if (cant_it >= 11) then
    aux_fid = med_vec_fid(cant_it-4:cant_it);
    change_fid = mean(abs(diff(aux_fid)));
    if (change_fid < deriv_min) then
        printf('\nSaliendo por cambio insuficiente');
    end
end;


     aux_salir = %F | ((med_fid > min_fid)&(fid_out > min_fid)
        &(abs(med_fid-fid_out)<diff_fid)) | (change_fid < deriv_min)|
```

```
            (cant_it == cant_max_it);
        Nosalir = ~aux_salir;
        cant_it = cant_it+1;
        count_print = count_print+1;
        if (count_print >= delta_print) then
            printf('\n Iteracion = %i, \t Fid = %f \t Paso = %f
                \t tiempo = %f', cant_it, fid_out, fp, toc());
            tic();
            count_print = 0;
        end
    end
    cant_it = cant_it-1;
    vec_fid = vec_fid(1:cant_it);
    med_vec_fid = med_vec_fid(1:cant_it);
    printf('\nFidelidad de salida: %f', vec_fid(cant_it));
endfunction



function QN = qb_QAN_QNet_create(nqbs)

    QN = list(nqbs);

    [comb_lst, cant_comb] = comb2qbs(nqbs);
    QN($+1) = comb_lst;

    CNot_lst = qb_QAN_CreateCNots(nqbs, comb_lst);
    QN($+1) = CNot_lst;
    QN($+1) = list();
    QN($+1) = %nan;
endfunction
```

```
function QN = qb_QAN_QNet_insertCell(QN, sts_out)
    nqbs = QN(1);
    comb_lst = QN(2);
    Cnot_lst = QN(3);
    QC = qb_QAN_QCell_create(nqbs, sts_out, comb_lst, Cnot_lst);
    lst_QCs = QN(4);
    lst_QCs($+1) = QC;
    QN(4) = lst_QCs;
endfunction

function [U_QN, QN] = qb_QAN_QNet_CalcU(QN)
    cant_cells = length(QN(4));
    if (cant_cells == 0) then
        U_QN = %nan;
    else
        aux_U = eye(2^QN(1),2^QN(1));
        for i=1:cant_cells
            [QC, UQC] = qb_QAN_QCell_U(QN(4)(i));

            aux_U = aux_U*UQC;
        end
        U_QN = aux_U
    end
    QN(5) = U_QN;
endfunction



function [QN, inc_fid, fid_out, vec_grad] = qb_QAN_QCells_AdaptUn_1s(QN, sts_in, sts_ou
    [lhs rhs]=argn();
    nqbs = QN(1);
```

```
lst_QC = QN(4)
cant_celdas = length(lst_QC);
ind_medio = 3*nqbs*cant_celdas;
ind_total = 2*ind_medio;

if (rhs < 4) then
    vec_grad = zeros(1:(ind_total));
end

if (rhs < 5) then
    parsU = [1e-9 1e-5 1e-3 0];
end

if (rhs < 6) then
    calc_mean =%F;
end

dfs = parsU(1);
dfp = parsU(2);
f_p = parsU(3);
f_m = parsU(4);

UQN = QN(5);
sts_calc_o = UQN*sts_in;


vec_pars_ent = [];
for i=1:cant_celdas
    vec_pars_ent = [vec_pars_ent, lst_QC(i)(4)]
end
```

```
vec_pars_sal = [];
for i=1:cant_celdas
    vec_pars_sal = [vec_pars_sal, lst_QC(i)(5)]
end

vec_pars = [vec_pars_ent vec_pars_sal];
cant_pars = length(vec_pars);



if calc_mean then

    fid_med = mean(abs(diag(sts_out'*sts_calc_o)));
    fid_in = fid_med;
else

    fid_in = abs(diag(sts_out'*sts_calc_o));
    ind_min = min(find(fid_in == min(fid_in)));
    fid_in = fid_in(ind_min);
end

vec_fid = zeros(1:cant_pars);

for i = 1:cant_pars
    aux_QN = QN;
    lst_QC = QN(4);
    aux_pars = zeros(vec_pars);
    aux_pars(i) = dfs+abs(dfp*vec_pars(i));
    aux_pars = vec_pars + aux_pars;
    pars_in = aux_pars(1:ind_medio);
    pars_out = aux_pars(ind_medio+1:$);
```

