Escuela
Politécnica
Superior

Computación Cuántica y sus aplicaciones en Inteligencia Artificial

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Vicent Baeza Esteve
Tutor/es:
Francisco Antonio Pujol Lopez

Universitat d'Alacant
Universidad de Alicante

Mayo 2024

Computación Cuántica y sus aplicaciones en Inteligencia Artificial

Estudio sobre la computación cuántica y sus aplicaciones en la inteligencia artificial

Autor
Vicent Baeza Esteve

Tutor/es
Francisco Antonio Pujol Lopez
Departamento de Tecnología Informática y Computación

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2024

Resumen

Actualmente, las demandas computacionales de los algoritmos de inteligencia artificial están creciendo a un ritmo veloz. Los modelos más potentes de inteligencia artificial actuales requieren de una cantidad de cómputo impresionante para ser entrenados y ejecutados. Con las tendencias actuales esta demanda de recursos no hará más que crecer en los próximos años, agravando aún más el problema que supone entrenar y ejecutar este tipo de modelos de inteligencia artificial.

Una de las posibles soluciones a este problema es el uso de la computación cuántica, una rama de la computación que utiliza ciertas propiedades de la física cuántica para acelerar y mejorar la eficiencia de ciertos algoritmos. Mediante el uso de ordenadores y procesadores especializados que exploten estas propiedades, se pueden diseñar algoritmos para ciertos problemas que reducen la complejidad computacional y, por lo tanto, las demandas computacionales.

En este trabajo se analizarán los elementos básicos de la computación cuántica y sus aplicaciones principales orientadas a la inteligencia artificial. Para cada una de las aplicaciones analizadas, también se estudiará su posible impacto en la eficiencia y eficacia, y algunas de las implementaciones actuales que la utilicen.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

La computación cuántica es un campo que nació en los años 80, como posible solución al rápido incremento de las demandas de procesamiento incurridas por los ordenadores de la época. Treinta años más tarde, esa demanda no ha hecho más que crecer. Uno de los campos que más ha hecho crecer esta demanda es la inteligencia artificial, que requiere de cantidades inmensas de datos y procesamiento para entrenar modelos de inteligencia artificial que resuelvan adecuadamente problemas muy complejos pero muy útiles, como el reconocimiento de imágenes y vídeo, el procesamiento de lenguaje natural o la detección de patrones o similitudes. Las grandes demandas de datos han hecho que actualmente sea necesario tener grandes conjuntos de ordenadores o superordenadores para entrenar los modelos más sofisticados.

Ante las crecientes demandas computacionales de la inteligencia artificial, la computación cuántica podría suponer un cambio de paradigma total, ya que muchas de las técnicas que veremos a continuación pueden suponer grandes mejoras en la eficiencia y eficacia a la hora de desarrollar, entrenar y ejecutar modelos de inteligencia artificial. El aprovechamiento de ciertas propiedades de la física cuántica hace que sea posible el desarrollo de mejores algoritmos y técnicas de lo que sería posible con ordenadores clásicos. Explicaremos en detalle más adelante muchas de estas técnicas, sus implementaciones actuales y su posible uso en el futuro cercano.

La computación cuántica supone un cambio de paradigma casi total respecto a la computación clásica. La manera de representar información, el procesamiento y depuración de datos, la lógica cuántica y sus puertas lógicas correspondientes y los circuitos cuánticos difieren mucho de sus equivalentes en los ordenadores actuales. Es por esto que su integración, aunque pueda reducir el coste temporal de la inteligencia artificial masivamente, haya sido reducida actualmente. No obstante, con el constante desarrollo tecnológico y algorítmico de la computación cuántica, su uso en la inteligencia artificial cada vez se está volviendo más y más común.

1.1 Objetivos

En este apartado se enumeran los objetivos concretos que se han tenido en cuenta durante el desarrollo de este trabajo.

- Analizar diferentes técnicas de la computación cuántica y el beneficio potencial que puede suponer su integración en la inteligencia artificial
- Estudiar los problemas específicos que supone la computación cuántica y las posibles soluciones a estos problemas
- Analizar casos de uso que verifiquen la validez del estudio realizado

1.2 Metodología

Para la realización de este TFG, se han analizado las aplicaciones y casos de uso que se han considerado más relevantes a la hora de integrar la inteligencia artificial con la computación cuántica. Como la computación cuántica es un campo emergente, no está del todo claro cuáles de sus muchas aplicaciones serán las más útiles a la hora de aplicarlas a la inteligencia artificial en el futuro. Se han priorizado las aplicaciones con mayor posible beneficio, así como las "desarrolladas", es decir, las que tienen una mayor cantidad de estudios anteriores y de implementaciones físicas.

En base al criterio anterior, se han analizado 3 aplicaciones: los algoritmos de multiplicación de matrices cuánticos (capítulo 4), la búsqueda de Grover (capítulo 5) y la computación adiabática cuántica (capítulo 6).

1.3 Estructura del trabajo

A continuación se describe la estructura del trabajo, junto con un breve resumen del contenido de cada capítulo.

- **Capítulo 2: Computación cuántica**: introduce al lector a la computación cuántica. Se explican los fundamentos físicos de la computación cuántica, estados cuánticos, bits cuánticos, puertas lógicas cuánticas y circuitos cuánticos.
- **Capítulo 3: Inteligencia artificial**: introducción a la inteligencia artificial y a muchos de los conceptos básicos de la misma. Se explican las neuronas artificiales, las redes neuronales, los diferentes métodos de aprendizaje y el estado actual de las redes neuronales modernas.
- **Capítulo 4: Algoritmos de multiplicación de matrices**: se estudian ciertos algoritmos cuánticos de multiplicación de matrices, y se comparan con los algoritmos actuales y se realiza un análisis de complejidad de los mismos. Se realiza también una estimación temporal de los algoritmos en los modelos actuales de redes neuronales.
- **Capítulo 5: Búsqueda de Grover**: se estudia la integración de la búsqueda de Grover en la inteligencia artificial. Se realiza un análisis completo de la búsqueda de Grover y sus múltiples aplicaciones en la inteligencia artificial.
- **Capítulo 6: Computación cuántica adiabática**: se analiza la computación adiabática, un tipo de computación cuántica que utiliza otros principios distintos a la computación cuántica basada en puertas y circuitos. También se estudian las arquitecturas y topologías D-Wave, una de las arquitecturas principales actuales en ordenadores adiabáticos cuánticos.
- **Capítulo 7: Conclusiones**: resumen de las conclusiones de todo el trabajo, y pequeño análisis de los problemas principales actuales que previenen la implementación de los ordenadores cuánticos en proyectos comerciales.

2 Computación cuántica

Antes de ver como aplicar la computación cuántica a la inteligencia artificial, es necesario introducir los conceptos necesarios para entender la computación cuántica. Empezaremos por conocer los fundamentos físicos que nos permiten realizar cálculos utilizando la física cuántica, y se irán viendo todas las estructuras algebraicas que se construyen a partir de esos fundamentos físicos. Estas estructuras, como los bits cuánticos, las puertas lógicas cuánticas o los circuitos cuánticos, nos proporcionan las abstracciones necesarias para poder razonar sobre algoritmos cuánticos.

2.1 Fundamentos físicos de la computación cuántica

La computación cuántica utiliza la física cuántica como base para realizar cálculos lógicos. A un tamaño suficientemente pequeño, la materia se comporta como ondas y partículas a la vez [1]. Esto se conoce como la dualidad onda-partícula, y es uno de los principios básicos de la física cuántica. A partir de esta dualidad, surgen muchos comportamientos interesantes que los ordenadores cuánticos pueden aprovechar para mejorar significativamente el tiempo requerido para realizar ciertos cálculos. Especialmente, se utilizan la superposición cuántica [2] y el entrelazamiento cuántico [3].

El principio de superposición cuántica es la base fundamental para muchos de los algoritmos cuánticos que estudiaremos a lo largo de este trabajo. La superposición cuántica consiste en que en los sistemas físicos cuánticos (ver apartado 2.1.1) se dan todas sus posibles configuraciones a la vez, hasta que sean observados. Específicamente, para cada sistema cuántico existen varios estados fundamentales, los posibles estados en los que puede estar el sistema tras ser observado. El estado actual es una combinación lineal de estos estados fundamentales, siendo la probabilidad de cada uno de los sistemas un factor en la combinación. La superposición y sus consecuencias para la computación cuántica se estudiarán con más detalle en el apartado 2.2.

El otro principio que vamos a necesitar, el entrelazamiento cuántico, también es un pilar importante de la computación cuántica. Este principio nos dice que podemos "entrelazar" dos sistemas cuánticos, haciendo que las probabilidades de ambos estén relacionadas. Como veremos en el apartado 2.2.4, esto nos permite generar probabilidades dependientes, de forma que podemos realizar cálculos en ambos sistemas a la vez. Esto, teóricamente, nos proporciona un *speedup* exponencial, ya que con cada sistema que entrelacemos multiplicamos la cantidad de estados posibles.

Utilizando estos dos principios, y algunos otros que se explicarán durante el desarrollo del presente trabajo, podemos realizar cálculos aprovechando las propiedades de la física cuántica para mejorar el tiempo de ejecución y la eficiencia de muchos algoritmos.

2.1.1 Sistemas físicos de los ordenadores cuánticos

Para implementar los ordenadores cuánticos, a lo largo de los años se han utilizado una gran variedad de sistemas físicos. La gran mayoría de estos sistemas físicos son sistemas binarios (sistemas que tienen dos estados fundamentales, $|0\rangle$ y $|1\rangle$). A continuación se puede ver una lista de algunas implementaciones físicas utilizadas en ordenadores cuánticos actuales:

Sistema	Estado $\|0\rangle$	Estado $\|1\rangle$
Polarización de un fotón	Polarización horizontal	Polarización vertical
Número de fotones	0 fotones	1 fotón
Espín de un electrón	Espín positivo	Espín negativo
Espín de un núcleo atómico	Espín positivo	Espín negativo
Punto cuántico [4]	Espín positivo	Espín negativo
Sentido de una corriente	Sentido horario	Sentido anti-horario
Par de puntos cuánticos	Electrón en punto izquierdo	Electrón en punto derecho
Semiconductor de doble capa	Electrón en capa inferior	Electrón en capa superior

Tabla 2.1: Sistemas físicos en la computación cuántica. Fuente: [5]

Además de los sistemas de la tabla anterior, existen muchos otros sistemas físicos que se utilizan actualmente. Muchos de ellos dependen de la arquitectura del procesador cuántico del ordenador, por lo que enumerar todas las implementaciones consistiría en enumerar todas las arquitecturas de ordenadores cuánticos. También existen sistemas cuánticos con más de dos estados [6], que requieren un menor número de sistemas para representar la misma información pero complican más cada uno de los sistemas.

Esta tabla solo representa los sistemas físicos para los ordenadores cuánticos basados en puertas lógicas cuánticas (ver apartado 2.3). Como se estudiará en el capítulo 6, existen otros tipos de ordenadores cuánticos que no siguen ese modelo de computación.

2.1.2 Notación bra-ket

Como consecuencia de que la computación cuántica esté basada en la física cuántica, hay muchas convenciones y formalismos de la física cuántica que se han adoptado en la computación cuántica. Entre ellos, el más prominente es la notación bra-ket [7] (del inglés *braket*), que se utiliza para describir los estados cuánticos y algunas de las operaciones básicas que se pueden realizar sobre los mismos.

Esta notación está formada por dos partes, los *bras* $\langle\phi|$ y los *kets* $|\psi\rangle$. Cada *ket* representa un estado cuántico, y cada *bra* representa un punto de referencia desde el que se observa el sistema cuántico. Al combinar un *bra* $\langle\phi|$ con un *ket* $|\psi\rangle$ de la forma $\langle\phi|\psi\rangle$, obtenemos la amplitud de que el sistema pase del estado $|\psi\rangle$ al estado $|\phi\rangle$ al ser observado. Las amplitudes, junto con su relación con la probabilidad de cada estado, se estudian en más detalle en el apartado 2.2.1.

2.1.2.1 Kets

Los sistemas cuánticos tienen estados fundamentales, a partir los cuales podemos construir todos los posibles estados del sistema. Aunque normalmente estos estados fundamentales son combinaciones de $|0\rangle$ y $|1\rangle$, en ciertos sistemas cuánticos podemos tener estados fundamentales más complejos. Estos sistemas forman una base vectorial, por lo que a veces se les conoce también como los estados base del sistema. Todos los posibles *kets* del sistema cuántico se pueden definir a partir de los estados base. Si $E_0, E_1, \ldots$ son los estados base del sistema, tendríamos la siguiente expresión para un *ket*, siendo α_{E_i} la amplitud del *ket* para la base E_i:

$$|\psi\rangle = \alpha_{E_1} E_1 + \alpha_{E_2} E_2 + \cdots = \sum_i \alpha_{E_i} E_i \tag{2.1}$$

Esto también se podría expresar utilizando el producto interior para obtener las amplitudes del *ket*:

$$|\psi\rangle = \langle E_1|\psi\rangle\, E_1 + \langle E_1|\psi\rangle\, E_2 + \cdots = \sum_i \langle E_i|\psi\rangle\, E_i \tag{2.2}$$

Si a partir de los estados base $E_1, E_2, \ldots$ definimos un espacio vectorial (que en nuestro caso sería un espacio de Hilbert complejo [8]), podemos expresar los *kets* como vectores dentro de ese espacio:

$$|\psi\rangle = \alpha_{E_1} E_1 + \alpha_{E_2} E_2 + \cdots = \begin{pmatrix} \alpha_{E_1} \\ \alpha_{E_2} \\ \cdots \end{pmatrix} = \begin{pmatrix} \langle E_1|\psi\rangle \\ \langle E_2|\psi\rangle \\ \cdots \end{pmatrix} \tag{2.3}$$

2.1.2.2 Bras

Un *bra* es el vector que al multiplicarlo con cualquier *ket* es igual a la amplitud de ese *bra* y *ket*. En otras palabras, multiplicar un *bra* y un *ket* es igual al producto interior:

$$\langle\psi|\,|\phi\rangle = \langle\psi|\phi\rangle \quad \forall\,|\phi\rangle \tag{2.4}$$

Podemos construir un *bra* a partir de su *ket*. El *bra* correspondiente a un *ket* se define como el vector transpuesto conjugado del *ket*. Sea $\hat{v}^\dagger$ el transpuesto conjugado del vector $\hat{v}$ y x^* el conjugado complejo del número x, tenemos la siguiente definición:

$$\langle\psi| = |\psi\rangle^\dagger = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \ldots \end{pmatrix}^\dagger = \begin{pmatrix} \alpha_1^* & \alpha_2^* & \ldots \end{pmatrix} \tag{2.5}$$

Como los *bras* y *kets* son vectores, podemos aplicar operaciones vectoriales sobre ellos. Esto incluye la suma y resta, el producto escalar y la multiplicación por valores escalares. Como veremos más adelante, no todos los posibles *kets* y *bras* son estados cuánticos válidos, ya que existen ciertas restricciones que limitan como podemos formar los estados.

De la notación *bra-ket*, utilizaremos el producto interior para extraer amplitudes de estados cuánticos. Si tenemos un estado $|\psi\rangle = \sum_i \alpha_{E_i} |E_i\rangle$, podemos extraer cualquier α_{E_i} utilizando el *bra* $\langle E_i|$:

$$\alpha_{E_i} = \langle E_i|\psi\rangle \tag{2.6}$$

También podemos extraer combinaciones lineales de amplitudes de cualquier combinación lineal de estados, aplicando las propiedades de los vectores:

$$(\langle E_1| + \langle E_2|)(|\psi_1\rangle + |\psi_2\rangle) = \langle E_1|\psi_1\rangle + \langle E_1|\psi_2\rangle + \langle E_2|\psi_1\rangle + \langle E_2|\psi_2\rangle \tag{2.7}$$

Utilizaremos la flexibilidad proporcionada por esta notación para simplificar algunos de los desarrollos matemáticos en los siguientes apartados.

2.1.2.3 Sistemas compuestos

En muchos de los siguientes apartados tendremos que trabajar con sistemas compuestos de otros sistemas. Para componer dos sistemas con los estados base $E_1, E_2, \ldots$ y $F_1, F_2, \ldots$ utilizaremos el producto tensorial. Cada estado base del sistema compuesto se puede obtener como un producto tensorial de dos estados bases $|E_i\rangle$ y $|F_j\rangle$:

$$|E_i, F_j\rangle = |E_i\rangle \otimes |F_j\rangle \tag{2.8}$$

También se puede omitir la coma, dejándonos con $|E_iF_j\rangle$. Omitiremos la coma solo cuando no cree ambigüedad. Por ejemplo, si tenemos dos sistemas binarios, cada uno con los estados base $|0\rangle$ y $|1\rangle$, tendríamos los siguientes estados base en el sistema compuesto por los dos sistemas:

$$\begin{aligned} |00\rangle &= |0,0\rangle = |0\rangle \otimes |0\rangle \\ |01\rangle &= |0,1\rangle = |0\rangle \otimes |1\rangle \\ |10\rangle &= |1,0\rangle = |1\rangle \otimes |0\rangle \\ |11\rangle &= |1,1\rangle = |1\rangle \otimes |1\rangle \end{aligned} \tag{2.9}$$

2.2 Bits cuánticos

El bit es la unidad fundamental de información en los sistemas de computación clásicos. Un bit es la cantidad de información mínima, un 0 o un 1, verdadero o falso. Normalmente, otros tipos de datos más complejos se construyen a partir de bits. La computación cuántica tiene un concepto análogo al bit, el cúbit (bit cuántico, del inglés *quantum bit*). Los bits cuánticos, como los bits clásicos, pueden estar en uno de los dos estados base, $|0\rangle$ o $|1\rangle$. No obstante, gracias al principio de superposición, los cúbits también pueden estar en una combinación de los dos estados al mismo tiempo. Por ejemplo, el estado $|+\rangle$ se encuentra en el punto medio entre $|0\rangle$ y $|1\rangle$, y se define matemáticamente con la siguiente expresión:

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.10}$$

Como estamos trabajando con 1 cúbit, solo tenemos dos estados base, $|0\rangle$ y $|1\rangle$. Esto significa que el cúbit puede estar en cualquier superposición de los dos estados base. Por lo tanto, un

estado cualquiera $|\psi\rangle$ lo podríamos definir de la siguiente forma:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \tag{2.11}$$

En una superposición, el coeficiente de cada estado se conoce como la amplitud de ese estado. Como las amplitudes de los estados son números complejos [9], α_0 y α_1 son números complejos en la anterior ecuación ($\alpha_0, \alpha_1 \in \mathbb{C}$). Esto significa que podemos definir estados utilizando números complejos, como el estado $|i\rangle$, que también se encuentra en el punto medio entre $|0\rangle$ y $|1\rangle$:

$$|i\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{i}{\sqrt{2}} |1\rangle = \frac{|0\rangle + i\,|1\rangle}{\sqrt{2}} \tag{2.12}$$

Como ya hemos visto anteriormente, podemos utilizar la notación bra-ket para extraer las amplitudes de un estado cuántico. No obstante, solo utilizaremos esa notación cuando no hayamos definido previamente las amplitudes del estado. Si α_x es la amplitud de $|\psi\rangle$ para $|x\rangle$, tenemos la siguiente equivalencia.

$$\alpha_x = \langle x|\psi\rangle \tag{2.13}$$

2.2.1 Midiendo bits cuánticos

Aunque un cúbit tenga ciertos valores para las amplitudes α_0 y α_1, no es físicamente posible obtener esos valores de forma directa, ya que no son valores observables. Para obtener información a cerca de un cúbit para el que no conocemos sus valores *a priori*, es necesario observarlo. A la hora de observar un cúbit, se mide su estado, que hace que el estado del cúbit colapse a uno de los estados base del sistema ($|0\rangle$ o $|1\rangle$ en el caso de 1 cúbit).

La probabilidad de que un cúbit colapse a cada uno de los estados depende del cuadrado de la magnitud de la amplitud del estado. Un cúbit en estado $\alpha_0 |0\rangle + \alpha_1 |1\rangle$ colapsará a los estados $|0\rangle$ y $|1\rangle$ con las siguientes probabilidades:

$$\begin{aligned} P(|0\rangle) &= |\alpha_0|^2 \\ P(|1\rangle) &= |\alpha_1|^2 \end{aligned} \tag{2.14}$$

Por ejemplo, si medimos el estado $|+\rangle$ que vimos anteriormente, tenemos $|\alpha_0|^2 = 0.5$ y $|\alpha_1|^2 = 0.5$, por lo que el estado tiene un 50% de probabilidad de colapsar a $|0\rangle$ y 50% de colapsar a $|1\rangle$. Como la probabilidad total del sistema siempre tiene que sumar 1, tenemos la siguiente restricción para sistemas de 1 cúbit:

$$P(|0\rangle) + P(|1\rangle) = |\alpha_0|^2 + |\alpha_1|^2 = 1 \tag{2.15}$$

Esta propiedad se conoce como la restricción de normalización, y también se aplica cuando tenemos sistemas más complejos. Como la única manera que tenemos para obtener información del sistema es medir su estado, no podemos conocer los valores exactos de α_0 y α_1. Tan solo podemos conocer el resultado al que han colapsado. Esta es una propiedad fundamental de los sistemas cuánticos, y veremos en algunos apartados ciertas técnicas que se utilizan

para mitigar esta limitación de los sistemas cuánticos y extraer el máximo de información posible del sistema.

2.2.2 Fase global

La fase global del sistema cuántico es la fase común de todas las amplitudes del sistema. Por ejemplo, un sistema con amplitudes $e^{i\theta}/\sqrt{2}$ tendría como fase θ. Normalmente, la fase global de un sistema se define como la fase de la primera amplitud (α_0 para sistemas de un cúbit), de forma que la fase relativa de la primera amplitud siempre sea 0. A partir de aquí, podemos calcular las fases relativas de cada una de las amplitudes del sistema quitando la fase global. Siendo θ_0 la fase de la primera amplitud, utilizando la fórmula de Euler [10] obtenemos las siguientes amplitudes para un sistema de un cúbit:

$$\begin{aligned} \alpha_0' &= \alpha_0 e^{-i\theta_0} \\ \alpha_1' &= \alpha_1 e^{-i\theta_0} \end{aligned} \tag{2.16}$$

Una propiedad muy interesante de los sistemas cuánticos es que la fase global del sistema no afecta al resultado. En otras palabras, modificar la fase de todas las amplitudes de un sistema no varía el resultado obtenido, siempre que se modifiquen todas las fases de igual forma. Esta propiedad se utiliza para simplificar los cálculos en ciertas situaciones, y para facilitar el razonamiento a cerca de los estados de los sistemas.

Por ejemplo, para sistemas de un cúbit, hay infinitos estados equivalentes al estado $|0\rangle$, ya que $e^{i\gamma}|0\rangle$ es exactamente el mismo estado pero variando la fase global por γ radianes. Todas las transformaciones que se pueden realizar en este sistema de 1 cúbit producirán el mismo resultado para $|0\rangle$ y para $e^{i\gamma}|0\rangle$.

2.2.3 Esfera de Bloch

Para representar visualmente el estado de un cúbit, podemos utilizar la esfera de Bloch. Esta es una visualización conveniente que nos ayudará a entender las operaciones que podemos realizar en un cúbit. La esfera de Bloch representa un cúbit como un punto en una esfera de radio 1 en $\mathbb{R}^3$.

