Universidad Internacional de la Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster en Computación Cuántica

Análisis de sentimiento en procesamiento cuántico del lenguaje natural en un contexto financiero

Trabajo Fin de Estudios

presentado por: Carlos Samper Quinto

Dirigido por: Rodrigo Gil-Merino y Rubio

Ciudad: Murcia

Fecha: Septiembre de 2023

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Resumen

La inteligencia artificial es una combinación de algoritmos planteados con el propósito de crear máquinas que presenten las mismas capacidades que el ser humano. Así, el aprendizaje automático es un subconjunto de la inteligencia artificial que permite que el sistema aprenda y mejore sin estar programado explícitamente para ello.

Dentro de este contexto se desarrolla este trabajo, realizando un análisis basado en el procesamiento del lenguaje natural y el procesamiento cuántico del lenguaje natural para un contexto financiero.

El primero es una rama de conocimiento dentro del aprendizaje automático centrada en la interpretación del lenguaje humano a través de su codificación en lenguaje máquina, plasmando la lingüística, semántica y contextualización de los textos en distintos parámetros que permiten al sistema aprender de ellos. El segundo es una extrapolación de lo expuesto anteriormente al paradigma de la computación cuántica, es decir, el procesamiento del lenguaje humano se realiza a través de una codificación para ser interpretada por ordenadores cuánticos, nutriéndose así de las distintas ventajas que éstos pudieran aportar a la ciencia descrita.

En este trabajo, además de plantear un análisis discursivo sobre las distintas aproximaciones más populares para el procesamiento cuántico del lenguaje natural, se elabora una codificación que trata de mejorar los resultados de dichas aproximaciones. Adicionalmente, se utiliza un modelo de procesamiento de lenguaje natural clásico para establecer comparaciones entre la eficiencia del paradigma cuántico sobre el paradigma clásico.

Finalmente, se recopila todo lo expuesto anteriormente para establecer conclusiones sobre la utilidad y viabilidad de la codificación en la aproximación cuántica sobre la aproximación clásica.

Palabras Clave: inteligencia artificial, aprendizaje automático, procesamiento del lenguaje natural, procesamiento cuántico del lenguaje natural, computación cuántica.

Abstract

Artificial intelligence is a combination of algorithms designed with the purpose of creating machines that have the same capabilities as humans. Thus, machine learning is a subset of artificial intelligence that allows the system to learn and improve without being explicitly programmed to do so.

Within this context, this work is developed, carrying out an analysis based on natural language processing and quantum natural language processing for a financial context.

The first is a branch of knowledge within machine learning focused on the interpretation of human language through its coding in machine language, capturing the linguistics, semantics and contextualization of texts in different parameters that allow the system to learn from them. The second is an extrapolation of what was previously stated to the paradigm of quantum computing, that is, the processing of human language is carried out through coding to be interpreted by quantum computers, thus drawing on the different advantages that they could bring to the science described.

In this work, in addition to proposing a discursive analysis of the different most popular approaches for quantum processing of natural language, a coding is developed that tries to improve the results of these approaches. Additionally, a classical natural language processing model is used to establish comparisons regarding the efficiency of the quantum paradigm over the classical paradigm.

Finally, everything discussed above is compiled to establish conclusions about the usefulness and viability of coding in the quantum approach over the classical approach.

Keywords: artificial intelligence, machine learning, natural language processing, quantum natural language processing, quantum computing.

1. Introducción

En la era digital en la que vivimos, la comprensión y manipulación del lenguaje humano se han convertido en una de las áreas de investigación más cruciales en el ámbito de la inteligencia artificial (Hovy, 2021), entendiendo por inteligencia artificial la combinación de algoritmos planteados con el propósito de crear máquinas que presenten las mismas capacidades que el ser humano (IBM, 2023a).

Como ciencia, el procesamiento del lenguaje natural, entendido como la tecnología que permite a las computadoras entender y trabajar con el lenguaje humano, como el texto y el habla (Ganguly, 2022), aborda problemas de distinta índole que presenta la propia comunicación, tales como semejanza de oraciones, la traducción automática, la extracción de información o el análisis de sentimiento. Para este trabajo, se toma como caso a estudiar, el problema de análisis de sentimiento en un contexto financiero, como excusa para tratar con el procesamiento cuántico del lenguaje natural.

El procesamiento cuántico del lenguaje natural emerge como un campo vanguardista que combina las prometedoras propiedades del paradigma de procesamiento de información basado en la mecánica cuántica, la computación cuántica, como son, la manipulación de múltiples estados simultáneamente o el entrelazamiento cuántico para el desarrollo de algoritmos, con la riqueza y complejidad del lenguaje humano (Du, 2022). El procesamiento cuántico del lenguaje natural, al igual que el procesamiento del lenguaje natural, se fundamenta en técnicas de inteligencia artificial que permiten a las computadoras aprender y mejorar su rendimiento sin necesidad de programación explícita, es decir, aprendizaje automático, que, a través de distintas arquitecturas estructurales, proporciona mejoras sustanciales sobre su análogo clásico (Coecke, 2020).

El análisis de sentimiento es un problema dentro del procesamiento del lenguaje que consiste en la clasificación de sentencias y textos en base a su intencionalidad y emoción a transmitir (Medhat, 2014). Algunos de los problemas que esta ciencia presenta para el paradigma clásico pueden ser la sencillez con la que se codifican los textos, perdiendo el contexto implícito en muchos casos, o la cantidad de parámetros a acumular para la implementación de los algoritmos, que crece de manera exponencial con la extensión de

los textos. Sin embargo, en el procesamiento cuántico del lenguaje natural se es capaz de reproducir espacios dimensionales exponencialmente más grandes que en el paradigma clásico, lo que solventa algunos de los problemas planteados (Liu, 2023).

No obstante, los dispositivos cuánticos sobre los que desarrollar esta ciencia se encuentran en la era cuántica de escala intermedia ruidosa, que es una manera de expresar que, a pesar de que se tiene potencia suficiente sobre las computadoras cuánticas para el desarrollo de algoritmos, aún no son lo suficiente estables las señales ni los errores en las mismas, por lo que presentan limitaciones, tanto en hardware como en software, que ponen en tela de juicio la eficiencia de estas aproximaciones frente a sus homónimos clásicos (Perskill, 2018).

Sin embargo, la tecnología permite desarrollar numerosas posibles aproximaciones al problema de análisis de sentimiento en el paradigma cuántico (Liu, 2019). Al tratarse de un problema de aprendizaje automático, la estructura de las aproximaciones es semejante, variando la arquitectura de algunos elementos o bloques durante el proceso. Los bloques a destacar que varían entre aproximaciones son el bloque de codificación, encargado de transformar datos a lenguaje máquina, y el bloque referente al modelo, que incluye parámetros ajustables en el proceso de aprendizaje.

Existen varias posibilidades en la arquitectura de cada bloque que, combinadas, establecen distintas aproximaciones para el mismo problema, de análisis de sentimiento y procesamiento cuántico del lenguaje natural en el caso que atañe.

Los modelos matemáticos inspirados en neuronas que tratan de emular el funcionamiento cognitivo y que son utilizados en aprendizaje automático o redes neuronales (Rosenblatt, 1960) y, particularmente, las redes neuronales cuánticas, es decir, modelos homónimos a su análogo clásico pero fundamentados en la computación cuántica, resultan una aproximación idónea como modelo para el problema de análisis de sentimiento, ya que, al utilizar una cantidad ingente de parámetros para su implementación, se encuentra una analogía directa con lo deseado para el problema descrito (Duncan, 2015).

Adicionalmente, la librería desarrollada para Python 3.9 por Cambridge Quantum, Lambeq (Kartsaklis, 2021), utiliza, como bloque de codificación, distintas herramientas

que ayudan a transcribir el texto en diagramas, capaces de captar el contexto y extrapolarlo a circuitos cuánticos ejecutables por los dispositivos. Y como modelo, emplea también redes neuronales cuánticas.

A pesar de resultar muy conveniente el uso de redes neuronales cuánticas para el desarrollo de análisis de sentimiento por las características descritas, se pueden construir algoritmos alternativos cuya lógica subyacente sea igual de válida para la implementación del problema.

En este proyecto, se establecen comparaciones entre aproximación relevantes para el desarrollo del problema de análisis de sentimiento en computación cuántica, así como la construcción de una nueva aproximación que, alejándose de las arquitecturas ordinarias, se trata de abrir una nueva vía como aproximación para el procesamiento cuántico del lenguaje natural.

2. Contexto y estado de la técnica

2.1. Introducción a la computación cuántica

En 1980 Paul Benioff, plantea la existencia de un computador adiabático, es decir, un dispositivo que intercambiara energía de manera interna sin interactuar con el exterior, rigiéndose por las leyes de la mecánica cuántica (Benioff, 1980). Interesado por las teorías de Paul, Richard Feynman en 1981, refuerza, modifica y amplia las ideas expuestas por Paul en una conferencia que daría impulso a esta novedosa ciencia, la computación cuántica (Feynman, 1982). Actualmente, se entiende la computación cuántica como un paradigma de procesamiento de información que se nutre de los principios de la mecánica cuántica como son el entrelazamiento y la superposición para generar un nuevo paradigma computacional, eventualmente más eficiente que su análogo clásico para determinados problemas.

Desde entonces, distintos físicos y matemáticos como Peter Shor o Lov Grover, han desarrollado algoritmos de gran relevancia en la industria que han demostrado que la computación cuántica es un campo disruptivo capaz de revolucionar numerosos ámbitos de la ciencia (Shor, 2002).

En computación cuántica, se establece como unidad fundamental el qubit que, por los criterios de Di Vincenzo (Di Vincenzo, 2005), debe tener dos estados bien diferenciados. Se establecen como $|0\rangle$ y $|1\rangle$, en notación de Dirac (Dirac, 1925). Estos estados son vectores ortogonales sobre un espacio de Hilbert que además, también por los criterios de Di Vincenzo, deben ser unitarios, es decir, de módulo uno.
Un espacio de Hilbert es una generalización del espacio euclídeo de dos y tres dimensiones reales habituales, a un espacio de dimensión arbitraria, incluyendo los espacios de dimensión infinita complejos (Behtouei, 2023). Así la representación vectorial de los elementos de la base de un qubit vienen dadas por la Ecuación 2.1.

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

Una de las características más relevantes de los qubits, es que se pueden encontrar en estado de superposición, es decir, pueden estar simultáneamente en ambos elementos de

la base (Behtouei, 2023). Más explícitamente, en la Ecuación 2.2.

$$|+\rangle = \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{2.2}$$

Se sigue de lo anterior que, al estar compuesta la base del espacio por dos elementos ortogonales y no estar limitados sus componentes al cuerpo de los reales, la dimensión del espacio que genera un qubit es $\mathbb{C}^2$.

La esfera de Bloch (Heusler, 2020), es una representación geométrica equivalente a un isomorfismo con el espacio de Hilbert generado por un qubit, para una visualización, comprensión y manipulación de los estados de manera sencilla. Cada punto en la esfera corresponde a un estado cuántico posible del qubit, y su superficie representa estados puros, entendiendo por estado puro.

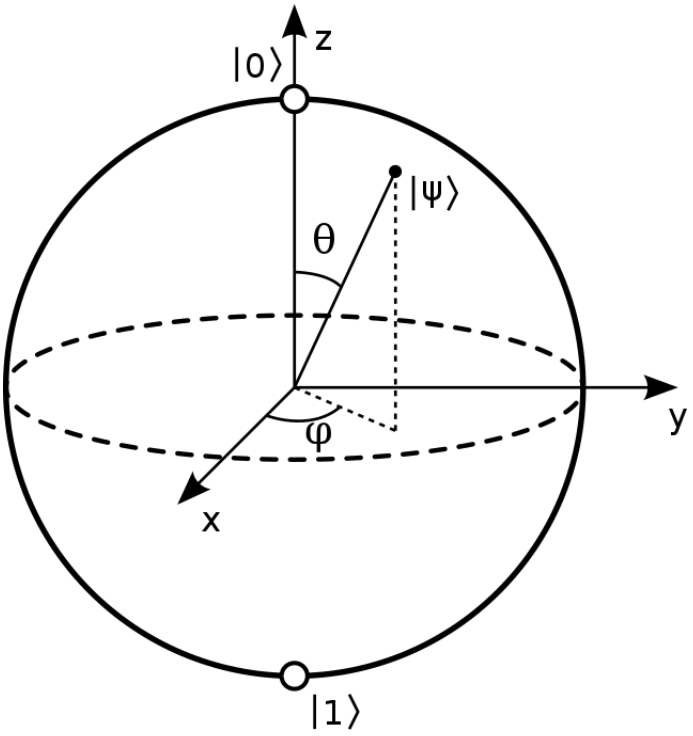


Figura 2.1: Esfera de Bloch para un qubit.

Fuente: adaptada de Heusler (2020).

Los estados cuánticos representan un sistema físico, una descripción matemática de las propiedades físicas de una partícula o sistema cuántico, que incluye información sobre su

posición, momento, y otras propiedades, y se caracteriza por amplitudes de probabilidad complejas. Estos estados describen la naturaleza probabilística y la superposición de posibles resultados en la mecánica cuántica (Nielsen, 2012).

Un estado genérico, $|\psi\rangle$ sobre la esfera de Bloch se define por dos ángulos (θ y ϕ) característicos, y los elementos de la base a través de la Ecuación 2.3

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.3}$$

De la expresión anterior, se deduce que, a través de rotaciones sobre la superficie de la esfera se puede cambiar de un estado cuántico a otro (Heusler, 2020). La matriz genérica de rotación sobre la esfera de Bloch es la Ecuación 2.4.

$$\hat{R}(\theta,\phi) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2}e^{-i\phi} \\ \sin\frac{\theta}{2}e^{i\phi} & \cos\frac{\theta}{2} \end{pmatrix} \tag{2.4}$$

Notar que la matriz expuesta en la Ecuación 2.4, presenta carácter unitario, es decir, una matriz genérica U, se considera unitaria si se cumple la Ecuación 2.5.

$$U^{\dagger}U = I \tag{2.5}$$

Donde en la Ecuación 2.5, $U^{\dagger}$ representa la matriz adjunta (conjugada traspuesta) de U (Nielsen, 2012).

Así, las puertas lógicas cuánticas de un qubit, se definen como matrices unitarias para ángulos particulares de la matriz genérica de rotación sobre la esfera de Bloch (Heusler, 2020). La puerta de un qubit más importante en computación cuántica, es la puerta Hadamard, que es aquella que transforma estados puros en estados en superposición, como se aprecia en la Ecuación 2.6. En circuito cuánticos, entendidos como sistemas compuestos por qubits y puertas cuánticas para el desarrollo de algoritmos y problemas, se la identifica por la Figura 2.2. Adicionalmente, y de cara a secciones futuras, se muestran las puertas de rotación Rx y Ry en las ecuaciones 2.7 y 2.8, respectivamente. Seguidas de sus representaciones en las figuras 2.3 y 2.4.

$$\hat{H} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.6}$$

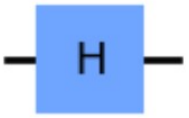


Figura 2.2: Representación de puerta Hadamard

Fuente: *elaboración propia*

$$\hat{Rx} = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{2.7}$$

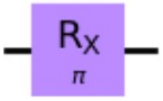


Figura 2.3: Representación de puerta Rx

Fuente: *elaboración propia*

$$\hat{Ry} = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{2.8}$$

Figura 2.4: Representación de puerta Ry

Fuente: *elaboración propia*

Así, de las ecuaciones 2.3 y 2.6 que, la mayoría de los estados posibles para un qubit son estados en superposición. Por cada qubit, se puede reproducir un espacio dimensional asociado tipo $\mathbb{C}^2$, por lo que espacios dimensionales o espacios fásicos de dimensiones $2^N \times 2^N$ se pueden mapear a espacios $N \times N$, como es el caso de espacios generados por moléculas en el campo de la química cuántica, o espacios generados por las posibles combinaciones de elementos en la lingüística.

Estos espacios fásicos generados por múltiples qubits presentan una dimensión dada por el producto tensorial entre los espacios generados por cada qubit, es decir, 2^N, como

se ha descrito. De manera más explícita, el estado cuántico compuesto por dos qubits en estados $|0\rangle$ viene dado por la Ecuación 2.9.

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.9}$$

No obstante, existen estados de múltiples qubits que no son capaces de expresarse como producto tensorial de sus espacios dimensionales, como en la Ecuación 2.10.

$$|B\rangle = \frac{1}{\sqrt{2}}[|00\rangle + |11\rangle] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.10}$$

Estos estados se encuentran en estado entrelazado. Así, se puede definir el entrelazamiento cuántico como un fenómeno en la mecánica cuántica en el que dos o más partículas cuánticas se vuelven interdependientes de tal manera que la información sobre una de ellas está intrínsecamente relacionada con la información de las otras, incluso a distancias significativas, lo que no tiene un equivalente en la física clásica (Behtouei, 2023).

De esta manera, las puertas cuánticas de dos qubits presentan matrices de dimensión $2^2 = 4$, y las de tres qubits $2^3 = 8$. Las puertas de dos o más qubits, a diferencia de las de un solo qubit, pueden presentar discrepancia en el tratamiento sobre los qubits aplicados. Así, se denomina qubit de control a aquellos qubits que los operadores asociados a las puertas de múltiples qubits no afectan, y qubits objetivo a aquellos que se ven afectados por el efecto de la lectura de qubits objetivo, entendiendo por lectura a la aplicación de la puerta sobre ellos.

La puerta mas importante sobre dos qubits es la CNOT, descrita por la matriz 2.1, esta puerta, en combinación con la puerta Hadamard, genera entrelazamiento sobre los qubits aplicados al colocar la puerta hadamard en el qubit de control. En circuitos cuánticos se identifica por la Figura 2.5.

$$C\hat{N}OT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figura 2.5: Representación de puerta CNOT

Fuente: *elaboración propia*

Asimismo, la puerta más relevante sobre tres qubits es la puerta Toffoli, que presenta dos qubits de control y un qubit objetivo. Esta puerta viene descrita por la siguiente matriz.

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

2.2. Aprendizaje automático y aprendizaje automático cuántico

El aprendizaje automático (*machine learning*, en inglés) , nace de la mano de Arthur Samuel, quien en 1952 desarrolló un programa para IBM que poseía la habilidad de aprender para analizar si un movimiento de damas era 'bueno' o 'malo' dependiendo del contexto

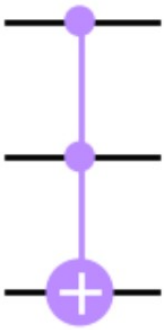


Figura 2.6: Representación de puerta Toffoli

Fuente: *elaboración propia*

actual de la partida (Samuel, 1959). Pronto, este programa llevaría a Arthur Samuel a conocer, en 1956, a J.McCarthy (conocido como el 'padre de la inteligencia artifical') para presentarle su trabajo. Posteriormente, Arthur acuñaría el término *"Machine Learning"* y lo definiría como «El campo de estudio que dota a los computadores de la habilidad de aprender sin estar explícitamente programados». Finalmente, en 1959 publica *'Some studies in machine learning using the game of checkers'* en *IBM Journal* donde se recogen sus estudios al respecto (Zhou, 2021).

De forma más adecuada para el contexto actual, el aprendizaje automático se define como un enfoque computacional que permite a las máquinas aprender patrones y tomar decisiones basadas en datos históricos sin ser programadas explícitamente a través de un proceso de entrenamiento (Schuld, 2018).

De esta manera, se puede definir el aprendizaje automático cuántico como una rama del aprendizaje automático que combina la computación cuántica, con las distintas técnicas y estructuras teóricas del aprendizaje automático (Schuld, 2018). Los procesos de aprendizaje automático cuántico, presentan así un carácter híbrido, introduciendo la computación cuántica por bloques en aquellas etapas del proceso donde se desee implementar las ventajas que pudiera aportar el paradigma.

La estructura general de un circuito de aprendizaje automático cuántico, se puede dividir en:

- mapa de características.

- operador u observable.
- modelo o ansatz.

El mapa de características, es una transformación lineal entre el espacio de partida, donde están codificadas las características del objeto a tratar, eventualmente textos para el problema a considerar, y el espacio fásico generado por los qubits, donde se puede operar con todas las características simultáneamente al ser un espacio fásico de mayor dimensionalidad, tal y como se ha mostrado en el Capítulo 1. Así, el mapa de características indica la forma en la que los datos se codifican. Suele ser la parte más importante del proceso de aprendizaje automático, ya que este bloque del proceso es el que crea el espacio de fases, que en la computación cuántica, se refiere a un espacio matemático que describe todas las posibles combinaciones de estados cuánticos de un sistema, donde cada punto representa un conjunto único de amplitudes de probabilidad para los estados.

La codificación de los datos de partida en el mapa de características se puede realizar a través de distintas características presentes en el qubit. Si se codifican en las amplitudes de probabilidad se trata entonces de una codificación en amplitud, por el contrario si se codifican por las amplitudes complejas, se trata de una codificación en fase (Noori, 2020).

Entonces, sobre el mapa de características se busca una solución para un problema concreto. Así, si el mapa de características no crea un espacio de fases que sea capaz de codificar los datos adecuadamente para el problema que se quiera abordar, se estará partiendo de una situación no óptima para el problema en cuestión. Al proceso de elección y codificación de datos de entrada en un mapa de características se le denomina usualmente *embedding*, aunque este es un término aún más general, se puede utilizar en este contexto (Alonso-Linaje, 2022).

