UNIVERSITAT D'ALACANT
Facultat de Ciències
Facultad de Ciencias

FACULTAD DE CIENCIAS

GRADO EN FÍSICA

TRABAJO FIN DE GRADO

CURSO ACADÉMICO 2021-2022

TÍTULO:

REDES NEURONALES CUÁNTICAS

AUTOR:

JAIME ABELLÁN BERMÚDEZ

Resumen

La naturaleza de este trabajo es puramente bibliográfico, mostrando los recientes avances en computación cuántica, aprendizaje autónomo y la combinación de estas dos disciplinas. Pero para poder comprender la importancia de estos avances debemos retroceder a los fundamentos en los que se sustentan tanto las redes neuronales cómo la computación cuántica. De esta manera este trabajo de dividirá en secciones claramente diferenciadas, pero que se alimentarán unas de las otras.

Primero veremos los fundamentos tanto conceptuales cómo matemáticos de las redes neuronales, se explorará el método de aprendizaje que usan junto con modelos básicos, y otros que son ampliamente usados en la actualidad, y que servirán de pilar para mostrar sus homónimos en el caso cuántico.

Tras esto, se describirá brevemente el formalismo de los ordenadores cuánticos, pasando por expresiones matemáticas básicas, hasta la definición de las puertas. También se hará una breve mención al estado actual de los ordenadores cuánticos.

Luego, nos meteremos de lleno en redes neuronales cuánticas, partiendo de los modelos clásicos, tratando de explicar los circuitos implicados y como cambiaría el proceso de aprendizaje de estas arquitecturas. Junto a esto, tendremos unas pinceladas sobre las aplicaciones que tienen, algunas nuevas para las redes neuronales pero otras ya conocidas.

A continuación se mostrarán estudios recientes que tratan de aunar las capacidades de las redes clásicas con las cuánticas, llegando a modelos muy interesantes que pueden ser aplicados actualmente. De nuevo, se tratará de modificar los tipos de redes clásicos vistos al comienzo del trabajo.

Y por último, se verán las ventajas y desventajas que plantean estos modelos cuánticos, junto con algunas conclusiones finales, y además se mostrarán algunas herramientas que permiten prototipar estos modelos.

Abstract

The nature of this final project is purely bibliographic, showing recent advances in quantum computing, Machine Learning, and the combination of these two disciplines. But in order to understand the importance of these advances, we must go back to the foundations on which both neural networks and quantum computing are based. In this way this work will be divided into clearly differentiated sections, but they will feed from each other.

We will first take a look at both the conceptual and mathematical foundations of neural networks. The way this basic networks learn will be explored alongside complex models that are widely use this days. This will serve as a pillar to show their namesakes in the quantum case.

After this, the formalism of quantum computers will be discuss, going through basic mathematical expression, up to the definition of computation gates. A brief mention of the current state of quantum computers will be done.

Then, we will delve into quantum neural networks, starting from classic models and trying to explain how the circuits involved will try to emulate classic models. We will also explain how the process of learning will change when working with quantum architectures along with exploring some applications these models might have.

Before finishing, we will show recent papers that try to combine the capacities of classical networks with quantum ones, leading to very interesting models that can be applied to current problems. Again, we will try to modify classical models saw at the beginning of the project.

And finally, we will explore the advantages and disadvantages of quantum models, along with some final conclusions. Some tools to implement these models will be provided.

Índice

1. Introducción **6**

2. Redes neuronales clásicas **6**

2.1. Perceptrón 7

2.2. Red Neuronal Artificial 8

2.3. Aprendizaje 8

2.4. Flujo de datos 11

3. Tipos importantes de redes neuronales **13**

3.1. Red Neuronal Convolucional 13

3.1.1. Arquitectura 14

3.1.2. Aprendizaje 15

3.2. Red Neuronal Recurrente 16

3.2.1. Arquitectura 16

3.2.2. Aprendizaje 17

3.2.3. LSTM 18

4. Computación Cuántica **22**

4.1. Bases de la computación cuántica 22

4.2. Puertas Cuánticas 23

4.3. Tipos de ordenadores cuánticos y estado actual 25

4.3.1. Computación cuántica basada en puertas 25

4.3.2. Templado cuántico 25

5. Redes Neuronales Cuánticas **26**

5.1. Modelos de aprendizaje automático cuántico 27

5.2. Red neuronal convolucional cuántica 28

5.3. Red neuronal recurrente cuántica 29

5.4. VQC: Circuitos cuánticos parametrizados 31

5.5. Aplicaciones 33

5.5.1. El problema cuántico de muchos cuerpos 34
5.5.2. Detección de la fase cuántica en 1D mediante QCNN 35
5.5.3. Aplicaciones QRNN 37

6. Redes neuronales híbridas **38**
6.1. Redes híbridas convolucionales 38
6.1.1. Aplicaciones 42
6.1.2. Análisis rayos X 44
6.2. Comentario sobre redes híbridas recurrentes 46

7. Conclusiones **47**
7.1. Ventajas técnicas 47
7.2. Desventajas técnicas 48
7.3. Futuro 48

8. Herramientas para implementar QNN **49**

A. Funciones de Activación **50**

Referencias **52**

1. Introducción

Antes de comenzar con el trabajo en sí mismo, voy a explorar las motivaciones de este. En los últimos años debido al rápido progreso de la tecnología y la aparición de servicios que producen ingentes cantidades de datos ha surgido una necesidad de clasificar y darle uso a estos datos para ofrecer mejores servicios y productos. Estamos hablando del conocido como *Big Data*, siendo esta disciplina la que agrupa numerosos métodos destinados a tratar los datos, incluyendo desde simples métodos que trabajan con datos estructurados, hasta inteligencias artificiales que son capaces de predecir patrones de comportamiento.

Estas inteligencias artificiales se agrupan en la disciplina conocida como Aprendizaje Automático (*Machine Learning*). Dentro de este se incluyen entre otras las redes neuronales, las cuales partiendo de la inmensa cantidad de datos que acumulan empresas e instituciones pueden ser entrenadas para clasificar consumidores, analizar datos no estructurados como es el lenguaje natural, vídeos, fotografías de redes sociales...

Todo esto generó la necesidad de crear algoritmos y modelos matemáticos que sean escalables y permitan trabajar de forma eficiente con estos datos. Así, tras el reciente avance de la computación cuántica, empresas e investigadores vieron una gran oportunidad para mejorar los modelos actuales y generar arquitecturas superiores y más eficientes.

A pesar de que el *software* cuántico está muy desarrollado, queda lejos de las capacidades actuales los ordenadores cuánticos. En la actualidad ya existen algoritmos que, una vez escalen los ordenadores cuánticos, superarán con creces a los modelos clásicos. Y no solo esto, ya que, como consecuencia de la aparición de esta disciplina, se han descubierto aplicaciones muy interesantes a ámbitos como la física.

2. Redes neuronales clásicas

Para poder entender correctamente cómo funcionan los modelos de aprendizaje autónomo, y en concreto las redes neuronales cuánticas, es necesario primero entender lo básico de éstas. Para ello se hablará tanto de las componentes más básicas de las redes neuronales como de los distintos tipos que existen, poniéndolas en conexión con sus aplicaciones principales, además de profundizar en aquellas que pueden tener una mayor importancia al usarlas junto con redes cuánticas. Primero hablaremos de la unidad mínima de estos algoritmos, el llamado **perceptrón**.

2.1. Perceptrón

Como bien se define en [1], el **perceptrón** es la red neuronal más simple que podemos crear, consiste de una única **capa de entrada**, y de un único nodo o **neurona de salida**. Para tratar de entender cómo recibe la información nuestra red hay que pensar en un vector con diferentes componentes, la dimensión de este vector definirá el número de entradas que tenemos. Nuestra salida será generalmente un valor numérico, o varios, según sea el objetivo de nuestro modelo.

Otra forma de entenderla es pensar en el modelo biológico de una neurona, en la que esta tiene las llamadas dendritas que hacen conexión con otras neuronas, recibiendo información, luego estaría el axón, que sería la salida, finalizando en más dendritas, que se conectan mediante sinapsis a otras neuronas.

Pero ¿cómo funcionan realmente estas neuronas?, en general el perceptrón lo que realiza es una suma ponderada de sus entradas, estas ponderaciones son los llamados **pesos**. Estos pesos se pueden cambiar para poder llegar a la solución requerida, optimizando de esta forma la salida de la neurona. Además, esta salida se ve modificada por una **función de activación**, que digamos actúa de tal forma que deja pasar solo algunas señales, o cambia la intensidad de estas señales según su valor, entendiendo las señales como valores numéricos. Así podemos encontrar diversas funciones de activación. En el anexo A se ven las más importantes. De todas formas, cuando se use alguna se mostrará su expresión matemática.

Además, es usual que se incluya una parte invariante en la función activación, siendo este el llamado **umbral** o ***bias***, entonces podemos definir la operación matemática de una neurona de la siguiente forma:

$$y = sign\left(\sum_{j=1}^{N} w_j x_j + b\right) \tag{1}$$

Donde la función *sign* es la función de activación. Podemos ver un esquema del modelo descrito en la siguiente figura.

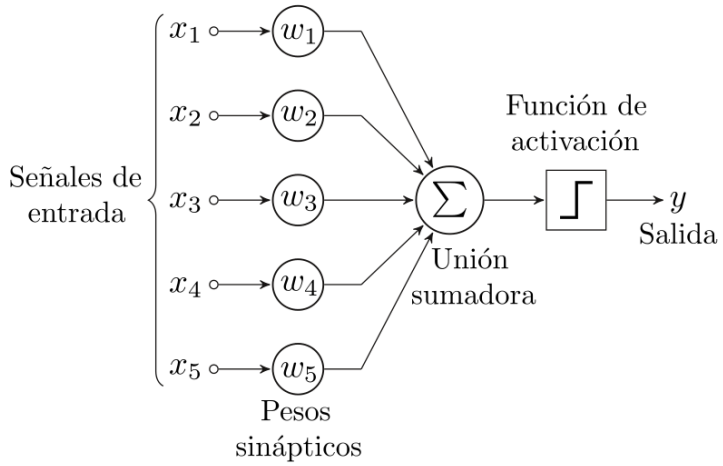


Figura 1: *Esquema Perceptrón*

2.2. Red Neuronal Artificial

Hemos descrito hasta aquí el funcionamiento de una sola neurona con múltiples entradas. La utilidad de este modelo es limitada, pero si combinamos cientos de estos nodos obtenemos las llamadas **redes neuronales**, que combinan nodos de entrada con **capas ocultas**. Dentro de estas capas ocultas los nodos se conectan de diversas formas, cada conexión con su propio peso y cada neurona, si así se requiere, con unas conexiones determinadas. Obviamente todo esto converge a una salida o varias, que nos dará el resultado que queremos tras un proceso de optimización de los pesos. En este ámbito se conocen como **redes neuronales profundas** aquellas que tiene múltiples capas ocultas, produciendo un aprendizaje mucho más complejo conforme aumentan estas capas.

En la **figura 2** tenemos un ejemplo de una red neuronal con diferentes capas, donde el cambio en la opacidad de las conexiones representa el peso de cada conexión. En este caso se han usado conexiones completas o lineales, es decir, las neuronas de cada capa se conectan con todas las neuronas de las capas contiguas. De la misma forma que para el caso anterior cada neurona tiene una función de activación a su salida. Este modelo se suele denominar perceptrón multicapa o **MLP**

2.3. Aprendizaje

Tras esto es natural preguntarnos cómo se obtienen estos pesos y, por tanto, cómo aprende nuestra red. Realmente se trata de una optimización, por lo que debe de haber

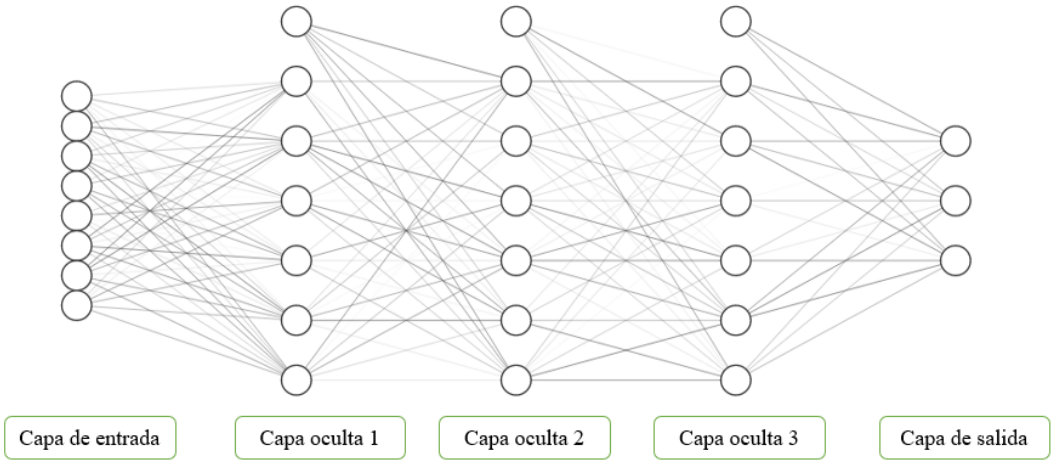


Figura 2: *Ejemplo Red Neuronal*

alguna forma de comparar cómo de precisa es nuestra red para el trabajo que le hemos encomendado, para ello usamos la llamada ***función coste***, o bien, la ***función pérdida***, que nos permiten saber cómo de bueno es nuestro modelo para predecir resultados. Las diferencias entre estas dos últimas es que la función coste es la media de la función pérdida de todos los pesos. Más información sobre estas funciones en [20]. Existen varios tipos diferentes, pero una que es usada ampliamente es la función coste **L2**, o cuadrática media.

$$C = \frac{1}{2N} \sum_{j}^{N} (y_j - \hat{y_j})^2 \tag{2}$$

Con $\hat{\mathbf{y}}$, el valor que devuelve nuestro modelo, mientras que $\mathbf{y}$, es el valor que debería dar. En el proceso de minimización de la función coste iremos modificando los pesos ($\mathbf{w_j}$).

Es importante tener en cuenta que no todos los pesos contribuyen de la misma forma al error generado, por lo que debemos que tener un mecanismo que nos diga cuánto error proporciona cada peso, para así modificar estos valores y dejar sin cambiar aquellos que apenas contribuyen al error. Esta forma de cambiar los pesos se conoce como **retropropagación** (*backpropagation*). Junto a esto vamos a usar el método de **descenso por gradiente**, que esencialmente consiste en usar cálculo diferencial para ver cómo afecta cada peso a la función perdida. Para entender bien esta idea, primero hagamos un ejemplo sencillo con una única neurona, y luego lo generalizaremos para una red completa.

Supongamos que tenemos una neurona con dos entradas, y con una salida, entonces

para obtener los nuevos pesos hacemos la siguiente operación.

$$W^{'} = W - \alpha \cdot \nabla C \rightarrow \nabla C = \sum_{j=1}^{2} \left(\frac{\partial C}{\partial w_j} \right) \tag{3}$$

Siendo $\boldsymbol{\alpha}$, la **tasa de aprendizaje**, que es una cantidad pequeña que determina cuánto cambian los pesos en cada iteración. Esta cantidad es determinada por el modelo que vamos a usar, si es muy grande puede ocurrir que diverja, porque nunca encontraremos un mínimo local, y si es muy pequeña, puede tomar enormes tiempos para converger, si lo hace.

Como se puede deducir de la ecuación (3), estamos trabajando con matrices, de esta forma es fácil generalizar para N pesos, pero si queremos tratar con una red neuronal de verdad tenemos que profundizar un poco más. Primero desarrollemos el gradiente de nuestra función coste o pérdida, según el método usado. En este caso se usará la **función de activación sigmoide**, que se define como:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

Siendo **x** la suma ponderada de las entradas que recibe la neurona. Esta operación de diferenciación habría que repetirla para cada peso, para así obtener los nuevos valores como vimos en la ecuación (3).