Aparentemente, tenemos 4 grados de libertad para un cúbit en un estado $\alpha_0|0\rangle + \alpha_1|1\rangle$, ya que α_0 y α_1 son números complejos con 2 grados de libertad cada uno. No obstante, podemos quitar un grado de libertad aplicando la restricción de normalización $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Podemos eliminar otro grado de libertad quitando la fase del sistema, de forma que la fase relativa de la primera amplitud sea 0. Siguiendo el desarrollo de [11], podemos expresar α_0 y α_1 a partir de los parámetros θ y ϕ:

$$\begin{aligned} \alpha_0 &= \cos\frac{\theta}{2} \\ \alpha_1 &= e^{i\phi}\sin\frac{\theta}{2} \end{aligned} \tag{2.17}$$

Un estado $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, por lo tanto, se expresaría así a partir de θ y ϕ:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \tag{2.18}$$

Interpretando los parámetros θ y ϕ como los ángulos de un sistema de coordenadas esféricas [12], obtenemos el siguiente vector en $\mathbb{R}^3$:

$$\vec{v} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix} \tag{2.19}$$

Tras esto, podemos dibujar el vector resultante sobre una esfera de radio 1. Tras hacerlo, nos queda la siguiente visualización del estado de un cúbit:

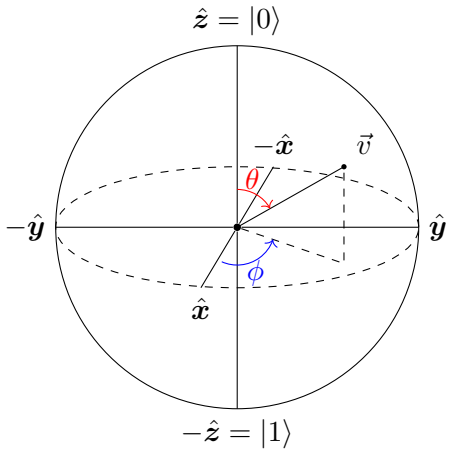

Figura 2.1: Representación de un vector $\vec{v}$ en la esfera de Bloch. Fuente: elaboración propia

Los dos estados base $|0\rangle$ y $|1\rangle$ se encuentran en los polos norte y sur de la esfera, en los dos extremos del eje $\boldsymbol{z}$. En los extremos del eje $\boldsymbol{x}$, nos encontramos los estados $|+\rangle$ y $|-\rangle$, que se definen con la siguiente fórmula:

$$\begin{aligned} |+\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\ |-\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned} \tag{2.20}$$

En los extremos de $\boldsymbol{y}$ se encuentran $|-i\rangle$ e $|i\rangle$, que se definen así:

$$\begin{aligned} |i\rangle &= \frac{|0\rangle + i\,|1\rangle}{\sqrt{2}} \\ |-i\rangle &= \frac{|0\rangle - i\,|1\rangle}{\sqrt{2}} \end{aligned} \tag{2.21}$$

La esfera de Bloch es una representación es muy útil para visualizar 1 cúbit, ya que agrupa todos los estados que tienen la misma fase global. De esta manera, al visualizar un estado o una transformación en la esfera de Bloch, mantendremos sólo la información imprescindible para la visualización, sin tener que preocuparnos de todos los estados equivalentes.

2.2.4 Múltiples cúbits

En la computación clásica, para 2 bits tenemos 4 estados posibles (00, 01, 10 y 11). Con dos cúbits, obtenemos 4 estados base: $|00\rangle$, $|01\rangle$, $|10\rangle$ y $|11\rangle$. Para representar un estado de un sistema de dos cúbits, necesitaremos asociar una amplitud a cada uno de los 4 estados fundamentales. Por lo tanto, el sistema de dos cúbits puede tener un estado como el siguiente:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle \tag{2.22}$$

Esto se puede generalizar fácilmente para n cúbits. Los estados base son todas las combinaciones de 0 y 1 de n elementos ($\{0,1\}^n$), y por lo tanto necesitamos una amplitud α_x para todo $x \in \{0,1\}^n$. Para describir un estado cualquiera $|\psi\rangle$ en un sistema de n cúbits, tendríamos la siguiente expresión:

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x\,|x\rangle \tag{2.23}$$

A la hora de medir estados en sistemas de múltiples cúbits, podemos medir cualquier cúbit del sistema. Al medir, todos los cúbits entrelazados colapsarán a un estado fundamental. En un sistema en el que todos los cúbits están entrelazados, la probabilidad de que el sistema colapse al estado fundamental $x \in \{0,1\}^n$ es la siguiente:

$$P(|x\rangle) = |\alpha_x|^2 \tag{2.24}$$

Como la suma de las probabilidades del sistema tiene que ser 1, tenemos la siguiente restricción de normalización que para los sistema de varios cúbits:

$$\sum_{x\in\{0,1\}^n} P(|x\rangle) = \sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1 \tag{2.25}$$

2.2.4.1 Entrelazamiento de cúbits

En sistemas de varios cúbits, los cúbits se pueden entrelazar mediante varios tipos de interacciones. Varios cúbits están entrelazados cuando están en un estado que no se puede expresar como un producto de los estados de cada uno de los cúbits individuales. Por ejemplo, el estado $|\Phi^+\rangle$, uno de los 4 estados de Bell [13], es un estado entrelazado:

$$\left|\Phi^+\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.26}$$

Para ver si un estado está entrelazado o no, tenemos que ver si es posible descomponer el estado de forma que tengamos amplitudes para cada cúbit. Si el primer cúbit está en el estado $|\psi_0\rangle = \alpha_0\,|0\rangle + \alpha_1\,|1\rangle$ y el segundo en el estado $|\psi_1\rangle = \alpha'_0\,|0\rangle + \alpha'_1\,|1\rangle$, podemos expresar cualquier estado no entrelazado de la siguiente forma:

$$\begin{aligned}|\psi\rangle &= |\psi_1\rangle \otimes |\psi_2\rangle \\ &= (\alpha_0 |0\rangle + \alpha_1 |1\rangle) \otimes (\alpha'_0 |0\rangle + \alpha'_1 |1\rangle) \\ &= \alpha_0\alpha'_0 |00\rangle + \alpha_0\alpha'_1 |01\rangle + \alpha_1\alpha'_0 |10\rangle + \alpha_1\alpha'_1 |11\rangle\end{aligned} \tag{2.27}$$

Para verificar si un estado está entrelazado o no, simplemente tenemos que ver si existe una combinación de valores de α_0, α_1, α'_0 y α'_1 que equivalga a las amplitudes del estado que queremos comprobar. Por ejemplo, para comprobar que $|\Phi^+\rangle$ está entrelazado, podemos realizar el siguiente desarrollo:

$$\begin{aligned}\langle 00|\Phi^+\rangle = \alpha_0\alpha'_0 = 1/\sqrt{2} &\implies \alpha_0 \neq 0 \wedge \alpha'_0 \neq 0 \\ \langle 01|\Phi^+\rangle = \alpha_0\alpha'_1 = 0 &\implies \alpha_0 = 0 \vee \alpha'_1 = 0 \\ \langle 10|\Phi^+\rangle = \alpha_1\alpha'_0 = 0 &\implies \alpha_1 = 0 \vee \alpha'_0 = 0 \\ \langle 11|\Phi^+\rangle = \alpha_1\alpha'_1 = 1/\sqrt{2} &\implies \alpha_1 \neq 0 \wedge \alpha'_1 \neq 0\end{aligned} \tag{2.28}$$

Como podemos ver en el desarrollo anterior, las ecuaciones son contradictorias, ya dos de ellas dicen que ninguna amplitud puede ser 0 y las otras dos requieren que al menos dos de las amplitudes sean 0. Como hay una contradicción, podemos concluir que $|\Phi^+\rangle$ es un estado entrelazado, ya que no se puede expresar como un producto de estados de los cúbits. Todo este proceso también se puede generalizar para n cúbits, aunque el desarrollo se vuelve mucho más complejo.

Existen 4 estados de Bell, todos de ellos entrelazados. Estos estados se utilizan frecuentemente en los algoritmos cuánticos, ya que a partir de ellos se puede llegar a estados con propiedades muy útiles.

$$\begin{aligned}|\Phi^+\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\ |\Phi^-\rangle &= \frac{|00\rangle - |11\rangle}{\sqrt{2}} \\ |\Psi^+\rangle &= \frac{|01\rangle + |10\rangle}{\sqrt{2}} \\ |\Psi^-\rangle &= \frac{|01\rangle - |10\rangle}{\sqrt{2}}\end{aligned} \tag{2.29}$$

Medir un cúbit de un estado entrelazado colapsa todos los cúbits que estaban entrelazados en ese estado. Esto ocurre de forma instantánea, incluso si los cúbits se encuentran a mucha distancia [14].

2.2.5 Notación vectorial

Para facilitar la representación de operaciones sobre cúbits, representaremos el estado de un sistema de uno o más cúbits como un vector de amplitudes, ordenado de la siguiente manera:

$$\alpha_0 |0\rangle + \alpha_1 |1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \tag{2.30}$$

Como muchas de las operaciones básicas que podemos aplicar sobre cúbits son lineares, esto nos permitirá expresar esas operaciones como matrices. A continuación se puede ver la notación para las superposiciones de 2 cúbits:

$$\alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \tag{2.31}$$

Para casos con más de dos cúbits, las amplitudes del estado estarán en el mismo orden, siendo la primera amplitud la correspondiente al estado de sólo ceros ($|0 \dots 0\rangle$) y la última la correspondiente al estado de sólo unos ($|1 \dots 1\rangle$):

$$\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle = \begin{pmatrix} \alpha_{0\dots0} \\ \alpha_{0\dots1} \\ \dots \\ \alpha_{1\dots0} \\ \alpha_{1\dots1} \end{pmatrix} \tag{2.32}$$

2.3 Puertas lógicas cuánticas

En la computación clásica, las operaciones fundamentales que podemos realizar sobre bits son las puertas lógicas. De igual manera, las operaciones básicas que podemos aplicar a los cúbits son las puertas lógicas cuánticas. Como los cúbits son más complejos que los bits clásicos, muchas de estas puertas también son más complicadas que las puertas lógicas clásicas.

Para representar las operaciones que veremos a continuación, utilizaremos matrices. Representaremos cada puerta como una matriz cuadrada, que al multiplicar un vector de amplitudes produzca el vector de amplitudes resultante tras aplicar la puerta. Por ejemplo, la siguiente matriz representa la puerta identidad, que deja un cúbit con las mismos amplitudes:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.33}$$

Para mantener la restricción de normalización, todas las operaciones que se pueden realizar sobre cúbits tienen que ser matrices unitarias. Una matriz U es una matriz unitaria si y solo si su inversa es igual a su transpuesta conjugada. En otras palabras, tiene que cumplir la siguiente restricción:

$$UU^\dagger = U^\dagger U = I \tag{2.34}$$

Como cada cúbit está formado por dos números complejos, las puertas cuánticas de 1 cúbit serán necesariamente matrices 2×2 de números complejos. Para puertas de n cúbits, tendremos matrices de $2^n \times 2^n$. Como todas las matrices unitarias tienen una matriz inversa, significa que cualquier puerta lógica cuántica es reversible, y que por lo tanto cualquier ope-

ración que realicemos sobre un sistema cuántico se puede deshacer. La única excepción es medir el estado del sistema, que es una operación irreversible.

A continuación, comentaremos las puertas lógicas cuánticas más habituales, junto con sus propiedades y usos.

2.3.1 Puertas de Pauli

Las puertas de Pauli (las puertas X, Y y Z) son tres puertas que actúan sobre un único cúbit. Estas puertas corresponden a una rotación alrededor de los ejes $\boldsymbol{x}$, $\boldsymbol{y}$ y $\boldsymbol{z}$ por π radianes (180 grados) en la esfera de Bloch. Como la puerta X rota alrededor del eje $\boldsymbol{x}$, realizará las transformaciones $|0\rangle \mapsto |1\rangle$ y $|1\rangle \mapsto |0\rangle$ en la esfera de Bloch:

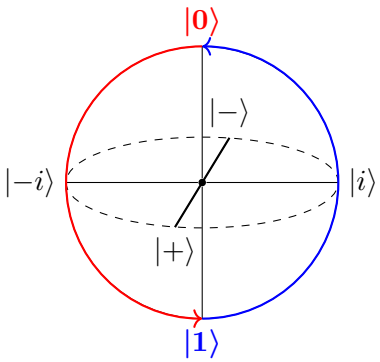

Figura 2.2: Puerta X en la esfera de Bloch. Fuente: elaboración propia

Como esta puerta "invierte" el bit del cúbit también se conoce como la puerta NOT cuántica. Además, realiza $|i\rangle \mapsto |-i\rangle$ y $|-i\rangle \mapsto |i\rangle$ en la esfera de Bloch, dejando $|+\rangle$ y $|-\rangle$ sin modificar. La representación matricial de la puerta X es la siguiente:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.35}$$

La puerta Y es como la X, solo que intercambia $|0\rangle$ con $|1\rangle$ y $|+\rangle$ con $|-\rangle$, y deja $|i\rangle$ y $|-i\rangle$ sin modificar en la esfera de Bloch.

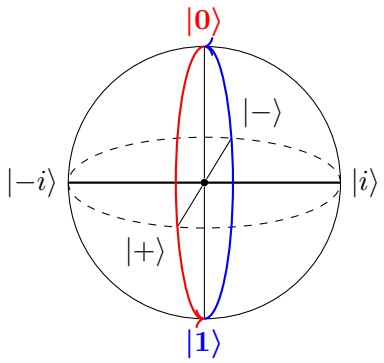

Figura 2.3: Puerta Y en la esfera de Bloch. Fuente: elaboración propia

La representación matricial de la puerta Y es la siguiente:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.36}$$

Cabe destacar que la puerta Y, a diferencia de la puerta X, introduce fase al sistema. Concretamente, como hace las transformaciones $|0\rangle \mapsto -i\,|-1\rangle$ y $|-1\rangle \mapsto i\,|0\rangle$, introduce fases i y $-i$. Esto significa que para sistemas de múltiples cúbits tendremos que tener esta fase en cuenta a la hora de operar utilizando esta puerta.

La puerta Z no modifica $|0\rangle$ y $|1\rangle$, e intercambia $|+\rangle$ con $|-\rangle$ y $|i\rangle$ con $|-i\rangle$:

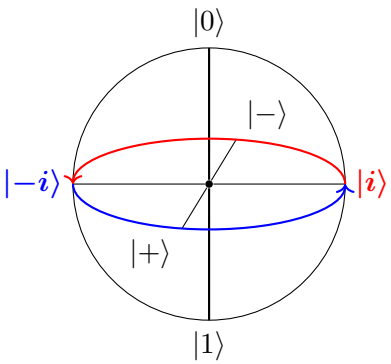

Figura 2.4: Puerta Z en la esfera de Bloch. Fuente: elaboración propia

La puerta Z corresponde con la siguiente matriz:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.37}$$

Como se puede ver en la matriz, la puerta Z también introduce fase, ya que hace $|1\rangle \mapsto -|1\rangle$. Como las 3 puertas son rotaciones de 180 grados, son involutivas, por lo que tenemos $X^2 = Y^2 = Z^2 = I$. A continuación se pueden ver los resultados exactos (con fase) de aplicar las 3 puertas a los 6 estados principales de la esfera de Bloch:

$\|\psi\rangle$	$X\|\psi\rangle$	$Y\|\psi\rangle$	$Z\|\psi\rangle$
$\|0\rangle$	$\|1\rangle$	$i\|1\rangle$	$\|0\rangle$
$\|1\rangle$	$\|0\rangle$	$-i\|0\rangle$	$-\|1\rangle$
$\|-\rangle$	$-\|-\rangle$	$i\|+\rangle$	$\|+\rangle$
$\|+\rangle$	$\|+\rangle$	$-i\|-\rangle$	$\|-\rangle$
$\|-i\rangle$	$i\|i\rangle$	$-\|-i\rangle$	$\|i\rangle$
$\|i\rangle$	$\|-i\rangle$	$\|i\rangle$	$\|-i\rangle$

Tabla 2.2: Efectos de las puertas X, Y y Z. Fuente: elaboración propia

2.3.2 Puerta Hadamard

La puerta Hadamard es otra de las puertas fundamentales cuánticas. Esta puerta, al igual que las puertas de Pauli, equivale a una rotación de π radianes en la esfera de Bloch. No obstante, esta puerta realiza la rotación a través de un eje situado a 45 grados entre el eje $\boldsymbol{x}$ y el eje $\boldsymbol{z}$ (el eje $\boldsymbol{(\hat{x}+\hat{z})/\sqrt{2}}$). La visualización en la esfera de Bloch de esta puerta se puede ver a continuación:

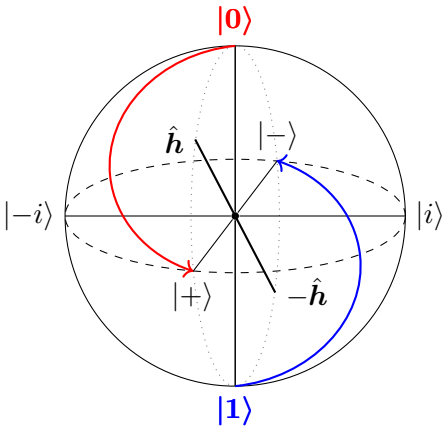

Figura 2.5: Puerta H en la esfera de Bloch. Fuente: elaboración propia

Como se puede ver, esta puerta intercambia los estados $|0\rangle$ y $|1\rangle$ con los estados $|+\rangle$ y $|-\rangle$. Los estados $|i\rangle$ y $|-i\rangle$ se intercambian entre ellos. La matriz que define a la puerta Hadamard es la siguiente:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.38}$$

La puerta, como representa una rotación de π radianes, es involutiva ($H^2 = I$). Sobre un cúbit realiza la transformación $(\alpha_0, \alpha_1) \mapsto (\alpha_0 + \alpha_1, \alpha_0 - \alpha_1)/\sqrt{2}$. Los efectos que tiene la puerta sobre los 6 estados principales de la esfera de Bloch son los siguientes:

$\|\psi\rangle$	$H\|\psi\rangle$
$\|0\rangle$	$\|+\rangle$
$\|1\rangle$	$\|-\rangle$
$\|-\rangle$	$\|1\rangle$
$\|+\rangle$	$\|0\rangle$
$\|-i\rangle$	$(1-i)\|i\rangle$
$\|i\rangle$	$(1+i)\|-i\rangle$

Tabla 2.3: Efectos de la puerta H. Fuente: elaboración propia

Como se puede ver, esta puerta intercambia los estados $|0\rangle$ y $|1\rangle$ por los estados $|+\rangle$ y $|-\rangle$ sin introducir fase, por lo que se dice que cambia la base del sistema de $\{|0\rangle, |1\rangle\}$ a $\{|+\rangle, |-\rangle\}$. La base $\{|+\rangle, |-\rangle\}$ se conoce como la base de Hadamard.

2.3.2.1 Puertas de cambio de base

Generalizando un poco más lo visto con la puerta Hadamard, se pueden definir las puertas de cambio de base. Esta es una familia de puertas cuánticas que cambia la base del sistema. $B(|\phi\rangle, |\psi\rangle)$ es la puerta que cambia la base de $\{|0\rangle, |1\rangle\}$ a $\{|\phi\rangle, |\psi\rangle\}$, y se puede definir de la siguiente forma:

$$B(|\phi\rangle, |\psi\rangle) = \begin{pmatrix} \langle 0|\phi\rangle & \langle 1|\phi\rangle \\ \langle 0|\psi\rangle & \langle 1|\psi\rangle \end{pmatrix} \tag{2.39}$$

Esta puerta es una puerta cuántica válida si y solo si $|\phi\rangle$ y $|\psi\rangle$ son estados cuánticos válidos y son vectores linealmente independientes [15]. Este tipo de base se puede generalizar para n cúbits con la siguiente matriz $2^n \times 2^n$:

$$B(|\psi_1\rangle, \ldots, |\psi_n\rangle) = \begin{pmatrix} \langle 0\ldots0|\psi_1\rangle & \ldots & \langle 1\ldots1|\psi_1\rangle \\ \vdots & \ddots & \vdots \\ \langle 0\ldots0|\psi_n\rangle & \ldots & \langle 1\ldots1|\psi_n\rangle \end{pmatrix} \tag{2.40}$$

Utilizando esta notación, podríamos definir la puerta H como $B(|+\rangle, |-\rangle)$. Otra base muy importante es la base de Bell $\{\Phi^+, \Psi^+, \Phi^-, \Psi^-\}$, cuya puerta de cambio de base es la siguiente:

$$\begin{aligned} Bell = B(\Phi^+, \Phi^-, \Psi^+, \Psi^-) &= \begin{pmatrix} \langle 00|\Phi^+\rangle & \langle 01|\Phi^+\rangle & \langle 10|\Phi^+\rangle & \langle 11|\Phi^+\rangle \\ \langle 00|\Psi^+\rangle & \langle 01|\Psi^+\rangle & \langle 10|\Psi^+\rangle & \langle 11|\Psi^+\rangle \\ \langle 00|\Phi^-\rangle & \langle 01|\Phi^-\rangle & \langle 10|\Phi^-\rangle & \langle 11|\Phi^-\rangle \\ \langle 00|\Psi^-\rangle & \langle 01|\Psi^-\rangle & \langle 10|\Psi^-\rangle & \langle 11|\Psi^-\rangle \end{pmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \end{aligned} \tag{2.41}$$

2.3.3 Puertas de desplazamiento de fase

Las puertas de desplazamiento de fase son una familia de puertas cuánticas que alteran la fase del cúbit al que se aplican, realizando $(\alpha_0, \alpha_1) \mapsto (\alpha_0, e^{i\varphi}\alpha_1)$. Las puertas de desplazamiento de fase se describen con la siguiente matriz, donde φ es el desplazamiento de la puerta:

$$P(\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} \tag{2.42}$$

Este tipo de puertas equivalen también a una rotación de φ radianes en la esfera de Bloch alrededor del eje $\boldsymbol{z}$. Las puertas de desplazamiento de fase más utilizadas son la puerta T, donde $\varphi = \pi/4$; la puerta S, donde $\varphi = \pi/2$; y la puerta de Pauli Z, donde $\varphi = \pi$.

$$\begin{aligned} T = P\left(\frac{\pi}{4}\right) &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \\ S = P\left(\frac{\pi}{2}\right) &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \\ Z = P(\pi) &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \end{aligned} \tag{2.43}$$

Los efectos de estas puertas sobre los estados principales de la esfera de Bloch se pueden ver a continuación:

$\lvert\psi\rangle$	$T\lvert\psi\rangle$	$S\lvert\psi\rangle$	$P(\varphi)\lvert\psi\rangle$
$\lvert 0\rangle$	$\lvert 0\rangle$	$\lvert 0\rangle$	$\lvert 0\rangle$
$\lvert 1\rangle$	$e^{i\pi/4}\lvert 1\rangle$	$i\lvert 1\rangle$	$e^{i\varphi}\lvert 1\rangle$
$\lvert -\rangle$	$(1, -e^{i\pi/4})/\sqrt{2}$	$\lvert -i\rangle$	$(1, e^{i\varphi})/\sqrt{2}$
$\lvert +\rangle$	$(1, e^{i\pi/4})/\sqrt{2}$	$\lvert i\rangle$	$(1, e^{i\varphi})/\sqrt{2}$
$\lvert -i\rangle$	$(1, -ie^{i\pi/4})/\sqrt{2}$	$\lvert +\rangle$	$(1, -ie^{i\varphi})/\sqrt{2}$
$\lvert i\rangle$	$(1, ie^{i\pi/4})/\sqrt{2}$	$\lvert -\rangle$	$(1, ie^{i\varphi})/\sqrt{2}$

Tabla 2.4: Efectos de las puertas de desplazamiento de fase. Fuente: elaboración propia

2.3.4 Puerta SWAP

La puerta $SWAP$ es una puerta que opera sobre dos cúbits. La puerta intercambia los valores de los cúbits. Si un cúbit es $|0\rangle$ y el otro es $|1\rangle$, se invertirán los valores. Como la puerta opera en varios cúbits, no la podemos representar mediante la esfera de Bloch. La matriz que representa esta puerta es la siguiente:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.44}$$

Si vemos como afecta a los 4 estados base para sistemas de 2 cúbits, tenemos la siguiente tabla:

$\|\psi\rangle$	$SWAP\|\psi\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|10\rangle$
$\|10\rangle$	$\|01\rangle$
$\|11\rangle$	$\|11\rangle$

Tabla 2.5: Efectos de la puerta $SWAP$. Fuente: elaboración propia

Como se puede ver en la tabla la puerta intercambia los valores del primer y del segundo cúbit, respecto a los estados $|0\rangle$ y $|1\rangle$. Cabe destacar que la puerta es simétrica respecto al primer y segundo cúbit, lo que implica que el efecto sobre cada uno de los cúbits es el mismo incluso si los intercambiamos entre sí. La puerta $SWAP$ también es involutiva, ya que $SWAP^2 = I$.