En la Figura 2.7 se muestra un mapa de características que transforma cada muestra de datos de las dos clases no linealmente separables en una muestra de datos con nuevas características en un espacio de dimensiones superiores donde las dos clases se vuelven linealmente separables.

La manera de identificar cuándo un mapa de características es adecuado para unos

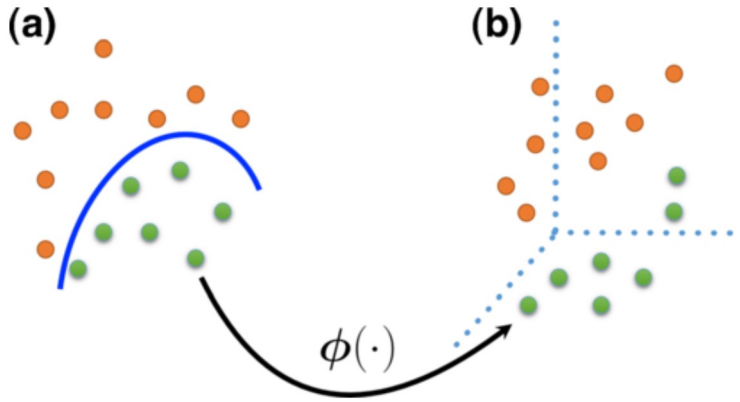


Figura 2.7: Conversión de espacios por mapa de características.
Fuente: adaptada de Noori (2020).

datos dependerá del tipo de problema a abordar. En el caso de un clasificador cuántico, el mapa de características se asocia con la transformación lineal $F(x_i)$, que queda plasmada como un operador unitario dentro del circuito cuántico. Tal operador representa la transformación entre espacios fásicos (Alonso-Linaje, 2022). La transformación introducida por este operador es:

$$\mathbf{C}^n \longrightarrow \mathbf{C}^m$$
$$x_i \longrightarrow F(x_i)$$

entonces, volviendo sobre la notación de Dirac, se tiene:

$$\langle 0|F^+(x_i)F(x_j)|0\rangle \approx 0 \tag{2.11}$$

Para x_i y x_j elementos de misma clase y:

$$\langle 0|F^+(x_i)F(x_j)|0\rangle \approx 1 \tag{2.12}$$

Para x_i y x_j elementos de distinta clase. Adicionalmente se puede calcular el error de la siguiente manera:

$$Error(F) = -\sum_{i,j} y_i y_j \langle 0|F^+(x_i)F(x_j)|0\rangle \tag{2.13}$$

De modo que si los elementos pertenecen a la misma clase entonces el error será mayor cuanto menos solapamiento se produzca entre los elementos.

El observable se refiere a un operador hermitiano, que es un operador lineal en un espacio vectorial complejo cuya matriz adjunta es igual a la matriz original transpuesta conjugada. Más explícitamente y en notación de Dirac, dados dos estados cuánticos $|\psi_a\rangle$ y $|\psi_b\rangle$ y un operador Q, se dice que Q es hermitiano si se cumple (Nielsen, 2012):

$$\langle\psi_a|\,|Q\psi_b\rangle = \langle Q\psi_a|\,|\psi_b\rangle \tag{2.14}$$

Dicho operador del sistema representa una propiedad física que se desea medir o calcular en el sistema cuántico, como la energía de un sistema químico o la expectativa de un valor de una variable de interés.

Una interpretación de la acción del observable, sobre el proceso de aprendizaje automático, sería verlo como la forma con la que se van a buscar separar los datos dentro del espacio de fases creado por el mapa de características.

Así, por ejemplo, en un clasificador lineal clásico, se trataría de una recta, es decir, se estaría buscando separar un conjunto de datos a través de una recta. Se sigue de lo anterior que, dependiendo del problema, puede que no sea posible encontrar una buena solución con el observable escogido, por ello es necesario adaptar la dimensionalidad y forma del observable al problema a abordar, por lo que es conveniente haber codificado adecuadamente los datos en el mapa de características, además de tener cierta intuición sobre el álgebra del problema.

El modelo u ansatz es una elección específica de puertas lógicas cuánticas dotadas de parámetros y conexiones entre qubits, que se utiliza como una aproximación inicial para representar el estado cuántico deseado o resolver un problema específico (Zhang, 2012). Los ansatz son componentes clave en algoritmos de optimización cuántica y, una vez vistos los otros dos bloques en la estructura del aprendizaje automático, se identifica con los grados de libertad, entendido como las dimensiones disponibles proporcionadas por los parámetros de las puertas, para moverse en el espacio dimensional generado por los qubits. De esta manera, un modelo con pocos grados de libertad no será capaz de recorrer de forma adecuada todo el espacio de fases, por ello, es necesario incluir suficientes parámetros a entrenar en el modelo para encontrar de forma óptima la solución al problema.

Respecto a los distintos procesos iterativos para la optimización de parámetros podemos dividirlos en:

- cálculo de error.
- optimización de parámetros.
- realimentación de parámetros.

Para contextualizar en qué punto del proceso de aprendizaje automático se da cada etapa, se recorre toda una iteración y se puntualiza dónde interviene cada uno (Lloyd, 2020).

En primera instancia, a través del mapa de características, se codifican unos datos dados de la manera mencionada anteriormente. Posteriormente, el modelo, que depende de ciertos parámetros que estarán inicializados para esta iteración, se combina con el observable para encontrar una primera solución. Es decir, el modelo establece los grados de libertad por los que el observable se puede mover en el espacio de fases, y el observable, como operador hermítico que es, determina si los puntos del espacio son soluciones al problema.

Esta solución se compara con la etiqueta de los datos iniciales en el caso de encontrarse en un proceso supervisado y, a través de la función de coste, que es una medida que cuantifica el error o la discrepancia entre las predicciones de un modelo y los valores reales en un problema, se obtiene el error entre la solución y la etiqueta mencionada. Notar que la función de coste contiene su propio algoritmo interno para determinar el error. Eventualmente y a modo de ejemplo se puede suponer un error cuadrático medio por simpleza.

Una vez se tiene el error entre el modelo de aprendizaje automático y los datos originales, se mete a un proceso de optimización a través de un optimizador junto con los parámetros usados para inicializar el trabajo. Por optimizador se entiende un algoritmo cuyo objetivo es minimizar o maximizar los resultados de una función de coste para un problema de optimización. De nuevo, este optimizador contiene su propio algoritmo para recalcular, a través de ciertos métodos, unos parámetros más óptimos que los anteriores

para el programa.

Por último, con esos nuevos parámetros optimizados se retroalimenta el sistema para encontrar, de nuevo, una solución más óptima si es que fuera posible.

Para ilustrar todo lo mencionado anteriormente se presenta la Figura 2.8

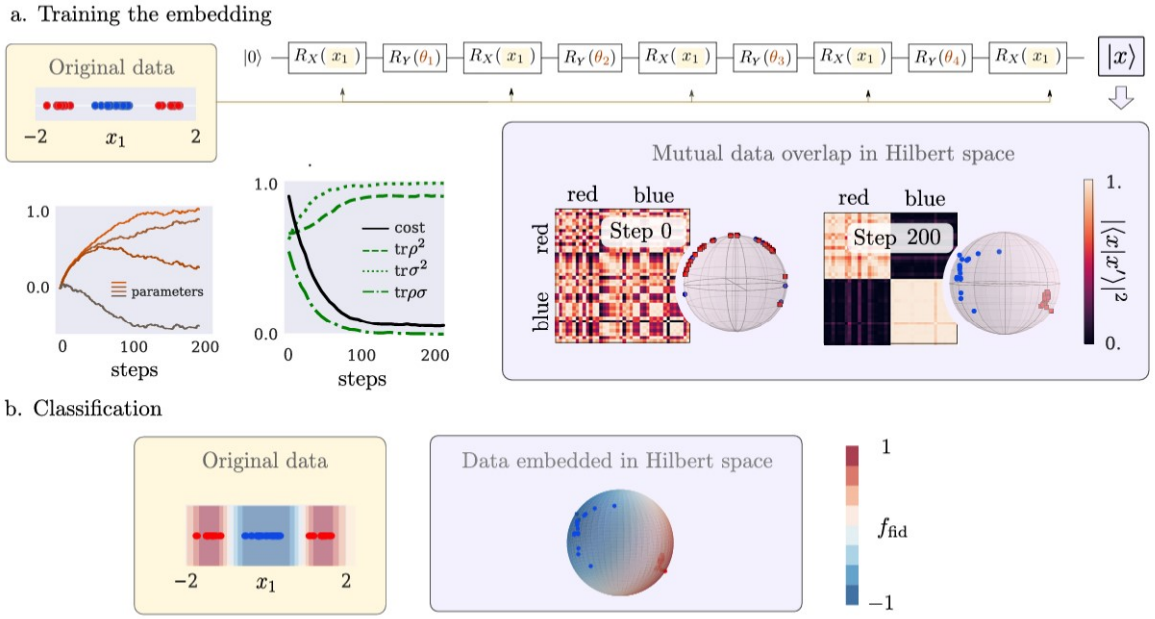


Figura 2.8: Diagrama de un proceso de aprendizaje automático cuántico.
Fuente: adaptada de Lloyd (2020).

La figura, separada en dos etapas muestra el entrenamiento de un *embedding* concreto. De izquierda a derecha y de arriba a abajo, se puede apreciar la distribución de datos iniciales, el mapa de características correspondiente elegido, el espacio de fases generado con subfiguras que plasman los resultados de la Ecuación 2.12, seguido del cálculo del error por la función de coste y la tasa de acierto, este último parámetro es un mero conteo de los datos bien etiquetados, al haber podido reproducirlos de manera fidedigna por el proceso de aprendizaje automático cuántico. En la segunda mitad inferior se aprecian los datos una vez han sido codificados por el mapa de características en el espacio de Hilbert asociado. En resumen, una figura que recoge de forma clara los pasos de un proceso de aprendizaje automático.

Notar que esta es una descripción general de un proceso de aprendizaje automático cuántico y que es posible que, dependiendo de las características de cada problema y proceso a utilizar, aparezcan o se eliminen etapas según las necesidades y objetivos del mismo.

Con todas las partes del proceso de aprendizaje descritas por separado solo queda la descripción de las **épocas**. Se definen como la cantidad de vueltas sobre el **cuadro de datos** para el proceso iterativo. En cada época se toman los valores últimos de la iteración y se introducen en el primer elemento del cuadro de datos para recorrerlo nuevamente, consiguiendo así un continuo sobre el proceso iterativo de retroalimentación.

La idea tras esta metodología es reajustar los parámetros más de una vez para cada oración. Se debe tener en cuenta que el tiempo de ejecución depende directamente de este parámetro. Además, una cantidad de épocas muy elevada puede conducir a un sobreajuste, de los parámetros, es decir, una adaptación demasiado exclusiva de los parámetros a la muestra, perdiendo el carácter generalista del modelo al captar, no solo la tendencia de los puntos en el espacio de fases, sino además el ruido que incorporan, prediciendo perfectamente la muestra y mal el resto de elementos del espacio de fases. El caso contrario, infraajuste, se da al no generar suficientes épocas, por lo tanto, el modelo no es capaz de captar las generalidades presentes en la muestra y en el espacio de fases (Ying, 2019).

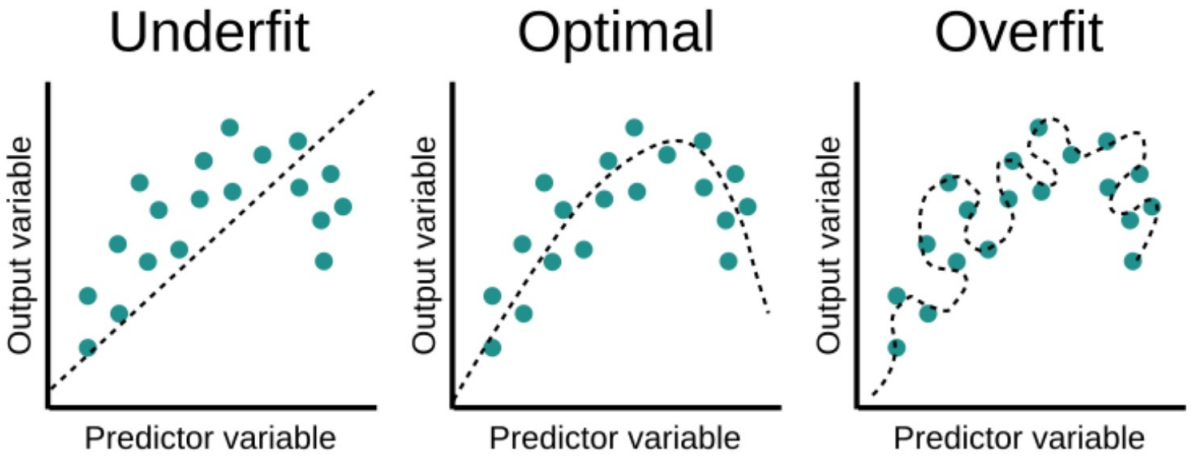


Figura 2.9: De izquierda a derecha: infraajuste de un modelo, ajuste adecuado del mismo y sobreajuste.

Fuente: adaptada de FastAI (2023).

2.3. Procesamiento del lenguaje natural y análisis de sentimiento.

El procesamiento del lenguaje natural (NLP, por sus siglas en inglés) se refiere a una rama de la inteligencia artificial que se centra en la interacción entre las computadoras y el lenguaje humano de manera natural. El objetivo principal del NLP es permitir que las máquinas comprendan, interpreten y generen lenguaje humano de manera similar a como lo hacen los seres humanos. (Ghosh, 2019)

A pesar de no poder atribuir a un único autor el desarrollo y acuñe del NLP por el solapamiento con otras ciencias, algunos autores ubican sus inicios en 1950, de la mano de Alan Turing, con su artículo *'Computing machinery and intelligence'*, donde propone el test de Turing (Bleich, 1995). Ahí, desde cierta perspectiva, se analiza la conexión lenguaje-maquina. Otros lo desplazan a 1954, situando su aparición más explícita en el *Experimento de Georgetown* en colaboración con IBM, dónde se realizaron traducciones automáticas entre el ruso y el inglés (Garvin, 1967). Sin embargo, existen vertientes que sitúan su aparición en 1980, o posteriormente, cuando ya se tenía un desarrollo más profundo no solo de la tecnología involucrada si no de las limitaciones (Ghosh, 2019).

El NLP se ocupa de desarrollar algoritmos, modelos y sistemas que permitan a las computadoras comprender y generar texto o habla de manera efectiva. Esto puede incluir tareas como la traducción automática, el análisis de sentimiento o la generación de texto entre otros.

Concretamente, y como se ha visto en la introducción, el análisis de sentimiento es un problema en el campo del procesamiento del lenguaje natural, que se aborda mediante técnicas de aprendizaje automático. Su objetivo es identificar, clasificar y cuantificar las emociones, opiniones o actitudes expresadas en un texto, ya sean positivas, negativas o neutrales. En esencia, el análisis de sentimiento busca comprender el tono emocional o subjetivo presente en el lenguaje humano, permitiendo extraer información valiosa sobre cómo las personas perciben y reaccionan ante diversos temas, productos, servicios o eventos (Shen, 2018).

El sentimiento captado por un proceso de aprendizaje automático, ya sea clásico o cuántico, se puede manifestar a través de distintas escalas: una puntuación para el texto analizado, una palabra que describa el sentimiento del texto, emoticonos con emociones, etc (Siciliani, 2018). Dejando al criterio de cada autor cuál escoger para el experimento a realizar según las necesidades e intenciones del mismo, además de contextos culturales que pudieran presentarse por el problema.

Las características que comparten todas estas escalas es que deben estar graduadas entre dos extremos, correspondientes al sentimiento más negativo establecido por el autor del análisis y el más positivo. Por lo que en el caso de haber seleccionado una escala numérica, se suele asignar el número menor, a lo que sería en otra escala, el sentimiento más negativo y el mayor al más positivo. Otras modificaciones presentes en escalas puede ser la manera de presentar el sentimiento, según el contexto del problema, así puede ser más adecuado incluir 'feliz' en lugar de 'felicidad' o viceversa.

Adicionalmente, las emociones escogidas por el autor para el problema en cuestión, deben presentar una relatividad en base al grado de discretización establecida por el mismo, es decir, a mayor discretización en el espectro de resultados, mayor cantidad de matices entre emociones se debe incorporar al análisis, tratando de evitar la ambigüedad entre las mismas.

Como ejemplo de lo mencionado, se pueden contemplar dos escenarios para un problema de análisis de sentimiento con temática en finales de cuentos infantiles. Uno en el que únicamente se distingue entre dos emociones y otro en el que se distingue entre tres. Para el primer caso, al tratarse de dos sentimientos como resultados únicamente, se debe buscar dos emociones lo más absolutas posibles y con coherencia para el problema descrito, así, una elección adecuada por la temática del problema podría ser: 'triste' y 'feliz', al estar hablando de finales de cuentos, ya que son dos emociones con significado sin matices. En el segundo escenario, al querer clasificar los textos en tres categorías, una elección de emociones adecuada y en un espectro distinto al escenario anterior podría ser: 'decepcionante', 'indiferente' y 'asombroso'.

2.4. Procesamiento cuántico del lenguaje natural

El objetivo del procesamiento de lenguaje natural cuántico (QNLP, por sus siglas en inglés) es definir los procesos y las asignaciones necesarias para las tareas de procesamiento de lenguaje natural en dispositivos de cómputo cuántico. Dada la etapa inicial de las tecnologías cuánticas, resulta esencial permitir la exploración de métodos válidos y nuevos algoritmos para aprovechar el espacio computacional que estas plataformas ofrecen. El NLP en el lenguaje de la mecánica cuántica ha visto una gran cantidad de trabajo en años recientes pero el hardware y las arquitecturas cuánticas en la era de los dispositivos cuánticos de escala intermedia y ruidosos (NISQ, por sus siglas en inglés) requiere no solo del evidente cambio en la estructura procedural para la adaptación al paradigma cuántico, si no una optimización mayor por las grandes limitaciones que presenta esta vertiente de conocimiento. (Villalpando, 2021)

Las respuestas a quién y cuando se constituye el QNLP no son claras ni precisas. Sería entonces conveniente ubicar su nacimiento con el de las redes neuronales cuánticas o el desarrollo de un marco de trabajo específico para su desarrollo como es *DisCoCat* (Meichanetzidis, 2020), que es el software por antonomasia para el desempeño de tareas QNLP por el bagage, sencillez y éxito que lleva presentando hasta la actualidad (Martinez, 2022). En cuanto a la forma de operar y los algoritmos tras este marco de trabajo serán abordados en secciones posteriores cuando se desarrolle dicho marco de trabajo.

No obstante, el uso del marco de trabajo *DisCoCat*, si bien es el más estandarizado, no es la única vía para la exploración de problemas de esta índole, otras aproximaciones como redes neuronales cuánticas puras pueden ser aproximaciones válidas con buenos resultados (Stein, 2023).

En resumen, el procesamiento del lenguaje natural cuántico, como se irá observando a lo largo de la disertación se encuentra en un punto poco maduro y abierto a muchas interpretaciones y aproximaciones, es parte de los objetivos de este trabajo establecer comparativas entre ellas para obtener resultados claros de cuál puede ser más óptima para el problema de análisis de sentimiento en finanzas.

2.5. Introducción a redes neuronales y redes neuronales cuánticas

El primer paso hacia las redes neuronales artificiales se da en 1943, cuando Warren McCulloch y Walter Pitts, desarrollaron los primeros modelos de redes neuronales. En el artículo titulado *'The Logical Calculation of Immanent Ideas in Nervous Activity '* (McCulloch, 1943), McCulloch explora cómo podrían funcionar las neuronas. Sus redes se basaban en elementos simples que se consideraban dispositivos binarios con umbrales fijos. Los resultados de su modelo consistían en funciones lógicas simples con un carácter de actividad nerviosa 'todo o nada'(Macukow, 2016).

En 1958, Rosenblatt, llevó a cabo un trabajo temprano sobre perceptrones (Rosenblatt, 2021). El Perceptrón es un dispositivo electrónico construido siguiendo principios biológicos que mostró habilidad para aprender a través de las iteraciones sobre el dispositivo.

Desde entonces, la complejidad de las redes neuronales así como de las funciones implicadas en las mismas ha ido evolucionando hasta la actualidad, incluyendo distintos tipos de capas de neuronas que aumentan la cantidad de parámetros a manejar así como las formas en las que dividir un problema(Macukow, 2016). Como se aprecia en la Figura 2.10, la estructura habitual de una neurona artificial está compuesta por unas entradas que reciben unos argumentos de un cuadro de datos $(x_1, ..., x_n)$ y una serie de pesos que actúan como parámetros $(w_1, ..., w_n)$ del ansatz en un proceso de aprendizaje automático. Un algoritmo interno $(\sum)$ que pondera los pesos para obtener un valor de salida (u). Por último, una función de activación típicamente polinomial $(f(\cdot))$, por la incapacidad de los computadores clásicos de reproducir funciones trigonométricas de manera exacta, genera una salida (y) que determina el resultado.

Es importante notar que, en redes neuronales, se mantiene la estructura de algoritmo de aprendizaje automático, por lo que la arquitectura de un algoritmo basado en redes neuronales presenta: un mapa de características, un ansatz, un observable y un optimizador.
El cambio de paradigma respecto al uso de neuronas artificiales clásicas, reside en la manera de construir el modelo y la manera de operar del mismo.

