Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Teoría de Representación Aplicada al Geometric Quantum Machine Learning

---

Alumno: Pablo Rey Valiente

Tutor: Juan José Álvarez Sánchez

Teoría de Representación Aplicada al Geometric Quantum Machine Learning

Pablo Rey Valiente

Índice general

Lista de figuras III

Resumen VII

Abstract IX

1. Descripción del Proyecto 1
1.1. Introducción 1
1.2. Objetivos 1
1.3. Entorno del Proyecto 2
1.4. Estructura de la Memoria 2

2. Metodología 5
2.1. Proceso de Desarrollo 5
2.2. Arquitectura 6
2.2.1. Arquitectura Física 6
2.2.2. Arquitectura Lógica 7
2.3. Definción de Siglas y Abreviaturas 8

3. Planificación 11
3.1. Estimación del Esfuerzo 11
3.2. Planificación Temporal 12
3.3. Presupuesto Económico 15
3.3.1. Recursos Humanos 15
3.3.2. Recursos Técnicos 16
3.3.3. Coste Final 17

4. Conceptos Teóricos 19
4.1. Computación Cuántica 19
4.1.1. Conceptos Básicos 19
4.1.2. Circuitos y Puertas Cuánticas 20
4.1.3. Sistemas Multi-Qubit 22
4.2. Teoría de Representación 23

5. Herramientas de Desarrollo **27**
5.1. Python como Lenguaje de Programación 27
5.1.1. PennyLane 28
5.1.2. PyTorch 31
5.1.3. Matplotlib 31
5.1.4. NumPy 33
5.2. Jupyter Notebook como IDE 33
5.2.1. Características de Jupyter Notebook 33
5.3. Otras Herramientas Utilizadas 34

6. Análisis **37**
6.1. Diseño 37
6.2. Requisitos 39
6.2.1. Requisitos funcionales 39
6.2.2. Requisitos no funcionales 41
6.3. Atributos de Calidad del Software 42

7. Implementación **45**
7.1. Ejemplos de Representaciones 45
7.1.1. Repesentación de permutación de S_3 en tres qubits 45
7.1.2. Repesentación diédrica de S_3 en un qubit 47
7.2. Geometric Quantum Machine Learning a través del Juego de las Tres en Raya 49

8. Conclusiones **61**

A. Manuales de Instalación **63**

B. Contenido Adjunto **65**

Índice de figuras

2.1. Interfaz de Jupyter Notebook. 7
2.2. Arquitectura lógica. 8

3.1. Estimación temporal vs Tiempo real. 14
3.2. Diagrama de Gantt. 15
3.3. Recursos técnicos. 16

4.1. Qubits de un circuito. 20
4.2. Ejemplo de un circuito cuántico. 20
4.3. Esfera de Bloch con distintas operaciones y estados cuánticos. 22
4.4. Simetrías de una esfera. 24
4.5. Una aplicación equivariante. 25

5.1. Ejemplo de la visualización de un circuito con qml.draw_mpl. 30
5.2. Amplitud de los estados base tras aplicar una rotación RX. 32
5.3. Ejemplo de celda Markdown sin compilar. 34
5.4. Ejemplo de celda Markdown compilada. 34

6.1. Diseño del proyecto. 38

7.1. Circuito cuántico que inicializa los qubits y aplica una puerta SWAP. . . . 46
7.2. Grupo diédrico D3 47
7.3. Circuito cuántico que aplica dos operaciones a un qubit. 48
7.4. Ejemplo de la ejecución de TicTacToeWidget trás insertar algunos símbolos. 49
7.5. Invarianza del resultado bajo simétrías. 50
7.6. Giros con las simetrías del cuadrado. 51
7.7. Circuito cuántico bajo el que se codifican los datos. 55
7.8. Comparación de precisiones de ambos modelos. 60

A.1. Cabecera de un archivo ejecutado por Jupyter Notebook. 64

Dedicado a
mi familia

Resumen

Este trabajo de fin de grado investiga la aplicación de la teoría de representación al aprendizaje automático cuántico geométrico, enfatizando especialmente el uso de simetrías para mejorar el rendimiento del modelo. Además de ser un trabajo de investigación, también sirve como recurso educativo, con el objetivo de elucidar conceptos matemáticos complejos en la computación cuántica para una audiencia más amplia. Al explicar cómo las simetrías pueden optimizar los algoritmos de aprendizaje, este estudio contribuye tanto a la comprensión teórica como a los avances prácticos en la computación cuántica.

Palabras claves: aprendidaje automático, teoría de representación, computación cuántica, PennyLane, Jupyter, simetría, tres en raya.

Abstract

This thesis investigates the application of representation theory to geometric quantum machine learning, particularly emphasizing the utilization of symmetries to enhance model performance. It serves not only as a research endeavor but also as an educational resource, aiming to elucidate complex mathematical concepts in quantum computing for a broader audience. By explaining how symmetries can optimize learning algorithms, this study contributes to both theoretical understanding and practical advancements in quantum computing.

Key words: machine learning, representation theory, quantum computing, PennyLane, Jupyter, symmetry, tic tac toe.

Capítulo 1

Descripción del Proyecto

1.1. Introducción

En las últimas décadas, la intersección entre la mecánica cuántica y el aprendizaje automático ha desvelado nuevos horizontes en la ciencia computacional. La aparición del aprendizaje automático cuántico ha abierto un sinfín de posibilidades, aprovechando las ventajas inherentes a la computación cuántica para abordar problemas computacionales complejos.

Dentro de este ámbito, la integración del aprendizaje automático cuántico geométrico (*Geometric Quantum Machine Learning*, GQML) se presenta como una vía prometedora. No sólo se basa en los principios de la computación cuántica, sino también en el profundo marco matemático de la teoría de la representación. La teoría de la representación, una rama del álgebra abstracta, desvela la estructura y las simetrías inherentes a los objetos matemáticos representándolos como transformaciones lineales.

La unión de la teoría de la representación con el aprendizaje automático cuántico ofrece una visión profunda de la comprensión y el aprovechamiento de las propiedades geométricas integradas en los sistemas cuánticos. Al explorar las simetrías estructurales presentes en los datos cuánticos, esta amalgama permite el desarrollo de algoritmos innovadores, que prometen una mayor potencia de cálculo y solidez en la resolución de problemas complejos.

Este proyecto profundiza en la estrecha relación entre la teoría de la representación y el aprendizaje automático cuántico geométrico. Su objetivo es desentrañar las conexiones subyacentes, explorar sus influencias mutuas y dilucidar cómo el aprovechamiento de los principios de la teoría de la representación puede mejorar las capacidades de los modelos de aprendizaje automático cuántico geométrico.

1.2. Objetivos

En este estudio teórico sobre la relación entre GQML y la teoría de representación, mi objetivo es explorar cómo estas dos áreas se conectan y cómo podemos utilizar la

teoría de representación para mejorar los modelos de aprendizaje automático cuántico. Con este proyecto, doy mis primeros pasos en el emocionante campo en expansión de la computación cuántica. Mi objetivo es aprender y profundizar en este tema, al mismo tiempo que consolido mis habilidades en Python y el manejo de Jupyter Notebook.

Concretamente, este Trabajo de Fin de Grado consistirá en la elaboración de dos cuadernos Jupyter. El primero de ellos es una introducción teórica que explora los conceptos básicos de la computación cuántica y la teoría de representación. Además, proporciona ejemplos ilustrativos para comprender mejor estas nociones.

El segundo cuaderno, se centra en la implementación práctica de estas ideas. Utilizaremos un modelo de juego de las tres en raya como caso de estudio y aplicaremos la teoría de representación para mejorar el rendimiento del modelo utilizando simetrías.

1.3. Entorno del Proyecto

En el contexto general del proyecto, el enfoque teórico cobra relevancia, especialmente en relación con los avances tecnológicos actuales, particularmente en el campo de la computación cuántica. Durante décadas, la computación cuántica ha sido principalmente teórica, ya que la tecnología necesaria para construir ordenadores cuánticos estaba fuera de alcance. Sin embargo, en la última década, se han puesto en funcionamiento los primeros ordenadores cuánticos, lo que ha permitido finalmente aplicar los resultados teóricos en la práctica [17]. Por ejemplo, en España ya están operativos varios nodos, uno de los cuales forma parte de una red europea de investigación y desarrollo en computación cuántica.

El entorno específico de este proyecto se basa en el uso de la biblioteca PennyLane, desarrollada por Xanadu [3], que proporciona herramientas y funcionalidades específicamente diseñadas para la implementación y simulación de algoritmos cuánticos en Python. PennyLane es una biblioteca de código abierto que permite a los investigadores y desarrolladores trabajar con circuitos cuánticos de manera intuitiva y eficiente, ofreciendo una amplia gama de capacidades para experimentar y explorar el potencial de la computación cuántica. Además, el entorno específico incluye el uso de cuadernos Jupyter como plataforma para desarrollar y presentar el trabajo, lo que facilita la integración de explicaciones teóricas, código ejecutable y visualizaciones interactivas. En conjunto, este entorno proporciona un marco flexible y accesible para investigar y experimentar con conceptos avanzados de computación cuántica y teoría de representación en el contexto de la ingeniería informática.

1.4. Estructura de la Memoria

El trabajo se articula siguiente el siguiente esquema:

- **Capítulo 1.** Descripción del proyecto. Descripción general del proyecto, objetivos, entorno de este y un esquema de la estructura del documento.
- **Capítulo 2.** Metodología. Metodología de trabajo a seguir y sus distintas fases, arquitectura y definición de siglas y abreviaturas.
- **Capítulo 3.** Planificación. Estimación temporal y económica, ligado a los costes derivados de la elaboración del proyecto, comprobando si la realidad se ha ajustado a la planificación prevista.
- **Capítulo 4.** Conceptos teóricos. Explicación teórica de las principales nociones con las que trabajaremos.
- **Capítulo 5.** Herramientas de desarrollo. Recopilación y análisis de los instrumentos utilizados para la consecución del proyecto.
- **Capítulo 6.** Análisis. Diseño de los componentes clave de los cuadernos, así como, una colección de requisitos.
- **Capítulo 7.** Implementación. Proceso de desarrollo de los distintos cuadernos y sus elementos
- **Capítulo 8.** Conclusiones. Comparativa final entre los modelos y visión a futuro del área de estudio.
- **Apéndice A.** Manual de instalación y uso para interactuar con los cuadernos de Jupyter.
- **Apéndice B.** Contenido ajunto junto con esta memoria.
- **Bibliografía.**

Capítulo 2

Metodología

En este capítulo se detalla la metodología de trabajo empleada en el desarrollo y ejecución del presente trabajo de fin de grado. La selección y aplicación de una metodología adecuada es fundamental para el éxito de cualquier proyecto, proporcionando la estructura, los procedimientos y las herramientas necesarias para alcanzar los objetivos establecidos.

2.1. Proceso de Desarrollo

La metodología elegida para el desarrollo del projecto es **RUP (Rational Unified Process** [9]. RUP se basa en su capacidad para manejar proyectos complejos y técnicamente desafiantes al proporcionar una estructura disciplinada, facilitar la gestión de requisitos y ser adaptable a los cambios que puedan surgir durante el desarrollo del proyecto. Algunas de sus principales características son:

- Dirigido por casos de uso, los cuales son los artefactos primarios para establecer el comportamiento deseado del sistema.
- Es Iterativo e incremental, así como adaptable y permite la evolución continua.
- Permite mediciones: de costos, tiempo, nivel de avance, etc.

Es por ello que, aunque nuestro projecto no consista en una aplicación software final destinada al usuario, podremos aprovechar las técnicas de ingeniería aprendidas durante los años en la carrera.

El ciclo de vida de RUP se descompone en 4 fases secunciales:

1. **Fase de Inicio:** En esta etapa, se establecen los fundamentos del proyecto. El objetivo principal es comprender y definir claramente el alcance del proyecto, identificar los *stakeholders* clave y sus necesidades, y establecer una base sólida para el trabajo futuro. En esta fase se define el caso de negocio y se elabora un plan inicial.

2. **Fase de Elaboración:** Aquí se profundiza en los detalles del proyecto. Se enfoca en la arquitectura del sistema, se identifican y resuelven los riesgos técnicos y se elabora un plan detallado para la implementación. Además, se establecen los cimientos técnicos del proyecto y se definen los recursos necesarios.

3. **Fase de Construcción:** Esta etapa se centra en la implementación del sistema. Se desarrolla el software de acuerdo con las especificaciones y diseños establecidos en las fases anteriores. Se realizan pruebas unitarias y se integran los diferentes componentes del sistema. Esta fase tiene como resultado un producto funcional y probado.

4. **Fase de Transición:** En esta última etapa, el sistema se prepara para su entrega al cliente o para su implementación en el entorno de producción. Se llevan a cabo pruebas de aceptación, se realiza la capacitación del usuario final, se prepara la documentación y se garantiza que el sistema esté listo para su despliegue.

Cada una de estas fases en RUP tiene sus propios hitos, entregables y objetivos específicos. El enfoque iterativo e incremental de RUP significa que estas fases no son necesariamente lineales y pueden repetirse o solaparse en diferentes momentos del proyecto para adaptarse a los cambios o refinamientos que puedan surgir.

Estas etapas muy generales hay que adaptarlas a nuestro caso peculiar. La primera fase consite en el estudio y aprendidaje personal sobre la teoría de representación y la computación cuántica. Realmente se centrará en entender los conocimientos básicos necesarios para comenzar a tratar el projecto, ya que el proceso de aprendidaje no finalizará aquí sino que será continuado a medida que avance el estudio. En la fase de elaboración se analizarán las dificultades y se comenzarán a aplicar los conceptos estudiados para diseñar la estructura del proyecto. Durante la fase de construcción se creará el producto, es decir, los documentos que componen el estudio como son los cuadernos Jupyter y la memoria. Está nos llevará la mayor parte del tiempo y para ello nos serviremos de las herramientas que detallaremos más adelante. Por último, en la fase cuarta simplemente nos aseguraremos que el producto final está completo, revisaremos posibles erratas y desarrollaremos las conclusiones finales.

2.2. Arquitectura

2.2.1. Arquitectura Física

En cuanto a la arquitectura física se refiere, por las características del trabajo, solo consideraremos la propia interna de **Jupyter Notebook** [12]. Sus archivos tienen la extensión ".ipynb" y están basados en JSON (JavaScript Object Notation), lo que significa que toda la estructura del notebook está representada en formato JSON (para verlo podemos abrir el archivo con un editor de texto plano).

El núcleo de IPython es utilizado por todas las interfaces de Jupyter, incluyendo el Notebook, la consola Qt, la consola IPython en el terminal y otras interfaces de terceros. IPykernel es un proceso separado encargado de ejecutar el código del usuario y calcular posibles completaciones. Los frontends, como el notebook o la consola Qt, se comunican con el núcleo de IPython mediante mensajes JSON enviados a través de sockets ZeroMQ. El diseño permite que un proceso de kernel esté conectado a múltiples frontends simultáneamente, lo que permite que diferentes interfaces tengan acceso a las mismas variables.

Jupyter Notebook y su interfaz flexible extienden el cuaderno más allá del código hacia la visualización, multimedia, colaboración y más. Además de ejecutar su código, almacena el código y la salida, junto con notas en formato markdown, en un documento editable llamado cuaderno. Cuando lo guarda, este se envía desde su navegador al servidor Jupyter, que lo guarda en disco como un archivo JSON con la extensión .ipynb.

El servidor Jupyter es un centro de comunicación. El navegador, el archivo de cuaderno en disco y el kernel no pueden comunicarse directamente entre sí. Se comunican a través del servidor Jupyter. El servidor Jupyter, no el kernel, es responsable de guardar y cargar cuadernos, por lo que puede editar cuadernos incluso si no tiene el kernel para ese lenguaje; simplemente no podrá ejecutar código. El kernel no sabe nada sobre el documento del cuaderno: solo recibe celdas de código para ejecutar cuando el usuario las ejecuta.