2.3.5 Puertas controladas

Las puertas controladas son una familia de puertas cuánticas que actúan en dos o más cúbits, donde algunos de los cúbits controlan si la operación se realiza o no. Por ejemplo, la puerta CX (o $CNOT$) es la versión controlada de la puerta de Pauli X. Si el primer cúbit está en $|1\rangle$, aplicará la puerta X al segundo cúbit. Si está en $|0\rangle$, el segundo cúbit se quedará sin modificar. Básicamente, realiza la siguiente transformación de estados:

$\|\psi\rangle$	$CX\|\psi\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|11\rangle$
$\|11\rangle$	$\|10\rangle$

Tabla 2.6: Efectos de la puerta CX. Fuente: elaboración propia

Esto se puede describir matemáticamente con la transformación $CX|a,b\rangle \mapsto |a, a \oplus b\rangle$, siendo $\oplus$ la puerta XOR. Es por esto que a veces se conoce esta puerta como la XOR cuántica. La representación matricial de la puerta CX es la siguiente:

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.45}$$

Como se puede ver, la matriz contiene la puerta de Pauli X en la esquina abajo derecha. Esto no es casualidad, ya que es como se construyen de manera general las puertas controladas. Si tenemos una puerta cualquiera de 1 cúbit U, entonces la versión controlada de la puerta

U sería la siguiente matriz:

$$CU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{pmatrix} \tag{2.46}$$

Esta puerta CU aplicará la puerta U solo cuando el primer cúbit esté en el estado $|1\rangle$, y no la aplicará cuando está en $|0\rangle$. La puerta CU realiza la siguiente transformación:

$\|\psi\rangle$	$CU\|\psi\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|1\rangle \otimes U\|0\rangle$
$\|11\rangle$	$\|1\rangle \otimes U\|1\rangle$

Tabla 2.7: Efectos de la puerta CU. Fuente: elaboración propia

Este proceso se puede aplicar para cualquier puerta U de 1 cúbit. También se puede construir de forma similar matrices con más de 1 cúbits de control. Por ejemplo, podríamos definir la puerta CCU como una puerta doble-controlada, que requiere que dos bits de control en vez de uno para realizar la puerta U. Esta puerta se definiría de la siguiente manera:

$$CCU = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & 0 & 0 & 0 & 0 & u_{10} & u_{11} \end{pmatrix} \tag{2.47}$$

De forma similar, también podemos generalizar la construcción de una puerta controlada para una puerta con cualquier número de cúbits. Sea W la matriz de una puerta de n cúbits, y I la puerta identidad de n cúbits, obtendríamos la siguiente matriz para la puerta CW, que sería una puerta de $n+1$ cúbits:

$$CW = \begin{pmatrix} I & 0 \\ 0 & W \end{pmatrix} \tag{2.48}$$

Para denotar una puerta U-controlada con n bits de control, utilizaremos la notación C^nU.

2.3.5.1 Puerta Toffoli

Una de las puertas controladas más utilizadas es la puerta Toffoli. La puerta Toffoli, el la puerta de Pauli X doble controlada. La matriz que la define es la siguiente:

$$CCX = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.49}$$

Como hemos añadido dos controles a una puerta de un cúbit, esta puerta es de 3 cúbits. En la tabla 2.8 se puede ver los efectos de esta puerta sobre los estados base para sistemas de 3 cúbits.

$\|\psi\rangle$	$CCX\|\psi\rangle$
$\|000\rangle$	$\|000\rangle$
$\|001\rangle$	$\|001\rangle$
$\|010\rangle$	$\|010\rangle$
$\|011\rangle$	$\|011\rangle$
$\|100\rangle$	$\|100\rangle$
$\|101\rangle$	$\|101\rangle$
$\|110\rangle$	$\|111\rangle$
$\|111\rangle$	$\|110\rangle$

Tabla 2.8: Efectos de la puerta Toffoli. Fuente: elaboración propia

Esta puerta se utiliza normalmente para implementar ciertas puertas de lógica clásica, ya que no se pueden implementar de forma directa con ordenadores cuánticos [16].

2.3.5.2 Puerta CZ

Otra puerta controlada muy utilizada es la puerta CZ, la puerta controlada de la puerta Pauli Z. Su matriz es la siguiente:

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{2.50}$$

Sus efectos se pueden ver a continuación:

$\|\psi\rangle$	$CZ\|\psi\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|10\rangle$
$\|11\rangle$	$\|1,-1\rangle$

Tabla 2.9: Efectos de la puerta CZ. Fuente: elaboración propia

2.3.5.3 Puertas de desplazamiento de fase controladas

De forma similar a la puerta CZ, también se pueden generar versiones controladas de las otras puertas de desplazamiento de fase:

$$CP(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix} \tag{2.51}$$

A partir de $CP(\varphi)$, podemos construir las puertas CT y CS:

$$\begin{aligned} CT = CP(\frac{\pi}{4}) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \\ CS = CP(\frac{\pi}{2}) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix} \end{aligned} \tag{2.52}$$

Los efectos de estas puertas son los siguientes:

$\|\psi\rangle$	$CT\|\psi\rangle$	$CS\|\psi\rangle$	$CP(\varphi)\|\psi\rangle$
$\|00\rangle$	$\|00\rangle$	$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$	$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|10\rangle$	$\|10\rangle$	$\|10\rangle$
$\|11\rangle$	$\|1,i\rangle$	$\left\|1,e^{i\pi/4}\right\rangle$	$\left\|1,e^{i\varphi}\right\rangle$

Tabla 2.10: Efectos de las puertas de desplazamiento de fase controladas. Fuente: elaboración propia

2.3.5.4 Puerta CSWAP

La puerta *CSWAP* es la puerta *SWAP*, pero con un cúbit de control. Esta puerta solo intercambia el valor de los dos últimos cúbits si el cúbit de control vale $|1\rangle$. Se define con la siguiente matriz:

$$CSWAP = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.53}$$

Esta puerta se utiliza principalmente para estimar el producto interior entre dos vectores cuánticos. Este tipo de estimación se conoce como *swap test*, y es la base de muchos algoritmos cuánticos que trabajan sobre vectores o matrices. En el apartado 4.2.1, podemos ver como se puede utilizar este tipo de estimaciones para construir un algoritmo cuántico de multiplicación de matrices.

Los efectos de esta puerta sobre un sistema de 3 cúbits se pueden ver en la tabla 2.11.

$\|\psi\rangle$	$CSWAP\|\psi\rangle$
$\|000\rangle$	$\|000\rangle$
$\|001\rangle$	$\|001\rangle$
$\|010\rangle$	$\|010\rangle$
$\|011\rangle$	$\|011\rangle$
$\|100\rangle$	$\|100\rangle$
$\|101\rangle$	$\|110\rangle$
$\|110\rangle$	$\|101\rangle$
$\|111\rangle$	$\|111\rangle$

Tabla 2.11: Efectos de la puerta CSWAP. Fuente: elaboración propia

2.3.6 Puertas compuestas

Las puertas cuánticas también se pueden componer a partir de otras puertas cuánticas. Como los efectos de las puertas se pueden representar con matrices, utilizando operaciones matriciales podemos formar puertas nuevas a partir de otras.

2.3.6.1 Puertas con exponentes

La forma mas sencilla de formar puertas cuánticas compuestas es poner múltiples puertas cuánticas en serie. Por ejemplo, podríamos formar una puerta U que aplique los efectos de dos puertas U_1 y U_2 utilizando la multiplicación de matrices:

$$U = U_2\, U_1 \tag{2.54}$$

Como son matrices, para que se aplique primero la puerta U_1 y después la U_2 tendremos que ponerlas de derecha a izquierda. Esto se puede generalizar fácilmente para cualquier número de puertas. Si tenemos las puertas $U_1 \dots U_n$, la puerta que equivale a aplicar U_1, luego U_2 y

así hasta aplicar U_n es la siguiente:

$$U = \prod_{i=1}^{n} U_{n+1-i} \tag{2.55}$$

Utilizando esta misma fórmula, podemos encontrar también el resultado de aplicar una puerta múltiples veces, U^n. La potencia de esta fórmula es que, como las puertas cuánticas válidas son matrices de números complejos, para que U^n sea una puerta válida n puede ser cualquier número complejo. Esto es especialmente útil para realizar puertas con exponentes fraccionales. Por ejemplo, la puerta $\sqrt{X} = X^{1/2}$ es una puerta que, tras aplicarla dos veces, realiza los mismos efectos que la puerta X. La representación matricial de la puerta se puede obtener utilizando álgebra matricial de forma sencilla:

$$\sqrt{X} = \sqrt{\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}} = \frac{1+i}{2} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \tag{2.56}$$

Otras puertas con exponentes fraccionales muy utilizadas son las puertas $\sqrt{Y}$ y $\sqrt{H}$:

$$\begin{aligned} \sqrt{Y} &= \sqrt{\begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix}} = \frac{1+i}{2} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \\ \sqrt{H} &= \sqrt{\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}} = \frac{1-i}{4} \begin{pmatrix} \sqrt{2}+2i & \sqrt{2} \\ \sqrt{2} & -\sqrt{2}+2i \end{pmatrix} \end{aligned} \tag{2.57}$$

2.3.6.2 Puertas en paralelo

Otro tipo de puerta compuesta muy útil son las puertas paralelas, que aplican los efectos de puertas en paralelo a varios cúbits. Las puertas paralelas se pueden construir utilizando el producto tensorial. Por ejemplo, si quisiéramos aplicar la puerta X y Z en paralelo (una a cada cúbit), tendríamos la siguiente puerta:

$$X \otimes Z = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \tag{2.58}$$

La puerta $X \otimes Z$ es una puerta de dos cúbits, y sus efectos son equivalentes a los de aplicar la puerta X al primer cúbit y la puerta Z al segundo. Esto se puede generalizar para cualquier número de puertas, de forma que para aplicar las puertas $U_1 \dots U_n$ en paralelo a n cúbits de forma de que la puerta U_i se aplique al cúbit i nos quedaría la siguiente puerta:

$$U = \bigotimes_{i=1}^{n} U_n \tag{2.59}$$

Si cada puerta U_i es una puerta de n_i cúbits, entonces la puerta paralela U será una puerta de $\sum n_i$ cúbits. Para construir una puerta que aplique la misma puerta U en paralelo a n cúbits, utilizaríamos la siguiente fórmula:

$$U^{\otimes n} = \bigotimes_{1}^{n} U = \underbrace{U \otimes U \otimes \cdots \otimes U}_{\text{n veces}} \tag{2.60}$$

La puerta paralela más útil que sigue este patrón es la transformada de Hadamard, una puerta que aplica la puerta H en paralelo a n cúbits. Por ejemplo, la versión de dos cúbits es la siguiente:

$$H^{\otimes 2} = H \otimes H = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{2.61}$$

Las familia de puertas $H^{\otimes n}$ normalmente se utiliza para inicializar estados cuánticos, ya que tras aplicarla a un estados formado por n cúbits inicializados en $|0\rangle$, deja el sistema en una superposición en la que todos los posibles estados tienen la misma amplitud, $1/\sqrt{2^n}$:

$$H^{\otimes n} |0^n\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \tag{2.62}$$

2.4 Circuitos cuánticos

Cuando tenemos muchas puertas cuánticas en el mismo algoritmo, representarlas con la notación matricial o la notación bra-ket se vuelve muy tedioso, especialmente cuando estamos trabajando con puertas que operan sobre muchos cúbits. Para intentar mitigar esto, existe una notación que representa un algoritmo cuántico como un circuito de puertas cuánticas.

2.4.1 Cables cuánticos

En un circuito cuántico, cada cúbit se representa con un cable. Si queremos representar un múltiples cúbits con un cable, especificaremos la cantidad de cúbits del cable con una pequeña línea diagonal. Además, a la izquierda del cable pondremos el valor inicial en el que empieza el cable, ya sea un valor constante ($|0\rangle$, $|1\rangle$, etc.) o una variable ($|\phi\rangle$, $|\psi\rangle$, etc).

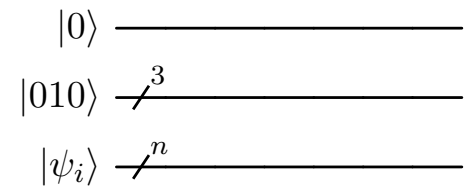

Figura 2.6: Circuito cuántico con tres cables de 1, 3 y n cúbits. Fuente: elaboración propia

En algunas ocasiones también es necesario representar bits clásicos en diagramas de circuitos cuánticos. Para esto existe un tipo de cable para bits clásicos. Los cables de bits se representan con dos líneas, mientras que los cables de cúbits se representan con una línea:

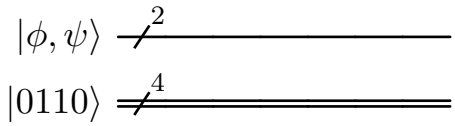

Figura 2.7: Circuito con un cable de cúbits (arriba) y de bits (abajo). Fuente: elaboración propia

2.4.2 Puertas en circuitos cuánticos

Para aplicar puertas lógicas a los cúbits de un cable, pondremos el nombre de la puerta dentro de un rectángulo que intersecte al cable que afecta. Si la puerta afecta a los cúbits de varios cables, haremos que el rectángulo intersecte a todos los cables que afecte:

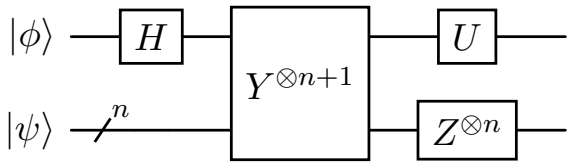

Figura 2.8: Circuito cuántico con varias puertas lógicas. Fuente: elaboración propia

Podemos tener puertas en serie poniendo varias puertas en el mismo cable, y puertas en paralelo poniendo cada puerta en cables diferentes. Para puertas controladas, se utiliza un punto negro para indicar el bit (o bits) de control, y el cuadrado con el nombre de la puerta para indicar la puerta en sí:

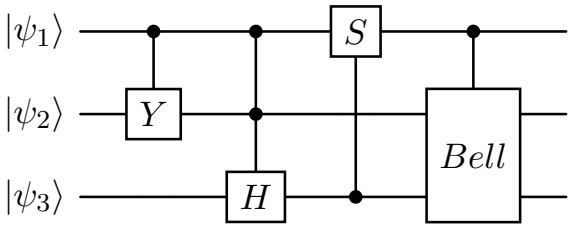

Figura 2.9: Circuito cuántico con las puertas CY, CCH, CS y *CBell*. Fuente: elaboración propia

La gran mayoría de puertas se representan utilizando el rectángulo junto con el nombre de la puerta. No obstante, existen algunas excepciones importantes.

2.4.2.1 Puertas X, CX y Toffoli

La puerta X en circuitos cuánticos se representa utilizando el símbolo $\oplus$. Como las puertas CX y Toffoli derivan de la puerta X, su representación en circuitos cuánticos también utiliza el símbolo $\oplus$, al igual que el resto de las puertas C^nX.

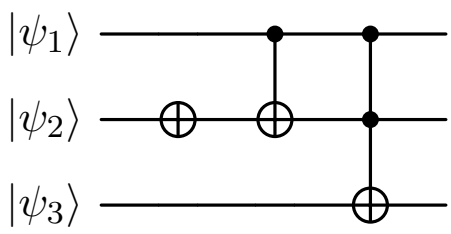

Figura 2.10: Circuito con X (izquierda), CX (centro) y Toffoli (derecha). Fuente: elaboración propia

2.4.2.2 Puerta CZ

La puerta CZ, a diferencia de la puerta de Pauli Z, se representa utilizando dos puntos:

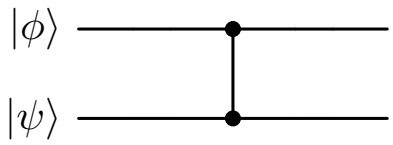

Figura 2.11: Circuito cuántico con una puerta CZ. Fuente: elaboración propia

Con esta notación, es ambiguo cuál de los dos cúbits es el control, y cuál es el cúbit afectado por la puerta. No obstante, como la puerta CZ produce los mismos efectos sin importar el orden de los cúbits, no importa esta ambigüedad. Esto también ocurre cuando se controla la puerta Z con más cúbits, como en la familia de puertas C^nZ.

2.4.2.3 Puertas SWAP y CSWAP

La puerta *SWAP* se representa con una línea que une dos símbolos $\times$, uno en cada cúbit que intercambia. La puerta *CSWAP* utiliza la misma notación, pero con un punto indicando el cúbit de control:

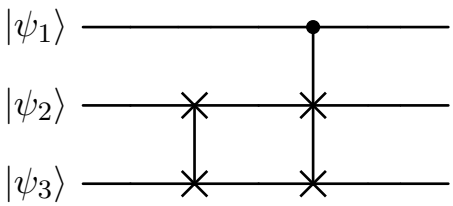

Figura 2.12: Circuito con SWAP (izquierda) y CSWAP (derecha). Fuente: elaboración propia

2.4.3 Medidores

Para obtener el resultado de los cables, necesitaremos medir los cúbits de los cables. Para medir un cable, se utiliza una "puerta" especial que representa el medidor. Para reflejar que los cúbits de un cable han sido medidos, cambiaremos el tipo de cable de cúbit a bit clásico,

ya que el resultado de medir cúbits es uno de los estados base, que se puede representar utilizando bits clásicos:

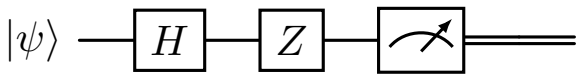

Figura 2.13: Circuito cuántico con un medidor. Fuente: elaboración propia

Estos bits clásicos, aunque normalmente solo se utilizan para almacenar el resultado del circuito, a veces se pueden utilizar para controlar si ciertas puertas posteriores se ejecutan o no, o cuántas veces se ejecuta una puerta. Esto se conoce como un control clásico, y se representa en circuitos cuánticos de forma similar a las puertas controladas, pero con el cúbit de control en un cable de bits:

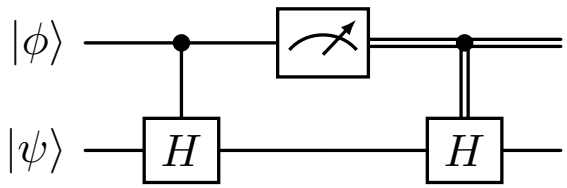

Figura 2.14: Circuito con CH, un medidor y H con control clásico. Fuente: elaboración propia

Este tipo de control no se suele utilizar, ya que tienen exactamente el mismo efecto que utilizar una puerta controlada antes de medir el cúbit. Por ejemplo, las dos puertas H de la figura 2.14 tienen un efecto idéntico sobre los dos cúbits. Esto lo sabemos gracias al principio de la medida diferida (*Deferred Measurement Principle*) [17].

3 Inteligencia artificial

En el campo de la inteligencia artificial se utilizan técnicas y arquitecturas para la resolución de problemas tan complejos que desarrollar un programa tradicional para resolverlos es inviable. Entre estos problemas se incluye la traducción de lenguaje natural, el reconocimiento de imágenes y vídeo, y la detección de patrones y/o semejanzas. Mediante ciertos mecanismos que veremos más adelante, es posible entrenar estas redes para que aprendan ciertos patrones. Hay varios tipos de aprendizaje que se pueden utilizar a la hora de entrenar modelos de inteligencia artificial:

- Aprendizaje supervisado: consiste en proveer a la red de muchos ejemplos etiquetados. A partir de estos ejemplos, la red es capaz de ir disminuyendo su error respecto a los ejemplos, y ir prediciendo mejor el conjunto de datos del que se extrajeron los ejemplos. Como para generar modelos muy precisos o complejos se requieren muchos datos, se requiere mucho tiempo de entrenamiento para que el modelo aprenda de todos los ejemplos. Si no se proveen de suficientes datos, o los datos no son de suficientemente buena calidad, se pueden obtener resultados subóptimos [18].

- Aprendizaje no supervisado: en el aprendizaje no supervisado, el modelo de machine learning extrae patrones de datos no etiquetados, mediante el descubrimiento de patrones o semejanzas en los datos. Aunque no requiera tantos datos como el aprendizaje supervisado, extraer información útil del modelo no es tan sencillo, ya que las categorías o asociaciones producidas son opacas [19]. También es posible que el modelo encuentre patrones que carezcan de relevancia real a la hora de aplicarlos al análisis que se pretendía hacer.

- Aprendizaje por refuerzo: el modelo aprende mediante la interacción con su entorno, y es recompensado o penalizado dependiendo en lo que haga. Estos algoritmos son específicos a un problema en concreto, pero permiten la optimización de tareas muy complicadas con la programación de unas heurísticas de recompensa/penalización relativamente simples. El modelo, mediante prueba y error, va aprendiendo poco a poco el entorno en el que está interactuando, y poco a poco va afinándose a las características de la función de recompensa y penalización [20].

La gran mayoría de arquitecturas de inteligencia artificial actuales están basadas en redes neuronales. Estas redes son conjuntos de neuronas artificiales, que son nodos que calculan un valor a partir de sus conexiones con otras neuronas artificiales. Este tipo de arquitecturas están inspirados en las redes neuronales biológicas encontradas en los cerebros y sus conexiones neuronales. Primero estudiaremos la estructura de una neurona artificial individual, y luego estudiaremos las redes neuronales en sí.