```
    for j = 1:cant_celdas
        lst_QC(j)(4) = pars_in((3*nqbs*(j-1)+1):3*nqbs*j);
        lst_QC(j)(5) = pars_out((3*nqbs*(j-1)+1):3*nqbs*j)
        [QC, UQC] = qb_QAN_QCell_U(lst_QC(j));
        lst_QC(j)(6) = UQC;
    end
    aux_QN(4) = lst_QC;
    [aux_UQN, aux_QN] = qb_QAN_QNet_CalcU(aux_QN);
    aux_QN(5) = aux_UQN;
    sts_calc = aux_UQN*sts_in;

    if calc_mean then
        aux_fid_in = mean(abs(diag(sts_out'*sts_calc)));
    else
        aux_fid_in = abs(diag(sts_out'*sts_calc));
        aux_ind_min = min(find(aux_fid_in == min(aux_fid_in)));
        aux_fid_in = aux_fid_in(aux_ind_min);
    end
    vec_fid(i) = aux_fid_in;
end

dif_fid = vec_fid - fid_in*ones(vec_fid);
dif_fid = dif_fid ./ ((dfs+abs(dfp*vec_pars)) +%eps);
aux_vec_grad = dif_fid;
aux_vec_grad = aux_vec_grad/(norm(aux_vec_grad)+%eps);

vec_grad = f_m*vec_grad + (1-f_m)*aux_vec_grad;
vec_grad = vec_grad/(norm(vec_grad)+%eps);
aux_pars = vec_pars + f_p*vec_grad;
```

```
aux_QN = QN;
lst_QC = QN(4);
pars_in = aux_pars(1:ind_medio);
pars_out = aux_pars(ind_medio+1:$);

for j = 1:cant_celdas
    lst_QC(j)(4) = pars_in((3*nqbs*(j-1)+1):3*nqbs*j);
    lst_QC(j)(5) = pars_out((3*nqbs*(j-1)+1):3*nqbs*j)
    [QC, UQC] = qb_QAN_QCell_U(lst_QC(j));
    lst_QC(j)(6) = UQC;
end
aux_QN(4) = lst_QC;
[aux_UQN, aux_QN] = qb_QAN_QNet_CalcU(aux_QN);
aux_QN(5) = aux_UQN;
sts_calc = aux_UQN*sts_in;



if calc_mean then

    fid_med = mean(abs(diag(sts_out'*sts_calc)));
    fid_out = fid_med;
else

    fid_out = abs(diag(sts_out'*sts_calc));
    ind_min = min(find(fid_out == min(fid_out)));
    fid_out = fid_out(ind_min);
end

if (fid_out < fid_in) then
    fid_out = fid_in;
```

```
    else
        QN = aux_QN;
    end
     inc_fid = fid_out - fid_in;
endfunction

function [QN_out, vec_fid, med_vec_fid] = qb_QAN_QCells_AdaptUn(QN, sts_in, sts_out, pa


    printf('\n QCell Adapt Un');
    [lhs rhs]=argn();
    cant_sts = size(sts_in, 'c');


    if (rhs < 4) then
        pars_opt = [1e-9 1e-5 1e-3 0 1 1 5000 1 0.99 1e-10 1e-6 0.9];
    end

    dfs = pars_opt(1);
    dfp = pars_opt(2);
    fp_ini = pars_opt(3);
    fm = pars_opt(4);
    inc_fp = pars_opt(5);
    dec_fp = pars_opt(6);
    cant_max_it = pars_opt(7);
    porc_train = pars_opt(8);
    min_fid = pars_opt(9);
    deriv_min = pars_opt(10)
    diff_fid = pars_opt(11)
    fid_filt = pars_opt(12);
```

```
med_fid = 0;
change_fid = 1;
cant_it = 1;
count_print = 0;
delta_print = floor(cant_max_it/100);
vec_fid = zeros(1:cant_max_it);
med_vec_fid = zeros(1:cant_max_it);
vec_paso = zeros(1:cant_max_it);
fid_out = 1000;
fp = fp_ini;

QN_out = QN;
[UQN, QN_out] = qb_QAN_QNet_CalcU(QN_out)
nqbs = QN(1)
nblqs = length(QN(4))
v_g = zeros(1:(2*3*nqbs*nblqs))
Nosalir = %T;
while Nosalir
    tic();

    pars_U1 = [dfs dfp fp fm];
    cant_entrena = round(porc_train*cant_sts);
    ind_sts = round((cant_sts-1)*rand(1:cant_entrena))+1;
    cant_media = %T;
    [QN_out, inc_fid, fid_out, vec_grad] = qb_QAN_QCells_AdaptUn_1s(QN_out, sts_in,

    vec_paso(cant_it) = fp;
    vec_fid(cant_it) = fid_out;
    med_fid = fid_filt*med_fid + (1-fid_filt)*fid_out; fidelidad
    med_vec_fid(cant_it) = med_fid;
```

```
    if (cant_it > 1) then
        if ((vec_fid(cant_it) > vec_fid(cant_it-1)) & (fp < 100*fp_ini)) then
            fp = inc_fp*fp;
        end
        if ((vec_fid(cant_it) <= vec_fid(cant_it-1)) & (fp > 0.1*fp_ini)) then
            fp = dec_fp*fp;
        end
    end


if (cant_it >= 11) then
    aux_fid = med_vec_fid(cant_it-4:cant_it);
    change_fid = mean(abs(diff(aux_fid)));
    if (change_fid < deriv_min) then
        printf('\nSaliendo por cambio insuficiente');
    end
end;


     aux_salir = %F | ((med_fid > min_fid)&(fid_out > min_fid)&(abs(med_fid-fid_out
    Nosalir = ~aux_salir;
    cant_it = cant_it+1;
    count_print = count_print+1;
    if (count_print >= delta_print) then
        printf('\n Iteracion = %i, \t Fid = %f \t Paso = %f \t tiempo = %f', cant_i
        count_print = 0;
    end
```