Para poder explicar de la mejor forma como es el proceso de aprendizaje con varias capas, voy a utilizar una notación que facilite este proceso junto con la siguiente red de ejemplo:

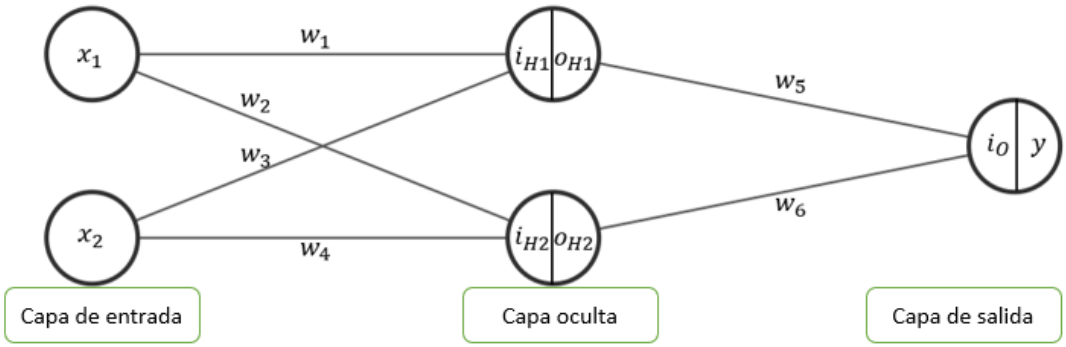


Figura 3: *Red Neuronal con notación.*

Como podemos ver de la figura, las neuronas tienen en su interior dos partes, la primera hace referencia a la entrada de esta, mientras que la segunda mitad se refiere a

la salida tras haber aplicado la función de activación. Luego en las conexiones tenemos los pesos que multiplican las salidas de las neuronas anteriores. Para comenzar a entender este proceso de aprendizaje, empezamos por el final, es decir, por la capa de salida, así tenemos que, usando la regla de la cadena:

$$\frac{\partial C}{\partial w_j} = \frac{\partial i_o}{\partial w_j}\frac{\partial y}{\partial i_o}\frac{\partial C}{\partial y}, \qquad con\, j = 5, 6 \tag{5}$$

El primer término es fácil de calcular, pues $i_o = w_5 o_{H1} + w_6 o_{H2}$. El segundo ya depende de la función de activación, siendo este caso la función sigmoide, que denotaremos como σ. Así la derivada será $\sigma'(\mathbf{i_o}) = \sigma(\mathbf{i_o})(\mathbf{1} - \sigma(\mathbf{i_o}))$. Para el último término también tenemos una solución pues sabemos nuestra función coste, que es la ecuación 2.

Ahora nos queda modificar los pesos anteriores, pero la cosa cambia un poco, veamos:

$$\frac{\partial C}{\partial w_j} = \frac{\partial i_{Hk}}{\partial w_j}\frac{\partial o_{Hk}}{\partial i_{Hk}}\frac{\partial C}{\partial o_{Hk}} \tag{6}$$

Los dos primeros términos se pueden calcular de forma similar al caso anterior, pero para obtener el último aplicamos de nuevo la regla de la cadena:

$$\frac{\partial C}{\partial o_{Hk}} = \frac{\partial i_o}{\partial o_{Hk}}\frac{\partial C}{\partial i_o} \tag{7}$$

Si nos fijamos bien, el primer término es fácil de obtener, mientras que el último fue obtenido anteriormente por lo que nos podemos ahorrar un cálculo.
Con esto ya tenemos todos las operaciones necesarias para poder actualizar los pesos en nuestra red simplificada. Este procedimiento es escalable a redes más complejas, que por tanto tendrán operaciones más extensas y enrevesadas, pero el principio será el mismo.

2.4. Flujo de datos

Hasta ahora se ha presentado el funcionamiento de una red, pero no hemos tocado el tema de qué características deben tener nuestros datos, o cómo debemos pasarlos a nuestro modelo para así tener resultados satisfactorios.

Imaginemos que tenemos una red con **m** entradas, y a su vez tenemos un conjunto de datos con **N** vectores de **m** componentes. Lo que haremos será dividir este conjunto de datos de forma aleatoria, entre un **conjunto de entrenamiento**, y otro de **prueba**. La diferencia entre estos es que uno será usado para entrenar la red y el otro para comprobar cómo de buena es esta, para así evitar sesgos en el entrenamiento y poder ver si funciona correctamente con datos distintos a los utilizados para entrenar.

Pero antes de alimentar la red neuronal con nuestro conjunto de entrenamiento es necesario preparar los datos, para ello tenemos que tener claro qué problema queremos resolver, y en función del mismo darles un significado a los valores numéricos. Por ejemplo, si queremos crear un clasificador de imágenes tenemos que tener una salida por cada clase de imagen, y el valor que saldrá de cada una de estas salidas será un porcentaje, generando de esta manera una distribución de probabilidad entre todas nuestras clases. De la misma forma tenemos que crear un mapeado de los datos de entrada para así poder normalizarlos, evitando en mayor o menor medida los problemas del ***exploding gradient***, y ***vanishing gradient*** [23].

Una vez visto el procesamiento de datos inicial, veamos cómo se produce el flujo de datos en sí mismo. A lo largo del desarrollo de esta disciplina se han creado diversos métodos para generar el flujo de datos con el que alimentamos nuestra red, pero en la actualidad es usual utilizar el llamado **descenso en gradiente estocástico**. Este método consiste en pasar un vector de datos cada vez y, utilizando la función pérdida, modificar los pesos de la forma que vimos en el apartado anterior.

Otros métodos lo que hacen es pasar un ***mini-batch***, o conjunto de vectores, para posteriormente en lugar de usar la función pérdida se optimizan los pesos con la función coste [20]. Esto tiene la ventaja de ser más rápido, pero en general tendremos peores modelos, pues esta forma de aprendizaje es menos fina cambiando los pesos.

Pero no basta con pasar nuestro conjunto una vez, suele ser necesario pasarlos numerosas veces hasta que el error no disminuya más. De forma general podemos resumir este proceso en una serie de pasos:

1. Inicialmente elegimos aleatoriamente los pesos con valores cercanos al 0.
2. Metemos las primeras entradas de nuestro conjunto de datos en la capa de entrada.
3. La red realiza las sumas y van pasando la información hacia delante.
4. Comparamos el resultado previsto con el que tenemos, y obtenemos el error generado.
5. Actualizamos los pesos según cuanto han contribuido individualmente al error, siendo la tasa de aprendizaje quien decide cuánto cambiamos los pesos.
6. Repetimos los pasos del 1 al 5 hasta que lo consideremos oportuno.

3. Tipos importantes de redes neuronales

Hasta ahora hemos visto el funcionamiento básico de las redes neuronales, pero existen diferentes arquitecturas en función de la aplicación que estas tengan. Aquí se mencionarán las más famosas, pero existen gran cantidad de modelos diferentes, de todas formas, puesto que los problemas a resolver no cambian mucho en el sentido de los tipos de datos disponibles, se han diseñado numerosos modelos que son variaciones de las arquitecturas que aquí se van a presentar.

3.1. Red Neuronal Convolucional

En esta sección se tratará de explicar un tipo muy importante de redes neuronales, que son las llamadas **Redes Neuronales Convolucionales** o **CNN**, para ello se usará [2] junto con el curso del MIT [18], donde se puede ampliar información sobre este tema.

El interés de desarrollar este tipo red proviene de la necesidad de analizar tanto imágenes como vídeos. El objetivo central es por tanto extraer información de carácter no estructurada de estos elementos.

Una forma sencilla pero costosa computacionalmente, es coger una foto, convertir los píxeles en un vector unidimensional, y pasarlo por una red. Hacer esto supone un gran costo computacional y se pierde mucha información por lo que se acabaron descartando en pro de esta arquitectura.

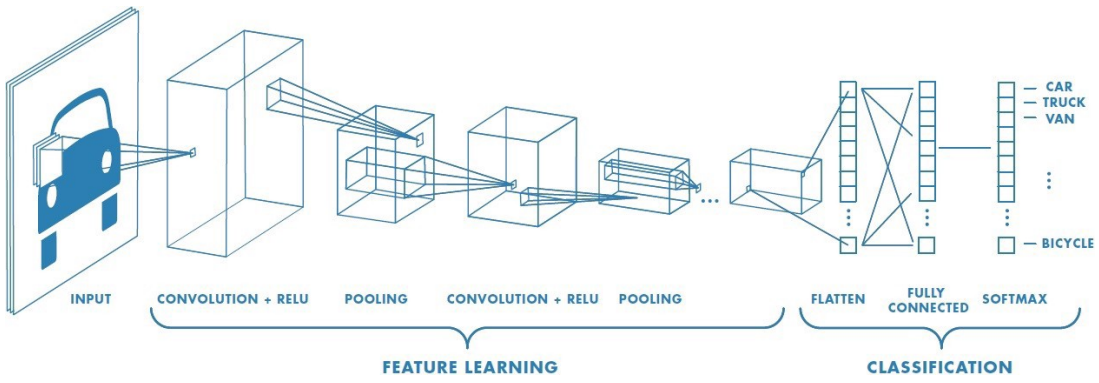


Figura 4: *Red Neuronal convolucional [26].*

De forma general podemos describir la arquitectura de la siguiente forma, primero una **capa convolucional** seguida de una **capa *ReLU***, luego una capa de puesta en común o ***pooling***, y por último una red neuronal como las vistas anteriormente con diferentes

funciones dependiendo el objetivo que tengamos. En la **figura 4** tenemos un esquema de esto.

3.1.1. Arquitectura

Primero expliquemos que hace la **capa convolucional**. Digamos que tenemos una foto de $N \times N$ píxeles, cada píxel con un valor dado, además si es una imagen RGB tendrá 3 **canales**, cada uno para cada color, teniendo entonces una profundidad. Luego tenemos lo que se conoce como **filtro** o **kernel**, con un tamaño digamos de $M \times M$, y podrán ser más de uno, añadiendo una profundidad al resultado de la operación de convolución, además se tiene que cumplir $M \leq N$.

Este filtro lo que hace es seleccionar las características que queremos de nuestra imagen, es decir, cambia la imagen para que queden solo las partes relevantes para la clasificación que queramos hacer. Siendo los valores de estos filtros pesos que tenemos que ir optimizando mediante el aprendizaje. Para extraer la información de las imágenes hacemos una convolución entre estas y los filtros, que no es más que un producto punto.

Matemáticamente esta operación la podemos ver en la siguiente figura, donde el filtro usado lo podemos ver en pequeño en la matriz que representa la imagen.

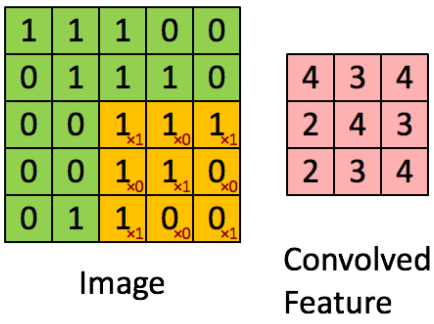


Figura 5: *Operación de convolución.[26]*

Dado que el resultado de la convolución depende del tamaño del filtro, puede ocurrir que, si la imagen de entrada no se divide perfectamente, queden valores fuera de la operación. Para ello se puede aplicar lo que se conoce como ***padding***, esto consiste en añadir una o más dimensiones en los bordes de la imagen, poniendo ceros como nuevos valores, evitando de esta forma que el *padding* afecte al resultado de la operación. Más información sobre cómo se realiza este paso se puede ver en [17].

Esta operación de convolución dará lugar a un **mapa de características**, que digamos contiene la información relevante de la imagen. Pero claro, no vamos a aplicar un solo filtro, si no que tendremos varios. Por tanto, tendremos como resultado un volumen, donde por cada dimensión de profundidad tendremos un mapa de características.

Tras esto tenemos otra capa, que recibe el nombre de **ReLU**, que consiste en usar la función de activación rectificadora. Se utiliza esta función para eliminar los posibles valores negativos de los mapas, y convertirlos en cero, mientras que si son valores positivos se quedan iguales.

Luego tenemos la capa de ***pooling***, esta capa tiene como objetivo principal realizar una reducción de la dimensionalidad, sin perder información de la imagen. Hay varias formas de hacer esto, pero voy a explicar el más común que recibe el nombre de ***max pooling***. Este método consiste en, a partir de nuestros mapas de características, tomar fragmentos de un tamaño determinado, digamos $p \times p$. Luego elegimos el valor más alto contenido en estas matrices fragmento, y a partir de estos conformamos una nueva matriz de menor tamaño, que en gran medida conserva la información importante.

Tras esto tenemos una red neuronal completamente conectada, que tomará como entradas las salidas de las capas previas pero aplanado en un vector unidimensional. Esta capa tendrá un número determinado de capas ocultas y las salidas que requiera el problema.

3.1.2. Aprendizaje

Pero, ¿cómo aprende esta red? De forma general podemos dividir el aprendizaje en dos, primero el relacionado con la convolución, donde se aprende a ver las características importantes para la clasificación que queremos. Y luego tendríamos el aprendizaje de la parte encargada de la clasificación, que funcionaría como una red neuronal normal.

La forma de optimizar la parte convolucional consiste en ir modificando los valores de los diferentes filtros, para así ir aprendiendo que características de las imágenes son importantes para la clasificación que estamos haciendo. Cuanto más complejas sean las imágenes por analizar, más capas convolucionales necesitaremos, o incluso tendremos que modificar el tamaño de los filtros usados.

El aprendizaje de la red neuronal es como el ya visto para las redes normales, usando como función coste la llamada ***Softmax***, o ***Cross Entropy***, estas funciones se pueden ver mejor en [20], pero ambas generan una distribución de probabilidades de diferentes formas.

Como ya he dicho, estas redes son ampliamente usadas en el reconocimiento de imá-

genes, que va desde una simple clasificación de animales, hasta reconocimiento facial y técnicas biométricas. También tienen un gran potencial en el sector médico para reconocer tumores o diferentes enfermedades analizando los resultados visuales de diferentes pruebas. El potencial que tienen es inmenso.

3.2. Red Neuronal Recurrente

Vamos a ver otro tipo importante de red neuronal, y estas son las llamadas **Redes Neuronales Recurrentes**, o **RNN**. Su importancia reside en la capacidad que tienen de analizar datos en forma de secuencias temporales, como puede ser procesamiento de lenguaje natural, o por ejemplo series temporales de datos financieros para predecir el futuro de un activo en concreto. Cabe mencionar que por la naturaleza de estas redes no se puede paralelizar su aprendizaje, por lo que tienen un coste computacional bastante elevado. Existen algunas variedades de estas que pueden llegar a ser paralelizadas, pero eso es un tema aparte. Esta sección se basará en el curso del MIT [19], y en [3].

3.2.1. Arquitectura

La arquitectura de estos modelos es algo diferente a lo visto anteriormente, pues el flujo de información es secuencial, de esta manera pongamos el ejemplo de procesamiento de lenguaje. Antes de poder introducir una frase en nuestro modelo, debemos tener alguna forma de convertir nuestra oración en algo entendible por los ordenadores como podría ser un vector por cada palabra. Esto recibe el nombre **codificación**.

En la **figura 6** tomada de [3], podemos ver una arquitectura simple de este modelo.

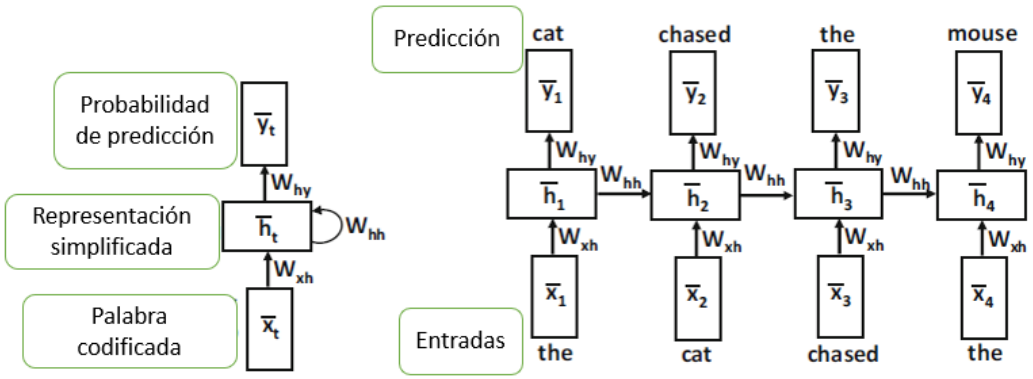


Figura 6: *Arquitectura red neuronal recurrente.*

Como se puede ver tenemos una frase de entrada, a la cual le falta la última palabra, y el objetivo de esta red es predecir cual será la siguiente palabra en la frase conociendo las anteriores. La caja con notación $\mathbf{h_i}$ representa las capas ocultas de nuestra red, que en este modelo recibe el nombre de **estado**. Las salidas serán probabilidades de tener una determinada palabra, eligiendo aquella con mayor probabilidad. Para el caso de una serie temporal, en lugar de hablar de palabras estaríamos hablando de valores numéricos.

