Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Aprendizaje Automático basado en Computación Cuántica

Clasificación de imágenes mediante Kernels Cuánticos

Autor: Javier Martínez Sánchez

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Aprendizaje Automático basado en Computación Cuántica

Clasificación de imágenes mediante Kernels Cuánticos

Autor: Javier Martínez Sánchez
Tutores: Teodoro Calonge Cano
Walter Vinci

Resumen

Recientemente se ha alcanzado la era de los dispositivos NISQ (Noisy Intermediate-Scalce Quantum), dispositivos que, pese a seguir siendo susceptibles al ruido, pueden mejorar sobradamente a los computadores clásicos en ciertas tareas. En este proyecto se trata de obtener una simulación de un modelo que mejora de forma teórica las capacidades de las soluciones clásicas existentes.

Una idea recurrente dentro de esta disciplina es hacer uso de circuitos cuánticos como redes neuronales, debido a la similitud en su estructura. De esta forma se pueden optimizar los parámetros con métodos similares. En este proyecto se ha resuelto un problema de clasificación binaria supervisada de imágenes. Para ello se ha utilizado el método del núcleo o «kernel» cuántico, que es similar a los kernels clásicos pero siendo ejecutado en un dispositivo cuántico ya sea real o simulado. La computación cuántica aporta espacios de Hilbert de alta dimensionalidad donde separar las distintas observaciones del problema. La mejora de la eficiencia teórica con respecto a los métodos clásicos ya ha sido demostrada.

El problema de clasificación resuelto en este caso se trata de la tradicional separación de imágenes de perros y gatos. Se trata de un enfoque híbrido donde una red neuronal convolucional clásica extraerá algunas características de las distintas imágenes que serán evaluadas por un circuito cuántico elevará la dimensionalidad de los datos haciendo la función de kernel para que posteriormente las imágenes serán clasificadas con máquinas de vectores soporte.

El auge de las tecnologías cuánticas ha llevado a algunas empresas a desarrollar librerías gratuitas con las que desarrollar proyectos y simulaciones en computadores cuánticos reales como es el caso de Pennylane desarrollado por Xanadu.

Abstract

Recently, the NISQ (Noisy Intermediate-Scalce Quantum) era devices has arrived, although devices are still susceptible to noise, they can improve on classical computers in certain tasks. The aim of this project is to obtain a simulation of a model that would improve the capabilities of existing classical solutions.

A common idea within this discipline is to use quantum circuits as neural networks, because of the similarity in their structure. In this way, parameters can be optimised using similar methods. In this project, a supervised binary image classification problem has been solved. For this purpose, the quantum kernel method has been used, which is similar to classical kernels, but being executed on a quantum device either real or simulated. Quantum computing provides high-dimensional Hilbert spaces in which is possible to separate the different observations of the problem. The improvement in theoretical efficiency over classical methods has already been demonstrated.

The classification problem solved in this case is the traditional separation of images of dogs and cats. It is a hybrid approach where a classical convolutional neural network extracts some features from the different images that are evaluated by a quantum circuit that raises the dimensionality of the data performing as the kernel function and then, the images will be classified with support vector machines.

The growth of quantum technologies has encouraged some companies to develop free libraries with which to develop projects and simulations on real quantum computers, as in the case of Pennylane developed by Xanadu.

A mis tutores, por guiarme en el mundo de la computación cuántica y en la realización de este proyecto.

A mis padres y abuelos por estar ahí siempre que lo he necesitado.

Índice general

Índice de figuras

Listings

Capítulo 1

Introducción

1.1. Motivación

La búsqueda de la automatización de tareas ha sido sinónimo de avance durante las últimas décadas. Esta permite reducir la carga de trabajo que recae sobre las personas. Generalmente esta automatización se lleva a cabo sobre tareas físicas. No obstante, con las técnicas de inteligencia artificial es posible reproducir hasta cierto punto algunas tareas que requieren de aprendizaje y tradicionalmente desarrollaban los seres humanos.

En el contexto actual donde se generan dos millones y medio de bytes de datos al día y es necesario gestionar inmensos volúmenes de datos, no es factible que las personas realicen todo tipo de tareas sobre los mismos. Algunas de estas tareas pueden ser resueltas mediante técnicas de Aprendizaje Automático. Estas técnicas permiten procesar volúmenes elevados de datos con una precisión aceptable y en un periodo de tiempo inalcanzable para los seres humanos.

A pesar de ser capaces de ser capaces de trabajar con grandes cantidades de datos, existen ciertas operaciones cuyo coste computacional es intratable para estos algoritmos. Bajo el paradigma de la computación cuántica existen varios algoritmos que podrán realizar estas tareas en una cantidad de tiempo asequible aprovechando el paralelismo que ofrece la computación cuántica. No obstante para implementar estos algoritmos de forma realmente eficiente es necesario un computador cuántico.

Actualmente los computadores cuánticos tienen cierta tasa de error experimental. Aunque existen técnicas de software que permiten reducir estos errores, es preferible realizar simulaciones para probar la viabilidad de estos métodos. En estas simulaciones se utiliza un procesador clásico y no tiene la capacidad para simular una gran cantidad de qúbits de forma eficiente.

1.2. Estado del arte

Bajo el paradigma clásico de computación, los sistemas deben distinguir entre dos estados 0 y 1. Sin embargo, los computadores cuánticos deben representar un estado cuántico continuo por lo que se requerirá una mayor precisión para evitar un error elevado. Dicho error puede influir en los resultados de las operaciones incluso si este es pequeño. Además, tal y como se encuentran construidos los ordenadores cuánticos en la actualidad, éstos son susceptibles a ruidos procedentes de sus materiales y de su entorno que alteran su estado.

Para que los computadores cuánticos sean tolerantes a fallos se requiere un gran número de qúbits o bits cuánticos. Para los algoritmos de mayor utilidad diseñados de forma teórica se requerirá un computador cuántico del oreden de un millón de qúbits. [64] Esto se debe a que dichos algoritmos son muy susceptibles al error.

Actualmente se dispone de computadores cuánticos de entre 50 y 100 qúbits los cuales forman parte de la tecnología cuántica ruidosa de escala intermedia (Noisy Intermediate-Scale Quantum o NISQ por sus siglas en inglés). Estos computadores pueden ser útiles en campos como la física cuántica, pero se encuentran lejos de suponer un cambio significativo. [41] Estos dispositivos son suficientemente grandes para mostrar la supremacía cuántica pero demasiado pequeños para ejecutar algoritmos cuánticos sofisticados.

Una idea de utilidad es utilizar técnicas derivadas del aprendizaje automático para establecer los parámetros de un circuito cuántico de forma que este aprenda a resolver tareas. El aprendizaje automático tiene como objetivo diseñar modelos que aprendan de la experiencia previa, sin ser formulados explícitamente. Hay una gran variedad de tareas que estas técnicas pueden abordar: clasificación, agrupamiento predicción, toma de decisiones o reconocimiento de patrones. Con las técnicas y herramientas actuales es posible manejar grandes cantidades de datos pese al gran coste computacional que esto conlleva. A diferencia de los ordenadores clásicos, los ordenadores cuánticos se benefician de los qúbits, que pueden contener combinaciones de 0 y 1 al mismo tiempo. Esto hace que los computadores cuánticos sean potentes a la hora de manejar y procesar grandes tensores, lo que los convierte en sistemas interesantes para la implementación de algoritmos de aprendizaje automático. [42]

Los modelos de aprendizaje automático cuántico tienen una mayor similitud con los métodos de núcleo que con las redes neuronales siguiendo su estructura matemática. Estos métodos analizan datos en espacios de Hilbert de alta dimensión a los que solo tenemos acceso a través de productos internos calculados mediante mediciones. La mayoría de modelos de aprendizaje automático cuántico supervisado pueden ser sustituidos por una máquina de vectores soporte que calcule las distancias entre los estados cuánticos que codifican los datos. Se ha demostrado que el entrenamiento de núcleos cuánticos encontrará un valor igual o más óptimo que el de un circuito variacional. [49]

La codificación de los datos en estados cuánticos tiene un papel fundamental desde la perspectiva de los núcleos cuánticos. La mejora de los algoritmos cuánticos respecto a los clásicos supone por regla general acceder a los datos de forma cuántica. Sin embargo, los núcleos cuánticos suponen una mejora incluso accediendo a los datos de forma clásica. [29]

Un problema estándar de Aprendizaje Profundo, más concretamente relativo a la clasificación de imágenes, es el de la clasificación de conjuntos de datos de fotografías de perros y gatos. Para ello se hace uso de redes neuronales convolucionales para reducir las imágenes a un pequeño número de características que posteriormente serán procesadas por capas densas para obtener una respuesta a este problema de clasificación. Para su entrenamiento se utilizan bucles de optimización como el descenso del gradiente junto a la retropropagación.

1.3. Objetivos y alcance

El objetivo de este trabajo consiste en la elaboración de un modelo de clasificación binaria de imágenes mediante un algoritmo de aprendizaje automático híbrido clásico cuántico. Este modelo debe reproducir la idea de los kernels cuánticos y su símil con las máquinas de vectores soporte utilizadas en aprendizaje automático clásico. También serán utilizados otros métodos relacionados con la inteligencia artificial y el aprendizaje profundo como las redes neuronales convolucionales para tratar el problema de la dimensionalidad presentes en los problemas de clasificación de imágenes.

1.4. Relación con asignaturas del grado

Visto el objetivo del proyecto, es evidente la relación entre el proyecto y la rama de aprendizaje automático estudiada en el grado de ingeniería informática. Las asignaturas donde se tratan los fundamentos de esta disciplina son: Técnicas de Aprendizaje Automático y Minería de Datos.

Capítulo 2

Marco teórico

2.1. Unidades de información cuánticas

Introducción

Así como en computación clásica la unidad de información básica es el bit (que puede tomar los valores 0 y 1) en computación cuántica este lugar lo ocupa el qúbit o «Quantum bit». Un qúbit es un vector en un espacio complejo de dos dimensiones. Análogamente se pueden definir los estados $|0\rangle$ y $|1\rangle$ como:

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix} |1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

Estos estados corresponden a las polarizaciones up y down de un fotón. En computación cuántica se pueden utilizar varias bases pero usaremos ésta ya que es la estándar.

Los qúbits además nos permiten estar entre los dos estados (superposición) de forma que un estado ϕ será denotado como:

$$|\phi\rangle := \alpha |0\rangle + \beta |1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.2}$$

Donde ϕ es una función de onda que debe cumplir la ecuación de Schrödinger y α y β son números pertenecientes al cuerpo de los complejos tal que:

$$|a|^2 + |b|^2 = 1 \tag{2.3}$$

Por tanto, $|a|^2$ es la probabilidad de observar el estado 0 y $|b|^2$ la de observar el estado 1. El orden de los vectores base es arbitrario, pero el estándar es representar $|0\rangle$ como $(1,0)^T$ y $|1\rangle$ como $(0,1)^T$.

Aunque un bit cuántico puede ponerse en infinitos estados de superposición, sólo es posible extraer la información de un único bit cuántico. La razón por la que no se puede obtener más información de un qúbit que de un bit clásico es que la información sólo puede obtenerse mediante una medida. Cuando se mide un qúbit, la medición cambia el estado a uno de los estados base. Como cada medición sólo puede dar lugar a uno de los dos estados, uno de los vectores base asociados al dispositivo de medición dado, así, al igual que en el caso clásico, sólo hay dos resultados posibles. Como la medición cambia el estado, no se puede medir el mismo estado de un qúbit dos veces. Además, los estados cuánticos no pueden clonarse, por lo que no es posible medir un qúbit de dos maneras, ni siquiera indirectamente, por ejemplo, copiando el qúbit y midiendo la copia en una base diferente de la original. [45]

Notación Bra/Ket de Dirac

El espacio de estados de un sistema cuántico está compuesto por posiciones, «momentums», polarizaciones y «spins» entre otros elementos. De las distintas partículas, se modela mediante un espacio de Hilbert de funciones de onda. Para la computación cuántica sólo necesitamos tratar con sistemas cuánticos finitos y basta con considerar espacios vectoriales complejos de dimensión finita con un producto interno que están abarcados por funciones de onda abstractas. Los espacios de estados cuánticos y las transformaciones que actúan sobre ellos pueden describirse en términos de vectores y matrices o en la notación más compacta bra/ket inventada por Dirac [15]. Los «Kets» escritos como $|x\rangle$ definen vectores columna y se utilizan como ya hemos visto para describir estados cuánticos. Por otro lado, los «Bra» escritos como $\langle x|$ denotan vectores columna que son el conjugado transpuesto del «Ket».

Escribir $\langle x|y\rangle$ denota el producto interno de los vectores $\langle x|e|y\rangle$. Unos ejemplos escritos de forma vectorial son:

$$\langle 0|0\rangle = 1, \langle 0|1\rangle = 0, \langle 1|0\rangle = 0, \langle 1|1\rangle = 1 \tag{2.4}$$

Por otra parte la notación $|x\rangle\langle y|$ es el producto externo:

$$|0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} |0\rangle\langle 1| = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} |1\rangle\langle 0| = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} |1\rangle\langle 1| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.5}$$

2.1.1. Propiedades

Un qúbit puede ser representado como una flecha apuntando en un espacio tridimensional esférico. En estado 0 apunta hacia arriba (up) y en estado 1 hacia abajo (down) como un bit clásico.

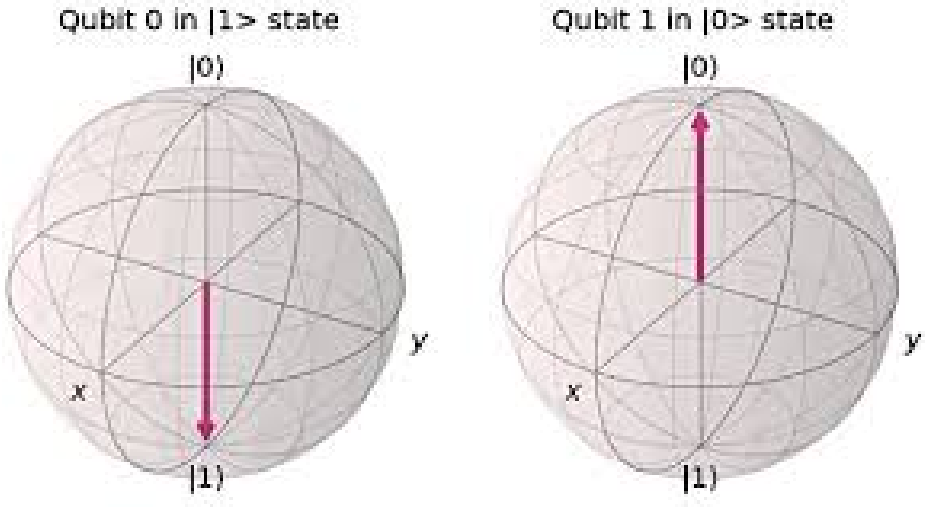


Figura 2.1: Representación de un qúbit [4]

Este espacio se denomina «Esfera de Bloch».

Superposición

Un qúbit también tiene la opción de estar en estado de superposición, que se da en caso de que la flecha apunta hacia cualquier otra dirección. Este estado de superposición es un estado de combinación de 0 y 1.

Al medir un qúbit, ahora la salida que da terminará siendo un 0 o un 1, pero cuál obtienes depende de una probabilidad establecida por la dirección de la flecha. Si la flecha apunta más hacia arriba que hacia abajo, es más probable que obtenga un 0 que un 1, y en caso contrario, es más probable que obtenga un 1 que un 0, y si está exactamente en el ecuador obtendrá cualquiera de los dos estados con un 50 % de probabilidad. Ese es el efecto de la superposición.

Algunos de estos estados que se encuentran en el ecuador frecuentemente utilizados son $|+\rangle, |-\rangle, |i\rangle\, y\, |-i\rangle$ se encuentran definidos como:

$$|+\rangle := \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \quad |-\rangle := \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \quad |i\rangle := \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}}i \end{pmatrix} \quad |-i\rangle := \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}}i \end{pmatrix} \tag{2.6}$$

Entrelazamiento

En una computadora clásica, los bits son independientes entre sí. El estado de un bit no se ve afectado por el estado de ninguno de los otros bits. En las computadoras cuánticas, los qúbits se pueden entrelazar entre sí, lo que significa que se vuelven parte de un gran estado cuántico. Sean dos qúbits que se encuentran en diferentes estados de superposición, pero que aún no están entrelazados, se pueden observar las probabilidades junto a ellos, y estas probabilidades actualmente son independientes entre sí, pero cuando los entrelazamos, sin embargo, tenemos que descartar esas probabilidades independientes y calcular una distribución de probabilidad de todos los estados posibles que podemos obtener: 00, 01, 10 y 11.

Los qúbits están entrelazados, si cambia la dirección de la flecha en un qúbit, cambia la distribución de probabilidad para todo el sistema, por lo que los qúbits ya no son independientes unos de otros, todos son parte del mismo estado. Esto es cierto sin importar cuántos qúbits existan en el sistema. Análogamente a la computación clásica el número de estados posibles de un sistema será de 2^n donde n es el número de qúbits. La diferencia central entre las computadoras clásicas y las computadoras cuánticas. Las computadoras clásicas pueden estar en cualquier estado que desee, pero solo pueden estar en un estado a la vez, mientras que las computadoras cuánticas pueden estar en una superposición de todos esos estados al mismo tiempo.

Interferencia

Para que la propiedad de superposición ya vista sea útil necesitaremos también esta. Aunque en la figura 1 se representa un qúbit mediante la esfera de Bloch, el estado de un qúbit se describe mediante una entidad más abstracta conocida como función de onda cuántica. Las funciones de onda son la descripción matemática fundamental de todo lo relacionado con la mecánica cuántica. Cuando tiene muchos qúbits entrelazados, todas sus funciones de onda se suman en una función de onda general que describe el estado de la computadora cuántica. Esta suma de funciones de onda forman la interferencia porque, al igual que con otro tipo de ondas, cuando las sumamos, pueden interferir constructivamente y sumarse para formar una onda más grande, o destructivamente para cancelarse entre sí. La función de onda general de la computadora cuántica es lo que establece las diferentes probabilidades de los diferentes estados, y al cambiar los estados de diferentes qúbits en particular podemos cambiar las probabilidades de que surjan diferentes estados globales cuando medimos el sistema cuántico.

Aunque la computadora cuántica puede estar en una superposición de millones de estados al mismo tiempo, cuando la medimos, obtenemos un solo estado. Cuando se usa una computadora cuántica para resolver un problema de cálculo, necesita usar interferencia constructiva para aumentar la probabilidad de la respuesta correcta, y usar interferencia destructiva para disminuir las probabilidades de las respuestas incorrectas, de modo que cuando se mida, la respuesta correcta salga con mayor probabilidad. La forma de hacer esto es el ámbito de los algoritmos cuánticos, y toda la motivación detrás de la computación cuántica es que, teóricamente, existen muchos problemas que pueden ser resueltos en una computadora cuántica que a día de hoy son intratables en las computadoras clásicas.

2.1.2. Múltiples qúbits

En computación clásica el estado de un sistema se define a través de los estados de sus componentes. Un aspecto del espacio de estados de un sistema cuántico de n qúbits es que, debido al entrelazamiento, el estado del sistema no siempre puede describirse en términos del estado de sus componentes.

Productos cartesiano y tensorial

Los espacios de estado individuales de n bits se combinan clásicamente a través del producto cartesiano. Los estados cuánticos, sin embargo, se combinan a través del producto tensorial. Sean V y W dos espacios de vectores complejos bidimensionales cuyas bases son $\{v_1, v_2\}$ y $\{w_1, w_2\}$ respectivamente. El producto cartesiano de estos dos espacios se define como la unión de las bases de sus componentes $\{v_1, v_2, w_1, w_2\}$. En particular, la dimensión del espacio de estados de múltiples partículas clásicas crece linealmente con el número de unidades, ya que la dimensión del producto cartesiano es igual a la suma de dimensiones de los factores.

El producto tensorial en los espacios descritos en el párrafo anterior tiene la base $\{v_1 \otimes w_1, v_1 \otimes w_2, v_2 \otimes w_1, v_2 \otimes w_2\}$. El orden de la base es arbitrario. En notación de Dirac se puede expresar de la siguiente forma: $|x\rangle \otimes |y\rangle = |xy\rangle$. Y por tanto la base de un sistema con dos qúbits podrá describirse como: $|00\rangle, |01\rangle, |10\rangle |11\rangle$. En general, un sistema de n qúbits tiene 2^n vectores base. Esto muestra el crecimiento exponencial del espacio de estados con el número de partículas cuánticas.

Los estados que no pueden descomponerse de este modo se denominan estados entrelazados. Estos estados representan situaciones que no tienen una contrapartida clásica. También son estados que muestran el crecimiento exponencial de los espacios de estados cuánticos con el número de partículas. Este es el motivo por el cual incluso la simulación de sistemas cuánticos pequeños en ordenadores tradicionales requiere una gran cantidad de recursos [45].

2.1.3. Medidas

Para obtener una aproximación sobre el estado de un sistema cuántico es necesario realizar medidas del mismo. El resultado de las medidas es probabilístico y el proceso cambia el estado del sistema medido. En el caso de disponer de dos qúbits cualquier sistema puede ser representado como la combinación lineal de los cuatro estados posibles tal que:

$$\begin{aligned} a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle = \\ |0\rangle \otimes (a|0\rangle + b|1\rangle + |1\rangle \otimes (c|0\rangle + d|1\rangle = \\ u|0\rangle \otimes (a/u|0\rangle + b/i|1\rangle + v|1\rangle \otimes (c/v|0\rangle + d/v|1\rangle \end{aligned} \tag{2.7}$$

Al medir el primer bit la probabilidad de que este sea $|0\rangle$ es de $|a|^2 + |b|^2$ y de que sea $|1\rangle$ de $|c|^2 + |d|^2$. En el caso de que sea 0 se proyectará el vector al estado $|0\rangle \otimes (a/u|0\rangle + (b/u|1\rangle$ y en caso contrario al estado $|1\rangle \otimes (c/v|0\rangle + (d/v|1\rangle$.

A efectos de la computación cuántica, la medición de múltiples bits puede tratarse como una serie de mediciones de un solo bit en la base estándar. Son posibles otros tipos de mediciones, como medir si dos qúbits tienen el mismo valor sin conocer el valor real de los dos. Pero este tipo de mediciones son equivalentes a transformaciones unitarias seguidas de una medición estándar de qúbits individuales, por lo que basta con considerar sólo las segundas.

En un sistema de dos qúbits el estado es un producto cartesiano del subespacio de estados con el primer qúbit en $|0\rangle$ y en $|1\rangle$. Cualquier estado puede ser escrito como la suma de dos vectores que se encuentren uno en cada subespacio. Para k qúbits existen 2^k posibles resultados de la medida que será realizada en el espacio del producto cartesiano de los qúbits $S_1 * ... * S_{2^k}$.

Las medidas también nos aportan información sobre el entrelazamiento cuántico de los qúbits. Se dice que no existe entrelazamiento si el hecho de realizar medidas en un qúbit no afecta al estado de otro. Esto sucede si por ejemplo nos encontramos en el estado $\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$ ya que el hecho de medir el primer qúbit y obtener 0 no condiciona las probabilidades de los estados del segundo qúbit.

Por otra parte el estado $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ sí se encuentra entrelazado, ya que obtener 0 en la medición del primer qúbit implica que las pobabilidades de ser $|1\rangle$ del segundo qúbit pasan de ser $1/2$ a ser 0. [45]

2.2. Operaciones cuánticas

Los cambios que se producen en un estado cuántico pueden describirse utilizando el lenguaje de la computación cuántica. Al igual que un ordenador clásico se construye a partir de un circuito eléctrico con cables y puertas lógicas, un ordenador cuántico se construye a partir de un circuito cuántico con cables y puertas cuánticas elementales para transportar y manipular la información cuántica.

Los circuitos de los computadores clásicos están formados por cables y puertas lógicas. Los cables se utilizan para transportar la información por el circuito, mientras que las puertas lógicas manipulan la información, convirtiéndola de una forma a otra [37]. En un sistema cuántico se pueden definir análogamente puertas lógicas cuánticas que realizan operaciones sobre qúbits.

Ya que un estado cuántico puede ser representado como un vector de 2^n elementos, donde n es el número de qúbits, las operaciones pueden ser representadas como matrices de $2^n * 2^n$. Estas matrices al multiplicar el o los qúbits iniciales modificarán en estado cuántico de los qúbits de una determinada forma.

Una ventaja que proporciona la computación cuántica es la posibilidad de obtener la matriz adjunta de dicha operación simplemente transponiéndola y obteniendo el conjugado compuesto de sus elementos. Para que esto sea posible, dichas matrices deben ser unitarias. Esta es la única restricción de las puertas cuánticas. Esto nos permite calcular el estado inicial de un sistema a partir de su salida. Esta propiedad es utilizada en criptografía cuántica para obtener los valores iniciales facilmente.

2.2.1. Puertas de un solo qúbit

Puerta NOT

La puerta NOT en computación clásica se puede definir como $0 \to 1, 1 \to 0$. En el caso cúantico es sencillo determinar cómo debe actuar una puerta NOT en el caso de los estados puros $|0\rangle$ y $|1\rangle$ pero no es tan sencillo cuando el qúbit se encuentra en el estado de superposición. La puerta not en un sistema cuántico actúa de forma lineal tal que:

$$\alpha |0\rangle + \beta |1\rangle \to \alpha |1\rangle + \beta |0\rangle \tag{2.8}$$

De forma que los estados $|0\rangle$ y $|1\rangle$ se intercambian.

La operación NOT se representa mediante la matriz:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.9}$$

Como podemos ver es una matriz complementaria a la de identidad. y por tanto

$$X \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \tag{2.10}$$

Las operaciones en un sólo qúbit se pueden expresar como matrices 2x2, ya que $n = 1$ las cuales deben cumplir la condición de que al ser aplicadas el estado resultante cumpla la condición: $|\alpha|^2 + |\beta|^2 = 1$ para ser unitarias.

Puerta Hadamard

Esta es una de las operaciones más importantes. Convierte el estado $|0\rangle$ en $|0\rangle + |1\rangle)/\sqrt{2}$ y $|1\rangle$ en $|0\rangle - |1\rangle)/\sqrt{2}$. Su expresión es la siguiente:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.11}$$

Una de las propiedades de esta operación es que $H^2 = \mathrm{I}$ por lo que aplicar esta puerta dos veces no cambia el estado del qúbit.

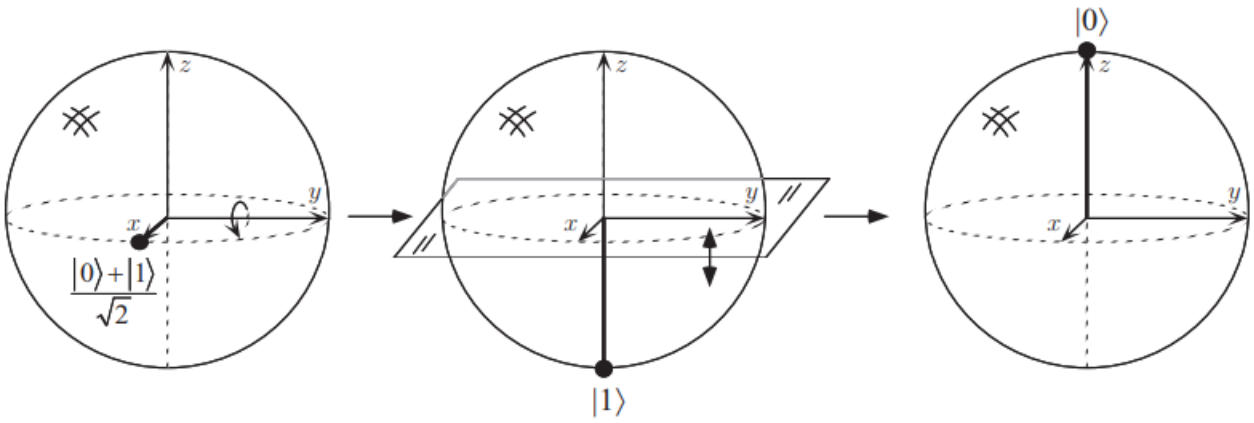


Figura 2.2: Representación de la operación de Hadamard en la esfera de Bloch [37]

Algunos ejemplos son:

$$
\begin{aligned}
|0\rangle &\text{---}\boxed{H}\text{---} |+\rangle \\
|1\rangle &\text{---}\boxed{H}\text{---} |-\rangle \\
|+\rangle &\text{---}\boxed{H}\text{---} |0\rangle \\
|-\rangle &\text{---}\boxed{H}\text{---} |1\rangle
\end{aligned}
\tag{2.12}
$$

Rotaciones

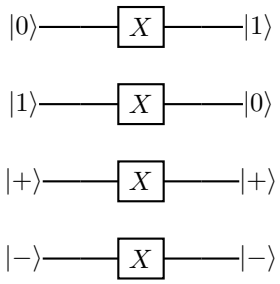
Una de las operaciones más sencillas es la de realizar una rotación sobre la esfera de Bloch.

Puertas de Pauli Estas puertas realizan una rotación de π radianes en cada uno de los tres ejes: X, Y y Z. Las puertas se definen de la siguiente manera:

Puerta de Pauli X Esta puerta realiza una rotación en el eje X de π radianes y se define como:

$$
\boxed{X} := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.13}
$$

Algunos ejemplos son:



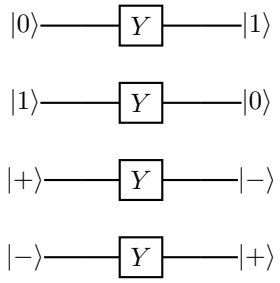
(2.14)

Podemos ver que es igual a la puerta NOT por lo que deducimos que la idea del NOT cuántico es equivalente a realizar una rotación de π radianes en la esfera de Bloch.