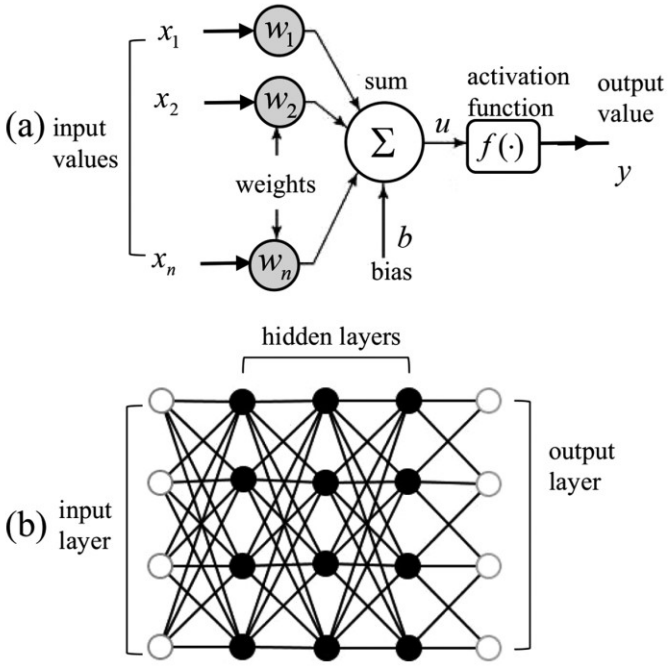


Figura 2.10: Modelo de neurona de McCulloch-Pitts.
Fuente: adaptada de Jia (2018).

Tal y como se ha expuesto las secciones anteriores en este mismo capítulo, algunas de las aproximaciones a abordar se fundamentan en redes neuronales cuánticas (QNN por sus siglas en inglés). Las QNN se nutren de la mecánica cuántica para incorporar variaciones sobre el funcionamiento clásico de las redes neuronales que suponen una ventaja en muchos casos (Mitarai, 2022). Algunas de las mejoras introducidas son en las funciones de activación por ejemplo, que en lugar de tener que estar restringidas a funciones polinomiales por el paradigma clásico, se pueden introducir funciones trigonométricas al ser perfectamente reproducibles por los qubits (Alonso-Linaje, 2022). Esto se debe a que los estados cuánticos asociados a los qubits, pueden presentar amplitudes en forma de funciones trigonométricas al tener un espectro continuo de valores.

La ventaja que esto supone respecto de las funciones de activación clásicas es que las funciones trigonométricas sí que son derivables en todo su dominio, por lo que al buscar los parámetros que optimizan la red neuronal se puede hacer a través de descenso del gradiente de la función de activación (Alonso-Linaje, 2022). El descenso de gradiente es un algoritmo de optimización que busca moverse hacia un mínimo local o global de la función

de una función de costo, en el caso que atañe, la función de activación de la neurona. Al ser derivables dichas funciones en todo su dominio, el algoritmo puede desplazarse por la función de manera continua para encontrar el mínimo que optimiza los parámetros.

A modo de ejemplo, se exponen dos funciones de activación de neuronas artificiales, una para el paradigma clásico y una para el cuántico, para plasmar las mejoras mencionadas.

Para el paradigma clásico, se establece una función lineal dependiente de parámetros dada por la Ecuación 2.15.

Siguiendo la nomenclatura de la Figura 2.10, los elementos $x_1, ..., x_n$ son los datos de entrada para la neurona, y $w_1, ..., w_n$ los pesos de la neurona a determinar.

$$g(\vec{w}, \vec{x}) = \theta_0 + x_1 w_1 + ... + x_n w_n \tag{2.15}$$

El error de forma clásica se establece como el conteo de etiquetas (e_i) mal asignadas por el modelo tras el proceso de optimización, es decir, según la Ecuación 2.16

$$E(g(\vec{w}, \vec{x})) = \frac{1}{m} \sum_{i=1}^{m} \frac{1 - e_i sg(g(\vec{w}, \vec{x}))}{2} \tag{2.16}$$

Sin embargo, en un modelo cuántico se puede definir una función trigonométrica, eventualmente la función arcotangente, siguiendo la nomenclatura se tiene la Ecuación 2.17.

$$g(\vec{w}, \vec{x}) = \arctan \theta_0 + x_1 \theta_1 + ... + x_n \theta_n \tag{2.17}$$

Por lo tanto su error queda plasmado en la Ecuación 2.18.

$$E(g(\vec{w}, \vec{x})) = \sqrt{\sum_{i=1}^{m} |g(\vec{w}, \vec{x}) - e_i|} \tag{2.18}$$

Es decir, se está buscando el mínimo de una función que, ahora sí, es derivable y, a través de la metodología del descenso de gradiente, se pueden encontrar los parámetros que minimizan la función error.

Otro rasgo importante es el uso de puertas de rotación como algoritmo interno ($\sum$ en la Figura 2.10) en lugar de una ponderación de pesos (Jia, 2018). Esta serie de características

generan un nuevo paradigma del que se sirve el experimento para tratar de conseguir una correcta clasificación del lenguaje natural.

3. Objetivos e hipótesis

Los objetivos planteados para el trabajo han sido los siguientes:

- Comprender la estructura y los fundamentos, así como profundizar en las bases del QNLP, focalizando dentro de este campo los problemas de análisis de sentimiento.
- Elaborar un proceso original de análisis de sentimiento en QNLP para un cuadro de datos orientado a las finanzas.
- Abordar el mismo problema que en el punto anterior pero desde otros paradigmas ya presentes en la técnica.
- Realizar comparaciones entre las distintas aproximaciones abordadas, así como para el caso clásico.

La hipótesis planteada para este trabajo y que recoge los objetivos anteriormente mencionados es la siguiente:

> Es posible crear una aproximación no basada en redes neuronales más eficaz para el problema de análisis de sentimiento en procesamiento cuántico del lenguaje natural que las ya establecidas en un entorno clásico y cuántico para un cuadro de datos de índole financiera.

4. Desarrollo del trabajo

Tal y como se ha comentado en la Sección 1, el hilo conductor de este trabajo es un recorrido por los distintos paradigmas abordados para el estudio de un problema de análisis de sentimiento en QNLP orientado a finanzas. Así, los distintos apartados de este Capítulo son las diferentes aproximaciones realizadas a nuestra hipótesis y objetivos.

4.1. Análisis propuesto

La primera aproximación elaborada para el problema de análisis de sentimiento en finanzas es la de autor. Así, como se especificó en el Capítulo 2, se intenta replicar el proceso expuesto para el aprendizaje automático cuántico, pero con variaciones pertinentes para el problema abordado.

Puesto que la linea de trabajo realizada presenta variaciones respecto a lo mencionado, es conveniente establecer un esquema que haga de hilo conductor a lo largo de esta Sección.

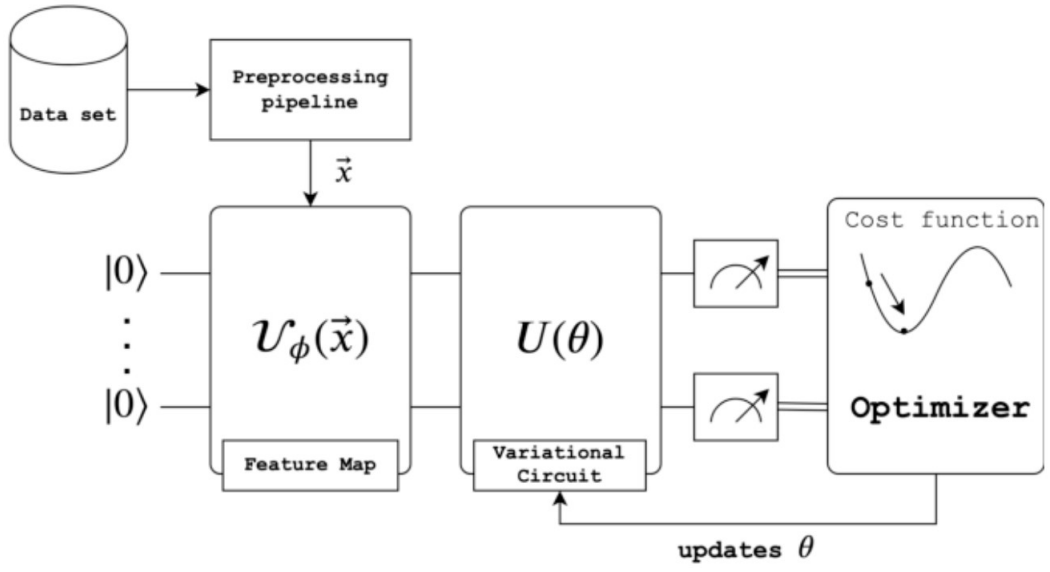


Figura 4.1: Esquema del flujo de trabajo de un clasificador variacional cuántico. Fuente: adaptada de Q-munity (2023).

En la Figura 4.1, se observa como se parte de un cuadro de datos. En el caso de estudio se genera un cuadro de datos que cumple una serie de características, tras un procesado

de dicho cuadro de datos se busca una codificación que se considera adecuada y se procede con la elaboración del mapa de características, así como del circuito variacional o *ansatz*. Tras medir y elaborar un algoritmo para la función de coste se introduce el resultado en un optimizador que devuelve unos parámetros más adecuados para el problema abordado.

Si bien lo descrito anteriormente es el flujo de trabajo que se ha seguido para el caso de estudio, es importante definir antes el entorno de trabajo así como los paquetes más importantes implicados en el proceso con el fin de plasmar la reproducibilidad del mismo.

Los dos entornos de trabajo utilizados para el desarrollo de este caso han sido *IBM Quantum* (IBM, 2023b) y *Visual Code Studio* (Visualstudio, 2023). Ambos se han utilizado con el lenguaje *Python 3.9*.

El primero de ellos, plataforma de acceso gratuito de IBM, tiene implementado *QisKit*, que es un conjunto de herramientas de código abierto desarrollado por IBM para trabajar con computadoras cuánticas y desarrollar aplicaciones cuánticas. *Qiskit* proporciona una serie de bibliotecas, módulos y herramientas que permiten a los desarrolladores y científicos de datos explorar y experimentar con la computación cuántica (QisKit, 2023a). Por lo que es una biblioteca idónea para el desarrollo de la Sección. Para el uso de *QisKit* en marcos de trabajo externos como el segundo mencionado, *Visual Code Studio*, se deben preinstalar en el dispositivo a trabajar dichos paquetes.

Creación del cuadro de datos

Dado que la hipótesis plantea el estudio de un problema de análisis de sentimiento para un cuadro de datos financiero, se indicaba una de las características que éste debía cumplir. Además, puesto que el cuadro de datos construido debe usarse en las distintas aproximaciones para la comparativa de casos de estudio, debe ceñirse también a las limitaciones de éstas, en el caso de que el cuadro de datos no sea adaptable en cada uno de los casos.

Por ello, y como se ve en las Sección 4.3, el cuadro de datos debe presentarse en inglés para el procesado, esto se debe a una identificación de términos a nivel interno en los

algoritmos de la librería. Por lo que, para evitar errores, el cuadro de datos se construye directamente en inglés. Adicionalmente, y también traído por la misma Sección, no debe presentar estructuras de caracteres complejas, tales como: emoticonos, onomatopeyas o urls. Tampoco debe contener oraciones con un gran número de palabras (consultar Sección 4.3 para más detalles). Adicionalmente y por la sección 4.4 debe haber una dependencia de contexto en las oraciones generadas, es decir, no debe haber palabras que conviertan el sentimiento de una oración en negativo o positivo de manera absoluta.

Estudios recientes emplean herramientas de inteligencia artificial de texto generativo como *ChatGPT* (OpenAI, 2022) para la generación de cuadro de datos con las características descritas (Stein, 2023). Pero esto presenta varios inconvenientes para el caso de estudio: se intentó replicar la metodología presentada en (Stein, 2023) pero, el acceso restringido a ChatGPT por no presentar una suscripción de tipo *premium*, así como la no reproducibilidad exacta de output por parte de la herramienta no generaba ninguna garantía de obtención de un mismo cuadro de datos para el uso del experimento, por lo que se optó por la creación de uno implementando manualmente los rasgos deseados.

Así, se toma oportuno utilizar el paquete *Pandas* de Python para la construcción del cuadro de datos original. *Pandas* es una biblioteca de Python que proporciona estructuras de datos y herramientas de análisis de datos eficientes para trabajar con tablas y series temporales. De esta manera se elabora un cuadro de datos como se expone en la Figura 4.2.

Como se observa en la Figura 4.2, se generan cinco columnas por una de las limitaciones comentadas, cada una de ellas contiene palabras que, posteriormente y a través de todas las combinaciones posibles pero manteniendo el orden de columnas, serán las que generen el primer cuadro de datos.

Las columnas han sido seleccionadas para incluir riqueza lingüística y variedad en el cuadro de datos.

- **Columna 1 o sujetos**: se incluyen sujetos en singular y plural. Únicamente se incluyen dos sujetos por simplicidad del experimento.
- **Columna 2 o verbos**: se incluyen acciones ligadas al campo financiero pero cuyo

```python
import pandas as pd
from itertools import product

# Definir las palabras para cada columna
columna1 = ["I", "we"]
columna2 = ["buy", "sell", "exchange", "use"]
columna3 = ['few', 'some', 'quite']
columna4 = ["bitcoin", "ethereum", "Google", "Apple"]
columna5 = ['yesterday', 'today', 'tomorrow']

# Generar todas las combinaciones posibles
combinaciones = list(product(columna1, columna2, columna3, columna4, columna5))

# Unir las palabras de cada combinación en una sola cadena
combinaciones = [" ".join(combinacion) for combinacion in combinaciones]

# Crear el DataFrame con una sola columna
df = pd.DataFrame(combinaciones, columns=["Text"])

# Mostrar el DataFrame
print(df)
```

```
                          Text
0     I buy few bitcoin yesterday
1         I buy few bitcoin today
2      I buy few bitcoin tomorrow
3    I buy few ethereum yesterday
4        I buy few ethereum today
..                           ...
283     we use quite Google today
284  we use quite Google tomorrow
285  we use quite Apple yesterday
286      we use quite Apple today
287   we use quite Apple tomorrow

[288 rows x 1 columns]
```

Figura 4.2: Código de generación de *dataframe* en IBM Quantum.

Fuente: *elaboración propia.*

sentimiento **depende del contexto**, lo que es un requisito impuesto por la Sección 4.4.

- **Columna 3 o modificadores**: en ánimo de generar un gradiente de positivismo o negatividad se incluyen 3 adverbios en esta columna.

- **Columna 4 o complementos directos**: los complementos incluidos en esta sección cumplen dos criterios, el primero de ellos es ser pertenecientes a un ámbito financiero, pues se tratan de **acciones de una empresa** o **criptomonedas**. El segundo criterio es que si bien pudieran parecer palabras de connotación neutral se utiliza una lógica que las dota de interpretación según el contexto, como se explica en pasos posteriores.

- **Columna 5 o tiempo**: por último se incluyen tres adverbios de tiempo con una finalidad similar a los de la Columna 3.

Etiquetado del cuadro de datos

Como se expuso en el Capítulo 2, un proceso de análisis de sentimiento, y más generalmente un proceso de aprendizaje automático, puede ser *no supervisado* o *supervisado*. Un proceso de aprendizaje automático supervisado es aquel en el que se conoce la etiqueta o puntuación para cada elemento del cuadro de datos, y se usan para inferir nuevas etiquetas de otro conjunto de datos. En los procesos no supervisados, por el contrario, no se disponen de etiquetas iniciales, y por lo tanto, se tratan de ir infiriendo en el proceso de aprendizaje automático (Villalpando, 2021).

El problema de análisis de sentimiento a abordar, se trata de un problema supervisado, con lo que se deben establecer etiquetas para cada una de las entradas.

Para etiquetar el cuadro de datos se opta por asignar una puntuación a cada palabra de la oración y luego sumar dichas puntuaciones, tras un reescalado se obtiene una puntuación total de la oración que va asignada a un sentimiento.

Puesto que se parte de la premisa de que las oraciones dependen de un contexto, y el cuadro de datos presenta una estructura fija por columnas, lo primero es establecer sobre qué columnas recae la carga lingüística del contexto para identificarlas y codificarlas adecuadamente en lo que se ha denominado para el trabajo presente *core contextual*.

Como hipótesis sobre el cuadro de datos, se establece el término *core contextual*. El *core contextual* se centra en las columnas de verbo y complemento directo, después el resto de columnas intensifican el sentimiento positivo o negativo. La elección de dichas columnas para el *core contextual* se fundamenta en la teoría lingüística de (Wilson, 2015) .

El papel del *core contextual* en el código desarrollado es cambiar todos los signos de positivos a negativos en las puntuaciones de las columnas *sujeto* y *modificador* si la combinación de palabras es la adecuada. La columna *tiempo* no se ve afectada por el *core contextual* como se explica más adelante.

Para asignar la etiqueta al *core contextual*, se realiza la suma binaria de las columnas

verbo y *complemento directo*, denotando por 0 si el *core contextual* posee connotación negativa o 1 en caso contrario.

La puntuación de los complementos directos se puede obtener de forma inmediata a través de la consulta de los valores de cotización en bolsa; el día 18/07/2023 se consultaron los valores de cotización en bolsa de bitcoin, ethereum, Google y Apple (Google, 2023), y se les asignó un 0 si el valor se encontraba por debajo del valor de cierre del día anterior, y un 1 en el caso contrario.

Para asignar la puntuación a los verbos, se establece para cada uno qué connotación debe tomar en caso de ser un 0 o un 1 el complemento directo, así se llega a la siguiente tabla:

Verbo	CD	Suma binaria	Core
Buy	0	1	1
	1	0	
Sell	0	0	0
	1	1	
Exchange	0	0	0
	1	1	
Use	0	0	0
	1	1	

Tabla 4.1: Etiquete del *core contextual*.

Fuente: *elaboración propia*

Una vez se tiene el *core contextual* etiquetado, se les asigna una puntuación al resto de elementos de la oración bajo la máxima:Cuando más alta sea la puntuación de una palabra más positiva es y cuanto más baja menos.

Se debe tener en cuenta que si el *core contextual* tiene una puntuación de 1, las palabras mantendrán su puntaje intacto, mientras que en otro caso se les invierte el signo a todas ellas. Así se etiquetan las columnas de sujeto, modificador y tiempo de la siguiente manera y bajo las siguientes lógicas (se explica la su significado en un contexto positivo y de él se desprende el caso contrario):

- **Sujetos:** cuanta más gente se beneficie de la acción más positivo será el sujeto.
- **Modificiadores:** cuanta más cantidad de complemento directo se obtenga mejor puntuación tendrá el sujeto.
- **Tiempo:** cuanto más segura sea la acción en el tiempo más positivo, es decir, cuanto más se adentre en el pasado más positivo sera el adverbio y cuanto más en el futuro peor. El tiempo no se ve afectado por el *core contextual* pues es un absoluto.

Bajo estas premisas se obtienen las puntuaciones, como se aprecia en la Figura 4.3.

```
# Cuanta mas gente lo haga mejor.
ss = {'I': 1, 'we': 2}

# Valores determinados por suma binaria con cds.
vv = {'buy': 1, 'sell': 0, 'exchange': 0, 'use': 0}

# Cuanto más mejor
mm = {'few': 1, 'some': 2, 'quite': 3}

# Valores variables por la bolsa, si baja 0 si sube 1.
cds ={'bitcoin': 1, 'ethereum': 0, 'Google': 0, 'Apple': 1}

# Cuanto mas tarde en el tiempo peor, mas puede variar.
tt = {'yesterday': 2, 'today': 1 , 'tomorrow': 0}
```

Figura 4.3: Puntuaciones asignadas a cada palabra.

Fuente: *elaboración propia.*

A continuación se suman todas las puntuaciones, por el funcionamiento del *core contextual*, se presentan distintas horquillas de valores que pueden tomar las oraciones dependiendo de éste, por lo que hay que hacer el siguiente reescalado en base a las horquillas generadas:

```
# Minimo positivo (vv + cds = 1): 2 (1+1+0)
# Maximo positivo (vv + cds = 1): 7 (2+3+2)

# Minimo negativo (vv + cds = 0): -5 (-2-3+0)
# Maximo negativo (vv + cds = 0): 0 (-1-1+2)

# DIVIDIMOS LAS ETIQUETAS EN 4 PUES TENEMOS LAS HORQUILLAS [-5,0] y [2,7]:
# [[-5,-3],[-2,0],[2,4],[5,7]] -> [0,1,2,3]
```

Figura 4.4: Explicación de las horquillas generadas en el etiquetado y reescalado.

Fuente: *elaboración propia.*

Para seguir, se nombra la columna con las oraciones generadas como *'Text'* y se le añade una nueva columna al cuadro de datos a la izquierda de ésta nombrada *'Score'* donde se colocan las puntuaciones de las oraciones.

Por último, se baraja el cuadro de datos y se dividen las filas en dos cuadros de datos distintos. Con proporción del 80 % de las filas se crea el cuadro de datos de entrenamiento y con el 20 % de las filas el de prueba (Ghosh, 2019).

Creación del mapa de características

La elección de un mapa de características adecuado es fundamental para abordar un problema, ya que puede ser que la codificación de los datos no genere un espacio de fases en el que se garantice la separabilidad de estos, tal y como se vio en el Capítulo 2. Así, las transformaciones realizas en el espacio origen tienen que garantizar que se pueden discernir los elementos en el espacio destino.

Si bien existen mapas de características prediseñados como *PauliFeatureMap*, *ZFeatureMap* ó *ZZFeatureMap* (QisKit, 2023b), en los que no se profundizará, que en general consiguen una buena codificación para los espacios destino, parte del experimento es la elaboración de una codificación propia, por lo que sirviéndose de rasgos generales de estos mapas de características como es el uso de puertas cuánticas de rotación, como la puerta $\hat{Rx}$ 2.7 y $\hat{Ry}$ 2.8, y la inclusión de relaciones entre qubits a través de puertas $C\hat{N}OT$ 2.1, para plasmar relaciones entre qubits, se construye el mapa de características presente en Figura 4.5.