Podemos entender este modelo como una función matemática, donde $\mathbf{h_t}$ es el estado a un tiempo dado, así podemos ir actualizándolo: $\bar{\mathbf{h}}_\mathbf{t} = \mathbf{f}\left(\bar{\mathbf{h}}_{\mathbf{t-1}}, \bar{\mathbf{x}}_\mathbf{t}\right)$. Esta función estará determinada por matrices de pesos que se usan en cada paso de tiempo, junto con la función de activación que se use. Podemos expresar de forma matemática estas evoluciones temporales usando la **tangente hiperbólica** como función de activación:

$$\bar{h}_t = tanh\left(W_{xh}\bar{x}_t + W_{hh}\bar{h}_{t-1}\right); \qquad \bar{y}_t = W_{hy}\bar{h}_t \tag{8}$$

3.2.2. Aprendizaje

Como siempre, tenemos que ver como aprende este modelo, para ello debemos recordar la función *softmax* [20].

$$L = -\sum_{t=1}^{T} log\left(\hat{p}_t^{j_t}\right) \tag{9}$$

$$\frac{\partial L}{\partial \hat{y}_t^k} = \hat{p}_t^k - I\left(k, j_t\right) \tag{10}$$

Con $\hat{p}_t^k$ un peso de nuestra matriz compartida, y $\mathbf{I}\left(\mathbf{k}, \mathbf{j_t}\right)$ siendo una función similar a la **delta de kronecker**, siendo igual a 1 si ambos argumentos son iguales, y cero en cualquier otro caso.
Con esto ya podemos hacer *backpropagation* como vimos en apartados anteriores, pero encontramos un problema, y es que los pesos son compartidos entre cada paso temporal, por lo que no podemos actualizarlos en cada paso de forma independiente, sino que debemos modificar el algoritmo de *backpropagation* para que contemple matrices de pesos compartidas.

Para ello debemos suponer que los parámetros en las diferentes capas son independientes, de esta forma introducimos unas variables temporales a partir de nuestras matrices de pesos. Así, hacemos *backpropagation* sobre estas variables suponiendo que son independientes unas de otras. Una vez hecho esto sumamos las contribuciones de las diferentes variables temporales para crear una actualización de pesos unificadas para nuestras ma-

trices de pesos originales. Esta forma de hacerlo recibe el nombre de ***backpropagation through time***, o ***BPTT***.

Al igual que con otros tipos de redes, podemos aumentar el número de capas ocultas, en este caso estados, para hacerla más profunda y permitir realizar predicciones más complejas, pero este modelo presenta numerosos problemas a la hora de hacer que aprendan. El más evidente es que no todas las frases tienen la misma longitud, por lo que la capa de entradas no puede ser fija en tamaño, aunque una forma de solucionar esto es usar una codificación que convierta todas las frases a un vector del mismo tamaño. Esto igualmente trae numerosos problemas, como es la pérdida de información al realizar la conversión.

Otro problema es que, debido a la profundidad de estas redes, existe el peligro de caer de nuevo en *exploding gradient* o *vanishing gradient* [23]. Hay numerosas soluciones para esto, pero cada una con sus propios problemas, por ejemplo, la normalización es una buena solución, pero limita enormemente el potencial de este modelo.

3.2.3. LSTM

Existe una variedad de *RNN* que permite arreglar el problema mencionado anteriormente en mayor o menor medida, y consiste en cambiar la ecuación de recurrencia que actualiza los estados usando la conocida como ***Long Short-Term Memory***, o ***LSTM***. El problema de antes tiene su origen en que las *RNN* no tienen la suficiente memoria, es decir, para frases muy largas es necesario conocer el principio de dicha oración, por lo que para tener en cuenta esas palabras podemos caer en los problemas de gradiente. Aquí es donde entra la *LSTM*.

Este modelo introduce una **puerta** o ***gate***, anteriormente también podemos decir que teníamos puertas, pero estas se reducían a la función de activación. En la siguiente figura podemos ver el esquema de estas nuevas puertas. Para entender bien este sistema debemos hacer un símil entre las flechas y tuberías, donde los círculos con una x o $+$ con como llaves de paso, que harán la operación indicada dependiendo del valor que presenta la flecha que apunta a estas.

Así, la diferencia más importante con las *RNN* normales es que tenemos el llamado ***cell state*** o **celda estado**, que dependiendo de si se abren o cierran las llaves se verá modificado o no. Esta forma de plantear las puertas nos permite tener una memoria a mayor largo plazo mitigando el problema del *vanishing gradient*. En la siguiente figura tenemos un esquema más detallado de su funcionamiento, con C_t la celda estado. Aquí la unión de dos tuberías implica la concatenación de dos vectores. La separación de una

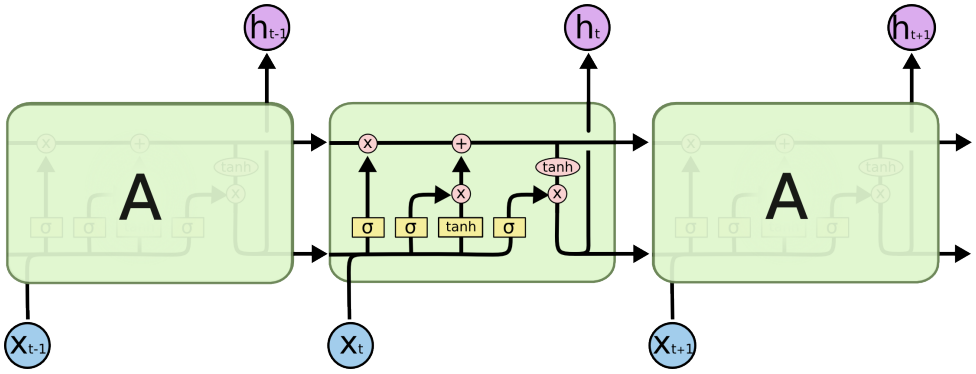


Figura 7: *Puerta LSTM [22].*

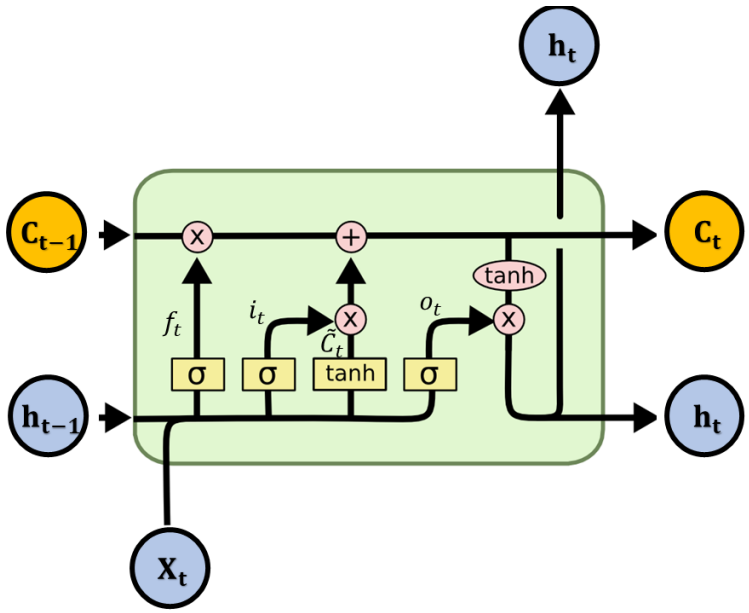


Figura 8: *LSTM en detalle.*

tubería en dos implica que se ha copiado el valor. También hay que mencionar que en los rectángulos amarillos tenemos una red neuronal del tamaño que sea cada vector, siendo su función aplicar las funciones de activación indicadas. Así tendremos un valor de salida entre 0 y 1 en el caso de la función sigmoide, que modulará los valores de la celda estado.

De forma general podemos dividir esta puerta en tres partes, la primera recibe el nombre de ***forget gate layer***, que se puede ver resaltada en la **figura 9**. Esta tiene el objetivo de decidir qué información vamos a eliminar de la celda estado teniendo en cuenta la entrada $\mathbf{x_t}$ y el estado $\mathbf{h_{t-1}}$. Podemos representar esta primera parte con una función matemática:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{11}$$

Siendo $\mathbf{W_f}$ una matriz de pesos que actúa sobre la concatenación de $\mathbf{h_{t-1}}$ y $\mathbf{x_t}$. Si esta operación nos da un 1, dejamos toda la información pasar, pero si tenemos un 0, no dejará nada pasar.

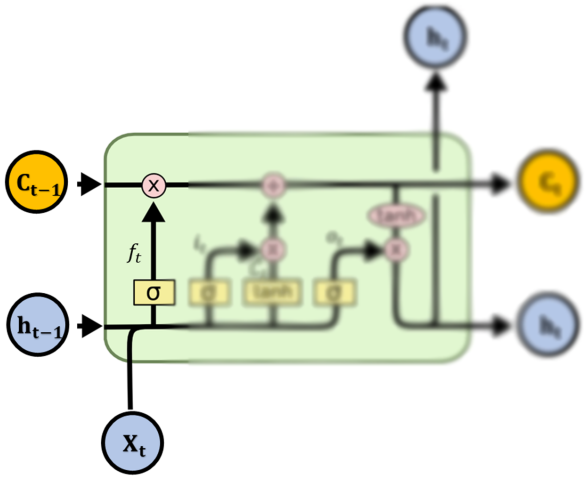


Figura 9: *Forget gate layer.*

El siguiente paso tenemos lo que se llama ***input gate layer*** que se ve resaltado en la **figura 10a**, que se compone de dos partes, siendo la primera la que decide que valores vamos a cambiar, y como vamos a modificar estos valores. La capa con función sigmoide recibe el nombre $\mathbf{i_t}$, luego tendremos una capa con una ***tanh***, que genera nuevos valores que podrán ser añadidos a la celda estado. Estos nuevos valores se indican como $\mathbf{\tilde{C}_t}$. Las expresiones son las siguientes:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{12}$$

$$\tilde{C}_t = tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{13}$$

Tras estos tenemos que actualizar el estado $\mathbf{C_{t-1}}$. Para hacerlo realizamos la siguiente operación matemática con los resultados que hemos obtenido anteriormente:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{14}$$

Tras esto, ya podemos decidir que salidas vamos a tener de esta puerta, siendo el ***output gate layer***, resaltada en la **figura 10b**. Siguiendo el esquema podemos ver que tenemos tres salidas, dos correspondientes a $\mathbf{h_t}$, y una para $\mathbf{C_t}$. Vemos que para decidir estos

(a) *Input gate layer.* **(b)** *Output gate layer.*

Figura 10: *Parte final LSTM.*

valores de salida tenemos otra función sigmoide que regula la llave de paso, siendo estas operaciones:

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \tag{15}$$

$$h_t = o_t \cdot tanh\left(C_t\right) \tag{16}$$

Terminando así el funcionamiento de las LSTMs. Existen otros tipos de puerta que mejoran ciertos aspectos como las ***GRU***, pero para el propósito de este trabajo no se profundizará más.

4. Computación Cuántica

4.1. Bases de la computación cuántica

Para el desarrollo de esta sección se asume que el lector tiene conocimientos básicos en mecánica cuántica y espacios de Hilbert.

La unidad básica de la computación clásica es el bit, mientras que en el caso de la cuántica esta unidad recibe el nombre de **qubit**. Pero en lugar de ser un 0 o un 1, existe en un estado de superposición de dos niveles, pudiendo expresarlo de la siguiente manera:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{17}$$

Siendo $|0\rangle$ y $|1\rangle$ la base de estados en la que vamos a trabajar. Los coeficientes están normalizados, siendo $|\alpha|^2 + |\beta|^2 = 1$. El cuadrado de cada coeficiente es la probabilidad de que al medir tengamos dicho estado.

Generalmente tendremos más de un qubit, por lo que debemos tener una forma de combinarlos. Para hacer esto hacemos uso del producto tensorial quedando expresiones del estilo: $|01\rangle$ queriendo decir que el primer qubit está en el estado $|0\rangle$ y el segundo en el $|1\rangle$. Un producto tensorial entre dos qubits tiene el siguiente aspecto:

$$|\Psi_1\rangle \otimes |\Psi_2\rangle = \alpha_0\,|00\rangle + \alpha_1\,|01\rangle + \alpha_2\,|10\rangle + \alpha_3\,|11\rangle \tag{18}$$

De nuevo, los coeficientes deben cumplir la condición de normalización. De forma general podemos expresar para n qubits nuestro estado compuesto:

$$|\Psi\rangle = \alpha_0\,|0\ldots00\rangle + \alpha_1\,|0\ldots01\rangle + \ldots + \alpha_{2^n-1}\,|1\ldots11\rangle = \sum_{i=0}^{2^n-1} \alpha_i\,|i\rangle \tag{19}$$

Con $|i\rangle$ haciendo referencia al enésimo estado de la base computacional $\{|0\ldots00\rangle\,, \ldots, |1\ldots11\rangle\}$.

Otra forma útil de tratar estos estados es con matrices de densidad, así con la notación usada anteriormente podemos definir una matriz de densidad pura como:

$$\rho_{pure} = |\Psi\rangle\langle\Psi| = \sum_{i,j=0}^{N-1} \alpha_i^*\alpha_j\,|i\rangle\langle j|\,, \quad \alpha_i \in \mathbb{C} \tag{20}$$

Usando α^* como el conjugado de α. También es posible tener matrices de densidad mixtas donde los coeficientes no factorizan y tenemos en consecuencia:

$$\rho_{mixed} = \sum_{i,j=0}^{N-1} \alpha_{ij}\,|i\rangle\langle j|\,, \quad \alpha_{ij} \in \mathbb{C} \tag{21}$$

4.2. Puertas Cuánticas

Para poder realizar operaciones con nuestro qubits, necesitamos mecanismos que hagan evolucionar los estados de estos, tal y como hacen las puertas en computación clásica. Clásicamente tenemos las puertas lógicas universales ***NAND*** y ***NOR***, las cuales pueden actuar en diferentes configuraciones cómo cualquier otra puerta lógica simplificando el diseño físico de los circuitos, mientras que en computación cuántica no tenemos estas puertas universales, por lo que nuestros circuitos se van a componer de diferentes puertas, cuya realización física puede llegar a ser muy variada y compleja según el tipo de ordenador que tengamos.

Así tenemos las **puertas cuánticas**, que harán evolucionar nuestro sistema. Pero estas deben ser transformaciones unitarias, es decir, cumplen la relación $U^\dagger U = UU^\dagger = I$, siendo $U^\dagger$ el conjugado del operador lineal unitario U.

Existen numerosas puertas cuánticas, aquí mostraré algunas que son usadas en el resto del trabajo.

Gate	Circuit representation	Matrix representation	Dirac representation
X	$-[X]-$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\|1\rangle\langle 0\| + \|0\rangle\langle 1\|$
Y	$-[Y]-$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$i\|1\rangle\langle 0\| - i\|0\rangle\langle 1\|$
Z	$-[Z]-$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\|1\rangle\langle 0\| - \|0\rangle\langle 1\|$
H	$-[H]-$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)\langle 0\| +$ $\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)\langle 1\|$
S	$-[S]-$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\frac{1}{\sqrt{2}}\|0\rangle\langle 0\| + \frac{1}{\sqrt{2}}i\|1\rangle\langle 1\|$
R	$-[R]-$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 0 & \exp(-i\pi/4) \end{pmatrix}$	$\frac{1}{\sqrt{2}}\|0\rangle\langle 0\| +$ $\frac{1}{\sqrt{2}}\exp^{-i\pi/4}\|1\rangle\langle 1\|$

Figura 11: *Puertas cuánticas [27]*

Junto a estas, existen unas puertas que serán las que más usaré, siendo estas las puertas

de rotación R_x, R_y, y R_z

$$R_x(\theta) = \begin{pmatrix} cos\,(\theta/2) & -isin\,(\theta/2) \\ -isin\,(\theta/2) & cos\,(\theta/2) \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} cos\,(\theta/2) & -sin\,(\theta/2) \\ sin\,(\theta/2) & cos\,(\theta/2) \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$

Mientras que las puertas expresadas en la **figura 11** pueden aplicarse sobre un solo qubit, existen otras puertas que puede actuar sobre dos o más qubits, normalmente uno es el qubit objetivo, y los otros sirven cómo qubits de control. Así, para la mayoría de los objetivos en computación cuántica es necesario trabajar con varios qubits al mismo tiempo.