Puerta de Pauli Y Esta puerta realiza la rotación en el eje Y:

$$\boxed{Y} := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.15}$$

Algunos ejemplos son:



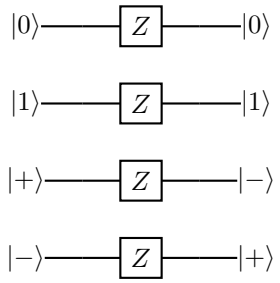
(2.16)

Es importante recordar que los números que determinan el estado de los qúbits pertenecen al cuerpo de los complejos.

Puerta de Pauli Z Por último, esta puerta realiza la rotación en el eje Z:

$$\boxed{Z} := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.17}$$

Algunos ejemplos son:



(2.18)

Puertas de rotación Estas puertas son una generalización de las puertas de Pauli. En esta caso el número de radianes que rota el valor de un qúbit sobre la esfera de Bloch no está fijado a π sino que es un parámetro libre.

Puerta de rotación X Rota α radianes en el eje X.

$$\boxed{R_x(\alpha)} := \begin{pmatrix} cos(\alpha/2) & -i\ sin(\alpha/2) \\ -i\ sin(\alpha/2) & cos(\alpha/2) \end{pmatrix} \tag{2.19}$$

$$|0\rangle \tag{2.20}$$

Puerta de rotación Y Rota α radianes en el eje Y.

$$\boxed{R_y(\alpha)} := \begin{pmatrix} cos(\alpha/2) & -sin(\alpha/2) \\ sin(\alpha/2) & cos(\alpha/2) \end{pmatrix} \tag{2.21}$$

Puerta de rotación Z Rota α radianes en el eje Z.

$$\boxed{R_z(\alpha)} := \begin{pmatrix} e^{-i\frac{\alpha}{2}} & 0 \\ 0 & e^{i\frac{\alpha}{2}} \end{pmatrix} \tag{2.22}$$

Puertas universales

Aunque existen infinitas puertas lógicas en los sistemas cuánticos para un sólo qúbit todas pueden ser expresadas como producto de las rotaciones en tres ejes. Por tanto toda transformación U podrá ser descompuesta tal que

$$U := \begin{pmatrix} \cos(\alpha/2) & -i\sin(\alpha/2) \\ -i\sin(\alpha/2) & \cos(\alpha/2) \end{pmatrix} \begin{pmatrix} \cos(\alpha/2) & -\sin(\alpha/2) \\ \sin(\alpha/2) & \cos(\alpha/2) \end{pmatrix} \begin{pmatrix} e^{-i\frac{\alpha}{2}} & 0 \\ 0 & e^{i\frac{\alpha}{2}} \end{pmatrix} \tag{2.23}$$

Donde α es un número real

$$\boxed{U} := \boxed{X} - \boxed{Y} - \boxed{Z} \tag{2.24}$$

2.2.2. Puertas de varios qúbits

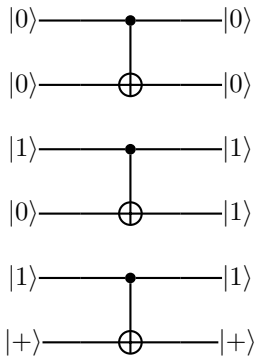
Existen analogías también para las puertas clásicas AND, OR y XOR. En este caso necesitaremos varios qúbits. Cualquier función de bits puede ser expresado mediante una puerta NAND por lo que esta es universal a diferencia de XOR. Esto se debe a que la puerta NAND puede cambiar la paridad de los bits.

Puerta CNOT La puerta CNOT (controlled NOT) usa dos qúbits como inputs: el qúbit de control y el qúbit objetivo. La puerta al igual que en computación clásica actuará sobre el qúbit objetivo cuando el estado del primer qúbit sea $|1\rangle$ mientras que el qúbit de control siempre será estático. La matriz de esta puerta es:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.25}$$

Al ser una operación de dos qúbits la matriz es de $2^2 * 2^2$. Algunos ejemplos de operaciones son:

Algunos ejemplos son:



(2.26)

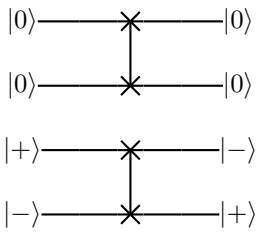
$$|00\rangle \to |00\rangle\,; |01\rangle \to |01\rangle\,; |10\rangle \to |11\rangle\,; |11\rangle \to |10\rangle\,. \tag{2.27}$$

Esta puerta es equivalente a la puerta xor en cierto sentido. No obstante, hay que tener en cuenta que las operaciones NAND y XOR no son reversibles, por lo que el uso de éstas provoca una pérdida de información al no poder obtener las entradas de ésta.

Puerta Swap Esta operación cambia el valor de dos qúbits entre ellos. En computación cuántica no es necesario utilizar métodos como el de la burbuja para conmutar dos valores. Se puede representar mediante la matriz:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.28}$$

Algunos ejemplos son:



(2.29)

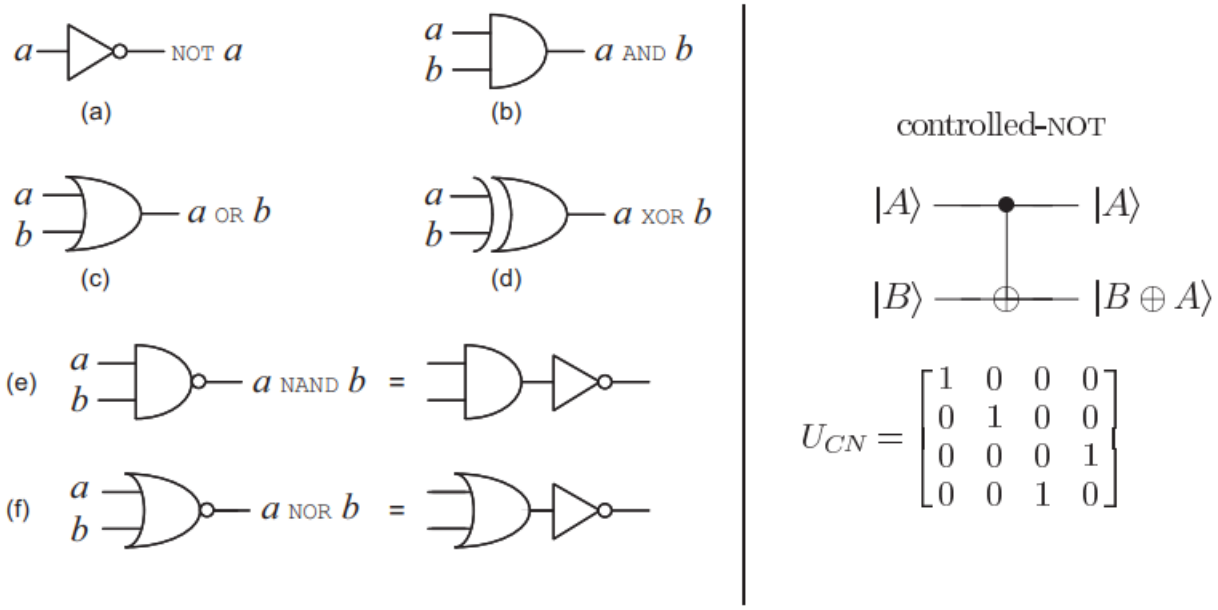


Figura 2.3: Representación de operaciones con qúbits [37]

2.3. Redes neuronales

2.3.1. Introducción

Inteligencia artificial La inteligencia artificial puede ser definida como la rama de la informática que trata con la automatización y el comportamiento inteligente. Los principios de este campo incluyen las estructuras de datos utilizadas en representación del conocimiento, los algoritmos necesarios para aplicarlo y los lenguajes y técnicas de programación utilizados en su implementación [31].

Aprendizaje automático El campo del aprendizaje automático se ocupa de la cuestión de cómo construir programas informáticos que mejoran automáticamente con la experiencia [35]. En este término el aprendizaje implica una mejora en la realización de ciertas tareas que un sistema adaptativo no ha encontrado previamente. Por tanto, el aprendizaje conlleva una generalización de un conjunto de problemas en base a la experiencia del sistema. Un sistema inteligente debe mejorar su capacidad de resolver un problema al realizarlo más veces. Si un sistema es capaz de reorganizar o reformular el conocimiento de una forma más compacta se dice que es capaz de aprender [33].

Aprendizaje Profundo El aprendizaje profundo es la disciplina dentro del aprendizaje automático que estudia algoritmos que intentan modelar abstracciones de alto nivel en datos mediante el uso de estructuras que contienen transformaciones no lineales de los datos con los que se trabaja [8]. Aunque frecuentemente el aprendizaje profundo es considerado algo equivalente a las redes neuronales, en secciones posteriores comprobaremos que hay más estructuras que se ajustan a estas características.

2.3.2. Elementos de las redes neuronales

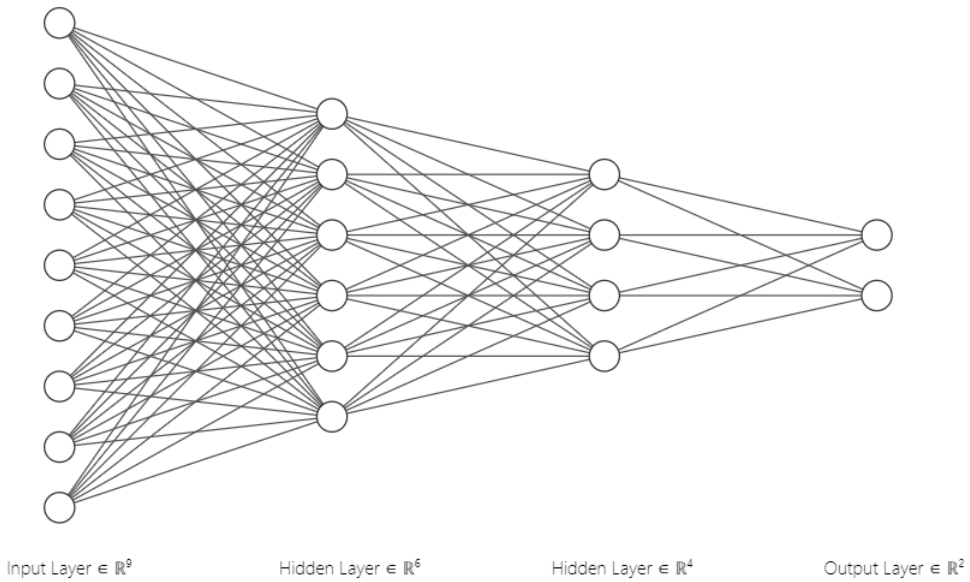


Figura 2.4: Representación de los nodos de una red neuronal. Realizado con la herramienta NN-SVG [28]

Una red neuronal estándar está formada por muchas unidades de procesamiento simples y conectadas llamadas neuronas, cada una de las cuales produce una secuencia de activaciones de valor real. Las neuronas de la primera capa se activan a través de sensores que perciben el entorno. Las siguientes capas de neuronas se activan a través de conexiones ponderadas de neuronas previamente activas [48].

La propiedad más interesante de las redes neuronales es la capacidad de adaptar su comportamiento en función de características cambiantes. Se han realizado estudios sobre métodos para mejorar el rendimiento de las redes neuronales mediante la optimización de los métodos de entrenamiento, el ajuste de los hiperparámetros, los parámetros de aprendizaje, las estructuras de la red y las funciones de activación [55].

Neurona

Una neurona artificial es una analogía de las neuronas biológicas en el campo de la inteligencia artificial [2]. Se define como una función matemática que toma una o más entradas numéricas y un sesgo (representando las dendritas de la neurona biológica), que realiza una suma ponderada de éstas (representando el soma) y transforma dicha suma mediante una función de activación (representando el axón).

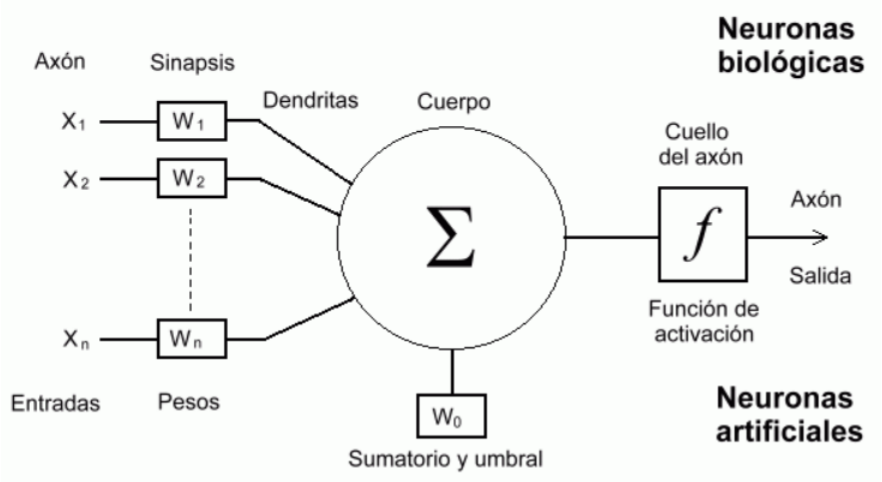


Figura 2.5: Analogía de una neurona biológica. [36]

Funciones de activación

Una función de activación es aquella que realiza una transformación no lineal de la suma ponderada de entradas en una neurona con el objetivo de transmitir la información como salida de dicha neurona. Esta información puede ser utilizada en capas posteriores o como salida de la red.

La precisión de las predicciones de una red neuronal depende del número de capas utilizadas y, sobre todo, del tipo de función de activación empleada. Se ha demostrado que el uso de dos o más capas en una red neuronal reduce el error significativamente [55].

Es importante tener en cuenta que una red neuronal sin funciones de activación o con funciones de activación lineales, se comportará de la misma forma que una regresión lineal ya que su salida estará formada por combinaciones lineales de las entradas (tantas como neuronas en la última capa). Por tanto, el hecho de utilizar más de una capa de neuronas no aportará nada a las predicciones realizadas por la red neuronal.

Función de activación sigmoide Es la función de activación más extendida. Se define como:

$$f(x) := \frac{1}{1 + e^{-x}} \tag{2.30}$$

Acota los valores de salida de una neurona entre 0 y 1.

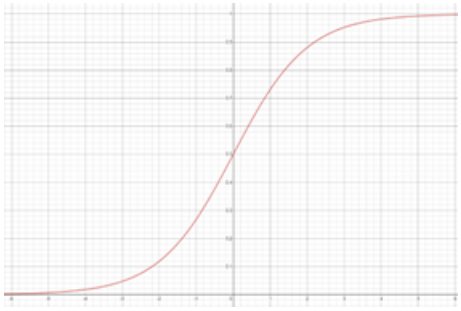


Figura 2.6: Función sigmoide

Es una función derivable infinitas veces, lo cual supone una importante ventaja. Por otro lado, el hecho de que el signo de la salida sea necesariamente positivo puede llegar a ser un problema.

Función de activación tangente hiperbólica La función TANH es similar a la sigmoide, pero está acotada entre -1 y 1 y es simétrica respecto al origen. Se puede definir como:

$$tanh(x) := 2 * \frac{1}{(1 + e^{-2x})} - 1 \tag{2.31}$$

Esta función es continua y diferenciable.

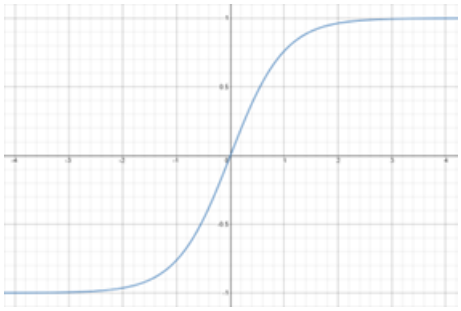


Figura 2.7: Función tangente hiperbólica

Función de activación ReLU La función ReLU (rectified liner unit en inglés) está definida por:

$$f(x) = max(0, x) \tag{2.32}$$

La ventaja de esta función de activación es la posibilidad de que una neurona no se active (en las anteriores solo pasaba si la suma ponderada es 0).

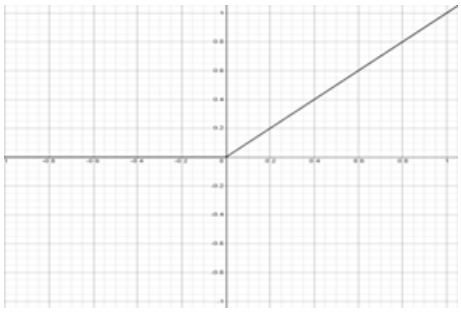


Figura 2.8: Función de activación Rectifier Linear Unit

En ocasiones el gradiente de estas funciones es 0 en cuyo caso los parámetros no se actualizan en el algoritmo de retropropagación.

Función de activación softmax La función softmax es una combinación de funciones sigmoide especialmente utilizadas para predicción multiclase. En un problema de K clases para cada clase j se define como:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.33}$$

La ventaja de esta función es que devuelve directamente la probabilidad de pertenencia a una clase [55].

2.3.3. Feed-forward Network

Perceptrón de una capa

Un perceptrón de una capa o perceptrón paralelo es una capa de un determinado número de neuronas sin ninguna conexión entre sí. Su salida está compuesta por distintas combinaciones lineales de las entradas [5].

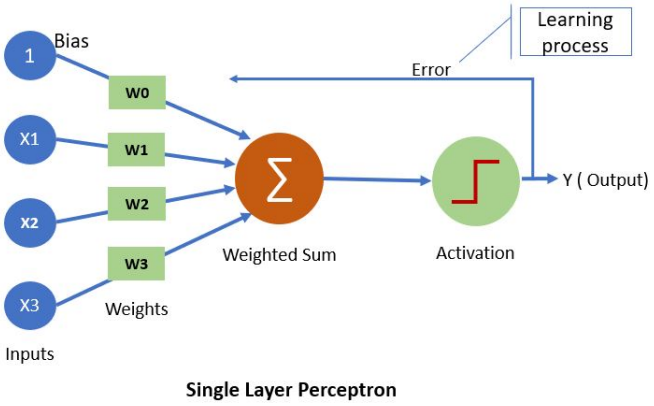


Figura 2.9: Perceptrón de una capa [6]

Perceptrón multicapa

En este modelo existen varias capas de neuronas. Las salidas de las primeras capas son utilizadas como entradas de las siguientes y las últimas proporcionan la salida de la red. Las capas que se encuentran en el

medio se denominan capas ocultas.

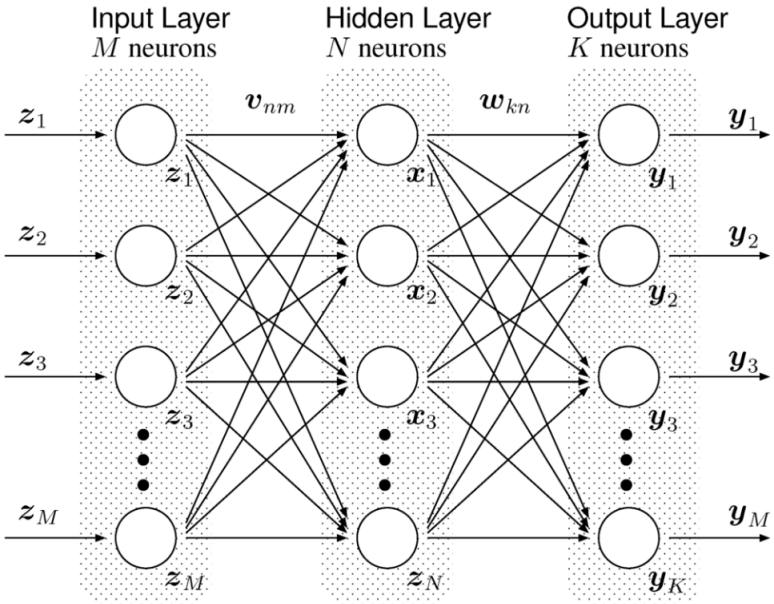


Figura 2.10: Perceptrón multicapa [22]

Es importante que la salida de las neuronas de cada capa pase por una función de activación antes de ser utilizada como entrada de la siguiente capa.

2.3.4. Entrenamiento

En el entrenamiento de este tipo de redes no se modifica la estructura de las mismas. Para mejorar las predicciones se actualizan los pesos de las neuronas de la red. Los pesos óptimos para la red son aquellos que minimizan la función de coste.

Función de coste o pérdida Esta función trata de medir el error de predicción de un modelo, es decir, la diferencia entre los valores reales y los predichos. Esta función debe ser minimizada para optimizar dicho modelo. Algunas funciones de coste son:

- RMSE

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{2.34}$$

- MAE

$$MAE = \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{n} \tag{2.35}$$

- Entropía cruzada

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} log(p_{ij}) \tag{2.36}$$

Descenso del gradiente El descenso del gradiente es un algoritmo que estima los parámetros de una función que la minimizan. Este algoritmo no se realiza mediante cálculo simbólico sino que utiliza una aproximación numérica. Este proceso solo necesita de las salidas de la función.

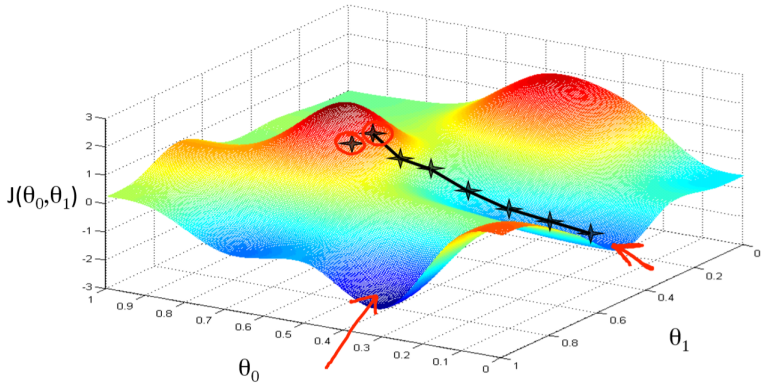


Figura 2.11: Pasos del algoritmo de descenso del gradiente en una función J [59]

Este algoritmo analiza el gradiente de la función en un punto y «se dirige» hacia donde el valor de la función decrezca más rápidamente. Un problema de este algoritmo es la posibilidad de caer en un mínimo local si la función de pérdida es no convexa.

Retropropagación Para aplicar el descenso del gradiente es necesario conocer el vector gradiente de la función de pérdida. Para calcularlo en redes neuronales de forma eficiente será necesario aplicar este algoritmo. Este algoritmo funciona únicamente en problemas de aprendizaje supervisado.

El algoritmo de retropropagación consiste en encontrar las derivadas de la función de coste con respecto a cada uno de los parámetros de una neurona y extender el cálculo a las neuronas de la capa anterior en función de los pesos que conecten ambas neuronas. Esto hace posible determinar el grado de implicación de la neurona en un resultado erróneo y así actualizar los pesos de forma conveniente.

2.4. Redes neuronales convolucionales

Una de las redes neuronales más populares es la red neuronal convolucional (CNN por sus siglas en inglés). Toma este nombre de la operación matemática lineal entre matrices llamada convolución. Las CNN tienen varias capas: la capa convolucional, la función de activación, la de «pooling» y la densamente conectada. Las capas convolucionales y densamente conectadas tienen parámetros, pero las capas de «pooling» y las funciones de activación no. La CNN tiene un excelente rendimiento en problemas de aprendizaje automático, especialmente las aplicaciones que tratan con datos de imágenes, como el gran conjunto de datos de este proyecto, la visión por ordenador, y en el procesamiento del lenguaje natural (NLP por sus siglas en inglés) [1].

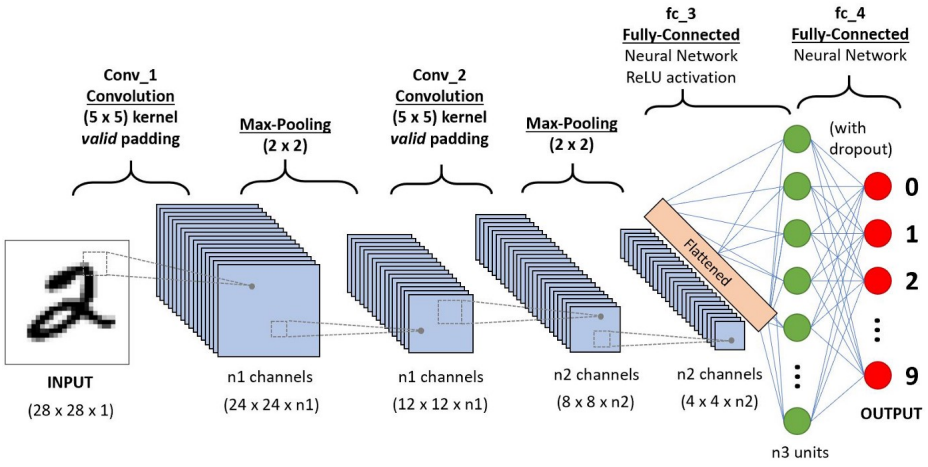


Figura 2.12: Representación de una red neuronal convolucional [47]

2.4.1. Convoluciones

Para entrenar una red neuronal con conjuntos de datos de muchas características (como imágenes o vídeos) se necesitarían muchas conexiones en la primera capa. Como entrenar tantos parámetros no es eficiente se pueden utilizar operaciones de convolución como alternativa.

Una posibilidad sería alimentar cada neurona de entrada solo con una parte de la imagen. De esta forma se reduciría el número de conexiones, pero sería insuficiente. También se podrían fijar los pesos para las primeras neuronas de forma que no fuera necesaria su actualización, esto es similar a tomar regiones contiguas de la imagen no disjuntas. Al no ser disjuntas se pueden extraer características aunque la característica a extraer se encuentre entre dos regiones. A esta operación se le denomina convolución. Además de esta operación se puede añadir un filtro a la imagen que puede ser utilizado para la detección de bordes. En una red convolucional los parámetros de la matriz de filtro se entrenan de forma similar a los pesos de las neuronas.

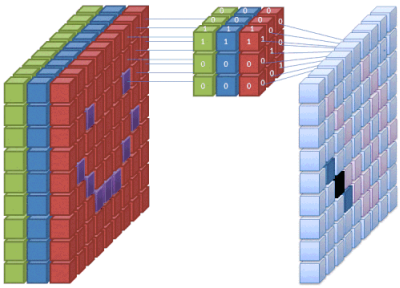


Figura 2.13: Convolución con los tres canales de color [62]

2.4.2. Stride

Las redes neuronales convolucionales tienen más opciones que proporcionan un sinfín de oportunidades para incluso disminuir más los parámetros y al mismo tiempo reducir algunos de los efectos de esta reducción. Una de estas opciones es el «stride».

Para controlar el solapamiento producido en las operaciones convolucionales al mover la «ventana» sobre la que se aplica el filtro se puede aplicar esta técnica. El «stride» es el número de píxeles que avanza la ventana cada vez que se realiza una convolución. Al aumentar el stride generaremos menos imágenes con solapamiento.

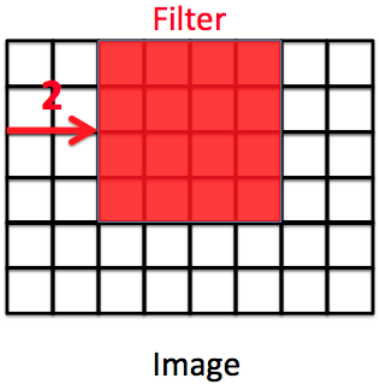


Figura 2.14: Stride de dos píxeles [12]

2.4.3. Relleno

Uno de los inconvenientes del uso de convolución es la pérdida de información que puede existir en el borde de la imagen. Como algunas características sólo se capturan cuando el filtro se desliza, nunca pueden ser detectadas. Un método muy sencillo, pero eficaz, para resolver este problema es utilizar el relleno de ceros. El relleno de ceros consiste en añadir ceros a los bordes de los mapas de características extraídos al aplicar los filtros. La otra ventaja del relleno de ceros es la capacidad de gestión que nos proporciona.

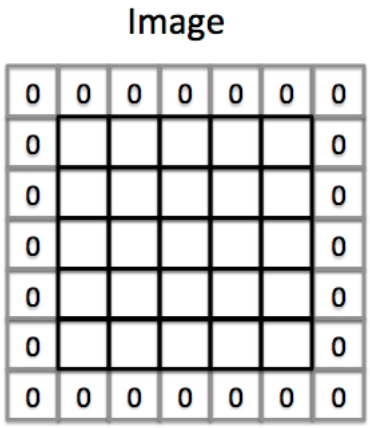


Figura 2.15: Relleno de ceros [11]

2.4.4. Peso compartido

El reparto de pesos aporta homogeneidad a las características del modelo. Ayuda a filtrar la característica aprendida independientemente de las propiedades espaciales. Al iniciar valores aleatorios para los filtros, éstos aprenderán a detectar los bordes si mejora el rendimiento. Es importante tener en cuenta que si necesitamos destacar que algo es espacialmente importante en la entrada, entonces no es recomendable usar un peso compartido.

2.4.5. Pooling

La idea principal del «pooling» es la simplificación de muestras para reducir la complejidad de las capas posteriores. En el ámbito del procesamiento de imágenes, puede considerarse similar a reducir la resolución. El «pooling» no afecta al número de filtros.

El «Max-Pooling» realiza una partición de una región en rectángulos y devuelve únicamente el máximo valor dentro de la región. El «stride» utilizado suele ser 2 aunque puede ser 1 para evitar la reducción de muestras.