Como se puede ver en la Figura 4.5, se ha creado un circuito cuántico cinco qubits, uno por cada palabra de la oración a codificar, lo que significa que se realizara una codificación en amplitud (Noori, 2020). Los qubits se nombran con nomenclaturas iguales que los diccionarios que contienen las palabras, de arriba a abajo: 's': sujeto, 'v': verbo, 'mod': modificador, 'cd': complemento directo y 't': tiempo. Por abreviar en adelante se referirá a cada qubit por su correspondiente abreviatura

Primero, para dar justificación a la conexión establecida nos centramos en los qubits relacionados con el *core contextual*, es decir 'v' y 'cd'. Se observa una Puerta Toffoli 2.1 entre el *core contextual* y los qubits que se mencionaron que se verían afectados por él

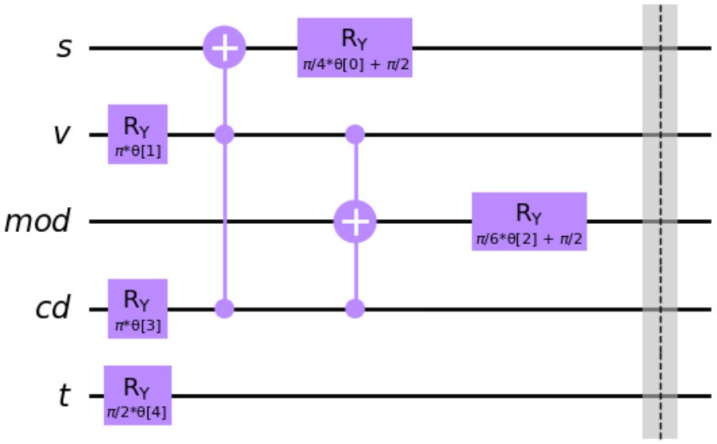


Figura 4.5: Mapa de características creado para el problema.
Fuente: *elaboración propia.*

según (Wilson, 2015), a saber 's' y 'mod'. Además el entrelazamiento es un rasgo común que se deseaba importar de los mapas de características prediseñados.

Si bien la explicación de que todas las puertas contenidas en el mapa de características sean de rotación RY se debe a su codificación en amplitud, la explicación del ángulo incluido en cada puerta de rotación del mapa de características debe abordarse por separado para su mejor comprensión.

De nuevo, entrando en los qubits del *core contextual*, se encuentra un parámetro, θ multiplicado por π, es decir, puesto que los valores de ambos parámetros solo pueden ser 0 o 1, se estaría encontrando una una codificación inicial de $|0\rangle$ o $|1\rangle$.

Respecto a 's', se emplea una fase con la expresión:

$$p(\theta) = \frac{\pi}{4}\theta + \frac{\pi}{2} \tag{4.1}$$

Se elige tal representación desde que los posibles valores son 1 para 'I' y 2 para 'we', quedando para los posibles valores del *core contextual*:

$$\theta = 1 \longrightarrow \frac{\pi}{4} \cdot 1 + \frac{\pi}{2} = \frac{3\pi}{4}$$

$$\theta = 2 \longrightarrow \frac{\pi}{4} \cdot 2 + \frac{\pi}{2} = \pi$$

$$\theta = -1 \longrightarrow \frac{\pi}{4} \cdot (-1) + \frac{\pi}{2} = \frac{\pi}{4}$$

$$\theta = -2 \longrightarrow \frac{\pi}{4} \cdot (-2) + \frac{\pi}{2} = 0 \tag{4.2}$$

Si se plasma en la esfera de Bloch (Heusler, 2020) estos estados, se tiene la Figura 4.10.

En la Figura 4.10 se observa, de izquierda a derecha y de arriba a abajo, en primer lugar, el estado asociado a 'I' para codificación positiva, seguido del estado asociado a 'I' para codificación negativa. En la parte inferior de la figura, el estado asociado a 'We' para codificación positiva, seguido del estado asociado a 'We' para codificación negativa.

Como se observa en 4.10 la codificación ha sido seleccionada para que, dependiendo del valor del *core contextual* 's' aporte mayor o menor positividad a la oración. En un principio se planteó codificar 'I' para que presentara una fase de $\frac{\pi}{2}$, ya que la fase asociada a 'We' es π y para mantener una proporcionalidad sería lo adecuado. Sin embargo, no sería posible discernir en qué caso se encuentra 'I', si en el de un contexto negativo o uno positivo, por ello se opta por elegir dicha codificación en fase, ya que sí hay un desplazamiento probabilístico relacionado con el *core contextual*.

Una vez vista la explicación de la elección de la fase para 's', se desprende inmediatamente la de 'mod' y 't', entendiendo que se selecciona una codificación gradual por el espectro de $[|0\rangle, |1\rangle]$ con la intención de plasmar la intensidad de sentimiento de la oración.

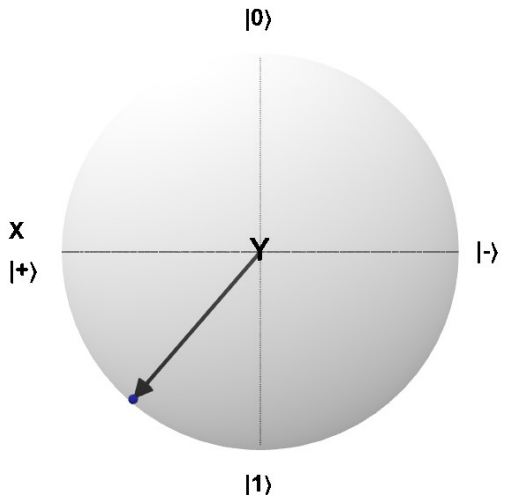


Figura 4.6: (a) Estado asociada a 'I' para connotación positiva.

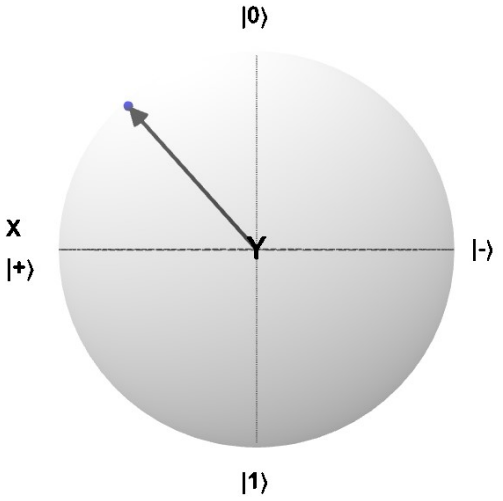


Figura 4.7: (b) Estado asociada a 'I' para connotación negativa.

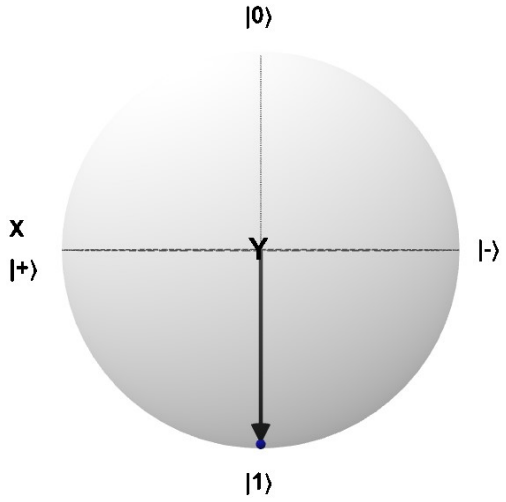


Figura 4.8: (c) Estado asociada a 'We' para connotación positiva.

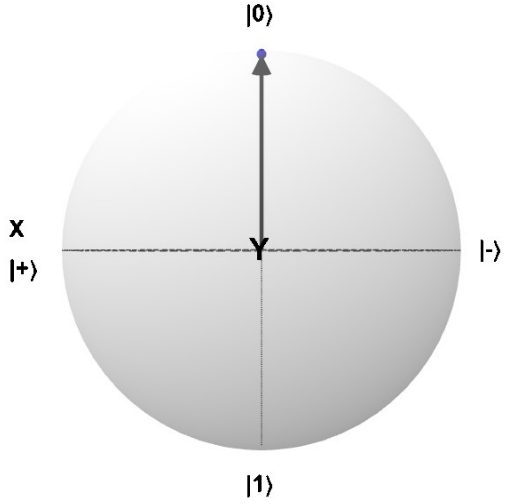


Figura 4.9: (d) Estado asociada a 'We' para connotación negativa.

Figura 4.10: Estados de los elementos 's' sobre la esfera de Bloch.
Fuente: adaptada de JavaFXpert (2020).

Creación del ansatz

Si bien *QisKit* presenta una función dentro del paquete **circuits** denominada **RealAmplitudes** (QisKit, 2023c) que genera un ansatz de amplitudes reales y que se pueden modi-

ficar ciertas características como: tipo de rotaciones, tipo de entrelazamiento o repeticiones del patrón entre otras, es parte del experimento el desarrollo de un ansatz propio para el estudio que se aborda. Así, basando su construcción en pautas incluidas en (Dallaire, 2019) como:

- Inclusión de conexión entre todos los qubits para una mayor variabilidad en el espacio de fases.
- Inclusión de puertas de rotación sobre la esfera de Bloch, tratando de evitar puertas de rotación sobre el eje Z, que es el que contiene ambos elementos de la base del espacio del qubit. El problema abordado se limita al espectro real por la poca variabilidad incluida, lo que como ya se ha especificado, se traduce en una codificación en amplitud.
- Inclusión de puertas Hadamard por la expansión del espacio de fases que genera. Si no se tiene una intuición clara de los estados buscados es recomendable incluir estas puertas.

De estas indicaciones se construye el ansatz de la Figura 4.11.

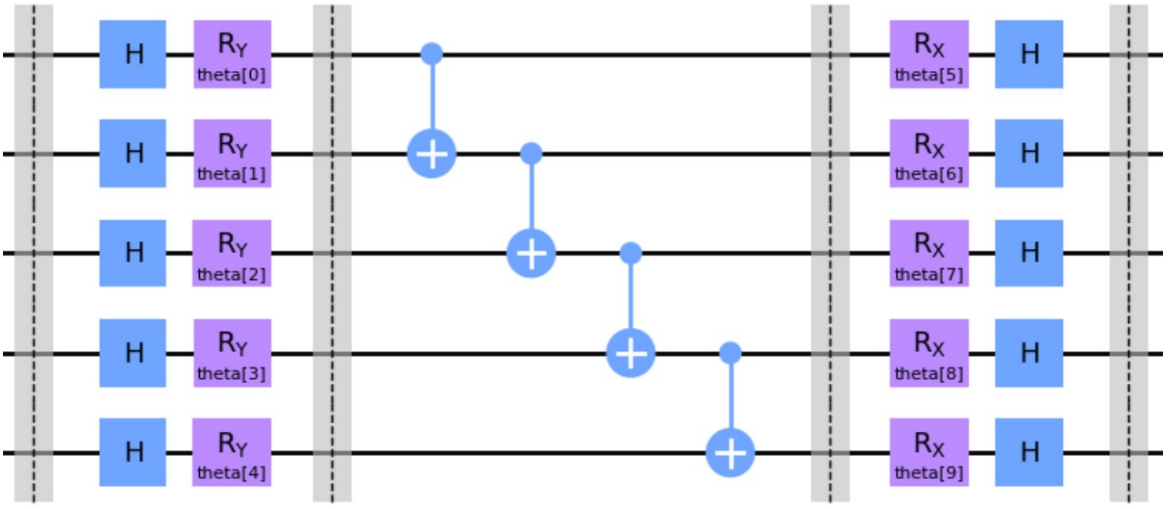


Figura 4.11: ansatz elegido para el estudio.

Fuente: *elaboración propia.*

Como se observa, el ansatz presenta dos hileras de puertas Hadamard a ambos extremos, una hilera de puertas de rotación Ry con un parámetro por puerta, entrelazamiento entre todos los qubits seguido de una hilera de puertas de rotación Rx con un parámetro

por puerta.

Se ha elegido incluir dos tipos de rotaciones con parámetros para mayor variabilidad y que se recorra adecuadamente toda la superficie de la esfera de Bloch, para lo cual las puertas Hadamard son solo una ayuda. Si se tiene en cuenta el entrelazamiento generado se observa como se cumplen todas las pautas designadas por (Dallaire, 2019).

Función de salida, coste y optimizador

Respecto a la función de salida, que se recuerda del Capítulo 2 que se trata de la función que se aplica a los resultados de una iteración para la predicción de la etiqueta de la oración introducida, puesto que clásicamente los valores de puntuación de la oración vienen determinados poro 's', 'mod' y 't', son estos qubits los que se hacen colapsar. Además, dentro de los estados obtenidos en los distintos conteos o *counts* tras la ejecución del circuito a través de los paquetes Aer, transpile y execute (QisKit, 2023d) (consultar Apéndice A para más detalles), se selecciona el estado con mayor cantidad de conteos, para finalmente sumar de forma individual todas las amplitudes del estado cuántico correspondiente, siendo coherente en ese sentido con su codificación. Así por ejemplo, si el estado con mayor conteos tras la ejecución es el estado $|101\rangle$ la operación realizada a la salida para obtener el puntaje de la oración es: 1+0+1 = 2.

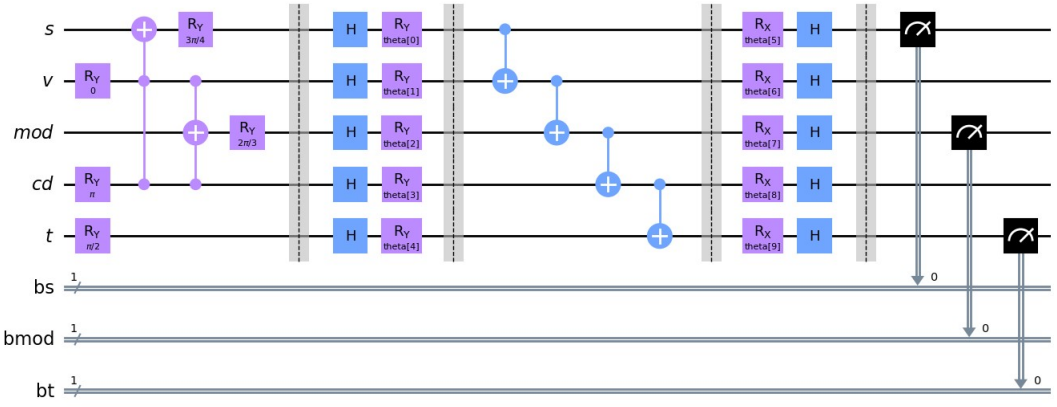


Figura 4.12: Estructura del circuito completo.

Fuente: *elaboración propia.*

Hay que notar que los posibles resultados de esta metodología de 'etiquete cuántico' son: 0,1,2,3 . Es decir, precisamente los valores de las etiquetas del cuadro de datos, con

lo que se puede medir el error directamente de ambas salidas.

La manera de calcular el error en la función de coste es simplemente a través del error cuadrático medio pues, al compartir etiqueta es una medida bastante directa sin necesidad de asignar una ponderación a la medida de dicho error (Villalpando, 2021).

Por último, el proceso de optimización de parámetros se realiza a con el algoritmo COBYLA (QisKit, 2023e). COBYLA (Constrained Optimization BY Linear Approximations) es un algoritmo de optimización numérica utilizado para resolver problemas de optimización no lineal con restricciones. Éste se usa a través del optimizador *minimize* que proporciona la librería *SciPy* (SciPy, 2023), pues los optimizadores de la librería *optimizers* de *QisKit* no se adecuaban bien a los requisitos del problema abordado, al solicitar parámetros de entrada para su configuración inexistentes en el desarrollo del análisis propuesto (para más detalles consultar ApéndiceA)

Procesos de entrenamiento y testeo

Una vez definidos los procesos de optimización se entra en la etapa de retroalimentación de parámetros y ensamble de procesos para la creación del modelo propiamente dicho.

Existe una diferencia fundamental en el proceso de entrenamiento entre los procesos NLP y el resto de procesos de aprendizaje automático, pues parámetros que podrían ser óptimos para una oración no tienen porque serlo para otros (Meichanetzidis, 2020). Esto se extrapola a los procesos de QNLP, por ello en el desarrollo se debe buscar un algoritmo con los parámetros óptimos encontrados para cada una de las oraciones con el fin de tratar de optimizar de forma general los parámetros para todo el cuadro de datos.

La metodología empleada para este proceso consiste en hacer pasar unos parámetros iniciales al proceso de aprendizaje automático y, tras optimizarlos para cada oración, conservar aquellos que consigan un error de la función de coste de 0, ya que son los que más se han conseguido adaptar al espacio de fases.
Tras ello se establece una media de todos los parámetros obtenidos, consiguiendo así, no unos parámetros óptimos para cada oración sino unos parámetros que, en conjunto, son los más adecuados para el cuadro de datos. En general, no hay una solución que no genere

ningún tipo de desavenencia sobre los resultados pues, si se elige priorizar ciertos parámetros se está sacrificando la certeza sobre el resto, por lo que una media simple sobre los parámetros más óptimos es una solución equilibrada que, a priori, puede aportar buenos resultados.

Para el proceso de testeo, en consecuencia de la estrategia seguida en el proceso de entrenamiento, se utilizan los parámetros obtenidos resultado de la media de los parámetros más óptimos.

4.2. Aproximación por redes neuronales cuánticas

Metodología

Para la implementación de la red neuronal cuántica a modo de clasificador para el cuadro de datos mostrado en la Sección anterior, se recurre a las librerías *QisKit* y *Scikit-Learn* (SciKit, 2023). Esta última proporciona acceso a herramientas para la construcción de redes neuronales cuánticas.

El proceso a seguir es el desarrollo de una red neuronal cuántica con la codificación del mapa de características y ansatz abordado en la Sección anterior. Posteriormente, se implementa otra red neuronal cuántica, pero esta vez con elementos proporcionados por las librerías, con el ánimo de discernir cómo de buena es la propia codificación para el paradigma que se aborda en esta Sección.

Un rasgo importante a marcar respecto al tratamiento de la Sección anterior es que el funcionamiento de redes neuronales cuánticas a través de dichas librerías se restringe para etiquetas de 0 y 1. Por lo que el primer paso es reescalar las etiquetas del cuadro de datos. Para ello se asignan los valores más bajos de los cuatro posibles presentes en el etiquetado previo (0,1) a 0, y los otros dos valores restantes más positivos (2,3) a 1.

Se usa la función **OpFlowQNN** (QisKit, 2023f) de *QisKit* para la construcción de la red neuronal cuántica. Esta función recibe como argumentos:

- **operator**: el operador de la matriz del circuito cuántico junto con el observable

utilizado. Este operador representa el estado cuántico parametrizado que se convierte en red neuronal cuántica. En primera instancia se selecciona el operador mas sencillo posible para la red, este es el operador identidad, I.

- **inputs_params**: los parámetros del operador relacionados con la codificación del mapa de características.
- **weight_params**: los parámetros del operador relacionados con el ansatz del circuito.
- **exp_val**: se selecciona el método por el que calcular el valor esperado para el proceso a ejecutar. En este caso simplemente *AerPauliExpectation()* del paquete Aer de *QisKit*.
- **gradient**: el gradiente por el que se minimiza la función convertido para el uso de la red neuronal. Se implementa *Gradient()* de *QisKit* que convierte la expresión del operador en un gradiente de primer orden.
- **quantum_instance**: la configuración del *backend* utilizada para el proceso de ejecución. Del paquete Aer se usa el *backend qasm_simulator*.

Una vez definida la red neuronal cuántica, se especifica el tipo de proceso a realizar con ella. En el caso que atañe, se está realizando un proceso de clasificación, por lo que se recurre a la función *NeuralNetworkClassifier()* de *QisKit*. Ésta recibe como argumentos la red neuronal cuántica definida y un optimizador, de nuevo se recurre al optimizador COBYLA, como en la sección anterior, resulta adecuado para el problema al poder abordar funciones diferenciables como es el caso de las funciones de activación de las neuronas cuánticas (Jia, 2018).

Por último, se introducen las etiquetas y características del *dataframe*, estas últimas no son más que las puntuaciones de cada palabra en la oración.

El proceso se repite para un observable mas complejo compuesto por las matrices de rotación sobre el eje 'y' y 'z' de la esfera de Bloch. Formando el operador YZ sobre cada qubit.

Tras el estudio del problema con elementos propios desarrollados, se repite el experimento de esta sección haciendo uso de un mapa de características y ansatz de *QisKit*.

Adicionalmente y a través de la función *CircuitQNN* (QisKit, 2023g) se puede reproducir un clasificador variacional cuántico directamente a través de los mismos argumentos que para la función *OPFlowQNN*, salvo a falta de incluir una función de coste.

Para ello se emplea una función incluida por defecto en la librería de *QisKit*: función de coste de pérdida de entropía cruzada binaria (BCE por sus siglas en inglés). Su objetivo principal es cuantificar la diferencia entre las probabilidades predichas por un modelo y las probabilidades reales de las etiquetas de los datos de entrenamiento (Bruch, 2019).

4.3. Librería *Lambeq*

Lambeq es una biblioteca de Python de nivel superior, modular y extensible de código abierto para el QNLP, creada por el equipo de Quantinuum (Quantinuum, 2023a).