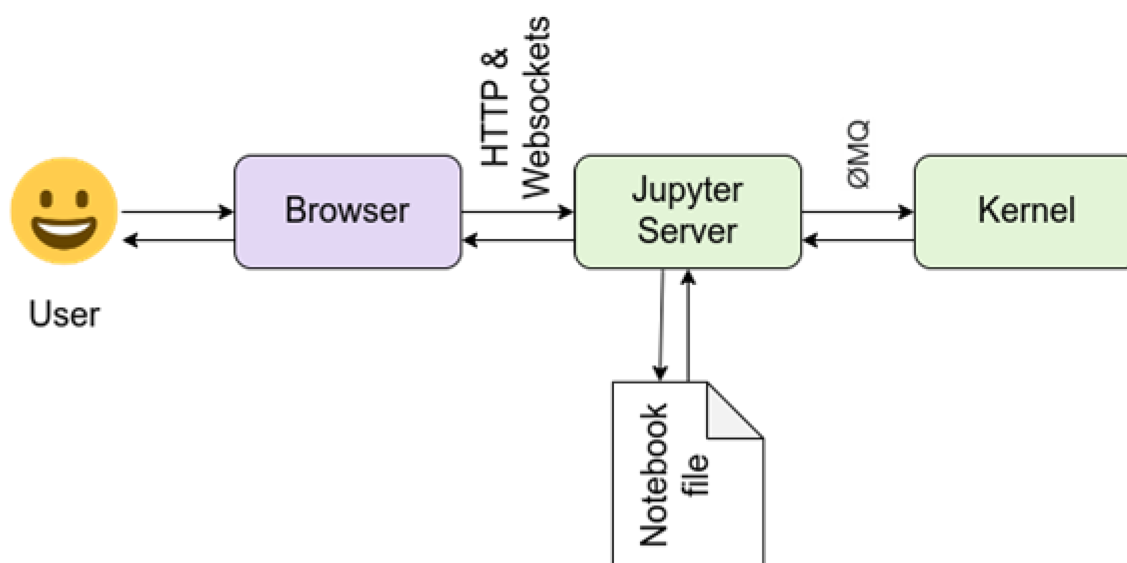

Figura 2.1: Interfaz de Jupyter Notebook.

2.2.2. Arquitectura Lógica

Por otro lado, debemos considerar lo siguiente para nuestra arquitectura lógica:

- **Librerías y Módulos:** Utilizaremos diversas librerías y módulos de Python para realizar tareas específicas, como son *PennyLane*, *NumPy*, *Matplotlib* o *PyTorch*.

Detallaremos más acerca de estas en la sección 5.1.

- **Documentación:** La documentación en forma de celdas de Markdown es crucial en Jupyter Notebook. Esto incluirá explicaciones detalladas, fórmulas matemáticas, imágenes y otros elementos para comunicar eficazmente el conocimiento.

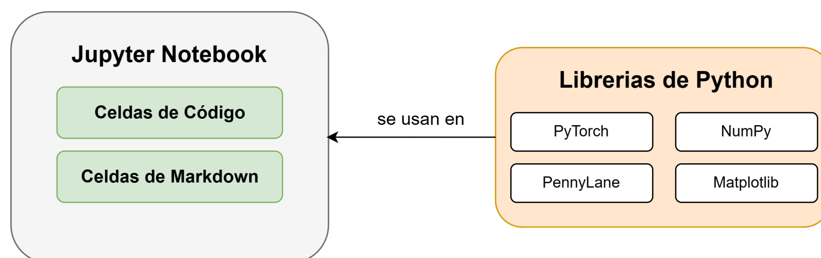

Figura 2.2: Arquitectura lógica.

2.3. Definción de Siglas y Abreviaturas

En esta sección se detallarán varias de las siglas o expresiones usadas a lo largo de este proyecto, con la intención de que se entiendan en todo momento los conceptos expuestos.

- **GQML.** Geometric Quantum Machine Learning.
- **RUP.** Rational Unified Process.
- **PERT.** Program Evaluation and Review Technique.
- $\boldsymbol{G}$**.** Grupo en el sentido matemático.
- $\boldsymbol{\phi}$, $\boldsymbol{R}$ **.** Representación de un grupo.
- $\boldsymbol{GL(n)}$**.** Grupo de matrices invertibles de dimension n.
- $\boldsymbol{S_3}$**.** Grupo simétrico o de permutaciones de 3 elementos.
- **CNN.** Convolutional Neural Networks.
- **GDL.** Geometric Deep Learning.
- **Puertas Cúanticas.** Mostraremos a continuación los símbolos usados para denotar a estas.
    - H. Puerta de Hadamard.
    - X, Y, Z. Puertas de Pauli.

- $R_X(\phi)$, $R_Y(\phi)$, $R_Z(\phi)$. Puertas rotacionales con ángulo ϕ sobre los ejes X, Y, Z, respectivamente.
- C_U. Puerta de control sobre la puerta U.

Capítulo 3

Planificación

3.1. Estimación del Esfuerzo

Aunque se haya considerado un proceso de desarrollo, para estimar el esfuerzo involucrado se requerirá explorar enfoques alternativos a las medologías tradicionales como puede ser la estimación de Puntos de Casos de uso. Esto es debido a que estos métodos se enfocan en la funcionalidad del usuario, mientras que este trabajo se concentra en un estudio teórico.

En este sentido, se ha decidido emplear la técnica de estimación **PERT (Program Evaluation and Review Technique)** [10], que se basa en tres valores distintos para cada tarea del proyecto: la duración más optimista, la duración más probable y la duración más pesimista para su realización. Esta técnica permite abordar la duración de cada tarea como una variable a través de estos tres puntos. El cálculo para la estimación del tiempo esperado se formula de la siguiente manera:

$$PERT = T_{estimado} = \frac{O + 4M + P}{6},$$

donde O es la estimación optimista (caso más favorable), P es la estimación pesimista (peor caso) y, por último, M es la estimación más probable. Podemos destacar las siguientes ventajas:

- Evaluación del tiempo y los recursos necesarios para las actividades.
- Evaluación de la suma de las necesidades de todas las actividades simultáneamente.
- Mejora de la comunicación interdepartamental en el marco de la recogida de datos.

Aunque tambien debemos mencionar algunas de sus inconventientes:

- La subjetividad de la secuencia de actividades debilita la precisión.
- El sesgo de los participantes que envían estimaciones reduce la precisión de los datos.
- Los recursos humanos y el tiempo necesarios para recopilar estimaciones pueden ser prohibitivos desde el punto de vista de los costes.

3.2. Planificación Temporal

Recordemos que el proyecto se estructuraba en cuatro fases: fase de inicio, fase de elaboración, fase de construcción y fase de transición. A pesar de que el proyecto ha sido llevado a cabo por un único individuo, resulta útil clasificar las tareas correspondientes a cada rol (en situaciones en las que una tarea requiere la participación de varios roles, se ha considerado que el tiempo se distribuye de manera equitativa para simplificar). Los roles considerados son:

- **Programador:** Responsable de crear los programas necesarios para el estudio.
- **Investigador:** Posee conocimientos en computación cuántica y matemáticas, y se encarga de redactar los informes y cuadernos.

Es de suma importancia que haya una coordinación y comunicación entre estos dos roles para poder compartir conocimientos y sacar el proyecto adelante. En este caso, una misma persona ha desempeñado los dos roles (luego esto último no supuso un problema) aunque inicialmente no necesariamente tendría por qué ser así. Esto servirá para calcular y estimar los recursos humanos utilizados.

Se ha establecido la jornada de trabajo en este proyecto por días de 4h, de lunes a viernes. Cabe recordar que este TFG lo estoy realizando a la vez que mi trabajo profesional de 40h semanales luego el proceso no será lineal. Sin embargo se sigue esta estructura por comodidad y se tratará de dedicar un total de 20h semanales a este proyecto lo que encajaría perfectamente con la jornada que hemos propuesto. Se han propuesto las siguientes tareas, estableciendo estimaciones optimistas, probables y pesimistas, con el objetivo de aplicar el metodo PERT y estimar el tiempo total de esta tesis.

- **Recolección y análisis de requisitos.**
- **Diseño del sistema.**
  - Diseño de la arquitectura.
  - Selección de herramientas.
- **Estimaciones del proyecto.**
  - Estimaciones temporales.
  - Estimaciones económicas.
- **Estudio teórico.**
  - Computación cuántica.
  - PennyLane.
  - Teoría de representación.

- **Implementación.**

  - Cuaderno Introducción.
    - Conceptos básicos.
    - Circuitos y puertas cuánticas.
    - Sistemas multi-qubits.
    - Teoría de representación.
    - Ejemplos de representaciones.
  - Cuaderno GQML_TicTacToe.
    - Introducción.
    - Juego de las tres en raya.
    - Codificación del modelo.
    - Teoría de representación.
    - Implementación y comparación.

- **Redacción de la memoria**

  - Descripción del Proyecto.
  - Metodologia.
  - Planificación.
  - Conceptos Teóricos.
  - Herramientas de Desarrollo.
  - Análisis.
  - Implementación.
  - Conclusiones.
  - Apéndices.

En la siguiente tabla podemos observar la estimación de los tiempos mediante el método PERT y su comparación con el tiempo real. Este último ha sido de **320h** mientras que el estimado es de algo más de **303h**, luego se ajusta bastante bien a nuestro rango establecido entre la estimación optimista y la pesimista.

	Tiempos (en horas)				
Tarea	**O**	**M**	**P**	**Estimado**	**Real**
Recolección y análisis de requisitos	8	10	12	10,00	10
Diseño del sistema					
Diseño de la arquitectura	7	8	10	8,17	9
Selección de herramientas	5	6	9	6,33	6
Estimaciones del proyecto					
Estimaciones temporales	12	15	18	15,00	17
Estimaciones económicas	8	9	12	9,33	10
Estudio teórico					
Computación cuántica	15	20	22	19,50	22
PennyLane	18	20	25	20,50	23
Teoría de representación	12	15	20	15,33	15
Implementación					
Conceptos básicos	10	12	15	12,17	14
Circuitos y puertas cuánticas	12	15	20	15,33	16
Sistemas multi-qubit	12	15	20	15,33	15
Teoría de representación	8	10	15	10,50	10
Implementación y comparación	20	25	27	24,50	25
Redacción de la memoria					
Descripción del proyecto	5	8	10	7,83	8
Metodología	10	13	15	12,83	12
Planificación	15	20	22	19,50	22
Conceptos teóricos	17	20	25	20,33	21
Herramientas de desarrollo	18	20	25	20,50	24
Análisis	8	10	12	10,00	9
Implementación	18	20	25	20,50	22
Conclusiones	4	5	6	5,00	5
Apéndices	3	5	6	4,83	5
Total (horas)	**245**	**301**	**371**	**303,33**	**320**

Figura 3.1: Estimación temporal vs Tiempo real.

Para finalizar esta sección mostraremos el diagrama de Gannt, el cual ha seguido, en la medida de lo posible, un flujo estructurado y secuencial. un diagrama de Gantt es una herramienta visual que se utiliza para planificar y organizar las distintas etapas y actividades de un proyecto. Este diagrama permite representar las tareas, su duración y su secuencia temporal de manera gráfica, lo que facilita la gestión del tiempo y el seguimiento del progreso del proyecto. Sin embargo, al ser un trabajo realizado por una sola persona, no ofrece mucha información adicional respecto a la optimización de la ruta crítica.

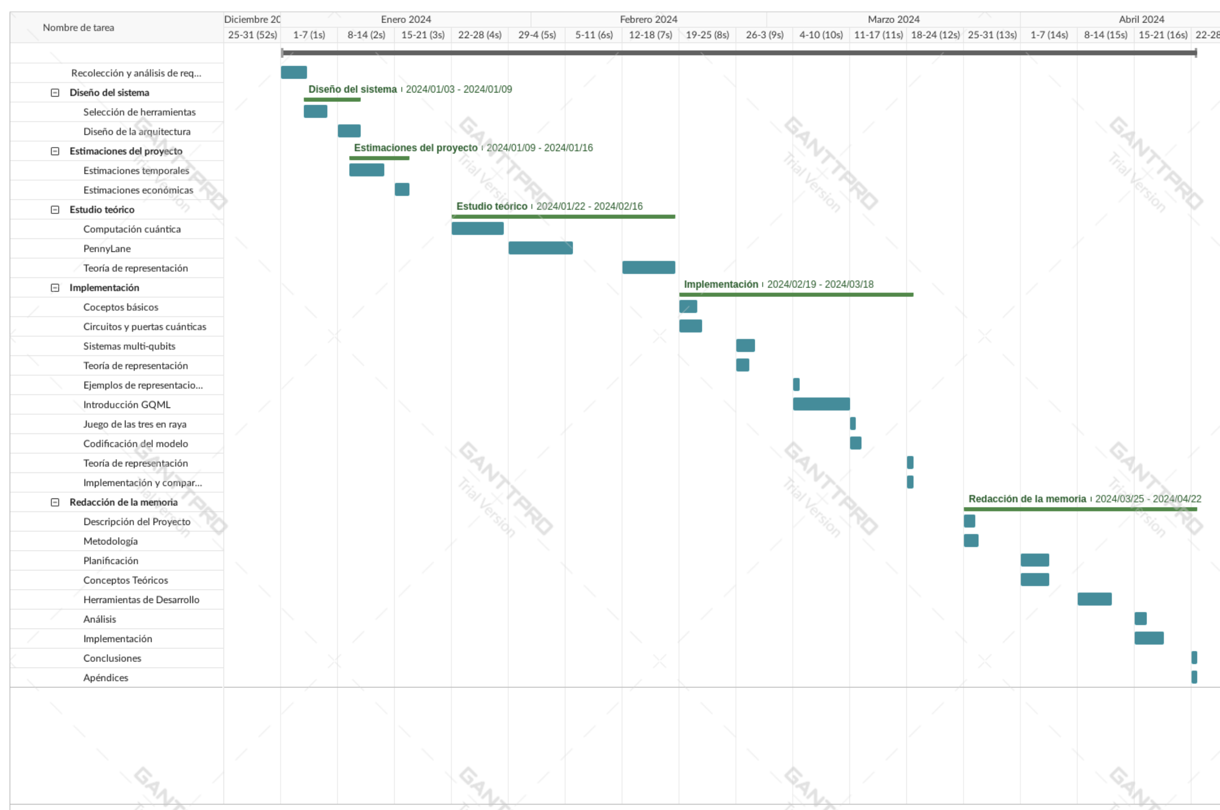

Figura 3.2: Diagrama de Gantt.

3.3. Presupuesto Económico

Para calcular el coste de este proyecto debemos considerar dos apartados principalmente: los recursos técnicos (herramientas software y hardware necesarias para la consecución final del Trabajo de Fin de Grado) y los recurso humanos (el coste de los trabajadores encargados de realizar el trabajo). Este proyecto, como ya hemos mencionado anteriormente, no tiene como finalidad una aplicación software destinada al usuario. Es por ello que este apartado no tendrá la rigurosidad que se exige en un trabajo de esas cárácterísticas, y supondrá simplemente una estimación de los costes de nuestro esfuerzo dedicado, tanto la parte de investigación como la parte de elaboración de este. Además, los recursos de hardware y sofware eran conocidos con antelación y no variables, luego no hay diferencia con lo planificado.

3.3.1. Recursos Humanos

Calcularemos estos costes en función de los roles mostrados en la sección anterior. Consideraremos un único miembro que realiza ambas funciones al mismo tiempo y cuyo salario será la media de ambos roles asumidos para simplificar los cálculos. Debemos considerar también el gasto derivado de inscribir al empleado a la seguridad social, lo

que supone un 30 % del salario bruto [11]. Luego el coste final de ambos roles serán aproximadamente de:

- **Programador:** 13 €/h.
- **Investigador:** 16 €/h.

Por tanto, esto supone de media unos 14,5 €/h, y con la duración de 320h del proyecto los recursos humanos suponen un coste total de 4640 €. Sin embargo, si tomamos nuestro tiempo estimado de 303h obtendríamos un coste de 4393,5 €.

3.3.2. Recursos Técnicos

En esta sección se consideraran las herramientas software y hardware utilizadadas, se considerarán con una vida útil de cinco años, y el gasto se hallará de forma proporcional al tiempo de uso estimado, ya que, por ejemplo el ordenador personal no lo estamos utilizando de manera exclusiva para elaborar este proyecto.

Recurso	Coste (€)	Porcentaje de uso	Total (€)
Ordenador portátil	634,00 €	12%	76,08 €
Ratón	10,00 €	12%	1,20 €
Conexión a Internet	30,00 €	400%	120,00 €
Jupyter Notebook	0,00 €	0%	0,00 €
Overleaf	0,00 €	0%	0,00 €
Draw.io	0,00 €	0%	0,00 €
Microsoft Office	0,00 €	0%	0,00 €
Coste total			197,28 €

Figura 3.3: Recursos técnicos.

En cuanto a los recursos hardware, a parte del ordenador portátil (Asus sVivoBook Intel Core i5 con 8GB de disco duro y una tajeta RAM de 256GB SSD) de se ha considerado también un ratón Logitech y una conexión a Internet estable necesaria para las diversas tareas, cuyo coste fue de unos 30€ al mes. Los precios del portatil (634€) y del ratón (10€) son únicos.