Hay que tener en cuenta que al reducir la muestra no se conserva la posición de la información. Por lo tanto, debe aplicarse sólo cuando la presencia de la información es más importante que la información espacial. Además, el «pooling» puede utilizarse con filtros y «strides» diferentes para mejorar la eficacia. Por ejemplo, un «max-pooling» de 3 por 3 con «stride» 2 mantiene algunos solapamientos entre las regiones.

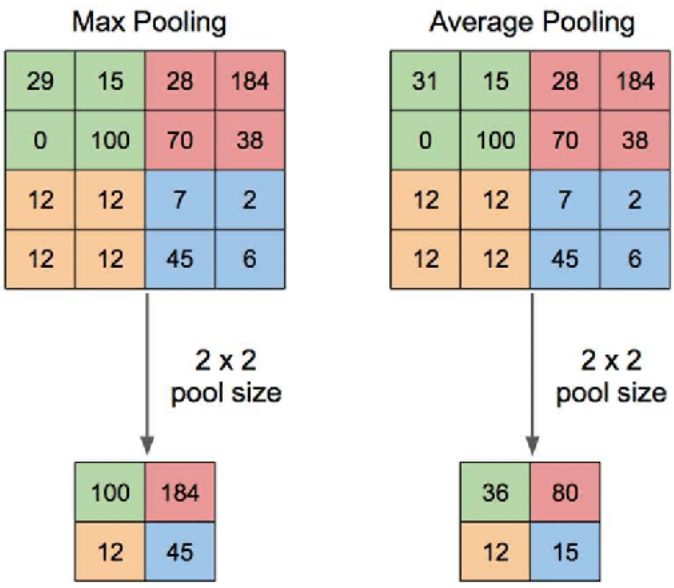


Figura 2.16: Ejemplos de max-pooling y average-pooling [44]

2.4.6. Capas densamente conectadas

Las capas totalmente conectadas son similares a la forma en que se disponen las neuronas en una red neuronal tradicional. Por lo tanto, cada neurona de una capa totalmente conectada está directamente conectada a cada neurona tanto de la capa anterior como de la siguiente. Cada uno de los nodos de los últimos cuadros de la capa de «pooling» están conectados a la primera capa densa. Estos son los parámetros más utilizados con la red neuronal convolucional dentro de estas capas, y requieren mucho tiempo de entrenamiento.

El mayor inconveniente de una capa totalmente conectada es que incluye muchos parámetros que necesitan un cálculo complejo con las instancias del entrenamiento. Los neuronas y conexiones pueden ser eliminadas utilizando la técnica de «dropout» [1].

2.4.7. Técnicas de aumento de datos

El aumento de datos o «data augmentation» es la técnica de aumentar el tamaño de los datos utilizados para el entrenamiento de un modelo. Para obtener predicciones fiables, los modelos de aprendizaje profundo suelen necesitar muchos datos de entrenamiento, que no siempre están disponibles. Por lo tanto, los datos existentes se aumentan con el fin de hacer un modelo que generalice mejor.

Se aplican ciertas transformaciones a las imágenes con las que se va a entrenar la red y se añaden al conjunto de datos. Aunque el aumento de datos se puede aplicar en varios dominios, se utiliza comúnmente en la visión por ordenador. Algunas de las técnicas de aumento de datos más comunes utilizadas para las imágenes son:

- Escalado: Redimensiona la imagen.
- Recorte: Se selecciona una parte de la imagen.
- Volteado: Se obtiene la imagen girada de forma vertical u horizontal.
- Relleno: Se añaden márgenes a la imagen. Similar a la operación realizada en la red convolucional, pero con la imagen original.

- Rotacion: La imagen se gira con un ángulo que puede ser determinado o aleatorio
- Translacción: La imagen se mueve vertical u horizontalmente
- Tranformaciones de color: Se basa en alterar los colores de los píxeles de las imágenes. Algunas de las posibilidades son:
    - Brillo
    - Contraste
    - Saturación

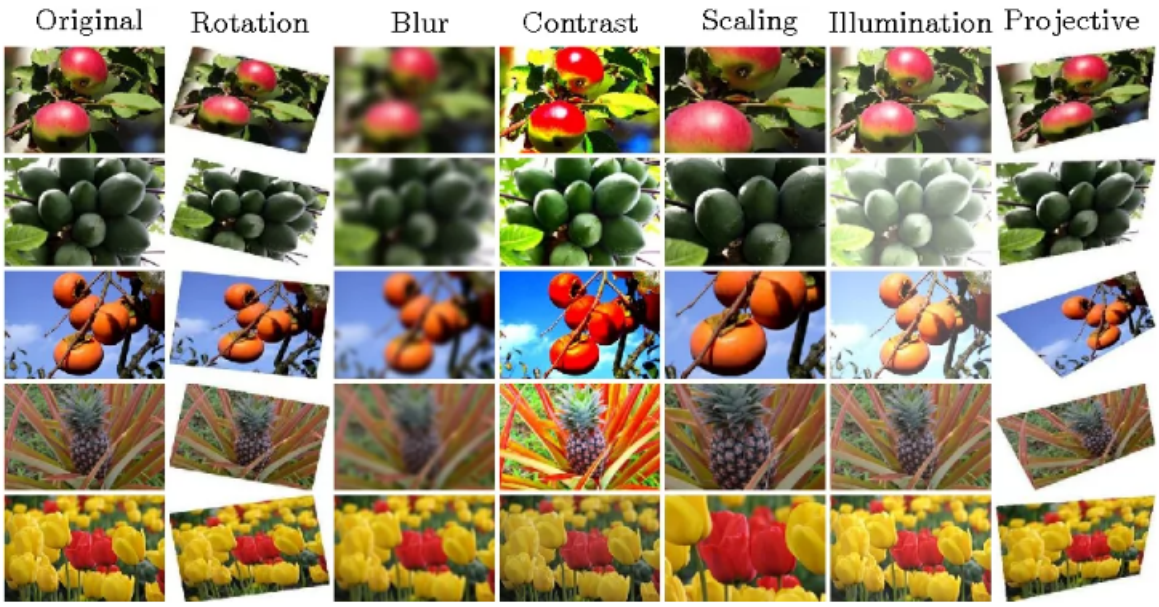


Figura 2.17: Ejemplos de imágenes y los resultados del data augmentation [43]

2.5. Máquinas de vectores soporte

Una máquina de vectores de soporte (SVM) es un algoritmo que aprende mediante el ejemplo a asignar etiquetas a los objetos. Un SVM es una entidad matemática, un algoritmo para maximizar una función matemática concreta con respecto a una determinada colección de datos [38].

Introducción

Una máquina de vectores de soporte es una generalización de un clasificador sencillo e intuitivo llamado clasificador de margen máximo. Este clasificador no puede aplicarse a la mayoría de los conjuntos de datos, ya que requiere que las clases sean separables linealmente. Las máquinas de vectores de soporte están pensadas para problemas de clasificación binaria [23].

2.5.1. Hiperplano

Un hiperplano es un subespacio geométrico de dimensión p-1 definido en un espacio de dimensión p. Un hiperplano está definido por:

$$\beta_0 + \beta_1 X_1 + \beta_{p-1} X_{p-1} = 0 \tag{2.37}$$

Donde los parámetros β_0, $\beta_1 X_1$ y $\beta_{p-1} X_{p-1}$ son los parámetros del hiperplano y X las coordenadas de los puntos pertenecientes a él. Un hiperplano nos permite separar dos regiones del espacio. Un punto pertenecerá a una región si cumple:

$$\beta_0 + \beta_1 X_1 + \beta_{p-1} X_{p-1} < 0 \tag{2.38}$$

o a la otra si cumple:

$$\beta_0 + \beta_1 X_1 + \beta_{p-1} X_{p-1} > 0 \tag{2.39}$$

De esta forma las inecuaciones de arriba determinan de qué lado está cada punto del espacio.

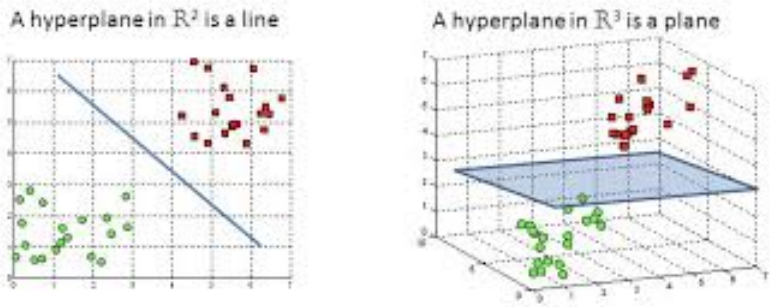


Figura 2.18: Hiperplanos en espacios de dos y tres dimensiones

Si tenemos un conjunto de datos de dimensión $n * p$, podemos asignar cada una de las instancias en función del lado que toman en dicho espacio. Por tanto, nuestro objetivo será determinar un hiperplano que separe el conjunto de datos en las dos clases que se deseen. Si dicho hiperplano existe, y al evaluar la expresión de la ecuación del plano con un punto se obtiene un resultado que dista significativamente de cero, la clasificación tendrá mayor probabilidad de acierto. Por el contrario, si el resultado es cercano a 0 no se tendrá tanta exactitud.

2.5.2. Clasificador

En general, si los datos pueden separarse perfectamente utilizando un hiperplano, entonces existirá un número infinito de estos ya que un hiperplano de separación puede desplazarse o girar ligeramente sin entrar en contacto con ninguna de las observaciones. Para construir un clasificador basado en un hiperplano de separación, se debe tener una forma razonable de decidir cuál de los infinitos hiperplanos de separación posibles utilizar.

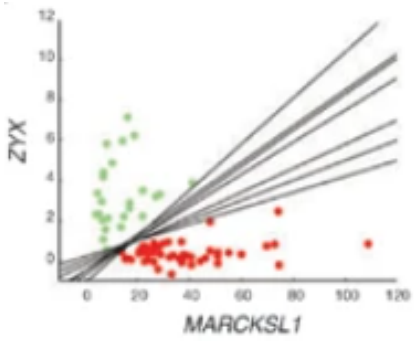


Figura 2.19: Hiperplanos que separan dos conjuntos de puntos [40]

Clasificador de margen máximo

Una elección natural es el hiperplano de margen máximo (también conocido como hiperplano de separación óptimo), que es el hiperplano de separación más alejado de las observaciones de entrenamiento. Es decir, podemos calcular la distancia perpendicular de cada observación de entrenamiento a un determinado hiperplano de separación; la menor de dichas distancias es la distancia mínima de las observaciones al hiperplano, y se conoce como margen. El hiperplano de margen máximo es aquel para el que el margen es mayor, es decir, es el hiperplano que tiene la distancia mínima más lejana a las observaciones de entrenamiento. Es posible clasificar una observación test basándonos en qué lado del hiperplano de margen máximo se encuentra.

Esto se conoce como el clasificador de margen máximo. Esperamos que un clasificador que tenga un gran margen en los datos de entrenamiento tenga también un gran margen en los datos de test y, por tanto, clasifique correctamente las observaciones de test. Aunque el clasificador de margen máximo suele tener éxito, también puede llevar a un sobreajuste cuando p es grande [23].

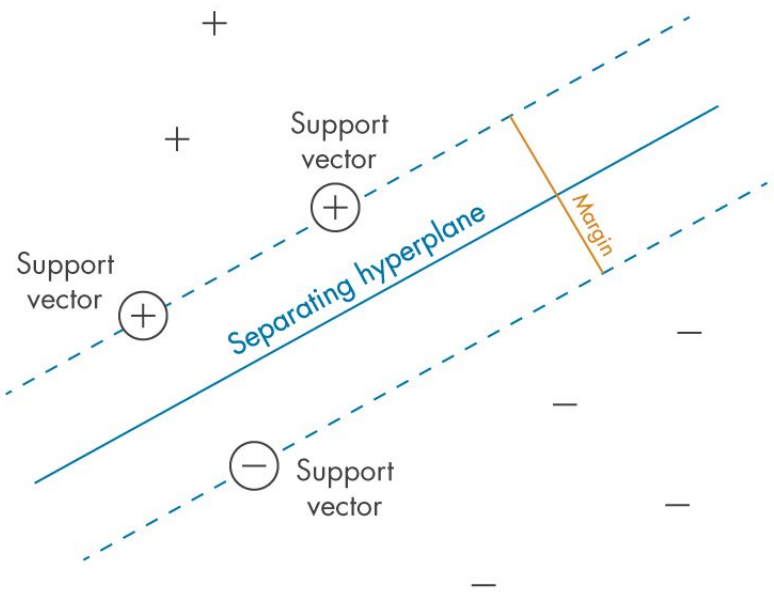


Figura 2.20: Hiperplano de margen máximo [32]

Los puntos más cercanos al plano por ambos lados se denominan vectores soporte. El hiperplano de margen máximo depende directamente de los vectores de soporte, pero no de las otras observaciones: un movimiento de cualquiera de las otras observaciones no afectaría al hiperplano de separación, siempre que el movimiento de la observación no le haga cruzar la frontera establecida por el margen. El hecho de que el hiperplano de margen máximo dependa directamente de sólo un pequeño subconjunto de las observaciones es una propiedad importante.

Construcción El problema de construir el clasificador de margen máximo para un conjunto de n observaciones con p características y una etiqueta de clase (1 o -1 dependiendo de la clase) asociada a cada observación es el de hallar el hiperplano que maximice el margen sujeto a:

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{2.40}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} \geq M \tag{2.41}$$

Esta última restricción garantiza que todas las observaciones estén en el lado correcto.

2.5.3. Caso no separable

El clasificador de margen máximo es una forma de realizar la clasificación, si existe un hiperplano de separación. Sin embargo, en muchos casos no existe un hiperplano de separación, por lo que no hay un clasificador de margen máximo. En este caso, el problema de optimización no tiene solución. Podemos ampliar el concepto de hiperplano de separación para desarrollar uno que separe las clases aproximadamente, utilizando el llamado margen suave. La generalización del clasificador de margen máximo al caso no separable se conoce como clasificador de vectores soporte.

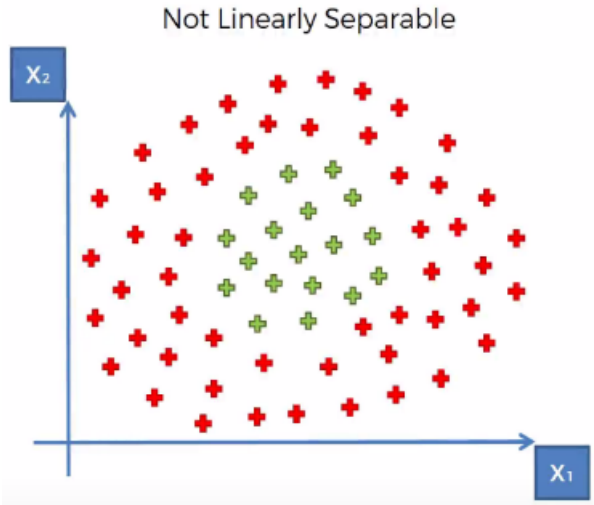


Figura 2.21: Conjunto de datos no separables [27]

Clasificadores de vectores soporte

Las observaciones que pertenecen a dos clases no son necesariamente separables por un hiperplano. De hecho, incluso si existe un hiperplano de separación, hay casos en los que un clasificador basado en un hiperplano de separación podría no ser la mejor opción. Un clasificador basado en un hiperplano de separación necesariamente clasificará perfectamente todas las observaciones de entrenamiento; esto puede conllevar un gran aumento de la sensibilidad a las observaciones individuales. La adición de una sola observación podría conducir a un cambio dramático en el hiperplano de margen máximo. El hiperplano de margen máximo resultante no es satisfactorio ya que este sería demasiado sensible a pequeños cambios. Esto es problemático porque, como se ha comentado anteriormente, la distancia de una observación con respecto al hiperplano puede considerarse una medida de nuestra confianza en que la observación se ha clasificado correctamente. Además, el hecho de que el hiperplano de margen máximo sea extremadamente sensible a un cambio en una sola observación sugiere que se puede haber sobreajustado a los datos de entrenamiento.

En este caso, podríamos estar dispuestos a considerar un clasificador basado en un hiperplano que no separe perfectamente las dos clases, en aras de una mayor robustez de las observaciones individuales, y una mejor clasificación de la mayoría de las observaciones de entrenamiento. Es decir, vale la pena clasificar mal unas pocas observaciones de entrenamiento para hacer un mejor trabajo en la clasificación de las observaciones restantes [23].

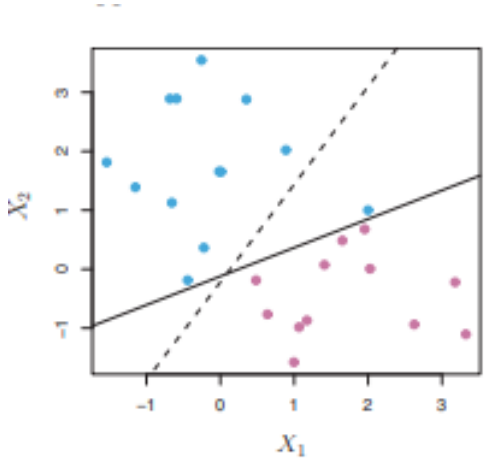


Figura 2.22: Variación del hiperplano separador por el efecto de un punto cercano a la frontera [23]

En el clasificador de vectores de soporte el margen no se respeta en todas las instancias de entrenamiento. En lugar de buscar el mayor margen posible para que cada observación esté no sólo en el lado correcto del hiperplano sino también en el del margen, permitimos que algunas observaciones estén en la parte incorrecta del margen, o incluso en la del hiperplano. La mayoría de las observaciones están en el lado correcto del margen. Sin embargo, en un pequeño subconjunto de las observaciones esto no sucede así.

Una observación puede estar no sólo en el lado equivocado del margen, sino también en el lado equivocado del hiperplano como se ha explicado anteriormente. Las observaciones en el lado equivocado del hiperplano corresponden a las observaciones de entrenamiento que son clasificadas erróneamente por el clasificador de vectores de soporte.

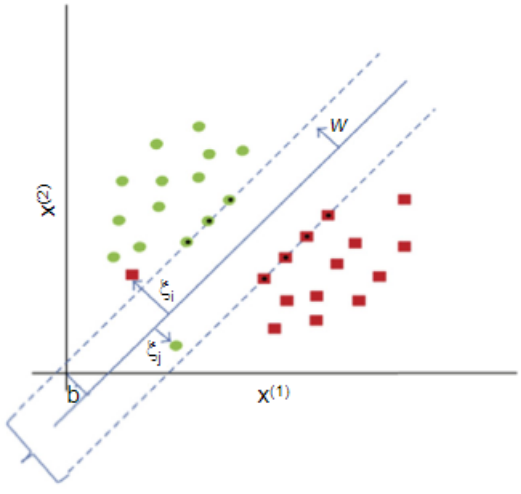


Figura 2.23: Algunas observaciones rebasan el margen y el hiperplano [38]

Actualizando las ecuaciones del clasificador de margen máximo tenemos:

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{2.42}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} \geq M(1 - \epsilon_i) \tag{2.43}$$

$$0 \geq \epsilon_i \geq C \tag{2.44}$$

Donde C es el parámetro de error máximo y ϵ_i las variables de holgura de cada una de las observaciones que están rebasando el margen. Se sigue utilizando el mismo proceso para realizar la clasificación. Si ϵ_i es mayor que 0 significa que la observación i supera el margen, si es mayor que 1 entonces supera también el hiperplano. Por tanto, el número de observaciones que pueden estar mal clasificadas no puede ser mayor que C.

Una propiedad importante del problema de optimización es que solo las observaciones que se encuentran en el margen o que rebasan el margen afectarán al hiperplano y, por tanto, al clasificador obtenido. En otras palabras, una observación que se encuentra estrictamente en el lado correcto del margen no afecta al clasificador del vector de soporte. Cambiar la posición de esa observación no cambiaría el clasificador en absoluto, siempre que su posición siga estando en el lado correcto del margen. Las observaciones que se encuentran directamente en el margen, o en el lado equivocado del margen, se conocen como vectores de soporte. Estas observaciones sí afectan al clasificador del vector de soporte.

El hecho de que la regla de decisión del clasificador de vectores de soporte se base sólo en un pequeño subconjunto de las observaciones de entrenamiento significa que es bastante robusto frente al comportamiento de las observaciones que están lejos del hiperplano [40]. Esta propiedad es distinta a la de otros métodos de clasificación .

Clasificación en espacios no lineales

El clasificador de vectores de soporte es un enfoque natural para la clasificación en el entorno de dos clases, si el límite entre las dos clases es lineal. Sin embargo, es frecuente que en la práctica los límites de clase no sean lineales.

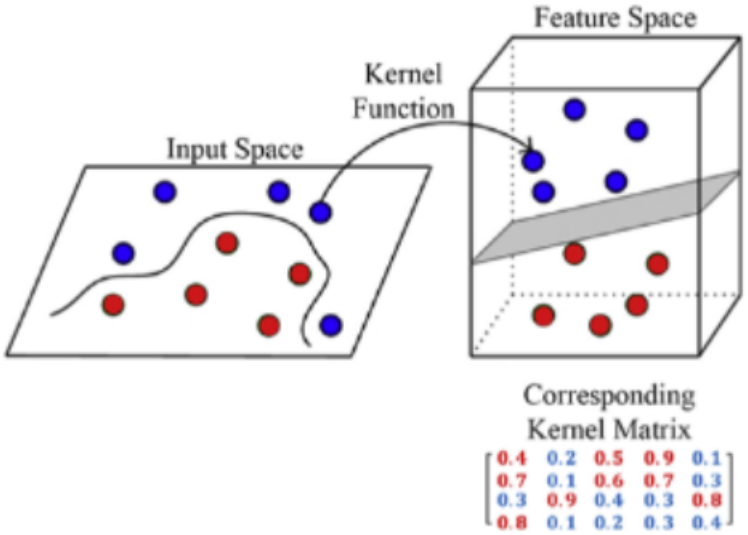


Figura 2.24: Transformación de los datos a un espacio de alta dimensión [38]

El rendimiento de la regresión lineal puede verse afectado cuando existe una relación no lineal entre los predictores y el resultado. En ese caso, consideramos la posibilidad de ampliar el espacio de características utilizando funciones de los predictores, como los términos cuadráticos y cúbicos, para abordar esta no linealidad. En el caso del clasificador de vectores soporte, podríamos abordar el problema de los límites no lineales entre las clases de forma similar, ampliando el espacio de características mediante funciones cuadráticas, cúbicas o polinómicas. Por ejemplo, en lugar de ajustar un clasificador de vectores de soporte utilizando p características, podríamos ajustar un clasificador de vectores de soporte utilizando 2p características [23]. En este caso las ecuaciones que se han visto anteriormente se modifican de la siguiente manera:

$$\sum_{j=1}^{p} \beta_{j1}^2 + \beta_{j2}^2 = 1 \tag{2.45}$$

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + ... + \beta_{j2} x_{ij}^2 \geq M(1 - \epsilon_i) \tag{2.46}$$

$$0 \geq \sum_{i=1}^{n} \epsilon_i \geq C \tag{2.47}$$

Como alternativa, se podrían considerar otras funciones de los predictores en lugar de los polinomios. Hay varias formas posibles de ampliar el espacio de características y, a menos que tengamos cuidado, podríamos acabar con un número enorme de características. Entonces, los cálculos se volverían inmanejables.

2.5.4. SVM

La máquina de vectores soporte (SVM) es una extensión del clasificador de vectores soporte que resulta de ampliar el espacio de características de una manera específica, utilizando kernels. Un kernel es una función que aumenta el espacio de características para conseguir una frontera no lineal entre las clases [40].

Este método requiere del producto interno definido por:

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \tag{2.48}$$

Por tanto el clasificador de vectores soporte puede ser representado como

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \tag{2.49}$$

Donde i = 1 , ..., n es cada una de las observaciones.

Para estimar estos parámetros necesitaremos $\binom{n}{2}$ productos internos, es decir uno por cada par de observaciones de entrenamiento. Para evaluar la función f(x), necesitamos calcular el producto interno entre el nuevo punto x y cada uno de los puntos de entrenamiento x_i. Sin embargo, resulta que α_i es distinto de cero sólo para los vectores soporte de la solución, es decir, si una observación de entrenamiento no es un vector soporte [13]. Así que si S es la colección de índices de estos puntos de soporte, se puede reescribir cualquier función de solución como:

$$f(x) = \beta_0 + \sum_{i \in S}^{n} \alpha_i \langle x, x_i \rangle \tag{2.50}$$

También puede ser representada como:

$$K(x_i, x_{i'}) \tag{2.51}$$

La función más simple es la que se conoce como kernel lineal y es equivalente al clasificador de vectores soporte. Su expresión es:

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j} \tag{2.52}$$

Otra posibilidad es utilizar un kernel polinómico:

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d \tag{2.53}$$

Este kernel polinómico es de grado d. Este tipo de kernels aporta más flexibilidad a la hora de elegir el kernel.

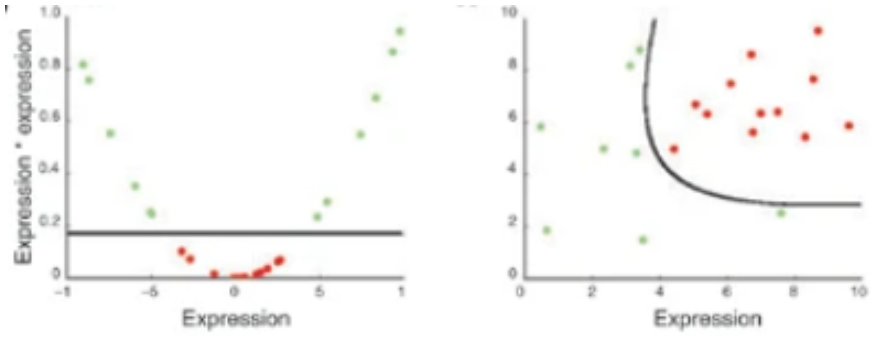


Figura 2.25: Máquina de vectores soporte con un kernel polinómico [40]

También existen kernels no polinómicos. Un ejemplo de esto es el kernel radial que tiene la siguiente forma:

$$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=i}^{p} (x_{ij} - x_{i'j})^2) \tag{2.54}$$

Donde γ es una constante positiva. Si una observación de test dada está lejos de una observación de entrenamiento x_i en términos de distancia euclidiana, entonces $\sum_{j=1}^{p}(x_j^* - x_{ij})^2$ será grande y la expresión del kernel, pequeña. En este caso, la observación no tendrá demasiada relevancia en el clasificador. En otras palabras, las observaciones de entrenamiento que están lejos de x^* no desempeñarán esencialmente ningún papel en la etiqueta de clase predicha para x^*. Esto significa que el kernel radial tiene un comportamiento muy local, en el sentido de que sólo las observaciones de entrenamiento cercanas tienen un efecto en la etiqueta de clase de una observación de test [23].

El uso de este kernel tiene varias ventajas. Una de las ventajas es computacional, y consiste en que utilizando kernels sólo hay que calcular $K(x_i, x_i')$ para los $\binom{n}{2}$ distintos pares.

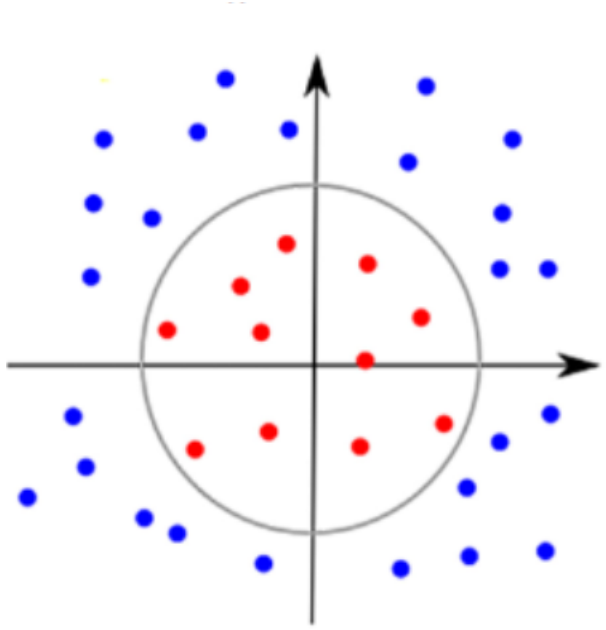


Figura 2.26: Los kernels radiales son útiles para ciertos conjuntos de datos [40]

Esto puede hacerse sin trabajar explícitamente en el espacio de características ampliado. La importancia de este punto radica en que en muchas aplicaciones de SVM como la que se tratará en este trabajo, el espacio de características ampliado es tan grande que los cálculos serían inabordables. Para algunos kernels, como el kernel radial, el espacio de características es implícito y de dimensiones infinitas.

Capítulo 3

Planificación

3.1. Características e infraestructura del proyecto

Al ser un trabajo universitario de fin de grado debemos atender a las características que esto implica. Se trata de un proyecto de ámbito académico realizado en convenio con la empresa externa HP SCDS y enmarcado dentro de la asignatura correspondiente. Este trabajo consta de 12 créditos ECTS. Siguiendo la normativa de la universidad cada crédito corresponde a 25 horas de trabajo y, por lo tanto, el proyecto deberá adecuarse a un tiempo de trabajo de 300 horas totales.

Este proyecto tiene su principal enfoque en la aplicación de las técnicas analizadas. En consecuencia, es importante que el fundamento teórico sea comprendido. Por otro lado no se requerirá de un análisis de requisitos tan detallado como en proyectos orientados al desarrollo de aplicaciones.

Para este tipo de proyectos es preferible un modelo en cascada sobre uno incremental ya que no hay «funcionalidades» que implementar como tal sino que el modelo se va elaborando paso a paso. Un problema que puede surgir a la hora de realizar la planificación es la falta de proyectos previos con características similares.