3.1 Neuronas artificiales

En machine learning, una neurona es un nodo de una red neuronal. Cada neurona recibe ciertas entradas y produce una única salida, su valor de activación. La neurona calcula este valor a partir de las entradas recibidas, multiplicando cada entrada por su peso correspondiente. También se le suma a la activación un valor fijo llamado *bias*. El resultado se pasa por la función de activación, que a partir del resultado de las sumas produce la salida de la neurona. Sea σ la función de activación, x las entradas, w los pesos y b el *bias*, la salida de una neurona se calcula utilizando la siguiente fórmula:

$$y = \sigma\left(b + \sum_{i=1}^{n} w_i x_i\right) \tag{3.1}$$

Para simplificar la expresión anterior, normalmente se añade una "entrada" x_0 cuyo valor es siempre 1, y se utiliza w_0 como *bias*:

$$y = \sigma\left(\sum_{i=0}^{n} w_i x_i\right) \tag{3.2}$$

Podemos representar visualmente la ecuación anterior de la siguiente forma:

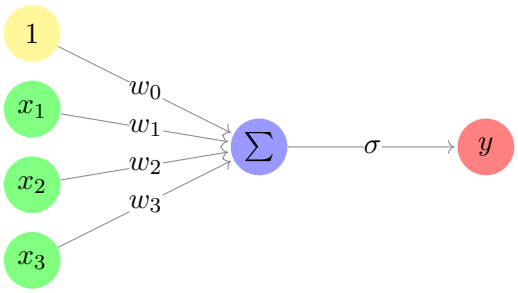

Figura 3.1: Visualización de una neurona con 3 entradas. Fuente: elaboración propia

Existen muchas funciones de activación, teniendo cada una sus usos. La gran mayoría son funciones no lineales, que se utilizan para hacer que la salida de las neuronas deje de tener una relación lineal con las entradas. Algunas de las funciones de activación más utilizadas son las siguientes:

- Lineal: $\sigma(x) = x$.
- Sigmoide: $\sigma(x) = 1/(1 + e^{-x})$.
- Softplus: $\sigma(x) = \ln(1 + e^{x})$
- Tangente hiperbólica: $\sigma(x) = \tanh(x)$
- ReLU: $\sigma(x) = \max(0, x)$

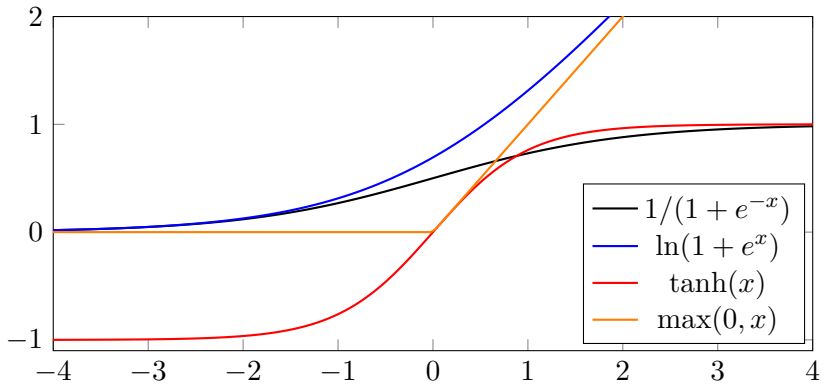

Figura 3.2: Representación gráfica de algunas funciones de activación. Fuente: elaboración propia

Cada una de las funciones tiene sus ventajas y sus desventajas. Las funciones que tienen un rango finito tienden a producir resultados más estables, mientras las que no tienden a hacer que la red se entrene más rápidamente. Para la estimación de valores se suelen utilizar últimas neuronas la función linear, ya que no limitan el rango de valores que puede producir la neurona.

3.1.1 Perceptrón

Con una neurona ya podemos construir un modelo de machine learning capaz de aprender ciertas funciones. Utilizando una neurona con la función de activación de signo, podemos construir un perceptrón, un clasificador binario lineal [21]. Aunque hay muchas funciones de activación de signo que se utilizan para implementar perceptrones, nosotros utilizaremos la siguiente:

$$\sigma(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases} \tag{3.3}$$

Un perceptrón entrenado es capaz de clasificar adecuadamente cualquier función en $\mathbb{R}^n$ que sea linealmente separable [22], siendo n la cantidad de entradas del perceptron. El perceptrón se puede representar como un hiperplano [23] en $\mathbb{R}^n$, que clasifica como 0 a todos los elementos de un lado y a 1 todos los elementos del otro lado.

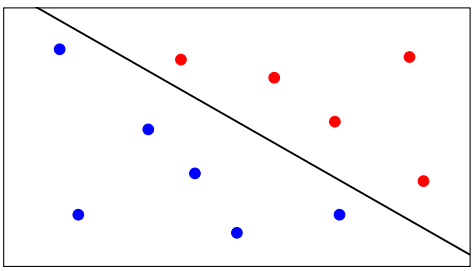

Figura 3.3: Representación gráfica de un perceptrón en $\mathbb{R}^2$. Fuente: elaboración propia

Aunque los perceptrones estén limitados a funciones linealmente separables, se pueden entrenar de forma muy simple. Para entrenar un perceptrón, tenemos que encontrar los pesos adecuados, de forma que para todos los ejemplos de datos el perceptron produzca la salida correspondiente. Sea m en número de ejemplos, n el número de entradas por ejemplo, x_{ik} la entrada i para el ejemplo k, y_k la salida esperada para el ejemplo k y w_i los pesos del perceptron, tendríamos que satisfacer la siguiente condición para entrenar el perceptrón:

$$\forall k \in [1, m], \quad y_k = \sigma\left(\sum_{i=0}^{n} w_i x_{ik}\right) \tag{3.4}$$

Cabe destacar que, como estamos utilizando la fórmula simplificada (Eq. 3.2), w_0 es el *bias*, y por lo tanto $x_{0k} = 1$ para todas las salidas. Para poder medir el progreso respecto a el resultado que necesitamos, definiremos una métrica que mide el error obtenido para un ejemplo concreto:

$$E_k = y_k - \sigma\left(\sum_{i=0}^{n} w_i x_{ik}\right) \tag{3.5}$$

El error será positivo si la salida esperada y_k es mayor que la salida actual, negativo si es menor, o 0 si la salida producida es correcta. Utilizando esta métrica, podemos actualizar los pesos para mitigar el error [24]. Si el error es positivo, significa que tenemos que hacer que el valor de la suma ponderada obtenido con las entradas $x_{0k} \dots x_{nk}$ sea mayor. Para las entradas positivas, tendremos que incrementar w_i, y para las negativas decrementarlo. Esto se puede realizar multiplicando cada entrada por una pequeña constante η, y añadiéndole esto a w_i.

$$E_k > 0 \implies \forall i \in [0, n],\ w_i \leftarrow w_i + x_{ik}\,\eta \tag{3.6}$$

Cuando el error es negativo, podemos realizar lo mismo pero con el signo invertido, ya que el objetivo entonces es disminuir la media ponderada, no aumentarla.

$$E_k < 0 \implies \forall i \in [0, n],\ w_i \leftarrow w_i - x_{ik}\,\eta \tag{3.7}$$

Si el error es cero, dejamos los pesos como están, ya que producen la salida adecuada.

$$E_k = 0 \implies \forall i \in [0, n],\ w_i \leftarrow w_i \tag{3.8}$$

Como sumamos $x_{ik}\,\eta$ cuando $E_k > 0$, restamos $x_{ik}\,\eta$ cuando $E_k < 0$ y no sumamos ni restamos nada cuando $E_k = 0$, podemos unificar las ecuaciones 3.6, 3.7 y 3.8 en una sola ecuación:

$$\forall i \in [0, n],\ w_i \leftarrow w_i + E_k\, x_{ik}\,\eta \tag{3.9}$$

De esta forma, se actualizarán los pesos corrigiendo un poco el error. Si repetimos esta ecuación para todos los ejemplos disponibles, eventualmente el perceptrón producirá el resultado correcto para todos, siempre que los ejemplos sean linealmente separables. A partir de esta ecuación, podemos elaborar un algoritmo de entrenamiento de perceptrones, que conseguirá entrenar correctamente un perceptron, siempre que el conjunto de datos que estamos aprendiendo sea linealmente separable.

Entradas: x (matriz $m \times n$), y (vector de m)
$E = 1$
while $E > 0$ **do**
  $E \leftarrow 0$
  for k **de** 1 **a** m **do**
    $y' \leftarrow \sigma(\sum_{i=0}^{n} w_i x_{ik})$
    $E_k \leftarrow y_k - y'$
    for k **de** 0 **a** n **do**
      $w_i \leftarrow w_i + E_k \, x_{ik} \, \eta$
    end for
  end for
end while

Figura 3.4: Algoritmo de aprendizaje para un perceptrón. Fuente: elaboración propia

El valor η que hemos utilizando se conoce como la *learning rate* del aprendizaje. Un valor más alto hace que se corrija el error más rápidamente, pero un valor demasiado alto puede llevar el aprendizaje a estancarse en ciertos casos. Para perceptrones, un valor recomendable es $\eta = 0.01$, aunque dependiendo de los datos concretos será necesario ajustar el valor.

3.2 Redes neuronales artificiales

Una red neuronal es simplemente un conjunto de neuronas conectadas entre sí. Las primeras neuronas de la red, que se conocen como las neuronas de entrada, directamente reciben sus valores en vez de calcularlos. Estas neuronas actuarán como fuente de datos para el resto de la red, y a través de ellas es como la red neuronal recibe datos. Las últimas neuronas, las neuronas de salida, calculan los valores que corresponden con las salidas producidas por la red. Las neuronas que no son neuronas de entrada o de salida se llaman las neuronas ocultas.

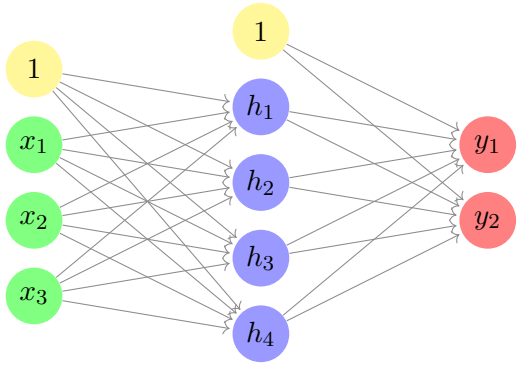

Figura 3.5: Red con 3 neuronas de entrada, 4 ocultas y 2 de salida. Fuente: elaboración propia

Las neuronas de una red neuronal normalmente se organizan en capas, como se puede ver en Fig. 3.5. La primera capa es la capa de entrada, la última la de salida, y las capas intermedias

son las capas ocultas. Cuando una red está estructurada por capas, cada capa suele obtener sus entradas de la capa anterior y pasarlas a la capa siguiente. Los cálculos de una capa son similares a los cálculos de una neurona, pero para cada una de las neuronas de la capa:

$$y_j = \sigma\left(\sum_{i=1}^{n} w_{ij}\, x_i\right) \tag{3.10}$$

Teniendo en cuenta que w es una matriz y x e y son vectores, podemos simplificar la expresión anterior utilizando multiplicación de matrices:

$$y_j = \sigma\left(\begin{bmatrix} w_{0j} & \dots & w_{nj} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}\right) = \sigma\left(w_{\bullet j}^{\mathsf{T}}\, x\right) \tag{3.11}$$

También podemos calcular todas las salidas con la misma operación:

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{00} & \dots & w_{n0} \\ \vdots & \ddots & \vdots \\ w_{0m} & \dots & w_{nm} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}\right) = \sigma\left(w^{\mathsf{T}} x\right) \tag{3.12}$$

Como se puede ver en las ecuaciones 3.11 y 3.12, la matriz de pesos w está transpuesta para que los cálculos sean equivalentes a los de la ecuación 3.10. El vector de entradas en ambas ecuaciones también tiene $x_0 = 1$, para seguir la simplificación del *bias* establecida en la ecuación 3.2. La ecuación 3.12 también se puede adaptar para recibir varias entradas a la vez, lo que nos dejaría con la siguiente ecuación:

$$y = \begin{bmatrix} y_{11} & \dots & y_{1p} \\ \vdots & \ddots & \vdots \\ y_{m1} & \dots & y_{mp} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{00} & \dots & w_{n0} \\ \vdots & \ddots & \vdots \\ w_{0m} & \dots & w_{nm} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ x_{11} & \dots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{np} \end{bmatrix}\right) = \sigma\left(w^{\mathsf{T}} x\right) \tag{3.13}$$

Una vez tenemos la fórmula exacta para calcular la salida de una capa a partir de sus entradas, podemos aplicar la misma fórmula para calcular el resultado de todas las capas. Como se puede ver en Figura 3.6, las redes pueden tener muchas capas y cada capa puede tener cualquier número de neuronas.

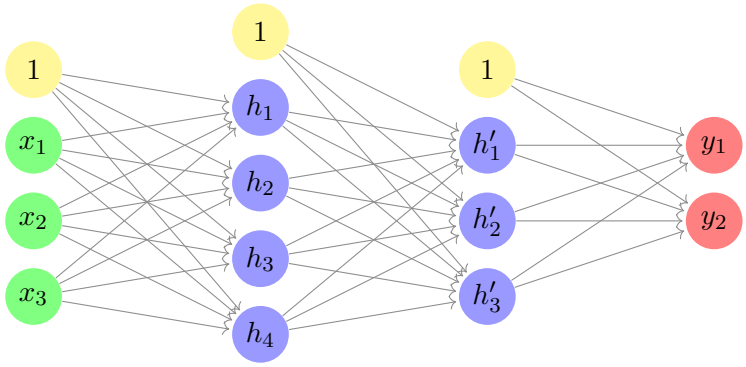

Figura 3.6: Red neuronal con 1 capa de entrada, 2 ocultas y 1 de salida. Fuente: elaboración propia

Cada capa tiene sus propios pesos para cada una de sus neuronas, además de su propia función de activación. Dependiendo de la tarea que queramos realizar también existen ciertos tipos de capas especializados.

3.3 Entrenamiento

Para poder entrenar redes neuronales, primero necesitamos calcular el error que ha cometido la red. Esto no lo podemos medir como en los perceptrones (ver ecuación 3.5), ya que para redes neuronales no es tan sencillo extraer el error de cada peso de la red. Por esto definiremos un error para toda la red, E, que será lo lejos que está el resultado de la red de la salida esperada. Sea y la salida de la red para la entrada x e $\hat{y}$ la salida esperada para esas mismas entradas, tendríamos la siguiente expresión, siendo L la función de error (*loss function* en inglés):

$$E = L(y, \hat{y}) \tag{3.14}$$

Normalmente nos interesa que la red aprenda de forma más rápida cuánto más alejada esté del valor ideal. Esto se puede implementar en la función L, utilizando funciones que se comporten de esta forma. Una de las funciones más utilizadas es la función de pérdida cuadrática, que suma el cuadrado de las diferencias de cada uno de los elementos. Se puede implementar de la siguiente forma:

$$L(y, \hat{y}) = (y - \hat{y})^2 \tag{3.15}$$

3.3.1 Backpropagation

Para el entrenamiento de redes neuronales, normalmente se utiliza un algoritmo llamado *backpropagation*. Este algoritmo calcula cuánto afecta al error cada uno de los pesos de cada capa. El algoritmo funciona calculando el error primero para las neuronas de la capa de salida, y "propaga" este error hacia atrás para calcular el error de las subsecuentes capas. Para estudiar las fórmulas de *backpropagation*, primero será necesario establecer los símbolos que utilizaremos en esas fórmulas. Estos símbolos son los siguientes:

- x_c: entrada recibida por la capa c
- w_c: pesos de la capa c
- σ_c: función de activación de la capa c
- σ'_c: primera derivada de la función de activación de la capa c
- z_c: valores internos de la capa c. $z_c = w_c^\mathsf{T} x_c$
- a_c: valores de activación de la capa c. $a_c = \sigma(z_c)$
- δ_c: error de cada neurona de la capa c
- ∇_c: gradiente de la capa c, el error para cada uno de los pesos de la capa c
- L: función de pérdida
- y: salida obtenida de la red
- $\hat{y}$: salida esperada
- E: error de la red. $E = L(y, \hat{y})$

Para calcular el error de cada neurona de la capa l, tenemos que ver cuanto hace variar esa neurona al error total. Esto se puede calcular con la derivada parcial del error respecto a los valores internos de esa neurona, de forma que nos queda la siguiente ecuación:

$$\delta_c = \frac{\partial E}{\partial z_c} \tag{3.16}$$

A partir de este valor se puede calcular fácilmente el gradiente de la capa c respecto al error de la red, como se explica detalladamente en [25]. Tras realizar los cálculos y demostraciones necesarias, nos queda la siguiente fórmula para calcular el gradiente a partir del error propagado:

$$\nabla_c = \frac{\partial E}{\partial w_c} = x_c \, \delta_c \tag{3.17}$$

Lo único que nos queda es hallar la fórmula para calcular δ_c. Empezamos aplicando la regla de la cadena para dividir la expresión en dos derivadas:

$$\delta_c = \frac{\partial E}{\partial z_c} = \frac{\partial E}{\partial a_c} \odot \frac{\partial a_c}{\partial z_c} \tag{3.18}$$

La anterior ecuación utiliza el producto de Hadamard $\odot$, que es la multiplicación elemento a elemento de todos los elementos de dos matrices. Como $a_c = \sigma(z_c)$, podemos simplificar aún más la expresión:

$$\delta_c = \frac{\partial E}{\partial a_c} \odot \frac{\partial \sigma_c(z_c)}{\partial z_c} = \frac{\partial E}{\partial a_c} \odot \sigma'_c(z_c) \tag{3.19}$$

Para las capas de salida, como la activación de la capa (a_c) es parte del la salida de la red (y), podemos expresar $\partial E / \partial a_c$ a partir de la función de pérdida [26]. Sea $\hat{y}_c$ el vector de salidas

esperadas para esta capa de salida (el subvector de $\hat{y}$ para a_c en vez de para todo y), tenemos la siguiente fórmula, siendo L' la derivada de L respecto a y:

$$\frac{\partial E}{\partial a_c} = \frac{\partial L(a_c, \hat{y}_c)}{\partial a_c} = L'(a_c, \hat{y}_c) \tag{3.20}$$

Por ejemplo, para $L(y, \hat{y}) = (y - \hat{y})^2$, tendríamos la siguiente derivada:

$$L'(y, \hat{y}) = 2(y - \hat{y}) \tag{3.21}$$

Y por lo tanto tendríamos la siguiente fórmula para el error de las capas de salida:

$$\delta_c = L'(a_c, \hat{y}_c) \odot \sigma_c'(z_c) = 2(a_c - \hat{y}_c) \odot \sigma_c'(z_c) \tag{3.22}$$

Para arquitecturas secuenciales, solo tenemos una única capa de salida, por lo que la anterior ecuación se puede simplificar aplicando que $y_c = y$:

$$\delta_c = L'(y, \hat{y}) \odot \sigma_c'(z_c) = 2(y - \hat{y}) \odot \sigma_c'(z_c) \tag{3.23}$$

Para el resto de capas, tenemos que formar el error a partir del error de las capas siguientes. Es necesario realizar un poco de desarrollo matemático adicional, como se puede ver en [25]. Sea c' la capa (o capas) que son inmediatamente siguientes a la capa c, entonces obtenemos la siguiente expresión:

$$\frac{\partial E}{\partial a_c} = w_{c'}^{\mathsf{T}} \delta_{c'} \tag{3.24}$$

Por lo tanto, obtenemos la siguiente expresión para calcular δ_c a partir de $\delta_{c'}$:

$$\delta_c = \frac{\partial E}{\partial a_c} \odot \sigma_c'(z_c) = w_{c'}^{\mathsf{T}} \delta_{c'} \odot \sigma_c'(z_c) \tag{3.25}$$

Para las arquitecturas secuenciales, como c' va a ser únicamente la capa $c + 1$, obtenemos la siguiente expresión:

$$\delta_c = w_{c+1}^{\mathsf{T}} \delta_{c+1} \odot \sigma_c'(z_c) \tag{3.26}$$

Juntando ambos casos (ecuaciones 3.22 y 3.25), podemos calcular el error δ_c de una capa c de la red neuronal de la siguiente forma, siendo c' la capa siguiente a c:

$$\delta_c = \begin{cases} L'(a_c, \hat{y}_c) \odot \sigma_c'(z_c) & \text{si } c \text{ es una capa de salida} \\ w_{c'}^{\mathsf{T}} \delta_{c'} \odot \sigma_c'(z_c) & \text{si } c \text{ no es una capa de salida} \end{cases} \tag{3.27}$$

Para arquitecturas secuenciales (ecuaciones 3.23 y 3.26), nos quedaría la siguiente fórmula:

$$\delta_c = \begin{cases} L'(y, \hat{y}) \odot \sigma_c'(z_c) & \text{si } c \text{ es una capa de salida} \\ w_{c+1}^{\mathsf{T}} \delta_{c+1} \odot \sigma_c'(z_c) & \text{si } c \text{ no es una capa de salida} \end{cases} \tag{3.28}$$

3.3.2 Descenso por gradiente

Como se vio en la ecuación 3.17, podemos calcular el gradiente de la capa c a partir de δ_c. Una vez hemos obtenido el gradiente, tenemos que actualizar los pesos de la red, de manera que el error se reduzca. Como el gradiente nos indica la dirección por la que es más rápido incrementar el error, para reducir los pesos tenemos que restar para cada peso su gradiente. Utilizaremos también una *learning rate*, al igual que para entrenar perceptrones (ver apartado 3.1.1).

$$w_c \leftarrow w_c - \eta \nabla_c \tag{3.29}$$

Este tipo de optimización se llama descenso por gradiente, y es una de las formas más simples de actualizar los pesos de una red neuronal una vez se ha obtenido el gradiente de todas las capas. Opcionalmente se le puede añadir inercia al algoritmo, controlada por el parámetro α, que determina lo rápido que cambia la inercia de la optimización:

$$\begin{aligned} \Delta w_c &\leftarrow \alpha \Delta w_c - \eta \nabla_c \\ w_c &\leftarrow w_c + \Delta w_c \end{aligned} \tag{3.30}$$

Existen muchos otros métodos de optimización [27], cada uno con sus complejidades y parámetros. Aunque muchos de los otros métodos pueden ser muy complejos en su funcionamiento, su objetivo es el mismo: reducir el error de la red lo máximo posible dado los gradientes.

A la hora de optimizar una red neuronal, el optimizador elegido se ejecuta una gran cantidad de veces, reduciendo un poco el error de la red en cada iteración. Al igual que con los perceptrones, escoger una *learning rate* adecuada es muy importante. Una *learning rate* demasiado pequeña hará que el algoritmo aprenda muy lentamente, mientras que una *learning rate* demasiado grande puede hacer que el algoritmo no llegue a converger, y que entre en bucle sin mejorar la red.

4 Algoritmos de multiplicación de matrices

Un área en la que la computación cuántica podría beneficiar substancialmente a la inteligencia artificial es la implementación de algoritmos más eficientes. Existen muchos algoritmos cuánticos que, utilizando diversas técnicas, realizan el mismo cálculo que un algoritmo clásico pero con menor complejidad computacional.

La multiplicación de matrices es uno de los algoritmos que más efecto tendría mejorarlo, ya que para matrices bastante grandes su cálculo demanda muchos recursos. Además, como ya se ha visto en los apartados 3.2 y 3.3.1, es uno de los cálculos más utilizados a la hora de entrenar y ejecutar redes neuronales, el principal tipo de modelo de inteligencia artificial actual. La implementación de algoritmos más eficientes de multiplicación de matrices podría reducir substancialmente el coste computacional requerido por los modelos de inteligencia artificial.

Empezaremos viendo los algoritmos actuales para la multiplicación de matrices, y luego veremos los algoritmos cuánticos. Por último, también realizaremos un análisis comparativo de la posible ahorro computacional que se podría realizar en redes modernas.

4.1 Algoritmos clásicos

Existen varios algoritmos de computación clásica para calcular la multiplicación de dos matrices. El algoritmo más sencillo es el algoritmo iterativo, que se basa en calcular uno a uno los resultados de la multiplicación:

Entradas: A (matriz $n \times m$) y B (matriz $m \times p$)
$C \leftarrow$ matriz $n \times p$ con todos los valores a 0
for i de 1 a n **do**
    for j de 1 a p **do**
        for k de 1 a m **do**
            $C_{ij} \leftarrow C_{ij} + A_{ik}B_{kj}$
        end for
    end for
end for

Figura 4.1: Algoritmo iterativo de multiplicación de matrices. Fuente: elaboración propia

Este algoritmo tiene complejidad $\Theta(nmp)$. Si lo considerásemos con matrices cuadradas ($n = m = p$), entonces nos quedaría la complejidad $\Theta(n^3)$. Aunque este algoritmo no es muy eficiente, lo utilizaremos de base a la hora de comparar el resto de algoritmos.

4.1.1 Algoritmo de Strassen

El algoritmo de Strassen, publicado por Volker Strassen en 1969 [28], es el primer algoritmo de multiplicación de matrices subcúbico. Con su publicación, Strassen demostró que la multiplicación de matrices se podía reducir más que $\Theta(n^3)$, lo que resultó en más investigación y en el descubrimiento de algoritmos de multiplicación de matrices con complejidades aún más reducidas.

