UNIVERSIDADE DE SANTIAGO DE COMPOSTELA


ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

# Revisión y análisis de modelos de inteligencia artificial cuántica

*Autor/a:*
**Manuel Timiraos López**

*Tutores:*
**Tomás Fernández Pena**
**Alberto J. Bugarín Diz**

**Grado en Ingeniería Informática**

**Julio 2024**

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática.

# Resumen

La computación cuántica es un campo prometedor hoy en día, que pretende revolucionar las ciencias de la computación aprovechando las ventajas de la mecánica cuántica. Su combinación con el campo de la inteligencia artificial da lugar al conocido como *Quantum Machine Learning*. En este trabajo se desarrolla una introducción a sus técnicas y modelos. Tras una revisión bibliográfica, se selecciona un modelo concreto: un clasificador basado en una novedosa técnica conocida como *data re-uploading*. Se implementará dicho modelo utilizando la biblioteca estándar Qiskit y se llevará a cabo una experimentación para evaluarlo.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

El aprendizaje automático cuántico (*Quantum Machine Learning*, QML) combina la inteligencia artificial con la computación cuántica, abriendo nuevas fronteras en la resolución de problemas complejos. Se trata de un área activa de investigación hoy en día.

En este primer capítulo se presentarán los objetivos de este trabajo de fin de grado. Además, se incluye una segunda sección en la que se explica la estructura del resto de la presente memoria.

## 1.1. Objetivos del trabajo

El primer objetivo de este trabajo es llevar a cabo, como indica el título, una revisión sistemática de las aplicaciones de la computación cuántica al campo de la inteligencia artificial. En concreto, nos referimos al campo de *Quantum Machine Learning*. Para ello es necesaria una primera introducción en el mundo de la computación cuántica, sobre la que recaerá una parte muy importante del tiempo dedicado al trabajo.

Tras la revisión, el siguiente objetivo consiste en la selección de un modelo concreto de especial interés, para su posterior estudio. El modelo en cuestión ha resultado ser el casificador de un solo cúbit basado en la técnica del *data re-uploading* [11]. Nos hemos fijado en él debido a su interés teórico, pues su arquitectura permite compararlo con un modelo clásico de aprendizaje automático.

El objetivo final es la implementación del modelo empleando la biblioteca Qiskit, que representa un estándar en compútación cuántica. Con esto se pretende acercar la implementación a una ejecución en hardware cuántico real, que sería posible gracias a Qiskit.

## 1.2. Organización de la memoria

Describimos a continuación la estructura del resto de la memoria. Para comenzar, en el capítulo 2 se desarrolla una introducción a los conceptos fundamentales de la computación cuántica. Esto es imprescindible para entender el resto del trabajo. Tras esto, se describe el campo del QML, incidiendo concretamente en un tipo de modelos: las redes neuronales cuánticas. Se les ha dado especial importancia, para entender cómo surge de modo natural el modelo propuesto en [11] (que explicamos en el siguiente capítulo).

El capítulo 3 combina dos cosas diferentes. En una primera sección se comentan los materiales que se han empleado para implementar y evaluar el modelo estudiado. La segunda sección consiste en la descripción teórica de dicho modelo; el clasificador de un solo cúbit basado en *data re-uploading* [11].

En el capítulo 4 se lleva a cabo la explicación de la implementación realizada, junto con la descripción de la experimentación que se ha diseñado para evaluar el modelo. Se discuten también aquí los resultados obtenidos.

Finalmente, el capítulo 5 está dedicado a sintetizar las conclusiones extraídas de la discusión de resultados. Además, se comentan posibles continuaciones del trabajo realizado.

Como anexos, se incluyen los manuales necesarios para la reproducción de los experimentos llevados a cabo en el trabajo.

# Capítulo 2

# Estado del conocimiento: QML

Este capítulo está dedicado a la contextualización del trabajo realizado. La dividiremos en dos fases: una introducción general a la computación cuántica y una descripción del campo más específico del *Quantum Machine Learning*. El libro [3] nos servirá como guía general para el desarrollo que expondremos a continuación.

## 2.1. Introducción a la computación cuántica

La computación cuántica es un campo de estudio muy prometedor hoy en día. Se trata de una nueva rama de las ciencias de la computación que proporciona nuevos paradigmas haciendo uso de los principios de la mecánica cuántica. Como es de esperar, los conceptos de este ámbito pueden resultar complejos en un primer momento. Sin embargo, merece la pena adentrarse en este mundo para sorprenderse con las interesantes ideas que han surgido y continúan surgiendo en los últimos años.

La idea general en computación cuántica consiste en aprovechar fenómenos cuánticos como la superposición o el entrelazamiento, con el objetivo de conseguir ventajas computacionales. Pero a la hora de llevar estos conceptos a un marco computacional aparece la necesidad de organizarlos y precisar bien el modelo con el que estamos trabajando.

En este contexto han surgido diferentes modelos computacionales. Por ejemplo, uno de los más conocidos es el de la computación cuántica adiabática (ver

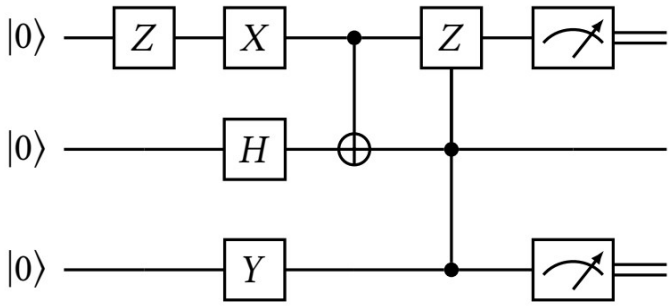



Figura 2.1: Ejemplo de circuito cuántico.

[3]), con aplicaciones a la resolución de problemas de optimización combinatoria. Se basa en trabajar explícitamente con el hamiltoniano del sistema cuántico en cuestión, haciéndolo evolucionar de manera adiabática hasta la solución del problema de optimización. Sin embargo, en este trabajo vamos a dejar de lado este enfoque. En cambio, nos centraremos en el conocido como modelo de circuito cuántico, que describimos a continuación

### 2.1.1. Circuito cuántico

El enfoque de computación cuántica más extendido es el modelo de circuito cuántico. En cierto sentido, este modelo se construye como analogía a los circuitos combinacionales clásicos. La diferencia fundamental radica en que, en lugar de trabajar con bits clásicos, aquí se opera con cúbits. En cierto sentido muy vago, se puede pensar en ellos como una extensión de los bits clásicos que, además de poder encontrarse en los estados 0 y 1, también se pueden encontrar en una "superposición" de ambos estados. Más adelante detallaremos a qué nos referimos con este término.

A grandes rasgos, la estructura de un circuito cuántico sería la siguiente. Se dispone de una serie de cúbits, inicialmente en el estado 0. Inspirados por las puertas lógicas, a continuación, se aplican a los qúbits las conocidas como puertas cuánticas. Finalmente, se realiza una medición de los qúbits. Esto consiste en obligar a cada cúbit, a pesar de encontrarse en una posible superposición de 0 y 1, a "colapsar" de manera probabilística a uno de los dos valores clásicos.

En la Figura 2.1, obtenida de [3], se muestra un ejemplo de circuito. Cada línea horizontal representa la evolución de un cúbit. Al comienzo, como se puede observar, cada cúbit se encuentra en el estado 0, denotado por $|0\rangle$. Los distintos

cuadrados con una letra en su interior representan las puertas cuánticas. Un detalle relevante que puede chocar en un primer momento es la cantidad de puertas diferentes que se aplican a un solo cúbit; algo que en los circuitos clásicos solo sucede con la puerta NOT. En el siguiente apartado veremos que esto tiene sentido, debido a la complejidad que presenta un solo cúbit. Por último, los cuadrados cuyo interior contiene una flecha junto a una curva representan la medición final que se realiza sobre el cúbit, dando como resultado un valor clásico.

No hemos comentado lo que significan las líneas verticales. Estas están relacionadas con puertas que involucran varios cúbits, y producen entrelazamiento entre ellos. En este trabajo no nos centraremos especialmente en estas cuestiones.

### 2.1.2. Cúbits

Hasta ahora hemos estado hablando sobre cúbits, como pieza fundamental para los circuitos cuánticos, pero no hemos explicado todavía en qué consisten exactamente.

Su nombre viene de juntar las palabras "bit cuántico". Como ya hemos comentado, estos nuevos objetos se pueden encontrar en una superposición de los estados 0 y 1, que denotaremos por $|0\rangle$ y $|1\rangle$. Concretamente, podemos expresar un estado genérico del cúbit mediante una combinación lineal de la forma

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,,$$

donde $a, b \in \mathbb{C}$ verificando $|a|^2 + |b|^2 = 1$, conocida como condición de normalización. De esta forma, si hacemos la siguiente identificación

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

se obtiene que $|\psi\rangle$ es un vector de dos dimensiones complejas que tiene módulo igual a 1:

$$a \begin{pmatrix} 1 \\ 0 \end{pmatrix} + b \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{C}^2.$$

Así, hemos visto que los estados del cúbit viven en un espacio vectorial complejo de dimensión 2, del cual $|0\rangle$ y $|1\rangle$ forman una base ortonormal, conocida como base computacional. Habitualmente, se refiere a este como espacio de Hilbert. Sin embargo, se trata del mismo concepto al trabajar con un espacio de dimensión finita.

Falta todavía dar sentido al aspecto probabilístico de los cúbits que habíamos mencionado. Ahora ya tenemos los ingredientes para concretar esta idea. Resulta

que la probabilidad de que, al realizar una medición, el cúbit colapse al estado $|0\rangle$, viene dada por el valor $|a|^2$. De manera análoga, $|b|^2$ es la probabilidad de que colapse al estado $|1\rangle$. La condición $|a|^2 + |b|^2 = 1$ garantiza la consistencia de este fenómeno, al asegurar que la probabilidad total es igual a 1.

### 2.1.3. Notación de Dirac

Hemos usado hasta ahora el símbolo $|\cdot\rangle$, conocido como *ket*, para representar vectores "columna". No obstante, esta notación pierde su sentido si no presentamos también su contraparte: el *bra*, denotado por $\langle\cdot|$. Este se usará para denotar vectores "fila".

Dado un vector $|\psi\rangle$, este tendrá un bra asociado, que vendrá dado por su adjunto, o traspuesto conjugado. Es decir, hacemos la asociación:

$$|\psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix} \longmapsto |\psi\rangle^\dagger = \langle\psi| = \begin{pmatrix} \overline{a} & \overline{b} \end{pmatrix}$$

Así, combinando el bra y el ket, se obtiene el *bra-ket* $\langle\cdot|\cdot\rangle$, una forma de denotar al producto escalar de $\mathbb{C}^2$. Veamos como funciona. Dados dos estados $|\psi_1\rangle = a\,|0\rangle + b\,|1\rangle$ y $|\psi_2\rangle = c\,|0\rangle + d\,|1\rangle$, obtenemos:

$$\langle\psi_1|\psi_2\rangle = \begin{pmatrix} \overline{a} & \overline{b} \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \overline{a}c + \overline{b}d.$$

### 2.1.4. Esfera de Bloch

Hasta ahora hemos estado trabajando con los estados cuánticos de manera puramente algebraica. En este apartado, presentaremos un modelo geométrico, conocido como la esfera de Bloch, que nos servirá para visualizar los estados del cúbit.

Se han presentado los estados como vectores de $\mathbb{C}^2$. Como cada número complejo está compuestos por dos coordenadas reales, los estados cuánticos requieren en principio de 4 coordenadas reales, resultando imposible su visualización en 3 dimensiones. Afortunadamente, la condición $|a|^2 + |b|^2 = 1$, anula uno de los 4 grados de libertad. Veamos por qué. Consideremos un estado $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$ y expresemos $a$ y $b$ en coordenadas polares:

$$a = r_1 e^{i\alpha_1}, \qquad b = r_2 e^{i\alpha_2}.$$

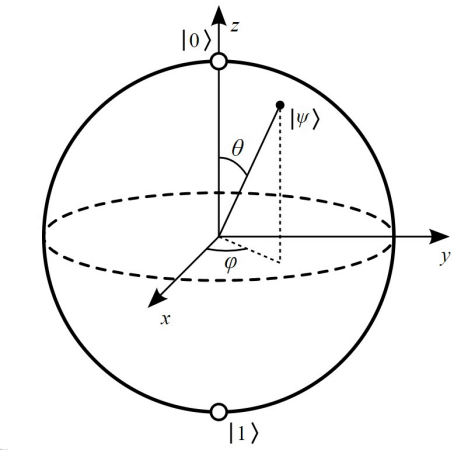



Figura 2.2: Esfera de Bloch.