Las partes implicadas en el proyecto son: el estudiante, el tutor externo de la empresa y el tutor académico que se coordinarán a través de reuniones.

3.2. Metodología de planificación

Debido a que este trabajo de fin de grado se trata de un proyecto del observatorio HP que se realiza en convenio con la empresa HP SCDS, la planificación se debe adaptar a la metodología de planificación de proyectos de la empresa. Esta es una metodología «AGILE» conocida como «SCRUM».

Al tratarse de un proyecto individual, no se aplica la metodología SCRUM al completo ya que no tiene sentido implementar una comunicación tan frecuente como indica este método. Por lo tanto, las reuniones diarias o «daily scrums» implícitas en esta metodología se sustituyen por el uso de mensajería en el caso de ser necesaria una comunicación entre tutor/es y alumno. Para la planificación se utilizarán sprints de dos semanas en los que se especificará un entregable que deberá ser realizado a lo largo del sprint. Las reuniones de «sprint planning » y «review meeting» se unificarán en reuniones de 30-45 minutos por sprint.

En las metodologías ágiles es importante que la documentación que se genera en el proyecto esté actualizada, aunque también es importante tener en cuenta que es preferible la comunicación directa sobre una documentación demasiado extensa. Se utilizará un repositorio donde sea posible acceder, añadir y modificar tanto scripts de código como otros archivos relativos a la documentación para que las partes tengan la posibilidad de observar el desarrollo del proyecto.

3.3. Recursos

- Ordenador
- Google Colaboratory
- Overleaf
- Zoom
- Kaggle
- Python
  - Pennylane
  - Pytorch/torchvision
  - Librerías de Scikit
  - Librerías de manejo de archivos y del ssistema
    - os
    - cv2
    - copy
    - zipfile
    - glob
  - Otras librerías gráficas
    - Matplotlib
    - Pil
- WebMail/Outlook
- GitLab
- Herramientas de representación gráfica
  - NN-SVG
  - Quirk

3.3.1. Discusión sobre herramientas empleadas

Lenguaje de programación

Teniendo en cuenta que el proyecto a desarrollar está relacionado con el aprendizaje automático, se deben buscar lenguajes que sean prácticos para esta tarea. Aunque a día de hoy existen varios lenguajes que disponen de bibliotecas de aprendizaje automático, los mas completos son Python y R, los cuales también permitirán la creación de cuadernos. Dado que nuestro proyecto no requiere de demasiadas representaciones gráficas ni análisis de datos, Python es mejor opción. Además las principales librerías relacionadas con computación cuántica están escritas en Python, lo que lo determinaría finalmente como lenguaje a utilizar.

Entorno de desarrollo

Los «productos» a desarrollar son cuadernos de Python. Algunas de las opciones disponibles son los Jupyter Notebooks y el entorno de desarrollo en linea Google Colaboratory. La primera opción es software local por lo que no requiere de conexión a internet, con la ventaja de que no se desconecta tras un periodo determinado de tiempo y permite aprovechar los recursos de la máquina. Por otro lado, Google Collaboratory es un entorno al cual se puede acceder desde cualquier dispositivo y las prestaciones de la máquina que se utiliza suelen ser superiores aunque algo limitadas en memoria RAM. Esto implica que Google Colaboratory no depende del sistema operativo utilizado. Teniendo en cuenta que el objetivo es realizar una pequeña simulación con pocos qúbits, la segunda opción será preferente.

Librerías de aprendizaje automático

Para técnicas de Aprendizaje Automático de propósito general se utilizará la librería Scikit-Learn. Ésta es la librería más utilizada para este propósito ya que contiene tanto algoritmos de clasificación como máquinas de vectores soporte (las cuales son cruciales en este proyecto), así como otras herramientas de preprocesado de datos, evaluación de modelos y algunos conjuntos de datos de prueba.

Para herramientas relacionadas específicamente con el Aprendizaje Profundo, disponemos de dos opciones: Tensorflow y Pytorch. Tensorflow aportaría varias ventajas. Es una librería más desarrollada con mejores opciones para poner un modelo en producción y la herramienta Tensorboard permite obtener visualizaciones de las redes neuronales que se están entrenando. No obstante dado que ninguna de esas ventajas es clave en este proyecto, se utilizará Pytorch ya que es una librería más acorde al estilo de Python y es la recomendada al realizar investigación.

Librería de computación cuántica

Las cuatro librerías que lideran el aprendizaje automático son: Qiskit de IBM, TensorFlow Quantum, Pennylane y Paddle. Las primeras dos son las más utilizadas:

- Qiskit otorga una gran flexibilidad a la hora de utilizar los recursos necesarios para simular computación cuántica.
- Tensorflow Quantum es descartado automáticamente ya que no se utilizará TensorFlow.
- Paddle en concreto se centra exclusivamente en deep learning y es la librería menos documentada.

Entre Qiskit y Pennylane, elegiremos Pennylane ya que, esta librería, permite la integración con las distintas librerías de Aprendizaje Profundo así como con otras librerías de computación cuántica y hardware relacionado con esta disciplina; además de tener un enfoque más directo hacia el aprendizaje automático cuántico.

Control de versiones

Se utilizará GitLab en lugar de GitHub puesto que es el sistema utilizado en la empresa para este proyecto.

Editor de texto

Se utilizará Overleaf como editor de texto ya que permite editar texto en formato Latex en línea.

Aplicación para de videoconferencia

Existen varias posibilidades: Zoom, Google Meet, Cisco Webex Meetings, Jitsi, etc. La aplicación utilizada de acuerdo a los tutores de la empresa será Zoom.

3.4. Tareas

Para elaborar una lista de las tareas a realizar en este proyecto es importante tener en cuenta tanto las características como el marco del mismo, de forma que sea posible estimar el tiempo de cada una de ellas. Para una estimación más precisa, las tareas se desglosarán en subtareas.

Teniendo en cuenta que la duración del trabajo debe estar en torno a las 300 horas y atendiendo a la importancia teórica del mismo, se debe dar relevancia al estudio de los distintos conceptos de aprendizaje automático y computación cuántica que se van a trabajar y al tiempo que requerirá la redacción del mismo, que será mayor que en trabajos de otro tipo. Para la planificación de la construcción, tanto de los experimentos iniciales como del modelo final, se seguirá una estructura de planificación CRISP-DM teniendo en cuenta el factor experimental de la parte cuántica. Por último, se detallarán otras tareas adicionales.

Tarea	Subtarea	Estimación de horas	Horas totales
Estudio	Estudio de los fundamentos de la computación cuántica	20	110
	Repaso de las técnicas y algoritmos de aprendizaje automático	30	
	Estudio del aprendizaje automático cuántico	30	
	Manejo de las herramientas para la elaboración de experimentos	30	
Experimentación	Comprensión de objetivos	15	100
	Comprensión de la estructura de los datos	5	
	Preprocesado de datos	20	
	Diseño del circuito cuántico	15	
	Modelización	25	
	Evaluación	20	
Redacción	Introducción	5	65
	Marco teórico y análisis de técnicas	35	
	Modelos	10	
	Conclusiones	5	
	Bibliografía	5	
	Otros	5	
Otras tareas	Planificación	15	25
	Reuniones	5	
	Documentación	5	

Cuadro 3.1: Coste de tareas.

3.5. Riesgos

Se contemplan los siguientes riesgos que podrían afectar a las estimaciones realizadas anteriormente.

Tarea	Riesgo	Probabilidad	Impacto
Estudio	Conceptos demasiado complejos	Alta	Medio
	Conceptos erróneos	Baja	Alto
	Objetivos poco claros	Media	Crítico
	Falta de documentación	Muy Baja	Alto
Experimentación	Modelos tardan demasiado en entrenarse	Media	Bajo
	Datos demasiado complejos para la infraestructura a implementar	Muy Baja	Medio
	Los modelos generados no clasifican el input correctamente	Baja	Medio
	El circuito cuántico no proporciona ninguna información	Baja	Medio
	La fase de test no es correcta	Muy Baja	Bajo
Redacción	Problemas con el editor de texto	Alta	Bajo
	La redacción no es correcta	Baja	Bajo
Otras tareas	Estimación incorrecta del coste de las tareas	Alta	Medio
	Imposibilidad de realizar reuniones	Muy Baja	Bajo
	Documentación no comprensible	Muy Baja	Bajo

Cuadro 3.2: Riesgos.

3.6. Seguimiento y Revisión

Estas son las actividades realizadas en cada sprint:

3.6.1. Seguimiento

Sprint 1

En este primer sprint se han estudiado los fundamentos de la Computación Cuántica sobre los que se desarrolla el proyecto. También se han explorado las distintas tecnologías que permiten realizar un modelo de Aprendizaje Profundo con Computación Cuántica entendiendo ejemplos de cuadernos de código disponibles en cada sitio web.

Sprint 2

En este sprint se profundiza en los conocimientos en computación cuántica y está dedicado a la lectura de cuadernos de la página oficial de Pennylane relacionados con el aprendizaje automático cuántico. También se realiza una primera lectura del artículo «Supervised quantum machine learning models are kernel methods» [49].

Sprint 3

En este periodo se revisan los conceptos ya conocidos sobre Aprendizaje Automático, especialmente los relacionados con el Aprendizaje Profundo y las redes neuronales convolucionales y, sobre todo, las máquinas de vectores soporte. También se realiza un modelo de clasificación clásico de un conjunto de datos estándar mediante el paquete SVM de Scikit-learn.

Sprint 4

En este sprint se investiga sobre tratamiento de imágenes con técnicas de Aprendizaje Automático Cuántico, distinguiendo el uso de convoluciones cuánticas como forma de «embedding» para realizar posteriormente una clasificación y el uso de capas convolucionales clásicas usando un circuito cuántico para la decisión.

Sprint 5

Este sprint se ha dedicado a la elaboración de los primeros modelos que involucran Computación Cuántica. Esto incluye el entrenamiento de un circuito ansatz mediante la interfaz de PyTorch y un modelo de Computación Híbrida que utiliza el circuito cuántico como capa de una red neuronal.

Sprint 6

Se realiza una versión alternativa del modelo de red neuronal cuántica del anterior sprint. Se comienza la redacción de la memoria del proyecto plasmando la planificación y comenzando el capítulo del marco teórico. Se construye un modelo para clasificación binaria basado en kernels cuánticos.

Sprint 7

Se lleva a cabo de nuevo un modelo de clasificación binaria mediante kernels cuánticos con otro conjunto de datos. Se amplía el capítulo de Aprendizaje automático mediante Computación Cuántica de esta memoria.

Sprint 8

Se amplían las secciones del fundamento teórico y Aprendizaje automático mediante Computación Cuántica. Se elabora un modelo de red convolucional clásica para el conjunto de datos de kaggle de clasificación binaria de perros y gatos.

Sprint 9

Se culmina la redacción de los capítulos de marco teórico y Aprendizaje automático mediante Computación Cuántica. Se aplica un modelo de máquinas de vectores soporte utilizando un kernel cuántico sobre las características extraídas por la red creada en el sprint anterior.

Sprint 10

Se realiza una revisión sobre la parte teórica de la memoria. También se entrena el circuito variacional del modelo creado mediante la optimización del KTA.

3.6.2. Revisión

Tras la realización del proyecto, podemos comprobar si la planificación ha sido adecuada. Primero comprobaremos qué riesgos han tenido lugar.

Tarea	Riesgo	Ha sucedido
Estudio	Conceptos demasiado complejos	Sí
	Conceptos erróneos	Sí
	Objetivos poco claros	Sí
	Falta de documentación	No
Experimentación	Modelos tardan demasiado en entrenarse	Sí
	Datos demasiado complejos para la infraestructura a implementar	No
	Los modelos generados no clasifican el input correctamente	No
	El circuito cuántico no proporciona ninguna información	No
	La fase de test no es correcta	No
Redacción	Problemas con el editor de texto	Sí
	La redacción no es correcta	Sí
Otras tareas	Estimación incorrecta del coste de las tareas	Sí
	Imposibilidad de realizar reuniones	No
	Documentación no comprensible	Sí

Cuadro 3.3: Comprobación de riesgos.

Medidas de contingencia

Estas son las acciones tomadas para mitigar el efecto de los riesgos acontecidos.

- Para solucionar el problema de tratar con conceptos complejos y no comprendidos se realizarán varias lecturas de la fuente de dichos conceptos y/o se buscarán nuevas.
- Desde una perspectiva de Aprendizaje Automático clásico, ha sido complicado entender exactamente dónde deben utilizarse los nodos cuánticos en la construcción del modelo objetivo. Para clarificar como elaborar un modelo de kernels cuánticos se ha tratado el concepto de los kernels cuánticos una reunión. Posteriormente se ha comprobado que la implementación realizada sigue la estructura fijada.
- Al tratar con imágenes, que cuentan con grandes cantidades de información, además de con una simulación de un circuito cuántico, el entrenamiento se prolonga más de lo deseado. La primera medida

para remediarlo es hacer uso de la GPU que proporciona Google Collaboratory aunque por tiempo limitado. También se reducirá el número de instancias utilizadas para entrenar el modelo cuántico.

- Se han realizado consultas a sitios web para ciertos problemas de redacción con el sistema generador LaTex.
- Para mejorar tanto la redacción de la memoria como la comprensibilidad de la documentación, se han revisado y reescrito ambas parcialmente.
- Ha sido necesario añadir horas debido a que la estimación inicial ha sido realizada sin experiencia previa y debido a los riesgos que se han producido.

Capítulo 4

Aprendizaje automático mediante computación cuántica

4.1. Conceptos iniciales

El área donde las ideas del aprendizaje automático y la computación cuántica se unen es llamado aprendizaje automático cuántico. Los ordenadores cuánticos podrían aminorar el tiempo que se tarda en entrenar o evaluar un modelo de aprendizaje automático. Podemos aprovechar las técnicas del aprendizaje automático para ayudarnos a descubrir códigos cuánticos de corrección de errores, estimar las propiedades de los sistemas cuánticos o desarrollar nuevos algoritmos cuánticos. Por otro lado, la computación cuántica puede ser utilizada como acelerador para algunas operaciones dentro de la inteligencia artificial al tener hardware más eficiente para algunas de ellas. Aunque aún no ha sido posible construir un ordenador tolerante a fallos, existe un interés creciente por la utilización de dispositivos cuánticos a corto plazo. La naturaleza diferenciable de los circuitos cuánticos, es decir, la posibilidad de modificar parámetros para ajustarse mejor a una tarea, muestra la relación que existe con el aprendizaje profundo [14].

4.1.1. Ansatz

Un ansatz es un circuito variacional que describe una subrutina consistente en una secuencia de puertas que se aplican a una serie de «cables». Un cable representa un qúbit a lo largo del tiempo. Un ansatz es similar a una red neuronal ya que a diferencia de un circuito clásico, los circuitos cuánticos cuentan con parámetros libres entrenables que pueden ser optimizados. Existen tres estructuras de ansatzs: [53]

Arquitectura en capas Una capa se define como una secuencia de puertas que se repiten. El número de repeticiones se considera un hiperparámetro del circuito [46].

Se consideran dos capas A y B donde A contendrá puertas de un solo cable (operaciones de un qúbit) y B contendrá tanto operaciones simples como entrelazadas. Los ansatze con esta estructura pueden distinguirse en las capas que están parametrizadas, en los tipos de puertas en cada capa y en el posicionamiento de las puertas en la capa B. Por regla general solo la capa A está parametrizada . No obstante, también existe la posibilidad de tener ambas capas parametrizadas [50].

Los circuitos «Instantaneous Quantum Polynomial» o IQP [56][3] son circuitos en los que las operaciones de la capa A son Hadamard, es decir, no contienen ningún parámetro mientras que la capa B tiene operaciones parametrizadas. Por último, también es posible generalizar los circuitos de dos bloques y poder construir

más capas.

Arquitectura de operador alternativo El ansatz de operador alternativo diseñado por Farhi, Goldstone y Gutmann en [17] es la base para el algoritmo de optimización cuántica aproximada que fue utilizado posteriormente para aprendizaje automático y otras aplicaciones. En este caso, se dividen las capas en dos bloques A y B en los que las transformaciones unitarias que determinan los bloques se definen mediante hamiltonianos que evolucionan a lo largo de un corto periodo de tiempo. Esto hace variar el estado del bloque A al estado base del bloque B de forma que se alternan sus características en un breve periodo δt [61].

Arquitectura de red de tensores También existen estructuras no consistentes en capas sino inspiradas en redes de tensores. Existen distintas posibilidades, como aplicar entrelazamiento en los qúbits en forma de árbol o aplicar transformaciones a distintos subconjuntos de qúbits. El tamaño de esta transformación es el hiperparámetro «bond» que añade complejidad cuanto mayor sea este [16].

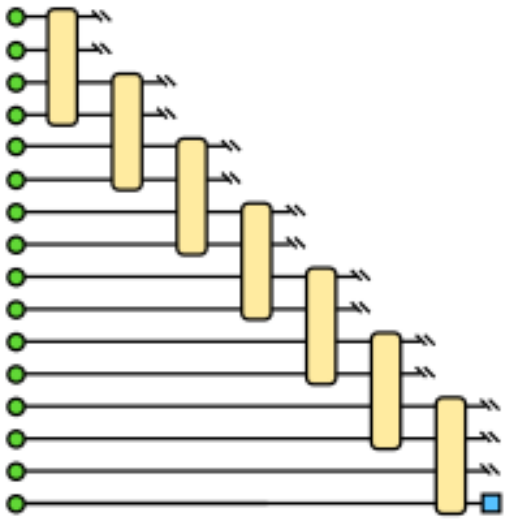


Figura 4.1: Circuito ansatz con estructura tensorial, de subconjuntos e hiperparámetro bond = 3, extraído de:[21]

4.1.2. Mapa de características cuántico

Muchos métodos clásicos de aprendizaje automático reexpresan sus datos de entrada en un espacio diferente para que sea más fácil trabajar con ellos, o porque el nuevo espacio puede tener algunas propiedades convenientes. El ejemplo más claro y el que nos ocupa en este proyecto son las máquinas de vectores soporte, las cuales clasifican mediante el uso de un hiperplano. Para datos que no sean linealmente separables en el espacio original, es necesario realizar una transformación mediante un mapa de características [50].

En el caso de la computación cuántica un mapa de características es una aplicación $X \to F$ donde F es un espacio de Hilbert de forma que el mapa de características actúa tal que $x \to |\phi(x)\rangle$ realizada mediante

un circuito variacional cuyos parámetros dependen de x [18].

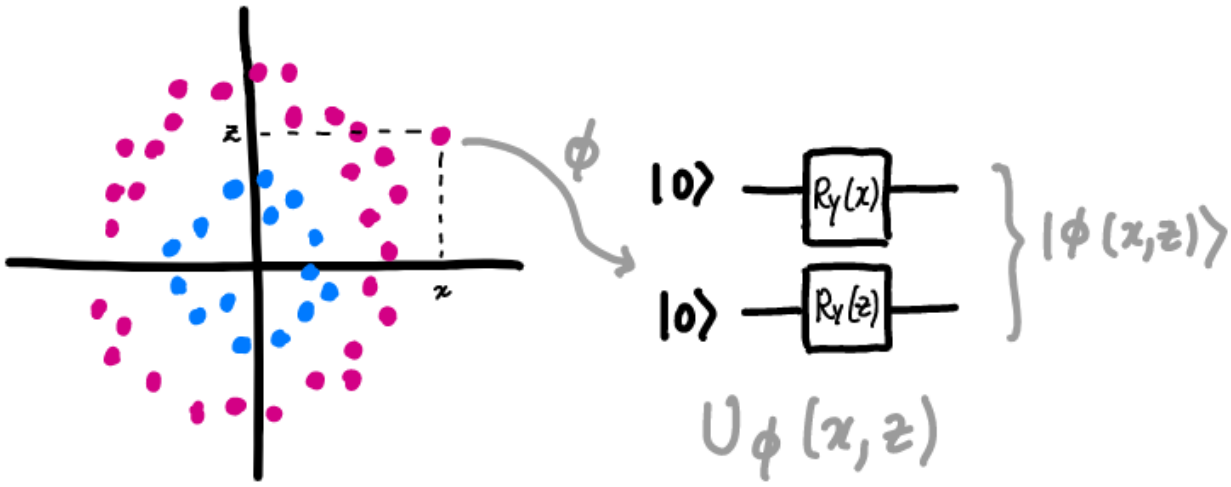


Figura 4.2: Representación de un mapa de características cuántico[18]

4.1.3. Quantum embedding

Un «quantum embedding» representa datos clásicos como estados cuánticos en espacios de Hilbert mediante un mapa de características [30]. El circuito cuántico transforma el dato x en el estado cuántico denotado como $|\psi_x\rangle$. Este proceso es fundamental en el uso de los kernels cuánticos [51]. Existen varios tipos de «embedding».

Basis Embedding Este «embedding» asocia cada dato de entrada con un estado de la base del sistema cuántico. Por lo tanto, los datos clásicos tienen que estar en forma de cadenas binarias. El estado cuántico incrustado es la traducción en forma de bits de una cadena binaria a los estados correspondientes de los subsistemas cuánticos. por ejemplo si tenemos que $X = 1110$, será representado en la salida del circuito como $|1110\rangle$. Para representar un dato con n características binarias se necesitará un sistema cuántico de n qúbits [18]. Para representar un conjunto de datos se utiliza la siguiente fórmula:

$$|D\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left| x^{(m)} \right\rangle \tag{4.1}$$

Como podemos ver esta es la forma más trivial de embedding.

Amplitude Embedding En esta técnica los datos son codificados mediante las amplitudes de un estado cuántico. Un dato clásico n-dimensional normalizado se representa con la amplitudes de un estado cuántico ψ como:

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i \, |i\rangle \tag{4.2}$$

Donde $N = 2^n$, x_i es el elemento i-ésimo de x y $|i\rangle$ es el i-ésimo estado de la base cuántica. En este caso para codificar un conjunto de datos necesitaremos:

$$|D\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle \tag{4.3}$$

donde α_i son los elementos del vector de amplitudes e $|i\rangle$ los estados de la base cuántica. El número de amplitudes a ser codificadas son N * M. Como un sistema de n qúbits permite codificar 2^n amplitudes, se debe cumplir que $n \geq log_2(N * M)$ [51].

Angle Embedding Este «embedding» se realiza aplicando rotaciones en los ejes x o y usando puertas cuánticas sobre los valores que se quieren codificar. Deben existir tantas rotaciones como características en un conjunto de datos [30].

4.1.4. Circuitos variacionales

Los circuitos variacionales son aquellos circuitos cuánticos que dependen de parámetros libres. Consisten en un estado inicial fijo (generalmente todos los qúbits a $|0\rangle$), un circuito cuántico parametrizado $U(\theta)$ y una medida de un observable como salida de este que puede estar dado por todos o un subconjunto de los cables del circuito.

Los circuitos variacionales se entrenan mediante un algoritmo de optimización clásico que realiza consultas al dispositivo cuántico. La optimización suele ser un esquema iterativo que busca mejores candidatos para los parámetros θ con cada paso.

Los circuitos variacionales se han popularizado como forma de algoritmos para los dispositivos cuánticos a corto plazo. Estos dispositivos sólo pueden ejecutar secuencias de pocas puertas, ya que sin tolerancia a fallos, cada puerta aumenta el error en la salida. Normalmente, un algoritmo cuántico se descompone en un conjunto de operaciones elementales estándar, que a su vez son implementadas por el hardware cuántico [50].

La idea del circuito variacional para los dispositivos a corto plazo es fusionar este procedimiento de dos pasos en uno solo, haciendo aprender el circuito óptimo al dispositivo para una tarea determinada. De este modo, las puertas parametrizadas de un dispositivo pueden utilizarse para formular el algoritmo, eliminando el uso de puertas elementales fijas. Además, los errores sistemáticos pueden corregirse automáticamente durante la optimización.

Los parámetros variacionales θ, junto a un posible conjunto adicional de parámetros no adaptables, entran en el circuito cuántico como argumentos para las puertas del circuito. Esto nos permite convertir la información clásica (los valores θ y x) en información cuántica. La información cuántica se convierte en información clásica evaluando el valor de la expectativa del observable. Más allá de la regla básica de que los parámetros θ se utilizan como argumentos de las puertas, la forma exacta en que se disponen las puertas (la arquitectura del circuito) es esencialmente arbitraria [18].

4.1.5. Gradiente cuántico

La salida de un circuito variacional es el valor esperado de un observable de medición, que puede escribirse formalmente como una "función cuántica"parametrizada f (θ) en los parámetros ajustables $\theta = \theta_1$, θ_2, ..., θ_n . Como con cualquier otra función de este tipo, se pueden definir derivadas parciales de f con respecto a sus parámetros.

Un gradiente cuántico es el vector de derivadas parciales de una función cuántica:

$$\nabla_\theta(\theta) = \begin{pmatrix} \partial\theta_1 f \\ \partial\theta_2 f \\ . \\ . \\ . \\ . \end{pmatrix} \tag{4.4}$$

Si se utilizan varias funciones para describir otra función se utiliza el Jacobiano de todas ellas:

$$J_\theta(\theta) = \begin{pmatrix} \partial\theta_1 f_1 & \partial\theta_1 f_2 & ... & ... & \partial\theta_1 f_n \\ \partial\theta_2 f_2 & \partial\theta_2 f_2 & ... & ... & \partial\theta_2 f_n \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ \partial\theta_2 f_n & ... & ... & ... & \partial\theta_n f_n \end{pmatrix} \tag{4.5}$$

El gradiente de una función cuántica $f(\theta)$ puede expresarse en muchos casos como una combinación lineal de otras funciones cuánticas mediante *reglas de desplazamiento de parámetros* [34]. Esto significa que los gradientes cuánticos pueden ser calculados por los ordenadores cuánticos, lo que abre la computación cuántica a la optimización basada en el gradiente, como el descenso del gradiente, que se utiliza ampliamente en el aprendizaje automático.

4.1.6. Reglas de desplazamiento de parámetros

La salida de un circuito variacional, es decir, la esperanza del observable, puede ser escrita como función cuántica parametrizada $f(\theta)$. La derivada parcial de $f(\theta)$ puede expresarse, en muchos casos, como una combinación lineal de otras funciones cuánticas. Es importante destacar que estas otras funciones cuánticas suelen utilizar el mismo circuito, variando únicamente en un desplazamiento del argumento. Esto significa que las derivadas parciales de un circuito variacional pueden calcularse utilizando la misma arquitectura de circuito variacional. A la forma de obtener estas derivadas parciales se les denomina reglas de desplazamiento de parámetro [54].

Por ejemplo, para calcular el gradiente en un circuito $f(x, \theta)$ compuesto por puertas Pauli:

$$U_i(\theta_i) = e^{-i\frac{\theta_i}{2}\hat{P}_i} \tag{4.6}$$

Cuyo gradiente se puede calcular tal que:

$$\nabla U_i(\theta_i) = -\frac{i}{2}\hat{P}_i U_i(\theta_i) \tag{4.7}$$

Podemos calcular el gradiente del circuito mediante sí mismo variando θ.

$$\nabla f(x, \theta) = \frac{1}{2}(f(x, \theta + \frac{\pi}{2}) - f(x, \theta - \frac{\pi}{2})) \tag{4.8}$$

4.1.7. Barren Plateau

Los «barren plateaus» son grandes regiones del espacio de parámetros de la función de costes donde la varianza del gradiente es casi cero, es decir, la región de la función de costes es plana. Esto significa que un circuito variacional inicializado en una de estas zonas no podrá ser entrenado con ningún algoritmo basado en el gradiente.

Se ha demostrado que la idea de que el fenómeno de los «barren plateaus» puede, en algunas circunstancias, evitarse utilizando funciones de coste que sólo requieren información de una parte del circuito. Estas funciones de coste locales pueden ser más robustas contra el ruido y pueden tener gradientes mejor manejados sin «barren plateaus» para los circuitos poco profundos.

Muchos algoritmos cuánticos variacionales están construidos para utilizar funciones de coste globales. La información de toda la medición se utiliza para analizar el resultado del circuito, y a partir de ella, se calcula una función de coste para cuantificar el rendimiento del circuito. Una función de coste local sólo tiene en cuenta la información de unos pocos qúbits, e intenta analizar el comportamiento de todo el circuito a partir de estos.

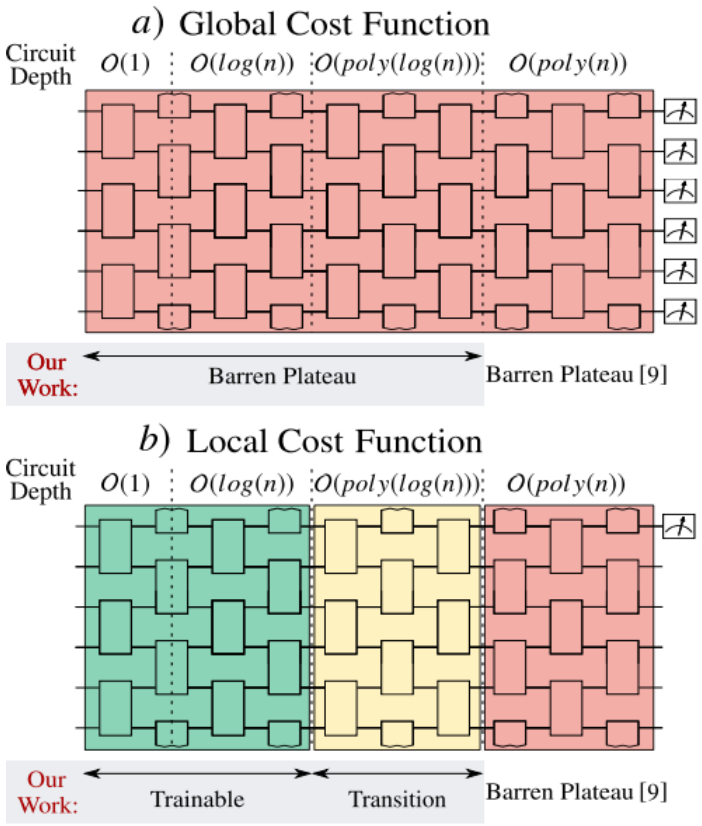


Figura 4.3: Órdenes de funciones de coste para los cuales es posible evitar un «barren plateau» según [10]

Las funciones de coste local están acotadas por las de coste global por lo que su valor será siempre menor.