Para realizar la multiplicación $C = AB$, el algoritmo primero divide las matrices A, B y C en 4 matrices de igual tamaño:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \tag{4.1}$$

Para realizar la división, es necesario que ambas dimensiones de la matriz sean pares. En caso de que una de las dimensiones sea impar, se le añadirá una fila o columna de 0s a la matriz para hacer que la dimensión sea par, y que se pueda dividir en 4 matrices del mismo tamaño. Esta dimensión extra luego se descartará a la hora de obtener el resultado.

La división de C en 4 matrices nos permite calcular C a partir de C_{11}, C_{12}, C_{21} y C_{22}, que a su vez las podemos calcular a partir de las subdivisiones de A y B:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned} \tag{4.2}$$

Estas ecuaciones no nos ahorrarían nada, ya que requeriríamos 8 multiplicaciones de matrices de la mitad de tamaño, que equivale a la complejidad de una multiplicación de matriz de tamaño normal ($8 \cdot n/2 \cdot m/2 \cdot p/2 = nmp$). La clave del algoritmo de Strassen en calcular ciertos valores intermedios para ahorrarnos una de las multiplicaciones. Definiremos de la siguiente forma los 7 valores intermedios que utilizaremos. Como se puede ver, cada uno de los valores se calcula utilizando una única multiplicación de matrices:

$$\begin{aligned} M_1 &\coloneqq (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 &\coloneqq (A_{21} + A_{22})B_{11} \\ M_3 &\coloneqq A_{11}(B_{12} + B_{22}) \\ M_4 &\coloneqq A_{22}(B_{21} + B_{11}) \\ M_5 &\coloneqq (A_{11} + A_{12})B_{22} \\ M_6 &\coloneqq (A_{21} + A_{11})(B_{11} + B_{12}) \\ M_7 &\coloneqq (A_{12} + A_{22})(B_{21} + B_{22}) \end{aligned} \tag{4.3}$$

A partir de estos 7 valores intermedios, podemos calcular C_{11}, C_{12}, C_{21} y C_{22} a partir de los 7 valores intermedios M_i, y por lo tanto el algoritmo se ahorra una de las 8 multiplicaciones de matrices.

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned} \tag{4.4}$$

El algoritmo de Strassen aplica este proceso de forma recursiva hasta que las matrices sean suficientemente pequeñas (por ejemplo, hasta que una de las dimensiones llegue a 1). A partir de todo esto, podemos ya definir el algoritmo de Strassen:

Entradas: A (matriz $n \times m$) y B (matriz $m \times p$)
if $n = 1 \vee m = 1 \vee p = 1$ **then**
  $C \leftarrow$ multiplicaciónIterativa(A, B)
else
  A_{11}, A_{12}, A_{21}, $A_{22} \leftarrow$ dividirEn4(A)
  B_{11}, B_{12}, B_{21}, $B_{22} \leftarrow$ dividirEn4(B)
  $M_1 \leftarrow$ strassen$(A_{11} + A_{22}, B_{11} + B_{22})$
  $M_2 \leftarrow$ strassen$(A_{21} + A_{22}, B_{11})$
  $M_3 \leftarrow$ strassen$(A_{11}, B_{12} + B_{22})$
  $M_4 \leftarrow$ strassen$(A_{22}, B_{21} + B_{11})$
  $M_5 \leftarrow$ strassen$(A_{11} + A_{12}, B_{22})$
  $M_6 \leftarrow$ strassen$(A_{21} + A_{11}, B_{11} + B_{12})$
  $M_7 \leftarrow$ strassen$(A_{12} + A_{22}, B_{21} + B_{22})$
  $C_{11} \leftarrow$ recortar$(M_1 + M_4 - M_5 + M_7, n/2, p/2)$
  $C_{12} \leftarrow$ recortar$(M_3 + M_5, n/2, p/2)$
  $C_{21} \leftarrow$ recortar$(M_2 + M_4, n/2, p/2)$
  $C_{22} \leftarrow$ recortar$(M_1 - M_2 + M_3 + M_6, n/2, p/2)$
  $C \leftarrow$ juntar4$(C_{11}, C_{12}, C_{21}, C_{22})$
end if

Figura 4.2: Algoritmo de Strassen. Fuente: elaboración propia

Las funciones auxiliares utilizadas en el algoritmo anterior son las siguientes:

- strassen(A, B): la función principal del algoritmo. Se una para realizar llamadas recursivas.
- multiplicaciónIterativa(A, B): algoritmo iterativo de multiplicación de matrices.
- dividirEn4(A): divide una matriz en 4 submatrices de igual tamaño. Si alguna de las dimensiones son impares, la expande con una fila/columna de ceros.
- recortar(A, n, m): devuelve una matriz con los mismos elementos pero con tamaño $n \times m$, descartando los elementos que se queden fuera de la matriz original.

- juntar4(A, B): junta 4 submatrices de igual tamaño para construir una matriz a partir de ellas.

En cada iteración, para calcular la multiplicación de matrices de tamaño $n \times n$ se realizan 7 multiplicaciones de tamaño $n/2 \times n/2$, y muchas sumas y restas. Si $f(n)$ es la complejidad del algoritmo para multiplicar matrices $n \times n$, nos queda la siguiente relación de recurrencia:

$$f(n) = 7 \cdot f\left(\frac{n}{2}\right) + O\left(n^2\right) \tag{4.5}$$

El término $O(n^2)$ es la complejidad de las sumas y restas, que como son de matrices $n/2 \times n/2$, serían $O((n/2)^2) = O(n^2)$. Resolviendo esta relación de recurrencia, nos queda la siguiente complejidad para el algoritmo de Strassen:

$$f(n) = O(n^{log_2 7}) \approx O(n^{2.807}) \tag{4.6}$$

Como podemos ver, la complejidad es menor que la del algoritmo iterativo. No obstante, este algoritmo requiere muchas más sumas y restas de matrices, cosa que introduce una constante computacional bastante grande y hace que este algoritmo solo sea mejor que el algoritmo iterativo para matrices grandes. En complejidad espacial, una implementación básica del algoritmo de Strassen requeriría $O(n^2)$ de espacio, mucho más que el $O(1)$ del algoritmo iterativo. No obstante, utilizando ciertas optimizaciones y reciclando el espacio en las iteraciones, se puede implementar con $O(1)$ de espacio adicional [29].

4.1.2 Algoritmo de Coppersmith-Winograd

Otro de los algoritmos más importantes de multiplicación de matrices es el algoritmo publicado por Don Coppersmith y Shmuel Winograd en 1990 [30]. Este algoritmo, nombrado Coppersmith-Winograd, fue el mejor algoritmo conocido de multiplicación de matrices hasta 2010. Aunque el algoritmo en sí es demasiado complejo para explicarlo aquí, la idea básica es la misma que para el algoritmo de Strassen: encontrar una manera de multiplicar dos matrices $n \times n$ con menos de n^3 multiplicaciones, y aplicarlo recursivamente a matrices más y más pequeñas. La versión original del algoritmo obtiene una complejidad de $O(n^{2.376})$.

El mejor algoritmo actual tiene una complejidad ligeramente menor, de $O(2^{2.372})$ [31]. Este algoritmo está basado en la misma técnica que el algoritmo de Coppersmith-Winograd y se publicó en 2023. Como sus complejidades son tan similares, utilizaremos el algoritmo de Coppersmith-Winograd en el análisis comparativo (ver apartado 4.3)

4.2 Algoritmos cuánticos

Aprovechando las propiedades de los ordenadores cuánticos, es posible construir algoritmos de multiplicación de matrices con complejidades más reducidas que los algoritmos clásicos. En este trabajo estudiaremos tres de los principales y realizaremos una análisis de complejidad entre ellos y los algoritmos actuales de multiplicación de matrices.

4.2.1 Por test de intercambio (swap test)

Este algoritmo cuántico utiliza una rutina cuántica conocida como *Swap test* (test de intercambio). Esta rutina permite estimar el producto interior de dos vectores de cúbits, que se puede luego utilizar para multiplicar dos matrices. Para realizar la multiplicación de matrices $C = AB$, sea $A_{i\bullet}$ la fila i de la matriz A, $B_{\bullet j}$ la columna j de la matriz B, xy el producto interior y $||v||$ el módulo del vector v, podemos calcular la multiplicación de matrices con la siguiente fórmula:

$$C_{ij} = ||A_{i\bullet}||\,||B_{\bullet j}||\,A_{i\bullet}B_{\bullet j} \tag{4.7}$$

Las magnitudes de los vectores se pueden calcular de manera sencilla en $O(n^2)$. El único paso de esta fórmula que no podemos calcular con complejidad cuadrática son los n^2 productos interiores, ya que si los calculamos directamente cada uno llevaría $O(n)$, por lo que tendríamos complejidad $O(n^3)$. Para reducir esta complejidad emplearemos un circuito cuántico conocido como el *Swap test*, que permite, tras ejecutar la rutina múltiples veces, cada vez producir una mejor aproximación del producto interior de dos vectores. Aquí solo se explicará la manera de construir el circuito para 1 cúbit, ya que cualquier otro caso es demasiado complejo de realizar a mano. No obstante, se puede construir una versión que funciona para vectores de cualquier número de cúbits con solo ligeramente más complejidad que en el caso de 1 cúbit [32]. El algoritmo de *Swap test* se basa en el siguiente circuito cuántico:

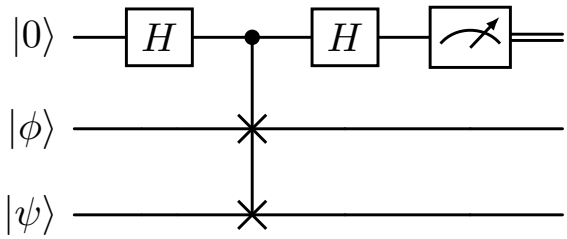

Figura 4.3: Circuito para una iteración de *Swap test* de 1 cúbit. Fuente: elaboración propia

Este circuito de 3 cúbits, primero aplica una puerta Hadamard (ver apartado 2.3.2) al primer cúbit, luego aplica una puerta CSWAP (ver apartado 2.3.5.4), utilizando el primer cúbit como control. Luego vuelve a aplicar una puerta Hadamard al primer cúbit. Por último, medimos el primer cúbit, para ver si produce 0 o 1.

El circuito cuántico empieza con el estado $|0, \phi, \psi\rangle$, siendo los estados ϕ y ψ variables desconocidas. Después de la primera puerta H, el circuito pasa al siguiente estado:

$$\begin{aligned}(H \otimes I \otimes I)\,|0, \phi, \psi\rangle &= H\,|0\rangle \otimes |\phi, \psi\rangle \\ &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |\phi, \psi\rangle \\ &= \frac{|0, \phi, \psi\rangle + |1, \phi, \psi\rangle}{\sqrt{2}}\end{aligned} \tag{4.8}$$

Tras la puerta $SWAP$ controlada, que invierte $|\phi\rangle$ y $|\psi\rangle$ cuando el primer cúbit vale 1, pasamos a este estado:

$$CSWAP\,\frac{|0,\phi,\psi\rangle + |1,\phi,\psi\rangle}{\sqrt{2}} = \frac{|0,\phi,\psi\rangle + |1,\psi,\phi\rangle}{\sqrt{2}} \tag{4.9}$$

Por último, aplicamos la segunda puerta Hadamard al primer cúbit, que nos deja en el siguiente estado:

$$(H \otimes I \otimes I)\,\frac{|0,\phi,\psi\rangle + |1,\phi,\psi\rangle}{\sqrt{2}} = \frac{|0,\phi,\psi\rangle + |1,\phi,\psi\rangle + |0,\psi,\phi\rangle - |1,\psi,\phi\rangle}{2} \tag{4.10}$$

Tras todo esto, lo único que nos queda es medir el primer cúbit, para lo que calcularemos la probabilidad de que el sistema se encuentre en cada uno de los casos. Si realizamos el cálculo, nos quedan los siguientes amplitudes para cada uno de los 8 estados:

$\|x\rangle$	α_x	$P(\|x\rangle) = \|\alpha_x\|^2$
$\|000\rangle$	$\phi_0\psi_0$	$\|\phi_0\psi_0\|^2$
$\|001\rangle$	$(\phi_0\psi_1 + \phi_1\psi_0)/2$	$\|\phi_0\psi_1 + \phi_1\psi_0\|^2/4$
$\|010\rangle$	$(\phi_0\psi_1 + \phi_1\psi_0)/2$	$\|\phi_0\psi_1 + \phi_1\psi_0\|^2/4$
$\|011\rangle$	$\phi_1\psi_1$	$\|\phi_1\psi_1\|^2$
$\|100\rangle$	0	0
$\|101\rangle$	$(\phi_0\psi_1 - \phi_1\psi_0)/2$	$\|\phi_0\psi_1 - \phi_1\psi_0\|^2/4$
$\|110\rangle$	$(\phi_1\psi_0 - \phi_0\psi_1)/2$	$\|\phi_1\psi_0 - \phi_0\psi_1\|^2/4$
$\|111\rangle$	0	0

Tabla 4.1: Resultado de un *Swap test* de 1 cúbit respecto a $|\phi,\psi\rangle$. Fuente: elaboración propia

La tabla contiene las amplitudes de cada estado α_x, junto con la probabilidad de cada uno de los estados, que es α_x^2. Si ahora sumamos las probabilidades de los estados cuándo el primer cúbit vale 0 (los estados $|000\rangle$, $|001\rangle$, $|010\rangle$ y $|011\rangle$) y cuando vale 1 (los estados $|100\rangle$, $|101\rangle$, $|110\rangle$ y $|111\rangle$) y simplificamos, podemos obtener las probabilidades de que el primer cúbit valga 0 o 1 tras ejecutar el algoritmo:

$$\begin{aligned} P\,(\text{primer cúbit} = 0) &= |\phi_0\psi_0|^2 + |\phi_1\psi_1|^2 + \frac{|\phi_0\psi_1 + \phi_1\psi_0|^2}{2} \\ P\,(\text{primer cúbit} = 1) &= \frac{|\phi_0\psi_1 - \phi_1\psi_0|^2}{2} \end{aligned} \tag{4.11}$$

Estos resultados son muy interesantes, ya que todos los componentes paralelos ($|\phi_0\psi_0|^2$ y $|\phi_1\psi_1|^2$) pertenecen al estado $|0\rangle$, pero los componentes perpendiculares ($|\phi_0\psi_1 + \phi_1\psi_0|^2$ y $|\phi_0\psi_1 - \phi_1\psi_0|^2$) pertenecen a ambos estados. Cuando los estados $|\phi\rangle$ y $|\psi\rangle$ son ortogonales, hay un 50% de que el primer cúbit sea 0 y otro 50% de que sea 1. Si $|\phi\rangle$ y $|\psi\rangle$ son paralelos, entonces el primer cúbit siempre será 0. De forma general, cuanto más paralelos sean los vectores de cúbits $|\phi\rangle$ y $|\psi\rangle$, mayor será la probabilidad de que el primer cúbit sea 0.

Si ejecutamos el algoritmo k veces, utilizando k copias de $|\phi\rangle$ y $|\psi\rangle$ ya preparadas, entonces podemos estimar el producto interior de $|\phi\rangle$ y $|\psi\rangle$. Siendo m_i el resultado de medir el primer cúbit cada una de esas veces, tenemos la siguiente fórmula para aproximar el resultado del producto interior de $|\phi\rangle$ y $|\psi\rangle$.

$$1 - \frac{1}{k}\sum_{i=1}^{k} m_i \tag{4.12}$$

Con esta fórmula podemos obtener con una precisión arbitrariamente alta el resultado del producto interior, ya que para mayor precisión simplemente hay que ejecutar el algoritmo más veces. Como la precisión es independiente del tamaño de las matrices a multiplicar, no aumenta la complejidad, por lo que este algoritmo es $O(1)$, sin tener en cuenta la preparación de los cúbits.

Este es el caso de 1 cúbit. Para vectores de varios cúbits, podemos utilizar el mismo algoritmo, modificado para que la puerta $SWAP$ controlada se realice para cada una de las parejas $|\phi\rangle_i$ y $|\psi\rangle_i$ de los vectores de cúbits $|\phi\rangle$ y $|\psi\rangle$:

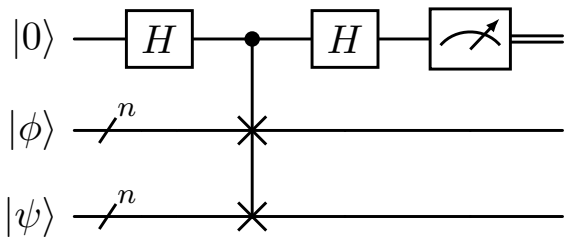

Figura 4.4: Circuito para una iteración de *Swap test* de n cúbits. Fuente: elaboración propia

La única desventaja con este algoritmo, es que construir y ejecutar la puerta Swap controlada para n cúbits no es constante. Utilizando ciertos trucos a la hora de ejecutar todas las puertas necesarias para todos los cúbits de la matriz, podemos reducir esta complejidad a $O(log^c\, poly\, m)$, donde $1 < c \leq 2$ y $polym$ es un polinomio de la cantidad de cúbits a preparar [33]. Es aún un problema abierto saber el valor exacto de c y el menor polinomio posible de m, por lo que nosotros ignoraremos ese factor en el análisis. Como ese factor es logarítmico, tampoco perdemos mucha precisión en el análisis si lo ignoramos.

Ahora, lo único que nos falta es construir todo el algoritmo para multiplicar matrices, utilizando las rutinas que hemos visto. Utilizaremos las siguientes funciones auxiliares a la hora de describir el algoritmo:

- prepararEstados(A, B): prepara los estados cuánticos necesarios para ejecutar un swap test para cada valor de las matrices A y B.

- swapTest($|x\rangle$): ejecuta un swap test para cada uno de los valores de las matrices A y B y devuelve el valor en una matriz de resultados que corresponde al resultado de cada swap test (0 o 1). Recibe como entrada los estados cuánticos ya preparados para realizar los swap tests de forma eficiente.

- calcularProductosMagnitudes(A, B): calcula $||A_{i\bullet}||\,||B_{\bullet j}||$ para todos los $i, j \in [1, n]$.

A partir de estas funciones, el algoritmo completo que calcula la multiplicación matricial $C = AB$ utilizando swap tests es el siguiente:

Entradas: A (matriz $n \times n$), B (matriz $n \times n$) y k (iteraciones de cada swap test)
$M \leftarrow$ calcularProductosMagnitudes(A, B)
$R \leftarrow$ matriz $n \times n$ con todos los valores a 0
for i de 1 a k **do**
  $|\psi\rangle \leftarrow$ prepararEstados(A, B)
  $R \leftarrow R +$ swapTest$(|\psi\rangle)$
end for
$C \leftarrow$ matriz $n \times n$
for i de 1 a n **do**
  for j de 1 a n **do**
    $C_{ij} \leftarrow M_{ij}\,(1 - \frac{1}{k} R_{ij})$
  end for
end for

Figura 4.5: Algoritmo de multiplicación de matrices basado en *Swap test*. Fuente: elaboración propia

Como la función swapTest() realiza un swap test para cada elemento de las matrices A y B, y cada test tiene complejidad $O(log^c\, poly\, n^2)$, la complejidad de la función es de $O(n^2\, log^c\, poly\, n^2)$. Ignorando el factor polinómico para facilitar el análisis, nos queda $O(n^2)$. La complejidades de prepararEstados() y calcularProductosMagnitudes() son $O(n^2)$, por lo que la complejidad total sigue siendo de $O(n^2)$.

Teniendo todo esto en cuenta, la complejidad final del algoritmo es de $O(n^2\, k)$, siendo k el número de iteraciones. Si cogemos una k que sea constante (un número de iteraciones que no dependa del tamaño de las matrices), entonces tenemos la complejidad $O(n^2)$. Esta complejidad es mucho menor que la que encontramos incluso en el mejor de los algoritmos clásicos ($O(n^{2.372})$), por lo que podría mejorar mucho la multiplicación de matrices si los ordenadores cuánticos se vuelven asequibles.

4.2.2 Otros algoritmos

Existen muchos otros algoritmos cuánticos para multiplicación de matrices, como algoritmos basados en el algoritmo HHL (un algoritmo cuántico de resolución de sistemas de ecuaciones [34]) y otros basados en SVE (un algoritmo de estimación de descomposición en valores singulares [35]). Estos algoritmos no tienen complejidades que sean fáciles de comparar con los algoritmos clásicos, ya que su complejidad depende de la matriz de entrada.

Más concretamente, según el análisis de complejidades realizado en [32], obtenemos la complejidad $\tilde{O}\left(\kappa(A)^2 n^{1.75} + n^2\right)$ para el algoritmo basado en HHL y $\tilde{O}\left(\kappa(A)\, n^{2.25}\right)$ para el algoritmo basado en SVE, siendo $\kappa(A)$ el número de condición de la matriz A al realizar el cálculo $C = AB$. El número de condición de una matriz X se define de la siguiente forma:

$$\kappa(X) = ||X|| \cdot ||X^{-1}|| \tag{4.13}$$

Como se puede ver en la ecuación 4.13, $\kappa(A)$ depende de A, por lo que las complejidades de los algoritmos basados en HHL y SVE dependen de los valores de la matriz de entrada. Es por esto que estos algoritmos no se incluirán en el análisis comparativo de algoritmos, y solo se analizará el algoritmo cuántico basado en *swap tests*.

4.3 Análisis teórico de las complejidades

En este capítulo hemos estudiado varios algoritmos clásicos y cuánticos de multiplicación de matrices. En la tabla 4.2 se puede ver un pequeño resumen de las complejidades de los algoritmos estudiados.

Algoritmo	Complejidad
Iterativo	$O(n^3)$
Strassen	$O(n^{log_2 7}) \approx O(n^{2.807})$
Coppersmith-Winograd	$O(n^{2.376})$
Basado en *Swap test*	$O(n^2)$

Tabla 4.2: Complejidades de los algoritmos estudiados. Fuente: elaboración propia

Para este análisis, utilizaremos el tamaño de las matrices de atención de ciertas redes neuronales modernas. Este análisis será una estimación basada en la complejidad de los algoritmos, e ignora las constantes temporales de los algoritmos y muchas de las optimizaciones que se realizan en la multiplicación de matrices en las redes neuronales modernas. No obstante, este análisis nos puede dar una aproximación del *speedup* que podríamos obtener si estos algoritmos se implementasen en el entrenamiento de los modelos de inteligencia artificial actuales.