De esta forma tenemos por ejemplo la puerta **CNOT**, que es una puerta NOT pero con un qubit de control. Si el qubit de control se encuentra en el estado $|1\rangle$ entonces el qubit objetivo cambia su estado, en caso contrario no ocurre nada. Esta puerta aparecerá en los circuitos de la siguiente forma:

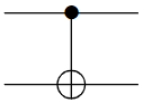


Figura 12: *Puerta cNOT.*

Luego tenemos la puerta **Toffoli**, que puede ser vista como una ccNOT, es decir, con dos qubits de control, actuando únicamente si los dos qubits de control están en el estado $|1\rangle$.

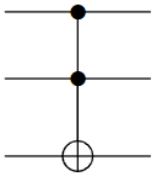


Figura 13: *Puerta Toffoli.*

Otra puerta de importancia es la **Swap**, que no será usada en este trabajo, pero su función es intercambiar el estado de dos qubits.

4.3. Tipos de ordenadores cuánticos y estado actual

Antes de continuar desarrollando algoritmos cuánticos es importante conocer el estado actual de las máquinas donde idealmente estaríamos ejecutando dichos algoritmos, por ello en esta sección voy a mencionar brevemente los dos tipos más importantes de ordenadores cuánticos que existen en la actualidad, cada uno con sus particularidades.

4.3.1. Computación cuántica basada en puertas

Estos son los ordenadores más prácticos, ya que son los más cercanos al funcionamiento de los ordenadores clásicos, puesto que la forma de realizar cálculos se basa en las puertas cuánticas, también son a veces llamados analógicos. Actualmente el ordenador cuántico de este tipo con mayor número de qubits es el perteneciente a IBM [30], con 127 qubits, siendo su funcionamiento principalmente exploratorio. El gran problema que estos ordenadores presentan es la coherencia temporal, por lo que a los ordenadores actuales se les llama ordenadores con ruido cuántico intermedio o **NISQ**, puesto que presenta un ruido los suficientemente bajo como para realizar numerosas tareas satisfactoriamente.

Existen diversas realizaciones físicas de este tipo de tecnología, pero que no exploraremos, destacando: los basados en superconductores, de iones atrapados, de resonancia magnética nuclear...

4.3.2. Templado cuántico

Este tipo de ordenador es el que más limitaciones tiene, en el sentido que no puede resolver problemas de tipo general, sino que sirve únicamente para algunos. Por ello las ventajas que poseen no son tan grandes comparados con los ordenadores clásicos. Estos sistemas permiten tener hasta 5000 qubits [8], pero de nuevo, las aplicaciones son limitadas y para los problemas claves de supremacía cuántica no son usados. De todas formas, presentan una muy buena plataforma para aplicaciones prácticas de la computación cuántica en el negocio y la investigación. A pesar de todo esto existe gran debate en la comunidad científica sobre si pueden ser considerados superiores a los ordenadores clásicos.

5. Redes Neuronales Cuánticas

Según lo que hemos visto de computación cuántica, parece difícil partir de un modelo de red neuronal clásico a uno cuántico. Por ello hay que ponerse creativo y buscar nuevos métodos. En general existen dos vías de investigación, una de ellas consiste en tratar de imitar los modelos clásicos en ordenadores cuánticos, mientras que la otra busca aprovechar ciertas cualidades cuánticas para crear modelos exclusivos para ordenadores cuánticos, llegando así a una suerte de **red neuronal cuántica**, ***QNN***. Para el propósito de este trabajo me centraré en cómo pasar los modelos clásicos vistos anteriormente a ordenadores cuánticos. Para ello debemos ver algunos conceptos que serán usados con frecuencia.

La base para obtener un modelo de aprendizaje autónomo es que sea capaz de modificar uno o más parámetros, por ello debemos tener un circuito que presente esta posibilidad. Así, un **circuito cuántico variacional**, **VQC** [15] consiste en un circuito cuántico que usando la puerta de rotación con parámetros libres, o en algunos casos puertas de otros tipos, permite hacer diversos tareas numéricas, como sería aproximación, optimización, clasificación... Así, un algoritmo que use este tipo de circuitos recibe el nombre de **algoritmo cuántico variacional**, o **VQA**. Este actúa como un algoritmo híbrido clásico-cuántico ya que la optimización de los parámetros se realiza de forma clásica, al menos en la mayoría de modelos actuales.

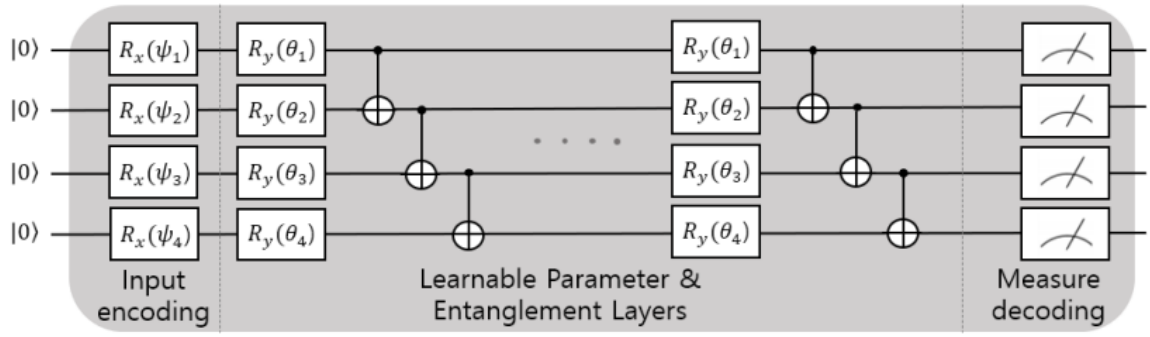


Figura 14: *Ilustración de una QNN [15].*

En la **figura (14)** tenemos un esquema de un circuito cuántico, que tendría como parámetros θ_i, junto con las respectivas puertas para su codificación y decodificación.

Antes de continuar con modelos más complejos, tratemos de describir el funcionamien-

to general de una posible QNN, ya que, a diferencia de las redes clásicas, debemos codificar la información clásica en estados cuánticos. Existen numerosas formas de hacer esto, cada una con sus propias ventajas y desventajas. En concreto, se puede ver en el capítulo 5 de [27] en mayor profundidad diversas maneras de codificar la información clásica. Para los objetivos de esta sección se usarán funciones arbitrarias para generar nuestros estados cuánticos, al igual que las respectivas funciones de decodificación.

Una vez tenemos unos estados cuánticos iniciales, que podrán ser denotados por la matriz $\boldsymbol{\rho_{in}}$, aplicaremos un circuito parametrizado o VQC, cuyas particularidades veremos más adelante. Este circuito hace la función principal del algoritmo, generando interacciones entre los diferentes estados y modificándolos según nuestros parámetros, que los podremos denotar con la letra $\boldsymbol{\theta}$ representando un vector de parámetros θ_i. Estos circuitos serán repetidos las veces que sean necesarias, y podemos catalogarlos como hicimos con las capas de las redes clásicas.

Tras esto, aplicaremos la operación medida sobre los qubits, decodificaremos el resultado, y obtendremos una salida $\bar{\mathbf{y}}_\mathbf{i}$. Evaluaremos el rendimiento de nuestra red respecto al valor de salida deseado $\mathbf{y_i}$ mediante el uso de una función pérdida, tal y como vimos para las redes clásicas. A continuación, utilizaremos un optimizador para mejorar los parámetros, actualizando el VQC. Este proceso se iterará hasta que converja a una precisión deseable.

5.1. Modelos de aprendizaje automático cuántico

A pesar de que me voy a centrar en los modelos de redes neuronales, cabe mencionar que existen diversas formas en las que la computación cuántica mejora los algoritmos de aprendizaje automático, cómo pueden ser métodos de agrupamiento (*clustering*), vecinos cercanos, **SVM** (*support vector machine)*, modelos de Ising...[27][32].

En concreto, la aplicación más común a este campo es resolver las operaciones algebraicas necesarias con algoritmos cuánticos que han demostrado ser sustancialmente más rápidos que los clásicos, siendo importante la inversión de matrices, la obtención de autovalores **VQE**, aplicaciones a matrices de tipo disperso, entre muchas otras. Pero quizás la aplicación que más futuro pueda tener fuera de las redes neuronales son los modelos basados en el algoritmo de búsqueda de Grover, el cuál podría revolucionar los centros de datos al reducir el tiempo necesario para encontrar datos [13].

5.2. Red neuronal convolucional cuántica

En esta sección vamos a explorar el modelo presentado en [7], mostrando como podría ser una arquitectura para una CNN en ordenadores cuánticos. El objetivo principal de estos investigadores fue recrear el modelo clásico, por lo que tendremos como ya vimos unas capas convolucionales, una red completamente conectada...

Esta arquitectura recibirá el nombre de **QCNN** (*quantum convolutional neural network*). Tendremos un estado cuántico desconocido de entrada, que definiremos con la matriz de densidad $\boldsymbol{\rho_{in}}$, luego habrá una **capa de convolución** que aplicará un VQC llamado $\mathbf{U_i}(\boldsymbol{\theta})$. Posteriormente en la **capa de pooling**, un número reducido de qubits serán medidos, y con el resultado de esta medida aplicaremos a los qubits cercanos unas puertas de rotación que denominaremos $\mathbf{V_j}(\boldsymbol{\theta})$, que de nuevo serán un VQC. Con este último paso eliminaremos las dependencias lineales, y la necesidad de aplicar una capa ReLU, cómo sí ocurría clásicamente.

El número de capas convolucionales y de pooling serán tantas como el sistema requiera, siendo deseable reducir al máximo el tamaño. Tras esto aplicaremos una **red completamente conectada** que será representada por la puerta **F** en los qubits restantes, que no es otra cosa más que un VQC creado específicamente para la clasificación en diferentes clases. Por último, medimos el resultado y decodificaremos esta información a una forma en la que sea utilizable.

Este modelo permite clasificar entradas con estados de **N** qubits con tan solo parámetros del orden $\mathbf{O}(\mathbf{log}(\mathbf{N}))$, lo que permite una muy buena implementación en problemas de aprendizaje autónomo.

Una vez explicado el funcionamiento, veamos algo más concreto. Para ello imaginemos que tenemos un conjunto de **M** vectores para entrenar nuestra red con la forma $(|\boldsymbol{\psi_\alpha}\rangle, \mathbf{y_\alpha})$, con $|\psi_\alpha\rangle$ el estado de entrada, e y_α la correspondiente salida. Usando la función coste cuadrática media, tenemos que el error se obtendrá con:

$$MSE = \frac{1}{2M}\sum_{\alpha=1}^{M}\left(y_i - f_{\{U_i,V_j,F\}}\,|\psi_\alpha\rangle\right)^2 \tag{22}$$

Con $\mathbf{f_{\{U_i,V_j,F\}}}\,|\boldsymbol{\psi_\alpha}\rangle$ el valor esperado para nuestro estado de entrada $|\psi_\alpha\rangle$. El aprendizaje consiste en ir optimizando los parámetros de los VQC hasta que el resultado converja a un resultado deseable, generalmente se usará el método del descenso en gradiente. En una sección posterior estudiaremos como se produce la optimización de los parámetros en nuestros circuitos cuánticos.

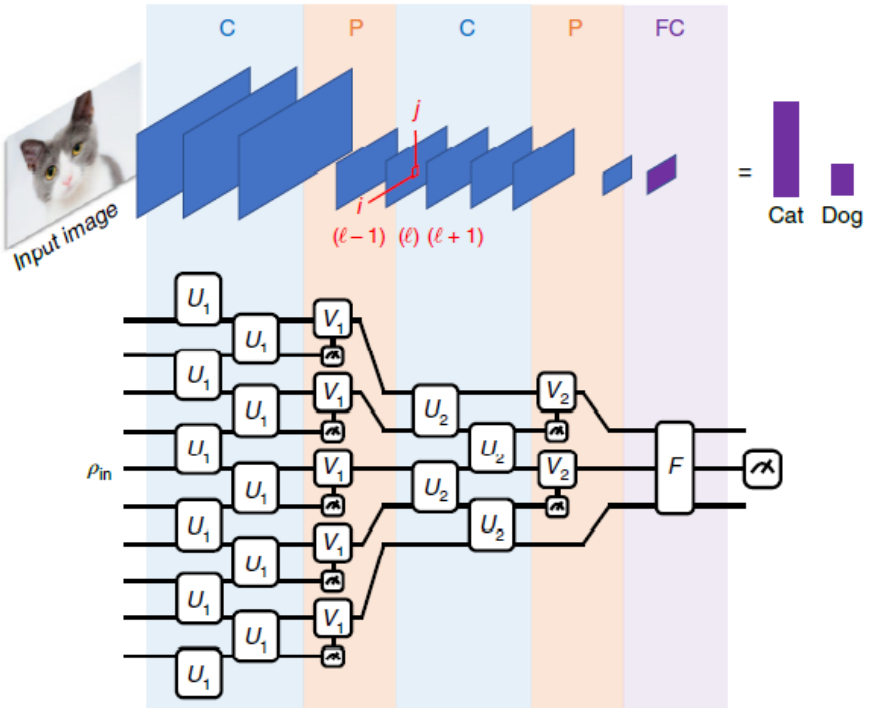


Figura 15: *Arquitectura QCNN [7]. (C: Convolutional. P: Pooling. FC: Fully connected.)*

5.3. Red neuronal recurrente cuántica

Ya vimos el gran potencial que presentan las RNN para predecir series temporales, por lo que es natural buscar algún algoritmo que funcione de forma similar en ordenadores cuánticos. Cómo para cualquier red neuronal cuántica, existen diversos enfoques, pero para el fin de este trabajo me parece interesante explicar el modelo que presentan *Yuto T. et al.* [29].

Al igual que para la QCNN, vamos a trabajar con VQC, donde la optimización de los parámetros se hará con ordenadores clásicos. Este modelo recibirá el nombre de **QRNN** (*Quantum recurrent neural network*).

Este modelo usará $\mathbf{N}$ qubits, con $\mathbf{N} = \mathbf{n_a} + \mathbf{n_b}$, teniendo dos subsistemas con sus propios qubits. Estos sistemas, $\mathbf{A}$ y $\mathbf{B}$, cumplen diferentes objetivos. Los qubits del primero de ellos nunca serán medidos, estos servirán de **celda de estado** para pasar la información en cada paso a la siguiente celda, funcionando como ya vimos en el apartado al estudiar las RNN. Luego, los qubits del sistema B son medidos e inicializados tras cada paso, para así obtener la predicción temporal deseada. De esta forma al inicializar de nuevo estos qubits podemos cargar la información del siguiente paso, siendo esta información codificada para así ser añadida al flujo de información.

Cada paso de tiempo consiste en tres partes distintas, teniendo una **fase de codificación**, otra de **evolución** y, por último, una de **medida**. Podemos ver estos pasos en la **figura 16**.

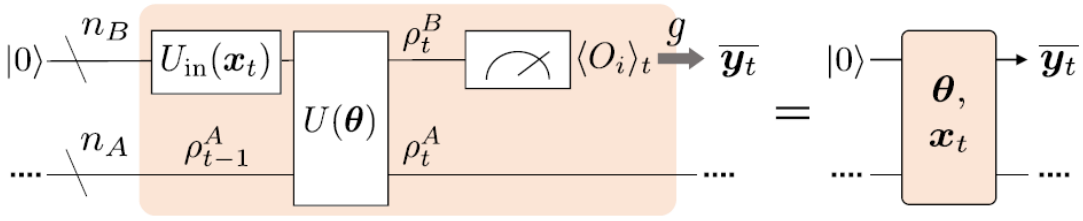


Figura 16: *Celda de una QRNN [29].*

En la **fase de codificación**, convertimos una entrada $\mathbf{x_i}$ a un estado cuántico, dependiendo la forma de este del tipo de codificación que usemos [27]. Esta codificación de los qubits del sistema B se realizará mediante el circuito $\mathbf{U_{in}(x_t)}$ aplicado sobre el estado inicializado $|\mathbf{0}\rangle^{\otimes \mathbf{n_B}}$, queriendo decir esta última expresión que todos los qubits del sistema B se encuentran en el estado $|0\rangle$. Mientras tanto, podemos ver cómo los qubits del sistema A ya poseen información, o más bien, no se encuentran en un estado cuántico trivial, sino que su estado es la matriz de densidad $\boldsymbol{\rho}^{\mathbf{A}}_{\mathbf{t-1}}(\boldsymbol{\theta}, \mathbf{x_0}, \ldots, \mathbf{x_{t-1}})$, la cual proviene de estados anteriores, y por tanto, de entradas de información previas.