4.1.8. Red neuronal cuántica

Una red neuronal cuántica (QNN) es un algoritmo de aprendizaje automático que combina conceptos de la computación cuántica y las redes neuronales. El término se ha utilizado para describir una gran variedad

de conceptos, que van desde los ordenadores cuánticos que emulan los cálculos exactos de las redes neuronales, hasta los circuitos cuánticos generales entrenables con una estructura muy diferenciada de la de un perceptrón multicapa [26].

Se han tratado de idear versiones cuánticas de las redes neuronales recurrentes y «feed-forward». Los modelos eran intentos de traducir la estructura modular y las funciones de activación no lineales de las redes neuronales al lenguaje de los algoritmos cuánticos. La mejoría de estos modelos sobre el aprendizaje automático aún no se ha determinado de forma concluyente [52].

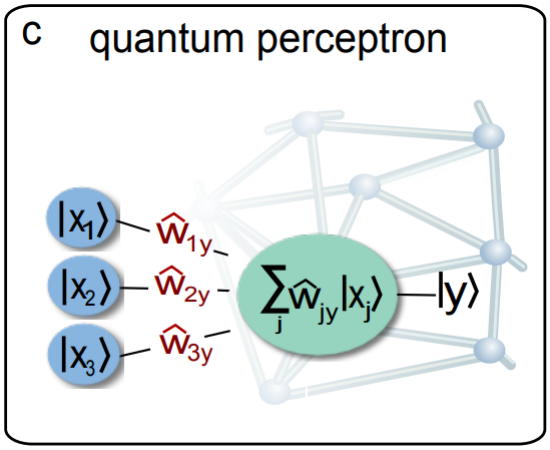


Figura 4.4: Se busca traducir el mecanismo de un perceptrón en una versión cuántica que no haga perder la dinámica de una red neuronal. Tomado de: [52]

El término red neuronal cuántica, se utiliza frecuentemente para referirse a los circuitos cuánticos variacionales o parametrizados. Aunque matemáticamente es bastante diferente del funcionamiento interno de las redes neuronales, la analogía pone de manifiesto la naturaleza modular de las puertas cuánticas en un circuito, así como el amplio uso de técnicas de entrenamiento de redes neuronales utilizadas también en la optimización de algoritmos cuánticos [57].

4.1.9. Computación híbrida

En el contexto de la computación cuántica, el término computación híbrida se refiere a la técnica de mezclar cálculos clásicos y cuánticos. Un algoritmo cuántico se optimiza con uno clásico. Esta es la base de la optimización de circuitos variacionales.

Los dispositivos cuánticos se utilizan para estimar las medias de los resultados de las mediciones, es decir, la esperanza de los observables de los circuitos cuánticos. Una función clásica de coste evalúa la bondad a partir de dichos observables [39].

Nodos cuánticos En computación híbrida clásica-cuántica se denomina nodo cuántico a cada una de las partes de un algoritmo cuyos cálculos se obtienen mediante unidades de información y operaciones cuánticas.

Grafos acíclicos dirigidos Un algoritmo híbrido sigue esta estructura si sus operaciones pueden ser representadas mediante un grafo dirigido acíclico cuyos nodos pueden ser, o bien operaciones clásicas, o bien nodos cuánticos.

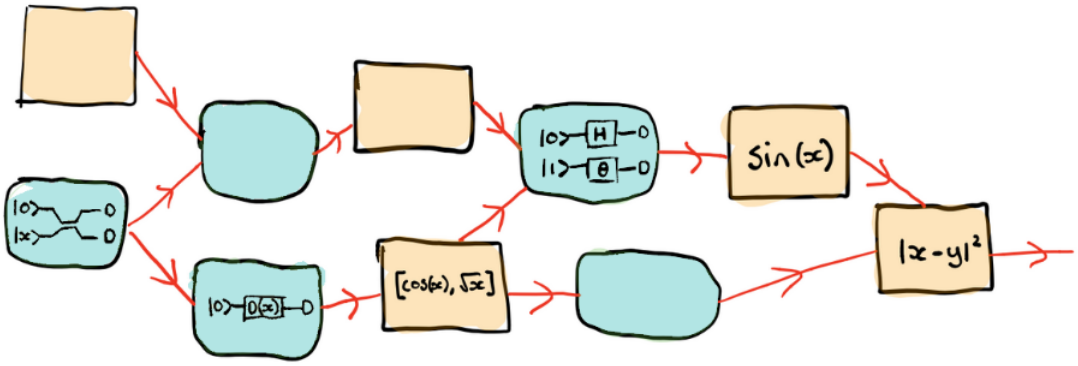


Figura 4.5: Grafo acíclico dirigido de un algoritmo de computación híbrida. En naranja las operaciones clásicas y en azul los nodos cuánticos. [60]

Retropropagación en computación híbrida . La posibilidad de calcular el gradiente en circuitos cuánticos variacionales nos permiten el uso del algoritmo de retropropagación en computación híbrida. De esta forma, podremos entrenar las partes clásicas y cuánticas de un algoritmo híbrido de una forma similar a la del Aprendizaje Profundo.

4.1.10. Programación diferenciable cuántica

En computación cuántica, se pueden calcular automáticamente las derivadas de los circuitos variacionales con respecto a sus parámetros de entrada. La programación cuántica diferenciable es un paradigma que aprovecha esto para hacer que los algoritmos cuánticos sean diferenciables y, por tanto, entrenables.

La programación diferenciable es un estilo de programación que utiliza la diferenciación automática para calcular las derivadas de las funciones con respecto a las entradas del programa. La programación diferenciable está relacionada con el aprendizaje profundo, pero tiene algunas diferencias. En la mayoría de modelos de aprendizaje profundo, la estructura de las redes neuronales, el número de nodos y el número y forma de las capas ocultas, se define de forma estática. Una red neuronal es diferenciable y se entrena mediante diferenciación automática y retropropagación, pero la estructura fundamental del programa no cambia.

En los últimos años los modelos se componen de bloques de funciones parametrizadas, y su estructura es dinámica: puede cambiar en función de los datos de entrada, pero sigue siendo entrenable y diferenciable. Un aspecto crítico es que esto se mantiene incluso en presencia del flujo de control clásico. Esto nos permite utilizar la programación diferenciable en algoritmos híbridos y hacer que todo el programa se pueda entrenar.

Diferenciación automática Si escribes un algoritmo para calcular alguna función $f(x, y)$ que incluye expresiones matemáticas, pero también puede incluir sentencias de flujo de control, entonces la diferenciación automática proporciona un algoritmo para calcular $\nabla f(x, y)$.

La diferenciación automática tiene un enfoque numérico, pero lo que la distingue es que es un método exacto de diferenciación. De forma similar a la diferenciación simbólica, cada componente del cálculo proporciona una regla para su derivada con respecto a sus entradas. Sin embargo, en lugar de que la entrada y la salida sean expresiones matemáticas, la salida de la diferenciación automática es el valor numérico de la derivada.

En esta técnica se busca encontrar la solución de derivada parcial aplicando una regla específica a cada tipo de operación por ejemplo para la suma $a = b + c$:

$$\frac{\partial a}{\partial b} = \frac{\partial b}{\partial b} + \frac{\partial c}{\partial b} \tag{4.9}$$

Para el producto $a = bc$:

$$\frac{\partial a}{\partial b} = \frac{\partial b}{\partial b}c + \frac{\partial c}{\partial b}b \tag{4.10}$$

De este modo, en un conjunto de operaciones, se pueden aplicar este tipo de reglas a lo largo de toda la expresión para obtener la derivada. Si por ejemplo existiera un si condicional en el algoritmo a diferenciar, únicamente se diferenciará la parte que satisface la condición. Este algoritmo se puede realizar hacia delante, es decir, empezando por las expresiones más simples hasta llegar a toda la expresión. No obstante, también existen algoritmos que realizan esto en el sentido contrario como la retropropagación utilizada en redes neuronales [7].

Diferenciación automática para computaciones cuánticas La capacidad de calcular gradientes cuánticos implica la posibilidad de aplicar algoritmos de diferenciación automática en computación cuántica y, por tanto, en computación híbrida.

Muchas operaciones cuánticas utilizan reglas de desplazamiento de parámetros para calcular el gradiente. Las reglas de desplazamiento de parámetros son un ejemplo de autodiferenciación hacia delante. Implican expresar el gradiente de una función como una combinación de sí misma en dos puntos diferentes. Sin embargo, al contraio que en los métodos de diferenciación finita, estos puntos se encuentran bastante alejados.

Esto puede extenderse directamente a los gradientes de las operaciones cuánticas y a los circuitos cuánticos completos. Simplemente se evalúa el circuito en dos puntos diferentes del espacio paramétrico. De este modo, es posible calcular el gradiente de secuencias de puertas parametrizadas. Una vez evaluados, los gradientes pueden ser utilizados en las siguientes partes de un cálculo híbrido.

4.2. Máquinas cuánticas de vectores soporte

Introducción

En la teoría de kernels se describe cómo representar la información en vectores que se encuentran en espacios muy grandes sin que sea necesario calcular estos vectores numéricamente. Los computadores cuánticos tienen que codificar los datos clásicos en los estados físicos de los sistemas cuánticos si se quiere que estos los aprendan. Este proceso es formalmente equivalente a un mapa de características que asigna los datos a los estados cuánticos. Los productos internos de estos estados cuánticos que codifican datos dan lugar a un kernel, un tipo de medida de similitud [49].

Los algoritmos cuánticos de aprendizaje a partir de datos pueden formularse de la misma forma que un kernel clásico, no obstante, el kernel es calculado por un ordenador cuántico. Pese a que las demostraciones que se realizan utilizando los modelos correctos son más cercanas a la teoría, es más común enmarcar estos modelos como redes neuronales [18].

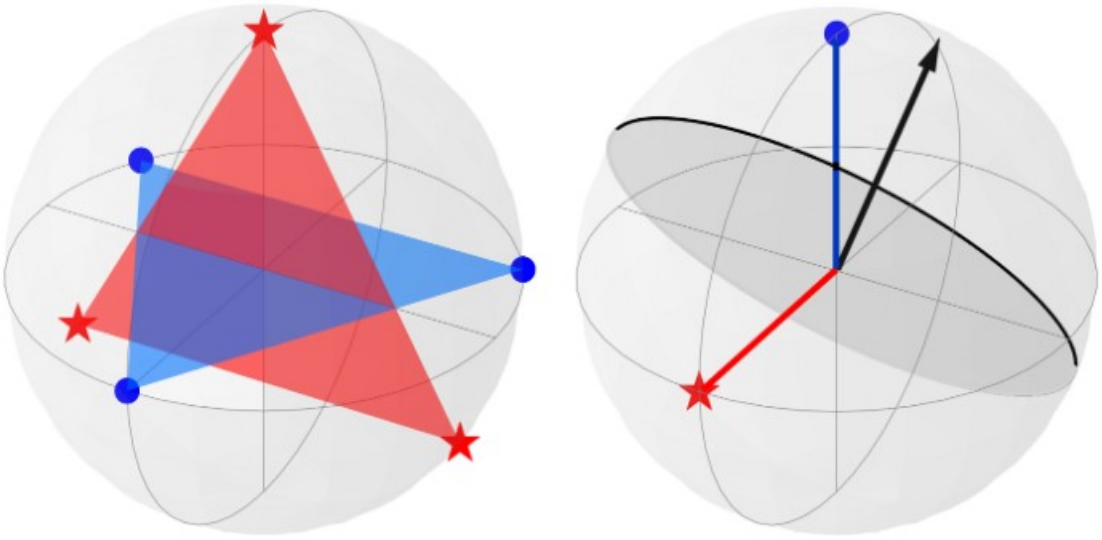


Figura 4.6: Representación de kernels cuánticos en qúbits [24]

La bondad de un modelo cuántico, su optimización y su capacidad de generalización vienen determinadas en gran medida por la forma de codificación de los datos en estados cuánticos que utiliza el Kernel. Mientras el propio kernel puede explorar espacios de estado de alta dimensión del sistema cuántico, los modelos cuánticos pueden ser entrenados y operados en un subespacio de baja dimensión debido a la propiedad de superposición de los qúbits.

En los modelos que no se asemejan a un modelo convencional de aprendizaje automático con parámetros no es necesario buscar cual es el mejor circuito variacional, ni solucionar los problemas conocidos como las «barren plateaus», pero será necesario calcular las distancias de los datos por pares.

Un beneficio a medio plazo puede derivarse también de las amplias herramientas de la computación cuántica, en la que se representa la información en espacios de alta dimensión de forma natural, y posiblemente de nuevos tipos de kernels derivados de los estudios en el campo de la física.

Los ordenadores cuánticos probablemente proporcionarán acceso a un procesamiento de álgebra lineal más rápido que, en principio, será capaz de ofrecer una velocidad de orden polinómico que permitirá a los kernels procesar grandes cantidades de datos sin depender de aproximaciones y heurísticas [49].

Comparación entre kernels clásicos y kernels cuánticos

Aunque los algoritmos cuánticos se pueden utilizar de muchas maneras, la mayoría buscan reemplazar los clasificadores y generadores clásicos de forma que estos puedan ser utilizados en un ordenador cuántico. Sin embargo, a día de hoy es más eficiente utilizar modelos de computación híbrida. En estos modelos se distinguen dos partes: la codificación cuántica o «embedding» y la medida o «measurement». La codificación consiste en asignar un estado cuántico a cada entrada. La medida la cual nos permitirá evaluar el modelo estadísticamente (Ya que no conocemos el valor real de los qúbits).

Al igual que en aprendizaje automático clásico, entrenar un modelo cuántico supone minimizar una función de pérdida que dependa de los datos de entrenamiento y que, en este caso se realizará encontrando la medida adecuada.

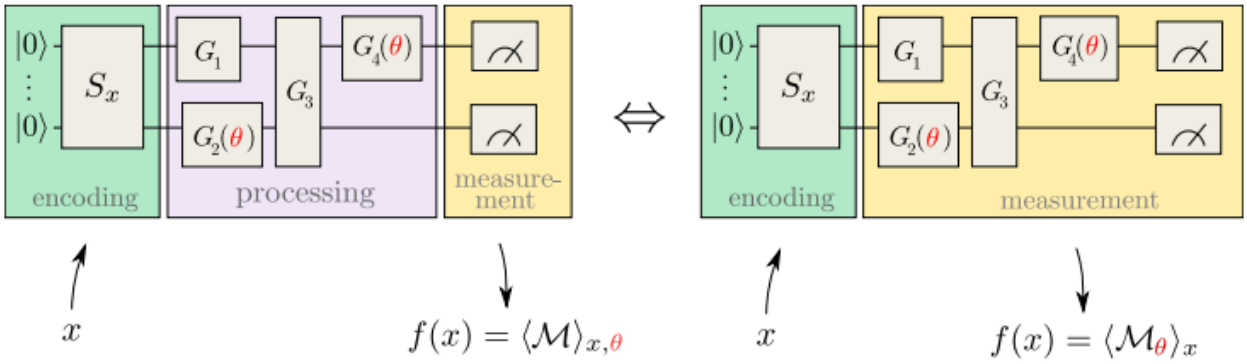


Figura 4.7: Interpretación de un circuito como modelo de aprendizaje automático

4.2.1. Mapas de características

Codificación en estados cuánticos

Matrices de densidad Para realizar un «mapping» (lo cual es necesario tanto en kernels como en computación cuántica) se necesitan definir las matrices de densidad [18].

$$\rho(x) = |\phi(x)\rangle \langle \phi(x)| \tag{4.11}$$

Las matrices de densidad son descripciones alternativas de los estados cuánticos como operadores hermitianos que son útiles porque pueden expresar distribuciones de probabilidad sobre estados cuánticos. Por tanto, describen estados mixtos en lugar de puros.

Embedding Como se explicó en el capítulo anterior, los embeddings son proyecciones de datos clásicos en estados cuánticos. La transformación se realiza mediante un circuito cuántico con unos estados iniciales arbitrarios y puertas cuánticas parametrizadas, de forma que las puertas realicen transformaciones sobre los qúbits en función de los datos de entrada.

Un circuito cuántico U puede tener operaciones que dependan de parámetros externos de forma que definimos la operación que realiza el circuito como U(x). Si disponemos de un estado inicial $|\psi\rangle$ el estado

final aplicandole el circuito será: $|\phi(x)\rangle = U(x)\,|\psi\rangle$. Tras esta operación los datos ya estarán codificados en un estado cuántico ya que la operación depende de estos. Podemos ver que esto se corresponde con un mapa de características que puede ser implementado mediante cualquier operación que dependa de los datos. El espacio de llegada al que pertenece ϕ es un espacio de Hilbert.

Mapa de datos Para transformar los datos se utiliza la siguiente aplicación:

$$\begin{aligned} \phi : \mathbf{X} &\to \mathbf{F} \\ \phi(x) &= |\phi(x)\rangle\,\langle\phi(x)| \end{aligned} \tag{4.12}$$

Donde $\mathbf{X}$ es el espacio de los datos y $\mathbf{F}$ es el espacio de $2^n * 2^n$ números complejos.

Kernels cuánticos

Un kernel cuántico se define como el producto interno de dos vectores de características tal que:

$$k(x, x') = |\,\langle\phi(x')|\phi(x)\rangle\,|^2 \tag{4.13}$$

Donde ϕ es un mapa de características. Esta función cumple la condición de los kernels de ser definida positiva. El conjugado compuesto de un kernel es también un kernel.

$$\langle\phi(x)|\phi(x')\rangle = \langle\phi(x')|\phi(x)\rangle^* \tag{4.14}$$

4.2.2. RKHS

Para entender lo que los kernels pueden aportar al aprendizaje automático cuántico, necesitamos otro concepto importante de la teoría de los kernels: el «reproducing kernel Hilbert space» (RKHS). Un RKHS es un espacio de características alternativo de un kernel y, por tanto, reproduce todo el comportamiento «observable» del modelo de aprendizaje automático. Más concretamente, es un espacio de características de funciones que se construyen a partir del kernel. El RKHS contiene una de estas funciones para cada entrada , así como sus combinaciones lineales. Minimizar funciones de coste en un espacio cuántico es equivalente a minimizar el coste sobre el RKHS del kernel cuántico. [49]

El kernel k sobre X en el espacio de Hilbert F creado al completar la expansión de las funciones $f() = k(x,)$. Para dos funciones pertenecientes a este espacio $f() = \sum_i \alpha_i k(x^i,)\; y\; g() = \sum_j \beta_j k(x^j,)$ el producto interno se define como:

$$\langle f, g\rangle_F = \sum ij\alpha_i\beta_j k(x^i, x^j) \tag{4.15}$$

donde $\alpha_i\; y\; \beta_j$ pertenecen al cuerpo de los números reales.

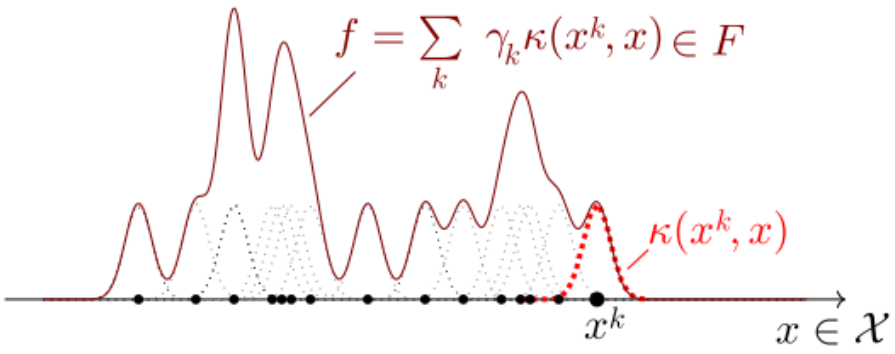


Figura 4.8: Funciones del espacio de Hilbert del kernel reproductor (RKHS)

El tamaño de espacio de los modelos y kernels cuánticos está limitado por las puertas que codifican los datos. La definición del producto interno para kernels es :

$$\langle k(x,), k(x',)\rangle_F = k(x, x') \tag{4.16}$$

Donde el mapa de características relaciona x con su kernel asociado k(x,) es decir, asocia un valor con una función. De esta forma se sustituye el espacio de características de Hilbert por un espacio de funciones siguiendo la propiedad de reproductora:

$$\langle f, k(x,)\rangle_F = f(x) \tag{4.17}$$

Para cada f del espacio F.

Por tanto, el kernel es la forma de evaluar las operaciones del modelo cuántico f como f(x). El RKHS es un espacio cuyas funciones elementales k(x,) asignan una distancia a cada dato. Si añadimos otro dato x' obtendremos la distancia entre los datos. Las funciones de F son , por tanto, combinaciones lineales de similaridades de datos. Esto funciona de la misma forma que otros tipos de kernels.[58]

Teorema de representación El teorema de representación muestra que los modelos óptimos pueden ser escritos como un kernel cuántico tal que:

$$f_{opt}(x) = \sum_{m=1}^{M} \alpha_m tr[\rho(x^m)\rho(x)] \tag{4.18}$$

4.2.3. Entrenamiento de modelos cuánticos

Ya que el objetivo será utilizar los kernels cuánticos como modelo de aprendizaje automático estudiaremos como podemos entrenar y optimizar dichos modelos desde esta perspectiva. Por regla general las medidas óptimas en este tipo de problemas tienen pocos parámetros. De igual forma encontrar los modelos cuánticos óptimos puede ser formulado como un problema de baja dimensión. La optimización de estos parámetros es equivalente a optimizar los parámetros sobre el RKHS asociado al modelo.[49]

Entrenamiento variacional frente a kernels cuánticos Dado que el kernel sólo depende de la forma en que se codifican los datos en los estados cuánticos, se puede concluir que la codificación de los datos define el mínimo de una determinada función de costes utilizada para entrenar modelos cuánticos. Para encontrar el modelo que minimiza (4.18) podríamos usar un circuito variacional para entrenar un circuito, pero esto solo exploraría un pequeño subconjunto de modelos posibles. Para entrenar un circuito que pueda expresar cualquier modelo cuántico y encuentre el óptimo se necesitarían $O(2^{2n})$ parámetros. Los kernels cuánticos necesitarán muchos menos parámetros, lo que implica un mejor escalado.

Parámetros necesarios para encontrar el modelo óptimo Al usar un kernel cuántico como modelo, encontrar el modelo óptimo es un problema que requiere únicamente de M parámetros, siendo M el número de observaciones del conjunto de datos. En problemas con función de pérdida convexa no tendremos problemas en encontrar el mínimo eludiendo el problema de los «barren plateaus». Con la función de pérdida «Hinge Loss» podremos utilizar máquinas de vectores soporte clásicas con un kernel cuántico [50]. El modelo óptimo encontrado en un entrenamiento basado en kernels es el óptimo de los modelos cuánticos.

Equivalencia entre la optimización de parámetros del modelo y del RKHS

En aprendizaje automático clásico para obtener modelos óptimos debemos minimizar una función de pérdida asociada a un problema específico. A este proceso se le denomina entrenamiento. En un modelo cuántico el entrenamiento se puede expresar como la minimización del riesgo empírico regularizado.

Minimización del riesgo empírico regularizado en modelos cuánticos

En un espacio de entrada X y un espacio de salida Y donde p es la distribución de probabilidad en X y L una aplicación $X * Y * \mathbb{R} \to [0, \infty)$ utilizada como función de coste que cuantifica la calidad de la predicción de un modelo cuántico $f(x) = tr[\rho(x)M]$.

El riesgo esperado de f bajo L, donde L puede depender de x viene dado por:

$$\mathbf{R}_L(f) = \int_{X*Y} L(x, y, f(x))dp(x, y) \tag{4.19}$$

Como desconocemos p podemos aproximar la expresión anterior por:

$$\hat{\mathbf{R}}_L(f) = 1/M \sum_{m=1}^{M} L(x^m, y, f(x^m)) \tag{4.20}$$

La minimización del riesgo empírico regularizado de modelos cuánticos requiere minimizar el riesgo empírico de todos los modelos cuánticos y minimizar la norma de M

$$\inf_{M \in F} \lambda ||M||_F^2 + \hat{\mathbf{R}}_L(tr[\rho(x)M]) \tag{4.21}$$

Donde $\lambda \in \mathbb{R}^+$ es un hiperparámetro positivo que controla el término de regularización. Dado que las funciones del RKHS son equivalentes a modelos cuánticos y la norma de M es equivalente a la norma de f podemos reescribir la última ecuación como:

$$\inf_{f \in F} \gamma ||f||_F^2 + \hat{\mathbf{R}}_L(f) \tag{4.22}$$

Esta es la expresión que minimiza el RKHS de un modelo cuántico.[49]

Medidas de modelos cuánticos óptimos como expansiones en el conjunto de entrenamiento

La función de un RKHS que minimiza el riesgo empírico regularizado puede ser expresado como la suma ponderada del kernel entre x y el conjunto de entrenamiento. Junto a la conexión con los modelos cuánticos y el RKHS esto nos permite escribir modelos óptimos de aprendizaje automático.

Medidas óptimas

La medida que minimiza el riesgo empírico regularizado es:

$$M_{opt} = \sum_m \alpha_m \rho(x^m) \tag{4.23}$$

En circuitos variacionales solo se optimiza un subconjunto de RKHS ya que las medidas están restringidas por un ansatz concreto. Por tanto, no podemos garantizar que la medida óptima sea expresada por dicho circuito. No obstante, sí podemos garantizar que existirá una medida para la cual el modelo cuántico tendrá un riesgo empírico regularizado menor que la mejor solución de un entrenamiento variacional.[49]

La optimización del modelo cuántico es un problema de optimización convexa de baja dimensión

La optimización del riesgo empírico de las funciones de pérdida convexas sobre las funciones en un RKHS puede formularse como un problema de optimización convexa de dimensión finita [20].

El hecho de que el problema de optimización sea de dimensión finita es importante. Sin embargo, los espacios de características en los que el modelo clasifica los datos suelen ser de muy alta dimensión. Esto es

obviamente cierto para el espacio de características de codificación de datos cuánticos, que es precisamente la razón por la que el aprendizaje automático cuántico variacional parametriza los circuitos con un pequeño número de parámetros entrenables en lugar de optimizar sobre todas las unidades/medidas. Pero incluso si optimizamos sobre todos los modelos cuánticos, los resultados garantizan que la dimensionalidad del problema está limitada por el tamaño del conjunto de datos de entrenamiento. La dimensión de este problema es exactamente igual al número de instancias de datos de entrenamiento.[49]

El hecho de que la optimización sea convexa significa que sólo hay un mínimo global Los problemas de optimización convexa pueden resolverse aproximadamente en un tiempo $\mathbf{O}(M^2)$ en el número de datos de entrenamiento. Aunque es prohibitivo para grandes conjuntos de datos, hace que se garantice que la optimización sea manejable. Además los ordenadores cuánticos podrían, en principio, ayudar a entrenar con un tiempo de ejecución de $\mathbf{O}(M)$ [9]

Alineamiento kernel-objetivo

Para poder entrenar un kernel cuántico necesitamos alguna medida de cómo se ajusta al conjunto de datos en cuestión. Realizar una búsqueda exhaustiva en el espacio de parámetros no es una buena solución porque consume muchos recursos y, como la precisión es una cantidad discreta, podríamos no detectar pequeñas mejoras.

Sin embargo, podemos recurrir a una medida más especializada: la alineación kernel-objetivo (KTA por sus siglas en inglés de kernel target alignment). La alineación kernel-objetivo compara la similitud predicha por el kernel cuántico con las etiquetas de los datos de entrenamiento. Se basa en la alineación del kernel, una medida de similitud entre dos kernels dadas sus matrices.

$$KA(K_1K_2) = \frac{Tr(K_1K_2}{\sqrt{Tr(K_1^2Tr(K_2^2}} \tag{4.24}$$

Explicación teórica Visto desde un lado más teórico, KA es el coseno del ángulo entre las matrices del núcleo K_1 y K_2 interpretados como vectores en el espacio de matrices con el producto escalar de Hilbert-Schmidt (o Frobenius)

$$\langle A, B\rangle = Tr(A^T B)$$

Esto refuerza la idea de la geometría de esta medida, dos núcleos que se alinean en un espacio vectorial.

Los datos de entrenamiento definen una función kernel óptima que expresa el etiquetado original en el vector y asignando a un par de puntos de datos el producto de las etiquetas correspondientes:

$$k_y(x_i, x_j) = y_i y_j \tag{4.25}$$

Mediante esta fórmula obtenemos 1 si los datos pertenecen a la misma clase y -1 en caso contrario. La matriz del kernel viene dada por yy^T. Por lo que el alineamiento kernel-objetivo se define como el alineamiento entre K_1 y yy^T es decir:

$$KA_y(K_1) = \frac{Tr(K_1 yy^T)}{\sqrt{Tr(K_1^2 Tr((yy^T)^2))}} \tag{4.26}$$

Tener un alineamiento kernel-objetivo alto es necesario para obtener buenos resultados pero no garantiza una tasa elevada de acierto.

Capítulo 5

Simulación práctica

En este capítulo se explicará cómo ha sido elaborado el modelo para este proyecto.