Para determinar el tamaño de las matrices con el que calcularemos la aproximación, utilizaremos las matrices de embedding, ya que todos los modelos que compararemos son *transformers* [36] y estas matrices suelen ser las matrices más grandes de este tipo de modelo. Aproximaremos el resultado del embedding como una única matriz cuadrada del mismo tamaño. Analizaremos los siguientes modelos:

Modelo	Matriz de *embedding*	Matriz cuadrada equivalente
GPT-2	1024×1600	1280×1280
GPT-3	2048×12288	5016.6×5016.6
GPT-3.5	4096×12288	7094.5×7094.5
GPT-4	32768×3072	10034×10034
GPT-4 Turbo	128000×3072	19830×19830

Tabla 4.3: Tamaño de *embedding* de los modelos analizados. Fuente: elaboración propia

Aproximaremos el coste de ejecución de cada modelo como el coste de multiplicar dos matrices

del mismo número de elementos que la matriz de *embedding*. Utilizaremos las dimensiones de una matriz cuadrada que tenga el mismo número de elementos para estimar la n de cada modelo. A partir de la n del modelo, estimaremos su coste para cada algoritmo utilizando la complejidad, donde sustituiremos la n de la complejidad por la n del modelo. Obtenemos los siguientes resultados:

Modelo	Iterativo	Strassen	Coppersmith-Winograd	Swap test
GPT-2	2.097×10^9	5.272×10^8	2.414×10^7	1.638×10^6
GPT-3	1.263×10^{11}	2.437×10^{10}	6.197×10^8	2.516×10^7
GPT-3.5	3.571×10^{11}	6.450×10^{10}	1.412×10^9	5.331×10^7
GPT-4	1.010×10^{12}	1.707×10^{11}	3.217×10^9	1.007×10^8
GPT-4 Turbo	7.798×10^{12}	1.155×10^{12}	1.623×10^{10}	3.932×10^8

Tabla 4.4: Estimación del coste de ejecución de los algoritmos estudiados. Fuente: elaboración propia

Modelo	Strassen	Coppersmith-Winograd	Swap test
GPT-2	25.1%	1.151%	0.0781%
GPT-3	19.3%	0.490%	0.0199%
GPT-3.5	18.1%	0.395%	0.0141%
GPT-4	16.9%	0.318%	0.0099%
GPT-4 Turbo	14.8%	0.208%	0.0050%

Tabla 4.5: Coste de ejecución respecto al algoritmo iterativo. Fuente: elaboración propia

4.3.1 Conclusiones del análisis

Como se puede ver en las tablas 4.4 y 4.5, el algoritmo basado en *swap tests* proporciona una reducción enorme de coste computacional, incluso teniendo en cuenta ya la reducción proporcionada por el algoritmo de Coppersmith-Winograd. Incluso si su implementación requiriese de una gran cantidad de cálculos adicionales, como la diferencia es tan grande con el resto de algoritmos el beneficio podría seguir siendo inmenso.

Además, como el beneficio crece a medida que crece la complejidad del modelo, si los modelos de inteligencia artificial se van volviendo más y más complejos, el incentivo para desarrollar este tipo de procesadores crecerá con el tiempo. También existen muchos otros algoritmos cuánticos, algunos con complejidades menores que el algoritmo de *swap tests* estudiado, por lo que el beneficio real podría llegar a ser mayor que el calculado en este análisis, que ya es bastante optimista.

5 Búsqueda de Grover

La búsqueda de Grover (*Grover's search* en inglés) es un algoritmos de búsqueda que aprovecha las propiedades de la computación cuántica para buscar elementos [37]. El algoritmo utiliza una función de búsqueda $s(x)$ y busca algún elemento x que devuelva $s(x) = 1$. El algoritmos trata la función de búsqueda como una caja negra, lo que significa que no aprovecha ninguna propiedad específica de la función. La búsqueda de Grover realiza la búsqueda de un elemento en $O(\sqrt{N})$, siendo N el número total de elementos. Como se demostró en [38], esto es óptimo, ya que cualquier algoritmo de búsqueda cuántico tendrá como mínimo la complejidad temporal $\Omega(N)$.

Aunque la búsqueda de Grover es un buen candidato para mejorar la eficiencia de muchos algoritmos, no se puede aplicar en todos los casos. Esto se debe, como veremos en el apartado 5.1.2, a que tenemos que construir eficientemente una puerta cuántica que dependen de $s(x)$. No obstante, no podemos simplemente evaluar $s(x)$ para todos los elementos, ya que entonces la complejidad del algoritmo sería $O(N)$. Dependiendo de ciertas propiedades de la función $s(x)$ concreta y de los elementos que estemos buscando, en ciertos casos es posible la construcción eficiente de esa puerta cuántica. En el apartado 5.2 podemos ver algunos algoritmos de inteligencia artificial en los que se ha conseguido integrar la búsqueda de Grover para conseguir una mejora de la complejidad temporal.

5.1 Algoritmo de Grover

El algoritmo de Grover se puede dividir en tres pasos principales: inicialización, búsqueda y inversión sobre la media. El primer paso solo se realiza en la primera iteración, mientras que los dos últimos se realizan una vez para cada iteración.

5.1.1 Inicialización

El primer paso en el algoritmo es inicializar el estado. Al principio, como no tenemos nada de información a cerca del problema a resolver, tendremos que inicializar cada uno de los cúbits a una superposición entre 0 y 1. Específicamente, lo realizaremos de tal manera que cada uno de los posibles estados tenga la exactamente misma amplitud. Esto se puede realizar de manera sencilla, inicializando los cúbits a $|0^n\rangle$ y aplicando la puerta Hadamard a todos los cúbits del sistema.

Para una búsqueda en un espacio de $N = 2^n$ elementos, necesitaremos inicializar n cúbits, y por lo tanto tendremos que aplicarles una puerta Hadamard paralela de n cúbits. Si N no fuese una potencia de dos, entonces necesitaríamos $n = \lceil log_2 N \rceil$ cúbits. Tras aplicar la inicialización a los cúbits, nos queda el siguiente estado cuántico:

$$|\phi\rangle = H^{\otimes n} |0^n\rangle = \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} |x\rangle \tag{5.1}$$

Como se puede ver, las amplitudes de todos los estados son $1/\sqrt{N}$, por lo que todos los estados tienen la misma probabilidad ($(1/\sqrt{N})^2 = 1/N$). Este estado es importante para construir ciertas de las puertas que utilizaremos en el algoritmo, por lo que lo llamaremos $|\phi\rangle$. Para entender mejor el algoritmo, también incluiremos en cada paso una visualización de como el algoritmo cambia las amplitudes de todos los estados de un sistema de 3 cúbits (ver figura 5.1). Como durante la ejecución de este algoritmo todas las amplitudes serán números reales, las visualizaremos utilizando un gráfico de barras.

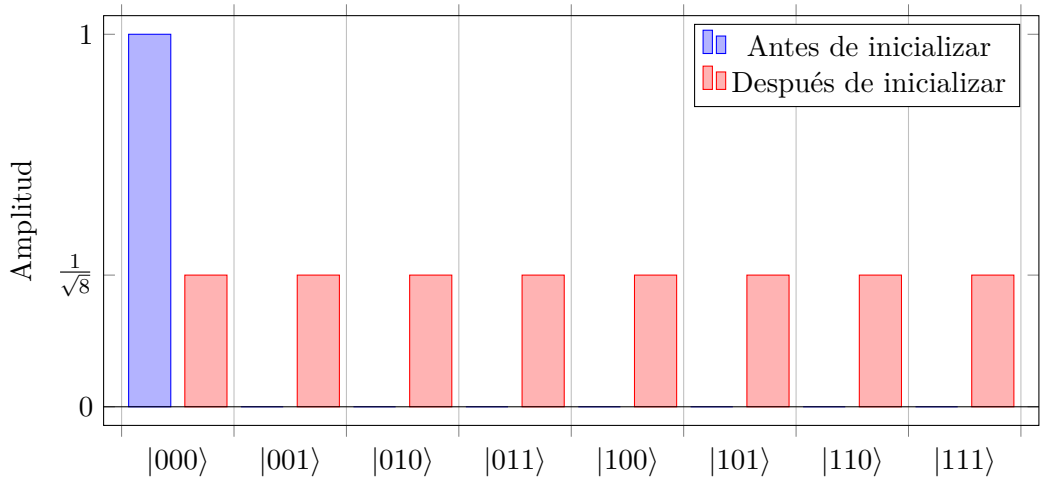

Figura 5.1: Inicialización de una búsqueda de Grover para 8 elementos. Fuente: elaboración propia

5.1.2 Búsqueda

Después de inicializar el estado, el siguiente paso es la búsqueda. Este paso invierte las amplitudes de todos los estados que estamos buscando. Dicho de otro modo, invierte las amplitudes de todos los estados para los que $s(x) = 1$. Construiremos una puerta lógica para realizar esta operación. Dada la función $s(x)$, podemos construirla a partir de la siguiente ecuación:

$$U_s |x\rangle = (-1)^{s(x)} |x\rangle \tag{5.2}$$

Entonces, para construir U_s simplemente tenemos que crear una matriz diagonal con los valores $(-1)^{s(x)}$ para cada estado x:

$$\begin{aligned} U_s &= diag\left((-1)^{s(|0\ldots0\rangle)}, (-1)^{s(|0\ldots1\rangle)}, \ldots, (-1)^{s(|1\ldots1\rangle)}\right) \\ &= \begin{pmatrix} (-1)^{s(|0\ldots0\rangle)} & 0 & \ldots & 0 \\ 0 & (-1)^{s(|0\ldots1\rangle)} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & (-1)^{s(|1\ldots1\rangle)} \end{pmatrix} \end{aligned} \tag{5.3}$$

Por ejemplo, para un sistema de 4 cúbits y una función s que devuelve 1 para $|10\rangle$ y 0 para

los demás estados, tendríamos la siguiente matriz:

$$U_s = diag(1, 1, -1, 1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.4}$$

Una vez construida la puerta U_s, tenemos que aplicársela a los estados iniciales. Visualmente, para una función de búsqueda para la que $s(|010\rangle) = 1$, tendríamos la siguiente representación:

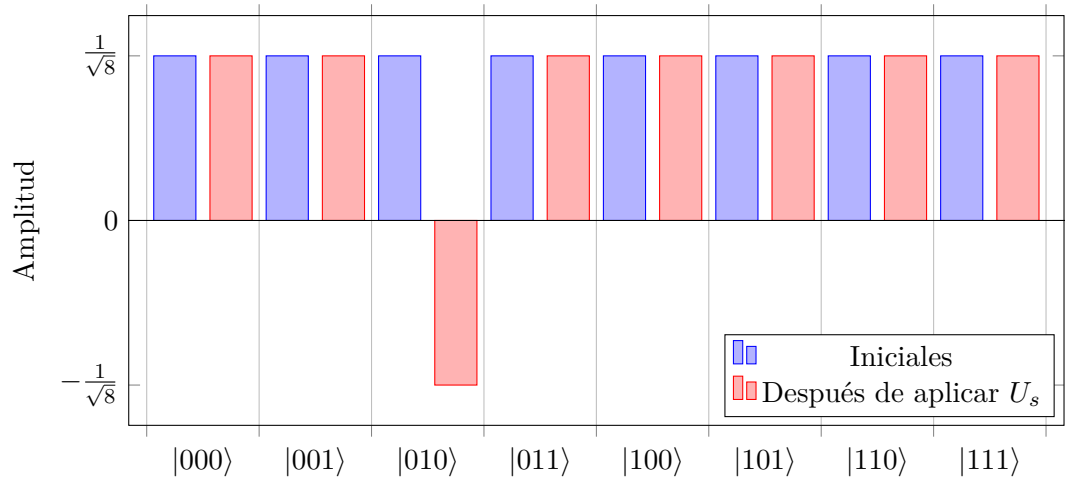

Figura 5.2: Amplitudes tras aplicar U_s en una búsqueda de Grover. Fuente: elaboración propia

Como se puede ver, solo se han invertido las amplitudes de los estados que estamos buscando.

5.1.3 Inversión sobre la media

El último paso del algoritmo de Grover es lo que se conoce como "Inversión sobre la media". Esta puerta, para cada amplitud, la invierte respecto a la amplitud media de todos los estados base. Si $\overline{\alpha}$ es la amplitud media para todos los estados base de $|\psi\rangle$, podemos definir el comportamiento de U_i de la siguiente forma:

$$U_i |\psi\rangle = \sum_{x \in \{0,1\}^n} (2\overline{\alpha} - \alpha_x) |x\rangle \tag{5.5}$$

De esta forma, para cada estado $|x\rangle$ su amplitud pasará de α_x a $2\overline{\alpha} - \alpha_x$, reflejando la amplitud respecto a la media $\overline{\alpha}$. Esta puerta, aprovechando ciertas propiedades de la física cuántica y de los tensores, se puede implementar de forma sencilla y eficiente. Si $|\phi\rangle$ es el estado del sistema tras ser inicializado, se puede implementar con la siguiente expresión:

$$U_i = 2 |\phi\rangle \langle\phi| - I \tag{5.6}$$

Esta expresión, que utiliza una proyección tensorial ($|\phi\rangle \langle\phi|$), implementa el comportamiento que hemos visto sin necesidad de calcular la media de las amplitudes. Como esto solo depende

de $|\phi\rangle$, que en sí solo depende del número de cúbits del sistema, se puede precalcular U_i. Visualmente, tras aplicar U_i, tendríamos las siguientes amplitudes:

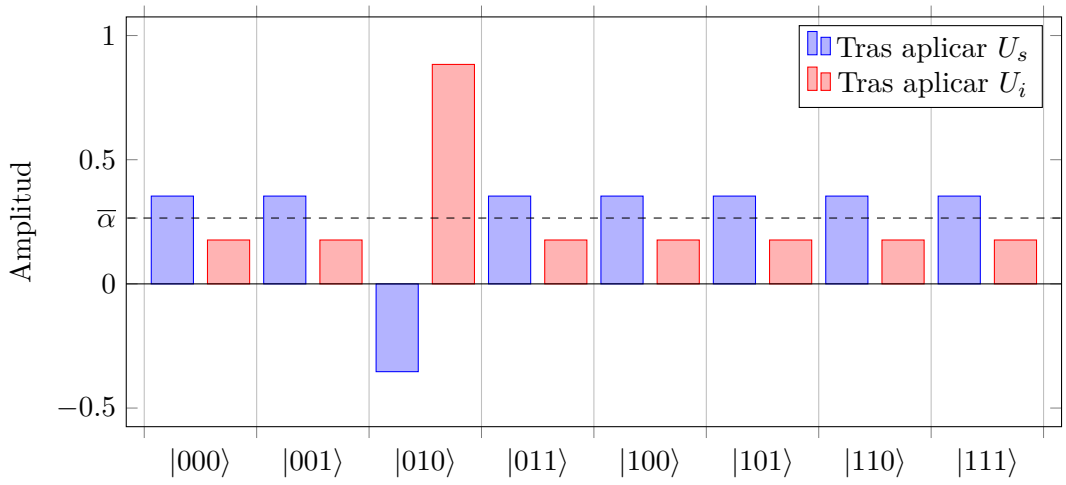

Figura 5.3: Amplitudes tras 1 iteración del algoritmo de Grover. Fuente: elaboración propia

5.1.4 Iteraciones

Todo lo que hemos visto hasta ahora describiría una sola iteración del algoritmo de Grover. Para aplicar el algoritmo al completo, tras inicializar los estados, hay que aplicar U_s y U_i varias veces. Para maximizar la probabilidad de los estados para los que $s(x) = 1$, tenemos que aplicar U_s y U_i k veces, donde k es la siguiente expresión:

$$k = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rfloor \tag{5.7}$$

M es la cantidad de estados para los que $s(x) = 1$, o lo que es equivalente, $M = \sum_{x\in\{0,1\}^n} s(x)$. Si aplicamos el algoritmo más veces que las que son óptimas, entonces $\overline{\alpha}$ se volverá negativo, por lo que al aplicar U_i estaríamos reduciendo la probabilidad de los estados. Podemos expresar el algoritmo de Grover con el siguiente circuito cuántico:

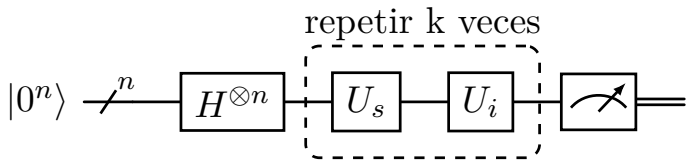

Figura 5.4: Circuito cuántico para el algoritmo de Grover. Fuente: elaboración propia

Visualmente, si ejecutamos el algoritmo iteración a iteración, podemos ver como progresivamente aumenta la probabilidad de los estados para los que $s(x) = 1$.

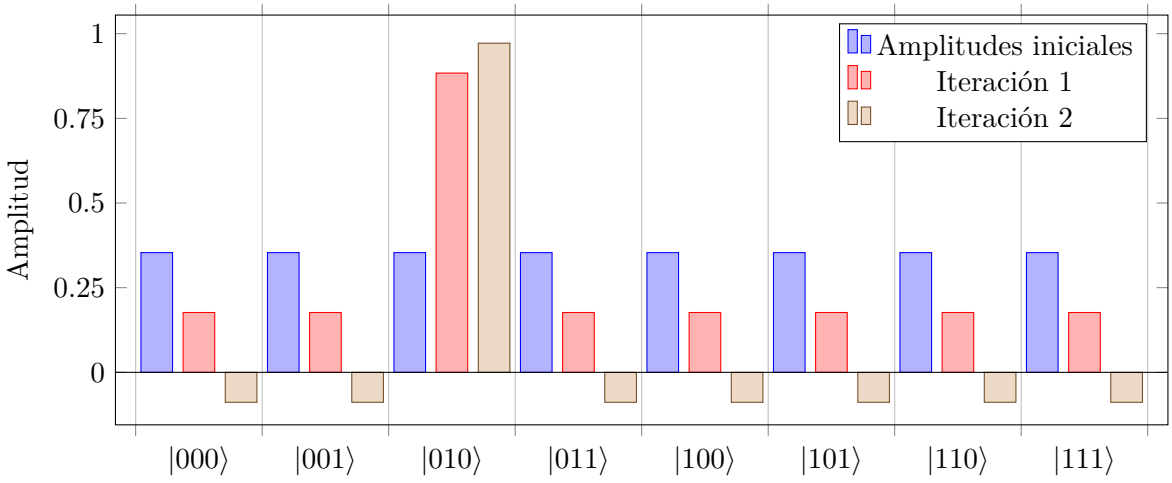

Figura 5.5: Amplitudes de una búsqueda de Grover del estado $|010\rangle$. Fuente: elaboración propia

Como se puede ver, tras cada iteración, el algoritmo de Grover aumenta las amplitudes de los estados que estamos buscando ($|010\rangle$ en este caso). Como tenemos 8 elementos (N = 8) y tenemos 1 elemento válido (M = 1), podemos calcular la cantidad de iteraciones que deberíamos aplicar a partir de la siguiente fórmula:

$$k = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rfloor = \left\lfloor \frac{\pi}{4}\sqrt{\frac{8}{1}} \right\rfloor = 2 \tag{5.8}$$

Como $k = 2$, si realizásemos otra iteración más, disminuiríamos la probabilidad de los estados que estamos buscando, por lo mejor es medir realizar dos iteraciones y luego medir el estado. No obstante, como se puede ver en el gráfico, la probabilidad de que el estado resultante sea el que queremos no es 100%. El algoritmo de Grover solo nos asegura que, tras realizar el número de iteraciones necesarias, tendremos más del 50% de probabilidad de encontrar un elemento para el que $s(x) = 1$.

5.1.5 Algoritmo completo

Para remediar este problema, si no encontramos un elemento válido podemos simplemente volver a ejecutar el algoritmo hasta que encontremos. Como la probabilidad de encontrar un elemento es $> 50\%$, de media lo encontraremos como mucho en 2 intentos, por lo que solo añade una constante y no afecta a la complejidad del algoritmo.

Teniendo en cuenta esto, podemos especificar el algoritmo de Grover al completo:

Entradas: U_s, $s(x)$, N, M
$n \leftarrow \lceil \log_2 N \rceil$
$k \leftarrow \lfloor \pi/4 \sqrt{N/M} \rfloor$
$|\phi\rangle \leftarrow H^{\otimes n} |0^n\rangle$
$U_i \leftarrow 2 |\phi\rangle \langle\phi| - I$
$sol \leftarrow 0$
while $sol = 0$ **do**
    $|\psi\rangle \leftarrow H^{\otimes n} |0^n\rangle$
    for j de 1 a k **do**
        $|\psi\rangle \leftarrow U_i U_s |\psi\rangle$
    end for
    $x \leftarrow \text{medir}(|\psi\rangle)$
    $sol = s(x)$
end while

Figura 5.6: Algoritmo de Grover para M conocida. Fuente: elaboración propia

Como k es proporcional a $\sqrt{N/M}$, la cantidad de iteraciones del algoritmo también es proporcional a la raíz cuadrada de N/M, por lo que tenemos la siguiente complejidad:

$$O(\sqrt{N/M}) \tag{5.9}$$

Si asumimos el peor caso de $M = 1$, podemos simplificar la complejidad:

$$O(\sqrt{N}) \tag{5.10}$$

Este algoritmo solo funciona si conocemos M, la cantidad de elementos para los que $s(x) = 1$. Cuando no sepamos la cantidad de elementos, podemos simplemente ejecutar el algoritmo con $k = N, N/2, N/4 \ldots N/2^t$. Con una probabilidad muy grande ($> 50\%$) el algoritmo encontrará una solución antes de la iteración $t = log_2(N/M)$. Si el algoritmo no ha encontrado una solución para cuando $2^t > N$, simplemente podemos volver a ejecutar toda la secuencia.

Como la probabilidad de encontrar un resultado correcto en una secuencia es de más de 50%, la cantidad de secuencias media no superará 2, por lo que solo supondrá una constante adicional, y no modificará la complejidad de $O(\sqrt{N})$. El algoritmo modificado sería el siguiente:

Entradas: U_s, $s(x)$, N
$n \leftarrow \lceil \log_2 N \rceil$, $sol \leftarrow 0$
$|\phi\rangle \leftarrow H^{\otimes n} |0^n\rangle$, $U_i \leftarrow 2 |\phi\rangle \langle\phi| - I$
while $sol = 0$ **do**
  $M \leftarrow 1$
  while $M < 2^N \wedge sol = 0$ **do**
    $k \leftarrow \lfloor \pi/4\sqrt{N/M} \rfloor$
    $|\psi\rangle \leftarrow H^{\otimes n} |0^n\rangle$
    for i de 1 a k **do**
      $|\psi\rangle \leftarrow U_i U_s |\psi\rangle$
    end for
    $x \leftarrow$ medir($|\psi\rangle$)
    $sol = s(x)$
    $M \leftarrow 2M$
  end while
end while

Figura 5.7: Algoritmo de Grover para M no conocida. Fuente: elaboración propia

5.2 Aplicaciones en la inteligencia artificial

Utilizando la búsqueda de Grover es posible construir algoritmos de machine learning que sean más rápidos que los que sería posible en un ordenador clásico. Para muchas de las aplicaciones de inteligencia artificial, el uso del algoritmo de Grover obtiene un *speedup* de $O(\sqrt{f(n)})$, siendo $O(f(n))$ la complejidad algorítmica original del algoritmo. Algunos de las aplicaciones para las que se han ideado algoritmos que aprovechan la búsqueda de Grover son las siguientes:

- K vecinos más próximos, un algoritmo de aprendizaje supervisado [39]
- *Clustering* mediante k-medianas, un algoritmo de aprendizaje no supervisado [40]
- Cálculo de atención para *transformers* y redes neuronales recurrentes (RNNs) [41]
- *Active learning agents* y otros algoritmos de aprendizaje por refuerzo [42]
- Entrenamiento de perceptrones [43]

Aunque de momento no se ha conseguido aprovechar el potencial de la búsqueda de Grover para todos los tipos de algoritmos de machine learning, en los casos en los que sí se ha conseguido aprovechar el beneficio es muy grande. El *speedup* obtenido es la raíz del coste total, por lo que podría reducir los cálculos necesarios varios órdenes de magnitud a la hora de entrenar modelos muy complejos.

6 Computación cuántica adiabática

La computación cuántica adiabática (*adiabatic quantum computing, AQC* en inglés) es un algoritmo de computación cuántica que se puede utilizar para resolver problemas de optimización con un gran número de variables. Aprovechando ciertas propiedades específicas de la física cuántica, el algoritmo es capaz de resolver problemas de optimización complejos en menos pasos que un ordenador normal. Este tipo de algoritmo no está basado en puertas o circuitos cuánticos, si no en otro tipo de computación cuántica conocida como evolución adiabática [44].