Luego, en la **fase de evolución**, aplicamos el VQC $\mathbf{U}(\boldsymbol{\theta})$ a todos los qubits del sistema. $U(\theta)$ tiene en cuenta la información de estados anteriores y de la nueva entrada, por lo que genera una interacción entre los sistemas A y B. Estos investigadores llaman a la matriz reducida de A y B $\boldsymbol{\rho}^{\mathbf{A}}_{\mathbf{t}}$ y $\boldsymbol{\rho}^{\mathbf{B}}_{\mathbf{t}}$ respectivamente después de haber aplicado $U(\theta)$.

Por último, en la **fase de medida**, procedemos a aplicar la operación de medida a los valores esperados de un conjunto de observables conmutativos del sistema B, que se llamarán $\{O_i\}$. Así, los valores esperados serán: $\langle \mathbf{O_i} \rangle_{\mathbf{t}} = \mathbf{Tr}\left[\boldsymbol{\rho}^{\mathbf{B}}_{\mathbf{t}} \mathbf{O_i}\right]$. Siendo Tr la operación que extrae la traza de la matriz correspondiente.

Estos valores esperados, deben ser decodificados a las predicciones $\bar{\mathbf{y}}_{\mathbf{t}}$. Tras haber hecho la medida de los datos de salida, inicializamos los qubits del sistema B a $|0\rangle^{\otimes n_B}$. Este proceso será repetido tantas veces, como predicciones $\bar{y}_t$ necesitemos.

El proceso de aprendizaje será como de costumbre, tendremos una **función pérdida L** dependiente de las predicciones y los valores de salida del conjunto de datos de entrenamiento. Con el objetivo de minimizar este valor usaremos un optimizador para ir cambiando los parámetros θ del VQC. La minimización se hará en un ordenador clásico,

aunque existen algunas propuestas para hacer esta optimización de forma cuántica en ordenadores de tipo templado cuántico.

5.4. VQC: Circuitos cuánticos parametrizados

A lo largo de las arquitecturas anteriores es central el concepto de circuito cuántico parametrizado o **VQC**, por ello es clave ver en qué consisten y cómo funcionarían.

Ya veremos más adelante en las redes neuronales cuánticas híbridas que a veces solo es necesario tener algunas capas cuánticas, de esta manera las puertas cuánticas más comunes en estos algoritmos son las de rotación. En el artículo [29] se propone un VQC junto con el desarrollo matemático del gradiente, veamos en más detalle cómo sería.

Nuestro circuito lo podemos poner:

$$\mathbf{U}(\boldsymbol{\theta}_\mathbf{i}) = \mathbf{R}_\mathbf{x}(\boldsymbol{\alpha}_\mathbf{i})\mathbf{R}_\mathbf{z}(\boldsymbol{\beta}_\mathbf{i})\mathbf{R}_\mathbf{x}(\boldsymbol{\gamma}_\mathbf{i}) \tag{23}$$

Con $\boldsymbol{\theta}_i = \{\alpha_i, \beta_i, \gamma_i\}$. Teniendo cada puerta de rotación y cada circuito sus propios parámetros, llegando a una mayor profundidad de aprendizaje. Así, dependiendo de la función que tenga nuestra red, el circuito en sí mismo puede ser muy diferente, pero veamos un ejemplo:

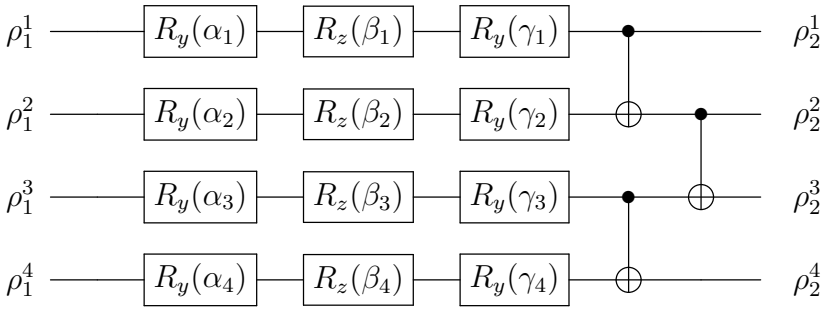


Figura 17: *Ejemplo de VQC, partiendo de $U(\boldsymbol{\theta})$.*

En la **figura 17** tenemos el circuito $U(\theta)$ junto con unas puertas cNOT, que por simplificar los cálculos posteriores quedan fuera del propio VQC. Este tiene una matriz de entrada ρ_1, siendo los superíndices la forma de denotar el estado cuántico que recibe cada qubit. Así en las posteriores $U(\theta)$, las matrices recibidas serán ρ_2, y así de forma sucesiva.

Este sería solamente un ejemplo, pero pueden existir circuitos con puerta de rotación controladas, cuya actuación depende de uno o más qubits dando lugar a sistemas muy

complejos y profundos.

Para completar este apartado veamos un desarrollo analítico del proceso de aprendizaje del circuito $U(\boldsymbol{\theta})$ propuesto anteriormente, sin tener en cuenta las puertas cNOT. Como explican los autores de [29] el proceso para obtener el gradiente analítico es similar al visto clásicamente, pero con la diferencia de tratar con estados cuánticos.

$$\frac{\partial}{\partial\theta_i}\left[U(\boldsymbol{\theta})\rho U^{\dagger}(\boldsymbol{\theta})\right] \tag{24}$$

Donde el superíndice † indica el traspuesto del conjugado complejo del operador al que se le aplica esta operación. Si nos fijamos, estamos haciendo el gradiente del valor esperado de ρ tras aplicar el circuito en cuestión, por lo que para conocer bien las dependencias con el parámetro tenemos que recordar la codificación que se ha realizado sobre los datos clásicos, siendo $\mathbf{x_i}$ los datos de entrada, $\mathbf{y_i}$ la salida deseada, e $\bar{\mathbf{y}}_\mathbf{i}$ la salida calculada por la red. Entonces tenemos que $\bar{\mathbf{y}}_\mathbf{t} = \mathbf{g}(\langle\mathbf{O}\rangle_\mathbf{t})$, con g la función de decodificación. También podemos tomar la función pérdida cómo la cuadrática media, que si la recordamos es: $L = \frac{1}{2}\sum_{t=0}^{T-2}\left(\bar{y_t} - x_{t+1}\right)^2$, donde se ha escrito x_{t+1} en lugar de y_t, puesto que la salida esperada en una RNN es la siguiente entrada del conjunto de datos. De esta forma el gradiente sobre la función pérdida será:

$$\frac{\partial L}{\partial\theta_i} = \sum_{t=0}^{T-2}\left(\bar{y}_t\left(\boldsymbol{\theta}, x_0 \ldots x_t\right) - x_{t+1}\right)\frac{\partial\bar{y}_t}{\partial\theta_i} \tag{25}$$

Siendo:

$$\frac{\partial\bar{y}_t}{\partial\theta_i} = \frac{\partial g\left(\langle O\rangle_t\right)}{\partial\theta_i} = \frac{\partial g}{\partial\langle O\rangle_t}\frac{\partial\langle O\rangle_t}{\partial\theta_i} \tag{26}$$

Los autores definen $\boldsymbol{\rho}^{\mathbf{B}}_{\mathbf{in,t}} = \mathbf{U_{in}}(\mathbf{x_t})\left|\mathbf{0}\right\rangle\left\langle\mathbf{0}\right|\mathbf{U}^{\dagger}_{\mathbf{in}}(\mathbf{x_t})$, que depende exclusivamente de la entrada en el paso de tiempo $\mathbf{t}$, y que por simplificar los cálculos se expresará simplemente como: $\boldsymbol{\rho}^{\mathbf{B}}_{\mathbf{in,t}}$.

También se define: $\boldsymbol{\rho}^{\mathbf{AB}}_{\mathbf{in,t}}\left(\boldsymbol{\theta}, \mathbf{x_0}, \ldots, \mathbf{x_t}\right) = \boldsymbol{\rho}^{\mathbf{A}}_{\mathbf{t-1}} \otimes \boldsymbol{\rho}^{\mathbf{B}}_{\mathbf{in,t}}$. Donde el subíndice t se refiere al paso temporal en el que nos encontramos, de esta forma podemos definir la serie t: $t = 0, 1, \ldots, T$, con T el número total de pasos.

Para simplificar los siguiente cálculos, ponemos $\rho^{AB}_{in,t}\left(\boldsymbol{\theta}, x_0, \ldots, x_t\right) = \boldsymbol{\rho}^{\mathbf{AB}}_{\mathbf{in,t}}(\boldsymbol{\theta})$. Una vez hecho esto veamos el gradiente del valor esperado:

$$\frac{\partial\langle O\rangle_t}{\partial\theta_i} = \frac{\partial}{\partial\theta_i}Tr\left[\rho^B_t O\right] = \frac{\partial}{\partial\theta_i}Tr\left[U(\boldsymbol{\theta})\rho^{AB}_{in,t}(\boldsymbol{\theta})U^{\dagger}(\boldsymbol{\theta})O\right] =$$

$$Tr\left[\frac{\partial U(\boldsymbol{\theta})}{\partial\theta_i}\rho^{AB}_{in,t}(\boldsymbol{\theta})U^{\dagger}(\boldsymbol{\theta})O\right] + Tr\left[U(\boldsymbol{\theta})\rho^{AB}_{in,t}(\boldsymbol{\theta})\frac{\partial U^{\dagger}(\boldsymbol{\theta})}{\partial\theta_i}O\right] + Tr\left[U(\boldsymbol{\theta})\frac{\partial\rho^{AB}_{in,t}(\boldsymbol{\theta})}{\partial\theta_i}U^{\dagger}(\boldsymbol{\theta})O\right]$$

Como vemos, conocemos los dos primeros términos puesto que provienen de la diferenciación de nuestro VQC, que tiene una expresión matricial y es fácilmente diferenciable. Pero no conocemos la derivada de la densidad de estados, por lo que tenemos que expandir la expresión y realizar la derivada.

$$\frac{\partial \rho_{in,t}^{AB}(\boldsymbol{\theta})}{\partial \theta_i} = \frac{\partial \rho_{t-1}^{A}(\boldsymbol{\theta})}{\partial \theta_i} \otimes \rho_{in,t}^{B} = \frac{\partial}{\partial \theta_i} Tr_B \left[U(\boldsymbol{\theta}) \rho_{in,t=1}^{AB}(\boldsymbol{\theta}) U^{\dagger}(\boldsymbol{\theta}) \right] \otimes \rho_{in,t}^{B} =$$

$$= \left\{ Tr_B \left[K_1 \right] + Tr_B \left[K_2 \right] + Tr_B \left[K_3 \right] \right\} \otimes \rho_{in,t}^{B}$$

$$Con: K_1 = \frac{\partial U(\boldsymbol{\theta})}{\partial \theta_i} \rho_{in,t-1}^{AB}(\boldsymbol{\theta}) U^{\dagger}(\boldsymbol{\theta}); \quad K_2 = U(\boldsymbol{\theta}) \rho_{in,t-1}^{AB}(\boldsymbol{\theta}) \frac{\partial U^{\dagger}(\boldsymbol{\theta})}{\partial \theta_i};$$

$$K_3 = U(\boldsymbol{\theta}) \frac{\partial \rho_{in,t-1}^{AB}(\boldsymbol{\theta})}{\partial \theta_i} U^{\dagger}(\boldsymbol{\theta})$$

Donde Tr_B es la operación traza sobre el sistema B. Los dos primeros términos pueden ser evaluados, mientras que para el último término es necesario expandir otra vez $U(\boldsymbol{\theta}) \frac{\partial \rho_{in,t-1}^{AB}(\boldsymbol{\theta})}{\partial \theta_i} U^{\dagger}(\boldsymbol{\theta})$ en términos de $\rho_{in,t-2}^{AB}$. Y como el lector debe de haber visto, es necesario iterar esta expansión hasta llegar al término inicial de entrada, quedando bien definido el gradiente de la función pérdida, y como ya se vio al estudiar el descenso en gradiente es el ingrediente necesario para que exista un aprendizaje de nuestro modelo.

Además, la forma en la que ha sido hecho el desarrollo permite una resolución más o menos general para cualquier tipo de VQC y matrices de densidad de estados, teniendo en cuenta las posibles particularidades que tengan las matrices de estados. El caso estudiado tiene la particularidad de que tenemos dos sistemas con diferentes qubits que se les fuerza a interaccionar. Por ejemplo, si tuviéramos la QCNN vista anteriormente, solo tendríamos una matriz de estado con diferentes pasos, y de tamaño menor conforme nos acercamos al final. El proceso de la obtención del gradiente sería similar salvaguardando estas diferencias.

5.5. Aplicaciones

En esta sección hemos visto cómo sería la arquitectura y el funcionamiento de algunos modelos de redes neuronales cuánticas puras, usando ordenadores clásicos exclusivamente para la optimización de parámetros, pero dadas las particularidades de la computación cuántica no es posible sustituir con estos los modelos clásicos. De todas formas, existen algunos casos en los que previsiblemente estos modelos superen con creces los algoritmos clásicos, no solo en eficiencia, sino resolviendo problemas que antes no eran posibles. A

continuación, veremos algunas de estas aplicaciones.

Más allá de aplicaciones para problemas de *Big Data* o inteligencia artificial, estos modelos pueden ser la respuesta a problemas físicos de gran importancia. Estoy hablando del **problema de muchos cuerpos cuántico**. Esta cuestión ha traído dolores de cabeza a los físicos en los últimos años por la dificultad de describir de forma precisa estos sistemas cuánticos tan complejos, cuya solución plantearía enormes avances en diversos sectores cómo la ciencia de materiales o la propia investigación en cuántica. Hasta ahora la forma de resolver este problema cae en las aproximaciones computacionales, pero por la naturaleza de los ordenadores clásicos, estos se ven limitados en cuanto a potencia cuando el sistema crece. Aquí es donde llegan los ordenadores cuánticos. En concreto, veremos cómo los modelos de redes neuronales cuánticas vistos plantean formas prometedoras de resolver estos sistemas de forma exacta y en tiempos menores.

5.5.1. El problema cuántico de muchos cuerpos

Antes de continuar, es necesario conocer el concepto de **ansatz**. Este se define como una solución inicial a un problema concreto, pero no será algo aleatorio, sino que se trata de elegir un buen punto de partida. En concreto, estamos hablando de aplicar una serie de puertas cuánticas a los qubits en su estado inicial. La utilidad de esto es reducir el tiempo necesario para llegar a una solución, así cómo llegar a una solución correcta.

Un ansatz de gran importancia es **MERA**, (*the multiscale entanglement renormalization ansatz*). es muy relevante ya que es una estructura que codifica de forma eficiente estados cuánticos de muchos cuerpos de sistemas de cristales con D dimensiones, que permite obtener los valores esperados de forma exacta [31].

Tal y cómo se describe en [31], MERA está compuesta de una red de tensores isometricos de dimensión $D+1$. Puesto que MERA tiene como prioridad preservar el entrelazamiento cuántico entre los qubits, funciona cómo un buen punto de partida para describir efectos cuánticos que no pueden ser descritos clásicamente. Este *ansatz* tiene varias aplicaciones, por ejemplo, el estudio de transiciones de fase, excitaciones de cuasi-partículas...

Volvamos al modelo de **QCNN** que vimos anteriormente. En ese mismo artículo [7], plantean como su arquitectura presenta una forma de usar MERA, pero al mismo tiempo introducen el concepto de **QEC** (*quantum error correction*), que consiste en un sistema para corregir errores cuánticos sin necesidad de hacer colapsar la función de onda.