El primer paso para realizar el proyecto es la obtención de datos. El conjunto de datos elegido contiene un conjunto de imágenes de perros y gatos y fue utilizado para una competición de Kaggle de Aprendizaje Automático consistente en determinar a qué animal se corresponde cada fotografía [25]. Este problema es uno de los ejemplos paradigmáticos de clasificación de imágenes.

Para acelerar el entrenamiento del modelo, se utilizará una GPU que proporciona Google en el entorno Colaboratory A.14.

5.1. Muestra

Esta es una muestra de las imágenes con las que se trabajará. Se ha obtenido mediante el código A.18.

Figura 5.1: Muestra de las imágenes de entrada

5.2. Red Convolucional

5.2.1. Parámetrros del modelo

Para esta primera parte de la construcción del modelo será necesario fijar algunos parámetros de entrenamiento. Para ello se asignan mediante prueba y error. Es importante que el entrenamiento no requiera de demasiados recursos y no es tan relevante que la tasa de acierto sea muy elevada ya que no es el clasificador objetivo del proyecto. El primero y más importante es el ratio de aprendizaje que determinará la variación de los parámetros del modelo tras clasificar cada instancia en este poceso y, por tanto, la estabilidad del mismo. Por otra parte, utilizaremos la técnica de agrupamiento por lotes (batches) para evitar un excesivo consumo de memoria, ya que el entorno de desarrollo podría desconectarse. Los participantes del reto original establecen generalmente entrenamientos de unas 100 épocas. No obstante, para ahorrar recursos cada lote será procesado en 5 épocas, es decir que evaluará cada instancia 5 veces para determinar el error cometido A.42.

5.2.2. Transformaciones de datos

Teniendo en cuenta que no todas las imágenes tienen el mismo tamaño, es necesario transformar las imágenes para poder entrenar la red neuronal. Entrenar una red con imágenes a distintas resoluciones puede producir problemas en la dispersión de los patrones reconocidos. [19]

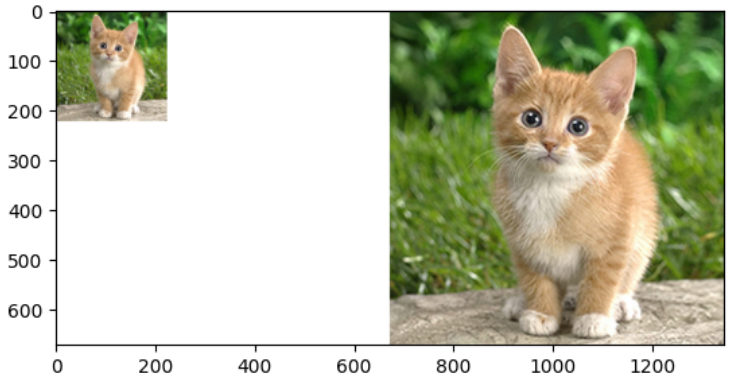


Figura 5.2: Escalado de una imagen [63]

Además de redimensionar la imagen, también se requerirá usar su información como tensor. A todos los conjuntos de datos se les aplicarán las mismas transformaciones A.20.

Para utilizar un conjunto de datos en la red a entrenar, PyTorch utiliza la clase *dataset* del paquete *utils*, la cual es necesario redefinir para obtener las instancias del mismo. Esto permitirá adaptarla al conjunto de datos con el que trabajamos estableciendo las operaciones necesarias para nuestro problema A.21.

5.2.3. Preparación de datos

El conjunto de datos de Kaggle viene ya dividido en un conjunto de entrenamiento y uno de prueba. No obstante, el conjunto de prueba no se encuentra etiquetado y, por tanto será necesario utilizar parte del conjunto de entrenamiento para la validación del modelo A.22.

5.2.4. Definición de la red convolucional

La estructura de la red convolucional se puede dividir en dos partes: la primera son las capas convolucionales y la segunda la red de capas densas. Para la definición de la red neuronal se usará la clase *Module* en el paquete de redes neuronales que es la base de los modelos en PyTorch, *nn*.

La parte convolucional de la red la forman tres capas compuestas de cuatro operaciones:

- La convolución que agrupa partes de cada imagen en varias más pequeñas, con un tamaño de kernel de 3 y un desplazamiento de 2 píxeles.
- Una operación de normalización de lote para evitar el posible sesgo en los parámetros
- Una función de activación ReLU
- El «pooling» para reducir la dimensión

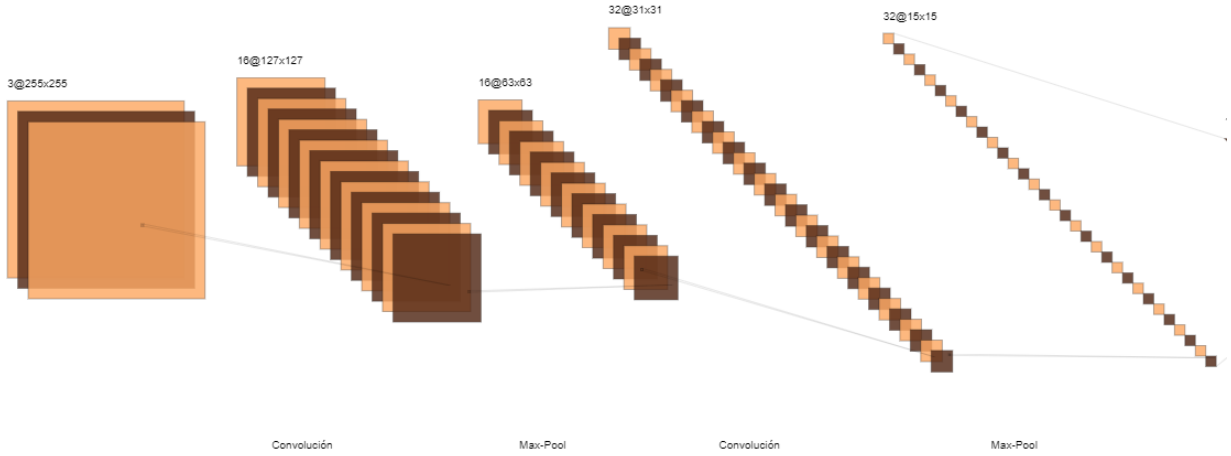


Figura 5.3: Estructura de la red convolucional

Por otra parte se utilizarán otras tres capas también con función de activación ReLU pero esta vez densamente conectadas para extraer aún más las características de los datos. La salida de la segunda de estas capas será la que utilicemos como entrada para la siguiente parte. La salida de la última capa se utiliza como entrenamiento de la CNN A.27.

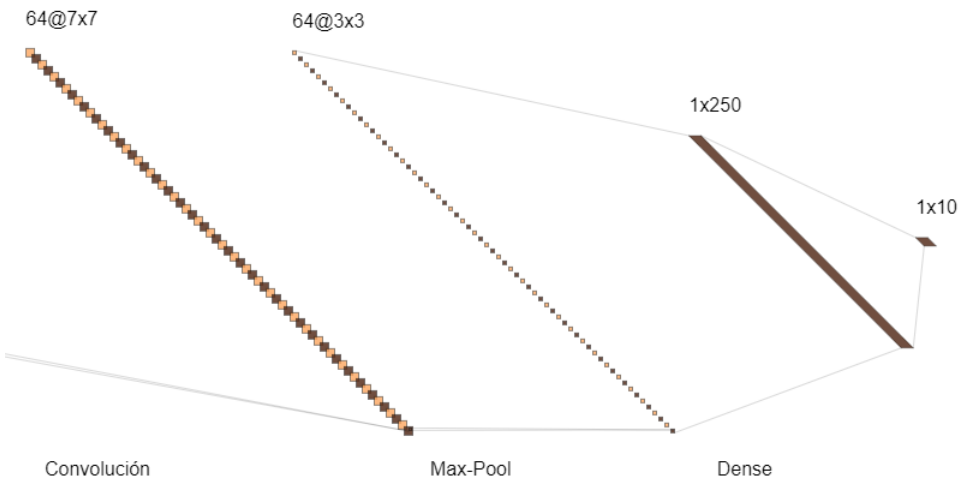


Figura 5.4: Estructura de la red densa

5.2.5. Entrenamiento

Para la optimización de parámetros se utilizará el optimizador Adam con el ratio de aprendizaje fijado A.28. Ya que nos encontramos con un problema de clasificación binaria, utilizaremos la entropía cruzada como función de pérdida.

Se utilizarrá un bucle para optimizar el modelo y se obtendrá la tasa de acierto tanto para el conjunto de entrenamiento como para el de validación A.29.

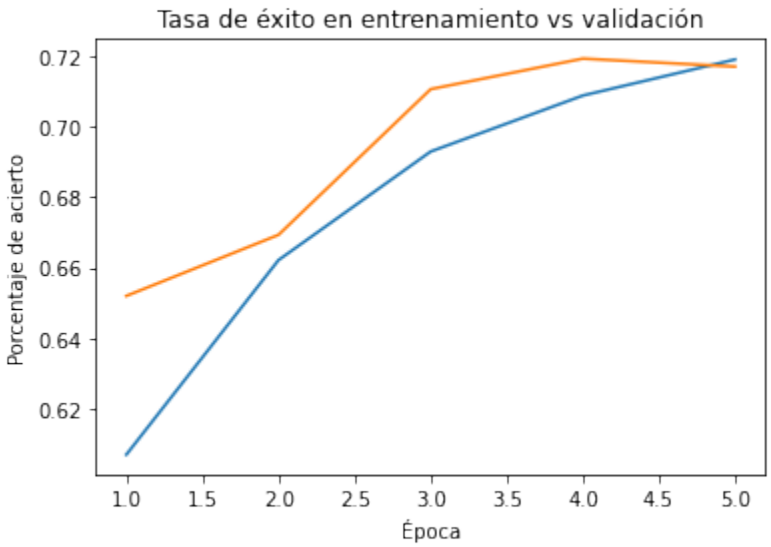


Figura 5.5: Tasa de acierto por épocas

Donde la línea azul representa la tasa de acierto en el entrenamiento y la naranja en validación. El modelo

entrenado obtiene una tasa de éxito del 71 %.

5.3. QSVM

5.3.1. Preparación de datos

Para la parte cuántica del modelo, se necesitará utilizar la salida de la penúltima capa de la red neuronal. Ésta contiene un vector de diez características extraídas de cada instancia.

Obtención de las características extraídas de la red convolucional

Se utilizará el modelo ya entrenado para obtener las características del conjunto de entrenamiento. Estas características se utilizarán para entrenar también el QSVM. Posteriormente, se obtiene también el conjunto de validación de manera análoga A.30.

Selección y análisis previo de las instancias de entrenamiento

Como se ha explicado en otras secciones, no se necesitará conocer el valor de cada punto en el nuevo espacio de alta dimensión, pero sí la distancia entre ellos. El problema que esto acarrea es el elevado coste computacional de añadir instancias al conjunto de entrenamiento. Esto no supondría un problema en un ordenador cuántico real, ya que se podría utilizar el paralelismo que proporciona el entrelazamiento de qúbits para acelerar el proceso. No obstante, al realizar una simulación en un ordenador clásico, utilizar todas las instancias de las que disponemos es inviable. Esto nos lleva a seleccionar 100 instancias para entrenar el modelo. Utilizaremos otras 100 para validarlo A.34.

Ya que la muestra es relativamente pequeña, es adecuado comprobar que, en el subconjunto seleccionado, ambas clases se encuentren de forma razonablemente balanceada. Para ello, se realiza una gráfica simple de la cantidad de instancias de cada clase A.36.

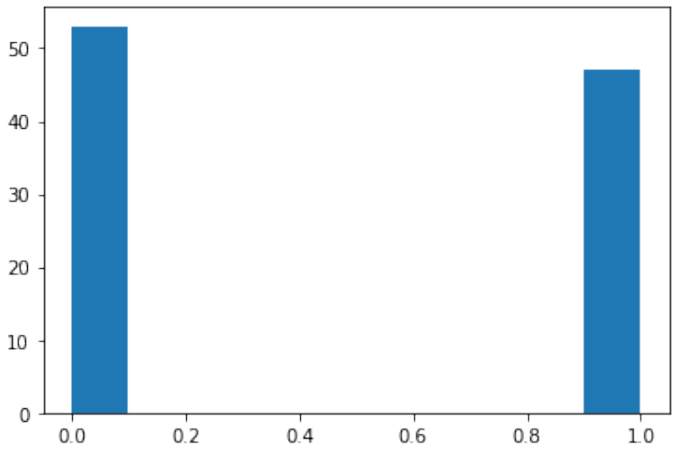


Figura 5.6: Gráfica del balance de ambas clases

Como se observa no hay demasiada diferencia entre ambas clases.

5.3.2. Definición del circuito cuántico

En este apartado se mostrará como se ha definido el circuito cuántico.

El primer paro es definir una capa para el circuito. En nuestro caso, cada capa se compondrá por una serie de operaciones unitarias, más concretamente una puerta Hadamard, y dos rotaciones.

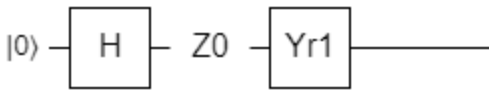


Figura 5.7: Parte no entrelazada de la capa del ansatz

A estas operaciones se le añaden otras puertas que entrelacen los qúbits mediante rotaciones con control, siguiendo el patrón anillo A.37.

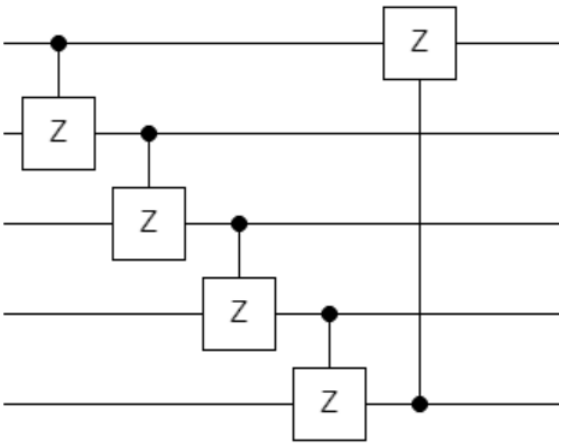


Figura 5.8: Patrón anillo

El ansatz se compone de varias de las capas definidas anteriormente.

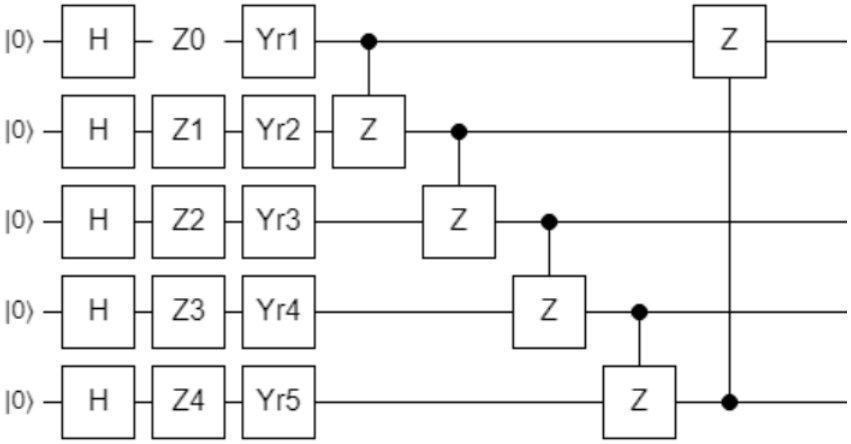


Figura 5.9: Forma de cada una de las capas que componen el circuito

Las puertas Z0, Z1, Z2, Z3 y Z4 son las encargadas de realizar el embedding. Al utilizar únicamente 5 qúbits solo es posible codificar 5 características. Este tipo de codificación es el denominado Angle Embedding. Añadiendo las tres capas el circuito tiene la siguiente forma:

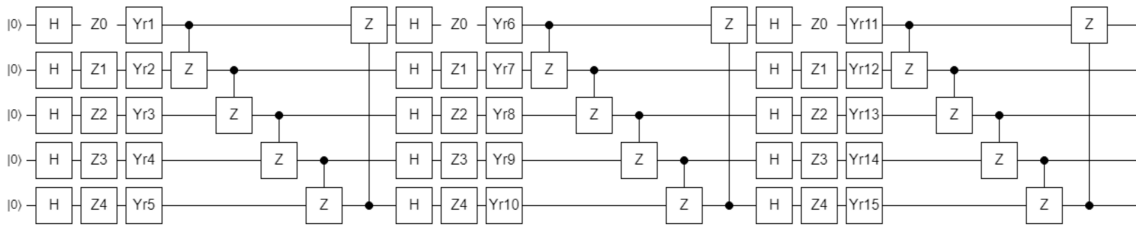


Figura 5.10: Forma del circuito

y posteriormente se añade el circuito de su conjugado compuesto donde se codifican las características de una segunda instancia A.38.

Se define una función que genera parámetros aleatorios entre 0 y 2π para inicializar el circuito a partir del número de qúbits y el número de capas del circuito A.39. Podemos comprobar las amplitudes de los estados de los qúbits utilizando parámetros e instancias aleatorias:

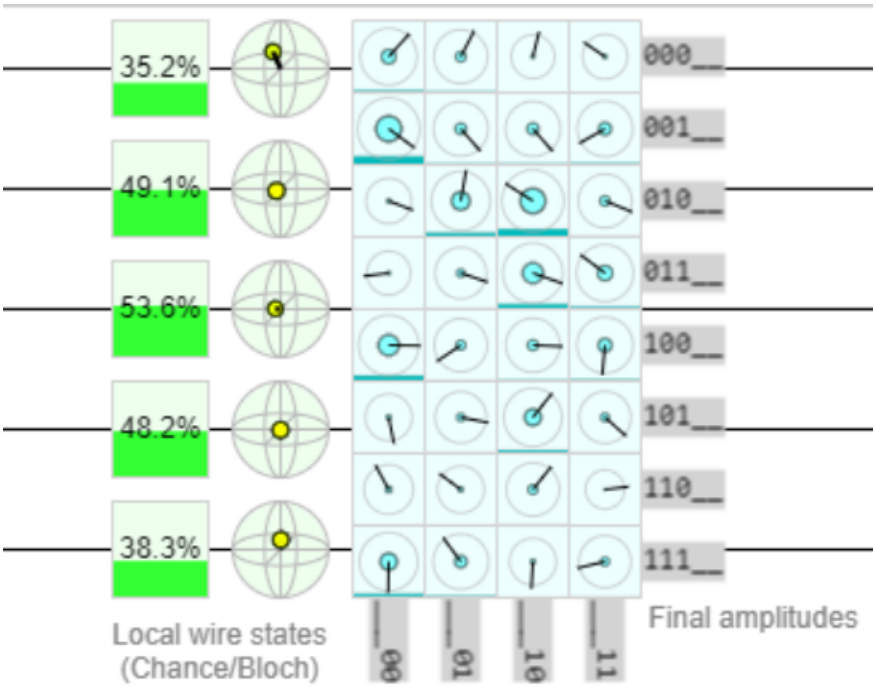


Figura 5.11: Amplitud de los estados medibles

Se define el kernel cuántico como la unión de los dos circuitos creados anteriormente y se enmarcan dentro de un nodo cuántico. Las mediciones de este kernel serán exactas ya que se utilizarán las probabilidades de medición en lugar de una muestra de mediciones. Se utiliza la probabilidad de observar todos los qúbits a 0 para realizar la separación A.40.

5.3.3. Modelo sin entrenar

Se elijen de forma arbitraria utilizar 3 capas y se simulan 5 qúbits ya que excederse en esto aumentaría demasiado la complejidad del circuito y utilizaremos únicamente cinco caracteríasticas de la CNN. El kernel inicial será el circuito creado con parámetros aleatorios A.42. A partir de este, es posible obtener la matriz por pares.

$$\begin{bmatrix} 1 & 0{,}629 & 0{,}067 & \dots & 0{,}024 & 0{,}007 & 0{,}573 \\ 0{,}629 & 1 & 0{,}279 & \dots & 0{,}339 & 0{,}099 & 0{,}904 \\ 0{,}067 & 0{,}279 & 1 & \dots & 0{,}233 & 0{,}392 & 0{,}23 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0{,}024 & 0{,}339 & 0{,}233 & \dots & 1 & 0{,}31 & 0{,}398 \\ 0{,}007 & 0{,}099 & 0{,}392 & \dots & 0{,}31 & 1 & 0{,}138 \\ 0{,}573 & 0{,}904 & 0{,}23 & \dots & 0{,}398 & 0{,}138 & 1 \end{bmatrix}$$

Para la creación del SVM se utiliza la librería Scikit-learn.

```
#Declaramos un modelo de maquinas de vectores soporte con el kernel
    definido anteriormente
svm = SVC(kernel=lambda X1, X2: qml.kernels.kernel_matrix(X1, X2,
    init_kernel)).fit(X_train, Y_train)
```

Listing 5.1: Kernel

La tasa de acierto del clasificador obtenido es de un 66 % A.45. No es una tasa de éxito demasiado elevada para el problema ya que el clasificador del ganador del concurso resolvió el problema con una tasa de acierto del 98, 9 %. Esto es probablemente debido a la escasez de observaciones derivada de la simulación de computación cuántica. No obstante, sometiendo el resultado a un contraste de hipótesis, podemos rechazar que la tasa de acierto del clasificador sea igual o menor al 50 %, por lo que el clasificador sí es válido.

5.3.4. Modelo con entrenamiento

Para comprobar la bondad del kernel se utilizará su alineamiento kernel-objetivo como medida. La librería Pennylane nos permite calcular dicho parámetro.

No obstante, la función *target_alignment* de Pennylane no es optimizable, por lo que se utilizará una función equivalente que será definida en el cuaderno.

```
def target_alignment(
    X,
    Y,
    kernel,
    assume_normalized_kernel=False,
    rescale_class_labels=True,
):
    #KTA del kernel y las etiquetas
    K = qml.kernels.square_kernel_matrix(
        X,
        kernel,
        assume_normalized_kernel=assume_normalized_kernel,
    )

   #Transformamos las etiquetas y obtenemos el producto interior por pares
    if rescale_class_labels:
        nplus = np.count_nonzero(np.array(Y) == 1)
```

```
        nminus = len(Y) - nplus
        _Y = np.array([y / nplus if y == 1 else -1 / nminus for y in Y])
    else:
        _Y = np.array(Y)

    T = np.outer(_Y, _Y)
    inner_product = np.sum(K * T)
    norm = np.sqrt(np.sum(K * K) * np.sum(T * T))
    inner_product = inner_product / norm

    return inner_product
```

Listing 5.2: Definición de la función del KTA

El KTA para el modelo con parámetros aleatorios y el conjunto de datos es: 0.067 A.48.

Se optimiza el KTA mediante descenso de gradiente. Debido a que la función de Pennylane encargada de esta tarea minimiza la función que se le proporciona, se utilizará la función opuesta. Se ha mejora ligeramente el KTA A.49.

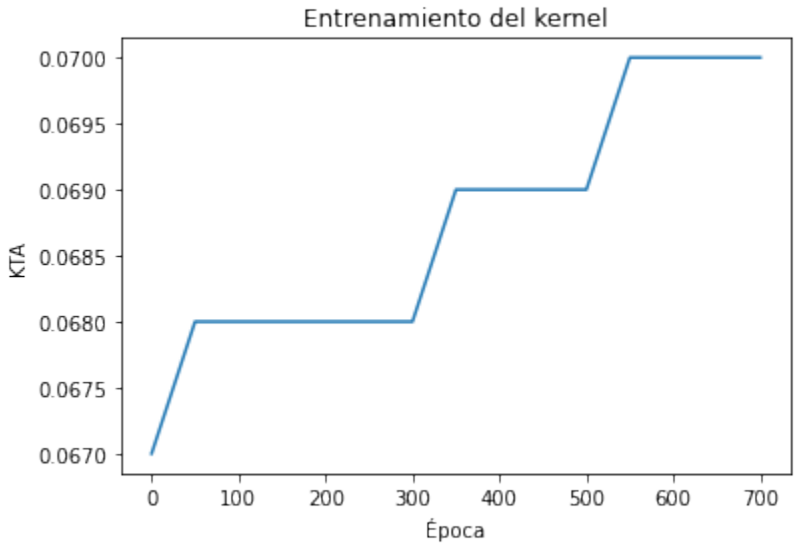


Figura 5.12: Entrenamiento del KTA

Se define el nuevo kernel, su matriz y un nuevo modelo con los parámetros del circuito entrenados y comprobamos la tasa de éxito del modelo entrenado A.50.

Tasa de acierto

La tasa de éxito es del 67 %. El resultado es ligeramente mejor con el modelo entrenado.

5.4. Resultados

Para finalizar el experimento, se define la función del modelo A.52 y se pone a prueba con el conjunto test A.53.

Se realiza finalmente una muestra de 48 imágenes test aleatorias con la etiqueta predicha A.54.

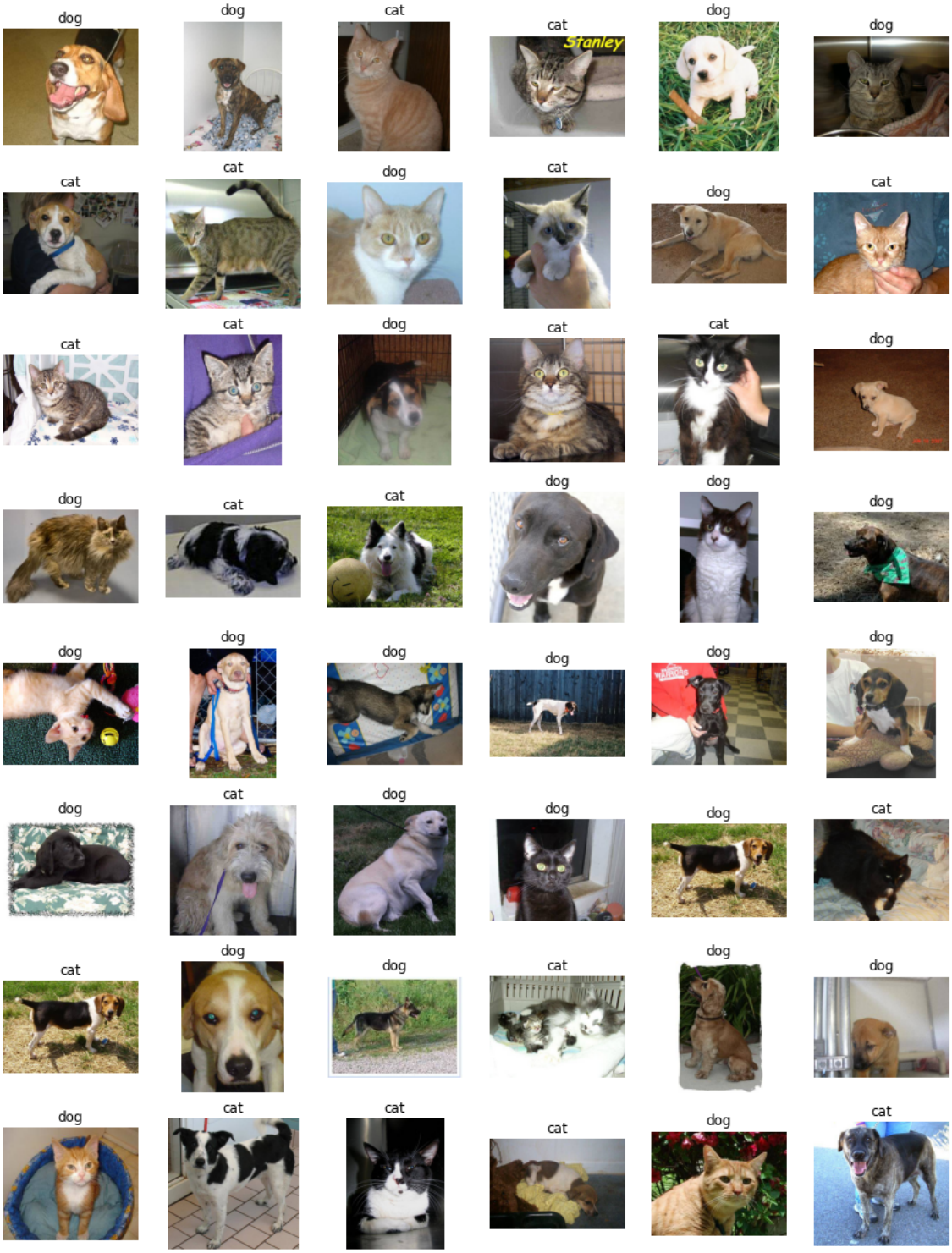


Figura 5.13: Resultados de la aplicación del modelo sobre el conjunto test

Capítulo 6

Resultados

6.1. Conclusiones

6.1.1. Aplicación del fundamento teórico

Inteligencia Artificial

La realización de este proyecto ha supuesto la puesta en práctica de varios conceptos relacionados con el Aprendizaje Automático. El uso de redes convolucionales es usado frecuentemente para clasificación de imágenes, no obstante el uso que se ha dado a esta técnica es la extracción de características. El cambio de objetivo no ha modificado la estructura de la misma que ha sido entrenada como un modelo para la clasificación. Para ello se han utilizado técnicas frecuentes en las CNN como las transformaciones de datos o el «pooling».

Por otra parte, las máquinas de vectores soporte son utilizadas habitualmente en problemas de clasificación binaria. La utilidad que aportan en dicha tarea se debe a las eficientes técnicas que corrigen los problemas que pueden surgir al utilizarlas directamente sobre el conjunto de datos. Entre estas técnicas se ha destacado el uso de kernels, utilizando en este caso técnicas de computación cuántica para este fin.