6.1 Algoritmo de evolución adiabática

La computación adiabática consiste en formular el problema de optimización que queremos resolver como los niveles de energía del sistema cuántico, de manera que la solución del problema corresponda con el estado con menor energía del sistema. En la mecánica cuántica, los niveles de energía de cada uno de los estados del sistema se conoce como el Hamiltoniano del sistema. En otras palabras, tenemos que encontrar un Hamiltoniano $\mathcal{H}_f$ cuyo estado cuántico de menor energía coincida con el estado cuántico del sistema que representa la solución a nuestro problema. Utilizaremos la notación $|\mathcal{H}\rangle_\downarrow$ para denotar el estado de menor energía de un Hamiltoniano $\mathcal{H}$.

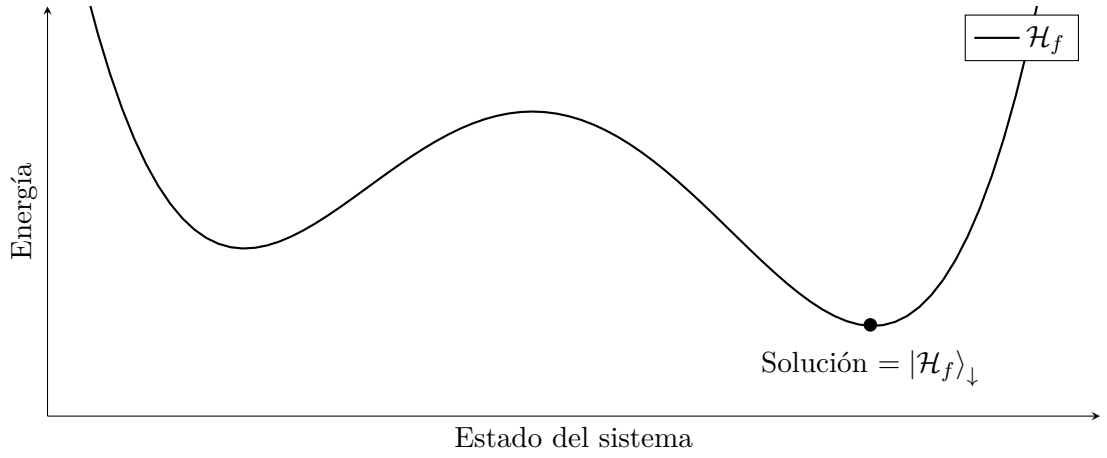

Figura 6.1: Hamiltoniano que resuelve un problema de optimización. Fuente: elaboración propia

Este paso puede ser un poco complicado, ya que depende del tipo específico de hardware que estemos utilizando. Distintas arquitecturas de ordenadores adiabáticos tendrán diferentes maneras de especificar condiciones a sus sistemas, y producirán Hamiltonianos diferentes. Más adelante analizaremos uno de estos tipos de arquitecturas, y veremos como construir

Hamiltonianos para resolver problemas en esa arquitectura y como la arquitectura regula los niveles de energía del sistema para variar el Hamiltoniano.

Una vez hemos encontrado el Hamiltoniano $\mathcal{H}_f$ (que es potencialmente muy complejo), ponemos el sistema a un Hamiltoniano simple, que su estado con menor energía sea conocido. Este Hamiltoniano, que denominaremos $\mathcal{H}_i$, normalmente es un Hamiltoniano con muy poca energía en uno de sus estados, y mucha más en cualquier otro. Tras configurar el Hamiltoniano del sistema a $\mathcal{H}_s$, entonces ponemos el sistema al estado con menor energía de $\mathcal{H}_i$, $|\mathcal{H}_i\rangle_\downarrow$. La solución del problema se encuentra en $|\mathcal{H}_f\rangle_\downarrow$, el estado de menor energía de $\mathcal{H}_f$. El último paso es transformar gradualmente el Hamiltoniano del sistema de $\mathcal{H}_i$ a $\mathcal{H}_f$. Si lo transformamos de forma suficientemente gradual, el teorema adiabático [45] nos garantiza que el sistema se mantendrá en el estado de menor energía, por lo que cuando el Hamiltoniano sea $\mathcal{H}_f$, el estado del sistema será la solución del problema de optimización. Este proceso se conoce como Evolución adiabática.

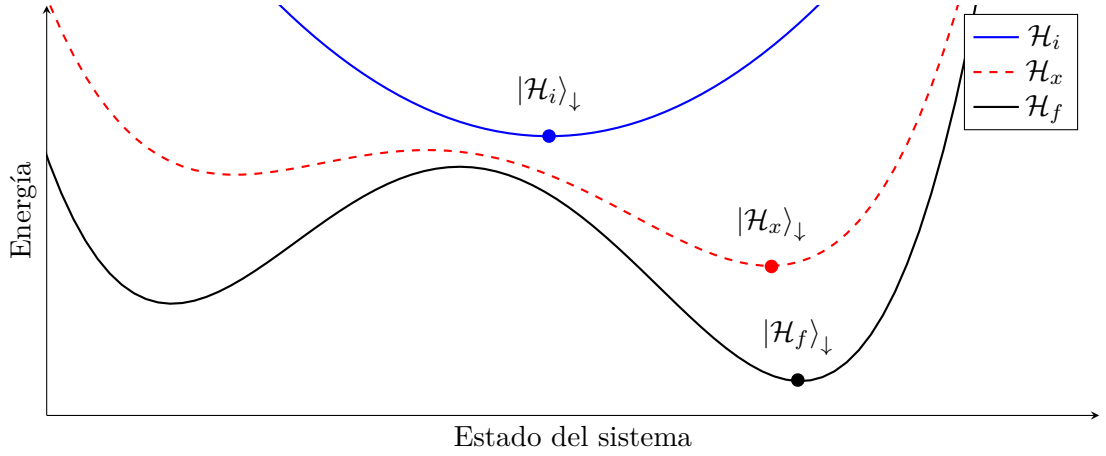

Figura 6.2: Evolución adiabática de $\mathcal{H}_i$ a $\mathcal{H}_f$ pasando por $\mathcal{H}_x$. Fuente: elaboración propia

Calcular la velocidad máxima a la que podemos mover el Hamiltoniano (y por lo tanto, la complejidad temporal del algoritmo) no es tarea fácil. Primero, necesitamos calcular el *Spectral gap* (salto espectral) de un Hamiltoniano. Sea $|\mathcal{H}\rangle_\downarrow$ el estado de menor energía de un Hamiltoniano, $|\mathcal{H}\rangle_{\downarrow 2}$ el segundo estado de menor energía de un hamiltoniano y $H|\psi\rangle$ la energía del estado $|\psi\rangle$ para el Hamiltoniano $\mathcal{H}$, entonces podemos calcular el *Spectral gap* de un Hamiltoniano utilizando la siguiente fórmula:

$$\Delta_\mathcal{H} = \mathcal{H}|\mathcal{H}\rangle_{\downarrow 2} - \mathcal{H}|\mathcal{H}\rangle_\downarrow \tag{6.1}$$

Ahora definimos $\mathcal{H}(t)$, el hamiltoniano del sistema en el instante t. En el instante inicial t_0 tenemos $\mathcal{H}(t_0) = \mathcal{H}_i$, y consecuentemente en el instante final t_f tenemos $\mathcal{H}(t_f) = \mathcal{H}_f$. A partir de todo esto podemos definir la fórmula que determina la velocidad v a la que nos podemos desplazar sin perturbar el sistema.

$$v \propto \left(\min_{t_0 \leq t \leq t_f} \Delta_{\mathcal{H}(t)} \right)^2 \tag{6.2}$$

Esta velocidad es proporcional al cuadrado del menor *Spectral gap* de $\mathcal{H}(t)$, para $t_0 \leq t \leq t_f$ [45]. Cuando más grande sea el mínimo *Spectral gap*, a mayor velocidad podremos modificar el hamiltoniano. Como la velocidad es inversamente proporcional a la complejidad temporal, obtenemos la siguiente complejidad:

$$O\left(\frac{1}{v}\right) = O\left(\frac{1}{\left(\min_{t_0 \leq t \leq t_f} \Delta_{\mathcal{H}(t)}\right)^2}\right) \tag{6.3}$$

Esta complejidad es complicada de comparar a cualquier la de otros algoritmos de optimización, ya que depende de los hamiltonianos por los que pase el sistema a la hora de ser optimizado. No obstante, es posible que para ciertos problemas esta manera de optimizarlos sea más rápida que otros métodos de optimización. Además, se ha demostrado que este tipo de computación es equivalente en potencia que a la computación cuántica basada en puertas lógicas cuánticas [46], por lo que no hay ningún problema de optimización que se pueda resolver por otros métodos cuánticos en complejidad $O(f(n))$ y no por este.

6.1.1 Evolución adiabática aproximada

En muchos casos, no queremos una solución óptima, si no una solución cercana a la solución óptima. En estos casos, podemos modificar el algoritmo de evolución adiabática para, en vez de garantizar la solución óptima, garantizar una solución que esté entre las n mejores soluciones.

Esto lo conseguimos considerando todos los *spectral gaps* entre las n mejores soluciones. Sea $|\mathcal{H}\rangle_{\downarrow n}$ el estado n de menor energía de $\mathcal{H}$, entonces tendríamos el siguiente *spectral gap* entre $|\mathcal{H}\rangle_{\downarrow n}$ y el siguiente estado de menor energía, $|\mathcal{H}\rangle_{\downarrow n-1}$:

$$\Delta_{\mathcal{H}}^{\downarrow n} = \mathcal{H}\,|\mathcal{H}\rangle_{\downarrow n} - \mathcal{H}\,|\mathcal{H}\rangle_{\downarrow n-1} \tag{6.4}$$

Entonces, el *spectral gap* total para los n estados de menor energía de $\mathcal{H}$ (las n mejores soluciones de $\mathcal{H}$), será el máximo *spectral gap* de cada uno de los estados:

$$\Delta_{\mathcal{H}} = \max_{1 \leq i \leq n} \Delta_{\mathcal{H}}^{\downarrow i} = \max_{1 \leq i \leq n} \left(\mathcal{H}\,|\mathcal{H}\rangle_{\downarrow i} - \mathcal{H}\,|\mathcal{H}\rangle_{\downarrow i-1}\right) \tag{6.5}$$

Si se selecciona una n suficientemente grande, este *spectral gap* será mucho mayor que el que hemos visto en la evolución adiabática pura. Como la velocidad a la que podemos evolucionar el sistema es proporcional al cuadrado del mínimo *spectral gap* (ver ecuación 6.2), aumentar el *spectral gap* nos permite aumentar cuadráticamente la velocidad máxima a la que podemos evolucionar el hamiltoniano del sistema. Como el tiempo requerido para realizar la evolución del hamiltoniano y obtener la solución al problema que estamos resolviendo es inversamente proporcional a la velocidad a la que podemos evolucionar el sistema (ver ecuación 6.3), este tipo de evolución adiabática tendrá un tiempo de ejecución mucho menor que la evolución adiabática pura. Si no es necesario obtener la solución óptima y sirve con obtener una solución buena, este tipo de evolución adiabática puede proporcionar tiempos de procesamiento mucho menores.

6.2 Problemas de satisfacibilidad

Este tipo de algoritmos son especialmente útiles para problemas de satisfacibilidad, es decir, problemas en los que tengamos que encontrar alguna solución que satisfaga todas las restricciones del problema. Estos problemas están caracterizados por las restricciones R_i, y para resolver el problema tenemos que encontrar un estado $|\psi\rangle$ que cumpla $R_1 \wedge R_2 \wedge \cdots \wedge R_M$, siendo R_i una de las M restricciones, que puede ser verdadera o falsa para cada estado. Para este tipo de problemas, es muy sencillo encontrar el hamiltoniano final $\mathcal{H}_f$. Para la restricción R_i, construiremos el siguiente hamiltoniano parcial:

$$\mathcal{H}_f^i \, |\psi\rangle = \begin{cases} a_i & \text{si } |\psi\rangle \text{ satisface } R_i \\ b_i & \text{si } |\psi\rangle \text{ no satisface } R_i \end{cases} \tag{6.6}$$

Para todo i, $a_i < b_i$, de forma que los estados que cumplan más restricciones tengan menor energía. Una vez tenemos los hamiltonianos parciales, podemos construir el hamiltoniano final $\mathcal{H}_f$ a partir de ellos:

$$\mathcal{H}_f = \sum_i \mathcal{H}_f^i \tag{6.7}$$

Visualmente, obtendríamos una gráfica de energía similar a la siguiente:

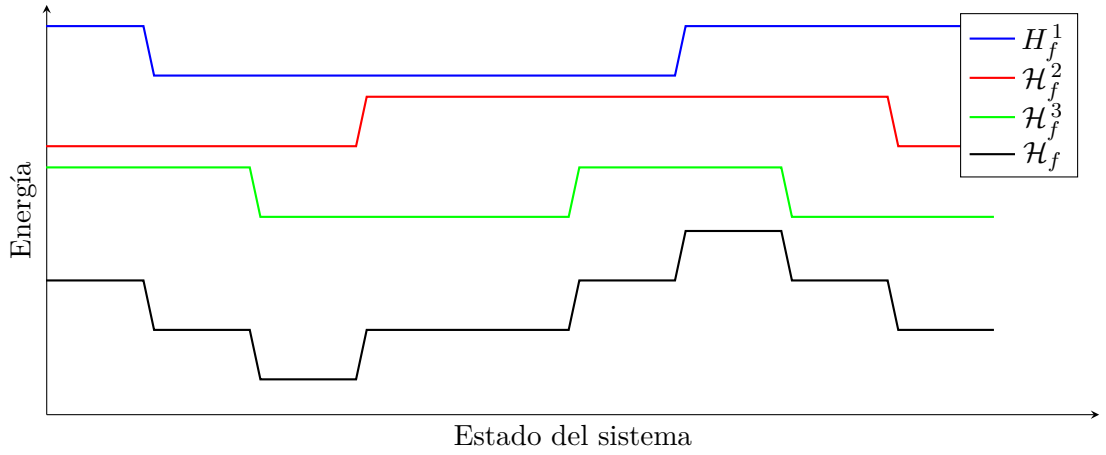

Figura 6.3: Hamiltonianos finales para un problema de satisfacibilidad. Fuente: elaboración propia

De esta manera, los únicos estados para los que $\mathcal{H}\,|\psi\rangle$ es mínimo serán los estados que cumplan todas las restricciones. En el caso de que no exista ningún estado que cumpla todas, el estado "solución" seguirá siendo el estado con menor energía.
Para realizar la construcción de los hamiltonianos parciales, se puede hacer de manera mucho más general. La única restricción importante para que los hamiltonianos parciales sean válidos, es que haya un salto significativo entre los estados que satisfacen y que no la satisfacen, de manera que ese salto sea mayor que la diferencia de energía de los estados que si satisfacen la condición. Podemos reflejar esto introduciendo dos variables nuevas en la fórmula, $v_i^a(|\psi\rangle)$ y $v_i^b(|\psi\rangle)$. Estas dos variables miden cuánto difiere el valor real del hamiltoniano con el valor

de la fórmula ideal que vimos anteriormente.

$$\mathcal{H}_f^i(|\psi\rangle) = \begin{cases} a_i + v_i^a(|\psi\rangle) & \text{si } |\psi\rangle \text{ satisface } R_i \\ b_i + v_i^b(|\psi\rangle) & \text{si } |\psi\rangle \text{ no satisface } R_i \end{cases} \tag{6.8}$$

Siempre que $v_i^a(|\psi\rangle)$ y $v_i^b(|\psi\rangle)$ sean mucho menores que el salto $b_i - a_i$, el hamiltoniano servirá para realizar la optimización.

6.3 Ordenadores D-Wave

A diferencia de los ordenadores cuánticos basados en puertas lógicas, si que existen modelos comerciales de ordenadores cuánticos adiabáticos. Esto se debe en gran parte a la compañía D-Wave Quantum Systems Inc., que en 2011 lanzó al mercado el primer ordenador cuántico comercial. El ordenador, nombrado "D-Wave One", utiliza un chip de 128 cúbits que es capaz de resolver ciertos problemas de optimización.

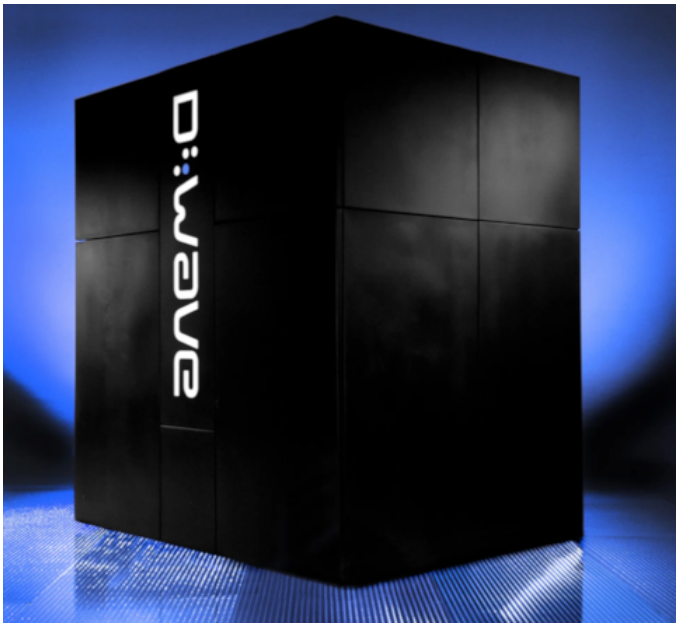

Figura 6.4: Imagen de un ordenador D-Wave One. Fuente: [47]

El ordenador D-Wave One mide tres metros de alto y más de dos de largo. Gran parte de este espacio es para aislar térmicamente el procesador del mundo exterior, ya los procesadores D-Wave operan a una temperatura de $20mK$ $(-273.13C)$, una temperatura tan baja que está solo a 0.02 grados del cero absoluto, la temperatura más baja posible.

Figura 6.5: Temperaturas de cada placa de refrigeración de un ordenador D-Wave. Fuente: [48]

El D-Wave One está limitado a problemas de optimización discreta, que son problemas de optimización en los que el resultado es discreto. Para problemas continuos, hay que reformular el problema, haciendo que sea necesario gastar cúbits adicionales para representar los estados resultado. No obstante, incluso con esta restricción, fue un gran primer paso en el desarrollo de los ordenadores adiabáticos, ya que con el potencial de sus 128 cúbits se ha avanzado mucho el campo. Desde entonces, D-Wave ha producido una variedad de diferentes modelos, cada uno con mayor número de cúbits que el anterior:

Modelo	Número de cúbits	Lanzamiento comercial
D-Wave One	128	2011
D-Wave Two	512	2013
D-Wave 2X	1152	2015
D-Wave 2000Q	2048	2017
Advantage	>5000	—

Tabla 6.1: Ordenadores adiabáticos D-Wave. Fuente: [49]

Aunque D-Wave One no fuese capaz de obtener mejores resultados que los ordenadores normales , los ordenadores modernos de D-Wave son mucho más rápidos para ciertos problemas de optimización [50]. Con cada generación no solo se mejora la cantidad de cúbits, si no que cada uno de los cúbits se puede configurar de manera más precisa, ofreciendo un mayor set de problemas que el ordenador puede resolver, y haciendo que la optimización sea más rapida

para otros.

6.3.1 Arquitectura D-Wave

En los ordenadores D-Wave, se usa una arquitectura basada en el enlazado de cúbits. Aunque la arquitectura específica es diferente en cada uno de los ordenadores [51], todos tienen la misma estructura general. Los hamiltonianos que puede producir los ordenadores D-Wave se determinan a partir de dos factores: el peso base de cada cúbit y la correlaciones entre cada par de cúbits [52]. Dividiremos el hamiltoniano final en dos partes, una para los pesos y otra para las correlaciones, y definiremos matemáticamente cada una por separado. Si $\mathcal{H}_p$ es el hamiltoniano de los pesos y $\mathcal{H}_c$ es el hamiltoniano de correlaciones, podemos definir el hamiltoniano final producido por el ordenador con la siguiente fórmula:

$$\mathcal{H}_f = \mathcal{H}_p + \mathcal{H}_c \tag{6.9}$$

El peso es la parte más sencilla del hamiltoniano. Para cada estado, se multiplicará el valor cada cúbit (0 o 1) con su peso p_i. Sumando todos los pesos, obtenemos el hamiltoniano de pesos $\mathcal{H}_p$, siendo $|\psi\rangle_i$ el valor del cúbit i en el estado $|\psi\rangle$:

$$\mathcal{H}_p(|\psi\rangle) = \sum_i p_i \, |\psi\rangle_i \tag{6.10}$$

El hamiltoniano de enlaces es algo más complejo. Este hamiltoniano representa una correlación entre dos cúbits. Si ambos cúbits miden lo mismo en el estado $|\psi\rangle$, se sumará a la energía del estado $|\psi\rangle$ el valor de la correlación para los dos cúbits. Matemáticamente, si c_{ij} es el valor de correlación para cada pareja de cúbits, podemos expresar $\mathcal{H}_c$ con esta fórmula:

$$\mathcal{H}_c(|\psi\rangle) = \sum_{i<j} c_{ij} \, |\psi\rangle_i \, |\psi\rangle_j \tag{6.11}$$

Por lo tanto, nos queda la siguiente fórmula para el hamiltoniano final:

$$\mathcal{H}_f(|\psi\rangle) = \sum_i p_i \, |\psi\rangle_i + \sum_{i<j} c_{ij} \, |\psi\rangle_i \, |\psi\rangle_j \tag{6.12}$$

Este hamiltoniano tiene la limitación de que solo afecta a estados cuánticos que no están en superposición, por lo que la solución obtenida siempre será un estado fundamental. Esto significa que los ordenadores de esta arquitectura producen bits como salida, y no cúbits, por lo que no se pueden utilizar como un paso intermedio de un algoritmo cuántico sin destruir la información almacenada en las superposiciones. Esto limita la posibilidad de utilizar este tipo de arquitecturas en conjunto con otras técnicas cuánticas para mejorar aun más el rendimiento a la hora de resolver ciertos problemas.

6.3.2 Topología Chimera

Aunque esta arquitectura parezca bastante limitada, sigue permitiéndonos resolver prácticamente cualquier problema de optimización discreta, dado un número suficiente de cúbits con suficientes correlaciones entre ellos. No obstante, las arquitecturas D-Wave no están limitadas solamente por el número de cúbits del procesador, si no que también están limitadas por

las correlaciones se pueden establecer. La mayoría de cúbits solo se pueden conectar con un número muy reducido de cúbits. Estas limitaciones físicas se conocen como la topología del procesador [51].

La primera topología desarrollada por D-Wave utilizada en sus ordenadores comerciales es la topología "Chimera". Esta topología está basada en celdas bidimensionales de cúbits. Una celda de tamaño n tiene n cúbits horizontales y n cúbits verticales, y cuenta con n^2 correlaciones internas, de manera que cada cúbits horizontal está conectado con todos los cúbits verticales y viceversa.

Visualmente, podemos representar una celda de la siguiente forma, siendo las filas cada uno de los cúbits verticales, las columnas los cúbits horizontales y las intersecciones las correlaciones internas de la celda:

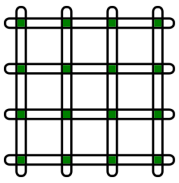

Figura 6.6: Celda D-Wave de tamaño 4, con 8 cúbits y 16 correlaciones internas. Fuente: [51]

Las arquitecturas Chimera están formadas por celdas de tamaño 4, posicionadas en una cuadrícula bidimensional. Además, para conectar las celdas de cúbits entre sí, también se conectan todos los cúbits que pertenecen a la misma fila o columna adjacentes, mediante conexiones externas entre celdas. De esta manera, un procesador de 32 cúbits de topología Chimera tendría las siguientes correlaciones:

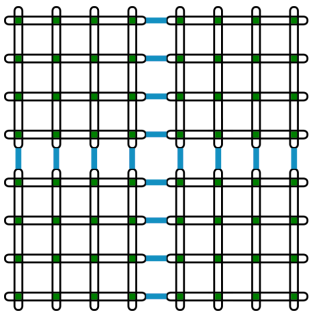

Figura 6.7: Correlaciones internas (verde) y externas (azul) de un Chimera de 32 cúbits. Fuente: [51]

De esta forma, cada cúbit tendrá 4 correlaciones internas, y 1 o 2 correlaciones externas. Los cúbits que se sitúan al borde de la cuadrícula de celdas tienen 1 correlación externa, mientras que los cúbits centrales tienen 2. En el caso ideal de que trabajásemos con un procesador de tamaño infinito, todos los cúbits tendrían 6 correlaciones. Por lo tanto, el número máximo de correlaciones por cúbit para este tipo de arquitecturas es 3, ya que al tener 6 correlaciones por cúbit, como cada correlación está en dos cúbits, tenemos 3 veces más correlaciones que

cúbits.