A un nivel alto, la biblioteca permite convertir cualquier oración en un circuito cuántico basado en un modelo compositivo dado y ciertas parametrizaciones y elecciones de ansatz. Además facilita el entrenamiento tanto para experimentos cuánticos como clásicos de procesamiento del lenguaje natural (Quantinuum, 2023a). Una descripción general de un proceso estándar de QNLP con Lambeq se muestra a continuación:

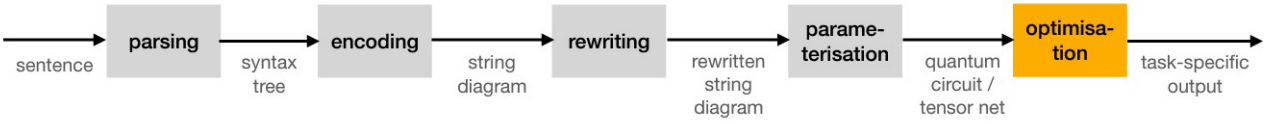


Figura 4.13: Proceso estándar de QNLP con la librería Lambeq.
Fuente: adaptada de Kartsaklis (2021).

En la Figura 4.13 se ve una serie de procesos que se desarrollan a continuación para la mejor comprensión de la librería y herramientas utilizadas.

Se debe tener en cuenta que, al igual que para redes neuronales cuánticas ejerciendo como clasificador, la librería Lambeq solo es capaz de clasificar en 0 y 1 para el análisis de sentimiento, por lo que se realiza el mismo procesado sobre el cuadro de datos que para la Sección anterior, quedando las etiquetas con la escala adecuada para la librería usada.

Además, es importante destacar que Lambeq es una librería con potencia limitada por la codificación en amplitud que realiza, así como la gran profundidad de los circuitos generados asociados a oraciones que, por encontrarse la computación cuántica en la era NISQ (Perskill, 2018). Esto provoca que, para poder realizar análisis en rangos de tiempo asumibles por las computadoras de acceso disponible se tenga que limitar la longitud de las oraciones de los cuadros de datos implicados en un rango de, aproximadamente, 0 y 7 palabras según los estudios realizados (Shen, 2018)(Martinez, 2022)(Stein, 2023).

Metodología

Lambeq, en primera instancia, toma las oraciones a analizar y las pasa por lo que se denomina en la Figura 4.13 como *parsing*. Este proceso se lleva a cabo a través de la propia clase de Lambeq *BobcatParser*. Y consiste principalmente en aplicar una descomposición en elementos, es decir, separar las oración por elementos como y elementos ortográficos para discriminar caracteres poco relevantes para el análisis morfológico como puntos o comas. Además se le asigna una etiqueta a cada palabra dependiendo del tipo de palabra morfológica que sea, esto ayuda a identificar estructuras gramaticales almacenadas en la propia librería Lambeq.

A la clase BobcatParser se le aplica un método denominado **sentences2diagrams**, que representa los bloques de *encoding* y *rewriting* en la Figura 4.13 donde, una vez descompuestas las oraciones en elementos, se identifica la gramática subyacente a la estructura interna de la oración y se establece un diagrama con ella que, posteriormente se convierte en el circuito codificado. Estos diagramas configuran la estructura del mapa de características y ansatz que se va a generar en el circuito asociado a la oración.

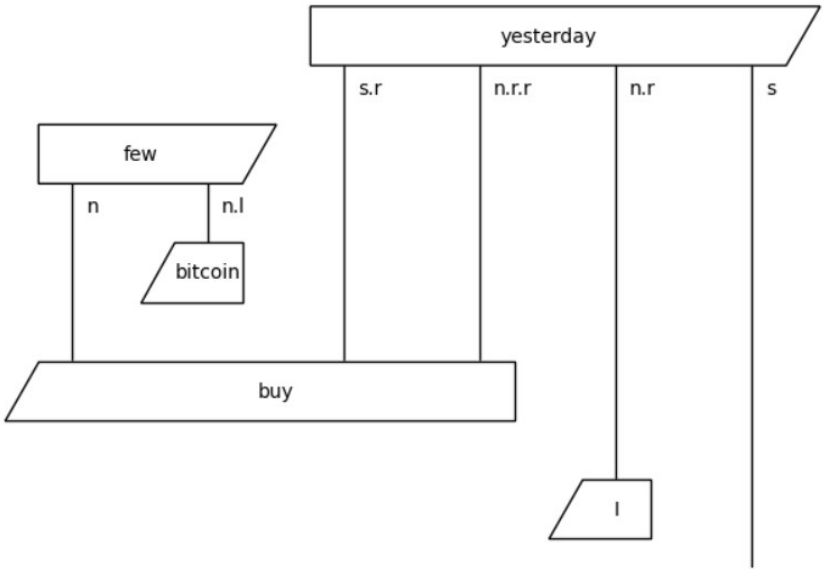


Figura 4.14: Diagrama generado por 'sentences2diagrams'.
Fuente: *elaboración propia.*

Como se observa en Figura 4.14, por la clase BobcatParser se identifica el tipo de palabra y se le asigna una serie de salidas dependiendo de su clase. Estas salidas hacen de entradas para otras palabras, componiendo la estructura gramatical y contextual de toda la oración.

En esta parte del proceso, Lambeq establece diferencias únicamente entre elementos que son verbos, que denomina como 's', y los que no lo son, que denomina como 'n'. Posteriormente y dependiendo de la categoría morfológica de cada palabra, asigna una serie de salidas y entradas. Las salidas se nombran por la categoría de palabra que son y las entradas por el tipo de palabras que esperan recibir y la ubicación de la misma. Así por ejemplo, la palabra 'few' presenta una salida, denominada 'n', que hace referencia a que no es un verbo, y para concretar que se trata de un adverbio presenta la entrada 'n.l' que refiere a que se ubica a la izquierda (left, en inglés) de un nombre ('I' en este caso).

Para la transformación de los diagramas gramaticales en circuitos cuánticos que la librería pueda procesar para aprendizaje automático, se recurre a la clase *IQPAnsatz* (Discopy, 2023). Esta clase crea un ansatz tipo IQP, es decir, un ansatz con una estructura

compuesta de puertas Hadamard, puertas control Z y puertas de rotación R_x y R_y con la cantidad de qubits indicada por el diagrama , tal y como se ve en la Figura 4.15.

```
>>> pprint = lambda c: print(str(c.foliation()).replace(' >>', '\n  >>'))
>>> pprint(IQPansatz(3, [[0.1, 0.2], [0.3, 0.4]]))
H @ H @ H
  >> CRz(0.1) @ qubit
  >> H @ CRz(0.2)
  >> qubit @ H @ H
  >> CRz(0.3) @ qubit
  >> qubit @ CRz(0.4)
>>> print(IQPansatz(1, [0.3, 0.8, 0.4]))
Rx(0.3) >> Rz(0.8) >> Rx(0.4)
```

Figura 4.15: Estructura interna de los ansatz generados por Lambeq.
Fuente: adaptada de Discopy (2023).

Con los circuitos cuánticos asociados a cada oración se debe construir el modelo y definir el entrenador para el proceso de aprendizaje automático.

Para el modelo se selecciona *TketModel* de la librería TKET de Quantinuum (Quantinuum, 2023a). TKET es un kit de desarrollo de software cuántico para la creación y ejecución de programas para computadores cuánticos basados en puertas. TKET es de código abierto y de fácil acceso a través del paquete PyTKET Python. El modelo TketModel toma los circuitos generados a través de 'IQPAnsatz' junto con un backend, en el caso que atañe se toma un motor estándar,'AerBackend()', para formar el modelo del proceso de entrenamiento.

Una vez se tiene el modelo para el entrenamiento, queda definir una función de coste y un optimizador. Para la primera se escoge *BinaryCrossEntropyLoss()* (Kulkarni, 2022). Es una función de pérdida comúnmente utilizada en problemas de clasificación en el campo del aprendizaje automático. Su objetivo principal es medir la discrepancia entre las probabilidades predichas por un modelo y las etiquetas reales de los datos. La fórmula que plasma lo anterior es:

$$\mathrm{BCE}(y, \hat{y}) = -\left(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})\right) \tag{4.3}$$

donde y representa la etiqueta real (1 para la clase positiva, 0 para la clase negativa), $\hat{y}$

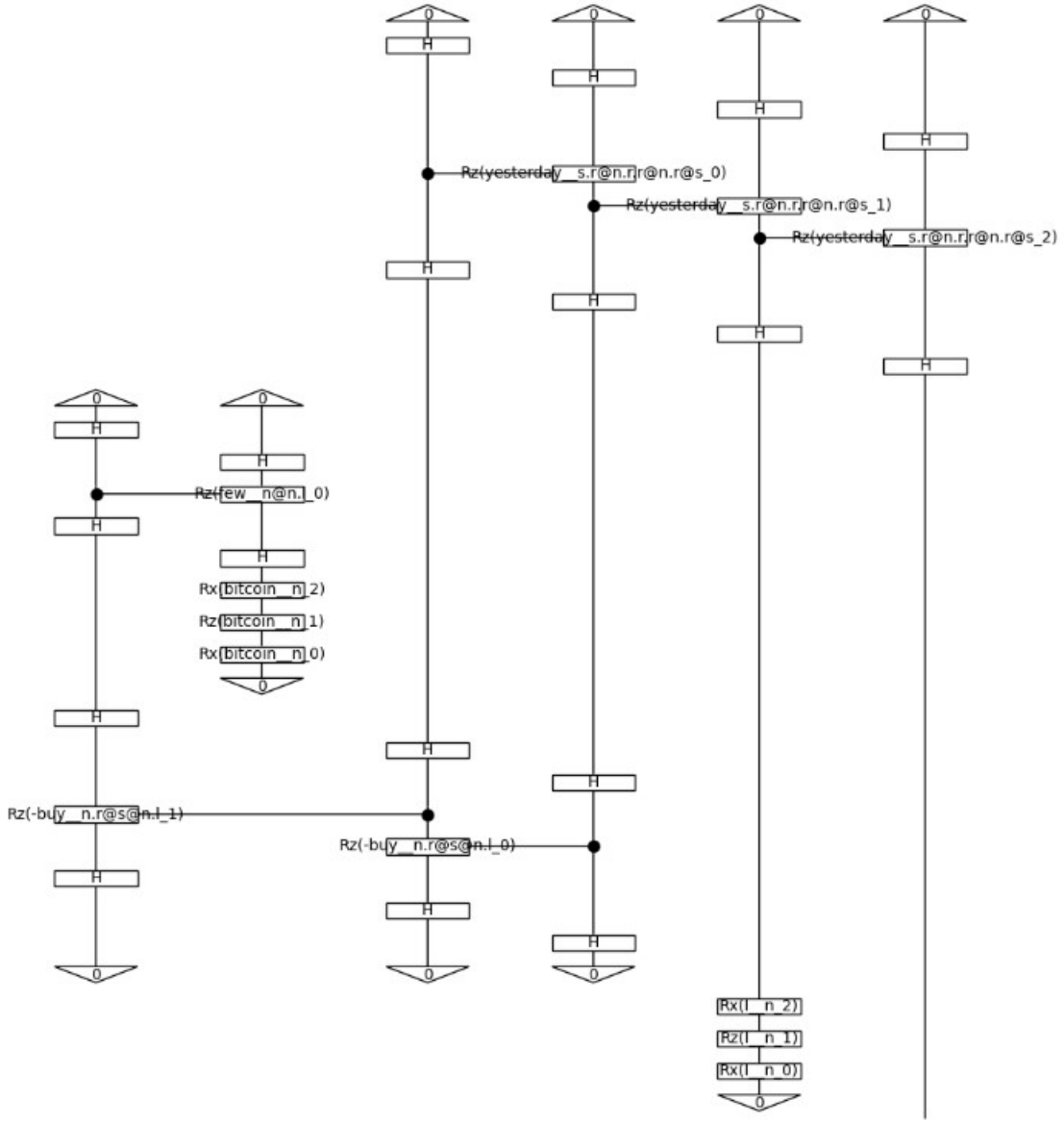


Figura 4.16: Transcripción del diagrama asociado a la oración.
Fuente: *elaboración propia.*

representa la probabilidad predicha por el modelo de que el ejemplo pertenezca a la clase positiva.

Por último, se selecciona como optimizador *SPSAOptimizer* de la propia librería Lambeq. El optimizador de aproximación estocástica de perturbación simultánea (SPSA), es

un método de descenso de gradiente para optimizar sistemas con múltiples parámetros desconocidos (Wiedmann, 2023). Como método de optimización, es adecuado para modelos de población a gran escala, modelado adaptativo y optimización de simulaciones. La característica principal de SPSA es la aproximación estocástica del gradiente, que requiere solo dos mediciones de la función objetivo, independientemente de la dimensión del problema de optimización, lo cual es conveniente por la alta dimensionalidad que puede alcanzar el espacio de fases generado en cada oración (Quantinuum, 2023b).

Con todo ello, a través de la clase *QuantumTrainer* de Lambeq, se ejecuta el modelo seleccionando una cantidad de épocas (epoch, en inglés), cantidad de circuitos simultáneos a tomar para el entrenamiento (batch size, en inglés) y semilla (seed, en inglés) para la replicabilidad del experimento (consultar Apéndice A para más detalles) y a través de la función de coste se determina el error con el que se obtienen los resultados que se clasifican.

```
from lambeq import QuantumTrainer, SPSAOptimizer

trainer = QuantumTrainer(
    model = TketModel.from_diagrams(all_circuits, backend_config=backend_config),
    loss_function = BinaryCrossEntropyLoss(),
    epochs=EPOCHS,
    optimizer=SPSAOptimizer,
    optim_hyperparams={'a': 0.05, 'c': 0.06, 'A':0.01*EPOCHS},
    evaluate_functions={'acc': acc},
    evaluate_on_train=True,
    verbose = 'text',
    seed= SEED
)
```

Figura 4.17: Entrenador utilizado por Lambeq.
Fuente: *elaboración propia.*

4.4. Caso clásico: modelo RoBERTa

El último caso de estudio y para comparativa con los casos que implementan computación cuántica, se recurre a un experimento a través del modelo de procesamiento de lenguaje natural de *Google* obtenido de la plataforma de modelos NLP *HuggingFace*, RoBERTa (Hugging Face, 2023).

La ventaja principal del modelo RoBERTa respecto a otros modelos de procesamiento del lenguaje natural probados como VADER (Hutto, 2023), es que al estar basado en

```
import torch

MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
model.save_pretrained(MODEL)
```

Figura 4.18: Importación del modelo RoBERTa por HuggingFace.
Fuente: *elaboración propia.*

el modelo de representaciones de codificador bidireccional de transformadores de Google (BERT por sus siglas en inglés) tiene la capacidad de análisis del lenguaje a nivel de contexto, siendo más adecuado para el experimento en cuestión al asemejarse a las condiciones de modelos previos de computación cuántica utilizados, como el modelo propio desarrollado o Lambeq.

BERT es un modelo de lenguaje desarrollado por Google en 2018 (Delvin, 2018). Es un tipo de modelo de procesamiento del lenguaje natural (NLP) basado en redes neuronales y utiliza la arquitectura de atención de transformers (Delvin, 2018).Las redes neuronales basadas en la arquitectura transformer revolucionaron el campo del procesamiento de secuencias y el aprendizaje automático. Su innovación central reside en el mecanismo de atención auto-atencional, el cual revoluciona la capacidad de modelar dependencias a largo plazo y relaciones semánticas complejas en secuencias. Esto se logra mediante la ponderación dinámica de cada posición en la secuencia de entrada, otorgando a estas redes la capacidad de contextualizar la información sin depender de una estructura de dependencia temporal lineal.

La mayoría de los modelos neurales de transducción de secuencias más competitivos tienen una estructura codificador-decodificador. Los modelos transformer en este contexto, presentan un codificador que mapea una secuencia de entrada de representaciones de símbolos $(x_1, ..., x_n)$ a una secuencia de representaciones continuas $(z_1, ..., z_n)$. Dado z, el decodificador genera entonces una secuencia de salida $(y_1, ..., y_m)$ de símbolos, uno a la vez. En cada paso, el modelo es auto-regresivo, lo que significa que consume los símbolos previamente generados como entrada adicional al generar el siguiente (Vaswani, 2017).

BERT se destacó por su capacidad para comprender el contexto y el significado de las palabras en un texto al considerar las palabras circundantes en ambas direcciones (izquierda y derecha) en lugar de solo una dirección, como lo hacían los modelos de lenguaje anteriores (Delvin, 2018).

Un enfoque de preentrenamiento BERT sólidamente optimizado (RoBERTa por sus siglas en inglés) es un estudio de replicación del preentrenamiento BERT, que incluye una evaluación cuidadosa de los efectos del ajuste de parámetros y el tamaño del conjunto de entrenamiento. Se encuentra que BERT estaba significativamente poco capacitado y se propone una receta mejorada para entrenar modelos BERT, que se ha denominado RoBERTa, que puede igualar o superar el rendimiento de todos los métodos posteriores a BERT. Las modificaciones incluidas son simples:

- (1) entrenar el modelo por más tiempo, con lotes más grandes, con más datos.
- (2) eliminar el objetivo de predicción de la siguiente oración.
- (3) entrenamiento en secuencias más largas.
- (4) cambiando dinámicamente el patrón de enmascaramiento aplicado a los datos de entrenamiento.

También se recopila un gran conjunto de datos nuevo (CC-NEWS) de tamaño comparable a los conjuntos de datos de uso privado, para controlar mejor los efectos del tamaño del conjunto de entrenamiento (Liu, 2019).

Metodología

Para el desarrollo de este experimento se hacen uso de las librerías:

- **NLTK**: la librería NLTK (Natural Language Toolkit) es una de las herramientas más populares y poderosas para el procesamiento del lenguaje natural (NLP, por sus siglas en inglés) en Python bajo un paradigma clásico. NLTK proporciona una amplia gama de herramientas y recursos para trabajar con texto y lenguaje natural de manera efectiva (Kumar, 2022).

- **Transformers**: La librería transformers se centra en implementar modelos de lenguaje basados en redes neuronales con arquitectura transformer, que ha revolucionado el campo del NLP desde su introducción. Los transformers son conocidos por su capacidad para manejar secuencias de texto de manera eficiente mediante mecanismos de atención (Delvin, 2018).

- **PyTorch**: una potente librería de aprendizaje profundo que se destaca por su soporte de tensores dinámicos, lo que facilita la construcción y depuración de modelos de manera flexible. Ofrece un conjunto completo de herramientas para construir y entrenar redes neuronales, así como diferenciación automática para calcular gradientes (Bartz-Beielstein, 2023).

En este punto del proceso se sigue a través de la librería NLTK para la tokenización de las oraciones, este proceso es común a la librería Lambeq pues, se separan las frases por palabras, se identifican el tipo morfológico de cada una y se les asigna una etiqueta con el tipo correspondiente.

La diferencia fundamental entre la metodología del paradigma cuántico de Lambeq y el paradigma clásico de RoBERTa es la finalidad con la que se etiquetan las palabras en el proceso anteriormente mencionado. Mientras que en el paradigma cuántico se etiquetan con tal de identificar la gramática en busca de una estructura extrapolable a circuitos cuánticos, en el paradigma clásico se asignan etiquetas con el tipo morfológico con la intención de asignarle una puntuación a cada palabra que, dependiendo de la arquitectura del proceso de aprendizaje automático, procederá de la optimización de parámetros de una red neuronal preentrenada o bien de una base de datos extensa con puntuaciones asignadas a cada token.El caso de modelos tipo BERT y concretamente RoBERTa, es un modelo basado en redes neuronales clásicas con lo que la puntuación a las palabras procede de parámetros preentrenados. Sin embargo, al ser un modelo que tiene en cuenta el contexto de la frase, como se ha visto al principio del capítulo, se manifiesta ponderando los valores de acuerdo al peso intencional de cada token dentro de la oración gracias a la librería Transformers.

Una vez separadas en elementos las oraciones, se procesan por el codificador del modelo para correrlas por RoBERTa y obtener así la puntuación correspondiente a cada una de

las palabras de la oración. Destacar que, el modelo RoBERTa, asigna tres tipo de puntuaciones: negativa, neutral y positiva. Y dentro de cada una de estas categorías presenta un espectro continuo entre 0 y 1. Por lo que el resultado final de cada oración tras haber pasado por RoBERTa presenta las tres puntuaciones como se muestra en Figura 4.19.

```
# Run for Roberta Model
encoded_text = tokenizer(example, return_tensors='pt')
output = model(**encoded_text)
scores = output[0][0].detach().numpy()
scores = softmax(scores)
scores_dict = {
    'roberta_neg': scores[0],
    'roberta_neu': scores[1],
    'roberta_pos': scores[2]
}

print(scores_dict)
```

```
{'roberta_neg': 0.021732017, 'roberta_neu': 0.7869983, 'roberta_pos': 0.19126976}
```

Figura 4.19: Puntuación asignada por RoBERTa a la oración *'example'*.
Fuente: *elaboración propia.*

Por la manera de RoBERTa de asignar puntuaciones es conveniente, al igual que en capítulos anteriores, reescalar el etiquetado del cuadro de datos original para la comparativa. Por ello se reescala a 0 y 1 tal y como se vió en la sección 4.2.

A continuación, para poder establecer la comparativa de forma adecuada, conviene realizar transformaciones sobre los resultados de RoBERTa. Para ello se crea una función que transforma las puntuaciones de RoBERTa en el marco de las reescaladas del cuadro de datos a través de la función *roberta_model* A. Lo que hace la función *roberta_model* de manera secuencial es:

- seleccionar cada frase de un cuadro de datos y pasarle RoBERTa.
- de las tres puntuaciones totales asiganadas por el modelo se queda únicamente con la positiva o *pos* y se compara como de cerca se queda de 0 y 1 del cuadro de datos original: si el original es 1, el score devuelve directamente el porcentaje de acierto ; si el original es 0, el score (s) es 1-s de acierto.
- con los resultados de las modificaciones anteriores se obtiene el parámetro de tasa de acierto.