Computación cuántica

La computación cuántica es un nuevo paradigma de computación que busca explotar el potencial de la superposición cuántica para mejorar la resolución de ciertas tareas mediante el paralelismo intrínseco en los bits cuánticos. El uso de este paradigma no proporciona resolución a ningún problema no computable mediante técnicas clásicas, como el problema de la parada, no obstante, proporciona de forma teórica una eficiencia notablemente mayor en algunos problemas como la factorización.

Existen también numerosos problemas con la computación cuántica. El primero de todos es que no es eficiente para todo tipo de tareas, lo que nos puede llevar a buscar un enfoque híbrido. Aunque el principal problema que encontramos a día de hoy es la escasez de qúbits en los sistemas cuánticos disponibles. Debido a la dificultad para que el hardware permita aislar físicamente los sistemas cuánticos de posibles perturbaciones externas, no disponemos de un computador cuántico fiable. La solución para este problema de sensibilidad al ruido pasa por aumentar el número de qúbits de un sistema lo cual supone un aumento inasequible del coste. Actualmente nos encontramos en la era «NISQ» con computadores de 50-100 qúbits en la que algunos ordenadores cuánticos ya sobrepasan a ordenadores clásicos en ciertas tareas. No obstante todavía estamos lejos de los sistemas tolerantes a fallos o «QPUs» de propósito general.

6.1.2. Técnicas de computación cuántica para el Aprendizaje automático

Aprendizaje Automático Cuántico

El auge del Aprendizaje Automático y el uso de algoritmos de programación diferenciable promueven la búsqueda de la agilización de los procesos de entrenamiento y clasificación de modelos. Esto recae directamente sobre la computación cuántica ya que la forma de algunos circuitos cuánticos parametrizados tienen la forma de una red neuronal clásica. Este hecho promueve el uso de la computación cuántica como sustituto natural de este tipo de clasificadores en los que los parámetros libres se optimizan mediante algoritmos relacionados con el gradiente, calculado de forma automática. A esto se puede añadir el carácter probabilístico tanto de la computación cuántica como de los modelos de aprendizaje automático.

Kernels cuánticos

Los kernels cuánticos aprovechan la alta dimensionalidad generada por el entrelazamiento cuántico de un pequeño número de qúbits para separar linealmente los datos. Al igual que en el uso de kernels clásicos, no es necesario calcular el valor que proporciona el mapa de características sobre cada dato, sino que es suficiente con conocer la similaridad en el espacio de Hilbert definida mediante el producto interior de los vectores generados por la codificación en estados cuánticos de cada par de observaciones.

El espacio de funciones en el que viven los kernels, es decir, el RKHS incluye el espacio de los modelos cuánticos. Esto, sumado a que buscar la mejor función dentro del RKHS es un problema de optimización covexa hace que un kernel cuántico mejore de forma teórica a cualquier otro algoritmo cuántico en la tarea de clasificación.

El problema de optimización de un kernel cuántico es de baja dimensión en un computador cuántico. No obstante, al realizar una simulación, un algoritmo cuántico no supone nunca una ventaja respecto a un algoritmo clásico. Mediante el KTA es posible optimizar un circuito en la tarea de clasificación, con la desventaja de no ser un problema de optimización convexa.

6.1.3. Resultados del modelo

Tras construir y evaluar el modelo, se puede comprobar la cantidad de dificultades a la hora de implementar este modelo utilizando la tecnología actual. Realizar una simulación cuántica en un computador clásico no es eficiente, esto ha implicado reducir el tamaño del conjunto de entrenamiento de forma drástica, lo cual implica una menor precisión en los resultados finales. La tasa de éxito obtenida inicialmente es de un 66 % y, tras el entrenamiento del circuito, de un 67 %. Pese a que el tiempo de entrenamiento ha sido superior a una hora, el modelo prácticamente no ha mejorado.

Teniendo en cuenta las altas tasas de éxito que se han logrado al resolver este problema mediante técnicas clásicas, es evidente que este experimento no tiene un alcance práctico a día de hoy. Sin embargo, la tecnología cuántica tiene una gran proyección de avance en las próximas décadas, pudiendo provocar grandes cambios en la concepción de los modelos de Aprendizaje Automático.

6.2. Futuras líneas de trabajo

Tras realizar este experimento, es posible pensar en otros nuevos que pueden ampliar el alcance de este.

- Al igual que en Aprendizaje Automático clásico es frecuente realizar un «benchamrking» de los distintos modelos en función de parámetros como el número de iteraciones de optimización y, sobre todo, el ratio de aprendizaje, podría ser interesante realizar un estudio sobre los distintos parámetros tanto de la red convolucional como del ansatz para comprobar como afectan a la bondad del modelo. Los principales problemas de esto son el alto coste que supondría aumentar el tiempo de entrenamiento y la posibilidad de llegar al sobreajuste.
- Una posible mejora que se puede proporcionar al modelo es el uso de más instancias de entrenamiento ya que el número de éstas es demasiado reducido. Como ya se ha visto esto requerirá una mejora significativa en el hardware ya que implica calcular más pares de distancias.
- Aunque el circuito cuántico utilizado no influye directamente sobre la clasificación realizada, sería interesante comprobar si existen circuitos con formas que son mejores separando instancias o sean más sencillos de optimizar.
- La computación cuántica también ofrece otros métodos para construir clasificadores. Uno de ellos es realizar la clasificación directamente midiendo los qúbits de un ansatz con parámetros entrenables. Otra posibilidad que se adaptaría mejor a este mismo problema sería el uso de redes «cuanvolucionales» que tratan imágenes de forma que cada píxel es procesado por una entrada de un circuito reduciendo su dimensionalidad al ser medida. En este caso se utilizaría el circuito cuántico como sustituto de las convoluciones.
- Otra posibilidad sería realizar «Transfer Learning» utilizando una red preentrenada como ResNet como paso previo a la red convolucional. Esto podría empeorar los resultados ya que no esta entrenada con datos exclusivos del dominio del problema. No obstante el tiempo de entrenamiento del modelo completo vería reducido y permitiría dedicar más tiempo a la parte cuántica.
- Existen ciertos servicios de algunas empresas como IBM que permiten ejecutar código en ordenadores cuánticos de forma remota. Un trabajo interesante sería comprobar el funcionamiento del modelo en estas máquinas. Los computadores todavía son susceptibles al ruido por lo que los resultados podrían no ser los esperados. También existen simuladores que se ejecutan dentro de ordenadores cuánticos reales.

Bibliografía

[1] Saad Albawi, Tareq Abed Mohammed y Saad Al-Zawi. «Understanding of a convolutional neural network». En: *2017 International Conference on Engineering and Technology (ICET)*. 2017, págs. 1-6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[2] Rami A. Alzahrani y Alice C. Parker. «Neuromorphic Circuits With Neural Modulation Enhancing the Information Content of Neural Signaling». En: *International Conference on Neuromorphic Systems 2020*. ICONS 2020. Oak Ridge, TN, USA: Association for Computing Machinery, 2020. ISBN: 9781450388511. DOI: 10.1145/3407197.3407204. URL: https://doi.org/10.1145/3407197.3407204.

[3] Juan Miguel Arrazola, Patrick Rebentrost y Christian Weedbrook. *Quantum supremacy and high-dimensional integration*. 2017. DOI: 10.48550/ARXIV.1712.07288. URL: https://arxiv.org/abs/1712.07288.

[4] Astronoo. *La computación cuántica*. [Online; accessed April 27, 2022]. 2022. URL: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSV_rCLQmeaK_anfuRoyCB9IMtQum-gI5ptoA&usqp=CAU.

[5] Peter Auer, Harald Burgsteiner y Wolfgang Maass. «A learning rule for very simple universal approximators consisting of a single layer of perceptrons». En: *Neural networks* 21.5 (2008), págs. 786-795.

[6] Suman Bathula. *Single Layer Perceptron*. URL: http://sumanthrb.com/wp-content/uploads/2019/05/SLP.jpg.

[7] Atilim Gunes Baydin y col. «Automatic differentiation in machine learning: a survey». En: (2015). DOI: 10.48550/ARXIV.1502.05767. URL: https://arxiv.org/abs/1502.05767.

[8] Yoshua Bengio, Aaron Courville y Pascal Vincent. «Representation Learning: A Review and New Perspectives». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), págs. 1798-1828. DOI: 10.1109/TPAMI.2013.50.

[9] Stephen Boyd, Stephen P Boyd y Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[10] M. Cerezo y col. «Cost function dependent barren plateaus in shallow parametrized quantum circuits». En: *Nature Communications* 12.1 (mar. de 2021). DOI: 10.1038/s41467-021-21728-w. URL: https://doi.org/10.1038%2Fs41467-021-21728-w.

[11] Ting-Hao Chen. *What is "padding" in Convolutional Neural Network?* URL: https://miro.medium.com/max/650/1*b77nZmPH15dE8g49BLW20A.png.

[12] Ting-Hao Chen. *What is "stride" in Convolutional Neural Network?* URL: https://miro.medium.com/max/706/1*EzHsj8GxCZ7pjfwL2ugFwQ.png.

[13] Andreas Christmann e Ingo Steinwart. «Support vector machines». En: (2008).

[14] Carlo Ciliberto y col. «Quantum machine learning: a classical perspective». En: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2209 (2018), pág. 20170551.

[15] Paul Dirac. *The Principles of Quantum Mechanics*. Oxford University Press, 1958.

[16] Yuxuan Du y col. «Expressive power of parametrized quantum circuits». En: *Physical Review Research* 2.3 (jul. de 2020). DOI: 10.1103/physrevresearch.2.033125. URL: https://doi.org/10.1103%2Fphysrevresearch.2.033125.

[17] Edward Farhi, Jeffrey Goldstone y Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. DOI: 10.48550/ARXIV.1411.4028. URL: https://arxiv.org/abs/1411.4028.

[18] Vojtěch Havlıéček y col. «Supervised learning with quantum-enhanced feature spaces». En: *Nature* 567.7747 (mar. de 2019), págs. 209-212. DOI: 10.1038/s41586-019-0980-2. URL: https://doi.org/10.1038%2Fs41586-019-0980-2.

[19] Andrew G. Howard. *Some Improvements on Deep Convolutional Neural Network Based Image Classification*. 2013. DOI: 10.48550/ARXIV.1312.5402. URL: https://arxiv.org/abs/1312.5402.

[20] Hsin-Yuan Huang y col. «Power of data in quantum machine learning». En: *Nature Communications* 12.1 (mayo de 2021). DOI: 10.1038/s41467-021-22539-9. URL: https://doi.org/10.1038%2Fs41467-021-22539-9.

[21] William Huggins y col. «Towards quantum machine learning with tensor networks». En: *Quantum Science and Technology* 4.2 (ene. de 2019), pág. 024001. DOI: 10.1088/2058-9565/aaea94. URL: https://doi.org/10.1088%2F2058-9565%2Faaea94.

[22] Teijiro Isokawa, Haruhiko Nishimura y Nobuyuki Matsui. «Quaternionic multilayer perceptron with local analyticity». En: *Information* 3.4 (2012), págs. 756-770.

[23] Gareth James y col. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[24] Caleb Johnson. *Training Quantum Kernels for Machine Learning Using Qiskit*. https://medium.com/qiskit/training-quantum-kernels-for-machine-learning-using-qiskit-617f6e4ed9ac.

[25] Kaggle. *Dogs vs. Cats Kaggle competition*. URL: https://www.kaggle.com/c/dogs-vs-cats.

[26] Nathan Killoran y col. «Continuous-variable quantum neural networks». En: *Physical Review Research* 1.3 (oct. de 2019). DOI: 10.1103/physrevresearch.1.033063. URL: https://doi.org/10.1103%2Fphysrevresearch.1.033063.

[27] Laptrinx. *Math behind Support Vector Machine(SVM)*. URL: https://cdn-images-1.medium.com/freeze/max/1000/1*yAE4__VwfSOk9ohcMtccPg.png?q=20.

[28] Álexander Lenail. *NN-SVG*. URL: http://alexlenail.me/NN-SVG/index.html.

[29] Yunchao Liu, Srinivasan Arunachalam y Kristan Temme. «A rigorous and robust quantum speed-up in supervised machine learning». En: *Nature Physics* 17.9 (2021), págs. 1013-1017.

[30] Seth Lloyd y col. *Quantum embeddings for machine learning*. 2020. DOI: 10.48550/ARXIV.2001.03622. URL: https://arxiv.org/abs/2001.03622.

[31] George F Luger. *Artificial intelligence: structures and strategies for complex problem solving*. Pearson education, 2005.

[32] Mathworks. *Hiperplanos óptimos como límites de decisión*. URL: https://es.mathworks.com/discovery/support-vector-machine/_jcr_content/mainParsys/image.adapt.full.medium.jpg/1630399112061.jpg.

[33] Chris Mellish. *Machine Learning*. 1997.

[34] K. Mitarai y col. «Quantum circuit learning». En: *Physical Review A* 98.3 (sep. de 2018). DOI: 10.1103/physreva.98.032309. URL: https://doi.org/10.1103%2Fphysreva.98.032309.

[35] Tom M Mitchell y Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[36] Universidad de Murcia. *Nuevas Tecnologías y Contaminación de Atmósferas, para PYMEs*. URL: https://www.um.es/LEQ/Atmosferas/Ch-VI-3/6-3-8.GIF.

[37] Michael A Nielsen e Isaac Chuang. *Quantum computation and quantum information*. 2002.

[38] William S Noble. «What is a support vector machine?» En: *Nature biotechnology* 24.12 (2006), págs. 1565-1567.

[39] Alberto Peruzzo y col. «A variational eigenvalue solver on a photonic quantum processor». En: *Nature Communications* 5.1 (jul. de 2014). DOI: 10.1038/ncomms5213. URL: https://doi.org/10.1038%2Fncomms5213.

[40] Derek A Pisner y David M Schnyer. «Support vector machine». En: *Machine learning*. Elsevier, 2020, págs. 101-121.

[41] John Preskill. «Quantum Computing in the NISQ era and beyond». En: *Quantum* 2 (ago. de 2018), pág. 79. DOI: 10.22331/q-2018-08-06-79. URL: https://doi.org/10.22331%2Fq-2018-08-06-79.

[42] Somayeh Bakhtiari Ramezani y col. «Machine learning algorithms in quantum computing: A survey». En: *2020 International joint conference on neural networks (IJCNN)*. IEEE. 2020, págs. 1-8.

[43] research.aimultiple. *What is Data Augmentation? Techniques and Examples*. URL: https://research.aimultiple.com/wp-content/webp-express/webp-images/uploads/2021/04/dataaugmention_image_alletranitons.png.webp.

[44] Researchgate. *Illustration of Max Pooling and Average Pooling*. URL: https://www.researchgate.net/publication/333593451/figure/fig2/AS:765890261966848@1559613876098/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max.png.

[45] Eleanor G. Rieffel y Wolfgang Polak. «An Introduction to Quantum Computing for Non-Physicists». En: (1998). DOI: 10.48550/ARXIV.QUANT-PH/9809016. URL: https://arxiv.org/abs/quant-ph/9809016.

[46] Jonathan Romero, Jonathan P Olson y Alan Aspuru-Guzik. «Quantum autoencoders for efficient compression of quantum data». En: *Quantum Science and Technology* 2.4 (ago. de 2017), pág. 045001. DOI: 10.1088/2058-9565/aa8072. URL: https://doi.org/10.1088%2F2058-9565%2Faa8072.

[47] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks*. URL: http://sumanthrb.com/wp-content/uploads/2019/05/SLP.jpg.

[48] Jürgen Schmidhuber. «Deep learning in neural networks: An overview». En: *Neural Networks* 61 (ene. de 2015), págs. 85-117. DOI: 10.1016/j.neunet.2014.09.003. URL: https://doi.org/10.1016%2Fj.neunet.2014.09.003.

[49] Maria Schuld. *Supervised quantum machine learning models are kernel methods*. 2021. DOI: 10.48550/ARXIV.2101.11020. URL: https://arxiv.org/abs/2101.11020.

[50] Maria Schuld y Nathan Killoran. «Quantum Machine Learning in Feature Hilbert Spaces». En: *Physical Review Letters* 122.4 (feb. de 2019). DOI: 10.1103/physrevlett.122.040504. URL: https://doi.org/10.1103%2Fphysrevlett.122.040504.

[51] Maria Schuld y Francesco Petruccione. *Supervised learning with quantum computers*. Vol. 17. Springer, 2018.

[52] Maria Schuld, Ilya Sinayskiy y Francesco Petruccione. «The quest for a Quantum Neural Network». En: *Quantum Information Processing* 13.11 (ago. de 2014), págs. 2567-2586. DOI: 10.1007/s11128-014-0809-8. URL: https://doi.org/10.1007%2Fs11128-014-0809-8.

[53] Maria Schuld y col. «Circuit-centric quantum classifiers». En: *Physical Review A* 101.3 (mar. de 2020). DOI: 10.1103/physreva.101.032308. URL: https://doi.org/10.1103%2Fphysreva.101.032308.

[54] Maria Schuld y col. «Evaluating analytic gradients on quantum hardware». En: *Physical Review A* 99.3 (mar. de 2019). DOI: 10.1103/physreva.99.032331. URL: https://doi.org/10.1103%2Fphysreva.99.032331.

[55] Sagar Sharma, Simone Sharma y Anidhya Athaiya. «Activation functions in neural networks». En: *towards data science* 6.12 (2017), págs. 310-316.

[56] Dan Shepherd y Michael J. Bremner. «Temporally unstructured quantum computation». En: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 465.2105 (feb. de 2009), págs. 1413-1439. DOI: 10.1098/rspa.2008.0443. URL: https://doi.org/10.1098%2Frspa.2008.0443.

[57] Gregory R. Steinbrecher y col. *Quantum optical neural networks*. 2018. DOI: 10.48550/ARXIV.1808.10047. URL: https://arxiv.org/abs/1808.10047.

[58] I. Steinwart y A. Christmann. *Support Vector Machines*. Information Science and Statistics. Springer New York, 2008. ISBN: 9780387772424. URL: https://books.google.es/books?id=HUnqnrpYt4IC.

[59] S. Suryansh. *Gradient Descent: All You Need to Know*. URL: https://miro.medium.com/proxy/1*f9a162GhpMbiTVTAua_lLQ.png.

[60] Pennylane dev team. *Quantum Node*. URL: https://pennylane.ai/qml/glossary/quantum_node.html.

[61] Guillaume Verdon, Michael Broughton y Jacob Biamonte. *A quantum algorithm to train neural networks using low-depth circuits*. 2017. DOI: 10.48550/ARXIV.1712.05304. URL: https://arxiv.org/abs/1712.05304.

[62] Wikimedia. *Convolutional Neural Network with Color Image Filter*. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/Convolutional_Neural_Network_with_Color_Image_Filter.gif/640px-Convolutional_Neural_Network_with_Color_Image_Filter.gif.

[63] Joshua Z.Zhang. *Compile ONNX Models*. 2013. URL: https://tvm.apache.org/docs/_images/sphx_glr_from_onnx_001.png.

[64] Frank Zickert. *What Does NISQ mean?* URL: https://towardsdatascience.com/what-does-nisq-mean-fadf1aa9fdd6.

Appendices

Apéndice A

Código de la parte experimental

A.1. Kaggle

Para la carga del conjunto de datos en nuestro sistema se utilizará la API que proporciona Kaggle. Se instalará dicha API mediante el comando pip del sistema.

```
! pip install kaggle
```

Listing A.1: Instalación de la API de kaggle

Para utilizar la API es necesario el archivo kaggle.json que se guardará en un directorio.

```
! mkdir ~/.kaggle
! cp /content/drive/MyDrive/universidad/TFGinformatica/kaggle.json ~/.
   kaggle/kaggle.json
! chmod 600 ~/.kaggle/kaggle.json #Modificamos los permisos de la carpeta
```

Listing A.2: Estableciendo directorios y permisos

Utilizamos la API para decargar el conjunto de datos

```
! kaggle competitions download -c dogs-vs-cats
```

Listing A.3: Carga de datos

Como los datos se encuentran comprimidos, utilizaremos el comando unzip para descomprimirlos y poder utilizarlos.

```
! unzip /content/dogs-vs-cats.zip
```

Listing A.4: Descompresión de archivos 1

```
! unzip /content/train.zip
```

Listing A.5: Descompresión de archivos 2

```
! unzip /content/test1.zip
```

Listing A.6: Descompresión de archivos 3

A.2. Instalación de librerías

La librería Pennylane requiere de instalación previa en la máquina virtual en la que se trabaja.

A.2.1. Instalación de Pennylane

```
!pip install pennylane
```

Listing A.7: Instalación de Pennylane

Es necesario hacer uso de librerías que permitan la gestión de archivos para el manejo de datos.

A.2.2. Importación de librerías

```
import os
import cv2
import copy
import zipfile
import glob
```

Listing A.8: Importación de librerías de manejo de directorios y archivos

Para poder visualizar los datos y los resultados se hará uso de algunas librerías gráficas.

```
import tqdm
from PIL import Image
import matplotlib.pyplot as plt
```

Listing A.9: Importación de librerías gráficas

Se utilizará Pandas para realizar operaciones sobre el conjunto de datos y las librerías de Scikit-learn para máqunas de vectores soporte y otras herramientas de Aprendizaje Automático.

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

Listing A.10: Importación de herramientas de aprendizaje automático

Como se ha explicado en secciones anteriores, se utilizará PyTorch como librería de aprendizaje Profundo. Para utilizarla es necesario importar algunos módulos del paquete torch. Ya que el problema requiere de procesamiento tensorial de imágenes, se importarán también módulos del paquete torchvision.

```
import torch
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset, ConcatDataset
```

Listing A.11: Importación de Pytorch y algunos módulos

```
import torchvision
from torchvision import datasets, models, transforms
```

Listing A.12: Importación de torchvision y algunos módulos

Para que Pennylane funcione correctamente, es necesario utilizar su propia versión de numpy, que será la encargada de realizar operaciones vectoriales.

```
#Se utilizara pennylane como libreria de computacion cuantica
import pennylane as qml
#Se importan  la version de numpy de Pennylane
from pennylane import numpy as np
```

Listing A.13: Importación de Pennylane y su versión de numpy

A.3. GPU

El entorno de ejecución Google Colab permite el uso de una GPU para acelerar el procesamiento de modelos. Podemos visualizar las características de la GPU y comprobar si tenemos una asignada para la ejecución en cuestión.

```
!nvidia-smi
```

Listing A.14: Imagen de la GPU

```
   Fri Oct  7 10:47:16 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   34C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
#Se comprueba si hay una CPU disponible
device="cuda" if torch.cuda.is_available() else "cpu"
torch.manual_seed(333)
if device == 'cuda':
    torch.cuda.manual_seed_all(333)
device
```

Listing A.15: Comprobación de la GPU

```
'cuda'
```

Tras haber cargado los archivos, se usarán estructuras de datos para poder utilizar las imágenes.

A.4. Muestra

Se obtiene la etiqueta a partir del nombre del fichero.

```
#Valor de la etiqueta
train_list[0].split('/')[-1].split('.')[0]
```

Listing A.16: Valor de la etiqueta

```
'cat'
```

```
# Se establecen las direcciones de los datos
train_directory = '/content/train'
test_directory = '/content/test1'

train_list = glob.glob(os.path.join(train_directory,'*.jpg'))
test_list = glob.glob(os.path.join(test_directory, '*.jpg'))
```

Listing A.17: Direcciones de los conjuntos de datos

Se muestran algunas imágenes aleatorias del conjunto que se utilizará en entrenamiento del modelo.

```
random_idx = np.random.randint(1,25000,size=24)

fig = plt.figure(figsize=(20,10))
i=1
for idx in random_idx:
    ax = fig.add_subplot(3,8,i)
    img = Image.open(train_list[idx])

    plt.axis('off')
    plt.imshow(img)
    i+=1

plt.axis('off')
plt.show()
```

Listing A.18: Muestra inicial

A.5. Red Convolucional

Se fijan los parámetros de entrenamiento.

```
#Se establecen los parametros para el modelo
learning_rate =0.0001
batch_size=100
epochs = 5
```

Listing A.19: Parámetros para el modelo

A.5.1. Transformaciones de datos

Se realizan transformaciones a las imágenes para que todas tengan el mismo tamaño.

```
#Operaciones de transformacion de datos sobre los tres conjuntos de datos
train_transforms =transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor()


])

val_transforms  =transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor()


])

test_transforms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor()


])
```

Listing A.20: Transformaciones de datos

Para utilizar un conjunto de datos en la red a entrenar, PyTorch utiliza la clase *dataset* del paquete *utils*, la cual es necesario redefinir para obtener las instancias del mismo. Esto permitirá adaptarla al conjunto de datos con el que trabajamos estableciendo las operaciones necesarias para nuestro problema.

A.5.2. Preparación de datos

```
#Se definen algunas operaciones sobre el conjunto de datos
class dataset(torch.utils.data.Dataset):
    def __init__(self,file_list,transform = None):
        self.file_list=file_list
        self.transform=transform

    def __len__(self):
        self.filelength =len(self.file_list)
        return self.filelength

    def __getitem__(self,idx):
        img_path =self.file_list[idx]
        img = Image.open(img_path)
        img_transformed = self.transform(img)

        label = img_path.split('/')[-1].split('.')[0]
        if label == 'dog':
            label=1
        elif label == 'cat':
            label=0

        return img_transformed,label
```

Listing A.21: Clase dataset

El conjunto de datos de Kaggle viene ya dividido en un conjunto de entrenamiento y uno de prueba. No obstante, el conjunto de prueba no se encuentra etiquetado y, por tanto será necesario utilizar parte del conjunto de entrenamiento para la validación del modelo.

```
#Obtenemos un conjunto de validacion
train_list,val_list = train_test_split(train_list , test_size =0.3)
```

Listing A.22: Split de datos

```
#Declaramos los conjuntos de datos y aplicamos las transformaciones
train_data = dataset(train_list,transform=train_transforms)
val_data = dataset(val_list,transform=val_transforms)
test_data = dataset(test_list,transform=test_transforms)
```

Listing A.23: Datasets

Pytorch utiliza otra estructura de datos a partir de la ya descrita clase *dataset* llamada *DataLoader*. Esta clase se encarga de organizar los datos en lotes del tamaño espacificado y desordenar las instancias.

```
#Cargamos los datos en estructuras de carga del tamano especificado
    anteriormente
train_loader = torch.utils.data.DataLoader(dataset = train_data,
    batch_size = batch_size,shuffle = True)
val_loader = torch.utils.data.DataLoader(dataset = val_data, batch_size =
    batch_size,shuffle = True)
test_loader = torch.utils.data.DataLoader(dataset = test_data, batch_size
    = batch_size,shuffle = True)
```

Listing A.24: Loaders

```
#Comprobamos las dimensiones de una imagen
test_data[0][0].shape
```

Listing A.25: Comprobación 1

```
torch.Size([3, 224, 224])
```

```
#Comprobamos los tamanos de los conjuntos y el numero de batches
print(len(train_data),len(train_loader))
print(len(val_data), len(val_loader))
print(len(test_data), len(test_loader))
```

Listing A.26: Comprobación 2

A.5.3. Definición de la red convolucional

Para la definición de la red neuronal se usará la clase *Module* en el paquete de redes neuronales que es la base de los modelos en PyTorch, *nn*.

Se define la parte convolucional de la red:

- La convolución que agrupa partes de cada imagen en varias más pequeñas, con un tamaño de kernel de 3 y un desplazamiento de 2 píxeles.
- Una operación de normalización de lote para evitar el posible sesgo en los parámetros
- Una función de activación ReLU
- El «pooling» para reducir la dimensión

```
#Definimos la red convolucional para extraer las caracteristicas de las
    imagenes
class Cnn(nn.Module):
    def __init__(self):
        super(Cnn,self).__init__()

        #Se establecen tres capas convolucionales con rectificador lineal
    y Max Pooling
        self.layer1 = nn.Sequential(
            nn.Conv2d(3,16,kernel_size=3,padding=0,stride=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(16,32,kernel_size=3,padding=0,stride=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2)

        )

```

```
22
23          self.layer3 = nn.Sequential(
24              nn.Conv2d(32,64,kernel_size=3,padding=0,stride=2),
25              nn.BatchNorm2d(64),
26              nn.ReLU(),
27              nn.MaxPool2d(2)
28
29          )
30
31          #Se anaden tres capas densas con rectificador lineal para obtener
    las caracteristicas
32          self.fc1 = nn.Linear(3*3*64,250)
33          self.fc2 = nn.Linear(250,10)
34          self.fc3 = nn.Linear(10,2)
35          self.relu = nn.ReLU()
36
37      def forward(self,x):
38          out =self.layer1(x)
39          out =self.layer2(out)
40          out =self.layer3(out)
41          out =out.view(out.size(0),-1)
42          out =self.relu(self.fc1(out))
43          out =self.relu(self.fc2(out))
44          out2 = out #Se anade otra salida para las diez caracteristicas que
     seran utilizadas en el modelo cuantico
45          out =self.fc3(out)
46          return out, out2
```

Listing A.27: Definición de la CNN

A.5.4. Entrenamiento

Se prepara el modelo y las funciones a utilizar.

```
1 #Se envia el modelo a la GPU y se establecen los optimizadores y las
     funciones de perdida
2 model = Cnn().to(device)
3 model.train()
4 optimizer = optim.Adam(params = model.parameters(),lr = learning_rate)
5 funcion_perdida = nn.CrossEntropyLoss()
```

Listing A.28: Parámetros de entrenamiento

Se utilizarrá un bucle para optimizar el modelo y se obtendrá la tasa de acierto tanto para el conjunto de entrenamiento como para el de validación.

```
1 #Entrenamiento de la red
2 for epoch in range(epochs):
3     epoch_loss =0
4     epoch_accuracy = 0
5
6     for data,label in train_loader:
7         #GPU
8         data= data.to(device)
```

```
        label = label.to(device)

        output = model(data)[0]
        loss = funcion_perdida(output,label)

        #optimizacion
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        #tasa de acierto
        acc = ((output.argmax(dim=1)==label).float().mean())
        epoch_accuracy += acc/len(train_loader)
        epoch_loss += loss/len(train_loader)

    print('Epoca : {}, tasa de exito en entrenamiento : {}, perdida en
   entrenamiento : {}'.format(epoch+1, epoch_accuracy,epoch_loss))

    #Validacion
    with torch.no_grad():
        epoch_val_accuracy =0
        epoch_val_loss = 0
        for data,label in  val_loader:
            data= data.to(device)
            label = label.to(device)

            val_output = model(data)[0]
            val_loss = funcion_perdida(val_output,label)

            #tasa de acierto
            acc = ((val_output.argmax(dim=1)==label).float().mean())
            epoch_val_accuracy += acc/len(val_loader)
            epoch_val_loss += val_loss/len(val_loader)
        print('Epoca : {}, tasa de exito en validacion : {}, perdida en
   validacion : {}'.format(epoch+1, epoch_val_accuracy,epoch_val_loss))
```

Listing A.29: Entrenamiento de la CNN

```
Epoca : 1,  tasa de exito en entrenamiento : 0.6069712042808533,
            perdida en entrenamiento : 0.6576410531997681
Epoca : 1,  tasa de exito en validacion : 0.6519998908042908,
            perdida en validacion : 0.6270735263824463
Epoca : 2,  tasa de exito en entrenamiento : 0.6621715426445007,
            perdida en entrenamiento : 0.6138336062431335
Epoca : 2,  tasa de exito en validacion : 0.6693333983421326,
            perdida en validacion : 0.6069552302360535
Epoca : 3,  tasa de exito en entrenamiento : 0.6929714679718018,
            perdida en entrenamiento : 0.5840359330177307
Epoca : 3,  tasa de exito en validacion : 0.7106664180755615,
            perdida en validacion : 0.5602877736091614
Epoca : 4,  tasa de exito en entrenamiento : 0.7088574767112732,
            perdida en entrenamiento : 0.5596455335617065
```

```
Epoca : 4,  tasa de exito en validacion : 0.7193329334259033,
            perdida en validacion : 0.5496042966842651
Epoca : 5,  tasa de exito en entrenamiento : 0.7190856337547302,
            perdida en entrenamiento : 0.5496646761894226
Epoca : 5,  tasa de exito en validacion : 0.7170664668083191,
            perdida en validacion : 0.5536209344863892
```

Comprobamos que el modelo entrenado obtiene una tasa de éxito del 71 %.