Como $r_1^2 + r_2^2 = |a|^2 + |b|^2 = 1$ y $0 \leq r_1, r_2 \leq 1$, existirá un ángulo $\gamma \in [0, \pi/2]$ tal que $\cos\gamma = r_1$ y $\sin\gamma = r_2$. Alternativamente, existirá un ángulo $\theta \in [0, \pi]$ tal que $\cos\frac{\theta}{2} = r_1$ y $\sin\frac{\theta}{2} = r_2$. En breves tendrá sentido este último artificio. De momento, podemos expresar nuestro estado de la siguiente manera:

$$|\psi\rangle = \cos\frac{\theta}{2}e^{i\alpha_1}|0\rangle + \sin\frac{\theta}{2}e^{i\alpha_2}|1\rangle\,.$$

Aplicaremos ahora otro argumento que nos permitirá restar otro grado de libertad más. Si multiplicamos el vector $|\psi\rangle$ por una constante $c \in \mathbb{C}$, el estado resultado será físicamente indistinguible de $|\psi\rangle$. Dejaremos la justificación de esto para el siguiente apartado. Considerando $c = e^{-i\alpha_1}$, obtenemos

$$|\psi\rangle \equiv e^{-i\alpha_1}|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}e^{i\varphi}|1\rangle\,,$$

donde $\varphi = \alpha_2 - \alpha_1$. Así, podemos representar $|\psi\rangle$ simplemente con los dos ángulos $\theta \in [0, \pi]$ y $\varphi \in [0, 2\pi]$. Es esta situación, podemos utilizar coordenadas esféricas para visualizar el estado en la superficie de una esfera, tal y como se muestra en la Figura 2.2, extraída de [5]. Se conoce como esfera de Bloch, y será de utilidad para interpretar la actuación de las puertas cuánticas que explicaremos a continuación.

### 2.1.5. Puertas cuánticas

Todavía nos falta formalizar un concepto muy importante: las puertas cuánticas. En concreto, nos centraremos en las puertas que actúan sobre un único cúbit.

Aunque pueda parecer extraño, las puertas cuánticas se corresponden con matrices unitarias, es decir, matrices complejas $U$ cuya inversa coincide con su adjunta:

$$U^\dagger U = UU^\dagger = I.$$

Estas matrices cumplen $|\det(U)| = 1$. Esto, en esencia, se traduce en que cualquier estado, tras aplicarle una o varias de estas matrices, seguirá verificando la condición de normalización, y por tanto la probabilidad total seguirá siendo 1. Este hecho garantiza la consistencia del modelo. Para indagar en los detalles de este razonamiento, se puede consultar el libro [15].

Las puertas cuánticas también presentan una interpretación geométrica, en términos de la esfera de Bloch. Resulta que cada matriz unitaria se corresponde con una rotación de la esfera de Bloch, con un eje de rotación y ángulo determinados. Los detalles relativos a este fenómeno se pueden consultar en el libro [10], que es una de las referencias más populares en el ámbito de la computación cuántica.

A continuación, veremos algunos ejemplos importantes de puertas cuánticas. Comenzaremos por la generalización de la puerta NOT clásica: la puerta $X$. Consiste en la matriz unitaria que intercambia los estados $|0\rangle$ y $|1\rangle$. Su expresión es la siguiente.

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

Esta es una de las tres matrices conocidas como matrices de Pauli, de gran importancia en la mecánica cuántica. Las otras dos son las que se muestran a continación:

$$\sigma_Y \equiv Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_Z \equiv Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Los nombres de estas puertas no son casuales. Cada una de ellas actúa en la esfera de Bloch como una rotación de $180^{\text{o}}$ sobre el eje correpondiente a su nombre. Presentamos ahora una de las puertas cuánticas más famosas: la puerta Hadamard.

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

Esta puerta sirve para construir de forma sencilla un estado de superposición con igual probabilidad de colapso a 0 que a 1. Veámoslo, haciendo actuar esta matriz sobre el estado $|0\rangle$, que hebitualmente que es el estado inicial de los cúbits en los circuitos cuánticos:

$$H|0\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle .$$

Hemos visto que con las matrices de Pauli es posible aplicar rotaciones de $180^{\text{o}}$ en la esfera de Bloch. Para terminar este apartado, exponemos las matrices que permiten rotar la esfera:

$$R_X(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \quad R_Y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \quad R_Z(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

Estas matrices se conocen como puertas rotacionales y son ejemplos de puertas parametrizadas. En el apartado siguiente veremos una versión más compleja, dentro de esta misma familia; muy importante para este trabajo.

### 2.1.6. Puerta universal

A partir de las puertas rotacionales se puede construir una puerta parametrizada más compleja, conocida como puerta universal. A continuación, se muestra su definición.

$$U(\theta, \varphi, \lambda) \equiv R_Z(\varphi)R_Y(\theta)R_Z(\lambda) = \begin{pmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\varphi}\sin\frac{\theta}{2} & e^{i(\varphi+\lambda)}\cos\frac{\theta}{2} \end{pmatrix}$$

Esta puerta depende de tres ángulos, y es capaz de generar cualquier otra puerta cuántica de un solo cúbit. En el libro [3] se puede consultar más información sobre esta puerta. Si se quiere hilar más fino con la teoría subyacente, conviene consultar [10].

### 2.1.7. Circuitos multi-cúbit

Hasta el momento, hemos estado trabajando únicamente con circuitos de un solo cúbit. La potencia de la computación cuántica proviene de la utilización de varios cúbits al mismo tiempo, y llevando a cabo el fenómeno conocido como entrelazamiento.

Como hemos explicado, los estados de un cúbit se representan con vectores de $\mathbb{C}^2$, espacio del cual los estados $|0\rangle$ y $|1\rangle$ forman una base. Para trabajar con

$n$ cúbits se construye un nuevo espacio que tendrá por base el conjunto:

$$\begin{gathered}|0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle\,, \\ |0\rangle \otimes |0\rangle \otimes \cdots \otimes |1\rangle\,, \\ \vdots \\ |1\rangle \otimes |1\rangle \otimes \cdots \otimes |1\rangle\,.\end{gathered}$$

El símbolo $\otimes$ denota el producto tensorial de dos vectores. En esencia, sin entrar en muchos detalles, se trata de una operación bilineal que sirve para generar un nuevo espacio vectorial combinando las bases de varios espacios de partida. Por ejemplo, en el caso de vectores de dimensión 2 funciona así:

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \otimes \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ a_2 \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ a_1 b_2 \\ a_2 b_1 \\ a_2 b_2 \end{pmatrix}$$

De este modo, en la multiplicación de vectores de la base computacional sucederá siempre algo de este estilo:

$$|0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

La base que describimos anteriormente también se suele denotar de las siguientes maneras:

$$|00\ldots0\rangle\,, |00\ldots1\rangle\,, \ldots, |11\ldots1\rangle \equiv |0\rangle\,, |1\rangle\,, \ldots, |2^n-1\rangle\,.$$

De este modo, un estado genérico se escribirá:

$$|\psi\rangle = a_0\,|0\rangle + a_1\,|1\rangle + \cdots + a_{2^n-1}\,|2^n-1\rangle\,,$$

cumpliendo la nueva condición de normalización: $\sum_{l=0}^{2^n-1} |a_l|^2 = 1$. En este caso, cada $|a_l|^2$ representa la probabilidad de obtener el estado $|l\rangle$ al realizar la medición simultánea de todos los cúbits del circuito.

¿Qué sucede en este caso con las puertas cuánticas? Como los estados están ahora representados por vectores de dimensión $2^n$, las puertas necesariamente serán matrices unitarias $2^n \times 2^n$. Veamos un ejemplo muy típico de puerta cuántica de dos cúbits: La puerta CNOT. Al ser 2 el número de cúbits, la matriz correspondiente ha de ser $4 \times 4$. En concreto:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

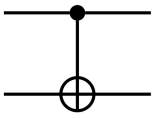


Figura 2.3: Representación de la puerta CNOT.

Fácilmente, se puede comprobar que esta puerta actúa sobre la base de la siguiente forma:

$$\text{CNOT}\,|00\rangle = |00\rangle\,,\ \text{CNOT}\,|01\rangle = |01\rangle\,,\ \text{CNOT}\,|10\rangle = |11\rangle\,,\ \text{CNOT}\,|11\rangle = |10\rangle\,.$$

Su comportamiento consiste en intercambiar los estados 0 y 1 del segundo cúbit (o sea, el de la derecha) solo cuando el estado del primero es 1. Es decir, el primero controla una NOT sobre el segundo; de ahí el nombre *controlled*-NOT, o CNOT. En la Figura 2.3, obtenida de [3], se muestra el diagrama correspondiente a esta puerta.

### 2.1.8. Entrelazamiento

Debido a su importancia, dedicamos un breve apartado a uno de los pilares fundamentales de la computación cuántica: el fenómeno del entrelazamiento.

Se dirá que un estado $|\psi\rangle$ es un estado producto si puede ser expresado como producto de otros estados $|\psi_1\rangle$ y $|\psi_2\rangle$. Es decir,

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle\,.$$

En caso contrario, se dirá que $|\psi\rangle$ está entrelazado. El siguiente es un ejemplo de estado producto. La clave está en que podemos factorizar fácilmente el estado $|0\rangle$, como se muestra a continuación.

$$\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) \otimes |0\rangle\,.$$

Al contrario, el estado $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ no puede ser factorizado de manera similar, siendo así un estado entrelazado. En la Figura 2.4 se muestra como construirlo con una puerta Hadamard y una CNOT.

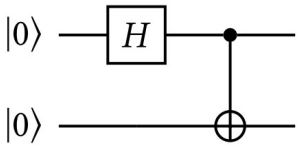



Figura 2.4: Circuito básico generador de entrelazamiento.

## 2.2. *Quantum Machine Learning*

En la primera sección de este capítulo hemos introducido varios de los conceptos fundamentales de la computación cuántica. Tras haber proporcionado esta base, procedemos ahora a adentrarnos en un campo más específico: el *Quantum Machine Learning* (QML).

### 2.2.1. Ideas generales

Como su nombre indica, a grandes rasgos el QML es una rama que surge de combinar ideas del aprendizaje automatico con la computación cuántica. No obstante, esta combinación puede surgir de formas esencialmente diferentes. Dedicamos este apartado a exponer la clasificación general de modelos de QML propuesta por Schuld y Petruccione [14]. Se trata de una división en cuatro grandes familias, en función de la naturaleza, clásica o cuántica, tanto de los datos como de los algoritmos utilizados.

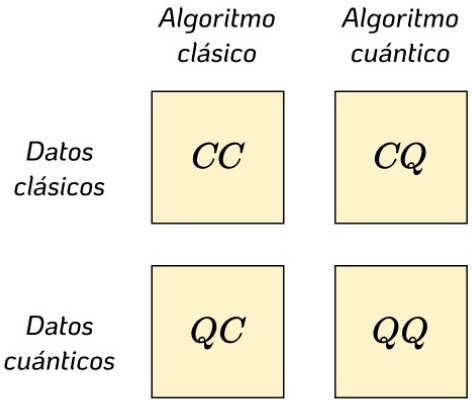



Figura 2.5: Las cuatro grandes familias del QML.

En la Figura 2.5 se muestra de forma esquemática la clasificación. A continuación, explicamos en qué consiste cada categoría.

- **CC**. En primer lugar, podríamos considerar parte del aprendizaje automático cuántico todas las técnicas de aprendizaje automático clásicas inspiradas en la cuántica. En este caso, tanto los datos como los ordenadores son clásicos, pero hay cierto toque cuántico involucrado en el proceso. Dado que no se utilizan ordenadores cuánticos reales en este enfoque, no entraremos en mucho detalle.