5. Resultados y discusión

En este Capítulo se exponen los resultados de cada sección planteada en el Capítulo anterior y, posteriormente, una discusión de los resultados estableciendo comparativas entre metodologías y ratios de acierto.

5.1. Resultado del análisis propuesto

Con el cuadro de datos original, de una extensión de 288 oraciones por las combinaciones descritas en su respectivo capítulo 4, se separa el 80 % del cuadro de datos para el proceso de entrenamiento, lo que supone un total de 230 filas. Para el proceso de testeo se selecciona el 20 % restante del cuadro de datos, es decir, 58 filas.
Se recuerda que la dimensionalidad del cuadro de datos creado para el experimento viene determinada por las restricciones descritas tanto en el Capítulo 4 donde se detalla su construcción y restriccioness, de ahí su corta extensión en comparación con cuadros de datos utilizados para estudios del procesamiento del lenguaje natural clásicos.

Respecto a las características del entrenamiento hay que especificar los siguientes parámetros (consultar Apéndice A para mas detalles):

- **Parámetros iniciales**: los parámetros con los que inicializar el modelo para la búsqueda de parámetros óptimos del ansatz, y que se extrapolan al proceso de testeo. Se selecciona un valor único de 0.5 para todos los parámetros, es decir, un tensor de dimensión 1×10 con 0.5 como valor único.
- **Épocas**: Cantidad de iteraciones retroalimentativas del cuadro de datos. Tras establecer varias pruebas con distintas cantidades de épocas, se fijan en cinco, ya que para mayor cantidad de épocas presenta sobreajuste en el error de entrenamiento y testeo, y una menor cantidad no plasma las tendencias del cuadro de datos. Si bien no es una cantidad comparable a la de modelos semejantes es por la corta extensión del cuadro de datos.

Se ha representado tanto la tasa de acierto frente a la cantidad de épocas, así como el error correspondiente a cada época.

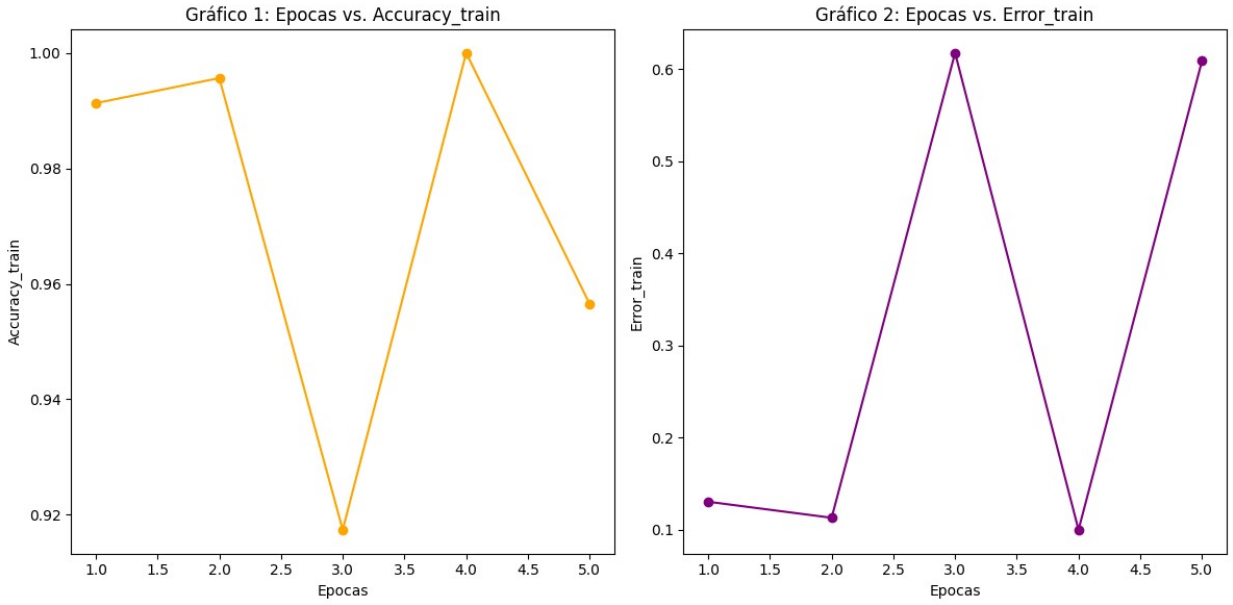


Figura 5.1: Resultados de entrenamiento según el análisis propuesto.
Fuente: *elaboración propia.*

Para la fase de testeo, se toman los parámetros óptimos de entrenamiento y se pasan al cuadro de datos de testeo de forma iterativa. Representado de la misma manera que para el proceso de entrenamiento se tiene la Figura 5.2.

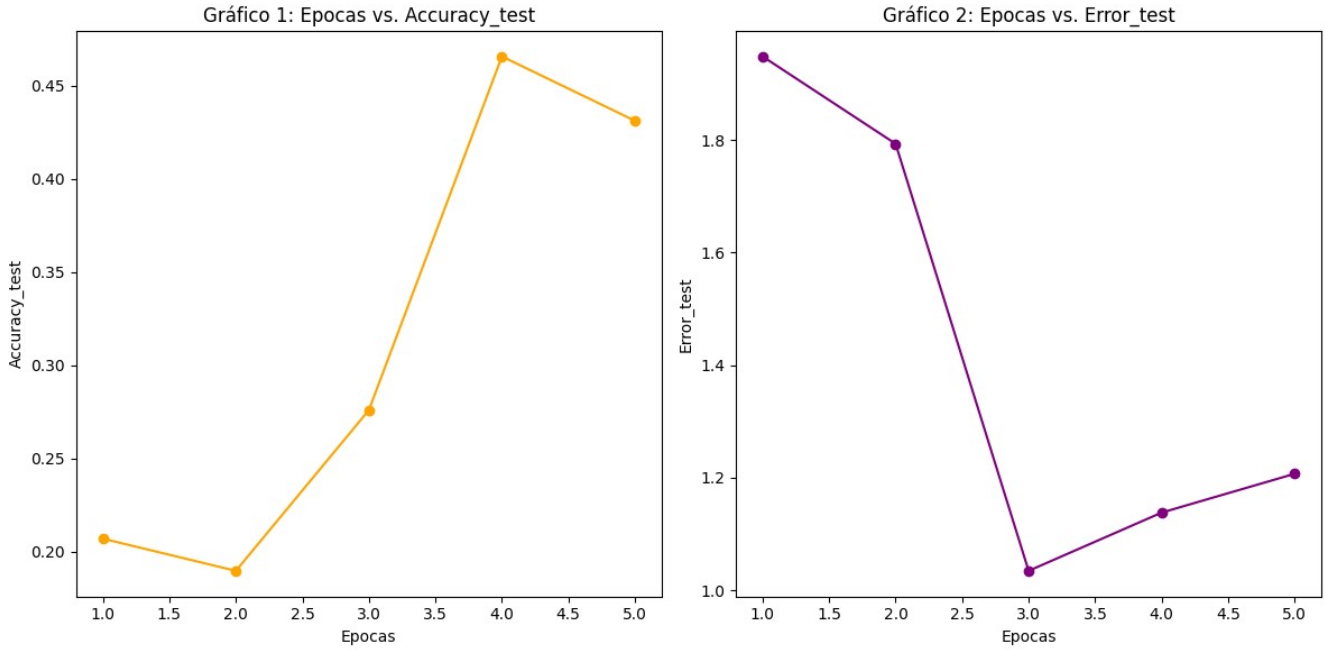


Figura 5.2: Resultados de testeo según el modelo propuesto.

Fuente: *elaboración propia.*

5.2. Discusión del análisis propuesto

Comenzando el análisis por los resultados obtenidos en la fase de entrenamiento y como se puede observar en la Figura 5.1, la tasa de acierto presenta una oscilación entre valores aproximados de [0,92, 1,00], lo que indica un ajuste casi perfecto. Si bien las oscilaciones parecen abruptas se debe a la precisión del eje de la tasa de acierto, pero no parecen ser representativas de algún otro fenómeno. Sin embargo, la cota de valores presentada para el entrenamiento sí que podría ser un indicio de sobreajuste al ceñirse casi perfectamente a lo largo de las épocas. Investigando más sobre éste fenómeno se puede apreciar que presenta un valor de $\approx 0{,}98$ en la primera época, haciendo que la hipótesis del sobreajuste debido a las épocas carezca de sentido. Sin embargo, no es excluyente de un sobreajuste por otro tipo de motivos, como puede ser un mapa de características construido en base al problema y no de forma genérica o poca variabilidad en la gramática de las oraciones implicadas en el cuadro de datos.

Respecto al diseño del mapa de características puede ser que al no haber incluido parámetros variables como sí se incluían en, por ejemplo, el *ZZFeatureMap* mencionado,

o el incluido en la aproximación de redes neuronales en el modelo propuesto por *QisKit*, *VQC*, se esté eliminando la suficiente variabilidad en el diseño de la estructura del circuito cuántico al haber fijado los parámetros que además compartían estructura lógica con la construcción del cuadro de datos como para no haber conseguido una tasa de acierto lejos del posible problema del sobreajuste.

El otro posible problema que ha podido generar estos resultados el la poca variabilidad gramatical introducida en el cuadro de datos al existir demasiada semejanza en las oraciones incluidas pero, por las limitaciones mencionadas en el Capítulo 4, no es un problema que se pueda subsanar si se quiere realizar el experimento en cuestión de una comparativa de distintas aproximaciones.

Centrando la atención sobre la Figura 5.2 pero en la gráfica derecha, es decir, la que representa el error medio a lo largo del proceso iterativo, se aprecia una concordancia con lo expuesto sobre la gráfica relacionada con la tasa de acierto para el entrenamiento, pues se recuerda que el error no está manifestado de forma relativa si no absoluta, plasmando el error medio en cada época en la función de coste. Así, un error presente de $[0{,}1, 0{,}6]$ es una proporcionalidad justificada con la poca variabilidad en la tasa de acierto, para indagar en algún comportamiento extraño que pudiera presentar (y que a priori no parece presentar). Habría que profundizar más en los posibles problemas expuestos con el proceso de entrenamiento, ya que respecto al programa creado para esta aproximación y expuesto en el Apéndice A, así como el desarrollo teórico explicado en el Capítulo 4, no parece haber fallas lógicas ni de otra índole.

La otra figura importante a analizar en esta sección es la Figura 5.2. Aquí sí se aprecia una tendencia ascendente característica de los procesos de aprendizaje automático al existir un aumento notorio en la tasa de acierto, pues el rango va desde $\approx 0{,}21$ a $\approx 0{,}45$, presentando un ascenso exponencial entre la tercera y la cuarta época. El punto a discutir aquí es si una tasa de acierto del 0,45 es suficiente para considerar que es un modelo adecuado o no. Para ello habrá que esperar a ver el resto de resultados obtenidos en las distintas aproximaciones y comparar.

El error mostrado en la Figura 5.2 respecto al proceso de testeo se aprecia una concor-

dancia afín a la tasa de acierto descrita anteriormente. Sin embargo, la discrepancia en los errores del proceso de entrenamiento y testeo (presentando una variabilidad del $\approx 300\,\%$ al alcanzar máximos de $\approx 0{,}6$ para el entrenamiento y $\approx 1{,}8$ para el testeo) vuelven a denotar un problema de sobreajuste no ligado a las épocas sino con algún bloque del algoritmo.

5.3. Redes neuronales cuánticas: resultados

En el caso de redes neuronales cuánticas, únicamente se tiene el resultado de la fase de testeo, ya que el entrenamiento se produce a nivel interno en el propio algoritmo de la función definida en el Capítulo 4.

Se recuerda que en el Capítulo4 se plantearon tres escenarios para el proceso de aprendizaje automático a través de esta aproximación:

- codificación propia desarrollada con observable simple identidad, *I*.
- codificación propia desarrollada con observable complejo compuesto por una combinación de rotaciones Y y Z, *YZ*.
- aproximación a través del modelo incluido en la librería *QisKit*, clasificador variacional cuántico, VQC.

Así las tasas de acierto para los escenarios mencionados tras las ejecuciones han sido:

- I: 0.60
- YZ: 0.36
- VQC: 0.95

5.4. Redes neuronales cuánticas: discusión

Existen dos grandes rasgos de los resultados que resultan muy llamativos: el primero es que un operador aparentemente más sencillo, *I*, lo que ofrece una menor variabilidad en el problema, presente mejores resultados que un operador más complejo, *YZ*; el otro rasgo llamativo es que el modelo predeterminado de *QisKit*, *VQC*, alcance una tasa de acierto del 0.95, lo que es un indicio de sobreajuste en el diseño de alguno de los bloques

del proceso de aprendizaje automático.

Respecto a la primera observación referente al tipo de observador con la que ejecutar el modelo, sorprende que con un observador operacional tipo identidad se consiga casi el doble de tasa de acierto que con uno más complejo, como es el operador compuesto YZ. Se puede especular que en la sección 5.2 se halla la respuesta al estar el mapa de características perfectamente definiendo el espacio de fases, con un operador identidad sería aparentemente suficiente para la clasificación del espacio, mientras que un operador con mayor grado de libertad que modifique el espacio de fases resultaría en peores resultados.

Abordando la segunda observación, puede parecer una contradicción con lo mencionado respecto al observable YZ, pero hay que recordar que en el proceso ejecutado para la obtención de ese resultado no posee la misma estructura que el proceso anterior, pues el algoritmo VQC presenta su propio mapa de características y ansatz, que además presenta 30 parámetros, lo que genera un espacio de fases mucho mas rico en cuanto a dimensionalidad se refiere. Por lo que no es de extrañar que al haber utilizado unos bloques en el proceso de aprendizaje automático mucho más complejos tanto en parámetros (10 en el análisis original frente a 50 en el algoritmo VQC) como en estructura, se consiga una mayor tasa de acierto al ser capaz de clasificar con muchos más grados de libertad.

5.5. Lambeq: resultados

Para esta aproximación hay que seleccionar una serie de parámetros para el entrenamiento y testeo, de manera similar al análisis propuesto.

- **Épocas**: similar al caso descrito en la sección de análisis original, se establece éste parámetro bajo el mismo fundamento y para una mejor comparativa en 5 épocas.
- **Batch size**: descrito ya en el Capítulo 4 como la cantidad de oraciones simultáneas para introducir al modelo, se fija en 20 debido a la dimensionalidad del cuadro de datos, ya que para esta aproximación se divide el cuadro de datos original en tres subsecciones en lugar de dos como el resto de casos al contemplar la etapa de desarrollo. Esta etapa la incluye la propia librería Lambeq para evitar problemas de ajuste. El cuadro de datos de desarrollo tiene 30 filas y el de desarrollo 28, que son

aproximadamente un 10 % del cuadro de datos original, tal y como dicta la documentación de la librería. Se establece en 20 porque el grupo de oraciones simultaneas escogidas debe ser menor que la longitud total del cuadro de datos y, si bien puede fijarse en cualquier otra cantidad menor que la extensión del 10 % mencionado, no hay estudios claros al respecto en este sentido.

- **Semilla**: Parámetro utilizado para la reproducibilidad del experimento, se fija en 2.

Tras la ejecución se presentan resultados en formato similar a los presentados por el análisis original, es decir, la tasa de acierto frente a las épocas y los errores de la función de coste frente a las épocas. Sin embargo aquí se plasman los resultados del cuadro de datos de entrenamiento y desarrollo, que ambos forman parte del proceso de entrenamiento, y posteriormente, el resultado único de la tasa de acierto para el testeo. Se plasman en la Figura 5.3.

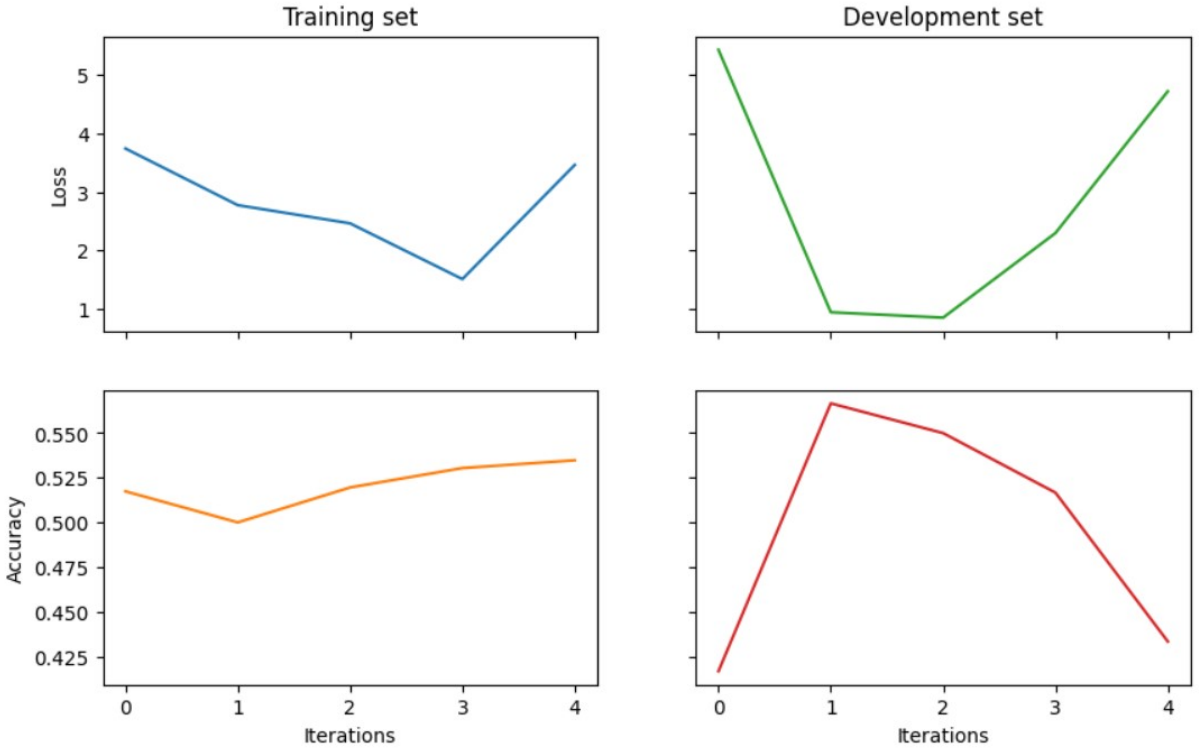


Figura 5.3: Resultados de entrenamiento de Lambeq.

Fuente: *elaboración propia.*

Para el proceso de testeo se obtiene una tasa de acierto de 0.50.

5.6. Lambeq: discusión

Fijándonos en la Figura 5.3, dividimos su análisis en dos partes: la primera referente al proceso de entrenamiento (izquierda) y la segunda referente al proceso de desarrollo (derecha). A su vez, cada parte se subdivide en error medio de la función de coste (arriba) y tasa de acierto (abajo).

En cuanto al entrenamiento, se puede apreciar un error de ≈ 4, bastante alto en comparación con procesos descritos anteriormente en la primera época, que disminuye paulatinamente hasta la cuarta época, donde se alcanza un valor de error medio de 1, para luego aumentar súbitamente en la quinta época. Dentro de los procesos analizados, un error medio de 1, habiendo visto las tasas de acierto relacionadas, podría considerarse un éxito, pero teniendo en cuenta la manera en la que se mantiene muy por encima de este a lo largo del resto de épocas, se puede concluir que no es un buen indicativo de éxito en el proceso.

La tasa de acierto presente en el entrenamiento, en contraposición al error del mismo, mantiene una tendencia constante a lo largo de las épocas con una media de 0,525. Si es cierto que se puede apreciar una correlación entre los máximos de la función error y los mínimos de la tasa de acierto para el entrenamiento, pero la experiencia observada en modelos previos manifiesta que la variabilidad en la tasa de acierto debería ser mayor.

Respecto al proceso de desarrollo, y viendo en conjunto la gráfica presentada para la función de error y la tasa de acierto, se observa una clara correlación al coincidir máximos de error con mínimos de tasa de acierto y viceversa. De nuevo esta diferencia de errores entre el bloque de entrenamiento y el de desarrollo es indicador de algún tipo de sobreajuste, como las evidencias que ya se encontraron en modelos anteriores, No obstante la tasa de acierto solo se diferencia en $\approx 20\,\%$ lo que por (FastAI, 2023) se encuentra en el límite de la presencia de sobreajuste.

Por último, en la fase de testeo se encuentra una tasa de acierto de 0.5 lo que no es discrepante con el proceso de entrenamiento, como sí lo fue para modelos anteriores.

5.7. RoBERTa: resultados

En el caso del modelo RoBERTa clásico no es necesario preestablecer ningún parámetro previo a la ejecución. En esta aproximación únicamente se divide el cuadro de datos en entrenamiento y testeo, al igual que en la aproximación original. Los resultados se plasman de manera absoluta y no por épocas, al no poder establecer dicho parámetro.
Así los resultados obtenidos han sido:

- **Entrenamiento**: 0.49
- **Testeo**: 0.42

5.8. RoBERTa: discusión

Los resultados obtenidos con el modelo clásico de RoBERTa serán tomados como referencia para determinar si los vistos anteriormente se pueden considerar como éxito o fracaso al ser un modelo infinitamente más probado y desarrollado que los modelos de computación cuántica para el procesamiento del lenguaje natural.