A.6. QSVM

Obtención de las características extraídas de la red convolucional

```
#Obtenemos los datos resultantes de la red neuronal para el entrenamiento
   del modelo cuantico
result_conv_x_train = np.array([])
result_conv_y_train = np.array([])
for data,label in train_loader:
    result_conv_x_train = np.append(result_conv_x_train, model(data.to(
   device))[1].detach().cpu())
    result_conv_y_train = np.append(result_conv_y_train, label.numpy())
```

Listing A.30: Conjunto de entrenamiento

```
#Obtenemos los datos resultantes de la red neuronal para la comprobacion
   del modelo cuantico
result_conv_x_test = np.array([])
result_conv_y_test = np.array([])
for data,label in  val_loader:
  result_conv_x_test = np.append(result_conv_x_test, model(data.to(device)
   )[1].detach().cpu())
  result_conv_y_test = np.append(result_conv_y_test, label.numpy())
```

Listing A.31: Conjunto de validación

```
result_conv_x_train.shape
```

Listing A.32: Visualización de una instancia

```
tensor([0., 1., 0., ..., 0., 1., 0.], requires_grad=True)
```

```
result_conv_x_train.shape
```

Listing A.33: Visualización de la longitud del conjunto

```
(175000,)
```

Selección y análisis previo de las instancias de entrenamiento

```
#Se seleccionan 100 instancias de entrenamiento y otras 100 de validacion
X_train = result_conv_x_train[0:1000].reshape(-1,10)
X_test = result_conv_x_test[1000:2000].reshape(-1,10)
Y_train = result_conv_y_train[0:100]
Y_test = result_conv_y_test[100:200]
```

Listing A.34: Selección de instancias

```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Listing A.35: Visualización de los subconjuntos obtenidos

```
((100, 10), (100, 10), (100,)), (100,))
```

Se realiza una gráfica simple de la cantidad de instancias de cada clase.

```
plt.hist(Y_train)
```

Listing A.36: Gráfica de clases

```
   (array([53.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 47.]),
tensor([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ], requires_grad=True),
<a list of 10 Patch objects>)
```

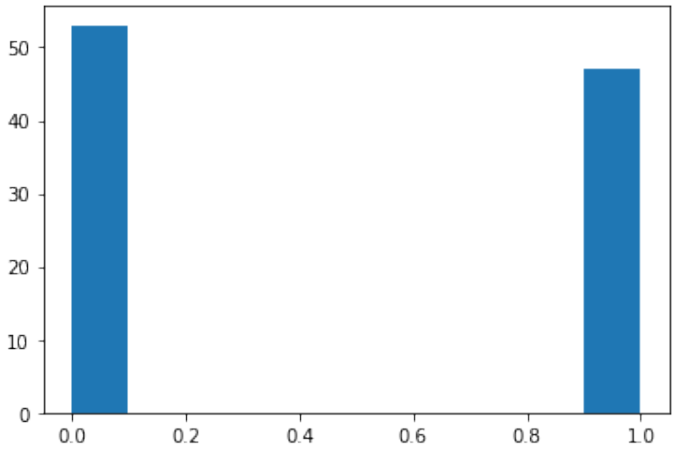


Figura A.1: Gráfica del balance de ambas clases

A.6.1. Definición del circuito cuántico

En este apartado se mostrará como definir el circuito cuántico con Pennylane.

```
def layer(x, params, wires, i0=0, inc=1):
    #Se construye cada una de las capas del circuito cuantico que
   realizara el embedding de los datos
    i = i0
    for j, wire in enumerate(wires):
        qml.Hadamard(wires=[wire])
        qml.RZ(x[i % len(x)], wires=[wire])
        qml.RY(params[0, j], wires=[wire])
        i += inc

    qml.broadcast(unitary=qml.CRZ, pattern="ring", wires=wires, parameters
   =params[1])
```

Listing A.37: Capa

```
def ansatz(x, params, wires):
    #Se construye el circuito
    for j, layer_params in enumerate(params):
        layer(x, layer_params, wires, i0=j * len(wires))
#Se obtiene el conjugado compuesto del ansatz
```

```
adjoint_ansatz = qml.adjoint(ansatz)
```

Listing A.38: Ansatz

Se define una función que genera parámetros aleatorios para inicializar el circuito a partir del número de qúbits y el número de capas del circuito.

```
def random_params(num_wires, num_layers):
    #Genera parametros variacionales aleatorios con la forma de ansatz
    return np.random.uniform(0, 2 * np.pi, (num_layers, 2, num_wires),
    requires_grad=True)
```

Listing A.39: Parámetros aleatorios

Se define el kernel cuántico como la unión de los dos circuitos creados anteriormente y se enmarcan dentro de un nodo cuántico. Las mediciones de este kernel serán exactas.

```
#Se define el nodo cuantico
dev = qml.device("default.qubit", wires=5, shots=None)
wires = dev.wires.tolist()
@qml.qnode(dev)
def kernel_circuit(x1, x2, params):
    ansatz(x1, params, wires=wires)
    adjoint_ansatz(x2, params, wires=wires)
    return qml.probs(wires=wires)
```

Listing A.40: Circuito

Se utiliza la probabilidad de observar todos los qúbits a 0.

```
def kernel(x1, x2, params):
    return kernel_circuit(x1, x2, params)[0]
```

Listing A.41: Kernel

A.6.2. Modelo sin entrenar

```
#Inicializamos con parametros aleatorios
init_params = random_params(num_wires=5, num_layers=3)
```

Listing A.42: Inicialización de parámetros

```
init_kernel = lambda x1, x2: kernel(x1, x2, init_params)

#Mostramos la matriz de pares
K_init = qml.kernels.square_kernel_matrix(X_train, init_kernel,
    assume_normalized_kernel=True)
with np.printoptions(precision=3, suppress=True):
    print(K_init)
```

Listing A.43: Inicialización del kernel

```
[[1.    0.629 0.067 ... 0.024 0.007 0.573]
 [0.629 1.    0.279 ... 0.339 0.099 0.904]
 [0.067 0.279 1.    ... 0.233 0.392 0.23 ]
 ...
 [0.024 0.339 0.233 ... 1.    0.31  0.398]
 [0.007 0.099 0.392 ... 0.31  1.    0.138]
 [0.573 0.904 0.23  ... 0.398 0.138 1.   ]]
```

Para la creación del SVM se utiliza la librería Scikit-learn.

```
#Declaramos un modelo de maquinas de vectores soporte con el kernel
    definido anteriormente
svm = SVC(kernel=lambda X1, X2: qml.kernels.kernel_matrix(X1, X2,
    init_kernel)).fit(X_train, Y_train)
```

Listing A.44: Kernel

Tasa de acierto

Al calcular la tasa de acierto se imprimirán también las clases predichas y las reales.

```
#Tasa de acierto
def accuracy(classifier, X, Y_target):
    print(classifier.predict(X) )
    print(Y_target)
    return 1 - np.count_nonzero(classifier.predict(X) - Y_target) / len(
    Y_target)
```

Listing A.45: Kernel

```
accuracy_init = accuracy(svm, X_test, Y_test)
print(f"La tasa de exito del kernel es: {accuracy_init:.3f}")
```

Listing A.46: Kernel

```
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0.
 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 1. 0.]
[0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1.
 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0.
 0. 1. 0. 1.]
La tasa de exito del kernel es: 0.660
```

A.6.3. Modelo con entrenamiento

```
def target_alignment(
    X,
    Y,
    kernel,
    assume_normalized_kernel=False,
    rescale_class_labels=True,
):
    #KTA del kernel y las etiquetas
    K = qml.kernels.square_kernel_matrix(
        X,
        kernel,
        assume_normalized_kernel=assume_normalized_kernel,
```

```
    )

   #Transformamos las etiquetas y obtenemos el producto interior por pares
    if rescale_class_labels:
        nplus = np.count_nonzero(np.array(Y) == 1)
        nminus = len(Y) - nplus
        _Y = np.array([y / nplus if y == 1 else -1 / nminus for y in Y])
    else:
        _Y = np.array(Y)

    T = np.outer(_Y, _Y)
    inner_product = np.sum(K * T)
    norm = np.sqrt(np.sum(K * K) * np.sum(T * T))
    inner_product = inner_product / norm

    return inner_product
```

Listing A.47: Definición de la función del KTA

```
#Comprobamos el KTA
kta_init = target_alignment(X_train, Y_train, init_kernel,
    assume_normalized_kernel=True)
print(f"El KTA para el modelo con parametros aleatorios y el conjunto de
    datos es: {kta_init:.3f}")
```

Listing A.48: Comprobación del KTA

```
El KTA para el modelo con parametros aleatorios y el conjunto de datos es: 0.067
```

```
params = init_params
opt = qml.GradientDescentOptimizer(0.2)

# Optimizacion del KTA
for i in range(700):

    # Se selecciona un conjunto de puntos para calcular el KTA
    subset = np.random.choice(list(range(len(X_train))), 8)

    # Se define una funcion de coste
    cost = lambda _params: -target_alignment( #Se calcula el opuesto del
    KTA para poder minimizarlo
        X_train[subset],
        Y_train[subset],
        lambda x1, x2: kernel(x1, x2, _params),
        assume_normalized_kernel=True,
    )


    # Optimizacion
    params = opt.step(cost, params)


```

```
    # Imprime el KTA actual
    if (i + 1) % 50 == 0:

        current_alignment = target_alignment(
            X_train,
            Y_train,
            lambda x1, x2: kernel(x1, x2, params),
            assume_normalized_kernel=True,
        )
        print(f"Step {i+1} - Alignment = {current_alignment:.3f}")
```

Listing A.49: Optimización del KTA

```
Step 50 - Alignment = 0.068
Step 100 - Alignment = 0.068
Step 150 - Alignment = 0.068
Step 200 - Alignment = 0.068
Step 250 - Alignment = 0.068
Step 300 - Alignment = 0.068
Step 350 - Alignment = 0.069
Step 400 - Alignment = 0.069
Step 450 - Alignment = 0.069
Step 500 - Alignment = 0.069
Step 550 - Alignment = 0.070
Step 600 - Alignment = 0.070
Step 650 - Alignment = 0.070
Step 700 - Alignment = 0.070
```

Se ha mejorado ligeramente el KTA.

```
# Declaramos el nuevo kernel
trained_kernel = lambda x1, x2: kernel(x1, x2, params)

# Obtenemos la matriz del nuevo kernel
trained_kernel_matrix = lambda X1, X2: qml.kernels.kernel_matrix(X1, X2,
    trained_kernel)

# Declaramos un SVM con la matriz obtenida
svm_trained = SVC(kernel=trained_kernel_matrix).fit(X_train, Y_train)
```

Listing A.50: Modelo nuevo

Tasa de acierto

```
#Tasa de exito
accuracy_trained = accuracy(svm_trained, X_test, Y_test)
print(f"The accuracy of a kernel with trained parameters is {
    accuracy_trained:.3f}")
```

Listing A.51: Tasa de éxito

```
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0.
 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 1. 0.]
[0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1.
 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0.
 0. 1. 0. 1.]
The accuracy of a kernel with trained parameters is 0.670
```

El éxito es del 67 %.

A.7. Resultados

A.7.1. Modelo

```
#Declaramos una funcion para ejecutar el modelo
def QuantumKernelsPlayDogsvsCats(X):
  result_conv_x = np.array([])
  result_conv_x = model(X.resize(1, 3, 224, 224).to(device))[1]
  return svm_trained.predict(result_conv_x.detach().cpu().numpy())[0]
```

Listing A.52: Definición del modelo híbrido

A.7.2. Prueba

```
QuantumKernelsPlayDogsvsCats(test_data[7][0])
```

Listing A.53: Prueba

```
    0.0
```

```
#Comprobamos con imagenes
from numpy.random.mtrand import randint
import random

id_list = []
class_ = {0: 'cat', 1: 'dog'}

fig, axes = plt.subplots(8, 6, figsize=(15, 20), facecolor='w')

for ax in axes.ravel():

    i = randint(0, 12500, 1)[0]
    label = QuantumKernelsPlayDogsvsCats(test_data[i][0])

    if label > 0.5:
        label = 1
    else:
        label = 0

    img_path = os.path.join(test_directory, '{}.jpg'.format(i))
    img = Image.open(img_path)

    ax.set_title(class_[label])
    ax.axis('off')
    ax.imshow(img)
```

Listing A.54: Gráfica

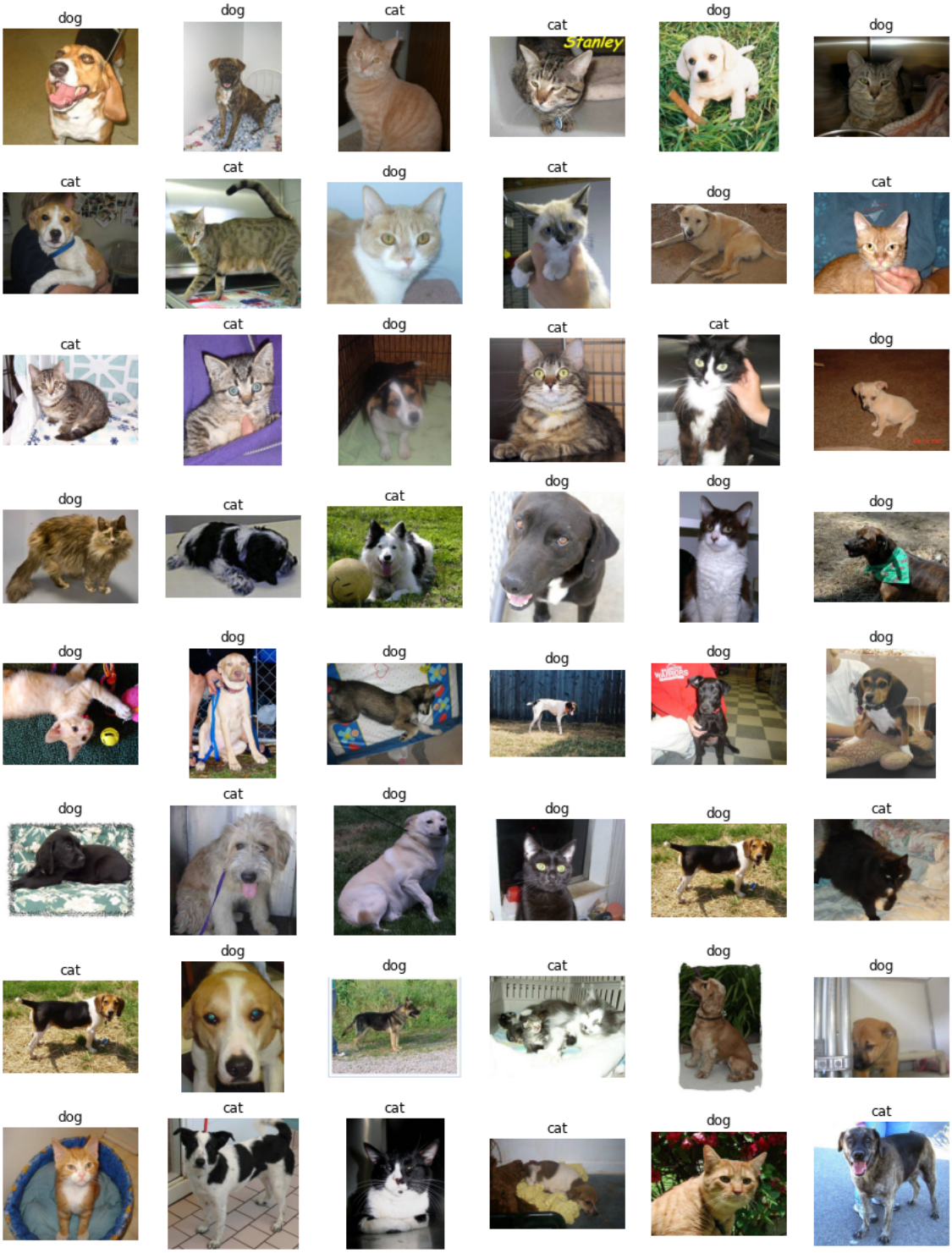


Figura A.2: Resultados de la aplicación del modelo sobre el conjunto test

Apéndice B

Librerías de Machine Learning

B.1. Kaggle

Kaggle en una plataforma dedicada a la Ciencia de Datos con un gran número de conjuntos de datos disponibles. Para poder utilizar dichos datos, Kaggle dispone de una API para obtenerlos mediante código.

B.1.1. API

El comando *kaggle competitions download -c [COMPETITION]* de la API de Kaggle se utiliza para la carga de datos de competiciones de Kaggle.

B.2. SciKit Learn

Esta es la librería de «Machine Learning» más utilizada de Python. Es de código libre y utiliza otras como NumPy, SciPY y matplotlib. A continuación se explican los elementos utilizados en el proyecto.

B.2.1. Selección de modelos

El método *sklearn.model_selection.train_test_split* se utiliza para separar los conjuntos de entrenamiento y test en dos subconjuntos.

B.2.2. SVM

La clase *sklearn.svm.SVC* instancia un clasificador de vetores soporte. Para entrenarlo se puede utilizar el método *fit(X, y)*. Para utilizar el modelo ya entrenado se utilizará el método *predict(X)*

Apéndice C

Manual de Pytorch

C.1. Manejo de la GPU

Pytorch posee herramientas para utilizar una GPU en el procesamiento tensorial de un modelo.

El paquete *torch.cuda* añade soporte para los tensores CUDA, que implementan la misma función que los tensores de la CPU, pero utilizan la GPU para los cálculos.

Para comprobar si existe una GPU disponible es posible utilizar la función *torch.cuda.is_available*. Esta devolverá un booleano positivo en caso de que dispongamos de una. Por otro lado, para establecer una semilla con la que se realizarán los cálculos que impliquen aleatoriedad se utilizará *torch.cuda.manual_seed(seed)* o *torch.cuda.manual_seed_all(seed)*. Donde seed es un entero que será utilizado como semilla.

C.2. Preprocesado de datos

C.2.1. Transformaciones

Las transformaciones y operaciones de data augmentation con PyTorch se realizan con el módulo transforms del paquete torchvision. Algunas de las transformaciones son:

- *transforms.Resize(size)* Redimensiona la imagen a un tamaño dado.
- *transforms.RandomResizedCrop(size)* Corta aleatoriamente una porción de la imagen y la redimensiona al tamaño elegido
- *transforms.RandomHorizontalFlip()* Voltea la imagen horizontalmente con la probabilidad indicada (por defecto 0.5)
- *transforms.ToTensor()* Convierte la imagen a tensor

C.2.2. Estructuras de datos

En este apartado se especifican las herramientas de carga de datos de PyTorch que se encuentran en el paquete *torch.utils.data*.

Tensor

Un tensor es una matriz multidimensional que contiene elementos de un solo tipo de datos. Los tensores en PyTorch se definen mediante la clase *torch.Tensor*. Esta clase posee la función *Tensor.to* que permite guardar la información del mismo en un dispositivo específico como una GPU. Para calcular el gradiente de un tensor se utilizará el método*torch.Tensor.backward()*.

DataLoader

La clase *torch.utils.data.DataLoader(dataset)* implementa varias funcionalidades sobre un conjunto de datos. Algunas de estas son:

- Manejo de conjuntos iterables y «mapeables».
- Orden de los datos
- Procesamiento en lotes
- Mejora en el procesamiento de datos con GPUs y CPUs

Dataset

El argumento principal de la clase *torch.utils.data.DataLoader* es dataset. Esta estructura contiene el conjunto de datos con el que trabajar. Estos pueden ser iterables o «mapeables». Esta clase es abstracta y por tanto se debe implementar para utilizarla. Al hacerlo es necesario sobreescribir el método *__getitem__()*.

C.3. Herramientas para el modelo

La clase base para los modelos de PyTorch es *torch.nn.Module*. Las clases de los modelos que se definan deben heredar de esta clase.

C.3.1. Capas del modelo y funciones de activación

Sequential

La clase *nn.Sequential* es un contenedor secuencial. A esta clase se agregarán módulos de PyTorch en el orden en que se pasen en el constructor. El método *forward()* de *Sequential* acepta cualquier entrada y la reenvía al primer módulo que contiene. A continuación, encadena las salidas a las entradas secuencialmente para cada módulo posterior, devolviendo finalmente la salida del último.

El valor que aporta una instancia de *Sequential* sobre la llamada a una secuencia de módulos es que permite tratar todo el contenedor como un único módulo, de forma que la realización de una transformación sobre el *Sequential* se aplica a cada uno de los módulos que almacena.

Capas de red neuronal

- *torch.nn.Conv2d(in_channels, out_channels, kernel_size)* Aplica una convolución 2D sobre una señal de entrada compuesta por varios planos de entrada.
- *torch.nn.BatchNorm2d(num_features* Normaliza por lotes una entrada 4D (un lote de entradas 2D con una dimensión adicional)
- *torch.nn.MaxPool2d(kernel_size)* Aplica un max pooling 2D sobre una señal de entrada compuesta por varios planos de entrada.

Funciones de activación

Las funciones de activación también forman parte de este módulo. Por ejemplo la función ReLU, en PyTorch *torch.nn.ReLU* aplica la función de rectificación lineal elemento a elemento.

C.3.2. Entrenamiento y función de pérdida

Funciones de pérdida

La función de pérdida *torch.nn.CrossEntropyLoss()* calcula la entropía cruzada entre una entrada y un objetivo.

Optimizadores

El paquete *torch.optim* dispone de varios algoritmos de optimización. Dentro de este paquete se encuentra el método *torch.optim.Adam(params)* que implementa el algoritmo Adam. El método *zero_grad* establece el gradiente a 0 y el método *torch.optim.Optimizer.step(closure)* se utiliza para realizar un paso del algoritmo.

Apéndice D

Manual de Pennylane

D.1. Nodos y aparatos

Para realizar operaciones cuánticas con Pennylane es necesario definir un nodo cuántico. En PennyLane, un «device» o aparato puede ser un dispositivo de hardware o un simulador de software. Cada aparato se define mediante su tipo y número de cables mediante la clase *device(name, wires=1)*. Hay distintos tipos de aparatos:

- Qúbit
- Gausiano
- Qúbit de TensorFlow
- Qúbit con diferenciación automática.

Es posible especificar en algunos aparatos el número de ejecuciones del mismo que serán utilizadas para la medición de los valores del circuito mediante el argumento *shots*. También es posible especificar este valor cada vez que se realiza una llamada al circuito.

Una vez inicializado el aparato, es posible definir un nodo cuántico. Los nodos cuánticos son una encapsulación abstracta de una función cuántica, descrita por un circuito cuántico. Los nodos cuánticos están vinculados a un solo aparato, que se utiliza para evaluar los valores de la esperanza y varianza de este circuito.

Para definir nodos cuánticos es necesario definir una función cuántica o circuito. Para considerar a una función como cuántica se deben cumplir las siguientes condiciones:

- Las funciones cuánticas deben contener una operación cuántica por línea, en el orden en que van a aplicarse. Se debe especificar sobre qué qúbits se realizan las operaciones.
- Las funciones cuánticas deben devolver observables medibles.

Para especificar un nodo se puede utilizar el decorador de python: @*qml.qnode*(*device*) .

D.2. Operaciones

PennyLane admite una gran variedad de operadores cuánticos, como puertas, canales ruidosos, preparadores de estado y mediciones. Estos operadores pueden utilizarse en funciones cuánticas. Las funciones aplicadas a los operadores extraen información o transforman los operadores.

D.2.1. Funciones de operador

Existen varias funciones para manipular operadores.

Para crear el operador adjunto de una función se utiliza la función *qml.adjoint(fn)*. La operación adjunta y la inversa de una operación unitaria son equivalentes. Las transformaciones adjuntas también pueden utilizarse para aplicar el adjunto de cualquier función cuántica. En este caso, la función *adjoint* acepta una única función.

D.2.2. Operadores de Qúbit

Rotaciones

Estas pueden ser o no parametrizadas. Lo serán cuando sea necesario especificar el ángulo de rotación. Existe una función de rotación para cada eje:

- *RX(phi)*
- *RY(phi)*
- *RZ(phi)*

Operaciones no parametrizadas

Estas operaciones carecen de parámetros. Un ejemplo es la función *Hadamard()* que aplica dicha transformación en el estado de un qúbit.

D.2.3. Medidas

Pennylane puede extraer diferentes tipos de resultados de las mediciones de los aparatos cuánticos: la esperanza de un observable, su varianza, una medición o las probabilidades de los distintos estados cuánticos base.

Para obtener el valor esperado de un sistema observable se utiliza la función *expval(op)* donde *op* es un observable.

Para obtener la probabilidad de cada estado cuántico base se utiliza *probs(wires=None, op=None)*. Esta función admite como parámetros tanto cables o un observable. Al utilizar cables, el nodo cuántico devolverá un array que contiene las probabilidades de medir cada estado base dado el estado actual del sistema. También se puede especificar un subconjunto de cables para obtener su distribución marginal de probabilidad.

D.2.4. Broadcast

Para aplicar varias operaciones unitarias a un conjunto de qúbits es posible utilizar la función *broadcast(unitary, wires, pattern)*. El argumento *unitary* es una operación cuántica que puede ser o no diseñada por el usuario. Por otro lado, *wires* especifica a qué «hilos» o qúbits se le aplicará el patrón. Por último, se especificará qué patrón se utilizará. Algunos de los patrones son:

- **“single”**: A cada qúbit.
- **“double” y “double_odd”**: A cada par de qúbits empezando por los impares en el segundo caso.
- **“chain” y “ring”**: A cada par de qúbits «vecinos» considerando el primero y último como tales en el segundo caso.
- También es posible crear un patrón personalizado

Los parámetros para las operaciones utilizadas pueden ser especificados.

D.3. Kernels

Para el uso de kernels cuánticos Pennylane dispone del paquete *qml.kernels*. Este paquete incluye funciones para entrenar a un kernel cuántico con conjuntos de datos de entrenamiento y de prueba para obtener la matriz asociada del kernel. Por otro lado, proporciona métodos de posprocesamiento para dichas matrices del kernel que pueden utilizarse para mitigar el ruido del aparato y los errores en el muestreo.

Para calcular la matriz del kernel por pares entre dos conjuntos de datos se puede utilizar la función *kernel_matrix(X1, X2, kernel)*. Con este método podemos evaluar la función kernel en cada par de puntos de datos, donde los puntos provienen de diferentes conjuntos de datos, como un conjunto de datos de entrenamiento y uno de prueba. Para calcular la raiz cuadrada de dicha matriz podemos utilizar la función: *square_kernel_matrix(X1, X2, kernel)*.

Para calcular el alineamiento kernel-objetivo se puede utilizar la función: *target_alignment(X, Y, kernel, assume_normalized_kernel=False, rescale_class_labels=True)*. Si el conjunto de datos está desequilibrado, es decir, si el número de instancias en las dos clases es diferente, el argumento *rescale_class_labels=True* aplicará un reescalado tal que

$$\tilde{y_i} = \frac{y_i}{n_{y_i}}$$

.

Para entrenar un kernel cuántico se puede utilizar el descenso del gradiente. Pennylane implementa esta técnica con la función *GradientDescentOptimizer(stepsize=0.01)*, donde el parámetro *stepsize* marca el tamaño del salto en cada paso de la opotimización. Para cada iteración, la clase *GradientDescentOptimizer* contiene el método *step*