- **QC**. En ocasiones, surgen algoritmos de aprendizaje automático clásico que procesan datos cuánticos. Por ejemplo, podríamos pensar en algoritmos que trabajan con datos generados por procesos cuánticos. En este enfoque, el aprendizaje automático no se tiene en cuenta como un objetivo, sino como una herramienta.

- **CQ**. Llegamos a la categoría que más nos interesa. En este caso, se trata de proponer modelos de aprendizaje automáticos basados esencialmente en computación cuántica, que trabajan con datos clásicos. Aquí se enmarcan tanto las máquinas de vector soporte cuánticas como las redes neuronales cuánticas [3], además del propio modelo estudiado en este trabajo [11].

- **QQ**. Existe una última categoría muy ambiciosa que pretende construir modelos cuánticos que, a su vez, traten datos cuánticos. Por ejemplo, la idea podría consistir en alimentar el modelo cuántico directamente con estados cuánticos. Este es un campo prometedor, pero la tecnología necesaria todavía no está suficientemente desarrollada.

Como hemos comentado, nos vamos a ubicar dentro de la familia CQ: datos clásicos y algoritmos cuánticos. Esta categoría todavía puede subdividirse en dos. El modelo en cuestión en cualquier caso estará basado en computación cuántica. Sin embargo, el proceso de optimización puede llevarse a cabo tanto de forma clásica como cuántica. La optimización cuántica [1] podría llegar a superar a la clásica, pero el hardware actual no permite su puesta en práctica. De este modo, nos centraremos en la subcategoría de CQ que consiste en emplear modelos cuánticos con técnicas de optimización clásicas. Los algoritmos de este tipo se denomina híbridos. Este enfoque es el ideal para los dispositivos NISQ (*Noisy Intermediate-Scale Quantum*). Estos son ordenadores cuánticos con un número moderado de qubits, capaces de realizar cálculos útiles a pesar de la presencia de ruido y errores.

### 2.2.2. Redes neuronales cuánticas

Nos centraremos ahora en una familia más concreta de modelos dentro del mundo del QML: las redes neuronales cuánticas [3] o QNNs, por sus siglas en inglés. Profundizaremos algo más en ellas ya que, dentro del QML, son los modelos más similares al que hemos estudiado e implementado en este trabajo.

En lugar de describir directamente las QNNs, primero vamos a abstraer de manera sencilla el funcionamiento de las redes neuronales clásicas. Podemos dividir la propagación hacia delante de una red neuronal en las siguientes etapas:

- **Preparación**. Consiste en llevar a cabo ciertas transformaciones sobre los datos de entrada, como pueden ser la normalización o el escalado.
- **Procesamiento**. Esta fase consiste en introducir los datos a través de la red neuronal. Se lleva a cabo una transformación de los datos que depende de ciertos parámetros optimizables.
- **Salida**. Finalmente, se devuelve la salida obtenida en la capa final.

Este esquema nos servirá para diseñar un modelo cuántico "análogo" a las redes neuronales. En primer lugar, la preparación consistirá en introducir de algún modo los datos clásicos en un circuito cuántico. Esto se llevará a cabo por medio de lo que se conoce como mapa de características. Esta es una pieza fundamental de las *Quantum Support Vector Machines*, que se explica en detalle en [3]. No entraremos en profundidad en este concepto, pero sí describimos a continuación un ejemplo muy sencillo, conocido como *angle encoding*. Si denotamos por $(x_1, \ldots, x_n) \in \mathbb{R}^n$ a nuestros datos de entrada clásicos, podemos "introducirlos" en un circuito de $n$ cúbits simplemente aplicándole una puerta rotacional a cada cúbit $k$, con un ángulo $x_k$. En la Figura 2.6 se muestra el circuito correspondiente.

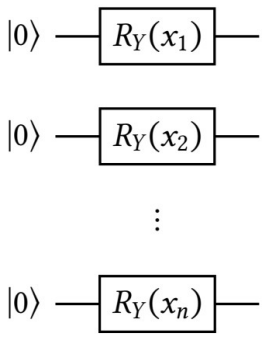



Figura 2.6: *Angle encoding.*

Continuando con la abstracción, para reproducir la fase de procesamiento de manera cuántica tan solo tenemos que aplicar un circuito que dependa de parámetros optimizables. Se pueden diseñar arquitecturas de este tipo estructuradas en capas, recordando en cierta manera a las redes neuronales clásicas. Este tipo de circuitos se conoce como *formas variacionales*. En la Figura 2.7 se muestra su esquema general. Básicamente están compuestas de circuitos $G_j$ dependientes de parámetros $\vec{\theta}_j$ intercalados con circuitos $U^j_{ent}$ cuyo objetivo es crear entrelazamiento entre los distintos cúbits. A modo de ejemplo, en la Figura 2.8 se muestra una forma variacional concreta, conocida como *Two-local*.

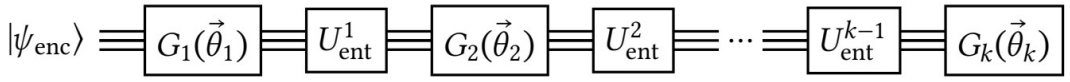



Figura 2.7: Esquema general de una forma variacional.

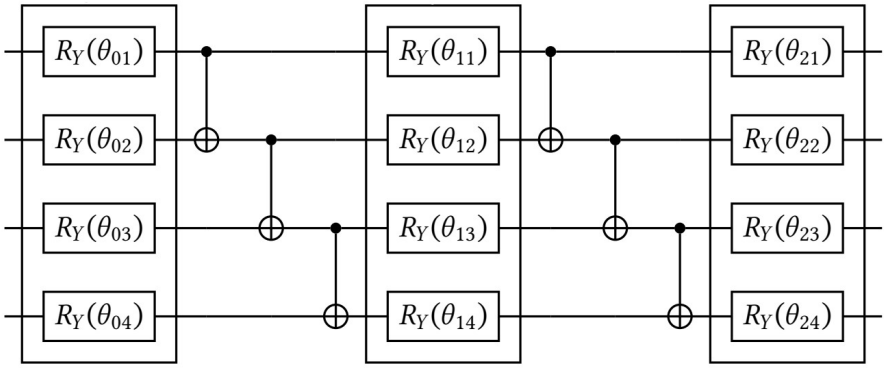



Figura 2.8: Arquitectura de la forma variacional *Two-local*.

Por último, en relación a la etapa de salida del modelo, debemos ser capaces de obtener un valor real al final del circuito. Esto, como no pude ser de otra manera, se consigue por medio de una medición. Por ejemplo, se puede fijar uno de los cúbits, y, ejecutando múltiples veces el circuito, aproximar la probabilidad de obtener el estado 0.

Ya hemos terminado de concretar los tres componentes que definen a un QNN: un mapa de características para introducir los datos clásicos, una forma variacional con parámetros optimizables y una medición final. La Figura 2.9 muestra esquemáticamente esta arquitectura general. Todas las figuras de este apartado se han obtenido de [3].

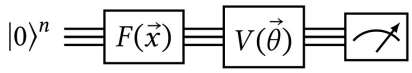



Figura 2.9: Esquema general de una QNN.

# Capítulo 3

# Materiales y métodos

En este capítulo combinamos la descripción teórica del modelo estudiado: clasificador basado en *data re-uploading* [11], junto a los materiales que se han empleado para implementarlo y evaluarlo.

## 3.1. Materiales

Esta sección está dedicada a la exposición de los materiales informáticos empleados para llevar a cabo este trabajo. En concreto, describiremos el hardware empleado para realizar los experimentos y, por otra parte, comentaremos las bibliotecas de Python utilizadas en el código que se ha desarrollado.

### 3.1.1. Recursos Hardware

Comenzaremos describiendo las características del sustento hardware de nuestras pruebas. Dada la naturaleza general del entrenamiento de modelos de aprendizaje automático, la manera más cómoda de trabajar consiste en ejecutar los entrenamientos en un servidor con buena capacidad de cómputo. En este caso, se ha utilizado uno de los servidores de computación GPGPU del CiTIUS. En concreto, el servidor *ctgpgpu4*. Se trata de un servidor PowerEdge R730, con dos procesadores Intel Xeon E52623v4 y 128 GB de memoria RAM. También dispone de dos tarjetas Nvidia GP102GL [Tesla P40]. No obstante, no se ha hecho uso de ellas. Para nuestro caso específico, el procesador ha sido suficiente para llevar a

cabo los entrenamientos en un tiempo razonable. Como sistema operativo, este servidor tiene la distribución de Linux denominada CentOS (versión 7.4).

Para la conexión a la red interna del CiTIUS ha sido necesario utilizar el servicio de VPN que proporciona el centro. Esto se ha combinado con la extensión Remote - SSH de Visual Studio Code para desarrollar el proyecto directamente en el servidor, trabajando así de manera más cómoda y eficiente.

### 3.1.2. Bibliotecas

Ahora comentaremos las bibliotecas que han sido necesarias en este trabajo. Dedicamos los siguientes apartados a detallar las más destacadas. Por supuesto, también se han utilizado las más típicas, como NumPy o Matplotlib. Incluso se ha utilizado Pickle para almacenar los modelos en disco de manera muy práctica.

#### Qiskit

Qiskit [8] es un kit de desarrollo creado por IBM para trabajar con ordenadores cuánticos. En concreto implementa el modelo de circuito cuántico, descrito en el capítulo 2. Es una de las bibliotecas más utilizadas en el mundo de la computación cuántica. Ha sido la pieza fundamental de este trabajo. A partir de la versión 1.0 de Qiskit, hay un componente imprescindible que es necesario instalar aparte: el paquete Qiskit Aer. Está diseñado específicamente para la simulación cuántica de alto rendimiento. Proporciona simuladores cuánticos que permiten a los usuarios ejecutar y probar los circuitos cuánticos diseñados con Qiskit en un entorno simulado antes de ejecutarlos en hardware cuántico real.

Existe también una biblioteca específica para QML, denominada Qiskit Machine Learning. No obstante, este módulo proporciona modelos rígidos que dificultan la adaptación del clasificador basado en *re-uploading* estudiado en este trabajo. Por esta razón se ha optado por trabajar directamente con Qiskit, sin utilizar Qiskit Machine Learning.

#### SciPy

SciPy es una biblioteca de Python de código abierto que amplía las capacidades de NumPy, ofreciendo herramientas y algoritmos adicionales para la computación científica y técnica. En particular, nos interesa el paquete *scipy.optimize*. Proporciona algoritmos de optimización, que utilizamos para minimizar nuestras

funciones de coste.

#### Qutip

QuTiP (Quantum Toolbox in Python) es una biblioteca de Python especializada en la simulación de sistemas cuánticos. En particular, nosotros la utilizaremos para la visualización de estados cuánticos en esferas de Bloch.

## 3.2. Clasificador basado en *Data re-uploading*

En esta sección se explicará el modelo en el que se basa este trabajo. Se trata de una simplificación del modelo de recarga propuesto en el artículo [11].

Al final del capítulo 2 explicamos las QNNs, describiendo el esquema general de su arquitectura (ver figura 2.9). Como ya explicamos, este tipo de modelos se componen de un mapa de características, una forma variacional y una medición. Definiremos ahora un nuevo esquema basado en la repetición continuada de las dos primeras fases. Se muestra en la figura 3.1. Nótese que el mapa de características usa el mismo dato de entrada $\vec{x}$ en cada repetición (de ahí el término "recarga"), mientras que la forma variacional va tomando parámetros optimizables independientes.

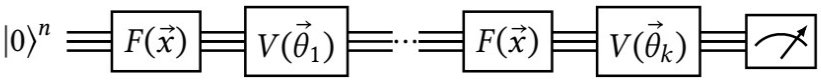



Figura 3.1: Esquema general del modelo de *re-uploading*.

En concreto, diseñaremos un modelo basado en este esquema, que tenga por objetivo funcionar como clasificador. Además, utilizaremos únicamente un cúbit para construirlo. En [11] generalizan este modelo con varios cúbits, pero el interés teórico radica especialmente en emplear un solo cúbit, ya que así es posible justificar su universalidad en términos del Teorema de Aproximación Universal de las redes neuronales clásicas [3]. En el apartado 3.2.5 entraremos en detalle en este tema.