Es interesante notar cómo incluso el modelo clásico no consigue una tasa de acierto superior al 0.5. El resto de modelos han conseguido en la mayoría de los casos tasas superiores de acierto, a pesar de encontrarse en vías de desarrollo o de presentar muchas menos implementaciones.

5.9. Resultados generales y discusión

En esta Sección se recogen todos los resultados plasmados en el Capítulo anterior para obtener conclusiones generales. Se debe tener en cuenta que aunque todos los modelos presentan algunos parámetros en común como resultados, otros parámetros no han estado presentes en todos ellos, como el error en la fase de testeo o la tasa de acierto durante el desarrollo. Por ello se ha seleccionado los parámetros que más aparecen en común a todos los modelos, estos son la tasa de acierto durante el entrenamiento, que aparece en tres de cuatro modelos, y la tasa de acierto, en testeo que aparece en todos los modelos.

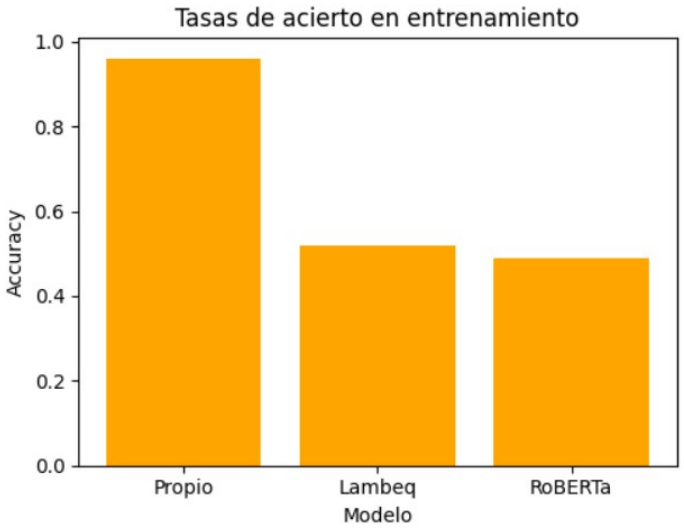


Figura 5.4: Tasa de acierto en entrenamiento.

Fuente: *elaboración propia.*

Los modelos que presentan el parámetro de tasa de acierto son: Análisis propuesto, Lambeq y RoBERTa, tal y como se aprecia en la Figura 5.4. Tomando como referencia al modelo RoBERTa como se comentó en el Capítulo anterior, se aprecia que el modelo Lambeq, con un valor de 0.52, es ligeramente mejor que su análogo clásico con 0.49, suponiendo una mejora del $\approx$ 10 %. Sin embargo, la diferencia en cuanto a capacidad de procesamiento de datos por parte de ambos modelos, pone de manifiesto si realmente merece la pena sacrificar la muestra de un procesamiento de lenguaje natural con tal de obtener tal mejora sobra la tasa de acierto.

Mientras que el caso del modelo desarrollado, es decir, 'Propio' en Figura 5.4, presenta una tasa de acierto para el entrenamiento de 0.96, lo que es considerablemente mejor que cualquier modelo mostrado. No obstante, los apuntes remarcados en la discusión sobre el diseño del modelo hace que se deban mantener tomar estos resultados con cautela.

Las tasas de acierto recogidas durante el testeo quedan reflejadas en la Figura 5.5.

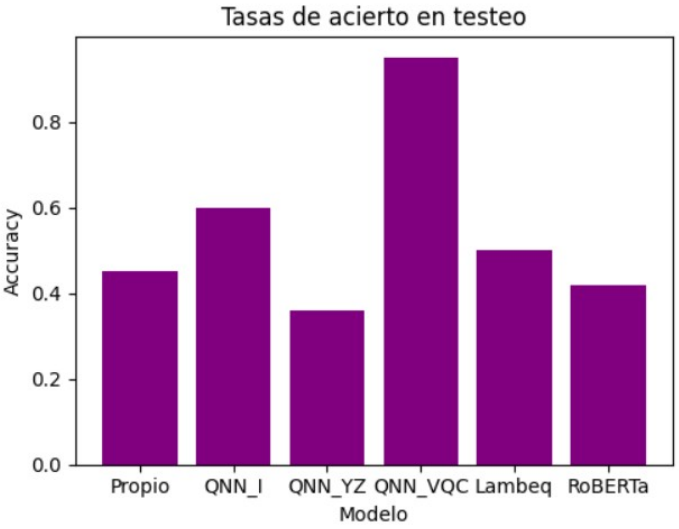


Figura 5.5: Tasa de acierto en testeo.

Fuente: *elaboración propia.*

Ahora sí, la tasa de acierto para el proceso de testeo, es un parámetro presente en todos los modelos con sus diferentes propuestas.

RoBERTa, que es el modelo de referencia, obtiene un resultado de 0.42, si bien podría considerarse un resultado heurístico al ser menor que la probabilidad de azar para clasificar entre 0 y 1, este será el que se usará para comparar el resto de modelos.
En el caso de 'Propio', se obtiene 0.45, lo que supone una mejora del $\approx 10\,\%$ sobre RoBERTa. Teniendo en cuenta que el modelo propio debía realizar una clasificación entre cuatro posibles valores mientras que RoBERTa únicamente entre dos, lo hace un éxito mayor. Ahora bien, los problemas manifestados en el proceso de entrenamiento, y como se ha manifestado anteriormente, hacen que se deban poner los resultado en cuarentena a falta de una investigación mayor en ese sentido.

Analizando los resultados mostrados para las redes neuronales cuánticas con observables I, YZ y el modelo de *QisKit* VQC, se tiene 0.60, 0.36 y 0.95 respectivamente, lo que supone un 133 %, 80 % y 211 % respecto al valor de RoBERTa.

Las razones para tales resultados han sido discutidos, pero lo que se concluye es que la clave para el análisis del procesamiento cuántico del lenguaje natural reside en la cantidad

de parámetros a incluir en el modelo, para la creación de un espacio de fases suficientemente grande.

Por último, Lambeq, presenta un 0.5 de tasa de acierto, lo que es una mejora del 20 % respecto a la referencia. Esta mejora se debe sopesar por las limitaciones plasmadas en cuanto a la extensión del cuadro de datos así como de su estructura interna. Además, aunque una mejora notoria del 20 % es sustancial, el resultado final obtenido es de 0.5 lo que ni si quiera garantiza una clasificación con garantías del cuadro de datos.

6. Conclusiones y trabajo futuro

En este Capítulo se recogen las conclusiones obtenidas del Capítulo anterior para posteriormente hacer un repaso por los objetivos e hipótesis con la intención de determinar cuáles se han cumplido y cuales refutado. Además, se plantean líneas de trabajo futuro y posibles experimentos.

6.1. Conclusiones

Por las discusiones recogidas se puede concluir respecto a los modelos de procesamiento cuántico del lenguaje natural lo siguiente:

- Un mapa de características construido en base a un cuadro de datos presenta indicios de sobreajuste. Aparentemente, descorrelacionado del proceso iterativo del aprendizaje automático. No obstante, a falta de repetir el proceso para otros cuadros de datos con mapas de características construidos bajo un procedimiento similar, no es una observación concluyente.

- Las redes neuronales cuánticas apuntan a ser la mejor aproximación para el procesamiento del lenguaje natural. Teniendo en cuenta las limitaciones impuestas y las tasas de acierto por el experimento presente desarrollado.

- La tasa de acierto en un procesamiento cuántico del lenguaje natural, apunta a depender directamente de la cantidad de parámetros presentes en el modelo de aprendizaje automático cuántico. De nuevo, al no reproducir diferentes escenarios es una observación no concluyente.

- El procesamiento cuántico del lenguaje natural solventa algunos de los problemas presentes en el procesamiento del lenguaje natural clásico, tal y como son la reproducibilidad de funciones de activación trigonométricas o búsqueda de mínimos locales en la función de optimización. No obstante, estas mejoras sobre el paradigma clásico se deben sopesar frente a las limitaciones que presenta el paradigma cuántico. La corta extensión de los mismos como la exclusión de caracteres y expresiones se encuentran entre las más reseñables.

- Por el presente análisis, se determina que, es posible el diseño de una aproximación alternativa a las redes neuronales para el procesamiento cuántico del lenguaje natural. Sin embargo, los problemas presentes en el análisis propuesto ponen de manifiesto que es una aproximación que se debe depurar y abordar con, posiblemente, otras técnicas a las empleadas para obtener resultados fehacientes y carentes de errores.

6.2. Revisión de hipótesis y objetivos

Una vez supervisadas las conclusiones obtenidas de las discusiones se repasan los objetivos y la hipótesis.

- El primer objetivo era comprender los fundamentos y procesos del procesamiento cuántico del lenguaje natural. Por las aproximaciones estudiadas de distintas índoles, y los programas desarrollados se da por cumplido este objetivo.

- El segundo objetivo era desarrollar un proceso original de QNLP. Se ha desarrollado uno propio. No obstante, los resultados parecen indicar la presencia de algún problema como se comentó en la discusión, por lo que se ha cumplido parcialmente, luego no se da por completado.

- El tercer objetivo era estudiar el problema de análisis de sentimiento desde varios paradigmas. Se han desarrollado códigos para cuatro paradigmas distintos, luego se da por cumplido.

- El cuarto objetivo era comparar resultados de los paradigmas. En el Capítulo 4 consiste precisamente en el desarrollo de éste punto.

Respecto a la hipótesis, era realizar una aproximación original capaz de ser más eficiente que el resto de aproximaciones ya desarrolladas y establecidas. A pesar de haber obtenido mejores resultados para alguna de ellas, los presentes problemas durante el entrenamiento y que no sea absolutamente superior al resto de paradigmas, deja la hipótesis lejos de confirmarse o refutarse, más bien, sienta las bases para retomar el experimento en trabajos futuros.

6.3. Trabajo futuro

Como futuras investigaciones que, desde la perspectiva experimentada, quedan por desarrollar, se proponen:

- Indagar en el problema de sobreajuste presente en el análisis original desarrollado.
- Establecer una comparativa más extensa para distintos observables en la aproximación de redes neuronales cuánticas.
- Desarrollar un análisis original con un mapa de características genérico y con parámetros en el mapa de características.
- Profundizar en la arquitectura de Lambeq con el ánimo de mejorar las limitaciones de la librería.

Bibliografía

Qiskit. (s. f.). Qiskit.org; Qiskit. Recuperado 17 de mayo de 2023, de https://qiskit.org/

Hovy. D. 2021. The Importance of Modeling Social Factors of Language: Theory and Practice. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 588–602, Online. Association for Computational Linguistics. DOI: 10.18653/v1/2021.naacl-main.49

What is artificial intelligence (AI)? (s. f.). Ibm.com. Recuperado 13 de julio de 2023, de https://www.ibm.com/topics/artificial-intelligence

Ganguly, S., Morapakula, S. N., & Bertel, L. G. A. (2022). An introduction to quantum natural language processing (QNLP). En Coded Leadership (pp. 1-23). CRC Press.

Du, S. L., Santana, S. H., Scarpa, G. (2022). A gentle introduction to Quantum Natural Language Processing. En arXiv [cs.CL]. http://arxiv.org/abs/2202.11766

Coecke, B., de Felice, G., Meichanetzidis, K., Toumi, A., Gogioso, S., & Chiappori, N. (2020). Quantum natural language processing. url: http://www.cs.ox.ac.uk/people/bob.coecke/QNLP-ACT.pdf.

Medhat, W., Hassan, A., & Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. Ain Shams Engineering Journal, 5(4), 1093-1113. https://doi.org/10.1016/j.asej.2014.04.011

Liu, Yaochen, Li, Q., Wang, B., Zhang, Y., & Song, D. (2023). A survey of quantum-cognitively inspired sentiment analysis models. ACM Computing Surveys. https://doi.org/10.1145/3604550

Preskill, J. (2018). Quantum computing in the NISQ era and beyond. En arXiv [quant-ph]. http://arxiv.org/abs/1801.00862

Liu, Yinhan, Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. En arXiv [cs.CL]. http://arxiv.org/abs/1907.11692

Rosenblatt, F. (1960). Perceptron Simulation Experiments. Proceedings of the IRE, 48(3), 301-309. https://doi.org/10.1109/jrproc.1960.287598

Duncan, B., & Zhang, Y. (2015). Neural networks for sentiment analysis on Twitter. 2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC).

Kartsaklis, D., Fan, I., Yeung, R., Pearson, A., Lorenz, R., Toumi, A., de Felice, G., Meichanetzidis, K., Clark, S., & Coecke, B. (2021). Lambeq: An efficient High-Level Python library for quantum NLP. En arXiv [cs.CL]. http://arxiv.org/abs/2110.04236

Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. Journal of Statistical Physics, 22(5), 563-591. https://doi.org/10.1007/bf01011339

Feynman, R. P. (1982). Simulating physics with computers. International Journal of Theoretical Physics, 21(6-7), 467-488. https://doi.org/10.1007/bf02650179

Shor, P. W. (2002). Algorithms for quantum computation: discrete logarithms and factoring. Proceedings 35th Annual Symposium on Foundations of Computer Science.

Divincenzo, D. P. (2005). The physical implementation of quantum computation. En Scalable Quantum Computers (pp. 1-13). Wiley-VCH Verlag GmbH & Co. KGaA.

Dirac, P. A. M. (1925). The fundamental equations of quantum mechanics. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 109(752), 642-653. https://doi.org/10.1098/rspa.1925.0150

Behtouei, M. (2023). Invariant Subspace Problem in Hilbert spaces: Exploring applications in quantum mechanics, control theory, operator algebras, functional analysis and accelerator physics. En arXiv [quant-ph]. http://arxiv.org/abs/2306.17023

Heusler, S., Schlummer, P., & Ubben, M. S. (2020). A knot theoretic extension of the Bloch sphere representation for qubits in Hilbert space and its application to contextuality and many-worlds theories. Symmetry, 12(7), 1135. https://doi.org/10.3390/sym12071135

Nielsen, M. A., & Chuang, I. L. (2012). Introduction to quantum mechanics. En Quantum Computation and Quantum Information (pp. 60-119). Cambridge University Press.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM journal of research and development, 3(3), 210-229. https://doi.org/10.1147/rd.33.0210

Zhou, Z.-H. (2021). Machine learning (S. Liu, Trad.; 1.a ed.). Springer.

Schuld, M., & Petruccione, F. (2018). Supervised Learning with Quantum Computers (Vol. 17). Springer.

Noori, M., Vedaie, S. S., Singh, I., Crawford, D., Oberoi, J. S., Sanders, B. C., & Zahedinejad, E. (2020). Analog-quantum feature mapping for machine-learning applications. Physical Review Applied, 14(3). https://doi.org/10.1103/physrevapplied.14.034034

Alonso-Linaje, G. (abril 2022). Visualizando «Redes Neuronales Cuánticas». YouTube. https://www.youtube.com/watch?v=TYgM5vJ8BX8&t=4s

Zhang, Y. (2012). Quantum computing via the Bethe ansatz. Quantum Information Processing, 11(2), 585-590. https://doi.org/10.1007/s11128-011-0268-4

Lloyd, S., Schuld, M., Ijaz, A., Izaac, J., & Killoran, N. (2020). Quantum embeddings for machine learning. En arXiv [quant-ph]. http://arxiv.org/abs/2001.03622

Ying, X. (2019). An Overview of Overfitting and its Solutions. Journal of physics. Conference series, 1168, 022022. https://doi.org/10.1088/1742-6596/1168/2/022022

Overfitting and underfitting. (s. f.). Fastaireference.com. Recuperado 25 de mayo de 2023, dehttps://www.fastaireference.com/overfitting

Ghosh, S., & Gunning, D. (2019). Natural Language Processing Fundamentals: Build intelligent applications that can interpret the human language to deliver impactful results. Packt Publishing.

Bleich, H. L. (1995). Alan Turing: the machine, the enigma, and the test. M.D. Computing: Computers in Medical Practice, 12(5), 330, 333-334.

Garvin, P. L. (1967). The Georgetown-IBM experiment of 1954: An evaluation in retrospect. En W. M. Austin (Ed.), Papers in linguistics in honor of Léon Dostert (pp. 46-56). De Gruyter.

Shen Ao. Sentiment analysis based on financial tweets and market information. In 2018 International Conference on Audio, Language and Image Processing (ICALIP), pages 321–326, 2018

Siciliani, L., & Girardi, D. (2018). The UNIBA system at the EVALITA 2018 Italian emoji prediction task. En EVALITA Evaluation of NLP and Speech Tools for Italian (pp. 129-134). Accademia University Press.

Villalpando, A., Lee J. O'Riordan, Baldois, K., Krišlauks, R.. (2021). D6.1 QNLP design and specification. https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5dc24ff50&appId=PPGMS

Meichanetzidis, K., Gogioso, S., de Felice, G., Chiappori, N., Toumi, A., & Coecke, B. (2020). Quantum natural language processing on near-term quantum computers. En arXiv [cs.CL]. http://arxiv.org/abs/2005.04147

Martinez, V., & Leroy-Meline, G. (2022). A multiclass Q-NLP sentiment analysis experiment using DisCoCat. En arXiv [cs.CL]. http://arxiv.org/abs/2209.03152

Stein, J., Christ, I., Kraus, N., Mansky, M. B., Müller, R., & Linnhoff-Popien, C. (2023). Applying QNLP to sentiment analysis in finance. En arXiv [cs.CL]. http://arxiv.org/abs/2307.11788

McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Bioph. 5, 115–133 (1943)

Macukow, B. (2016). Neural Networks – State of Art, Brief History, Basic Models and Architecture. In: Saeed, K., Homenda, W. (eds) Computer Information Systems and Industrial Management. CISIM 2016. Lecture Notes in Computer Science(), vol 9842. Springer, Cham. https://doi.org/10.1007/978-3-319-45378-1_1

Rosenblatt, F. (2021). The perceptron: A probabilistic model for information storage and organization (1958). En Ideas That Created the Future (pp. 183-190). The MIT Press.

Jia, Z.-A., Yi, B., Zhai, R., Wu, Y.-C., Guo, G.-C., & Guo, G.-P. (2018). Quantum neural network states: A brief review of methods and applications. En arXiv [quant-ph]. http://arxiv.org/abs/1808.10601

Mitarai, K. (2022). Quantum features and quantum neural network. The Brain & Neural Networks, 29(4), 202-210. https://doi.org/10.3902/jnns.29.202

Building a Varaitional Quantum Classifier. (s. f.). Q-munity. Recuperado 27 de mayo de 2023, de https://www.qmunity.tech/tutorials/building-a-variational-quantum-classifier

IBM quantum. (s. f.). IBM Quantum. Recuperado 27 de mayo de 2023, de https://quantum-computing.ibm.com/

Extensions, L. M. A. (s. f.). Visual Studio Code - code editing. Redefined. Visualstudio.com. Recuperado 30 de mayo de 2023, de https://code.visualstudio.com/

OpenAI.IntroducingChatGPT,2022. https://chat.openai.com

Wilson, D., & Sperber, D. (2015). Outline of relevance theory. HERMES - Journal of Language and Communication in Business, 3(5), 35. https://doi.org/10.7146/hjlcb.v3i5.21436

Google Finance. (s. f.). Google.com. Recuperado 18 de julio de 2023, de https://www.google.com/finance/?sa=X&ved=2ahUKEwipkoHfktqAAxWWVqQEHVDoB5cQ6M8CegQIDhAH

ZZFeatureMap - qiskit 0.44.0 documentation. (s. f.). Qiskit.org. Recuperado 10 de junio de 2023, de https://qiskit.org/documentation/stubs/qiskit.circuit.library.ZZFeatureMap.html

javafxpert.github.io. (s.f.). Representación gráfica de un cúbit. Recuperado de: https://javafxpert.github.io/grok-bloch/

RealAmplitudes - qiskit 0.44.0 documentation. (s. f.). Qiskit.org. Recuperado 15 de junio de 2023, de https://qiskit.org/documentation/stubs/qiskit.circuit.library.RealAmplitudes.html

Dallaire-Demers, J. Romero, L. Veis, S. Sim y A. Aspuru-Guzik, circuito ansatz de baja profundidad para preparar estados fermiónicos correlacionados en una computadora cuántica, Quantum Sci. Technol. 4, 045005 (2019). https://doi.org/10.1088/2058-9565/ab3951

AerSimulator — qiskit aer 0.12.2 documentation. (s. f.). Qiskit.org. Recuperado 20 de junio de 2023, de https://qiskit.org/ecosystem/aer/stubs/qiskit_aer.AerSimulator.html

COBYLA - qiskit 0.44.1 documentation. (s. f.). Qiskit.org. Recuperado 20 de junio de 2023, de https://qiskit.org/documentation/stubs/qiskit.algorithms.optimizers.COBYLA.html

SciPy. (s. f.). Scipy.org. Recuperado 20 de junio de 2023, de https://scipy.org/

Scikit-learn. (s. f.). Scikit-learn.org. Recuperado 4 de agosto de 2023, de https://scikit-learn.org/stable/

OpflowQNN — Qiskit Machine Learning 0.6.1 documentation. (s. f.). Qiskit.org. Recuperado 5 de julio de 2023, de https://qiskit.org/ecosystem/machine-learning/stubs/qiskit_machine_learning.neural_networks.OpflowQNN.html

CircuitQNN — Qiskit Machine Learning 0.6.1 documentation. (s. f.). Qiskit.org. Recuperado 7 de julio de 2023, de https://qiskit.org/ecosystem/machine-learning/stubs/qiskit_machine_learning.neural_networks.CircuitQNN.html

Bruch, S., Wang, X., Bendersky, M., & Najork, M. (2019). An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval.