```
    end
    cant_it = cant_it-1;
    vec_fid = vec_fid(1:cant_it);
    med_vec_fid = med_vec_fid(1:cant_it);
    printf('\nFidelidad de salida: %f', vec_fid(cant_it));
endfunction


function [QN, net_fid, out_fid, med_fid, UQN] = qn_QAN_QNet_adapt(QN, sts_in_cell, sts_

    printf('\n Net_Adapt');
    min_fid = pars_opt_cell(9);
    net_fid=[];
    med_fid=[];
    out_fid=[];
    cant_max_cells = pars_opt_net(1);

    h=scf();

    cant_cells = 1;
    printf('\n Creando celda: %i', cant_cells);
    tic();
    QN = qb_QAN_QNet_insertCell(QN, sts_out_cell);
    printf('\t tiempo = %f', toc());
    lst_QCs = QN(4);
    aux_QC = lst_QCs(cant_cells);
    [aux_QC, vec_fid, med_vec_fid] = qb_QAN_QCell_AdaptU(aux_QC, sts_in_train, sts_out_
    lst_QCs(cant_cells) = aux_QC;
    QN(4) = lst_QCs;
    aux_UQC = aux_QC(6);
    QN(5) = aux_UQC;
```

```
fid = vec_fid($);
net_fid=fid;
med_fid = [med_fid med_vec_fid];
out_fid =[out_fid vec_fid];

clf(h);
scf(h);
plot(out_fid);

while((fid < min_fid) & (cant_cells < cant_max_cells))
    cant_cells = cant_cells + 1;
    printf('\n Creando celda: %i', cant_cells);
    sts_create = QN(5)'*sts_out_cell;
    sts_train = QN(5)'* sts_out_train;
    QN_aux = QN
    tic();
    QN = qb_QAN_QNet_insertCell(QN, sts_create);
    printf('\t tiempo = %f', toc());
    lst_QCs = QN(4);
    aux_QC = lst_QCs(cant_cells);
    [aux_QC, vec_fid, med_vec_fid] =
        qb_QAN_QCell_AdaptU(aux_QC, sts_in_train, sts_train, pars_opt_cell);
    lst_QCs(cant_cells) = aux_QC;
    QN(4) = lst_QCs;
    aux_UQC = aux_QC(6);
    QN(5) = QN(5)*aux_UQC;
    med_fid=[med_fid med_vec_fid]
    out_fid = [out_fid vec_fid];

    clf(h);
    scf(h);
```

```
        plot(out_fid);

        [QN, vec_fid, med_vec_fid] = qb_QAN_QCells_AdaptUn(QN,
        sts_in_train, sts_out_train, pars_opt_cell)
        fid = vec_fid($);
        med_fid=[med_fid med_vec_fid]
        out_fid = [out_fid vec_fid];
        net_fid=fid;

        clf(h);
        scf(h);
        plot(out_fid);
    end

    UQN = QN(5);
endfunction

function [vec_pars, vec_pars_ent, vec_pars_sal] = qn_QAN_QNet_ArgumExtract(QN)
    lst_QC = QN(4);
    cant_celdas = length(lst_QC);

    vec_pars_ent = [];
    for i=1:cant_celdas
        vec_pars_ent = [vec_pars_ent, lst_QC(i)(4)]
    end

   vec_pars_sal = [];
    for i=1:cant_celdas
        vec_pars_sal = [vec_pars_sal, lst_QC(i)(5)]
    end
```

```
    vec_pars = [vec_pars_ent vec_pars_sal];
endfunction

function QN = qn_QAN_QNet_ArgumUpdate(QN, vec_pars, vec_pars_sal)
    [lhs, rhs] = argn();
    nqbs = QN(1);
    lst_QC = QN(4);
    cant_celdas = length(lst_QC);
    ind_medio = 3*nqbs*cant_celdas;
    ind_total = 2*ind_medio;

    if (rhs < 3)
        if length(vec_pars) <> ind_total then
            error('Dimensión incorrecta del vector de parámetros');
        end
        vec_pars_sal = vec_pars(ind_medio+1:$);
        vec_pars = vec_pars(1:ind_medio);
    else
        if ((length(vec_pars)<> ind_medio) | (length(vec_pars_sal)<> ind_medio)) then
            error('Dimensión incorrecta de los vectores de parámetros');
        end
    end

    for j = 1:cant_celdas
        lst_QC(j)(4) = vec_pars((3*nqbs*(j-1)+1):3*nqbs*j);
        lst_QC(j)(5) = vec_pars_sal((3*nqbs*(j-1)+1):3*nqbs*j);
        [QC, UQC] = qb_QAN_QCell_U(lst_QC(j));
        lst_QC(j)(6) = UQC;
    end
    QN(4) = lst_QC;
    [UQN, QN] = qb_QAN_QNet_CalcU(QN);
```

```
endfunction
```