### 3.2.1. Motivación de la recarga de información clásica

Con vistas a construir un clasificador universal de un solo cúbit, un modelo cuántico basado en una fase de carga seguida de una de procesamiento tiene dos limitaciones importantes. En primer lugar, como un cúbit solo tiene dos grados de libertad, estaríamos limitados a trabajar con datos bidimensionales. Por otra parte, tras cargar los datos en el circuito, la transformación que el circuito aplicará al estado cuántico siempre será, en esencia, lo mismo: una rotación en la esfera de Bloch. Con una sola rotación no es posible representar patrones de separación no triviales.

Es necesario cambiar de estrategia para lograr nuestro objetivo, para lo cual nos inspiraremos en las redes neuronales clásicas. En este tipo de modelos, los datos son procesados por una determinada sucesión de capas de neuronas. Sin embargo, ahora nos centraremos simplemente en lo que sucede en la primera capa de la red. Es en este punto donde entran los datos a la red. La clave ahora está en observar que cada dato es procesado múltiples veces; una por cada neurona de esta capa. De este modo podemos decir que los datos se cargan varias veces o se "recargan" en la red.

Basándose en esta idea, Pérez-Salinas et al. [11] proponen construir un circuito cuántico en el que los datos se introduzcan varias veces. Como vamos a trabajar con un único cúbit, sólo nos queda una alternativa: recargar los datos a lo largo del procesamiento del cúbit. En la figura 3.2, extraída de [11], se muestra esquemáticamente esta idea, a modo de comparación de los dos modelos. A la izquierda se representa una red neuronal de una sola capa oculta. Como se puede observar, cada uno de los datos de entrada se introduce en todas las neuronas, representadas por cuadrados. Tras esto, es necesario añadir una neurona final que combine los resultados para producir la salida. En el lado derecho de la figura se muestra el esquema de nuestro modelo cuántico de un cúbit. En este caso, los datos también se cargan varias veces en el modelo, pero ahora lo hacen a lo largo de las distintas capas del circuito, que se corresponden con rotaciones unitarias. Las llamaremos unidades de procesamiento. En cada paso, el clasificador recibe información tanto de la unidad anterior como de la entrada introducida de forma clásica. La salida final del circuito será un estado cuántico codificando múltiples repeticiones de la carga de los datos de entrada condicionadas por parámetros de procesamiento.

En la sección 3.2.5 se analizará con más detalle la relación entre este modelo y las redes neuronales clásicas de una sola capa.

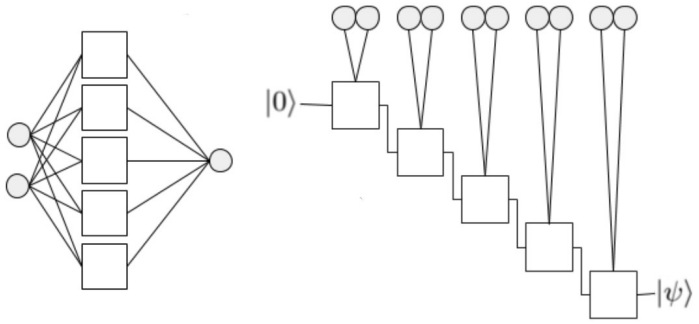



Figura 3.2: Comparación de esquemas de recarga.

### 3.2.2. Procesamiento combinado con la recarga

En esta sección procedemos a desarrollar de forma más explícita el modelo presentado en la figura 3.2. Comenzaremos explicando cómo se introducen los datos clásicos en las unidades de procesamiento. Como no puede ser de otra manera, se hará a través de puertas parametrizadas. De este modo, el clasificador de un solo cúbit pertenecerá a la categoría de los circuitos cuánticos parametrizados.

Como ya hemos comentado, en cada capa del modelo se aplicará una rotación unitaria, que sabemos que pueden ser parametrizadas mediante puertas universales:

$$U(\phi^1, \phi^2, \phi^3) = \begin{pmatrix} \cos\frac{\phi_1}{2} & -e^{i\phi_3}\sin\frac{\phi_1}{2} \\ e^{i\phi_2}\sin\frac{\phi_1}{2} & e^{i(\phi_2+\phi_3)}\cos\frac{\phi_1}{2} \end{pmatrix} \in \mathrm{SU}(2). \tag{3.1}$$

Con el objetivo de resumir esta notación, escribiremos $U(\vec{\phi})$, siendo $\vec{\phi} = (\phi^1, \phi^2, \phi^3)$. Recordemos que, en cada capa, necesitamos introducir tanto el dato de entrada $\vec{x} = (x^1, x^3, x^3)$ como una serie de parámetros ajustables que permitan el aprendizaje del modelo. Teniendo todo esto en cuenta, la estructura del clasificador cuántico de un cúbit se podría expresar de la siguiente manera:

$$\mathcal{U}(\Phi, \vec{x}) \equiv U(\vec{\phi}_N)U(\vec{x}) \cdots U(\vec{\phi}_1)U(\vec{x}), \tag{3.2}$$

donde $\Phi = \{\vec{\phi}_1, \ldots, \vec{\phi}_N\}$ sería el conjunto de vectores de parámetros de las distintas capas, que introducimos con el objetivo de simplificar de nuevo la notación.

Antes de aplicar el circuito, el cúbit se encontrará en el estado $|0\rangle$. Tras aplicarle todas las puertas, pasará al estado

$$|\psi\rangle = \mathcal{U}(\Phi, \vec{x})\,|0\rangle\,. \tag{3.3}$$

Al igual que en otros modelos de QML vistos en el capítulo anterior, la idea para el entrenamiento consistirá en construir una función de coste a partir de este estado

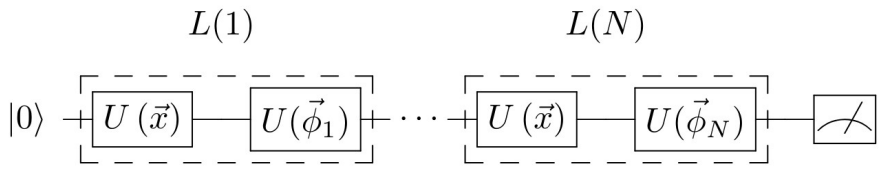



Figura 3.3: Esquema original.

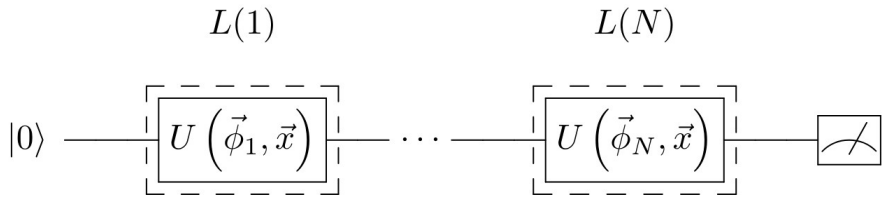



Figura 3.4: Esquema comprimido.

$|\psi\rangle$, y utilizarla para ajustar los parámetros, a medida que se aplica el circuito iterando por todos valores de $\vec{x}$ que haya en el conjunto de entrenamiento. Cabe destacar que, a medida que se añadan más capas al circuito, mejor capacidad de representación tendrá y más efectiva resultará la clasificación que realice. Esto es análogo a lo que sucede con el aumento del número de neuronas de la red neuronal de una sola capa. Aunque ya se profundizará en esta cuestión más adelante, en la sección 3.2.5.

Para poder referirnos a cada capa del modelo de manera independiente, introduciremos el concepto de *capa de procesamiento*, definida de la siguiente manera:

$$L(i) \equiv U(\vec{\phi}_i)U(\vec{x}). \tag{3.4}$$

Así, el clasificador se podrá escribir también como sigue:

$$\mathcal{U}(\Phi, \vec{x}) = L(N) \cdots L(1). \tag{3.5}$$

En la figura 3.3 se muestra un diagrama de la arquitectura que acabamos de diseñar. La profundidad de este circuito es $2N$. Sin embargo, hay una forma sencilla de compactarlo, reduciendo la profundidad a $N$. Tan solo hay que incorporar los parámetros ajustables y el dato de entrada en el mismo paso, utilizando una sola puerta universal en lugar de usar dos. De esta forma, cada capa se escribirá como $L(i) = U(\vec{\phi}, \vec{x})$. El diagrama resultante se muestra en la figura 3.4. Tanto esta como la figura 3.3 también se han obtenido de [11].

La puerta universal solo admite tres parámetros reales, así que hay que detallar cómo se van a combinar los parámetros ajustables $\vec{\phi}_i$ con el dato de entrada

$\vec{x}$. Podríamos pensar en sumar o multiplicar ambos vectores coordenada a coordenada. Sin embargo, los autores proponen una operación un poco más elaborada: una combinación de las dos cosas. El objetivo de esto es dar más sentido a la analogía con el modelo de red neuronal. Duplicaremos los parámetros ajustables en cada capa, resultando en un vector de pesos $\vec{w}_i$, y otro vector $\vec{\theta}_i$, que realizará el papel del sesgo en las neuronas clásicas. A continuación, se muestra la expresión resultante:

$$L(i) = U(\vec{\theta}_i + \vec{w}_i \circ \vec{x}), \tag{3.6}$$

donde $\vec{w}_i \circ \vec{x} = (w_i^1 x^1, w_i^2 x^2, w_i^3 x^3)$ es el producto de Hadamard de dos vectores. Este cambio en los parámetros nos obliga a retocar la notación con la que denotamos el circuito. Al igual que introducíamos el conjunto $\Psi$, ahora consideraremos los conjuntos de vectores de parámetros $\Theta = \{\vec{\theta}_1, \ldots, \vec{\theta}_N\}$ y $\Omega = \{\vec{w}_1, \ldots, \vec{w}_N\}$. Así, la manera más apropiada para denotar el circuito será:

$$\mathcal{U}(\Phi, \vec{x}) \equiv \mathcal{U}(\Theta, \Omega, \vec{x}). \tag{3.7}$$

Con este nuevo diseño se ha conseguido reducir la profundidad a la mitad. Cabría preguntarse ahora si es posible hacer más combinaciones de puertas para acortar todavía más el circuito. No obstante, se perdería la no linealidad inherente a la aplicación de rotaciones consecutivas, lo cual podría afectar al rendimiento del modelo.

Hasta ahora no hemos entrado en un detalle bastante relevante: la dimensión de los datos de entrada, es decir, la dimensión del vector $\vec{x}$. Tal y cómo ha sido presentado, nuestro modelo parece estar pensado únicamente para el caso en el que dicha dimensión es igual a 3. Esto tiene una solución relativamente sencilla. En caso de que la dimensión sea menor que 3, simplemente establecemos el resto de coordenadas de $\vec{x}$ a 0. Sin embargo, si la dimensión es mayor que 3, sería necesario extender la dimensión de $L(i)$ de la siguiente manera:

$$L(i) = U\left(\vec{\theta}_i^{(k)} + \vec{w}_i^{(k)} \circ \vec{x}^{(k)}\right) \cdots U\left(\vec{\theta}_i^{(1)} + \vec{w}_i^{(1)} \circ \vec{x}^{(1)}\right), \tag{3.8}$$

donde hemos dividido cada punto $\vec{x}$ en $k$ vectores de dimensión 3, que escribimos $\vec{x}^{(j)}$, con $j = 1, \ldots, k$. Nótese que, de esta manera, la complejidad del circuito tan solo aumenta linealmente con la dimensión del espacio de los datos de entrada.

### 3.2.3. Medición

Tras haber descrito la arquitectura a utilizar, procedemos a explicar qué mediciones haremos sobre el cúbit después de aplicarle todas las puertas del circuito.

Dichas mediciones serán utilizadas para computar la función de coste. Recordemos que, en este punto, el vector de estado será:

$$|\psi(\Theta, \Omega, \vec{x})\rangle = \mathcal{U}(\Theta, \Omega, \vec{x})\,|0\rangle\,. \tag{3.9}$$

Se ha denotado explícitamente que este estado depende tanto de los conjuntos de parámetros $\Theta$ y $\Omega$, como del dato de entrada $\vec{x}$. Esto será cómodo más adelante para ver con un simple vistazo que la función de coste efectivamente depende de los parámetros del modelo.