De forma general podemos ver como la QCNN funciona de forma inversa al MERA, es decir, con MERA partimos de un estado cuántico generado por una secuencia de capas

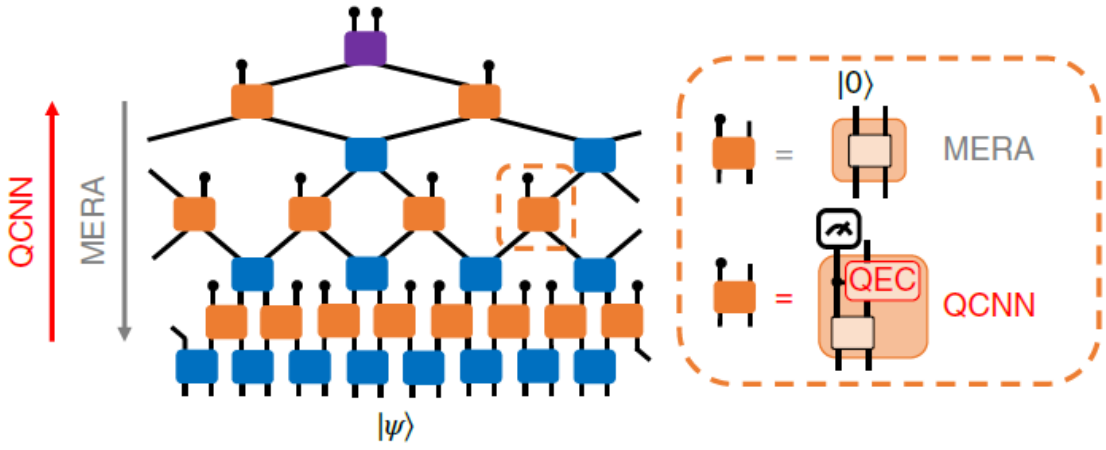


Figura 18: *MERA [7].*

unitarias e isométricas aplicadas sobre un estado inicial, por ejemplo $|00\rangle$. La capa isométrica introduce inicialmente un conjunto de nuevos qubits en un estado predeterminado, creciendo la red de forma exponencial. Este sistema permite un alto rango de correlaciones para sistemas críticos, y como vemos, la QCNN tiene una estructura bastante similar, pero en sentido contrario. Así, para cualquier $|\psi\rangle$ con una representación en MERA, tendremos una QCNN que reconocerá $|\psi\rangle$ de forma determinista en la medida. Puede ser entendido como la función inversa de MERA.

También es importante ver que MERA permite hacer uso del algoritmo de **QPR**, que permite obtener la fase cuántica del problema de muchos cuerpos.

5.5.2. Detección de la fase cuántica en 1D mediante QCNN

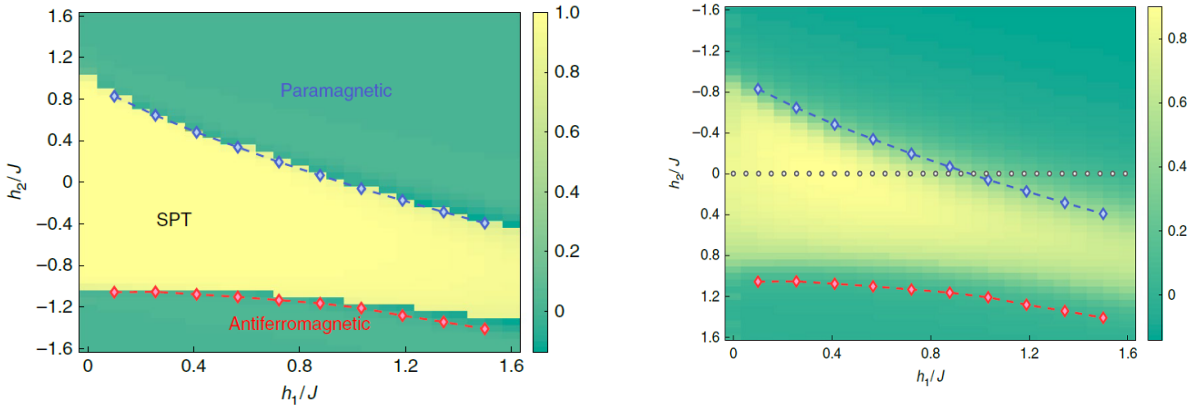
Para ver el potencial que tiene el modelo de QCNN [7], los investigadores lo aplican a la **QPR** (*quantum phase recognition*) de una clase de sistema de muchos cuerpos en 1D. Se considera en concreto el sistema conocido cómo: Fase $\mathcal{P}$, $\mathbb{Z}_2 \times \mathbb{Z}_2$ topológica protegida por simetría o fase **SPT**. Esta fase contiene la cadena de Haldane $S = 1$,(ver [9]), y estados fundamentales $|\psi\rangle$ de una familia de Hamiltonianos con spin $-1/2$ con condiciones de contorno abiertas. El Hamiltoniano en cuestión tiene la siguiente forma:

$$H = -J \sum_{i=1}^{N-2} Z_i X_{i+1} Z_{i+2} - h_1 \sum_{i=1}^{N} X_i - h_2 \sum_{i=1}^{N-1} X_i X_{i+1} \tag{27}$$

Con X_i, Z_i, operadores de Pauli aplicados al spin en la posición i. Donde h_1, h_2 y J son parámetros del Hamiltoniano en cuestión. Para el caso de $h_2 = 0$, el Hamiltoniano

puede ser resuelto de forma exacta con la **transformación de Jordan-Wigner** [21], confirmando que la fase $\mathcal{P}$ está caracterizada por parámetros de orden no local. Mientras que si $h_1 = h_2 = 0$ todos los términos conmutan y un estado fundamental será el estado en cluster 1D. Así, el objetivo perseguido por los autores será identificar si un estado fundamental desconocido dado por el diagrama de fase pertenece a $\mathcal{P}$.

No entraremos más en detalle en cómo se resuelve este sistema, pero en el artículo [7] se detalla el proceso de varias formas junto con el desarrollo matemático. Para el objetivo que estamos persiguiendo en esta sección, mostraré los resultados que obtuvieron los investigadores al compararlo con métodos tradicionales.



(a) *Diagrama de fase 1.*

(b) *Diagrama de fase 2.*

Figura 19: *Diagramas de fase [7].*

La **figura 19a** representa el diagrama de fase del Hamiltoniano del sistema expuesto, donde los diamantes azules y rojos representan los límites de frontera extraídos de simulaciones numéricas **DMRG** (*density matrix renormalization group* [10]), mientras que el mapa de color del fondo representa la salida de una QCNN construida de forma exacta para un tamaño de entrada de $\mathbf{N = 45}$ spines. De nuevo, en el artículo puede verse el procedimiento llevado a cabo.

Mientras que la **figura 19b** muestra el resultado de una QCNN entrenada para un sistema de $\mathbf{N = 15}$ spines partiendo de valores iniciales aleatorios. Para facilitar el cálculo se tomó $h_2 = 0$, permitiendo resolver el sistema con Jordan-Wigner, que en la figura es representado por los puntos grises. Los diamantes de color rojo y azul son los límites de frontera de fase provenientes del cálculo numérico DMRG y el mapa de color es el resultado previsto por nuestra QCNN.

Para concluir esta sección es importante recalcar las capacidades que presentan los modelos de aprendizaje autónomo para resolver problemas físicos, pues únicamente con optimizaciones son capaces de llegar a resolver sistemas muy complejos, sin necesidad de crear de forma precisa un algoritmo a mano como se venía haciendo clásicamente. Aún queda un largo camino por recorrer para poder simular sistemas grandes puesto que estamos limitados por los ordenadores cuánticos actuales. Además, es importante ver como son necesarios un menor número de parámetros para obtener resultados similares a los clásicos, mostrando una clara superioridad por parte de la computación cuántica.

5.5.3. Aplicaciones QRNN

Los investigadores de [29] probaron algunas series temporales arbitrarias para comprobar las capacidades de este modelo, llegando a la conclusión de que mientras que el modelo presenta futuro, las aplicaciones reales aún quedan lejos, ya que se necesitan numerosos qubits y es necesario que estos mantengan la coherencia por un largo tiempo. Esta arquitectura en concreto inicializa de nuevo un conjunto qubits tras cada paso, pero el otro sistema debe mantener la coherencia por largos tiempos, lo que dificulta que se pueda aplicar a tareas complejas.

Sin embargo, si que podrá ser usado con fines físicos, por ejemplo, predecir la evolución temporal de sistemas cuánticos partiendo de unas condiciones iniciales, o volviendo a aplicaciones clásicas, predecir series temporales donde los algoritmos clásicos no son capaces de extraer ciertas correlaciones que si podrían en el caso cuántico, dado la mayor profundidad de las interacciones entre qubits.

Por último, existe un modelo de red neuronal recurrente cuántica importante que no he desarrollado pues está basado en ordenadores de tipo **templado cuántico** [5], que también realiza la optimización de forma cuántica. Esta muestra una gran precisión a la hora de reconocer dígitos dibujados a mano en la **librería MNIST**. La superioridad frente a los algoritmos clásicos es cuestionable, pero muestra como ya existen técnicas para emplear la computación cuántica. A pesar de esto, si que consigue mejorar el rendimiento frente a los ordenadores clásicos, pues el número de pasos necesarios disminuye con el tamaño de la secuencia, mostrando una ventaja del algoritmo cuántico.

6. Redes neuronales híbridas

Tras lo visto en apartados anteriores, el lector puede deducir que las redes neuronales cuánticas tienen aplicaciones casi en exclusiva a modelar fenómenos cuánticos, y en cierta manera así es. Pero debido al crecimiento en los últimos años de la inteligencia artificial existe un gran interés en aplicar las ventajas cuánticas a situaciones reales en la industria y la tecnología. Una forma de hacerlo sería mediante el uso de redes neuronales híbridas, es decir, combinando redes clásicas con alguna capa formada por circuitos cuánticos, en concreto con algún VQC.

A pesar de la aparente facilidad de implementar estos modelos, todavía no existen aplicaciones a gran escala. Sin embargo, los esfuerzos realizados tanto por empresas, cómo por entidades públicas dejan ver el gran interés que suscitan en el sector.

Por otro lado, debemos tener en cuenta que estamos tratando con un sector todavía en su infancia, y la información de investigaciones a las que he podido tener acceso es limitada, dado la escasa publicación de artículos con la suficiente consistencia tanto teórica como práctica. Por ello me centraré en algunas investigaciones que considero interesantes, pudiendo existir otras de gran importancia a las que no menciono. Habiendo hecho esta aclaración, sigamos.

6.1. Redes híbridas convolucionales

En apartados anteriores hemos visto las arquitecturas de una CNN con ordenadores clásicos y una QCNN que emplea únicamente ordenadores cuánticos salvo para optimizar parámetros. Veamos un modelo desarrollado por *Junhua L. et al.* [16], que llamarán **QCCNN** (*quantum-classical convolutional neural network*).

Como ya vimos el proceso de carga datos clásicos a cuánticos requiere de una operación de codificación. En este artículo en concreto, los autores deciden usar la **codificación qubit**:

$$\theta_j \rightarrow cos\left(\theta_j\right)|0\rangle + sin\left(\theta_j\right)|1\rangle \tag{28}$$

De esta manera, si tenemos un conjunto de dato de entrada con tamaño **N**, el resultado de esta codificación será un estado cuántico de N qubits, que estará contenido en un espacio de Hilbert de dimensión 2^N. Sin embargo, esta forma de codificar será prohibitiva para los ordenadores cuánticos actuales, pues el número de qubits es limitado. Por ello, al igual que ocurría con los ordenadores clásicos, necesitamos hacer uso de la operación de convolución, para así reducir el tamaño total.

Por esta razón, el VQC se implementará en la capa de convolución, donde además de reducir el tamaño, los entrelazamientos cuánticos pueden dar lugar a relaciones que los ordenadores clásicos no podrían hacer con la misma eficiencia. Siendo así el objetivo de este modelo crear un mapa de características cuántico. Esta primera capa la llamaremos **capa convolucional cuántica**. Tras esta tendremos una red neuronal clásica que tiene la tarea de clasificar.

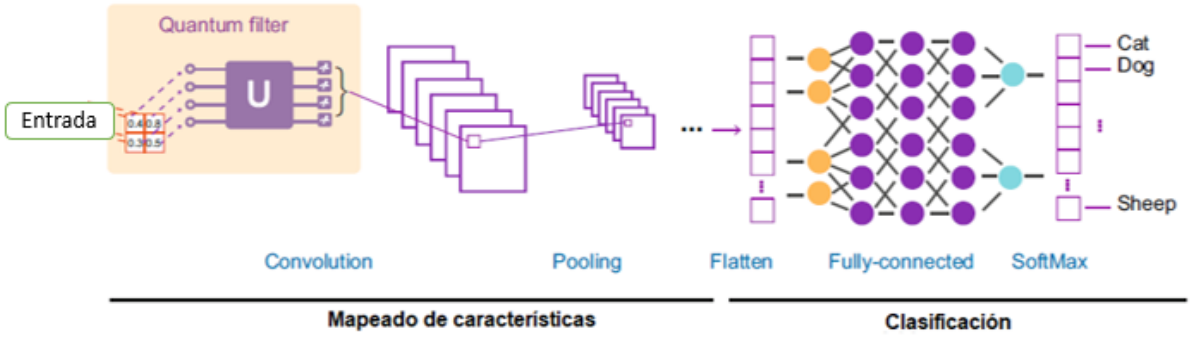


Figura 20: *Arquitectura QCCNN [16]*

.

La **figura 20** nos muestra la arquitectura a gran escala del modelo que vamos a estudiar, siendo muy similar a los modelos ya vistos al principio del trabajo, con la única diferencia de que tenemos un **filtro cuántico**. De esta manera, imaginemos que tenemos un conjunto de datos de dos dimensiones $v \times h$, con un filtro de tamaño $m \times n$. Entonces el mapeado es realizado con la siguiente operación lineal:

$$A_{1:m,1:n} \rightarrow \sum_{1 \leq i \leq m, 1 \leq j \leq n} A_{i,j} W_{i,j} \tag{29}$$

Sin entrar en mucha más profundidad, la matriz de salida tendrá las dimensiones $(v-m+1) \times (h-n+1) \times d \cdot k$, siendo $d \cdot k$ el producto de la profundidad de color y el número de filtros. Pero en nuestro modelo el filtro es cuántico por lo que debemos crear un circuito. Este lo podemos ver en la **figura 21**.

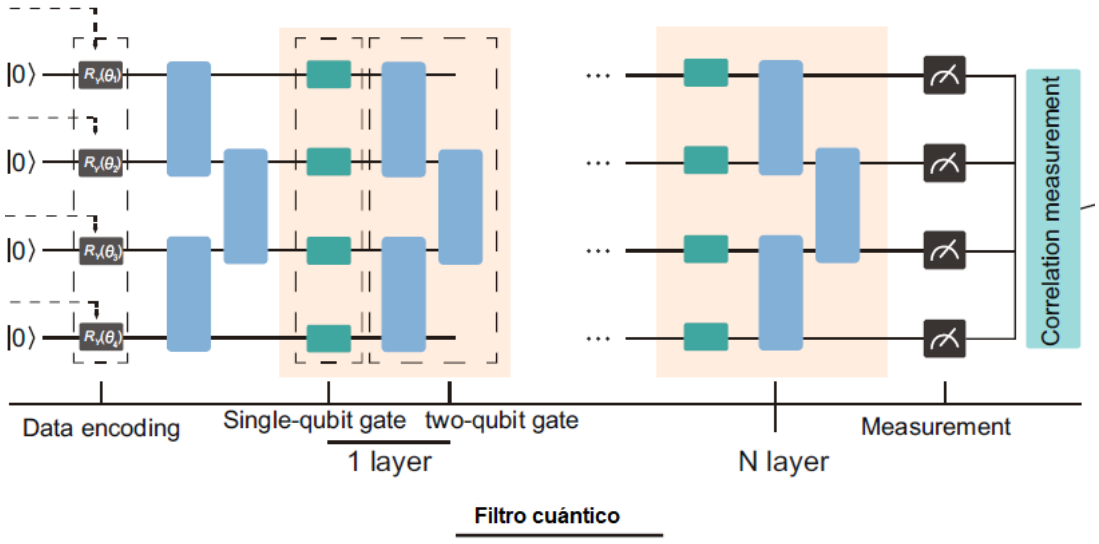


Figura 21: *Filtro cuántico [16].*

Así, nuestro filtro toma una imagen de tamaño $m \times n$, y lo codifica en estados cuánticos: $\left|\Psi^i\left(A_{p:(p+m-1),q:(q+n-1)}\right)\right\rangle$, con $N = mn$ siendo el número de qubits necesarios. Indicando que estamos tratando con el estado de entrada por el superíndice i, mientras que el superíndice o indica el estado de salida.