En cuanto a las herramientas software utilizadas, en la mayoría de ellas como son Visual Studio Code, Jupyter Notebook, Overleaf o Draw.io, ofrecen licencias gratuitas para trabajar con ellos. Además, Microsoft Office y el sistema operativo Windows 10 se incluyen dentro de la compra del ordenador, luego el coste reflejado de todo esto será de 0€. Por tanto, los recursos técnicos han supuesto, tanto para el valor estimado como para el real, un valor de 197,28 €.

3.3.3. Coste Final

Una vex que hemos analizado estos costes, se estima que el proyecto final tendrá un coste de desarrollo de **4590,78 €**. Por otro lado, atendiendo a los tiempos reales de elaboración del proyecto hemos tenido un coste total de **4837,28 €**

Capítulo 4

Conceptos Teóricos

4.1. Computación Cuántica

4.1.1. Conceptos Básicos

En la computación cuántica, las unidades fundamentales de información se llaman "**qubits**" en lugar de "bits" como en la computación clásica [2]. A diferencia de los bits, que pueden representar un 0 o un 1, los qubits pueden estar en una superposición de estados, lo que significa que pueden representar simultáneamente 0 y 1. Esto permite una mayor capacidad de cómputo y un procesamiento más rápido en ciertas tareas. Más concretamente, el marco matemático de la computación cuántica es el álgebra lineal. Un qubit se representa como un estado, esto es, un vector columna de dos elementos. Los más básicos son los análogos de los estados "0" y "1" de un bit, los cuales en la notación de Dirac (o también llamada bra-ket) se representan por:

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

Los dos estados $|0\rangle$ y $|1\rangle$ son de gran relevancia ya que forman una base. En álgebra lineal, una base es un conjunto de vectores a partir de los cuales puedes generar todo el espacio como una combinación lineal de estos.

A menudo veremos la notación $|\psi\rangle$, que representa un qubit en un estado arbitrario. Lo que hace que los qubits sean tan especiales es que pueden existir en un estado de **superposición** en algún lugar entre $|0\rangle$ y $|1\rangle$. Matemáticamente, el estado de un qubit en superposición es una combinación lineal de los estados base.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix},$$

donde α y β son números complejos tales que

$$\alpha\alpha^* + \beta\beta^* = 1,$$

y * indica el complejo conjugado. Estos α y β se llaman **amplitudes** o amplitudes probabilísticas y llevan información sobre la fuerza relativa de $|0\rangle$ y $|1\rangle$ en el estado.

Al comienzo de un algoritmo se crean estados cuánticos y se ponen en superposición. Para realizar un cálculo cuántico significativo, necesitaremos una forma de obtener información de los qubits al final de un algoritmo; es decir, una forma de medir los qubits. La medición en computación cuántica es probabilística. Cuando medimos, no podemos ver si un qubit está en superposición, sino que observamos el qubit en estado $|0\rangle$ o en estado $|1\rangle$. Estas probabilidades vienen determinadas por:

$$\text{Prob(medir y observar } |0\rangle) = |\alpha|^2 = \alpha\alpha^*$$
$$\text{Prob(medir y observar } |1\rangle) = |\beta|^2 = \beta\beta^*$$

Medir un qubit una vez nos proporciona un único bit de información: en qué estado hemos observado el qubit. Esto no nos dice mucho sobre el estado, sólo que este estado base concreto está implicado en la superposición. Para obtener una imagen más clara, tenemos que medir muchas veces y observar la distribución de los resultados.

4.1.2. Circuitos y Puertas Cuánticas

Ahora que hemos visto que es un qubit y algunos conceptos previos, veámos como se expresan los cálculos cuánticos. Un **circuito cuántico** son una forma de representar visualmente la secuencia de operaciones que se realizan en los qubits a lo largo de un cálculo. A continuación vemos un ejemplo de un circuito cuántico real:

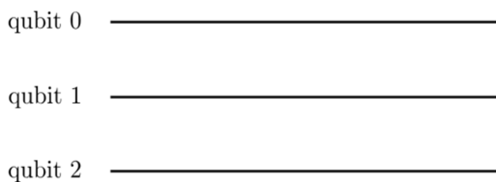

Figura 4.1: Qubits de un circuito.

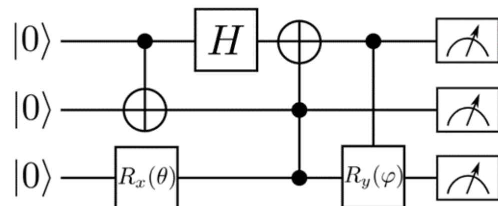

Figura 4.2: Ejemplo de un circuito cuántico.

Un circuito empieza con una colección de **hilos** (en inglés, “wires”) que representan cada uno un qubit. Ahora que sabemos qué son los qubits y cómo expresar cálculos sobre ellos, es hora de hacer una transición importante: ¿qué hacemos exactamente con los qubits? ¿Cuáles son las diferentes operaciones posibles y cómo funcionan? Las operaciones sobre un qubit se llaman **puertas** (en ingles, "gates"). Hay una amplia variedad de ellas, cada una tendrá diferentes efectos sobre los qubits y afectará a uno o a varios cada vez. El paso final de cualquier cálculo cuántico es la medición de uno o varios qubits; al fin y al cabo, tenemos que obtener la respuesta de alguna manera.

Dado un estado inicial arbitrario $|\psi\rangle$, una operación sobre un solo qubit envía

$$|\psi\rangle \rightarrow |\psi'\rangle = U|\psi\rangle,$$

donde $|\psi'\rangle$ es el nuevo estado cuántico. Sin embargo, recordemos que los vectores de estado de los qubits tienen algunas propiedades especiales: en particular, están normalizados, es decir, tienen longitud 1. Por tanto, cualquier matriz que opere con qubits va a requerir una estructura que preserve esta propiedad. Las matrices de este tipo se denominan **matrices unitarias** (aquellas matrices U cuya inversa U^{-1} es igual a su traspuesta conjugada U^*). Ahora supongamos que tenemos una operación U. Mientras conozcamos como actúa sobre los dos estados de la base computacional, podemos usarlo para evaluar su acción sobre cualquier otro estado. Sea $|\psi\rangle = p|0\rangle + q|1\rangle$, entonces aprovecharemos el hecho de que la multiplicación matriz-vector es lineal:

$$U|\psi\rangle = U(p|0\rangle + q|1\rangle) = U(p|0\rangle) + U(q|1\rangle) = p \cdot U|0\rangle + q \cdot U|1\rangle.$$

Mostraremos ahora algunas de las puertas cuánticas principales con las que trabajaremos:

- **Puerta X (NOT):** La puerta cuántica X (o X-Pauli), también conocida como NOT cuántico, invierte el estado de un qubit. Su matriz es: $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, y aplicada a los estados bases invierte los qubits:
$$X|0\rangle = |1\rangle, \quad X|1\rangle = |0\rangle.$$
- **Puerta Y:** La puerta cuántica Y, también conocida como Y-Pauli, es similar a una operación NOT cuántica, pero también aplica una fase compleja.
$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \qquad Y|0\rangle = -i|1\rangle, \quad Y|1\rangle = i|0\rangle.$$
- **Puerta Z:** La puerta cuántica Z (o Z-Pauli) realiza una rotación de π radianes alrededor del eje Z.
$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad Z|0\rangle = |0\rangle, \quad Z|1\rangle = -|1\rangle.$$
- **Puerta H (Hadamard):** La puerta cuántica H (Hadamard) crea superposiciones.
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad H|1\rangle = |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Los estados $|+\rangle$, $|+\rangle$ forman también una base, denomiada la base de Hadamard. Notemos ahora que las puertas X, Y y Z son las tres matrices de Pauli σ_x, σ_y y σ_z y actúan sobre un qubit. Se corresponden, respectivamente, con rotaciones de π radianes sobre los ejes x, y y z de la esfera de Bloch. De forma general, también tenemos las puertas cuánticas de rotación sobre dichos ejes y se definen por:

$$R_x(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}, \quad R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}.$$

Y en la siguiente imagen podemos ver la representación en la esfera de Bloch [7]:

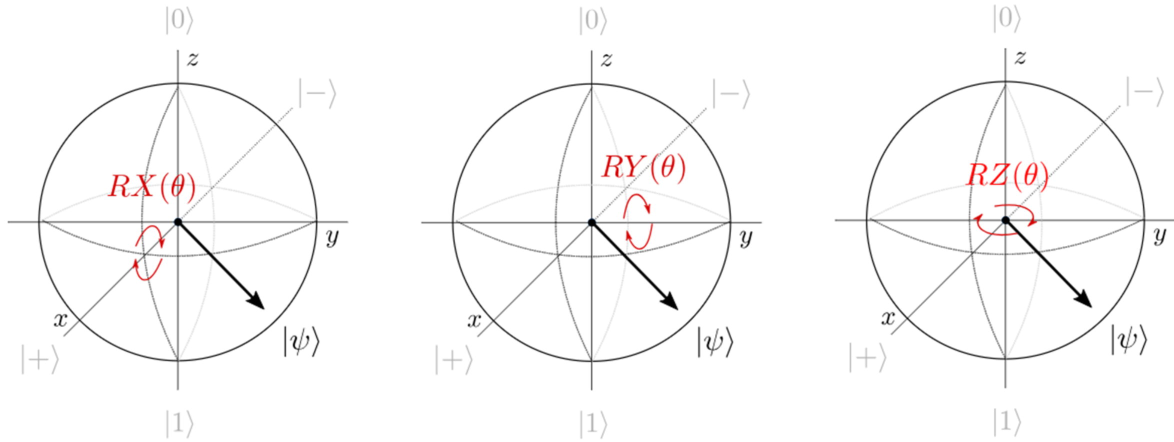

Figura 4.3: Esfera de Bloch con distintas operaciones y estados cuánticos.

4.1.3. Sistemas Multi-Qubit

Como hemos visto, es frecuente trabajar con varios qubits al mismo tiempo. Para ello los qubits se combinan a través del producto tensorial. Por ejemplo una base del espacio generado por dos qubits vendrá dada por los distintos productos tensoriales que se pueden dar entre $|0\rangle$ y $|1\rangle$:

$$\begin{aligned}
\underbrace{|0\rangle \otimes |0\rangle}_{\text{qubit 1} \otimes \text{qubit 0}} = |00\rangle &= \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad |0\rangle \otimes |1\rangle &= |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \\
|1\rangle \otimes |0\rangle = |10\rangle &= \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad |1\rangle \otimes |1\rangle &= |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}.
\end{aligned}$$

qubit 1 $\otimes$ qubit 0

Las puertas cuánticas para más de un qubit se utilizan para realizar operaciones en varios qubits a la vez, lo que permite la creación de entrelazamiento y cómputo cuántico más avanzado. Dos de las puertas cuánticas más importantes para múltiples qubits son la puerta Control ("Controlled Gate") y la puerta Swap. La **puerta Control**, a menudo denotada como **C**, es una operación que controla la aplicación de otra puerta cuántica a un qubit objetivo basado en el estado de un qubit de control. Si el qubit de control está en el estado $|1\rangle$, se aplica la operación; de lo contrario, no se hace nada. De forma general, la puerta Control-U aplicada sobre una puerta U podemos verla como

$$CU = \begin{bmatrix} I_2 & 0 \\ 0 & U \end{bmatrix}$$

$$CU|00\rangle = |00\rangle, \quad CU|01\rangle = |01\rangle, \quad CU|10\rangle = |1\rangle \otimes U|0\rangle, \quad CU|11\rangle = |1\rangle \otimes U|1\rangle.$$

Por otro lado, la puerta $SWAP$ apliacada a dos qubits intercambia el valor de estos y su matriz es:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$SWAP|00\rangle = |00\rangle, \quad SWAP|01\rangle = |10\rangle, \quad SWAP|10\rangle = |01\rangle, \quad SWAP|11\rangle = |11\rangle.$$

4.2. Teoría de Representación

Para trabajar con las simetrías primero debemos adentrarnos en el mundo de la teoria de representación de grupos [6]. La teoría de representación es fundamental en física cuántica, donde los grupos de simetría y sus representaciones son esenciales para describir las simetrías de sistemas físicos y entender sus propiedades. En el contexto cuántico, los elementos de un grupo pueden representarse mediante operadores lineales (matrices) en el espacio de Hilbert, y las representaciones permiten estudiar las propiedades cuánticas bajo transformaciones de simetría.

Veamos primero entonces que entendemos por grupo. Un **grupo** es un conjunto G junto con una operación binaria en G (denotada aquí por $\circ$) que combina dos elementos arbitrarios $a, b \in G$, en un elemento $a \circ b \in G$, tal que se cumplen los siguientes axiomas de grupo:

- **Asociatividad**: Para todo $a, b, c \in G$ se tiene que $(a \circ b) \circ c = a \circ (b \circ c)$.
- **Elemento identidad**: Existe un elemento $e \in G$ tal que, para todo $a \in G$, se tiene que $e \circ a = a \circ e = a$. Este elemento es único y se denomina elemento identidad del grupo.
- **Elemento inverso**: Para cada $a \in G$, existe un elemento $b \in G$ tal que $a \circ b = b \circ a = e$, donde e es el elemento identidad. Para cada a, el elemento b es único: se denomina el inverso de a y se denota por a^{-1}.

Con los grupos definidos, estamos en posición de articular qué es una **representación**: Sea ϕ una aplicación que envía g en el grupo G a una transformación lineal $\phi(g) : V \to V$, para algún espacio vectorial V, que satisface

$$\phi(g_1 g_2) = \phi(g_1) \circ \phi(g_2) \quad \text{para todo } g_1, g_2 \in G.$$

La idea aquí es que al igual que los elementos en un grupo actúan entre sí para alcanzar otros elementos, es decir, $g \circ h = k$, una representación nos envía a una transformación que actúa en un espacio vectorial de manera que $\phi(g) \circ \phi(h) = \phi(k)$. De esta manera, estamos representando la estructura del grupo como un mapa lineal. Para una representación,

nuestra aplicación debe enviarnos al grupo lineal general $GL(n)$ (el espacio de matrices invertibles $n \times n$ con la multiplicación de matrices como la multiplicación de grupo). Nótese cómo esto es tanto un grupo, y por virtud de ser una colección de matrices invertibles, también un conjunto de transformaciones lineales (todas son matrices invertibles que pueden actuar sobre vectores fila). Fundamentalmente, la teoría de representación se basa en la observación común de que el álgebra lineal es fácil y la teoría de grupos es abstracta. Entonces, ¿qué pasa si podemos estudiar grupos a través de mapas lineales?

Ahora bien, debido a la importancia de la unitariedad en la técnica cuántica, nos interesan especialmente las representaciones unitarias: representaciones en las que los mapas lineales son matrices unitarias. Si podemos identificarlas, tendremos una forma de codificar grupos de forma natural en los circuitos cuánticos (que en su mayoría están formados por puertas unitarias).

¿Esto como se relaciona con las simetrías? Una gran cantidad de simetrías se pueden caracterizar como un grupo, donde todos los elementos del grupo dejan algún espacio que estamos considerando sin cambios. Consideremos un ejemplo: las simetrías de una esfera. Cuando pensamos en esta simetría, probablemente pensamos algo como "es lo mismo sin importar cómo lo rotamos o lo volteamos de izquierda a derecha, etc.". Hay esta idea de ser invariante bajo alguna operación. También tenemos la idea de poder deshacer estas acciones: si rotamos en una dirección, podemos rotar de nuevo. Si volteamos la esfera de derecha a izquierda, podemos volcarla de izquierda a derecha para volver al punto de partida (también notamos que todos estos inversos son únicos). Trivialmente, también podemos no hacer nada. ¿Qué estamos describiendo exactamente aquí? Tenemos elementos que corresponden a una acción en una esfera que se pueden invertir y para los cuales existe una identidad. También es trivial en este caso que si consideramos tres operaciones a, b, c del conjunto de rotaciones y reflexiones de la esfera, que si combinamos dos de ellas entonces $a \circ (b \circ c) = (a \circ b) \circ c$. Las operaciones son asociativas, luego estamos defindiendo un grupo.