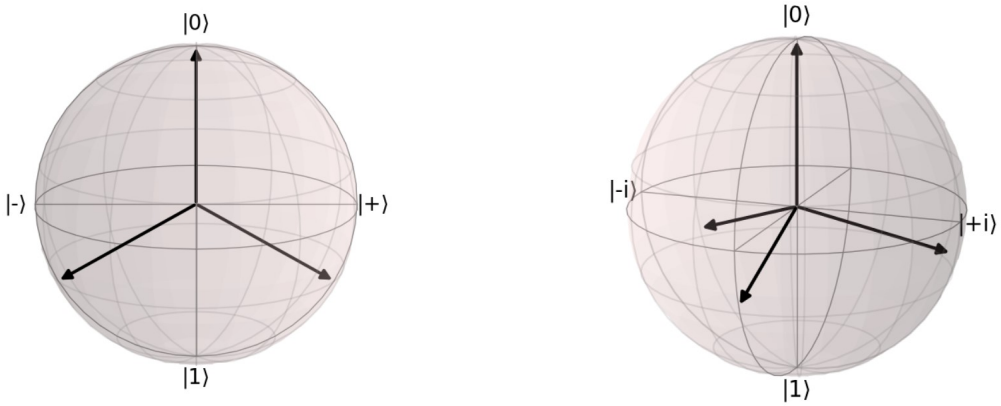
La idea fundamental necesaria para construir un clasificador a partir de mediciones cuánticas consiste en seleccionar ciertos estados específicos que se identificarán con las clases del conjunto de datos. El clasificador será entrenado para distribuir los datos de entrada sobre la esfera, tratando de acercar cada uno al estado que representa a la clase correspondiente.

El principio que nos guiará para escoger los estados en cuestión está basado en la idea de ortogonalidad maximal entre estados. Esto consiste en considerar un conjunto de estados que sean tan "ortogonales" como sea posible. Si solo tenemos dos clases, como sabemos de álgebra, se pueden escoger dos vectores tales que su producto escalar es exactamente cero. Sin embargo, es imposible encontrar más de dos vectores mutuamente ortogonales en un espacio de dimensión 2. Nos tendremos que conformar con encontrar vectores cuyos productos escalares sean tan cercanos a cero como sea posible.

Dos vectores ortogonales resultan en puntos diametralmente opuestos en la esfera de Bloch. De este modo, en el caso de tener un problema de clasificación de dos clases, bastará con identificar dichas clases con los estados $|0\rangle$ y $|1\rangle$. Los ángulos en la esfera de Bloch están relacionados directamente con el resultado del producto escalar entre los vectores correspondientes. De este modo, cuando tengamos más de dos clases seleccionaremos estados cuyos ángulos en la esfera de Bloch sean tan grandes como sea posible para el número de vectores en cuestión. Por ejemplo, en el caso de tres clases, los tres estados se dispondrán formando un triángulo equilátero, puesto que es la manera de maximizar el menor ángulo en un conjunto de tres vectores. De forma análoga, si disponemos de cuatro estados, estos se corresponderán con los vértices de un tetraedro regular. En la figura 3.5 se representan gráficamente dichas configuraciones en la esfera.

### Función de coste basada en fidelidad

Tras haber explicado los fundamentos necesarios para entender cómo nuestro circuito puede llegar a convertirse en un clasificador, describiremos a continuación




(a) Tres estados formando un triángulo equilátero.

(b) Cuatro estados formando un tetraedro regular.

Figura 3.5: Estados máximalmente ortogonales en la esfera de Bloch

una función de coste que servirá para entrenarlo. Está basada en la interpretación geométrica descrita anteriormente. Para cada dato de entrada $\vec{x}$, queremos forzar al correspondiente $|\psi(\Theta, \Omega, \vec{x})\rangle$ a estar lo más cerca posible del estado $|\tilde{\psi_s}\rangle$ que represente la clase de $\vec{x}$. Para medir esta "distancia" entre estados, utilizamos el concepto de *fidelidad entre estados*, dado en este caso por la siguiente expresión:

$$\left|\left\langle \tilde{\psi_s} \middle| \psi(\Theta, \Omega, \vec{x}_\mu) \right\rangle\right|^2 \tag{3.10}$$

El idea entonces será maximizar la fidelidad media de todo el conjunto de entrenamiento o, lo que es lo mismo, minimizar la función:

$$\chi_f^2(\Theta, \Omega) = \sum_{\mu=1}^{M} \left(1 - \left|\left\langle \tilde{\psi_s} \middle| \psi(\Theta, \Omega, \vec{x}_\mu) \right\rangle\right|^2\right), \tag{3.11}$$

donde $M$ es el tamaño del conjunto de entrenamiento.

### Fidelidad ponderada

Ahora construiremos otra función de coste, resultado de refinar las ideas anteriores. En primer lugar, denotaremos por $|\tilde{\psi_c}\rangle$ al conjunto de estados maximalmente ortogonales seleccionados para representar a las clases de nuestro conjunto de datos. Definimos la fidelidad de cada dato $\vec{x}$ con la clase $c$ de la siguiente manera:

$$F_c(\Theta, \Omega, \vec{x}) = \left|\left\langle \tilde{\psi_c} \middle| \psi(\Theta, \Omega, \vec{x}) \right\rangle\right|^2 \tag{3.12}$$

Ahora compararemos esta cantidad con la fidelidad esperada de una clasificación exitosa, que denotaremos $Y_c(\vec{x})$. Este valor representaría el valor de la fidelidad de $\vec{x}$ con la clase $c$ en el caso que el clasificador hubiese logrado llevar el estado $|\psi(\Theta, \Omega, \vec{x})\rangle$ exactamente hasta el vector $|\tilde{\psi_c}\rangle$ asociado a la clase correcta de $\vec{x}$. Es decir, se tiene que:

$$Y_c(\vec{x}) = \left|\left\langle \tilde{\psi_c} \middle| \tilde{\psi_s} \right\rangle\right|^2. \tag{3.13}$$

A modo de ejemplo, explicitaremos estos valores para el caso de un conjunto de datos con tres clases. Los estados en forma de triángulo de la figura 3.5a se corresponden con los siguientes vectores:

$$|\tilde{\psi_1}\rangle = |0\rangle\,, \quad |\tilde{\psi_2}\rangle = \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle\,, \quad |\tilde{\psi_3}\rangle = \frac{1}{2}|0\rangle - \frac{\sqrt{3}}{2}|1\rangle \tag{3.14}$$

Haciendo los cálculos, se obtiene que $Y_c(\vec{x}) = 1$ si $c$ es la clase de $\vec{x}$ y $Y_c(\vec{x}) = \frac{1}{4}$ en caso contrario.

Con estas definiciones, ya podemos construir la nueva función de coste. La idea será minimizar la media de las diferencias entre $F_c(\Theta, \Omega, \vec{x})$ y $Y_c(\vec{x})$ para cada valor de $\vec{x}$ y de $c$. Añadiremos además unos pesos $\vec{\alpha} = (\alpha_1, \ldots, \alpha_{\mathcal{C}})$, siendo $\mathcal{C}$ el número de clases, con el objetivo de flexibilizar ligeramente la función ponderando las distintas fidelidades. Estos pesos le dan el nombre a la función de coste, que se denominará *fidelidad ponderada*. A continuación, presentamos la fórmula completa.

$$\chi^2_{wf}(\vec{\alpha}, \Theta, \Omega) = \frac{1}{2}\sum_{\mu=1}^{M}\left(\sum_{c=1}^{\mathcal{C}}\left(\alpha_c F_c(\Theta, \Omega, \vec{x}_\mu) - Y_c(\vec{x_\mu})\right)^2\right) \tag{3.15}$$

Una diferencia importante con la función de coste anterior es que esta calcula más fidelidades. En concreto, $\chi^2_f$ tan solo calcula una fidelidad por cada dato $\vec{x}$ del conjunto de entrenamiento, mientras que $\chi^2_{wf}$ calcula $\mathcal{C}$ fidelidades por cada $\vec{x}$. Esto podría tener impacto si hay un gran número de clases. No obstante, para valor de $\mathcal{C}$ pequeños, como los que trabajamos aquí, esto no debería ser muy relevante.

Cabe destacar que también existe una diferencia cualitativa entre las dos funciones, relativa al trasfondo de lo que se busca con cada una de ellas. Minimizar la función $\chi^2_f$ simplemente tiene como objetivo acercar cada vector $|\psi(\Theta, \Omega, \vec{x})\rangle$ al estado representante de la clase correcta del dato $\vec{x}$. Por su parte, $\chi^2_{wf}$ no solo hace esto, sino que además busca una configuración específica de las fidelidades, alejando a $|\psi(\Theta, \Omega, \vec{x})\rangle$ de los estados representantes de las clases incorrectas.

### 3.2.4. Método de minimización

Ahora que ya hemos descrito las funciones de coste que dirigirán el entrenamiento de nuestro clasificador, debemos concretar qué método de optimización se utilizará para minimizarlas. Con el objetivo de replicar las condiciones de los experimentos realizados en el artículo [11], emplearemos el algoritmo que mejores resultados les ha proporcionado a estos autores: L-BFGS-B.

A pesar de estar basadas en un circuito cuántico, $\chi_f$ y $\chi_{wf}$ son al fin y al cabo funciones reales de varias variables reales. De este modo, tiene sentido aplicarles métodos de optimización clásicos diseñados para este tipo de funciones. El más habitual es el Descenso del Gradiente Estocástico (SGD, por sus siglas en inglés). En [11] emplean también este algoritmo, pero con peores resultados. Explican esto argumentando que SGD es un algortimo bastante sensible a mínimos locales, especialmente con conjuntos de entrenamiento pequeños. En nuestro caso, las funciones de coste se obtienen de la aplicación reiterada del circuito cuántico y, por tanto, incluyen un gran producto de funciones trigonométricas independientes. Esto provoca la aparición de numerosos mínimos locales

El algoritmo *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) es otro método de optimización no lineal sin restricciones; menos sensible a mínimos locales. Se basa en calcular aproximaciones de la matriz Hessiana para guiar el descenso de la función de coste. En [4] se explica en detalle el procedimiento en cuestión. Una variante de este método es el algoritmo L-BFGS, presentado en [9]. Se trata de una versión limitada en memoria que, en lugar de almacenar y operar sobre la matriz Hessiana completa, lo que hace es almacenar una cantidad limitada de vectores de actualización que representan la aproximación de manera implícita.

Finalmente, L-BFGS-B consiste una extensión de L-BFGS que permite trabajar con restricciones en las variables de optimización. En concreto, permite incluir cotas superiores e inferiores. En [2] se detalla esta variante. Sin embargo, en nuestro caso, las funciones de coste no tienen ninguna restricción en sus variables, de manera que, aunque en [11] se exponga que se está utilizando L-BFGS-B, en esencia se trata simplemente de L-BFGS.

### 3.2.5. Universalidad del clasificador de 1 cúbit

Para terminar el capítulo, explicaremos con un poco más de detalle la relación que existe entre el modelo presentado y las redes neuronales de una sola capa; en el sentido de su comportamiento como aproximadores universales.

En primer lugar, recordamos el Teorema de Aproximación Universal [11] (UAT, por sus siglas en inglés) de las redes neuronales. Para explorar su demostración, presentada en [7], son necesarios conocimientos avanzados de análisis funcional. Nosotros nos vamos a conformar simplemente con enunciarlo, tal y como se hace en [11], con el objetivo de concretar a qué nos referimos al decir que las redes neuronales funcionan como aproximadores de funciones.