Si recordamos nuestra función de codificación podemos parametrizar nuestro circuito en función de $\boldsymbol{\theta}$, y así hacer evolucionar el estado cuántico aplicando el VQC, quedando el estado de salida:

$$\left|\Psi^o\right\rangle = U\left(\boldsymbol{\theta}\right)\left|\Psi^i\left(A_{p:(p+m-1),q:(q+n-1)}\right)\right\rangle \tag{30}$$

Después de haber hecho evolucionar nuestro estado cuántico, medimos el valor esperado del observable $\mathbf{Z}^{\otimes \mathbf{N}}$, que no es otra cosa que el **mapa de características cuántico**. Podemos expresar el mapa de características de la siguiente forma:

$$A_{p:(p+m-1),q:(q+n-1)} \rightarrow \left\langle\Psi^o\right| Z^{\otimes N}\left|\Psi^o\right\rangle \tag{31}$$

Si nos fijamos bien, el resultado de la convolución y el mapeado ya no es lineal, por lo que no necesitamos una función de activación como ocurría para los modelos clásicos donde teníamos una capa de ReLU. Además, por la forma de nuestro algoritmo los qubits pueden ser reutilizados. Tras cada iteración pasaríamos nuestra matriz de entrada y una vez haya completado la convolución podemos volver a usar estos qubits.

El VQC que tenemos hace uso de puertas para uno y dos qubits. Usaremos la puerta de rotación $\mathbf{R_y}(\boldsymbol{\theta})$, que ya vimos su aspecto cuando estudiamos algunas puertas cuánticas. También haremos uso de la puerta cNOT para generar entrelazamiento entre los qubits, creando las correlaciones en el mapa de características.

El número total de operaciones que tenemos que aplicar, o mejor dicho, el número de puertas cuánticas que usamos crece de forma polinómica con el número de qubits N tal que $\mathbf{O} \backsim \mathbf{poly}(\mathbf{N})$. Cada una de estas capas será interpretada como una dimensión extra en profundidad, así a mayor profundidad, obtendremos generalmente mejores resultados por la existencia de un mayor número de filtros cuánticos. Un ejemplo de dos capas de esta parte del circuito cuántico lo podemos ver en la **figura 22**

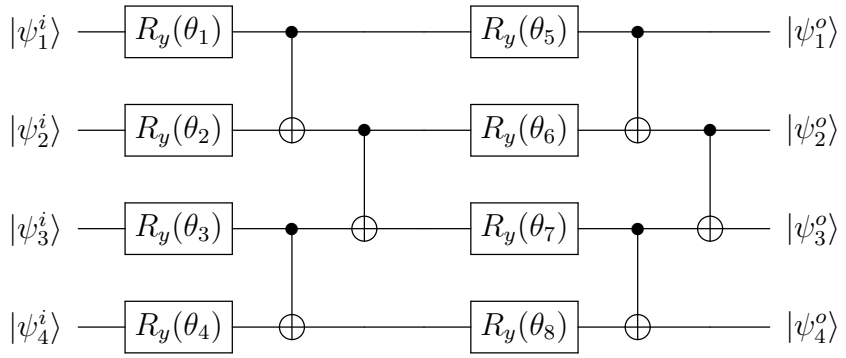


Figura 22: *Capa del filtro cuántico.*

Una vez hecho esto, debemos optimizar los parámetros. La forma de hacer esto consiste en obtener el gradiente de la función pérdida. Los autores usaron el **método de las diferencias finitas** para obtener las derivadas. Veamos cómo hacer este proceso de diferenciación.

La principal diferencia con el método del descenso en gradiente clásico es que ahora trabajamos con estados cuánticos, y por tanto las matemáticas son diferentes, como ya pudimos ver al obtener el gradiente del VQC para la QRNN. De todas formas, estos autores tienen otra forma de realizar la optimización.

Empezamos calculando el gradiente del mapa de características cuántico, que escribimos en la ecuación (31). Tendremos dos casos, uno de ellos será cuando nuestra entrada sea una constante y la derivada respecto a esta entrada no sea necesaria. El segundo caso que nos podemos encontrar es que la entrada sea la salida de pasos previos y por tanto sea necesaria calcular la derivada respecto a esta entrada para poder continuar con el proceso de gradiente. En este último caso podemos incluir una puerta R_y, cuyo parámetro serán

los valores de la entrada, quedando el problema reducido al primer caso.

$$\frac{\partial \langle \psi^o | Z^{\otimes N} | \psi^o \rangle}{\partial \theta_j} = \frac{1}{2} \left(\langle \psi^i | U^\dagger \left(\theta_j^+ \right) Z^{\otimes N} U \left(\theta_j^+ \right) | \psi^i \rangle - \langle \psi^i | U^\dagger \left(\theta_j^- \right) Z^{\otimes N} U \left(\theta_j^- \right) | \psi^i \rangle \right) \tag{32}$$

Donde $\theta_j^{\pm}$ varía el j-ésimo parámetro del vector θ en $\pm\frac{\pi}{2}$. Pongamos que la función pérdida de nuestro modelo es $\mathbf{F}(\boldsymbol{\theta}) = \mathbf{f}\left(\langle \boldsymbol{\psi^o}| \mathbf{Z}^{\otimes \mathbf{N}} |\boldsymbol{\psi^o}\rangle\right)$, con f una función pérdida cualquiera. Una vez tenemos la función diferenciada solo queda usar la regla de la cadena sucesivamente para obtener el valor optimizado de los pesos, como ya se vio anteriormente.

Puesto que estamos tratando con redes híbridas, lo ideal será usar un método que trate diferente las capas cuánticas de las clásicas, usando los métodos más eficientes para cada caso. De esta forma el proceso de aprendizaje es sencillo ya que podemos aplicar el gradiente en descenso de forma corriente, pero cuando tengamos que optimizar estados cuánticos podemos usar la técnica vista antes, donde obtenemos la derivada del mapa de características cuántico mediante el uso del método de las diferencias finitas. Hay otras formas de hacer esto, tal y como vimos para las QRNN, pero estos investigadores decidieron de hacerlo así.

6.1.1. Aplicaciones

Una vez aclarada la forma de diferenciar los estados cuánticos, podemos integrar el método de gradiente clásicos con el cuántico. Usando, por ejemplo, descenso en gradiente estocástico, podemos crear un optimizador que se encargue de obtener el valor de la función pérdida, y vaya actualizando los pesos tanto del VQC, cómo de la red clásica. Estos mismos investigadores haciendo uso de simuladores de ordenadores cuánticos, decidieron probar su modelo con la librería de prueba **Tetris** que contiene imágenes de pequeño tamaño. Sin entrar en mucho detalle de como entrenaron esta red, obtuvieron los resultados vistos en las **figuras 23 y 24**.

En las gráficas presentadas, la linea azul representa el resultado con una capa clásica CNN, mientras que la línea negra sería la correspondiente a dos capas CNN. La línea azul a rayas representa una capa de la QCCNN vista anteriormente, siendo la línea negra a rayas la relativa a dos capas QCCNN. La **figura 23a** y **23c** hace referencia a la precisión y función pérdida respectivamente para un conjunto de entrenamiento usando 2 clases en la clasificación, mientras que la **figura 23b** y **23d** es la correspondiente a 4 clases.

A primera vista podemos ver cómo el uso de QCCNN devuelve unos mejores resultados, incluso una vez ha convergido la precisión, se obtiene valores menores para la función

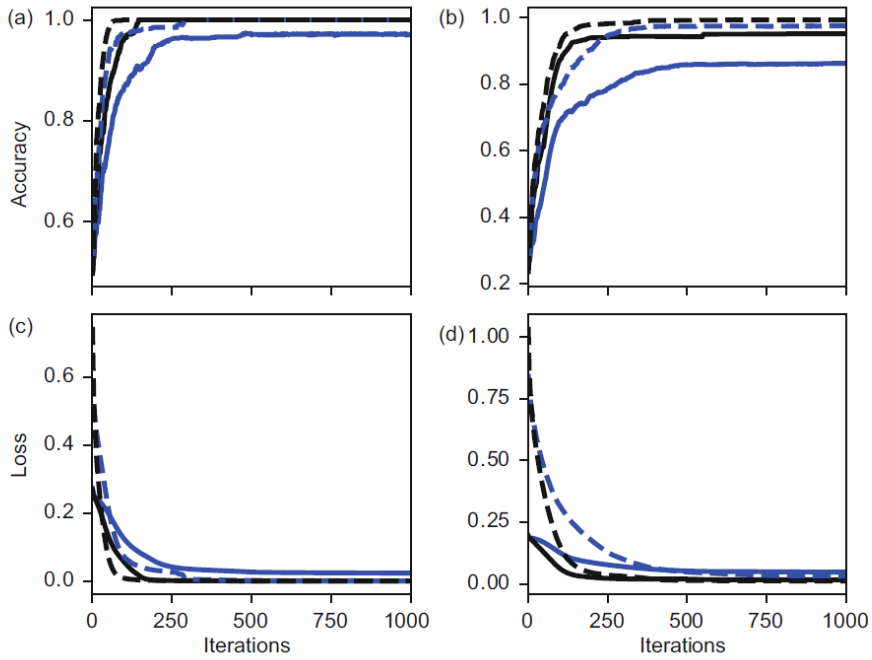


Figura 23: *Resultados Tetris [16].*

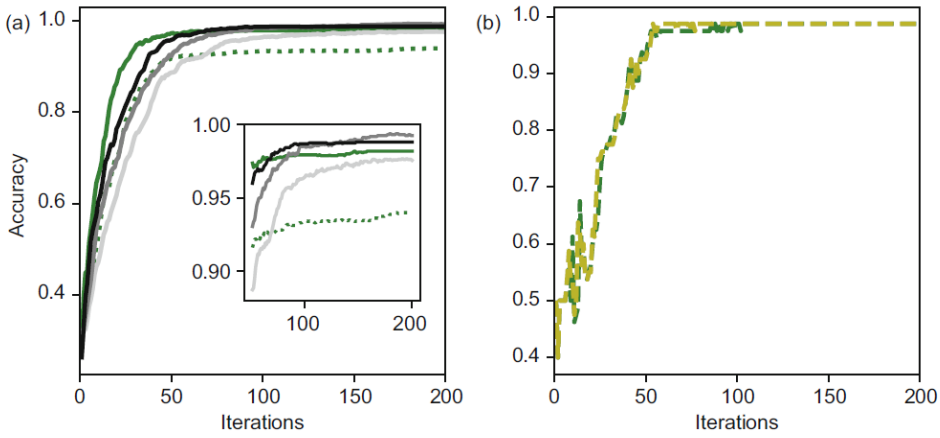


Figura 24: *Resultados Tetris 2 [16].*

pérdida, lo que demuestra una mayor profundidad en el aprendizaje. No solo eso, sino que la velocidad de convergencia es superior al modelo clásico en este caso. Para ver hasta dónde llegan las ventajas de este modelo, los investigadores realizaron otra comparativa que podemos ver en la **figura 24**.

En esta nueva prueba se probaron modelos diferentes, así se muestran QCCNN con 1, 2 y 4 dimensiones de profundidad en el mapa de características cuánticas, siendo respecti-

vamente las líneas de color gris más claro a más oscuro. Luego para la CNN de dos capas tenemos la línea de puntos verde. La línea sólida de color verde representa la precisión para una CNN de dos capas modificada donde la primera capa convolucional se le ha añadido una dimensión de *padding*.

Luego en la **figura 24** tenemos la línea a rayas verde que representa una QCCNN de una capa con ruido cuántico, pues fue simulado el efecto que tendría su aprendizaje en ordenadores cuánticos reales, más detalles de cómo se simuló este ruido se puede encontrar en el artículo [16]. La línea de color amarillo a rayas representa un entrenamiento ideal de la misma QCCNN.

La conclusión que podemos extraer de estos resultados es que estas redes híbridas pueden potencialmente obtener mejores modelos. El problema cómo siempre viene del tamaño del sistema, así para un tamaño de ventana **N**, el mapa de características cuántico requiere $\mathbf{O(Nd)}$ aplicaciones de puertas cuánticas, con d la dimensión de dicho mapa, mientras que en el caso clásico son necesarias $O(N)$ operaciones. Dicho esto, es evidente que las QCCNN no presentan una mejora en cuanto a eficiencia en el número de operaciones. Pueden existir otras codificaciones que ayuden a disminuir el número de puertas necesarias, pero hasta ahora no hay ningún candidato que no comprometa la calidad del modelo.

6.1.2. Análisis rayos X

Hasta ahora hemos visto aplicaciones limitadas al entorno académico, pues las imágenes usadas son para comprobar cómo de bueno es nuestro modelo, pero *Essam H. et al.* [12], crearon una arquitectura QCCNN para diferenciar a partir de radiografías de pecho, si el paciente tenia COVID-19. Así, los autores llaman a este modelo **HQCNN** (*hybrid quantum convolutional neural network*). No entraremos a describir la arquitectura usada pero no se aleja de la vista anteriormente. Más detalles de esta pueden verse en [11] y [12].

El objetivo central del artículo expuesto era realizar una comparativa entre las capacidades de los modelos clásicos de CNN y esta HQCNN a la hora de reconocer pacientes de COVID haciendo uso exclusivamente de imágenes por rayos X de la zona pectoral: **CXR**. Una diferencia clave es el uso de puertas de rotación cuánticas para codificar la información clásica en estados cuánticos. También se han usado capas convolucionales clásicas junto a la capa cuántica.

El conjunto de datos del entrenamiento consiste en imágenes correspondientes a CXR

de casos de neumonía, CXR normales, y otras CXR para casos de COVID-19. De esta forma se divide el conjunto en tres muestras D1 tomando imágenes normales y con COVID, D2 tomando COVID y neumonía, y por último D3 con todo tipo de imágenes. Los resultados para cada conjunto de datos comparados con otros modelos de CNN se pueden ver en el propio artículo, mostrando una clara mejoría del sistema híbrido respecto a los modelos clásicos. Mientras que en la siguiente imagen se hace una comparativa entre otros estudios específicos, donde se crean modelos a la carta para este caso concreto, mientras que la HQCNN está generalizada. Las referencias a los resultados de estos otros estudios se pueden consultar en el artículo.

Study	COVID-19 Images	Class	ACC.(%)	Sns.(%)	F1-measure(%)
MobileNetV2	224	multi	96.7	98.66	-
ResNet18	180	multi	88.9	85.9	84.4
Truncated Inception-Net	162	multi	99.9	98	98
DarkNet	125	binary	98.08	95.13	96.51
DarkNet	125	multi	87.02	85.35	87.37
CNN+LSTM	1512	binary	99.4	99.3	99.9
CoroNet	280	binary	99	99.3	98.5
CoroNet	280	multi	95	96.9	95.6
VGG16	250	binary	96	96	94
GSA-DenseNet121	99	binary	98.38	98.5	98
Inception+FO-MPA	200	binary	98.7	-	98.2
Inception+FO-MPA	219	binary	99.6	-	99
CapsNet	1050	binary	97.24	97.42	97.24
CapsNet	1050	multi	84.22	84.22	84.21
EfficientNet	504	binary	99.62	99.63	99.62
EfficientNet	504	multi	96.70	96.69	97.11
HQCNN	1161	binary	99	99.7	99.3
HQCNN	1161	multi	88.6	88.7	88.8

Figura 25: *Comparativa de modelos para detección de COVID [12].*

Las siglas **ACC.** hacen referencia a la precisión con el conjunto de prueba de cada modelo, mientras que **Sns.** muestra la sensibilidad, esto es, el cociente entre el número de positivos identificados correctamente entre los falsos positivos, sumado a los falsos negativos. Esta es una medida común al evaluar métodos en medicina. Luego, el término F1 es otra métrica muy importante, pero que no entraremos a evaluar. Más información sobre estas métricas se puede ver en el artículo.

De esta forma podemos ver cómo la precisión multiclase de la HQCNN es la más baja comparada con otras investigaciones, lo mismo ocurre con la sensibilidad y F1, donde solamente supera algunos estudios. Sin embargo, si nos fijamos en el modelo de clasificación

binaria, esta supera al resto de investigaciones en cuanto a sensibilidad y F1. Pero para ser justos, estos otros modelos usaron redes con más capas que la HQCNN que fue entrenada usando 9 capas, siendo tres de ellas convolucionales clásicas, y una convolucional cuántica, mientras que las otras redes hicieron uso de arquitecturas verdaderamente profundas y que requieren gran potencia computacional para ser entrenadas en un tiempo razonable. Por ejemplo, una red basada en CNN + LSTM usó 20 capas, con 12 de ellas convolucionales.

Para finalizar, podemos ver en la figura **26** un resumen con los resultados del entrenamiento con los diferentes conjuntos de datos para la HQCNN.