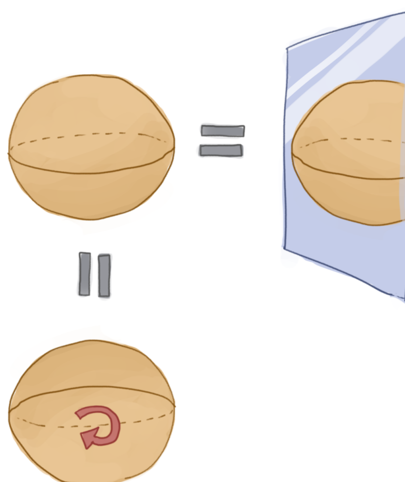

Figura 4.4: Simetrías de una esfera.

Veamos ahora el concepto de equivarianza y por qué es tan importante para nosotros. Sea G nuestro grupo, y V y W, con elementos v y w respectivamente, espacios vectoriales sobre un cuerpo, con una aplicación f entre ellos. Supongamos que tenemos la representación $\varphi : G \to GL(V)$ y $\psi : G \to GL(W)$. Además, sea φ_g la representación de g como aplicación lineal en V y ψ_g de igual manera pero en W. Diremos que f es **equivariante** si

$$f(\varphi_g(v)) = \psi_g(f(v)), \quad \text{para todo } g \in G.$$

La importancia de este tipo de mapeo en el aprendizaje automático radica en que si nuestras capas de redes neuronales son mapas equivariantes, entonces dos entradas relacionadas por alguna simetría intrínseca (tal vez sean reflexiones) preservan esta información en las salidas.

Por ejemplo, consideremos la siguiente figura. Vemos un tablero con una cruz en un cuadrado específico a la izquierda y alguna codificación numérica de esto a la derecha, donde el 1 representa la ubicación de la X en la cuadrícula numérica. Presentamos un mapeo equivariante entre estos dos espacios con respecto a una acción de grupo que es una rotación o un intercambio (aquí una rotación de π). Podemos aplicar una acción de grupo a la cuadrícula original y luego mapearla a la cuadrícula numérica, o podríamos mapearla a la cuadrícula numérica y luego aplicar la acción de grupo. La equivarianza exige que el resultado de cualquiera de estos procedimientos sea el mismo.

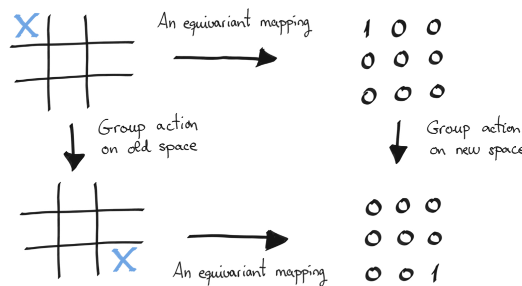

Figura 4.5: Una aplicación equivariante.

Dada la enorme cantidad de datos de entrada necesarios para entrenar una red neuronal, el principio de que se pueden precodificar estructuras de simetría conocidas en la red nos permite aprender mejor y más rápido. De hecho, es la razón del éxito de las redes neuronales convolucionales (Convolutional Neural Networks, CNN) para el análisis de imágenes, donde se sabe que son equivariantes con respecto a las traslaciones. Al centrarnos en las representaciones unitarias (y, por tanto, en los circuitos cuánticos), queremos extender esta idea al aprendizaje automático cuántico.

Capítulo 5

Herramientas de Desarrollo

5.1. Python como Lenguaje de Programación

Python [8] es un lenguaje de programación interpretado, de alto nivel y de propósito general que ha ganado popularidad en el ámbito de la informática y la ciencia de datos debido a su sintaxis clara y legible. Esta sección se enfoca en destacar las características y el uso de Python como herramienta clave en el desarrollo de este proyecto.

La sintaxis simple y legible de Python ha permitido su adopción en una amplia gama de aplicaciones. Su filosofía de diseño hace hincapié en la legibilidad del código y la facilidad de uso, lo que resulta especialmente útil para la implementación de algoritmos complejos y la resolución de problemas. Entre las características principales de Python se encuentran:

- Tipado dinámico: no se requiere declarar el tipo de variable antes de usarla.
- Uso de indentación en lugar de llaves o palabras clave para delimitar bloques de código.
- Amplia biblioteca estándar que cubre áreas como manipulación de archivos, redes, interfaces gráficas, entre otros.
- Interpretado, lo que facilita la escritura y ejecución de programas sin necesidad de una compilación explícita.

La elección de Python ha facilitado significativamente el proceso de desarrollo, permitiendo una implementación rápida y eficiente de las funcionalidades requeridas en este proyecto. En particular nos permitía manejar cómodamente la librería *PennyLane* para trabajar con circuitos cuánticos, como detallaremos a continuación.

Listing 5.1: Ejemplo de una función con Python

```
def on_button_click(self, button):
        if self.finished:
            return
        if button.description == " ":
            button.description = self.current_player
            row, col = self.get_button_position(button)
            self.board[row][col] = self.current_player
            if self.check_winner(self.current_player):
                with self.output:
                    print(f"Player {self.current_player} wins!")
                    self.finished = True
            elif all(self.board[i][j] != " " for i in range(3) for j in
    range(3)):
                with self.output:
                    print("It's a draw!")
                    self.finished = True
            else:
                self.current_player = "O" if self.current_player == "X"
    else "X"
```

5.1.1. PennyLane

PennyLane [3] es un marco de código abierto diseñado para facilitar la implementación y optimización de algoritmos cuánticos híbridos, combinando computación cuántica y clásica. Desarrollado por Xanadu Quantum Technologies, PennyLane se destaca por su integración directa con bibliotecas de aprendizaje profundo como PyTorch y TensorFlow. Al centrarse en algoritmos híbridos, permite la construcción de modelos que aprovechan la capacidad de procesamiento cuántico en conjunción con técnicas de aprendizaje profundo, brindando así una herramienta poderosa para la investigación en informática cuántica y aprendizaje automático cuántico. Además, PennyLane es versátil y es compatible tanto con dispositivos cuánticos reales como con simuladores cuánticos. Esto facilita la transición de algoritmos desde simuladores a hardware cuántico real.

En nuestro caso, para aprender a trabajar con la librería y conocer las bases de la computación cuántica nos hemos servido de los tutoriales de Xanadu [7], así como de la documentación de la página oficial de PennyLane. A continuación mostraremos algunos de sus principales usos y funcionamiento.

Listing 5.2: Ejemplo de uso de PennyLane para trabajar con circuitos cuánticos

```
# Importamos la biblioteca PennyLane como qml
import pennylane as qml

# Creamos un dispositivo cuantico de nueve qubits utilizando PyTorch
    como backend
dev = qml.device("default.qubit.torch", wires=9)

# Definimos un circuito cuantico utilizando la notacion qml.qnode para
    que PennyLane lo pueda evaluar
@qml.qnode(dev)
def circuit():

    # Puertas cuanticas actuando sobre los qubits correspondientes
    qml.PauliX(wires=0)
    qml.Hadamard(wires=1)

    # Devolvemos el valor esperado de la medicion en el eje Z del qubit
    0
    return qml.expval(qml.PauliZ(0))

# Mostramos el circuito
fig, ax = qml.draw_mpl(circuit)()
```

En Pennylane los circuitos cuánticos se definen como unas funciones especiales a través del decorador `qnode`, al cual se le pasa un objeto de tipo `device` para indicar el tipo de "dispositivo" en el que correrá el circuito. Algunas de sus opciones son:

- `name`: El nombre del dispositivo cuántico que se utilizará.
- `wires`: El número de qubits en el dispositivo.
- `shots`: El número de ejecuciones del circuito para obtener estadísticas de medición.
- `analytic`: Un booleano que indica si se deben realizar cálculos analíticos en lugar de simulaciones numéricas. Si es True, se realiza una simulación analítica; si es False, se utiliza una simulación numérica.
- `backend_options`: Opciones adicionales específicas del backend.
- `cache`: Un booleano que indica si se debe usar el caché para almacenar resultados intermedios.

El simulador de qubits predeterminado en PennyLane es `name` = `"default.qubit"`. Realiza simulaciones numéricas de circuitos cuánticos en una máquina clásica. En nuestro caso estamos utilizando `"default.qubit.torch"`, que es similar al predeterminado pero utiliza PyTorch como backend, lo que significa que las operaciones cuánticas se pueden diferenciar automáticamente para su uso en algoritmos de aprendizaje automático basados en PyTorch. En nuestro ejemplo también podemos ver algunos ejemplos de aplicación de

puertas cuánticas como son, entre otras, la puerta de Pauli X (`qml.PauliX(0)`, actuando sobre el qubit 0) o la de Hadamard (`qml.Hadamard(1)`, sobre el qubit 1), donde entre paréntesis se indica el qubit sobre el que esta actuando (o los qubits en caso de ser puertas multiqubit).

Ahora veremos las diversas devoluciones que podemos obtener cuando ejecutamos un circuito cuántico en PennyLane:

- `qml.state()`: Estado cuántico en la base de cálculo.
- `qml.expval(op)`: Devuelve el valor esperado de un observable cuántico especificado `op`. Este resultado es un número real y se utiliza para medir la expectativa de una propiedad cuántica.
- `qml.var(op)`: Devuelve la varianza de un observable cuántico `op`. Similar a `qml.expval`, pero mide la dispersión alrededor del valor esperado.
- `qml.sample(op)`: Realiza una medición estocástica de un observable cuántico `op`. Devuelve una serie de muestras binarias (0 o 1) que representan resultados de mediciones repetidas.
- `qml.probs(wires)`: Devuelve las probabilidades de obtener cada posible resultado en las mediciones. Es útil para observar la distribución de probabilidades de un sistema cuántico.

Nuestra operación `qml.expval(qml.PauliZ(0))` proporciona el valor medio que se esperaría al realizar mediciones repetidas del observable cuántico (en este caso, la proyección en el eje Z del qubit 0).

Para terminar mecionaremos la importancia de la funcion `qml.draw_mpl` que aplicada a nuestro circuito anterior 'circuitños sirve para mostrar los qubits y las operaciones que actúan sobre ellos:

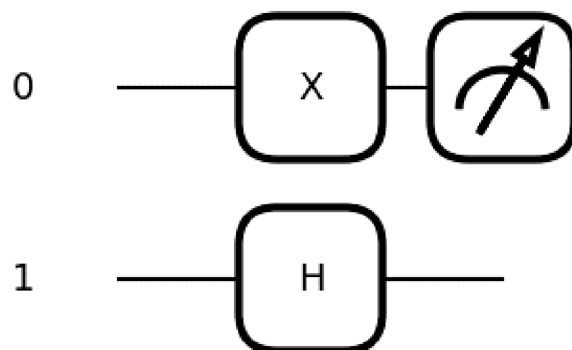

Figura 5.1: Ejemplo de la visualización de un circuito con qml.draw_mpl.

5.1.2. PyTorch

PyTorch [13] es un marco de aprendizaje profundo de código abierto que destaca por su estructura dinámica y flexible. Desarrollado por Facebook, ofrece una interfaz intuitiva para construir y entrenar modelos de aprendizaje profundo. La característica clave es su enfoque dinámico en gráficos computacionales, lo que facilita la experimentación y la inspección del modelo durante la ejecución. PyTorch utiliza tensores para representar datos y se integra bien con bibliotecas populares, convirtiéndolo en una opción preferida para desarrolladores y científicos de datos en el campo del aprendizaje profundo. A continuación veamos algunos de sus usos:

Listing 5.3: Ejemplo de uso de PyTorch

```
# Importamos la biblioteca PyTorch
import torch

# Fijamos las semillas para que sean reproducibles, garantizando asi que
    operaciones aleatorias se generen de la misma manera en cada
    ejectucion
torch.backends.cudnn.deterministic = True
torch.manual_seed(16)

#  creamos un tensor de Pytorch
def create_board():
    return torch.tensor([[0, 0, 0], [0, 0, 0], [0, 0, 0]])

# usamos Pytorch para calcular el error cuadratico medio
def cost_function(params, input, target):
    output = torch.stack([torch.hstack(circuit(x, params)) for x in
    input])
    vec = output - target
    sum_sqr = torch.sum(vec * vec, dim=1)
    return torch.mean(sum_sqr)
```

Al utilizar `torch.zeros`, estamos creando un tensor de PyTorch, que es un tipo específico de estructura de datos que se beneficia de las capacidades de cálculo en paralelo de PyTorch y se puede utilizar en operaciones específicas de tensores. Además, la `torch.stack` combina tensores en una nueva dimensión y `torch.hstack` realiza una concatenación horizontal de tensores.

5.1.3. Matplotlib

Matplotlib [14] es una potente biblioteca de visualización de datos en Python que proporciona herramientas flexibles y de alta calidad para la creación de gráficos estáticos, interactivos y animados. Ya sea para explorar patrones en datos, comunicar hallazgos o generar visualizaciones interactivas, Matplotlib proporciona las herramientas necesarias para transformar datos en información visualmente impactante. Su comunidad activa y documentación exhaustiva hacen de Matplotlib una opción fundamental en el kit de herramientas de cualquier persona involucrada en la visualización de datos con Python.

Listing 5.4: Ejemplo de uso de Matplotlib

```
import pennylane as qml
import numpy as np
import matplotlib.pyplot as plt

dev = qml.device('default.qubit', wires=1)

@qml.qnode(dev)
def apply_rx(theta, state):
    if state == 1:
        qml.PauliX(wires=0)

    qml.RX(theta,0)
    return qml.state()

angles = np.linspace(0, 4*np.pi, 200)
output_states = np.array([apply_rx(t, 0) for t in angles])

plt.figure()
plt.plot(angles, output_states[:, 0].imag, label='|0> Imag')
plt.plot(angles, output_states[:, 0].real, label='|0> Real')
plt.plot(angles, output_states[:, 1].imag, label='|1> Imag')
plt.plot(angles, output_states[:, 1].real, label='|1> Real')
plt.xlabel('Theta')
plt.ylabel('Amplitud')
plt.legend()
plt.grid(True)
plt.show()
```

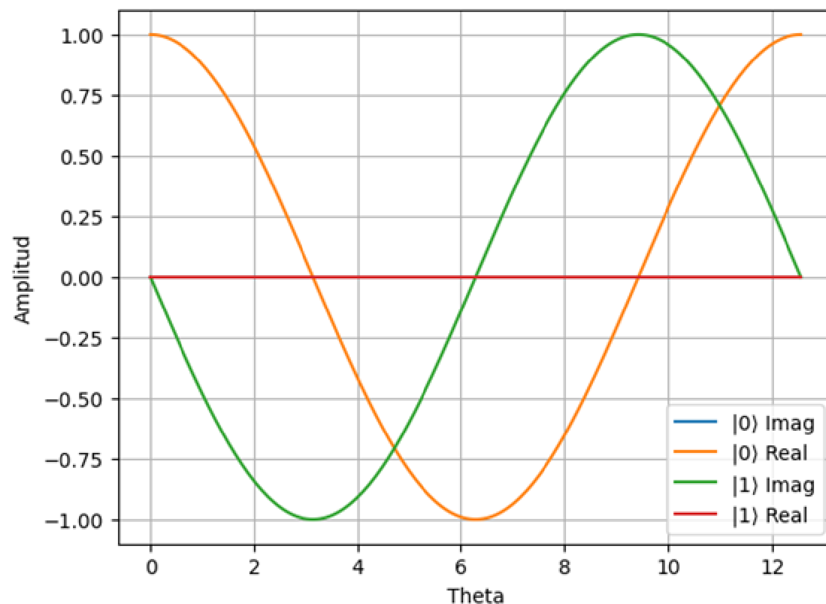

Figura 5.2: Amplitud de los estados base tras aplicar una rotación RX.

5.1.4. NumPy

Para terminar, NumPy [15] es una biblioteca fundamental para la computación científica en Python. Proporciona herramientas para trabajar con matrices y arreglos multidimensionales, junto con funciones matemáticas de alto rendimiento que operan en estos arreglos.

Listing 5.5: Ejemplo de uso de Numpy

```
import numpy as np

# Defimos las matrices R_(123) y R_(12)
w = np.exp(2*np.pi*1j/3)
R_123 = np.array([[w, 0], [0, 1/w]])
R_12 = np.array([[0, 1], [1, 0]])
```

5.2. Jupyter Notebook como IDE

Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos que contienen código, visualizaciones y texto explicativo. En el contexto de este proyecto, se ha utilizado Jupyter Notebook como un entorno interactivo para desarrollar y presentar el análisis, las visualizaciones y los resultados obtenidos. Los cuadernos Jupyter están divididos en celdas las cuales pueden ser de código ejecutable o no (Markdown).