Como hay 8 cúbits en cada célula de tamaño 4, para una topología Chimera de $n \times m$ celdas tendremos $8nm$ cúbits. Para esa topología, tendremos también la siguiente fórmula para calcular la cantidad de correlaciones:

$$c = 16nm + 4n(m-1) + 4m(n-1) = 4(6ab - a - b) \tag{6.13}$$

El procesador de D-Wave One utilizaba una topología Chimera, pero con una cuadrícula de 4×4 celdas, que lo dotaba con 128 bits de potencia. Utilizando la fórmula, podemos calcular que tiene un total de 352 correlaciones en todo el procesador. Visualmente, la arquitectura del procesador sería la siguiente:

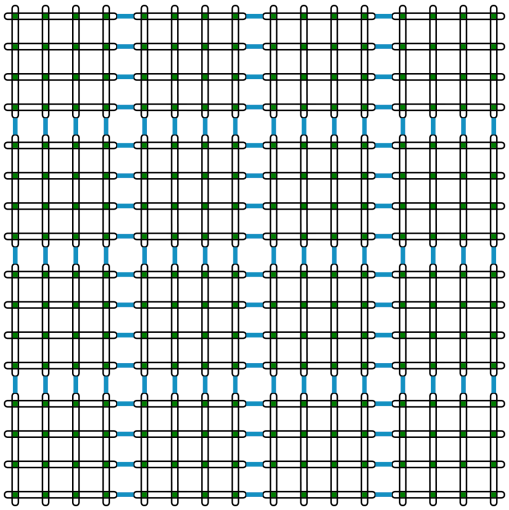

Figura 6.8: Visualización del procesador D-Wave One. Fuente: elaboración propia, basada en [51]

El resto de procesadores que utilizan la topología Chimera son demasiado grandes para visualizarse. No obstante, podemos calcular las correlaciones y los cúbits de sus arquitecturas:

Modelo	Topología	Cúbits	Correlaciones	Correlaciones por cúbit
D-Wave One	Chimera 4×4	128	352	5.50
D-Wave Two	Chimera 8×8	512	1472	5.75
D-Wave 2X	Chimera 12×12	1152	3360	5.83
D-Wave 2000Q	Chimera 16×16	2048	6016	5.88

Tabla 6.2: Topologías Chimera utilizadas en los procesadores D-Wave. Fuente: elaboración propia

Como se puede ver en la tabla, cuantos más cúbits mayor es la cantidad de correlaciones por cúbits, ya que hay más cúbits centrales con 6 correlaciones. Por mucho que aumentemos los cúbits, nunca se llegará al ideal de 6 correlaciones por cúbit, ya que en algún momento el procesador tendrá que terminar y tendremos que tener cúbits con solo 5 correlaciones.

6.3.3 Topología Pegasus

Los nuevos procesadores D-Wave del modelo Advantage utilizan una topología diferente, la topología Pegasus. En esta topología los cúbits están organizados en parejas en vez de en celdas. Cada cúbit está conectado a 12 cúbits perpendiculares, además de estar conectado al otro cúbit de su pareja y al siguiente y anterior cúbit paralelo.

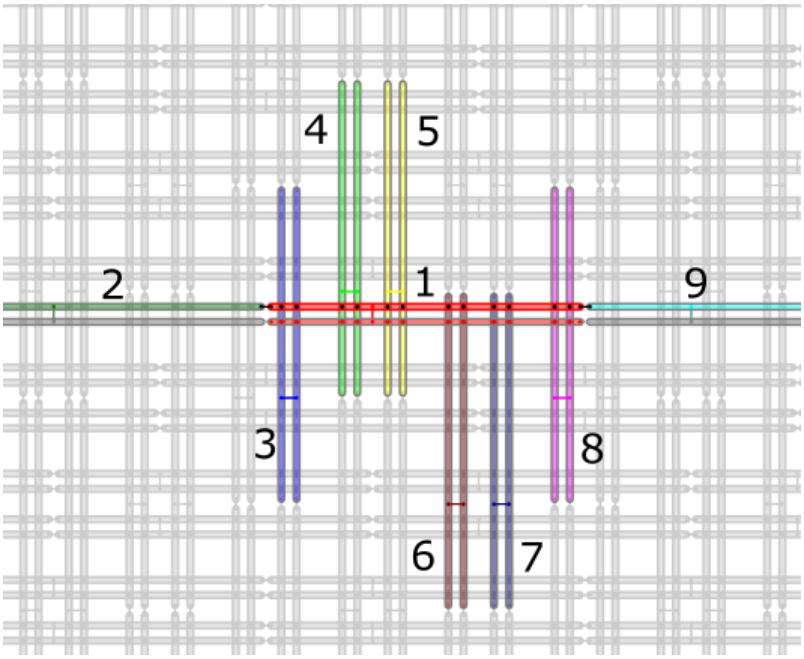

Figura 6.9: Correlaciones de los cúbits de la topología Pegasus. Fuente: [51]

Como se puede ver en la figura anterior, el cúbit superior de la pareja 1 está conectado a todos los cúbits de las parejas 3, 4, 5, 6, 7 y 8, a los cúbits superiores de las parejas 2 y 9, y al otro cúbit de la pareja 1. Esto hace un total de 15 correlaciones.

Utilizando esta topología, las correlaciones máximas de un cúbit pasan de 6 a 15. Con esta densidad de correlaciones, se podrán resolver problemas mucho más complejos sin necesidad de utilizar cúbits adicionales, o resolver problemas similares utilizando significativamente menos cúbits.

6.4 Aplicaciones en la inteligencia artificial

La computación adiabática, gracias a su *hardware* relativamente avanzado comparado con el resto de ordenadores cuánticos, ha sido uno de los métodos que más se ha investigado para la implementación de la computación cuántica en la inteligencia artificial. Algunos de los algoritmos de inteligencia artificial para los que se ha conseguido utilizar la computación adiabática son los siguientes:

- Clasificadores binarios [53]

- Máquinas de vectores de soporte (*support-vector machines*, SVMs) [54]
- Redes neuronales embebidas en hardware [55]

La computación adiabática, dado su avance tecnológico comparado con los ordenadores cuánticos basados en puertas lógicas, probablemente sea una de las primeras implementaciones de la computación cuántica que se aplicarán a la inteligencia artificial. Aunque con ordenadores cuánticos basados en puertas se pueden incorporar muchas mejoras adicionales, la implementación de evolución adiabática en el entrenamiento de modelos de inteligencia artificial modernos podría constituir una gran reducción en la cantidad de tiempo y recursos necesarios para completar el entrenamiento de dichos modelos.

7 Conclusiones

Como se ha podido ver durante el desarrollo de este trabajo, la computación cuántica podría suponer un increíble beneficio para la inteligencia artificial. Ya sea mediante mejores algoritmos o mediante técnicas que permiten disminuir drásticamente el tiempo y los recursos necesarios, la computación cuántica podría suponer una revolución para la inteligencia artificial.

La implementación de muchas de las mejoras estudiadas es muy compleja. La naturaleza de los ordenadores cuánticos requiere que, para su implementación, aprovechemos sus propiedades y ventajas específicas, que son muy diferentes de las propiedades de los ordenadores actuales. No obstante, el posible beneficio de las mejoras estudiadas es tan grande, que un cambio radical en la manera en la que programamos es un coste aceptable.

El mayor obstáculo actual es el tecnológico. Los ordenadores cuánticos actuales y su *hardware* es muy caro de fabricar, ya que requiere de componentes muy especializados y precisos; y también es caro de operar, ya que normalmente los ordenadores cuánticos se tienen que mantener a temperaturas muy cercanas al cero absoluto [56]. Además, se requieren de equipos especializados para su uso y mantenimiento. Todos estos factores hacen que los ordenadores cuánticos actuales cuesten decenas de millones de dólares para ser adquiridos, y varios millones de dólares por año para su mantenimiento y uso [57]. Esto los hace prohibitivos en la mayoría de aplicaciones comerciales, haciendo que actualmente se suelan utilizar exclusivamente para la investigación.

No hay duda que si se consigue mitigar los costes tecnológicos, la computación cuántica podría suponer un antes y un después en el entrenamiento y ejecución de los modelos de inteligencia artificial. Esta reducción de costes y tiempo de entrenamiento podría suponer el abaratamiento del desarrollo de modelos de potencia similar a los modelos actuales, y también podría hacer viable el entrenamiento de modelos mucho más potentes a los actuales, sin la necesidad de incrementar exageradamente el coste y/o tiempo de entrenamiento.

7.1 Aportaciones

Las aportaciones de este TFG con las siguientes:

- Estudio de la computación cuántica aplicada a la IA: Se han explorado diversas aplicaciones de la computación cuántica aplicadas a la inteligencia artificial, priorizando aquellas con mayor beneficio potencial o con más desarrollo previo e implementaciones físicas.

- Explicación de la computación cuántica y la inteligencia artificial: se ha explicado en detalle muchos de los conceptos fundamentales de la computación cuántica y la IA necesarios para entender el resto del trabajo realizado.

- Análisis de algoritmos de multiplicación de matrices: Se han evaluado varios algoritmos, tanto clásicos como cuánticos, para la multiplicación de matrices. Se ha hecho notar que el algoritmo cuántico basado en *swap tests* tiene una complejidad cuadrática, lo cual es significativamente menor que los mejores algoritmos clásicos disponibles.

- Comparación de complejidades de algoritmos de matrices: Se ha realizado un análisis comparativo de las complejidades de diferentes algoritmos de multiplicación de matrices, destacando la ventaja del algoritmo cuántico en términos de eficiencia.

- Análisis de la búsqueda de Grover: se ha realizado una breve explicación y un desarrollo de la búsqueda de Grover, junto con un pequeño análisis de los posibles beneficios que la misma podría traer a la inteligencia artificial. También se han investigado las posibles aplicaciones de la misma, viendo algunas implementaciones que han conseguido aprovecharla para mejorar la eficiencia de entrenamiento y aprendizaje.

- Estudio de la computación cuántica adiabática: se han investigado las posibles aplicaciones de la computación adiabática, explicando en detalle ciertos conceptos fundamentales necesarios para su entendimiento. Se ha destacado su particular utilidad en problemas de optimización y satisfacibilidad, y se han investigado sus posibles aplicaciones en la IA. También se han analizado algunos ordenadores y procesadores adiabáticos comerciales, estudiando su arquitectura y topología.

Estas contribuciones muestran un esfuerzo significativo por entender y mejorar la integración de la computación cuántica con la inteligencia artificial, resaltando tanto las ventajas potenciales como los desafíos actuales

7.2 Posibles ampliaciones

Durante la realización de este trabajo se han analizado y estudiado ciertas aplicaciones de la computación cuántica aplicadas a la inteligencia artificial. Como se explicó en el apartado 1.2, se han priorizado las aplicaciones más interesantes, que podrían tener mayor beneficio o que dispongan de mayor cantidad de estudios o de implementaciones físicas.

A continuación se enumeran algunas de las aplicaciones que no se han analizado, junto con un pequeño resumen de su posible uso en la inteligencia artificial y la razón de porqué no se han analizado:

- Convoluciones cuánticas: el uso de circuitos cuánticos para implementar convoluciones de matrices. Su implementación beneficiaría a las redes neuronales convolucionales, que se utilizan para el procesamiento de imágenes, audio y vídeo, entre otros usos. No se ha analizado ya que sería una versión menos general y con menos beneficios que el análisis ya realizado para los algoritmos de multiplicación de matrices, que se aplican a todas las redes neuronales y no solo a las convolucionales.

- Redes neuronales cuánticas (*Quantum Neural Networks*, QNNs): se trata de implementar redes neuronales completamente con ordenadores cuánticos, incluyendo la ejecución entera de la red y posiblemente el entrenamiento de la misma. Su implementación podría ahorrar un gran coste en los ordenadores cuánticos, la inicialización de los estados,

ya que al procesar datos de forma completamente cuántica solo tendrían que inicializarse una vez. No se ha analizado debido a que no existen implementaciones de este concepto que no sean con redes neuronales muy pequeñas, y que aún no existen estudios sobre si es posible implementar todas las operaciones necesarias para el funcionamiento de una red neuronal de forma eficiente con ordenadores cuánticos.

- Codificación en amplitudes (*Amplitude encoding*): consiste en realizar operaciones codificando los valores de la operación en las amplitudes en vez de los estados. Esto, aunque supone una mayor complejidad de cálculo y mucha más inestabilidad numérica, podría suponer un gran ahorro de tiempo, ya que permite hacer muchísimas operaciones en paralelo. No se ha analizado ya que, aunque en teoría se puedan realizar operaciones con esta codificación, solo se ha conseguido implementar en casos muy concretos, y no está claro aún que se pueda aplicar a la IA de forma general en entornos reales.

Bibliografía

[1] Tim Davis, "Explainer: what is wave-particle duality." https://theconversation.com/explainer-what-is-wave-particle-duality-7414. Accedido: 30/3/2024.

[2] Sheroy Cooper, "Quantum superposition - explained simply and in-depth." https://medium.com/@sheroy.cooper/quantum-superposition-explained-simply-and-in-depth-82736420a939. Accedido: 30/3/2024.

[3] Jesse Emspak, "What is quantum entanglement?." https://www.space.com/31933-quantum-entanglement-action-at-a-distance.html. Accedido: 30/3/2024.

[4] Nanowerk, "What are quantum dots?." https://www.nanowerk.com/what_are_quantum_dots.php. Accedido: 31/3/2024.

[5] Wikipedia, "Qubit: Physical implementations." https://en.wikipedia.org/wiki/Qubit#Physical_implementations. Accedido: 29/3/2024.

[6] P. B. R. Nisbet-Jones, J. Dilley, A. Holleczek, O. Barter, and A. Kuhn, "Photonic qubits, qutrits and ququads accurately prepared and delivered on demand." https://arxiv.org/abs/1203.5614. Accedido: 7/5/2024.

[7] Brain_Boost, "Quantum mechanics: What is bra-ket notation?." https://medium.com/@Brain_Boost/quantum-mechanics-what-is-bra-ket-notation-a69b505f9cc4. Accedido: 31/3/2024.

[8] Quantiki, "Hilbert spaces." https://www.quantiki.org/wiki/hilbert-spaces. Accedido: 31/3/2024.

[9] Scott Aaronson, "Why are amplitudes complex?." https://scottaaronson.blog/?p=4021. Accedido: 31/3/2024.

[10] Kim Thibault, "Euler's formula: A complete guide." https://mathvault.ca/euler-formula/. Accedido: 31/3/2024.

[11] Pranav Viswanath, "Quantum states and the bloch sphere." https://medium.com/quantum-untangled/quantum-states-and-the-bloch-sphere-9f3c0c445ea3. Accedido: 1/4/2024.

[12] Wolfram Mathworld, "Spherical coordinates." https://mathworld.wolfram.com/SphericalCoordinates.html. Accedido: 1/4/2024.

[13] Daniel Winton, "What are bell states?." https://www.aliroquantum.com/blog/what-are-bell-states. Accedido: 1/4/2024.

[14] Quantum Physics Lady, "Quantum nonlocality." https://quantumphysicslady.org/glossary/quantum-nonlocality/. Accedido: 1/4/2024.

[15] Marco Taboga, "Linear independence." https://www.statlect.com/matrix-algebra/linear-independence. Accedido: 1/4/2024.

[16] Wikipedia, "Quantum logic gate: Logic function synthesis." https://en.wikipedia.org/wiki/Quantum_logic_gate#Logic_function_synthesis. Accedido: 1/4/2024.

[17] Wikipedia, "Deferred measurement principle." https://en.wikipedia.org/wiki/Deferred_measurement_principle. Accedido: 1/4/2024.

[18] IBM, "What is overfitting?." https://www.ibm.com/topics/overfitting. Accedido: 1/4/2024.

[19] IBM, "¿que es el aprendizaje no supervisado?." https://www.ibm.com/es-es/topics/unsupervised-learning. Accedido: 1/4/2024.

[20] AWS, "¿qué es el aprendizaje mediante refuerzo?." https://aws.amazon.com/es/what-is/reinforcement-learning/. Accedido: 1/4/2024.

[21] DataScientest, "Perceptrón: ¿qué es y para qué sirve?." https://datascientest.com/es/perceptron-que-es-y-para-que-sirve. Accedido: 7/5/2024.

[22] Joaquín Amat Rodrigo, "Algoritmo perceptrón: linealmente separable." https://cienciadedatos.net/documentos/50_algoritmo_perceptron#linealmente_separable, 2018. Accedido: 24/3/2024.

[23] Joaquín Amat Rodrigo, "Algoritmo perceptrón: hiperplano." https://cienciadedatos.net/documentos/50_algoritmo_perceptron#hiperplano, 2018. Accedido: 24/3/2024.

[24] W3 Schools, "Training a perceptron." https://www.w3schools.com/ai/ai_training.asp. Accedido: 23/3/2024.

[25] Michael Nielsen, "Neural networks and deep learning: proof of the four fundamental equations." http://neuralnetworksanddeeplearning.com/chap2.html#proof_of_the_four_fundamental_equations_(optional), 2019. Accedido: 24-3-2024.

[26] Michael Nielsen, "Neural networks and deep learning: the four fundemental equations behind backpropagation." http://neuralnetworksanddeeplearning.com/chap2.html#the_four_fundamental_equations_behind_backpropagation, 2019. Accedido: 24/3/2024.

[27] "Keras: optimizers." https://keras.io/api/optimizers/rmsprop/. Accedido: 24/3/2024.

[28] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.

[29] J. Huang, T. Smith, G. Henry, and R. van de Geijn, "Strassen's algorithm reloaded." https://jianyuhuang.com/papers/sc16.pdf, 2016. Accedido: 7/5/2024.

[30] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, 1990. Computational algebraic complexity editorial.

[31] V. V. Williams, Y. Xu, Z. Xu, and R. Zhou, "New bounds for matrix multiplication: from alpha to omega." https://arxiv.org/abs/2307.07970, 2023. Accedido: 7/5/2024.

[32] C. Shao, "Quantum algorithms to matrix multiplication." https://arxiv.org/abs/1803.01601, 2018. Accedido: 7/5/2024.

[33] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[34] A. Zaman, H. J. Morrell, and H. Y. Wong, "A step-by-step hhl algorithm walkthrough to enhance understanding of critical quantum computing concepts." https://arxiv.org/abs/2108.09004, 2023.

[35] Alessandro Luongo, "Quantum algorithms: Sve-based quantum algorithms." https://quantumalgorithms.org/chap-svebased.html#spectral-norm-and-the-condition-number-estimation, 2023. Accedido: 25/3/2024.

[36] Maxime, "What is a transformer?." https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04, 2019. Accedido: 25/3/2024.

[37] L. K. Grover, "A fast quantum mechanical algorithm for database search." https://arxiv.org/abs/quant-ph/9605043, 1996. Accedido: 4/4/2024.

[38] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, "Strengths and weaknesses of quantum computing." https://arxiv.org/abs/quant-ph/9701001, 1997. Accedido: 4/4/2024.

[39] Nathan Wiebe and Ashish Kapoor and Krysta Svore, "Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning." https://arxiv.org/abs/1401.2142, 2014. Accedido: 3/4/2024.

[40] Esma Aïmeur and Gilles Brassard and Sébastien Gambs, "Quantum speed-up for unsupervised learning." https://link.springer.com/article/10.1007/s10994-012-5316-5, 2012. Accedido: 3/4/2024.

[41] Y. Gao, Z. Song, X. Yang, and R. Zhang, "Fast quantum algorithm for attention computation." https://arxiv.org/abs/2307.08045, 2023. Accedido: 3/4/2024.

[42] G. D. Paparo, V. Dunjko, A. Makmal, M. A. Martin-Delgado, and H. J. Briegel, "Quantum speedup for active learning agents." https://arxiv.org/abs/2307.08045, 2014. Accedido: 3/4/2024.

[43] N. Wiebe, A. Kapoor, and K. M. Svore, "Quantum perceptron models." https://arxiv.org/abs/1401.4997, 2016. Accedido: 3/4/2024.

[44] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum computation by adiabatic evolution." https://arxiv.org/abs/quant-ph/0001106, 2000. Accedido: 7/5/2024.

[45] T. Kato, "On the adiabatic theorem of quantum mechanics," 1950.

[46] S. Zbinden, A. Bärtschi, H. Djidjev, and S. Eidenbenz, "Embedding algorithms for quantum annealers with chimera and pegasus connection topologies," in *High Performance Computing* (P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, eds.), (Cham), pp. 187–206, Springer International Publishing, 2020.

[47] N. Atlas, "Harvard researchers fold proteins with d-wave quantum computer." https://newatlas.com/harvard-d-wave-quantum-computer/25558/, 2012. Accedido: 5/4/2024.

[48] E. D. Dahl and V. Goliber, "Hardware and software advances in quantum annealing." https://www.suny.edu/media/suny/content-assets/images/research/events/Hardware-Software-Advances-in-Quantum-Annealing-DWaveSlides-Dahl_Goliber.pdf, 2019. Accedido: 5/4/2024.

[49] Wikipedia, "D-wave systems." https://en.wikipedia.org/wiki/D-Wave_Systems. Accedido: 12/5/2024.

[50] J. King, S. Yarkoni, M. M. Nevisi, J. P. Hilton, and C. C. McGeoch, "Benchmarking a quantum annealing processor with the time-to-target metric," 2015.

[51] "D-wave qpu architecture: Topologies." https://docs.dwavesys.com/docs/latest/c_gs_4.html. Accedido: 8/3/2024.

[52] "D-wave qpu annealing implementation and controls." https://docs.dwavesys.com/docs/latest/c_qpu_annealing.html. Accedido: 8/3/2024.

[53] H. Neven, M. Drew-Brook, W. G. Macready, V. S. Denchev, J. Zhang, and G. Rose, "Nips 2009 demonstration: Binary classification using hardware implementation of quantum annealing." https://static.googleusercontent.com/media/www.google.com/de//googleblogs/pdfs/nips_demoreport_120709_research.pdf, 2009. Accedido: 5/4/2024.

[54] Z. Li, X. Liu, N. Xu, and J. Du, "Experimental realization of a quantum support vector machine." https://arxiv.org/abs/1410.1054, 2015. Accedido: 5/4/2024.

[55] M. Benedetti, J. Realpe-Gómez, R. Biswas, and A. Perdomo-Ortiz, "Quantum-assisted learning of hardware-embedded probabilistic graphical models." https://arxiv.org/abs/1609.02542, 2017. Accedido: 5/4/2024.

[56] Kiutra, "Quantum computer temperature: Do they need to be cold?." https://kiutra.com/quantum-computer-temperature-do-they-need-to-be-cold/, 2023. Accedido: 28/4/2024.

[57] J. Dargan, "What is the price of a quantum computer in 2024?." https://thequantuminsider.com/2023/04/10/price-of-a-quantum-computer/, 2023. Accedido: 28/4/2024.