Class	ACC.(%)	Sns.(%)	Prc.(%)	F1-measure(%)
	D1			
normal	97.8	100	98	99
COVID-19	99.3	97	99	98
	D2			
COVID-19	97.5	99	98	98
pneumonia	99.7	99	100	99
	D3			
normal	89.3	77	89	83
COVID-19	97.5	99	98	98
pneumonia	94.6	92	85	88

Figura 26: *Resultados HQCNN tras entrenamiento [12].*

6.2. Comentario sobre redes híbridas recurrentes

Siguiendo la temática del trabajo, ahora debería explicar alguna arquitectura híbrida de redes neuronales recurrentes, pero en este caso no he podido encontrar ningún modelo que cumpla las características híbridas, puesto que a priori no hay ventajas de este modelo sobre los casos clásicos o puramente cuánticos. Ya que es difícil implementar VQC dentro de esta arquitectura. Una posibilidad sería usar algoritmos cuánticos para simular la celda de estado y usando las propiedades cuánticas se podrían evitar los problemas que ya vimos asociados al gradiente. Pero el problema con esto es la coherencia, dificultando la implantación de un modelo de este estilo.

7. Conclusiones

En este trabajo se ha hecho un estudio sobre las redes neuronales, desde su base en ordenadores clásicos, pasando por su equivalente en ordenadores cuánticos, y llegando a ver las posibilidades de combinar ambos tipos de algoritmos. Cómo ya se dijo al comienzo, el interés de este tipo de estudio bibliográfico proviene de la creciente popularidad del *Machine Learning*, y de la constante mejora de los ordenadores cuánticos que permiten ir implementando modelos cada vez más complejos. Pero cómo ya hemos visto, un equivalente de las redes clásicas que son usadas actualmente no parece posible con la tecnología cuántica actual, de todas formas, estos tienen un futuro prometedor, por ello trataré de resumir las ventajas y desventajas que tienen y tendrán las redes neuronales cuánticas tal y cómo han sido descritas en este trabajo.

Esta discusión la dividiré en dos partes, la primera destinada a evaluar las ventajas y desventajas a nivel técnico que presentan la QNN frente a las redes clásicas. En la segunda parte plantearé el posible futuro que le veo a esta disciplina tras haber investigado el actual estado del arte.

7.1. Ventajas técnicas

Las ventajas que podemos encontrar en los sucesivos modelos de QNN provienen de la propia naturaleza de la computación cuántica, la mayoría provienen del uso del entrelazamiento cuántico, que permite reducir enormemente el número de operaciones necesarias para resolver un problema concreto. Además, gracias a las propiedades de estos ordenadores hemos visto cómo el número de qubits necesarios para realizar una misma operación con bits clásicos es menor en el caso cuántico. Aunque realmente esto último plantearía una ventaja real si pudiéramos configurar sistemas con los mismos bits que en los ordenadores clásicos, algo que ahora no es posible, pero en un futuro podría presentar una verdadera revolución para problemas costosos computacionalmente.

Otra ventaja proviene de la naturaleza de los estados cuánticos, en concreto a la hora de resolver sistemas algebraicos, en concreto obtener autovalores, invertir matrices... [24]. Igualmente la razón por la que no hay aplicaciones a gran escala, es que el número de qubits actuales no permite cargar matrices de gran tamaño que son las que suelen ser encontradas al tratar con simulaciones científicas o aplicaciones industriales para resolver problemas numéricos.

Pero volviendo al núcleo de las QNN, la ventaja principal es que aprende más rápido

que su homónimo clásico, y no sólo eso, sino que, como vimos, para algunos casos el valor de la función pérdida sigue disminuyendo después de que en el caso clásico se estabilizará.

7.2. Desventajas técnicas

Más que desventajas en el sentido estricto, es mejor hablar de dificultades técnicas para la implementación de los modelos vistos. El primero, y más importante que dificulta la creación de algoritmos más complejos y extensos, es la coherencia de los qubits. El problema actual de los ordenadores cuánticos es esencialmente este, la dificultad de mantener los qubits sin interaccionar con el entorno. Esto, y otras razones causan el problemático ruido cuántico, que dificulta la implementación de estos algoritmos.

El número de qubits necesarios es prohibitivo para los ordenadores cuánticos actuales y para los que habrá en un futuro cercano. Para poder implementar modelos que sean usados en aplicaciones reales y que supongan verdaderamente una ventaja sobre las redes clásicas, necesitamos tener un número alto de qubits que permitan cargar conjuntos de datos con una alta dimensionalidad, permitiendo generar las relaciones pertinentes entre los datos de entrenamiento.

También existe el problema de la codificación de la información clasíca en cuántica y viceversa, ya que no tenemos ningún método para almacenar estos estados cuánticos, por lo que es necesario generar un sistema que se encargue de la tarea de codificación, generando mayores dificultades a la implementación de los algoritmos.

Por último, y más importante, es que estamos tratando de emular algoritmos diseñados para ordenadores clásicos en ordenadores cuánticos, los cuales funcionan de forma radicalmente diferente a lo que estamos acostumbrando, y tratar de arreglar estas diferencias retocando el algoritmo clásico original nos aleja del verdadero potencial que tiene la computación cuántica.

7.3. Futuro

El futuro de esta disciplina es tremendamente prometedor, y solo estamos viendo la punta de las ventajas que podrían tener para la tecnología implementar QNN a gran escala, en ámbitos del reconocimiento del habla, imágenes, predicciones de modelos físicos, cómo dinámica atmosférica, simulaciones de nuevos materiales, etc. Pero el triunfo de esta tecnología está íntimamente ligada al avance de los ordenadores cuánticos, y cómo hemos visto, el *software* cuántico está muy avanzado, mientras que el *hardware* queda muy atrás.

También es prometedor la posible invención de algoritmos de aprendizaje autónomo de inspiración puramente cuántica, que aproveche al máximo las propiedades de estos ordenadores y dejen atrás las restricciones clásicas. En mi opinión, el verdadero futuro se encuentra en la combinación de ambos tipos de redes, en las que se exploten los puntos fuertes que tiene cada tecnología, llegando a implementaciones flexibles y eficientes, tanto en el ámbito del negocio cómo en la investigación científica.

8. Herramientas para implementar QNN

Para concluir el trabajo me parece adecuado mencionar que uno de lo objetivos iniciales era implementar alguno de los modelos vistos en Python con el fin de mostrar las herramientas que existen para que cualquier persona con conocimientos de programación y algo de computación cuántica pudiera aprender QNN, pero debido a la gran cantidad de información a desarrollar, no me ha sido posible cumplir este objetivo inicial. De todas formas, me parece interesante hacer mención a estas herramientas.

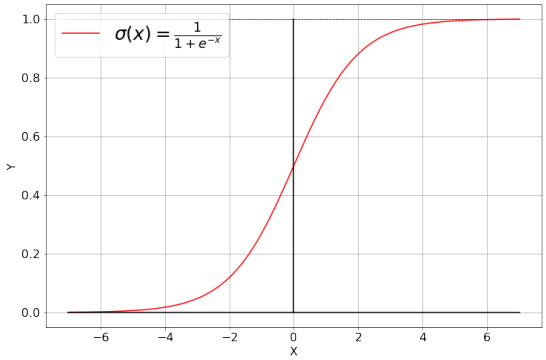
En primer lugar, para la implementación de redes neuronales clásicas, existen diversas librerías en Python, teniendo *Pytorch* [4] y *Tensorflow* [6], que ambas se parecen a *Keras*, la que podríamos llamar precursora de estas.

Luego para implementar circuitos cuánticos, IBM desarrolló la librería *Qiskit* [25] que permite implementar circuitos en forma de código Python, y además permite cargar estos circuitos en ordenadores cuánticos reales alojados en IBM. De forma gratuita se pueden acceder a ordenadores de hasta 5 qubits, que para la mayoría de los proyectos de prueba es suficiente si uno se pone imaginativo.

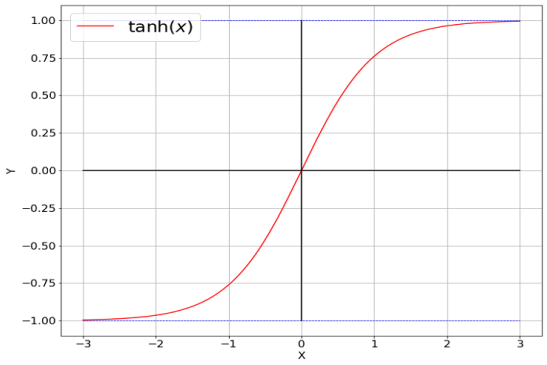
Por último, para la implementación de redes híbridas, existen dos posibilidades, usar *Quantum Tensorflow* que implementa sus propios simuladores de circuitos cuánticos, o bien usar *Qiskit* junto a *Pytorch*, como podemos ver en el sitio web [14]. Siendo realmente interesante esta última posibilidad al poder usar tus propios modelos clásicos, y ver cómo les afectaría introducir alguna capa de carácter cuántico, siendo además posible usar no solo simuladores de alta calidad de IBM, sino ordenadores cuánticos reales.

A. Funciones de Activación

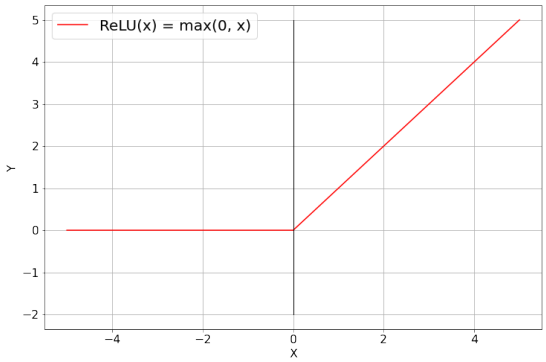
Como ya se ha visto existen diferentes funciones de activación, aquí mostraremos la utilidad y el comportamiento de las más usuales. Cabe mencionar que estas pueden ser modificadas según lo requiera nuestro modelo. Las figuras mostradas han sido generadas con código propio en Python.



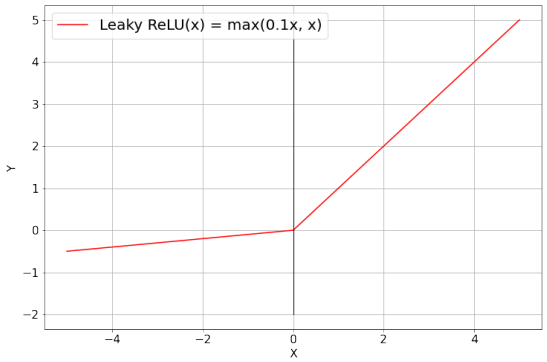
(a) *Función de activación Sigmoide.*



(b) *Función de activación Tangente Hiperbólica.*



(c) *Función de activación ReLU.*



(d) *Función de activación Leaky ReLU.*

Figura 27: *Funciones de activación más importantes.*

La ventaja de la **función sigmoide** es que existe entre 0 y 1, por lo que es útil para modelos donde queremos trabajar con probabilidades.

Otra función muy usada es la **tangente hiperbólica**, que presenta ventajas cuando queremos eliminar aquellos valores cercanos al cero y hacer énfasis en aquellos que tiene valores lejanos al cero.

La función **ReLU** o *Rectified Linear Unit*, es ampliamente usada en todos los modelos de redes neuronales, ya que nos permite normalizar nuestros datos en valores únicamente

positivos, llevando a cero todos los valores negativos.

Una alternativa al ReLU, es el llamado **Leaky ReLU**, el cual soluciona algunos problemas que presenta la ReLU, o si queremos tener en cuenta algunos valores negativos.

Existen numeras funciones según las necesidades que tengamos, pero también debemos tener en cuenta que estas deben ser diferenciadas, y la complejidad de estas puede llegar a determinar la eficiencia de nuestro modelo, por ello es importante tener en cuenta muchas funciones. En la siguiente figura tomada del trabajo de *Sagar Sharma* en [28], podemos ver un resumen.

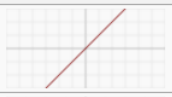
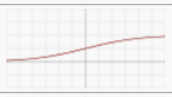
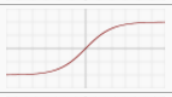
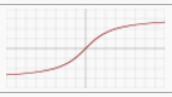
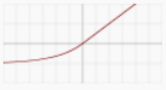
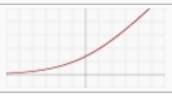
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figura 28: *Resumen de algunas funciones de activación.*

Referencias

[1] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, pages 5–6, 2018.

[2] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, pages 315–338, 2018.

[3] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, pages 271–290, 2018.

[4] Meta AI. Pytorch. https://pytorch.org/.

[5] Johannes Bausch. Recurrent quantum neural networks. https://arxiv.org/abs/2006.14619, 2020.

[6] Google Brain. Tensorflow. https://www.tensorflow.org/.

[7] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, pages 1–6, 2019.

[8] D-Wave. Wave systems: The practical quantum computing company. https://www.dwavesys.com/.

[9] F. D. M. Haldane. Nonlinear field theory of large-spin heisenberg antiferromagnets: Semiclassically quantized solitons of the one-dimensional easy-axis néel state. *Phys. Rev. Lett.*, 50:1153–1156, Apr 1983.

[10] Karen Hallberg. The dmrg method and applications. https://cond-mat.de/events/correl19/talks/hallberg.pdf.

[11] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, and Tristan Cook. Quanvolutional neural networks: Powering image recognition with quantum circuits. https://arxiv.org/abs/1904.04767, 2019.

[12] Essam H Houssein, Zainab Abohashima, Mohamed Elhoseny, and Waleed M Mohamed. Hybrid quantum-classical convolutional neural network model for COVID-19 prediction using chest x-ray images. https://doi.org/10.1093%2Fjcde%2Fqwac003, feb 2022.

[13] IBM. Grover's algorithm. https://quantum-computing.ibm.com/lab/docs/iqx/guide/grovers-algorithm.

[14] IBM. Hybrid quantum-classical neural networks with pytorch and qiskit. https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html.

[15] Yunseok Kwak, Won Joon Yun, Soyi Jung, and Joongheon Kim. Quantum neural networks: Concepts, applications, and challenges. https://arxiv.org/abs/2108.01468, 2021.

[16] Junhua Liu, Kwan Hui Lim, Kristin L. Wood, Wei Huang, Chu Guo, and He-Liang Huang. Hybrid quantum-classical convolutional neural networks. https://doi.org/10.1007%2Fs11433-021-1734-3, aug 2021.

[17] Ismail Mebsout. Convolutional neural networks' mathematics. https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0, Oct 2020.

[18] MIT. Introduction to deep learning: Convolutional neural networks. http://introtodeeplearning.com/, 2021. Accessed: 2022-03-20.

[19] MIT. Introduction to deep learning: Recurrent neural networks. http://introtodeeplearning.com/, 2021. Accessed: 2022-03-26.

[20] Mohammed Zeeshan Mulla. Cost, activation, loss function|| neural network|| deep learning. what are these? https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de, May 2020.

[21] Michael A. Nielsen. The fermionic canonical commutation relations and the jordan-wigner transform, 2005.

[22] Christopher Olah. Understanding lstms. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. Accessed: 2022-03-20.

[23] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2012.

[24] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. https://doi.org/10.1038%2Fncomms5213, jul 2014.

[25] IBM Research. Qiskit. https://qiskit.org/.

[26] Sumit Saha. A comprehensive guide to convolutional neural networks. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, Dec 2018. Accessed: 2022-03-20.

[27] M. Schuld and F. Petruccione. *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Springer International Publishing, 2018.

[28] Sagar Sharma. Activation functions in neural networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, Sep 2017. Accessed: 2022-04-01.

[29] Yuto Takaki, Kosuke Mitarai, Makoto Negoro, Keisuke Fujii, and Masahiro Kitagawa. Learning temporal data with a variational quantum recurrent neural network. *Physical Review A*, 103(5), may 2021.

[30] Maika Takita, Theodore J. Yoder, Oliver Dial, Markus Brink Chow, Jerry, and Robert Davis. Ibm quantum breaks the 100-qubit processor barrier. https://research.ibm.com/blog/127-qubit-quantum-processor-eagle, Feb 2021.

[31] G. Vidal. Class of quantum many-body states that can be efficiently simulated. *Physical Review Letters*, 101(11), sep 2008.

[32] Peter Wittek. *Quantum Machine Learning: What Quantum Computing Means to Data Mining*. Elsevier, 08 2014.