5.2.1. Características de Jupyter Notebook

Algunas de las características clave de Jupyter Notebook incluyen:

- **Interactividad:** Permite ejecutar y modificar el código en celdas individuales, lo que facilita la experimentación y la depuración.
- **Mixtura de Código y Texto:** Ofrece la capacidad de incluir tanto código ejecutable como texto enriquecido (Markdown), lo que facilita la explicación y documentación del proceso de desarrollo.
- **Visualización Integrada:** Permite la integración de gráficos, tablas y visualizaciones dentro del documento, lo que facilita la presentación de resultados.
- **Soporte Multiplataforma:** Es compatible con varios lenguajes de programación, incluyendo Python, R, Julia, entre otros.
- **Facilidad para Compartir:** Permite compartir y colaborar en tiempo real a través de plataformas como GitHub, facilitando la presentación y el intercambio de información.

El proceso de ejecución de celdas Markdown en Jupyter Notebook es una combinación de análisis de Markdown, conversión a HTML y presentación en el entorno de notebook utilizando tecnologías web (como JavaScript y CSS).

```
### Introducción

##### **Conceptos Básicos**
En la computación cuántica, las unidades fundamentales de información se llaman "qubits" en lugar de "bits" como en la computación clásica. A diferencia de los bits, que pueden
representar un 0 o un 1, los qubits pueden estar en una superposición de estados, lo que significa que pueden representar simultáneamente 0 y 1. Esto permite una mayor capacidad
de cómputo y un procesamiento más rápido en ciertas tareas. Más concretamente, el marco matemático de la computación cuántica es el álgebra lineal. Un qubit se representa como
un estado, esto es, un vector columna de dos elementos. Los más básicos son los análogos de los estados "0" y "1" de un bit, los cuales en la notación de Dirac (o también
llamada bra-ket) se representan por:
$$|0 \rangle = \begin{pmatrix}1 \\0 \end{pmatrix}, \qquad |1 \rangle = \begin{pmatrix}0 \\1 \end{pmatrix}.$$

Los dos estados $|0 \rangle$ y $|1 \rangle$ son de gran relevancia ya que forman una base. En álgebra lineal, una base es un conjunto de vectores a partir de los cuales puedes
generar todo el espacio como una combinación lineal de estos.

A menudo veremos la notación $|\psi \rangle$, que representa un qubit en un estado arbitrario. Lo que hace que los qubits sean tan especiales es que pueden existir en un estado
de superposición en algún lugar entre $|0 \rangle$ y $|1 \rangle$. Matemáticamente, el estado de un qubit en superposición es una combinación lineal de los estados base.
$$
|\psi \rangle = \alpha |0 \rangle + \beta |1 \rangle = \begin{pmatrix}\alpha \\ \beta \end{pmatrix},
$$
donde $\alpha$ y $\beta$ son números complejos tales que
$$
|\psi \rangle = \alpha |0 \rangle + \beta |1 \rangle = \begin{pmatrix}\alpha \\ \beta \end{pmatrix},$$ donde $\alpha$ y $\beta$ son números complejos tales que
$$\alpha \alpha^* + \beta \beta^* = 1,$$
```

Figura 5.3: Ejemplo de celda Markdown sin compilar.

Introducción

Conceptos Básicos

En la computación cuántica, las unidades fundamentales de información se llaman "qubits" en lugar de "bits" como en la computación clásica. A diferencia de los bits, que pueden representar un 0 o un 1, los qubits pueden estar en una superposición de estados, lo que significa que pueden representar simultáneamente 0 y 1. Esto permite una mayor capacidad de cómputo y un procesamiento más rápido en ciertas tareas. Más concretamente, el marco matemático de la computación cuántica es el álgebra lineal. Un qubit se representa como un estado, esto es, un vector columna de dos elementos. Los más básicos son los análogos de los estados "0" y "1" de un bit, los cuales en la notación de Dirac (o también llamada bra-ket) se representan por:

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

Los dos estados $|0\rangle$ y $|1\rangle$ son de gran relevancia ya que forman una base. En álgebra lineal, una base es un conjunto de vectores a partir de los cuales puedes generar todo el espacio como una combinación lineal de estos.

A menudo veremos la notación $|\psi\rangle$, que representa un qubit en un estado arbitrario. Lo que hace que los qubits sean tan especiales es que pueden existir en un estado de superposición en algún lugar entre $|0\rangle$ y $|1\rangle$. Matemáticamente, el estado de un qubit en superposición es una combinación lineal de los estados base.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix}\alpha \\ \beta\end{pmatrix},$$

Figura 5.4: Ejemplo de celda Markdown compilada.

El uso de Jupyter Notebook ha enriquecido el proceso de desarrollo, análisis y presentación de resultados en este proyecto, proporcionando un entorno interactivo y versátil para trabajar con código, datos y explicaciones.

5.3. Otras Herramientas Utilizadas

- **Visual Studio Code.** Visual Studio Code (VS Code) es un potente editor de código fuente desarrollado por Microsoft. Es altamente personalizable y cuenta con una amplia gama de extensiones que admiten múltiples lenguajes de programación y tecnologías. Entre sus carcaterístocas destacan: una interfaz amable y ligera, soporte para depuración integrada y una amplia gama de extensiones para personalizar y adaptar el entorno de desarrollo. Nos ha permitido trabajar con Jupyter Notebook y desarrollar nuestro código en Python de manera sencilla.

- **LaTeX y Overleaf.** LaTeX es un sistema de composición de textos utilizado para la creación de documentos científicos y técnicos. Overleaf es una plataforma en línea que permite escribir, editar y colaborar en proyectos LaTeX en la nube, gracias a la cual hemos desarrollado esta memoria

- **Draw.io.** Es una herramienta gratuita y de código abierto que permite crear diagramas de flujo, diagramas UML, organigramas y más directamente en el navegador. Con esta herramienta hemos creado los diversos gráficos necesarios para el trabajo.

- **Microsoft Excel.** Microsoft Excel es una aplicación de hoja de cálculo ampliamente utilizada para la manipulación, análisis y visualización de datos numéricos. Tiene herramientas para cálculos complejos y análisis de datos, dispone también de gráficos dinámicos y personalizables para visualizar información, así como de funciones integradas para trabajar con grandes conjuntos de datos. Excel nos ha permitido diseñar las tablas presentes en este documento.

Capítulo 6

Análisis

En esta sección de análisis, se abordarán dos aspectos fundamentales del proyecto. En primer lugar, se detallará el diseño de los cuadernos que componen el estudio. Esta parte proporcionará una visión general de la arquitectura del proyecto, destacando cómo se organizan y comunican los componentes clave.

En la segunda parte, se examinarán los requisitos del sistema, tanto funcionales como no funcionales. Los requisitos funcionales describen las capacidades específicas que debe tener el sistema, detallando las acciones que debe ser capaz de realizar. Por otro lado, los requisitos no funcionales establecen criterios de calidad que el sistema debe cumplir, como el rendimiento, la seguridad o la usabilidad. Este análisis permitirá comprender las necesidades y expectativas del sistema desde diferentes perspectivas, sentando las bases para su desarrollo y evaluación.

6.1. Diseño

El proyecto se compone, como ya hemos mencionado anteriormente, de dos cuadernos Jupyter:

- **Introducción.ipynb:** En este cuaderno, exploramos los conceptos fundamentales detrás de la computación cuántica y la teoría de representación. Esto nos servirá como base para adentranos en el *Geometric Quantum Machine Learning* del siguiente documento. Más concretamente se tratan las nociones de:
  - Qubit.
  - Circuitos cuánticos.
  - Puertas cuánticas.
  - La esfera de Bloch.
  - Teoría de representación.

  Las explicaciones teóricas se acompañan de pequeños programas que nos permiten interactuar y estudiar los conceptos de forma práctica, facilitando su entendimiento

[7]. Para terminar, se exponen dos ejemplos distintos de representaciones sobre el grupo de simetría S_3, donde podemos encontrar las analogías entre las operaciones sobre estos y las puertas cuánticas.

- **GQML_TicTacToe.ipynb:** Este cuaderno de Jupyter aborda el papel fundamental de la simetría en el aprendizaje automático cuántico, destacando su capacidad para reducir el espacio de parámetros y mejorar la generalización del modelo [6]. Explora, sirviéndonos del juego de las tres en raya (o en inglés *Tic Tac Toe*), cómo la inclusión de simetrías subyacentes en los datos puede optimizar el rendimiento del algoritmo.

  En la primera parte de este se expone el problema y se acompaña de un ejemplo donde poder simular partidas y aplicar simetrías. Después, lo codificamos en nuestro problema de aprendizaje automático geométrico. Para la implementación, se crean las funciones necesarias para generar nuestros conjuntos de datos (tableros con su ganador correspondiente), así como los circuitos cuánticos que nos devuelven los valores esperados, uno de ellos respetando las clases de simetría (`circuit`) y otro que no (`circuit_no_sym`). Finalmente se entrena el modelo con ambos circuitos y se comparan las precisiones para evaluar el rendimiento de estos.

Para implementar ambos documentos se han necesitados diversos paquetes que completan el diagrama del trabajo de fin de grado.

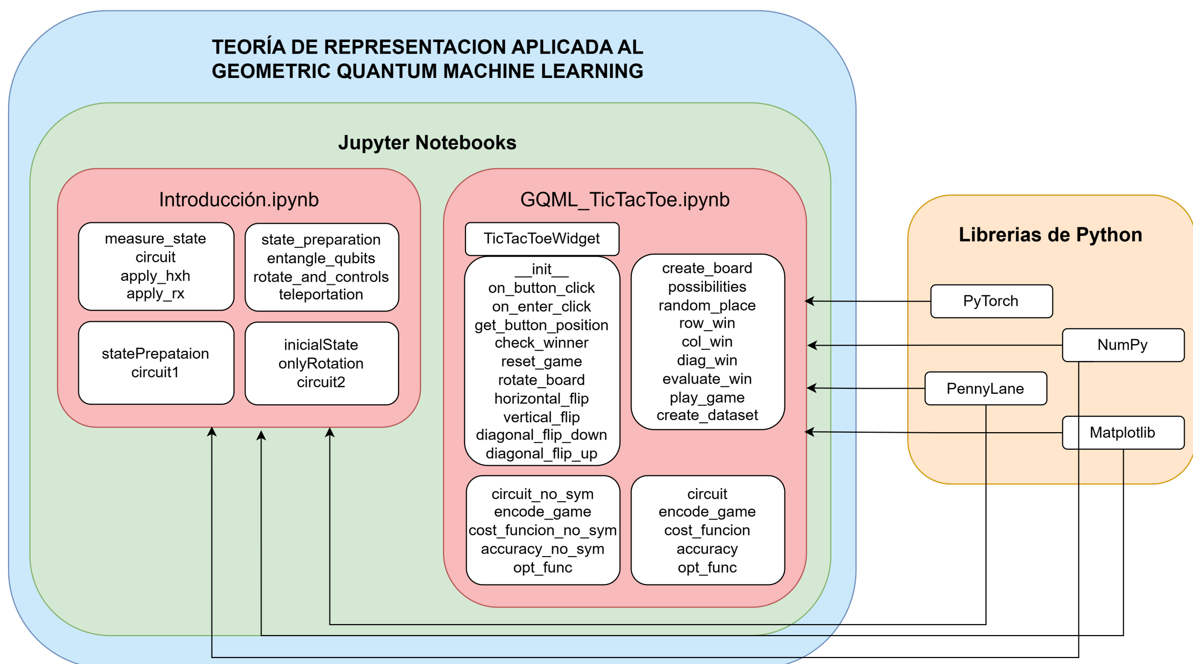

Figura 6.1: Diseño del proyecto.

6.2. Requisitos

6.2.1. Requisitos funcionales

ID	RF.01
Requisito	Ejecución de código.
Descripción	El sistema debe permitir la ejecución de código Python en entornos Jupyter Notebook.
Prioridad	Alta

ID	RF.02
Requisito	Visualización de resultados.
Descripción	Se debe proporcionar la capacidad de visualizar los resultados de los algoritmos cuánticos, como gráficos y tablas, en el cuaderno.
Prioridad	Alta

ID	RF.03
Requisito	Documentacion integrada.
Descripción	Se requiere la capacidad de agregar documentación en formato Markdown para describir el contexto, los objetivos y los métodos utilizados en el estudio.
Prioridad	Alta

ID	RF.04
Requisito	Explicación teórica básica.
Descripción	Es necesario sentar las bases teóricas para comprender el Geometric Quantum Machine Learning (GQML).
Prioridad	Alta

ID	RF.05
Requisito	Visualización de circuitos cuánticos
Descripción	El usuario debe poder visualizar los circuitos cuánicos implementados con la librería PennyLane de Python.
Prioridad	Media

ID	RF.06
Requisito	Código de apoyo a la teoría
Descripción	Se debe proporcionar pequeños programas que faciliten la comprensión de los conceptos teóricos básicos.
Prioridad	Media

ID	RF.07
Requisito	Representaciones de grupos
Descripción	Se debe proporcionar ejemplos de representaciones que sirvan de enlace con la computación cuántica.
Prioridad	Media

ID	RF.08
Requisito	Razonamiento del proceso
Descripción	Se debe explicar el proceso llevado a cabo para implementar el modelo para el juego de las tres en raya.
Prioridad	Alta

ID	RF.09
Requisito	Estructura de la memoria
Descripción	El documento debe ser acorde con la estructura acordada, conteniendo todos los apartados esenciales.
Prioridad	Alta

6.2.2. Requisitos no funcionales

ID	RNF.01
Requisito	Rendimiento.
Descripción	El sistema debe ser capaz de ejecutar algoritmos cuánticos de manera eficiente, minimizando el tiempo de procesamiento y maximizando la capacidad de respuesta.
Prioridad	Alta

ID	RNF.02
Requisito	Escalabilidad.
Descripción	El sistema debe ser escalable y capaz de manejar conjuntos de datos de diferentes tamaños. conjuntos de datos del mundo real.
Prioridad	Media

ID	RNF.03
Requisito	Interfaz de usuario intuitiva.
Descripción	Se espera que la interfaz de usuario sea fácil de usar y comprensible para los usuarios, incluso aquellos que no están familiarizados con el entorno de Jupyter Notebook.
Prioridad	Media

ID	RNF.04
Requisito	Compatibilidad.
Descripción	Se requiere que el sistema sea compatible con diferentes sistemas operativos, navegadores web y dispositivos para garantizar la accesibilidad y la portabilidad.
Prioridad	Alta

ID	RNF.05
Requisito	Claridad en la redacción
Descripción	Es necesario que la redacción del trabajo sea clara y comprensible, facilitando la interpretación de los resultados y la comprensión del análisis realizado en el estudio.
Prioridad	Alta

6.3. Atributos de Calidad del Software

Atributos de calidad [16] son aquellas cualidades no funcionales que se consideran deseables en un sistema de software. Sin embargo, no todos los sistemas de software necesitan considerar todas estas características, ya que algunas pueden ser más relevantes que otras según el contexto. Es importante tener en cuenta que no siempre es posible maximizar todas estas características simultáneamente.

Es importante distinguir entre características de calidad y requisitos, ya que algunas de estas cualidades pueden incorporarse al diseño de manera diferente al análisis, como restricciones de arquitectura o influencias del entorno. En cuanto a nuestro proyecto, destacamos las siguientes:

- **Simplicidad.** Se busca mantener una estructura clara y fácil de entender en los cuadernos, evitando la complejidad innecesaria para facilitar su comprensión y mantenimiento. El trabajo se divide en apartados y el código en celdas *Markdown* relativamente pequeñas lo que facilita el entendimiento.

- **Correctud, consistencia y completitud.** Se garantiza que los cuadernos cumplan con los estándares de calidad establecidos, asegurando la coherencia y la exhaustividad en el análisis de los datos y los resultados obtenidos.

- **Robustez.** Se procura que los cuadernos sean capaces de manejar diferentes escenarios y situaciones de manera estable y confiable, evitando fallos inesperados y garantizando su correcto funcionamiento. Esto es consecuencia de la arquitectura y el funcionamiento de Jupyter.

- **Flexibilidad.** Se diseñan los cuadernos de manera que puedan adaptarse fácilmente a cambios en los datos de entrada o en los requisitos del análisis, permitiendo una rápida reconfiguración y actualización.

- **Eficiencia.** Se optimizan los cuadernos para minimizar el tiempo de ejecución y el consumo de recursos, garantizando un análisis rápido y eficaz de los datos.

- **Usabilidad.** Facilidad con la que los usuarios pueden interactuar con los cuadernos. Es importante diseñarlos de manera intuitiva y amigable para una navegación clara y

comprensión fácil del contenido y el código, con ayuda de comentarios que permitan cumplir con este atributo.