**Teorema 1 (UAT)** *Sea $I_m = [0,1]^m$ el cubo m-dimensional y $\mathcal{C}(I_m)$ el espacio de funciones continuas en $I_m$. Sea $\varphi : \mathbb{R} \to \mathbb{R}$ una función continua, acotada, no constante y $f \in \mathcal{C}(I_m)$. Entonces, para cada $\epsilon > 0$, existe un entero $N$ y una función $h : I_m \to \mathbb{R}$ definida por*

$$h(\vec{x}) = \sum_{i=1}^{N} \alpha_i \varphi(\vec{w}_i \cdot \vec{x} + b_i),$$

*con $\alpha_i, b_i \in \mathbb{R}$ y $\vec{w}_i \in \mathbb{R}^m$, tal que $h$ es una realización aproximada de $f$ con precisión $\epsilon$, es decir,*

$$|h(\vec{x}) - f(\vec{x})| < \epsilon$$

*para todo $\vec{x} \in I_m$.*

En términos de redes neuronales, $\varphi$ es la función de activación de las neuronas (por ejemplo, la función sigmoide), $\vec{w}_i$ son los pesos de cada neurona, $b_i$ los sesgos y $\alpha_i$ los pesos que ponderan la salida de cada neurona. De este modo, el teorema establece que es posible "reconstruir" cualquier función continua con una red neuronal de una sola capa, siempre y cuando se incluya un número suficiente de neuronas.

Tras presentar esto, en el artículo [11] se realiza un desarrollo bastante complejo y engorroso para justificar que el modelo de recarga de un solo cúbit también es un aproximador universal. No obstante la explicación concluye de manera difusa, despistándonos muy fácilmente. En un artículo posterior ([12]) en el que participan varios de los autores de [11], se argumenta de manera más precisa la misma idea. En este artículo reformulan un poco los parámetros considerados y la construcción de las puertas. En concreto, para cada capa del circuito consideran

$$U^{UAT}(\vec{x}; \vec{\theta}) \equiv U^{UAT}(\vec{x}; \vec{\omega}, \alpha, \varphi) = R_y(2\varphi) R_z(2\vec{\omega} \cdot \vec{x} + 2\alpha),$$

donde $\omega \in \mathbb{R}^3$, y $\alpha, \varphi \in \mathbb{R}$. Con esta formulación resulta más intuitiva la comparación con la red neuronal de una sola capa. Los vectores $\vec{\omega}$ y los escalares $\alpha$ se corresponden respectivamente con los pesos y los sesgos de las neuronas; mientras que $\varphi$ juega el rol del coeficiente de la neurona. La no linealidad que proporcionarían las funciones de activación, en este caso viene dada por el hecho de parametrizar matrices de rotación. Habiendo aclarado todo esto, finalmente enunciamos el nuevo teorema [12].

**Teorema 2 (Quantum UAT)** *Sean $f$ y $\phi$ un par de funciones $f : I_m \to [0,1]$ y $\varphi : I_m \to [0, 2\pi)$, tales que $z(\vec{x}) = f(\vec{x})e^{i\phi(\vec{x})}$ es una función compleja continua en $I_m = [0,1]^m$. Entonces existe un entero $N$ y un conjunto de parámetros $\{\vec{\theta}_1, \ldots, \vec{\theta}_N\}$ tales que*

$$\left| f(\vec{x})e^{i\phi(\vec{x})} - \langle 1| \prod_{i=1}^{N} U^{UAT}(\vec{x}; \vec{\theta}_i) |0\rangle \right| < \epsilon$$

Nuevamente, la demostración requiere de herramientas de análisis funcional para su comprensión. Se puede consultar en [12].

Como observación final a tener en cuenta, nótese que ambos teoremas tan solo garantizan la existencia de las funciones aproximadoras. No determinan cuál debe ser el tamaño $N$ ni cuánto han de valer los parámetros en ningún caso.

# Capítulo 4

# Pruebas y discusión de los resultados

Este capítulo está dedicado a describir la experimentación realizada para evaluar el clasificador basado en *re-uploading* descrito en el capítulo anterior.

## 4.1. Implementación del clasificador en Qiskit

Recordemos que el clasificador cuántico basado en recarga había sido propuesto en el artículo [11]. Los autores llevaron a cabo una implementación del modelo en cuestión programándola desde cero en Python, sin utilizar bibliotecas de computación cuántica. Es decir, codificaron su propio simulador, en el que probaron distintas configuraciones del modelo. El código en cuestión es público. Se puede consultar en el repositorio [13] de Adrián Pérez Salinas; uno de los autores.

En nuestro caso, la idea ha sido replicar la implementación del modelo, pero empleando una biblioteca estándar de computación cuántica: Qiskit (junto al simulador Aer).Esto nos ha permitido generar un código que se puede adaptar más fácilmente para su ejecución en un ordenador cuántico real.

Como es habitual, y prácticamente obligatorio, en el ámbito del aprendizaje automático, el conjunto de datos considerado en cada caso se divide en un conjunto de entrenamiento y otro de prueba (de menor tamaño). Así, se garantiza la independencia de la evaluación con el posible sobreajuste del modelo. En todos

los problemas de clasificación de este estudio se ha considerado el mismo volumen de datos para cada conjunto: 200 puntos en el conjunto de entrenamiento y 50 en el de prueba. Sin embargo, para las representaciones gráficas que veremos más adelante, se han considerado conjuntos de prueba mucho más grandes: 4000 puntos. Esto se ha hecho siguiendo los pasos de [13], con el objetivo de obtener resultados más visuales, con una mayor densidad de puntos.

La métrica empleada para la evaluación de los modelos ha sido el *accuracy*, que básicamente representa la proporción de datos del conjunto de prueba que se han clasificado correctamente.

En las siguientes secciones, detallamos los problemas de clasificación con los que se ha puesto a prueba al modelo. Como la implementación es esencialmente diferente, los resultados obtenidos diferirán ligeramente de los presentados en [11]. Para cada problema de clasificación se han considerado modelos de 1 a 7 capas, cada uno entrenado con las dos funciones de coste explicadas en la sección 3.2.3.

## 4.2. Clasificación binaria

Comenzaremos explicando los probemas de clasificación más sencillos que se han considerado: los problemas de clasificación binaria. Según lo explicado en la sección 3.2.3, al tratarse de problemas con únicamente dos clases, el rendimiento del modelo debería ser más prometedor a priori.

### 4.2.1. Frontera circular

El primer problema considerado será simplemente la clasificación de puntos dentro y fuera de un círculo. Crearemos un dataset de puntos aleatorios del plano con ambas coordenadas comprendidas en el intervalo $[-1, 1]$. El etiquetado de dichos puntos se hará de la siguiente manera. Consideramos un círculo de radio $R = \sqrt{2/\pi}$. Los puntos que se encuentren dentro del círculo se etiquetarán como pertenecientes a la clase 1, y los de fuera, a la clase 0. El valor del radio se ha escogido para que el área del círculo coincida con el área de la región exterior, garantizando así el balanceo de las clases.

En la tabla 4.1 se muestran los resultados de la evaluación del modelo para este primer problema de clasificación. Se pueden observar unos valores de *accuracy* bastante elevados; muchos por encima de 0,9. Esto supone una muy buena

| | **ACC** | |
|---|---|---|
| **N** | $\chi^2_f$ | $\chi^2_{wf}$ |
| 1 | 0.42 | 0.42 |
| 2 | 0.96 | 0.94 |
| 3 | 0.92 | 0.94 |
| 4 | 0.90 | 0.96 |
| 5 | 0.88 | 0.96 |
| 6 | 0.98 | 0.94 |
| 7 | 0.92 | 0.92 |

Tabla 4.1: Resultados de clasificación del círculo

noticia sobre nuestro clasificador. Partimos de que, al menos en este caso básico, la clasificación se lleva a cabo de manera exitosa. Los valores obtenidos para la función $\chi^2_{wf}$ son prácticamente los mismos que se presentan en [11]. Los de la función $\chi^2_f$ son incluso ligeramente superiores a los de [11] en varios casos. De aquí no podemos concluir mucho acerca de la comparación entre las funciones de coste, ya que los resultados de ambas son muy similares. Además de los valores de *accuracy* dla tabla, también hemos diseñado dos maneras de visualizar gráficamente la evaluación de los modelos. La primera de ellas consiste en representar en un cuadrado los puntos del conjunto de prueba, coloreando cada uno de ellos en función de la clase asignada por el modelo. Se dibuja también el círculo en cuestión, para poder visualizar cuáles de ellos se han predicho correctamente y cuáles no. En la figura 4.1 se muestra esta representación gráfica para el modelo que mejor resultado ha obtenido; el clasificador de 6 con la función de coste $\chi^2_f$ .

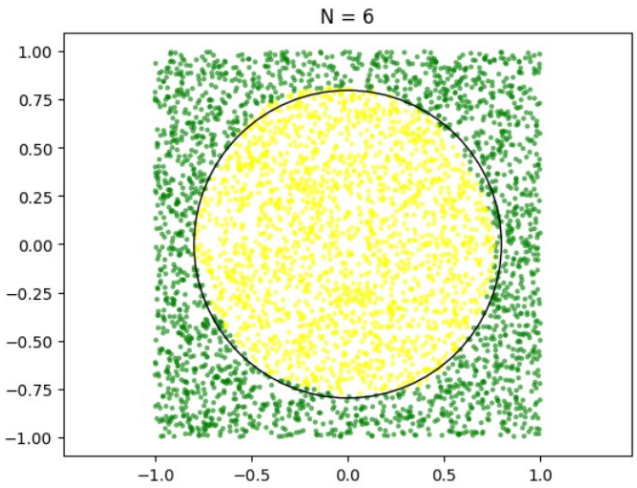



Figura 4.1: Evaluación gráfica del modelo con 6 capas y función de coste $\chi^2_f$ .

En este trabajo también hemos llevado a cabo otra visualización gráfica re-

lativa para ayudar a la evaluación de los modelos. Nos hemos inspirado en representaciones como las que se pueden ver en el artículo [6]. La idea consiste en mostrar explícitamente la esfera de Bloch de nuestro cúbit, dibujando en ella una nube de puntos, de manera que cada uno representa el estado cuántico final tras la ejecución del circuito para un punto del conjunto de evaluación. Se coloreará cada punto en función de su proximidad a los estados representantes de las clases (en este primer caso $|0\rangle$ y $|1\rangle$). En la figura 4.2 se muestra esta representación relativa al mismo modelo que se consideró en la figura 4.1.

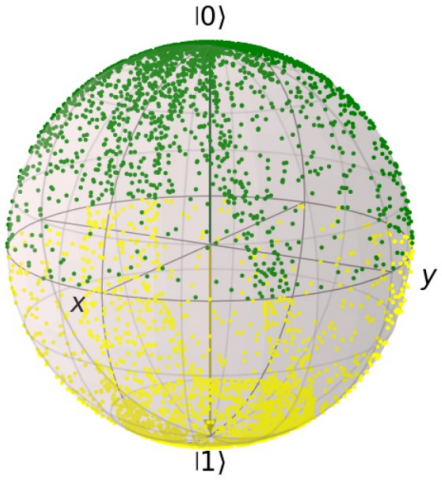



Figura 4.2: Representación de los puntos en la esfera de Bloch tras aplicar el modelo de 6 capas y función de coste $\chi^2_f$ .

Se puede apreciar como la mayor parte de los puntos se concentran en los polos de la esfera. Este es el comportamiento esperado para un modelo que está clasificando correctamente un porcentaje muy elevado (94 %) de los puntos.

### 4.2.2. Regiones no convexas

El problema de clasificación que acabamos de presentar consiste en dos regiones, tales que una de ellas es un conjunto convexo: el círculo. Para estudiar si la convexidad de las regiones facilita de algún modo el buen rendimiento del clasificador, preparamos un nuevo problema de clasificación empleando como frontera una curva senoidal: $f(x) = -2x + \frac{3}{2}\sin \pi x$. De este modo, ambas regiones resultantes serán no convexas.