Capítulo 7

Implementación

En este capítulo se expondrán los desarrollos software del proyecto, explicando su elaboración y detallando las partes y funciones que lo componen. Estos se encuentran implementados en los cuadernos `Introducción.ipynb` y `GQML_TicTacToe.ipynb`.

7.1. Ejemplos de Representaciones

7.1.1. Repesentación de permutación de S_3 en tres qubits

El grupo simétrico $G = S_n$ esta definido com el grupo cuyos elementos son todos biyecciones π de un grupo finito en sí mismo. También, si consideramos un conjunto de n elementos $\{1, 2, ..., n\}$, podemos ver S_n como las permutaciones que se pueden realizar en este conjunto [1].

Consideremos el espacio de Hilbert para 3 qubits $V = (\mathbb{C}^2)^{\otimes 3}$ y el grupo simétrico S_3, el cual tiene $3! = 6$ elementos. Como cada permutación se puede obtener como transposición de dos elementos (por ejemplo $(0, 2, 1) = (0, 2)(2, 1)$) tenemos que el conjunto $\{(0, 1), (1, 2), (2, 0)\}$ genera S_3. Una representación $R : S_3 \to GL(V)$ viene dada en la base computacional por:

$$P_\pi \cdot |i_2 i_1 i_0\rangle = |i_{\pi^{-1}(2)} i_{\pi^{-1}(1)} i_{\pi^{-1}(0)}\rangle$$

donde P_π denota una permutación de S_3. Por ejemplo, si ponemos $\pi = (0, 1)$ (usando la notación de ciclo), entonces $P_{(0,1)} \cdot |i_2 i_1 i_0\rangle = |i_2 i_0 i_1\rangle$. Esto significa que $P_{(0,1)} = \text{SWAP}_{0,1}$ (puerta cuántica de intercambio de dos qubits). Por tanto, con la representación de permutación del grupo simétrico podemos permutar qubits (y de forma más general, índices tensoriales).

Listing 7.1: Circuito que aplica la puerta SWAP (permutación)

```
1 import pennylane as qml
2 import numpy as np
3
4 dev = qml.device("default.qubit", wires=3)
5
```

```
@qml.qnode(dev)
def statePreparation():
    qml.BasisState(np.array([0, 0, 1]), wires=[0, 1, 2])  # Inizialiamos el circuito al estado |001>
    return qml.state()

@qml.qnode(dev)
def circuit1():
    qml.BasisState(np.array([0, 0, 1]), wires=[0, 1, 2])
    qml.SWAP(wires=[1, 2])  # Aplicamos la compuerta SWAP al primer y segundo qubit
    return qml.state()

# Mostramos los estados antes y despues de la puerta SWAP
print("Estado inicial antes de aplicar SWAP:")
print(statePreparation())

state_final = circuit1()
print("\nEstado despues de aplicar SWAP:")
print(state_final)

# Mostramos el circuito
fig, ax = qml.draw_mpl(circuit1)()
```

Estado inicial antes de aplicar SWAP:
`[0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]`
Estado después de aplicar SWAP:
`[0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]`

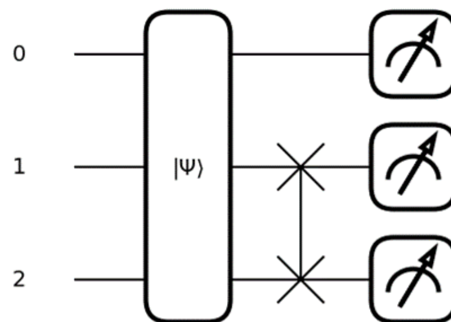

Figura 7.1: Circuito cuántico que inicializa los qubits y aplica una puerta SWAP.

En este ejemplo creamos un circuito cuántico con 3 qubits y lo inicializamos en el estado:

$$|001\rangle = \underset{\text{qubit 2}}{|0\rangle} \otimes \underset{\text{qubit 1}}{|0\rangle} \otimes \underset{\text{qubit 0}}{|1\rangle}$$

Tras aplicar la puerta SWAP a los qubits 0 y 1, esto es, realizar una permutación $P_{(0,1)}$, obtenemos el estado $|010\rangle$. Esto quiere decir que realizar dicha permutación del grupo S_3 puede verse como aplicar una puerta SWAP en un circuito de 3 qubits. Por tanto, como

hemos visto que podemos generar el grupo simétrico S_3 con tres elementos podremos trabajar con las puertas $\text{SWAP}_{0,1}$, $\text{SWAP}_{1,2}$ y $\text{SWAP}_{2,3}$.

7.1.2. Repesentación diédrica de S_3 en un qubit

Ahora veremos una representación muy distinta a la anterior para el mismo grupo, este caso sobre un solo qubit actuando sobre $V = \mathbb{C}^2$. Para ello tendremos en cuenta que el grupo S_3 es también D_3, el grupo de simetrías del triangulo [1]. Los 6 elementos de este grupo consisten en 3 rotaciones y 3 reflexiones.

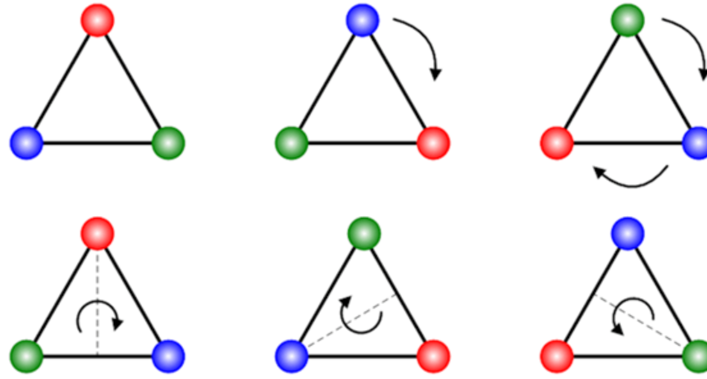

Figura 7.2: Grupo diédrico D3

Este grupo se puede generar con una rotación $(1, 2, 3)$ y una reflexión $(1, 2)$, y vamos a definir una representación que actúa simultáneamente en dos triangulos dados en cada copia de $\mathbb{C}$ como las raíces cúbicas de la unidad $\{1, w, w^{-1}\}$ donde $w = e^{2\pi i/3}$. Estas dos aplicaciones se definen en la base computacional como:

$$R_{(123)} = \begin{pmatrix} w & 0 \\ 0 & w^{-1} \end{pmatrix}, \qquad R_{(12)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Otra forma de ver estas aplicaciones es de la siguiente manera: $R_{(123)}$ rota dos triangulos en la dirección opuesta al ángulo w, por

$$R_{(123)}R_{(123)}|0\rangle = R_{(123)} \begin{pmatrix} w \\ 0 \end{pmatrix} = R_{(123)}(w|0\rangle) = w^2|0\rangle \tag{7.1}$$

$$R_{(123)}R_{(123)}|1\rangle = R_{(123)} \begin{pmatrix} 0 \\ w^{-1} \end{pmatrix} = R_{(123)}(w^{-1}|1\rangle) = w^2|1\rangle, \tag{7.2}$$

y $R_{(12)}$ es la reflexión que cambia los dos estados (y, por tanto, los dos triangulos)

$$R_{(12)}|0\rangle = |1\rangle.$$

$$R_{(12)}|1\rangle = |0\rangle.$$

Listing 7.2: Circuito que aplica una reflexión y rotación

```
import pennylane as qml
import numpy as np

# Defimos las matrices R_(123) y R_(12)
w = np.exp(2*np.pi*1j/3)
R_123 = np.array([[w, 0], [0, 1/w]])
R_12 = np.array([[0, 1], [1, 0]])

dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def inicialState():
    # Estado inicial |0>
    return qml.state()

@qml.qnode(dev)
def onlyRotation():
    qml.QubitUnitary(R_123, wires=[0], id = "123") # rotacion
    return qml.state()

@qml.qnode(dev)
def circuit2():
    qml.QubitUnitary(R_123, wires=[0], id = "123") # rotacion
    qml.QubitUnitary(R_12, wires=[0], id="12") # reflexion
    return qml.state()

state_inicial = inicialState()
print("Estado inicial |0>: \n", state_inicial)

state_rotations = onlyRotation()
print("Estado tras rotacion: \n", state_rotations)

state_final = circuit2()
print("Estado tras rotacion y reflexion: \n", state_final)

# Mostramos el circuito
fig, ax = qml.draw_mpl(circuit2)()
```

Estado inicial $|0\rangle$: `[1.+0.j 0.+0.j]`
Estado tras rotación: `[-0.5+0.8660254j 0. +0.j ]`
Estado tras rotación y reflexión: `[ 0. +0.j -0.5+0.8660254j]`

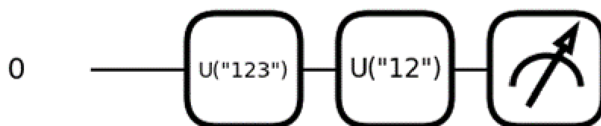

Figura 7.3: Circuito cuántico que aplica dos operaciones a un qubit.

Supongamos que comenzamos con un estado cuántico inicial $|0\rangle$, o equivalentemente, el vector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$. En este ejemplo hemos aplicado sobre este vector las matrices de rotación R_{123} y de reflexión R_{12}, obteniendo como resultado:

$$R_{(12)}R_{(123)}|0\rangle = R_{(12)}R_{(123)}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} w \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ e^{2\pi i/3} \end{pmatrix} = e^{2\pi i/3}|1\rangle.$$

En el contexto de S_3, el grupo de simetría para tres elementos, la permutación específica que hemos generado con $R_{(12)}R_{(123)}$ está relacionada con la acción de intercambiar dos elementos. En particular, la matriz de reflexión $R_{(12)}$ se corresponde con la permutación $(1,2)$ que intercambia los elementos 1 y 2. La matriz de rotación $R_{(123)}$ corresponde a la permutación cíclica (123), que realiza un ciclo de permutación entre los elementos 1, 2 y 3. Al componer estas dos matrices, estamos realizando la acción de primero rotar (cíclicamente) y luego reflejar. La fase compleja ($e^{2\pi i/3}$) es una es una rotación en el espacio complejo como consecuencia de esta composición.

7.2. Geometric Quantum Machine Learning a través del Juego de las Tres en Raya

Ahora, para analizar los beneficios del GQML, nos centraremos en el juego de las tres en raya, también conocido como Tic-Tac-Toe en inglés [6]. Es un juego de estrategia para dos jugadores que se juega en un tablero de 3x3. El objetivo del juego es ser el primero en formar una línea de tres de tus propios símbolos, ya sea horizontal, vertical o diagonalmente, en el tablero. Nuestra tarea de aprendizaje consiste en tomar un conjunto de juegos completados etiquetados con sus resultados y enseñar al algoritmo a identificarlos correctamente.

Este tablero de nueve elementos tiene la simetría del cuadrado, también conocido como grupo diédrico. Esto quiere decir que es simétrico bajo rotaciones de $\frac{\pi}{2}$ radianes (o 90 grados) y giros sobre las líneas de simetría del cuadrado (vertical, horizontal y las dos diagonales). Con código de la clase `TicTacToeWidget` podremos simular partidas de este juego y aplicar las diferentes simetrías bajo las cuales el resultado no se verá afectado.

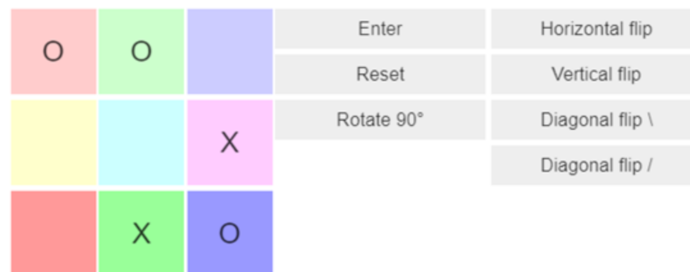

Figura 7.4: Ejemplo de la ejecución de TicTacToeWidget trás insertar algunos símbolos.

Para insertar un símbolo en el tablero podemos pulsar directamente sobre una de las casillas o pulsar el botón “Enter” que lo hará de forma aleatoria. El botón “Reset” limpiará el tablero y el resto de botones simulan cada una de las simetrías mencionadas.

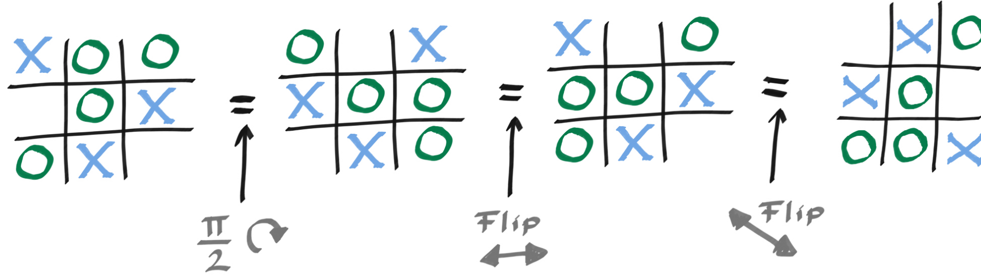

Figura 7.5: Invarianza del resultado bajo simétrías.

Ahora codificaremos esto en nuestro problema de apredizaje automático cuántico. Primero, codifiquemóslo de forma clásica. Consideramos un vector v de nueve elementos, cada uno de los cuales identifica un cuadrado del tablero. Las entradas de este vector pueden ser $+1, 0, -1$, representando una circunferencia, ningún símbolo, o una cruz. Codificamos nuestro tablero con un vector $y = (y_O, y_-, y_X)$ con un $+1$ en la posición correcta y un -1 en las otras. Por ejemplo $(-1, -1, 1)$ representará una X en la posición relevante.

Para crear el modelo cuántico, tomemos nueve qubits que representarán las casillas de nuestro tablero (el orden será de izquierda a derecha y de arriba a abajo). Los inicializaremos todos como $|0\rangle$, lo cual observamos que deja el tablero invariante bajo las simetrías del problema (puedes voltear y rotar todo lo que quieras, seguirán siendo ceros sin importar tu mapeo). Luego buscaremos aplicar rotaciones de un solo qubit $R_x(\theta)$ en qubits individuales, codificando cada una de las posibilidades en las casillas del tablero en un ángulo de $\frac{2\pi}{3}$ entre si. Para nuestras compuertas parametrizadas, tendremos una rotación de un solo qubit $R_x(\theta_1)$ y $R_x(\theta_2)$ en cada punto. Luego usaremos $CR_y(\theta_3)$ para compuertas de entrelazamiento de dos qubits. Esto implica que, para cada codificación, necesitaremos, aproximadamente, 18 parámetros de rotación de qubit único y $\binom{9}{2} = 36$ rotaciones de compuertas de dos qubits. Veamos cómo, utilizando simetrías, podemos reducir esto.

El secreto consistirá en codificar las simetrías en el conjunto de puertas, de modo que los observables que nos interesan respeten intrínsecamente las simetrías. Para ello seleccionaremos las colecciones de puertas que conmutan con las simetrías. En general, podemos utilizar la fórmula del giro para ello:

Sea S el grupo que codifica nuestras simetrías y se U una representación unitaria de S. Entonces,

$$T_U[X] = \frac{1}{|S|} \sum_{s \in S} U(s) X U(s)^*$$

define una proyección sobre el conjunto de operadores que conmutan con todos los elementos de la representación, esto es, $[T_U[X], U(s)] = 0$ para todo X y $s \in S$. En esta

fórmula el operador * indica la adjunta de una matriz y el conmutador de dos operadores A y B se define como $[A, B] = AB - BA$.

El proceso de giro aplicado a una matriz unitaria arbitraria nos dará una nueva unitaria que conmuta con el grupo como requerimos. Un receta para crear una unitaria que conmuta con nuestras simetrías es "girar el generador de la puerta", esto es, pasamos de la puerta $W = \exp(-i\theta H)$ a la puerta $W' = \exp(-i\theta T_U[H])$. Y cuando cada término de la fórmula de giro actúa sobre qubits diferentes, entonces este unitario se simplificaría aún más a:

$$W' = \bigotimes_{s \in S} U(s) \exp(-i\theta) U(s)^*.$$

Así que volvamos ea nuestra elección de puertas: rotaciones sobre un solo qubit $R_x(\theta)$ y $R_y(\theta)$, y puertas $CR_y(\phi)$ de entrelazamiento de dos qubits. ¿Qué conseguiremos girando esto?

En este caso particular, podemos observar la acción de la operación de giro geométricamente, ya que las simetrías involucradas son todas permutaciones. Consideremos la rotación R_x actuando sobre un qubit. Ahora, si este qubit está en la ubicación central en la cuadrícula, entonces podemos girar alrededor de cualquier eje de simetría que deseemos, y esta operación deja el qubit invariante, por lo que hemos identificado inmediatamente una compuerta equivariante. Si el qubit está en las esquinas, entonces el girado enviará esta rotación del qubit a cada una de las otras esquinas. De manera similar, si un qubit está en el borde central, entonces la compuerta de rotación será enviada alrededor de los otros bordes. Así que podemos ver que la operación de giro es una suma sobre todos los posibles resultados de realizar la acción de simetría (la suma sobre las acciones del grupo de simetría). Habiendo hecho esto, podemos ver que para una rotación de un solo qubit, los mapas invariantes son rotaciones en el qubit central, en todas las esquinas y en todos los bordes centrales (cuando sus ángulos de rotación están fijados para ser los mismos).

Como ejemplo, consideremos la siguiente figura, donde tomamos una compuerta R_x en la esquina y luego aplicamos todas las simetrías de un cuadrado. El resultado de este giro nos lleva a tener la misma compuerta en todas las esquinas.

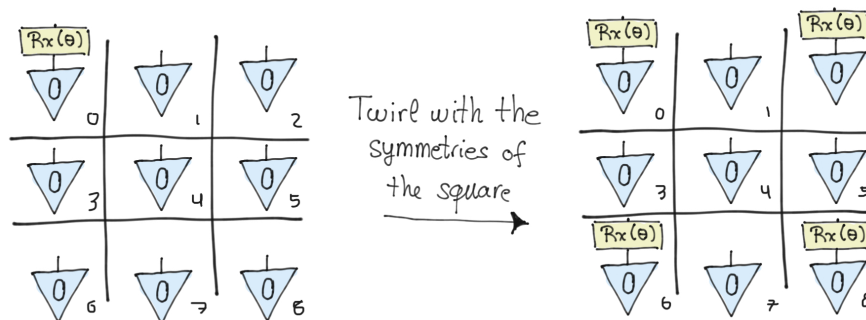

Figura 7.6: Giros con las simetrías del cuadrado.

En el caso de las puertas entrelazadas, la situación es similar. Hay tres clases invariantes, el centro entrelazado con todas las esquinas, con todas las aristas y las aristas emparejadas en un anillo. La predicción se obtiene mediante una codificación binaria midiendo los valores de expectativa de tres observables invariantes:

$$O_{-} = Z_{centro} = Z_4$$

$$O_O = \frac{1}{4} \sum_{i \in aristas} Z_i = \frac{1}{4}[Z_0 + Z_2 + Z_6 + Z_8]$$

$$O_X = \frac{1}{4} \sum_{i \in bordes} Z_i = \frac{1}{4}[Z_1 + Z_3 + Z_5 + Z_7]$$

$$\hat{\mathbf{y}} = (\langle O_O \rangle, \langle O_{-} \rangle, \langle O_X \rangle)$$

Esta es la codificación cuántica de las simetrías en un problema de aprendizaje. Una predicción para un punto de datos dado se obtendrá seleccionando la clase para la cual el valor esperado observado es el más grande. Ahora que tenemos una codificación específica y hemos decidido sobre nuestros observables, necesitamos elegir una función de costo adecuada para optimizar. Utilizaremos una función de pérdida l_2 actuando sobre pares de juegos y etiquetas $D = (g, y)$ donde D es nuestro conjunto de datos.

Primero generemos algunos juegos. Aquí estamos creando un pequeño programa que jugará a las tres en raya contra sí mismo de manera aleatoria. Al completarse, imprimirá el ganador y el tablero ganador, con círculos como $+1$, empate como 0 y cruces como -1. Hay $26,830$ juegos posibles diferentes, pero solo muestrearemos unos pocos cientos. Ahora se detallarán las funciones diseñadas para implementar partidas de las tres en raya y crear los datos de entrenamiento y validación:

- `create_board()`: crea un tablero vacío (una matriz 3x3 con todos los valores 0).
- `possibilities(board)`: comprueba si hay casillas vacías en el tablero `board` y devuelve todas ellas.
- `random_place(board, player)`: selecciona un lugar alazar del tablero para el jugador `player`.
- `row_win(board, player)`: función booleana para comprobar si hay un ganador por tener tres en fila.
- `col_win(board, player)`: función booleana para comprobar si hay un ganador por tener tres en una columna.
- `diag_win(board, player)`: función booleana para comprobar si hay un ganador por tener tres a lo largo de una diagonal.

- `evaluate_game(board)`: comprueba si se han cumplido las condiciones para ganar o si se ha producido un empate.
- `play_game()`: función principal para iniciar el juego.
- `create_dataset(size_for_each_winner)`: crea un conjunto de datos balanceado para entrenamiento y validación.

Listing 7.3: Crear conjuntos de datos

```
NUM_TRAINING = 450
NUM_VALIDATION = 600

with torch.no_grad():
    dataset = create_dataset(NUM_TRAINING // 3)
    dataset_val = create_dataset(NUM_VALIDATION // 3)
```

Tras esto las variables `dataset` y `dataset_val` contrendrán los datos de entrenamiento y validación para nuestro modelo. Sus elementos serán pares donde el primero de ellos corresponderá con el tablero final del juego (en un array (tensor) de 9 elementos), y el segundo indicará el ganador, en caso de haberlo, o si no indicará si hay empate (con un entero -1, 0 o +1).

Ahora procedamos a generar los valores esperados del circuito pertinentes que respeten las clases de simetría que hemos definido sobre las mediciones de uno y dos qubits.

Listing 7.4: Circuito cuántico sin respetar las clases de simetría

```
import pennylane as qml
import matplotlib.pyplot as plt

dev = qml.device("default.qubit.torch", wires=9)

ob_center = qml.PauliZ(4)
ob_corner = (qml.PauliZ(0) + qml.PauliZ(2) + qml.PauliZ(6) + qml.PauliZ
    (8)) * (1 / 4)
ob_edge = (qml.PauliZ(1) + qml.PauliZ(3) + qml.PauliZ(5) + qml.PauliZ(7)
    ) * (1 / 4)

@qml.qnode(dev)
def circuit(x, p):

    qml.RX(x[0], wires=0)
    qml.RX(x[1], wires=1)
    qml.RX(x[2], wires=2)
    qml.RX(x[3], wires=3)
    qml.RX(x[4], wires=4)
    qml.RX(x[5], wires=5)
    qml.RX(x[6], wires=6)
    qml.RX(x[7], wires=7)
    qml.RX(x[8], wires=8)

```

```
23     qml.RX(p[0], wires=4)
24     qml.RY(p[1], wires=4)
25 
26     qml.RX(p[2], wires=0)
27     qml.RX(p[2], wires=2)
28     qml.RX(p[2], wires=6)
29     qml.RX(p[2], wires=8)
30 
31     qml.RY(p[3], wires=0)
32     qml.RY(p[3], wires=2)
33     qml.RY(p[3], wires=6)
34     qml.RY(p[3], wires=8)
35 
36     qml.RX(p[4], wires=1)
37     qml.RX(p[4], wires=3)
38     qml.RX(p[4], wires=5)
39     qml.RX(p[4], wires=7)
40 
41     qml.RY(p[5], wires=1)
42     qml.RY(p[5], wires=3)
43     qml.RY(p[5], wires=5)
44     qml.RY(p[5], wires=7)
45 
46     qml.CRY(p[6], wires=[0, 1])
47     qml.CRY(p[6], wires=[2, 1])
48     qml.CRY(p[6], wires=[2, 5])
49     qml.CRY(p[6], wires=[8, 5])
50     qml.CRY(p[6], wires=[8, 7])
51     qml.CRY(p[6], wires=[6, 7])
52     qml.CRY(p[6], wires=[6, 3])
53     qml.CRY(p[6], wires=[0, 3])
54 
55     qml.CRY(p[7], wires=[4, 0])
56     qml.CRY(p[7], wires=[4, 2])
57     qml.CRY(p[7], wires=[4, 6])
58     qml.CRY(p[7], wires=[4, 8])
59 
60     qml.CRY(p[8], wires=[1, 4])
61     qml.CRY(p[8], wires=[3, 4])
62     qml.CRY(p[8], wires=[5, 4])
63     qml.CRY(p[8], wires=[7, 4])
64 
65     return [qml.expval(ob_center), qml.expval(ob_corner), qml.expval(
       ob_edge)]
66 
67 fig, ax = qml.draw_mpl(circuit)([0] * 9, 18 * [0])
```

Veamos también la misma serie de puertas (la visualización del circuito será idéntica), pero esta vez se aplican independientemente unas de otras, por lo que no estaremos preservando las simetrías con nuestras operaciones. Prácticamente esto también significa más parámetros, ya que antes los grupos de puertas se actualizaban juntos.

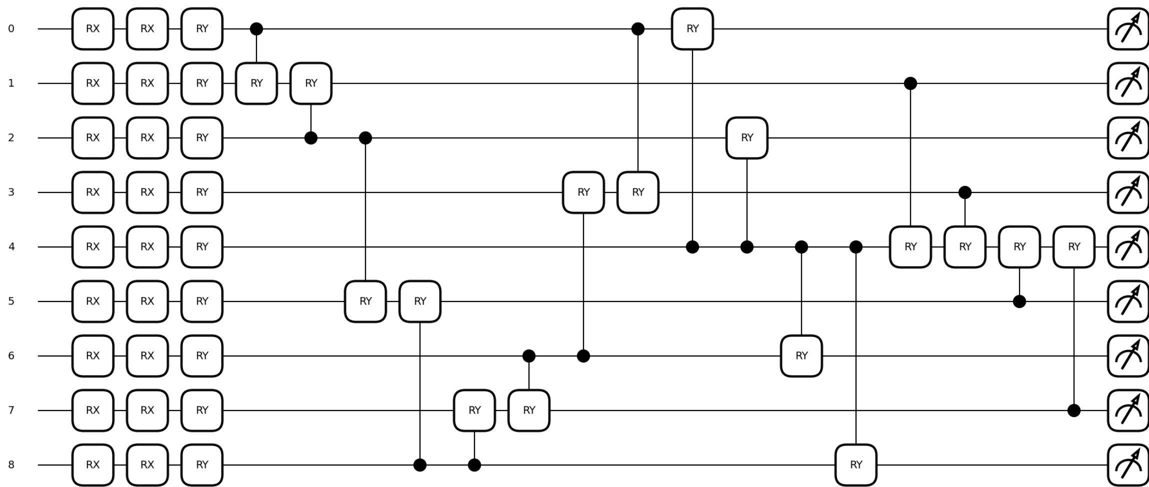

Figura 7.7: Circuito cuántico bajo el que se codifican los datos.

Listing 7.5: Circuito cuántico respetando las clases de simetría

```
1  @qml.qnode(dev)
2  def circuit_no_sym(x, p):
3  
4      qml.RX(x[0], wires=0)
5      qml.RX(x[1], wires=1)
6      qml.RX(x[2], wires=2)
7      qml.RX(x[3], wires=3)
8      qml.RX(x[4], wires=4)
9      qml.RX(x[5], wires=5)
10     qml.RX(x[6], wires=6)
11     qml.RX(x[7], wires=7)
12     qml.RX(x[8], wires=8)
13 
14     qml.RX(p[0], wires=4)
15     qml.RY(p[1], wires=4)
16 
17     qml.RX(p[2], wires=0)
18     qml.RX(p[3], wires=2)
19     qml.RX(p[4], wires=6)
20     qml.RX(p[5], wires=8)
21 
22     qml.RY(p[6], wires=0)
23     qml.RY(p[7], wires=2)
24     qml.RY(p[8], wires=6)
25     qml.RY(p[9], wires=8)
26 
27     qml.RX(p[10], wires=1)
28     qml.RX(p[11], wires=3)
29     qml.RX(p[12], wires=5)
30     qml.RX(p[13], wires=7)
```

```
31
32      qml.RY(p[14], wires=1)
33      qml.RY(p[15], wires=3)
34      qml.RY(p[16], wires=5)
35      qml.RY(p[17], wires=7)
36
37      qml.CRY(p[18], wires=[0, 1])
38      qml.CRY(p[19], wires=[2, 1])
39      qml.CRY(p[20], wires=[2, 5])
40      qml.CRY(p[21], wires=[8, 5])
41      qml.CRY(p[22], wires=[8, 7])
42      qml.CRY(p[23], wires=[6, 7])
43      qml.CRY(p[24], wires=[6, 3])
44      qml.CRY(p[25], wires=[0, 3])
45
46      qml.CRY(p[26], wires=[4, 0])
47      qml.CRY(p[27], wires=[4, 2])
48      qml.CRY(p[28], wires=[4, 6])
49      qml.CRY(p[29], wires=[4, 8])
50
51      qml.CRY(p[30], wires=[1, 4])
52      qml.CRY(p[31], wires=[3, 4])
53      qml.CRY(p[32], wires=[5, 4])
54      qml.CRY(p[33], wires=[7, 4])
55
56      return [qml.expval(ob_center), qml.expval(ob_corner), qml.expval(
    ob_edge)]
```

Obsérvese de nuevo cómo, aunque estos circuitos tienen una forma similar a los anteriores, están parametrizados de forma diferente. Tenemos que alimentar el vector **y** compuesto por el valor esperado de estos tres operadores en la función de pérdida y utilizarlo para actualizar nuestros parámetros. La siguiente función toma un juego (representado por un tablero y el resultado) y lo codifica en un formato que pueda ser procesado por el algoritmo de aprendizaje. Convierte el tablero del juego en un vector x donde cada elemento representa una posición en el tablero, y ajusta los valores para estar en el rango adecuado para el espacio de parámetros cuánticos. Luego, determina el resultado del juego y lo codifica en un vector y.

Listing 7.6: Función para codificar las partidas

```
1  import math
2
3  def encode_game(game):
4      board, res = game
5      x = board * (2 * math.pi) / 3
6      if res == 1:
7          y = [-1, -1, 1] # gana circunferencias
8      elif res == -1:
9          y = [1, -1, -1] # gana cruces
10     else:
11         y = [-1, 1, -1] # empate
```

```
12     return x, y
```

Recordemos que la función de pérdida mide la discrepancia $\mathcal{L}(D)$ entre las predicciones del modelo y los valores reales para un conjunto de datos D. En este caso, utilizaremos el error cuadrático medio l_2:

$$\mathcal{L}(D) = \frac{1}{|D|} \sum_{(g,y)\in D} ||\hat{y}(g) - y||_2^2.$$

Tenemos que definirla y luego podemos empezar nuestra optimización. El objetivo es ajustar los parámetros de manera que minimicen la pérdida y mejoren la capacidad del modelo para predecir correctamente los resultados del juego.

Listing 7.7: Calculo del error cuadrático medio para nuestro problema de clasificación

```
1 def cost_function(params, input, target):
2     output = torch.stack([torch.hstack(circuit(x, params)) for x in
    input])
3     vec = output - target
4     sum_sqr = torch.sum(vec * vec, dim=1)
5     return torch.mean(sum_sqr)
```

En el siguiente código mostramos el proceso de entrenamiento de nuestro circuito de preservación de simetría en los datos. Para ello definimos la función `accuracy` la cual es la función de precisión que evalua el rendimiento del modelo con la que realizaremos la comparación con el otro. `opt_func` será la función de optimización para calcular y actualizar la pérdida, gracias a la cual optimizaremos los parámetros del modelo `params`. Hemos establecido el número de épocas en 15 y obtendremos la pérdida y la precisión en cada una de estas observando progresivamente la mejoría.