Como se puede observar en la tabla 4.2, el rendimiento del clasificador para este problema es similar al del caso circular. De hecho, los resultados correspon-

dientes a la función $\chi^2_{wf}$ son ligeramente superiores a los previos. Además, en este caso sí se ve una superioridad de la $\chi^2_{wf}$ sobre $\chi^2_f$ . Los resultados obtenidos son muy similares a los de [11].

| | **ACC** | |
|---|---|---|
| **N** | $\chi^2_f$ | $\chi^2_{wf}$ |
| 1 | 0.42 | 0.44 |
| 2 | 0.82 | 0.8 |
| 3 | 0.82 | 0.94 |
| 4 | 0.94 | 0.96 |
| 5 | 0.94 | 0.96 |
| 6 | 0.92 | 0.98 |
| 7 | 0.94 | 0.98 |

Tabla 4.2: Resultados de clasificación del problema no convexo

En la figura 4.3 se muestra nuevamente la evaluación gráfica del modelo con mejor resultado; el de 7 capas con función de coste $\chi^2_{wf}$ . Se ha dibujado también la gráfica senoidal en cuestión para apoyar la interpretación del resultado

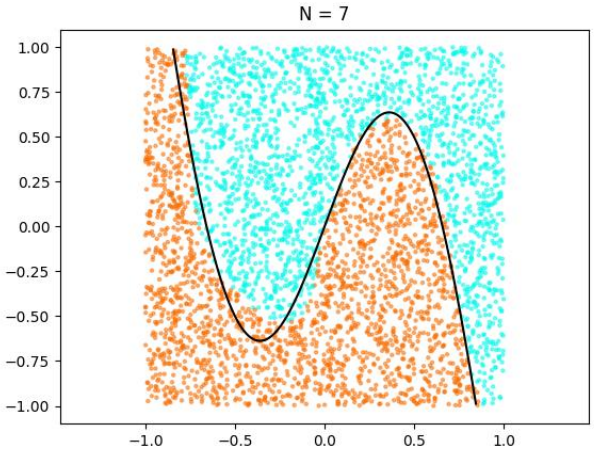
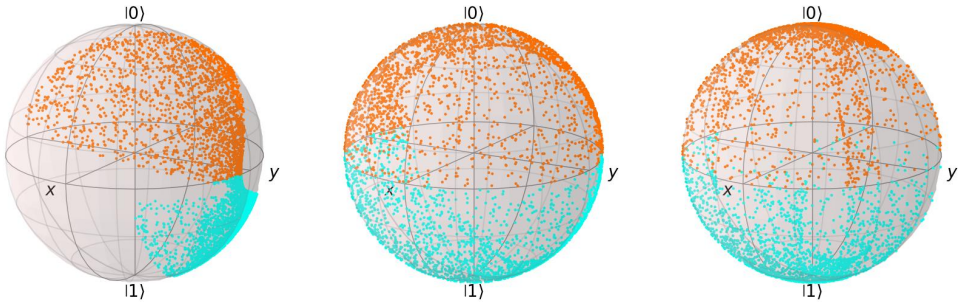



Figura 4.3: Evaluación gráfica del modelo con 7 capas y función de coste $\chi^2_{wf}$ .

Para completar un poco más la evaluación gráfica del modelo, incluimos también en este caso la representación de los puntos en la esfera de Bloch tras aplicar los modelos con distinto numero de capas. Se trata de la figura 4.4. En ella se puede ver la evolución de la representación interna del modelo a medida que la arquitectura se vuelve más compleja. Como se aprecia en la figura, el aumento de la precisión del clasificador se corresponde con una distribución de puntos más centrada en los polos de la esfera.




(a) 2 capas (b) 4 capas (c) 7 capas

Figura 4.4: Evolución de la representación en la esfera de Bloch

Este problema de clasificación se diseñó con la intención de poner a prueba el modelo en un escenario no convexo. Sin embargo, al tratar de complicar la geometría de una de las clases, se simplificó en cierto sentido la de la otra. Es decir, la clase exterior en el problema del círculo tenía la propiedad topológica de tener un "agujero" en su interior; pero en el problema no convexo se ha perdido esta propiedad. En el siguiente apartado propondremos un nuevo problema de clasificación que sí la presente en una de sus clases y, además, complique la estructura de la otra clase.

### 4.2.3. Corona circular

Como acabamos de comentar, exponemos ahora un nuevo problema de clasificación binaria más complejo que los anteriores. Una de las clases vendrá dada por la pertenencia a una corona circular. El resto de puntos, tanto los de fuera del círculo externo como los de dentro del círculo interno, conformarán la segunda clase. De este modo, hemos creado una clase (la corona) con un agujero, mientras la otra clase, además de presentar también un agujero, ha dejado de ser un conjunto conexo.

En la figura 4.3 se muestran los resultados de la evaluación para este problema de clasificación. Esta vez, la complejidad añadida sí se ha visto reflejada en el rendimiento del modelo. Si bien el *accuracy* sigue alcanzando valores valores muy razonables, ya no son tan altos como en los casos anteriores. la tabla también nos muestra otro fenómeno claro: ahora la función de coste $\chi^2_{wf}$ , salvo en un caso esporádico extraño ($N = 4$), sí domina de manera clara a $\chi^2_f$ . Cabe destacar también que en [11] se obtuvieron datos ligeramente por encima de los nuestros en este caso.

| | ACC | |
|---|---|---|
| **N** | $\chi^2_f$ | $\chi^2_{wf}$ |
| 1 | 0.46 | 0.52 |
| 2 | 0.52 | 0.72 |
| 3 | 0.56 | 0.7 |
| 4 | 0.9 | 0.72 |
| 5 | 0.7 | 0.72 |
| 6 | 0.76 | 0.86 |
| 7 | 0.78 | 0.86 |

Tabla 4.3: Resultados de clasificación de la corona circular

En este caso, el aumento de la precisión del modelo en función del número de capas ha sido mucho más progresivo que en los casos anteriores. De este modo, resulta muy vistosa la representación gráfica de la clasificación. En la figura 4.5 se ve claramente como, poco a poco, el modelo cada vez logra ajustar la nube de puntos azul a la forma de la corona circular.

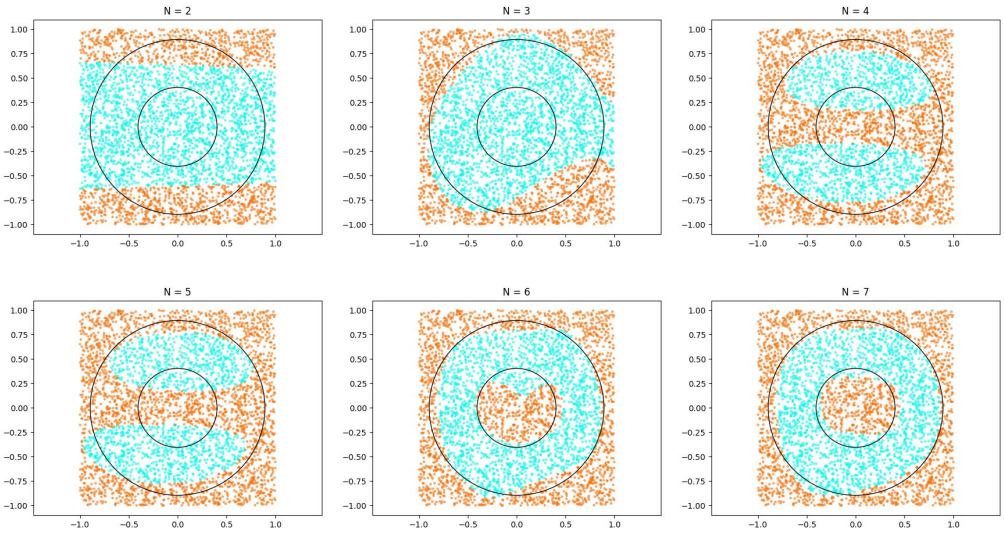



Figura 4.5: Evolución de la clasificación de la corona circular

Con esto damos por terminada la evaluación del modelo basado en *re-uploading* para problemas de clasificación binaria. En la siguiente sección, comenzaremos la experimentación con problemas multiclase.

## 4.3. Clasificación multiclase

A continuación, consideraremos dos problemas de clasificación multiclase: uno con tres clases y otro con cuatro. Como hemos visto en la sección 3.2.3, la clasificación de más de dos clases empleando un cúbit resulta considerablemente más compleja. En la explicación de la ortogonalidad maximal, se dejaba entrever que la forma de clasificar en este caso resulta, en principio, menos potente.

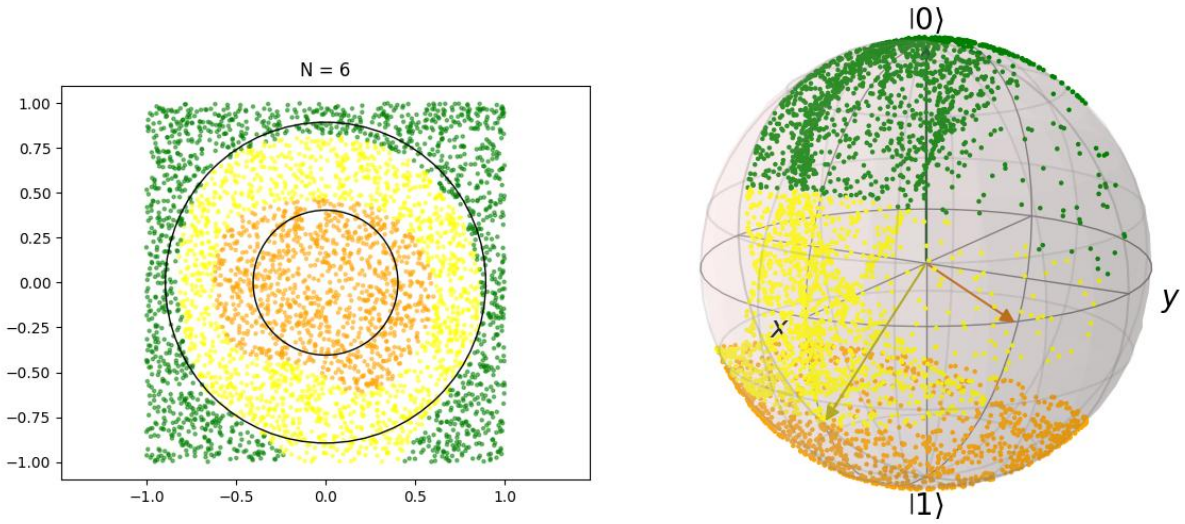
### 4.3.1. Problema de tres clases

Hay una manera muy sencilla de adaptar uno de los problemas anteriores a un problema de tres clases. Consiste simplemente en recuperar la corona circular, y declarar el círculo interno como una clase independiente. En la tabla 4.4 se muestran los resultados para este problema de clasificación. Aunque son relativamente comparables, estos datos son peores que los que presentan en [11]. Un hecho curioso es que, en este caso, la función $\chi^2_f$ supera a $\chi^2_{wf}$ . En [11] sucede lo mismo.

| | **ACC** | |
|---|---|---|
| **N** | $\chi^2_f$ | $\chi^2_{wf}$ |
| 1 | 0.5 | 0.36 |
| 2 | 0.76 | 0.14 |
| 3 | 0.76 | 0.8 |
| 4 | 0.74 | 0.58 |
| 5 | 0.84 | 0.84 |
| 6 | 0.9 | 0.82 |
| 7 | 0.82 | 0.68 |

Tabla 4.4: Resultados de la clasificación con 3 clases

En la figura 4.6 se muestra la evaluación gráfica correspondiente al modelo que ha obtenido el mejor resultado. Lógicamente, ahora se emplean tres colores para representar las clases, pero la idea es exactamente la misma que en los problemas anteriores. A la izquierda se dibuja la corona circular y se colorean los puntos según las clases, pero vemos que no se ajusta tan bien cómo lo hacía en el caso de dos clases. Por otra parte, a la derecha se muestra la esfera de Bloch correspondiente. La novedad principal es que se han incluido (mediante flechas de colores) los tres vectores representantes de las clases.




(a) Representación directa.

(b) Esfera de Bloch con 3 clases.

Figura 4.6: Evaluación gráfica del modelo con 6 capas y función de coste $\chi^2_f$ .