Listing 7.8: Entramiento de nuestro circuito respetando simetrías

```
1 from torch import optim
2 import numpy as np
3
4 # inicializacion de parametros del modelo y configuracion del optimizador
5 params = 0.01 * torch.randn(9)
6 params.requires_grad = True
7 opt = optim.Adam([params], lr=1e-2)
8
9 max_epoch = 15
10 max_step = 30
11 batch_size = 10
12
13 encoded_dataset = list(zip(*[encode_game(game) for game in dataset]))
14 encoded_dataset_val = list(zip(*[encode_game(game) for game in dataset_val]))
15
16 # funcion de precision para evaluar el rendimiento del modelo
17 def accuracy(p, x_val, y_val):
18     with torch.no_grad():
19         y_val = torch.tensor(y_val)
20         y_out = torch.stack([torch.hstack(circuit(x, p)) for x in x_val])
```

```
21         acc = torch.sum(torch.argmax(y_out, axis=1) == torch.argmax(y_val, axis
    =1))
22         return acc / len(x_val)
23 
24 # mostramos la precision inicial del modelo antes del entrenamiento
25 print(f"accuracy without training = {accuracy(params, *encoded_dataset_val)}")
26 
27 x_dataset = torch.stack(encoded_dataset[0])
28 y_dataset = torch.tensor(encoded_dataset[1], requires_grad=False)
29 
30 # inicializamos estas listas para almacenar los costos y precision en cada
    epoca
31 saved_costs_sym = []
32 saved_accs_sym = []
33 for epoch in range(max_epoch):
34     rand_idx = torch.randperm(len(x_dataset))
35     # mezclar los datos en cada epoca
36     x_dataset = x_dataset[rand_idx]
37     y_dataset = y_dataset[rand_idx]
38 
39     costs = []
40 
41     # iterar sobre mini lotes de datos y actualizar los parametros del modelo
42     for step in range(max_step):
43         x_batch = x_dataset[step * batch_size : (step + 1) * batch_size]
44         y_batch = y_dataset[step * batch_size : (step + 1) * batch_size]
45 
46         # funcion de optimizacion para calcular y actualizar la perdida
47         def opt_func():
48             opt.zero_grad()
49             loss = cost_function(params, x_batch, y_batch)
50             costs.append(loss.item())
51             loss.backward()
52             return loss
53 
54         # optimizar los parametros del modelo
55         opt.step(opt_func)
56 
57     # calcular el costo promedio en esta epoca
58     cost = np.mean(costs)
59     saved_costs_sym.append(cost)
60 
61     if (epoch + 1) % 1 == 0:
62         # calculamos la precision de la validacion
63         acc_val = accuracy(params, *encoded_dataset_val)
64         saved_accs_sym.append(acc_val)
65 
66         res = [epoch + 1, cost, acc_val]
67         print("Epoch: {:2d} | Loss: {:3f} | Validation accuracy: {:3f}".format
    (*res))
```

```
Accuracy without training = 0.2383333295583725
```

```
Epoch:  1 | Loss: 2.996221 | Validation accuracy: 0.153333
Epoch:  2 | Loss: 2.838961 | Validation accuracy: 0.415000
Epoch:  3 | Loss: 2.721652 | Validation accuracy: 0.535000
Epoch:  4 | Loss: 2.686487 | Validation accuracy: 0.553333
Epoch:  5 | Loss: 2.608699 | Validation accuracy: 0.548333
Epoch:  6 | Loss: 2.648471 | Validation accuracy: 0.591667
Epoch:  7 | Loss: 2.630698 | Validation accuracy: 0.585000
Epoch:  8 | Loss: 2.544674 | Validation accuracy: 0.585000
Epoch:  9 | Loss: 2.630653 | Validation accuracy: 0.570000
Epoch: 10 | Loss: 2.595081 | Validation accuracy: 0.576667
Epoch: 11 | Loss: 2.586225 | Validation accuracy: 0.578333
Epoch: 12 | Loss: 2.600443 | Validation accuracy: 0.578333
Epoch: 13 | Loss: 2.652541 | Validation accuracy: 0.576667
Epoch: 14 | Loss: 2.585265 | Validation accuracy: 0.580000
Epoch: 15 | Loss: 2.598611 | Validation accuracy: 0.580000
```

Por otro lado, procedemos con el circuito que no preserva la simetría. El código que utilizamos es idéntico solo que ahora para obtener la precisión (`accuracy_no_sym`) y el coste (`cost_function_no_sym`) usaremos el otro circuito, que hemos definido como `circuit_no_sym`. La salida en este caso es la siguiente:

```
accuracy without training = 0.22166666388511658
Epoch:  1 | Loss: 3.025290 | Validation accuracy: 0.235000
Epoch:  2 | Loss: 2.918151 | Validation accuracy: 0.280000
Epoch:  3 | Loss: 2.824333 | Validation accuracy: 0.385000
Epoch:  4 | Loss: 2.747958 | Validation accuracy: 0.501667
Epoch:  5 | Loss: 2.693046 | Validation accuracy: 0.466667
Epoch:  6 | Loss: 2.659418 | Validation accuracy: 0.446667
Epoch:  7 | Loss: 2.641402 | Validation accuracy: 0.460000
Epoch:  8 | Loss: 2.626516 | Validation accuracy: 0.481667
Epoch:  9 | Loss: 2.616884 | Validation accuracy: 0.480000
Epoch: 10 | Loss: 2.610851 | Validation accuracy: 0.496667
Epoch: 11 | Loss: 2.606585 | Validation accuracy: 0.508333
Epoch: 12 | Loss: 2.599107 | Validation accuracy: 0.506667
Epoch: 13 | Loss: 2.592962 | Validation accuracy: 0.505000
Epoch: 14 | Loss: 2.589474 | Validation accuracy: 0.515000
Epoch: 15 | Loss: 2.584630 | Validation accuracy: 0.518333
```

Por último, mostremos los resultados en una gráfica y veamos cómo difieren los dos métodos de entrenamiento.

Listing 7.9: Visualizar las precisiones de ambos modelos

```
from matplotlib import pyplot as plt

```

```
plt.title("Validation accuracies")
plt.plot(saved_accs_sym, "b", label="Symmetric")
plt.plot(saved_accs, "g", label="Standard")

plt.ylabel("Validation accuracy (%)")
plt.xlabel("Optimization steps")
plt.legend()
plt.show()
```

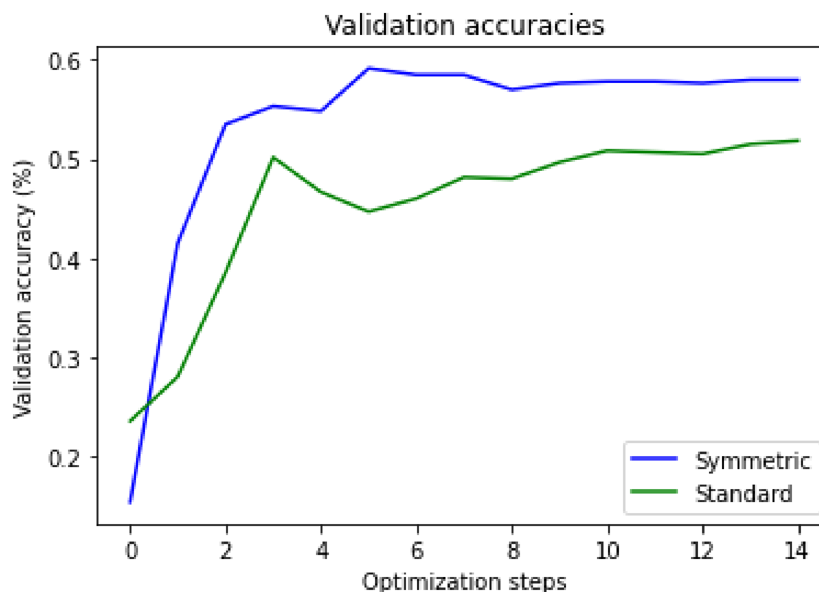

Figura 7.8: Comparación de precisiones de ambos modelos.

Lo que podemos ver entonces es que prestando atención a las simetrías intrínsecas al problema de aprendizaje y reflejándolo en un conjunto de puertas equivariantes hemos conseguido mejorar nuestras precisiones de aprendizaje, a la vez que utilizamos menos parámetros. Aunque el circuito que tiene en cuenta las simetrías supera claramente al ingenuo, cabe destacar que las precisiones de aprendizaje en ambos casos no son ideales, dado que se trata de un juego resuelto. Así que prestar atención a las simetrías ayuda, pero tampoco es una solución mágica.

El uso de simetrías en el aprendizaje automático, tanto cuántico como clásico, es un campo en desarrollo, así que podemos esperar que surjan nuevos resultados en los próximos años.

Capítulo 8

Conclusiones

Como proyecto de fin de grado, este trabajo ha representado para mí un hito significativo en mi trayectoria académica. Durante el desarrollo de esta investigación, he tenido la oportunidad de aplicar y consolidar los conocimientos adquiridos a lo largo de mi formación universitaria en el grado en matemáticas e ingeniería informática de servicios y aplicaciones. Estos me han ayudado a aprender de forma eficiente un nuevo e interesante área como es el de la computación cuántica, la cual ha ampliado mi comprensión de cómo se entrelazan. En particular, la asignatura de *Sistemas Inteligentes* sirvió para asentar los conceptos básicos de aprendidaje automático, que han sido fundamentales para abordar los desafíos planteados en este proyecto.

Además, más allá de su valor académico, este proyecto ha tenido un propósito divulgativo y didáctico. Uno de los objetivos principales ha sido crear un recurso accesible y comprensible para aquellos interesados en adentrarse en el apasionante mundo de la computación cuántica y el Geometric Quantum Machine Learning (GQML). En este sentido, los cuadernos Jupyter desarrollados en este trabajo no solo ofrecen una introducción teórica a los conceptos fundamentales, sino que también proporcionan ejemplos prácticos y experimentos reproducibles que permiten a los lectores explorar y entender mejor estos temas complejos.

Además, a través de la implementación de modelos de juegos de las tres en raya y la aplicación de conceptos de teoría de representación, se ha buscado demostrar de manera práctica la relevancia y aplicabilidad de los principios teóricos discutidos en la computación cuántica. Este enfoque práctico no solo ayuda a consolidar el aprendizaje, sino que también fomenta una comprensión más profunda de los conceptos tratados.

En resumen, este proyecto ha sido una experiencia enriquecedora y gratificante, que me ha permitido combinar y aplicar mis conocimientos en un área emergente y apasionante como es la computación cuántica y el GQML. A través de su enfoque divulgativo y didáctico, espero que este trabajo pueda servir como un recurso útil y accesible para aquellos interesados en explorar este fascinante campo, el cual ofrece un vasto territorio para la investigación futura. Con el continuo progreso en esta área y el desarrollo de

nuevas tecnologías y técnicas, podemos esperar ver avances emocionantes y significativos en los próximos años.

Apéndice A

Manuales de Instalación

En esta sección se detallan los pasos requeridos para reproducir el entorno de desarrollo utilizado en el proyecto, permitiendo así la ejecución del desarrollo final en otro sistema. Los procedimientos se realizaron en un equipo con sistema operativo Windows 10, aunque para otros sistemas operativos los pasos son similares con pequeñas variaciones.

1. **Instalación de Python.**
   a) Descargue la última versión de python desde el sitio web oficial: `https://www.python.org/`.
   b) Durante la instalación, asegúrese de marcar la opción "Agregar Python al PATH" para facilitar el acceso desde la línea de comandos.
2. **Instalación de Visual Studio Code.**
   a) Descargue e instale Visual Studio Code desde el sitio web oficial: `https://code.visualstudio.com/`.
   b) Siga las instrucciones de instalación proporcionadas por el instalador.
3. **Instalación de Jupyter Notebook.**
   a) Abra una terminal o línea de comandos.
   b) Ejecute el siguiente comando para instalar Jupyter Notebook utilizando pip, el gestor de paquetes de Python:

   ```
   pip install jupyterlab
   ```
4. **Configuración de Visual Studio Code para Python.**
   a) Abra Visual Studio Code.
   b) Instale la extensión "Python" desde la tienda de extensiones de Visual Studio Code.

   c) Reinicie Visual Studio Code si es necesario.

5. **Instalación de librerías.** El proyecto, como ya se ha comentado y explicado a lo largo de este trabajo, utiliza varias bibliotecas de Python que deben estar instaladas para ejecutar correctamente el código. Para ello simplemente ejecutaremos el siguiente comando en la terminal:

```
pip install torch pennylane matplotlib numpy
```

6. **Ejecución de Jupyter Notebook.**
   a) Abra una terminal o línea de comandos.
   b) Navegue hasta el directorio donde desea almacenar sus notebooks.
   c) Ejecute el siguiente comando para iniciar el servidor de Jupyter Notebook:

```
jupyter notebook
```

Trás ejecutar este último comando se abrirá automáticamente su navegador web predeterminado con la interfaz de Jupyter Notebook. Ahora simplemente navega hasta la ubicación donde se encuentre almacenado el cuaderno deseado (con la terminación ".ipynb").

Figura A.1: Cabecera de un archivo ejecutado por Jupyter Notebook.

Apéndice B

Contenido Adjunto

En esta sección se detallan los contenidos de los archivos adjuntos que se entregan junto con la memoria del proyecto.

- **Cuadernos de desarrollo.** *Notebooks* de Jupyter en los que se ha implementado el proyecto. Para abrirlos, se hará siguiendo las indicaciones del manual de uso (apéndice A).
  - *Introducción.ipynb*: En este cuaderno, exploramos los conceptos fundamentales detrás de la computación cuántica y la teoría de representación. Esto nos servirá como base para adentranos en el *Geometric Quantum Machine Learning* del siguiente documento.
  - *GQML_ TicTacToe.ipynb*: Este cuaderno de Jupyter aborda el papel fundamental de la simetría en el aprendizaje automático cuántico, destacando su capacidad para reducir el espacio de parámetros y mejorar la generalización del modelo. Explora, sirviéndonos del juego de las tres en raya (o en inglés *Tic Tac Toe*), cómo la inclusión de simetrías subyacentes en los datos puede optimizar el rendimiento del algoritmo.
- **Imagenes:** Carpeta que contiene las imágenes necesarias de los cuadernos.

Bibliografía

[1] Michael Ragone, Paolo Braccia, Quynh T. Nguyen, Louis Schatzki, Patrick J. Coles, Frédéric Sauvage, Martín Larocca y M. Cerezo, *Representation Theory for Geometric Quantum Machine Learning*, arXiv preprint `arXiv:2210.07980[quant-ph]` (2022).

[2] Michael A. Nielsen e Isaac L. Chang, *Quantum Computation and Quantum Information*, Cambridge (2010).

[3] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*, arXiv preprint `arXiv:1811.04968[quant-ph]`, (2022).

[4] Johannes Jakob Meyer, Marian Mularski, Elies Gil-Fuster, Antonio Anna Mele, Francesco Arzani, Alissa Wilms, Jens Eisert *Exploiting symmetry in variational quantum machine learning*, arXiv preprint `arXiv:2205.06217[quant-ph]`, (2022).

[5] Rounak Jha, Debaiudh Das, Avinash Dash, Sandhya Jayaraman, Bikash K. Behera y Prasanta K. Panigrahi, *A Novel Quantum N-Queens Solver Algorithm and its Simulation and Application to Satellite Communication Using IBM Quantum Experience*, arXiv preprint `arXiv:1806.10221[quant-ph]`, (2018).

[6] *Introduction to Geometric Quantum Machine Learning*, Richard East. Accedido el 24/03/2024. URL: `https://pennylane.ai/qml/demos/tutorial_geometric_qml/`

[7] *Xanadu Quantum Codebook*, Xanadu. Accedido el 09/12/2023. URL: `https://codebook.xanadu.ai/`

[8] *Documentanción de Python*, Python Software Foundation. Accedido el 21/01/2024. URL: `https://docs.python.org/3/`

[9] *Modelo RUP*, Software Recopilation. Accedido el el 16/03/2024. URL: `https://softwarerecopilation.wordpress.com/modelo-rup/`

[10] *A Three-Point Estimating Technique: PERT*, Proyect Management Academy. Accedido el 16/03/2024. URL: `https://projectmanagementacademy.net/resources/blog/a-three-point-estimating-technique-pert/`

[11] *¿Cuál es el coste de la empresa al contratar a un trabajador?*, Kenjo Blog. Accedido el 14/03/2024. URL: `https://blog.kenjo.io/es/cual-es-el-coste-de-la-empresa-al-contratar-a-un-trabajador`

[12] *Jupyter Architecture*, Jupyter Team. Accedido el 11/02/2024. URL: https://docs.jupyter.org/en/latest/projects/architecture/content-architecture.html

[13] *PyTorch Documentation*, PyTorch Contributors. Accedido el 27/01/2024. URL: https://pytorch.org/docs/stable/index.html

[14] *Matplotlib Documentation*, The Matplotlib development team. Accedido el 27/01/2024. URL: https://matplotlib.org/stable/index.html

[15] *NumPy Documentation*, NumPy. Accedido el 27/01/2024. URL: https://numpy.org/doc/

[16] *Atributos de calidad*, Uqbar Foundation. Accedido el 12/04/2024. URL: https://wiki.uqbar.org/wiki/articles/atributos-de-calidad.html

[17] *Así es el mapa mundial de la computación cuántica*, BBVA Foundation. Accedido el 15/04/2024. URL: https://www.bbva.com/es/innovacion/asi-es-el-mapa-mundial-de-la-computacion-cuantica/