### 4.3.2. Problema de cuatro clases

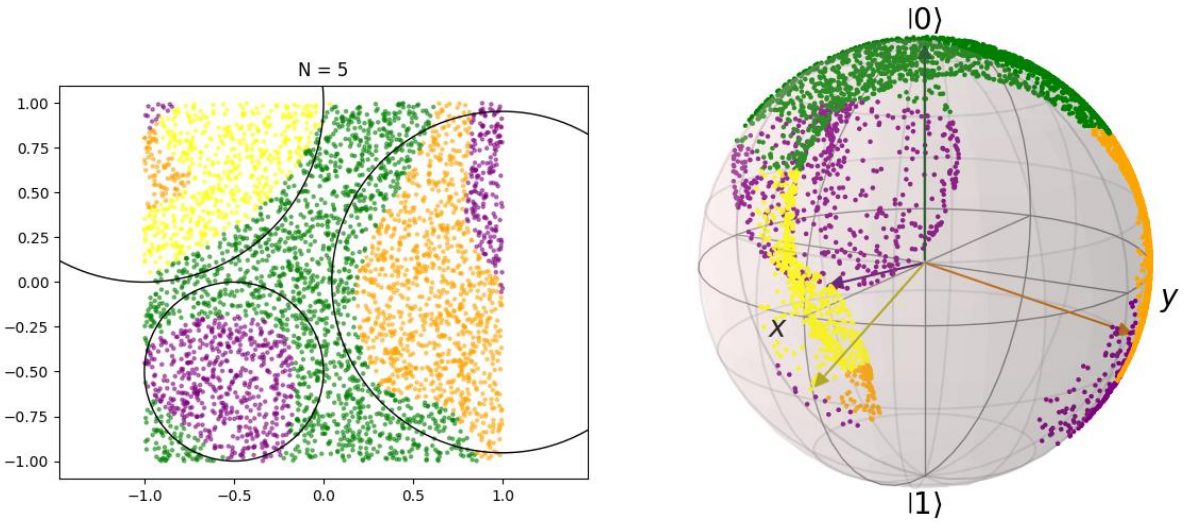
Para finalizar, preparamos también un problema de cuatro clases. Como no se puede adaptar otro problema tan fácilmente como sucedió en el caso anterior, diseñamos de cero un problema de cuatro clases. Consistirá en una disposición de tres círculos disjuntos, de manera que cada uno de ellos conforma una clase y el exterior representa la clase restante. En el tabla 4.5 se muestran los resultados de la clasificación. Estos sí son notablemente inferiores a los de [11]. Hemos vuelto a obtener una ligera superioridad de $\chi^2_{wf}$ .

| | **ACC** | |
|---|---|---|
| **N** | $\chi^2_f$ | $\chi^2_{wf}$ |
| 1 | 0.52 | 0.26 |
| 2 | 0.5 | 0.4 |
| 3 | 0.58 | 0.64 |
| 4 | 0.56 | 0.34 |
| 5 | 0.7 | 0.76 |
| 6 | 0.64 | 0.66 |
| 7 | 0.66 | 0.7 |

Tabla 4.5: Resultados de la clasificación con 4 clases

Por completitud, se muestra en la figura 4.7 la evaluación gráfica del modelo más aventajado para este problema; utilizando ahora cuatro colores. Sorpren-

dentemente, a pesar de los aparentemente malos resultados, en ambos dibujos parece que los puntos se ajustan razonablemente bien a las regiones y vectores correspondientes.




(a) Representación directa.

(b) Esfera de Bloch con 4 clases.

Figura 4.7: Evaluación gráfica del modelo con 5 capas y función de coste $\chi^2_{wf}$ .

# Capítulo 5

# Conclusiones y posibles ampliaciones

En este capítulo final resumimos las principales aportaciones del trabajo y las conclusiones extraídas de la experimentación realizada en el capítulo anterior. Además, se comentarán varias posibilidades de continuación que admite este trabajo.

En primer lugar, recordamos que nuestra aportación principal ha sido la implementación del clasificador de un solo cúbit basado en *re-uploading*. En concreto, la novedad respecto al artículo de referencia [11] ha sido es desarrollo de un código que utiliza la biblioteca Qiskit, estándar en computación cuántica. Esto sitúa al modelo en una implementación más cercana a una posible ejecución el hardware cuántico.

En cuanto a los resultados de la experimentación, en general se ha obtenido un buen rendimiento del modelo, con valores de *accuracy* elevados. En el caso de clasificación binaria, los resultados son muy similares a los presentados en [11], llegando a superarlos en algunos casos. No obstante, la clasificación multiclase no ha sido tan exitosa. Si bien para el problema de tres clases los valores no son muy inferiores a los de [11], para el problema de cuatro clases sí existe una diferencia notoria. Los autores de [11] utilizan una implementación propia, programando un simulador desde cero. De esta forma, su código y el nuestro son esencialmente diferentes. Es muy probable que sea este hecho la causa de las diferencias en los resultados.

Otro análisis interesante de los resultados surge de la comparación entre las dos funciones de coste utilizadas. En [11] concluyen que la función $\chi^2_{wf}$ es superior

a $\chi^2_f$ . En nuestro caso, la situación es la siguiente. En los problemas de clasificación binaria y en el de cuatro clases, la comparación coincide con la conclusión de [11], salvo para el primer problema (en el que ninguna función domina a la otra claramente). Sin embargo, en el problema de tres clases obtenemos sorprendentemente una superioridad de $\chi^2_f$ frente a $\chi^2_{wf}$ . Cabe destacar que en [11], justo para este mismo problema, al menos no se obtiene una superioridad muy clara de $\chi^2_{wf}$ . Tras todo esto, sí podemos concluir que en general la función $\chi^2_{wf}$ es ligeramente más apropiada para entrenar los clasificadores. Esta es la conclusión esperada en base a la explicación dada en la sección 3.2.3 sobre la interpretación de ambas funciones.

Para terminar, describiremos un par de posibles continuaciones de este trabajo. En primer lugar, la lectura del artículo [11] lleva naturalmente a la idea de implementar un nuevo clasificador utilizando varios cúbits en lugar de uno solo. No hemos hecho esto por dos motivos. Por una parte, el objetivo del trabajo era implementar el modelo (el clasificador de un cúbit) en el que nos fijamos debido a su interés teórico. Un modelo con más cúbits, aunque pueda obtener mejores resultados, no presenta una base teórica tan sólida. La segunda razón es que los resultados que obtienen en [11] para varios cúbits, si bien mejoran respecto a considerar un solo cúbit, la mejora es muy ligera, puesto que ya tenían muy buenos resultados en todos los casos. Sin embargo ahora, a la vista de nuestros "malos" resultados en clasificación multiclase, sí consideramos que tendría sentido probar la implementación con varios cúbits con la esperanza de conseguir mejorar el rendimiento.

Por otro lado, la continuación más interesante del trabajo sería la ejecución de nuestro código en hardware cuántico real. Teníamos la intención de utilizar el nuevo ordenador cuántico del CESGA, llamado Qmio. Sin embargo, no lo han puesto a disposición de la comunidad hasta este pasado mes de junio, imposibilitándonos las tramitaciones correspondientes en una fecha tan cercana a la entrega del trabajo.

Dado que las funciones de coste que empleamos utilizan directamente los vectores de estado del cúbit, la implementación requeriría una cierta adaptación para su ejecución en hardware cuántico real. La razón es que, en un cúbit de verdad, no es posible acceder al estado exacto, tan solo se pueden hacer mediciones que colapsan a los estados $|0\rangle$ y $|1\rangle$. De este modo, para obtener las probabilidades (fidelidades) que construyen las funciones de coste, habría que ejecutar el circuito un número de veces elevado, obteniendo así una aproximación estocástica de dichas probabilidades. La ejecución en hardware cuántico permitiría comprobar el efecto de ruido en el rendimiento del clasificador, lo cual conformaría otro estudio enormemente interesante, puesto que esta característica es muy difícil de reproducir mediante simulación.

# Apéndice A

# Manual técnico

La implementación de modelo que hemos utilizado para los experimentos se puede consultar en el siguiente enlace: `https://nubeusc-my.sharepoint.com/:f:/r/personal/manuel_timiraos_rai_usc_es/Documents/TimiraosLopezManuel_TFG_GREI?csf=1&web=1&e=bafKg3`.

En este manual se describe la configuración de bibliotecas y versiones necesarias para poder reproducir la ejecución del código. Se trata de una guía muy sencilla, puesto que no se requiere de un gran número de bibliotecas.

Lo primero es crear un entorno de Python (versión 3.10.14). En él, utilizaremos el comando *pip* para instalar las bibliotecas. Comenzamos instalando la herramienta de *pip* que permite buscar bibliotecas: *pip_search*. Con los siguientes comandos, buscamos Qiskit y la instalamos.

```
pip install pip_search
pip_search qiskit
pip install qiskit
```

Es muy importante asegurarse de que la versión instalada de Qiskit sea la 1.0, ya que justo en esta versión se produce una reestructuración interna del paquete, cambiando todas las dependencias. En caso de instalar una versión anterior, será mejor crear un nuevo entorno, ya que es complicado desinstalar bien las dependencias antiguas.

Tras instalar el paquete principal de Qiskit, hay que instalar de forma independiente el simulador Aer:

```
pip install qiskit-aer
```

Para poder realizar las representaciones gráficas también hay que instalar lo siguiente:

```
pip install matplotlib
pip install pylatexenc
```

Por último, para poder generar las representaciones de las esferes de Bloch, utilizamos la bilioteca QuTiP:

```
pip install qutip
```

Con esto, queda listo el entorno para poder ejecutar las pruebas deseadas.

# Apéndice B

# Manual de usuario

La implementación de modelo que hemos utilizado para los experimentos se puede consultar en el siguiente enlace: `https://nubeusc-my.sharepoint.com/:f:/r/personal/manuel_timiraos_rai_usc_es/Documents/TimiraosLopezManuel_TFG_GREI?csf=1&web=1&e=bafKg3`.

Este codigo se distribuye en ficheros de Python que contienen las diferentes funciones necesarias para la generación de los datasets, el entrenamiento de los modelos, la evaluación y la generación de las visualizaciones gráficas. También se proporcionan varios *notebooks* a modo de ejemplo, para saber cómo utilizar las distintas funcionalidades.

A continuación, explicamos la distribución de dichos *notebooks*. En primer lugar *gen_data_notebook.ipynb*, está dedicado a la generación de los conjuntos de datos relativos a todos los problemas de clasificación vistos en el trabajo. Los conjuntos de datos generados se almacenan en archivos (ya separados en entrenamiento y prueba) para su posterior uso. En la carpeta *datasets* ya se disponen los que se han utilizado para entrenar los modelos del trabajo.

Para entrenar los modelos, debería consultarse *train_notebook.ipynb*. Ahí se muestran varios ejemplos. También se incluye el código con el que se han entrenado todos los modelos evaluados en el trabajo. En la carpeta *modelos* se disponen ya todos los modelos evaluados en el trabajo.

Los ejemplos de evaluación se llevan a cabo en *test_notebook.ipynb*. Finalmente, para completar la evaluación con las representaciones gráficas que se han visto en el trabajo, se proporciona *graphic_test_notebook.ipynb*.

# Bibliografía


[1] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[2] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

[3] Elias F Combarro, Samuel González-Castillo, and Alberto Di Meglio. *A Practical Guide to Quantum Machine Learning and Quantum Optimization: Hands-on Approach to Modern Quantum Algorithms*. Packt Publishing Ltd, 2023.

[4] Yu-Hong Dai. Convergence properties of the bfgs algoritm. *SIAM Journal on Optimization*, 13(3):693–701, 2002.

[5] Artículo de la wikipedia. Esfera de bloch. `https://es.wikipedia.org/`. Accedido por última vez 26 de junio de 2024.

[6] Philip Easom-Mccaldin, Ahmed Bouridane, Ammar Belatreche, and Richard Jiang. On depth, robustness and performance using the data re-uploading single-qubit classifier. *IEEE Access*, 9:65127–65139, 2021.

[7] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[8] IBM. Qiskit. `https://www.ibm.com/quantum/qiskit`. Accedido por última vez 29 de junio de 2024.

[9] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[10] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[11] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, 2020.

[12] Adrián Pérez-Salinas, David López-Núñez, Artur García-Sáez, Pol Forn-Díaz, and José I Latorre. One qubit as a universal approximant. *Physical Review A*, 104(1):012405, 2021.

[13] Adrián Pérez Salinas. Universal qlassifier. `https://github.com/AdrianPerezSalinas/universal_qlassifier`. Accedido por última vez 1 de julio de 2024.

[14] Maria Schuld and Francesco Petruccione. *Machine learning with quantum computers*, volume 676. Springer, 2021.

[15] Robert S Sutor. *Dancing with Qubits: How quantum computing works and how it can change the world*. Packt Publishing Ltd, 2019.