Quantinuum. (s. f.). Quantinuum.com. Recuperado 11 de julio de 2023, de https://www.quantinuum.com/qai/lambeq

Discopy.Quantum.Ansatze — DisCoPy. (s. f.). Discopy.org. Recuperado 20 de julio de 2023, de https://docs.discopy.org/en/main/_modules/discopy/quantum/ansatze.html

Kulkarni, C., Rajesh, M., & Shylaja, S. S. (2022, December). Dynamic binary cross entropy: An effective and quick method for model convergence. In 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 814-818). IEEE.

Wiedmann, M., Hölle, M., Periyasamy, M., Meyer, N., Ufrecht, C., Scherer, D. D., Plinge, A., & Mutschler, C. (2023). An empirical comparison of optimizers for quantum machi-

ne learning with SPSA-based gradients. En arXiv [quant-ph]. http://arxiv.org/abs/2305.00224

Training: Quantum case — lambeq 0.3.3 documentation. (s. f.). Github.Io. Recuperado 1 de agosto de 2023, de https://cqcl.github.io/lambeq/tutorials/trainer_quantum.html

cardiffnlp/twitter-roberta-base-sentiment · Hugging Face. (s. f.). Huggingface.co. Recuperado 7 de agosto de 2023, de https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment

Hutto, C. J. (s. f.). vaderSentiment: VADER Sentiment Analysis. Recuperado 15 de agosto de 2023, de https://github.com/cjhutto/vaderSentiment.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional Transformers for language understanding. En arXiv [cs.CL]. http://arxiv.org/abs/1810.04805

Kumar, S., Jailani, N. A., Singh, A. R., & Panchal, S. (2022). Sentiment analysis on online reviews using machine learning and NLTK. 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI).

Bartz-Beielstein, T. (2023). PyTorch Hyperparameter Tuning - A Tutorial for spotPython. En arXiv [cs.LG]. http://arxiv.org/abs/2305.11930

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. En arXiv [cs.CL]. http://arxiv.org/abs/1706.03762

A. Apéndices

Teoria tras el circuito

```
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *

# qiskit-ibmq-provider has been deprecated.
# Please see the Migration Guides in https://ibm.biz/provider_migration_guide for more detail.
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler, Estimator, Session, Options

# Loading your IBM Quantum account(s)
service = QiskitRuntimeService(channel="ibm_quantum")

# Invoke a primitive. For more details see https://qiskit.org/documentation/partners/qiskit_ibm_runtime/tutorials.html
# result = Sampler("ibmq_qasm_simulator").run(circuits).result()


from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
```

Implementacion del problema

Generamos un dataframe

```
import pandas as pd
from itertools import product

# Definir las palabras para cada columna
columna1 = ["I", "we"]
columna2 = ["buy", "sell", "exchange", "use"]
columna3 = ['few', 'some', 'quite']
columna4 = ["bitcoin", "ethereum", "Google", "Apple"]
columna5 = ['yesterday', 'today', 'tomorrow']

# Generar todas las combinaciones posibles
combinaciones = list(product(columna1, columna2, columna3, columna4, columna5))

# Unir las palabras de cada combinación en una sola cadena
combinaciones = [" ".join(combinacion) for combinacion in combinaciones]

# Crear el DataFrame con una sola columna
df = pd.DataFrame(combinaciones, columns=["Text"])

# Mostrar el DataFrame
print(df)
```

```
                            Text
0     I buy few bitcoin yesterday
1         I buy few bitcoin today
2      I buy few bitcoin tomorrow
3    I buy few ethereum yesterday
4        I buy few ethereum today
..                            ...
283     we use quite Google today
284  we use quite Google tomorrow
285  we use quite Apple yesterday
286      we use quite Apple today
287   we use quite Apple tomorrow

[288 rows x 1 columns]
```

Etiquetamos el dataframe segun nuestro criterio de codificacion

```
# Recordemos que en analisis de sentimiento 0 es negativo y 1 positivo.

# ASIGNAMOS UN SCORE FECHA 18/07/2023
# Si baja en bolsa es 0 si sube es 1

# bitcoin = 1
# ethereum = 0
# google = 0
```

```
# apple = 1

# La puntuacion de los verbos esta sujeta al estado de las acciones (suma binaria):

# buy google = x 0 = 1
# buy bitcoin = x 1 = 0
# buy = 1

# sell google = x 0 = 0
# sell bitcoin = x 1 = 1
# sell = 0

# exchange google = x 0 = 0
# exchange bitcoin = x 1 = 1
# exchange = 0

# use google = x 0 = 0
# use bitcoin = x 1 = 1
# use = 0


# AJUSTAR PUNTUACIONES

# Cuanta mas gente lo haga mejor.
ss = {'I': 1, 'we': 2}

# Valores determinados por suma binaria con cds.
vv = {'buy': 1, 'sell': 0, 'exchange': 0, 'use': 0}

# Cuanto más mejor
mm = {'few': 1, 'some': 2, 'quite': 3}

# Valores variables por la bolsa, si baja 0 si sube 1.
cds ={'bitcoin': 1, 'ethereum': 0, 'Google': 0, 'Apple': 1}

# Cuanto mas tarde en el tiempo peor, mas puede variar.
tt = {'yesterday': 2, 'today': 1 , 'tomorrow': 0}


# Creamos una lista de diccionarios para el proceso iterativo:
diccionarios = [ss, vv, mm, cds, tt]


# TRABAJAR AQUI, HAY QUE ETIQUETAR BIEN LA FRASE

# EL VERBO Y EL CDS SE SUMAN BINARIAMENTE PARA DETERMINAR SI EL SUJETO Y LA CNATIDAD SUMAN O RESTAN.

# EL TIEMPO SE SUMA INDEPENDIENTEMENTE DEL RESTO.

# Minimo positivo (vv + cds = 1): 2 (1+1+0)
# Maximo positivo (vv + cds = 1): 7 (2+3+2)

# Minimo negativo (vv + cds = 0): -5 (-2-3+0)
# Maximo negativo (vv + cds = 0): 0 (-1-1+2)

# DIVIDIMOS LAS ETIQUETAS EN 4 PUES TENEMOS LAS HORQUILLAS [-5,0] y [2,7]:
# [[-5,-3],[-2,0],[2,4],[5,7]]

def etiquetar_frase(frase):

    '''
    Funcion para generar añadir labels al dataframe en base a su puntuación.

    args*
    frase: frase a labelear del dataframe
    '''

    lista_frase = frase.split()
    puntuacion_palabras = []

    for palabra, diccionario in zip(lista_frase, diccionarios):
        puntuacion_palabras.append(diccionario[palabra])



    p_ss = puntuacion_palabras[0]
    p_vv = puntuacion_palabras[1]
    p_mm = puntuacion_palabras[2]
    p_cds = puntuacion_palabras[3]
    p_tt = puntuacion_palabras[4]

    suma = 0
```

```
    if p_vv ^ p_cds == 1:
        suma = p_ss + p_mm + p_tt
    else:
        suma = -p_ss -p_mm + p_tt

    score = 0

    if -5 <= suma <= -3:
        score = 0
    if -2 <= suma <= 0:
        score = 1
    if 2 <= suma <= 4:
        score = 2
    if 5 <= suma <= 7:
        score = 3

    return score


df['Score'] = df['Text'].apply(etiquetar_frase)

# Obtener el orden de las columnas permutadas
columnas_permutadas = df.columns[::-1]

# Permutar las columnas del DataFrame
df_permutado = df.reindex(columns=columnas_permutadas)
df = df_permutado

print(df)
```

```
          Score                          Text
     0        1   I buy few bitcoin yesterday
     1        1       I buy few bitcoin today
     2        1    I buy few bitcoin tomorrow
     3        2  I buy few ethereum yesterday
     4        2      I buy few ethereum today
     ..     ...                           ...
     283      0     we use quite Google today
     284      0  we use quite Google tomorrow
     285      3  we use quite Apple yesterday
     286      3      we use quite Apple today
     287      3   we use quite Apple tomorrow

     [288 rows x 2 columns]
```

Shuffleamos y dividimos el dataframe en 2 para tener grupo Train y Test

```
# BARAJAMOS EL DATAFRAME

import pandas as pd

# Barajar las filas del DataFrame
df_barajado = df.sample(frac=1, random_state=42).reset_index(drop=True)

# Mostrar el DataFrame barajado
df = df_barajado
print(df)
```

```
          Score                              Text
     0        2         I sell few Apple yesterday
     1        0          we buy some bitcoin today
     2        1          we use few ethereum today
     3        1        I sell few Google yesterday
     4        2          we sell few bitcoin today
     ..     ...                               ...
     283      0        we sell few Google tomorrow
     284      2        I sell quite Apple tomorrow
     285      3       I exchange quite Apple today
     286      1       we use some Google yesterday
     287      1  I exchange quite Google yesterday

     [288 rows x 2 columns]
```

```
# Creamos 3 subconjuntos con el mismo dataframe: train, dev y test

# Separamos el dataframe en 3 dataframes que seran train, dev y test, con proporciones de 80%,  20%

total = df.shape[0]
var_a = int(total * 0.8)
#var_b = int(total * 0.1)
#var_c = var_a + var_b
```

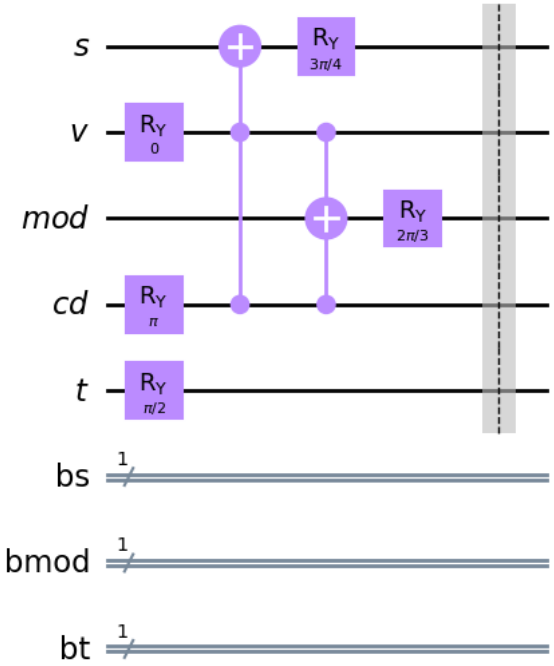
```
df_train = df[:var_a]
df_test = df[var_a:]
#df_dev = df[var_a: var_a + var_b].reset_index(drop=True)
#df_test = df[var_a + var_b: var_a + 2*var_b ].reset_index(drop=True)


print(
    df_train.shape,
    #df_dev.shape,
    df_test.shape
)
```

```
 (230, 2) (58, 2)
```

Creamos una funcion que codifique la frase en un circuito

```
from qiskit.circuit import Parameter
import numpy as np

def circuito_frase(fila):
    '''
    Funcion para codificar una frase en un circuito.
    Toma la frase spliteada, busca cada termino en un diccionario y le asigna su puntuacion al parametro en cuestion.
    Por ejemplo:
    I buy bitcoin ->
    ['I', 'buy', 'bitcoin'] ->
    I = 0 ->
    thetas = 0 ->
    cb.ry(0*(2*np.pi/6), sujeto) = cb.ry(0)
    ...

    args
    fila: fila del dataframe df.
    '''

    sujeto = QuantumRegister(1,'s')
    verbo = QuantumRegister(1,'v')
    mod = QuantumRegister(1,'mod')
    cd = QuantumRegister(1,'cd')
    tiempo = QuantumRegister(1,'t')

    cs = ClassicalRegister(1,'bs')
    #cv = ClassicalRegister(1,'bv')
    cm = ClassicalRegister(1,'bmod')
    #ccd = ClassicalRegister(1,'bcd')
    ct = ClassicalRegister(1,'bt')

    # circuito base
    cbb = QuantumCircuit(sujeto, verbo, mod, cd, tiempo, cs, cm, ct)

    frase = fila['Text']

    frase_split = frase.split()

    vp = []

    for palabra, diccionario in zip(frase_split, diccionarios):

        for clave, valor in zip(diccionario.keys(), diccionario.values()):
            if clave == palabra:
                vp.append(valor)

    # El sujeto y el mod puede ser positivo o negativo dependiendo de la suma binaria de vv y cd

    cbb.ry(vp[1]*np.pi, verbo)
    cbb.ry(vp[3]*np.pi, cd)

    cbb.ccx(verbo, cd, sujeto)
    cbb.ccx(verbo, cd, mod)

    cbb.ry(((vp[0]*np.pi/4)+np.pi/2), sujeto)
    cbb.ry(((vp[2]*np.pi/6)+np.pi/2), mod)

    cbb.ry(vp[4]*(np.pi/2), tiempo)

    cbb.barrier()

    return cbb
```

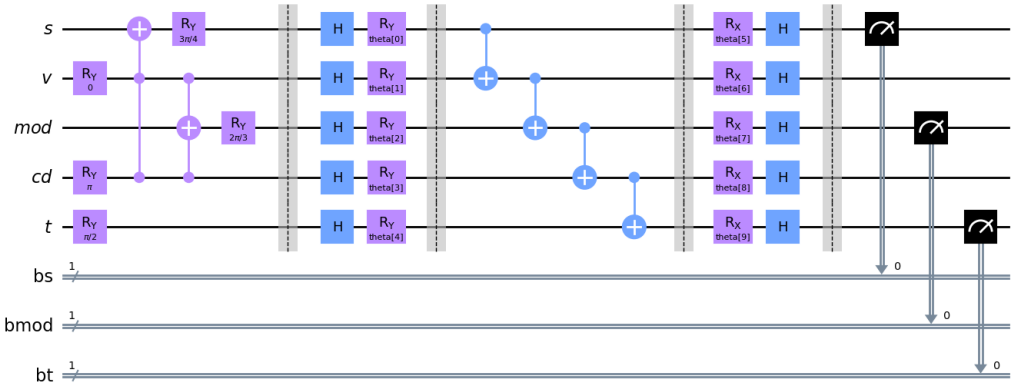
```
ejemplo = df.iloc[6]
circuito_frase(ejemplo).draw()
```



Creamos la funcion que genera el ansatz

```
from qiskit.circuit import ParameterVector

def ansatz(fila):
    '''
    Función para generar un asatz de la forma:

    H Ry º Rx H
    H Ry x Rx H

    con un parámetro en cada puerta.

    Parte de la funcion circuito_frase, lo que añade el ansatz a la codificicacion.
    '''

    cbb = circuito_frase(fila)

    params_ansatz = ParameterVector('theta', length=10)

    cbb.h([0,1,2,3,4])
    cbb.ry(params_ansatz[0], 0)
    cbb.ry(params_ansatz[1], 1)
    cbb.ry(params_ansatz[2], 2)
    cbb.ry(params_ansatz[3], 3)
    cbb.ry(params_ansatz[4], 4)
    cbb.barrier()
    cbb.cx(0,1)
    cbb.cx(1,2)
    cbb.cx(2,3)
    cbb.cx(3,4)
    cbb.barrier()
    cbb.rx(params_ansatz[5], 0)
    cbb.rx(params_ansatz[6], 1)
    cbb.rx(params_ansatz[7], 2)
    cbb.rx(params_ansatz[8], 3)
    cbb.rx(params_ansatz[9], 4)
    cbb.h([0,1,2,3,4])
    cbb.barrier()

    # Medimos todo menos el verbo y el cd
    cbb.measure(0,0)
    cbb.measure(2,1)
    cbb.measure(4,2)
```

```
    return cbb

ansatz(ejemplo).draw()
```



Creamos la funcion que pondere los valores de salida

```
from qiskit import Aer, execute

def ansatz_plus(fila):

    cbb = ansatz(fila)
    score = fila['Score']

    def funcion_coste(params):

        # Indexamos los parametros
        bound_circuit = cbb.bind_parameters(params)

        # Obtenemos las counts
        backend = Aer.get_backend('qasm_simulator')
        job = execute(bound_circuit, backend)
        counts = job.result().get_counts(bound_circuit)

        # Obtenemos el estado con mayor counts
        max_count_state = max(counts, key=counts.get) # es un string

        num_list = max_count_state.split()

        total_sum = sum(int(num) for num in num_list)

        # NUESTRA PREDICCION ES LA SUMA DE TODOS LOS ELEMENTOS DEL ESTADO CON MAS COUNTS, DEBERA APROXIMARSE TODO LO POSIBLE AL VALOR REA
        prediccion = total_sum

        error = (score - prediccion)**2

        return error

    return funcion_coste, cbb


params_iniciales = [0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]
funcion_coste_buena = ansatz_plus(df.iloc[12])[0]
resultado_final = funcion_coste_buena(params_iniciales)
print(resultado_final)
```

```
1
```

Lo metemos a un optimizador clasico COBYLA

```
#from qiskit.algorithms.optimizers import COBYLA
from scipy.optimize import minimize

def optimizacion(fila, params, tol = 0.01):

    funcion_coste_buena = ansatz_plus(fila)[0]

    result = minimize(funcion_coste_buena, params, method='COBYLA', tol= tol)

    return result


params_iniciales = [0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]
optimizacion(df.iloc[12], params_iniciales)
```

```
          fun: 1.0
        maxcv: 0.0
      message: 'Optimization terminated successfully.'
         nfev: 44
       status: 1
      success: True
            x: array([1.5       , 0.5       , 0.50597628, 0.5       , 0.5       ,
       0.49219387, 0.5       , 0.5       , 0.5       , 0.50182985])
```

Una vez que esta optimizado obtenemos el sentimiento

```
def obtener_sentimiento(fila, resultado_optimizacion):

    params_optimizados = resultado_optimizacion.x
    error_coste = resultado_optimizacion.fun

    cbb = ansatz_plus(fila)[1]

    # Indexamos los parametros
    bound_circuit = cbb.bind_parameters(params_optimizados)

    # Obtenemos las counts
    backend = Aer.get_backend('qasm_simulator')
    job = execute(bound_circuit, backend)
    counts = job.result().get_counts(bound_circuit)

    # Obtenemos el estado con mayor counts
    max_count_state = max(counts, key=counts.get) # es un string

    num_list = max_count_state.split()

    total_sum = sum(int(num) for num in num_list)

    emociones = {'0': 'sad', '1':'dissapointed', '2':'happy', '3':'elated'}
    emocion_final = emociones[str(total_sum)]

    return total_sum, emocion_final, params_optimizados, error_coste


params_iniciales = [0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]
xx = optimizacion(df.iloc[12], params_iniciales)
obtener_sentimiento(df.iloc[12], xx)
```

```
(1,
 'dissapointed',
 array([1.5 , 0.5 , 0.5 , 0.5 , 0.5 , 0.5 , 0.5 , 0.5 , 0.5 , 0.51]),
 0.0)
```

Ahora juntamos todo lo anterior en una modelo de entrenamiento

```
# TRAIN ITERATIVO

"""
def train_iterativo(df, params_ini):

    print('Comenzando entrenamiento')

    total_filas = len(df)
    filas_recorridas = 0
```

```
    paso_porcentaje = total_filas // 10

    df_entrenado = pd.DataFrame(columns = ['Score', 'Sentiment', 'Optimal_parameters'])

    accuracy_list = []

    for index, row in df.iterrows():

        # PROGRESO
        filas_recorridas += 1
        if filas_recorridas % paso_porcentaje == 0:
            porcentaje = (filas_recorridas//paso_porcentaje)*10
            print(f"Progreso: {porcentaje}%")

        frase = row['Text']

        if index == 0 :
            result_op = optimizacion(row, params_ini)
            score_final, emocion_final, parametros_optimizados = obtener_sentimiento(row, result_op)
            df_entrenado.loc[index] = [score_final, emocion_final, parametros_optimizados]
            params_entrenados = parametros_optimizados

        else:

            result_op = optimizacion(row, params_entrenados)
            score_final, emocion_final, parametros_optimizados = obtener_sentimiento(row, result_op)
            df_entrenado.loc[index] = [score_final, emocion_final, parametros_optimizados]
            params_entrenados = parametros_optimizados

        if df_entrenado.iloc[index]['Score'] == df.iloc[index]['Score'] :
            accuracy_list.append(1)

    accuracy_sum = sum(accuracy_list)
    accuracy_train = accuracy_sum/total_filas

    print('')
    print('')
    print(f'Los parametros iniciales eran: {params_ini}')
    print('')
    print(f'Los parametros optimizados son: {params_entrenados}')
    print('')
    print(f'Durante el proceso de entrenamiento se ha obtenido una eficacia de {accuracy_train}')
    print('_____________________________________________________________________________')

    return params_entrenados, accuracy_train
    """

     '\ndef train_iterativo(df, params_ini):\n    \n    print(\'Comenzando entrenamiento\')\n    \n    total_filas = len(df)\n
     filas_recorridas = 0\n    paso_porcentaje = total_filas // 10\n    \n    df_entrenado = pd.DataFrame(columns = [\'Score\',
     \'Sentiment\', \'Optimal_parameters\'])\n\n    accuracy_list = []\n    \n    for index, row in df.iterrows():\n        \n        #
     PROGRESO \n        filas_recorridas += 1\n        if filas_recorridas % paso_porcentaje == 0:\n            porcentaje =
     (filas_recorridas//paso_porcentaje)*10\n            print(f"Progreso: {porcentaje}%")\n        \n        frase = row[\'Text\']\n
     \n        if index == 0 :\n            result_op = optimizacion(row, params_ini)\n            score_final, emocion_final,
     parametros_optimizados = obtener_sentimiento(row, result_op)\n            df_entrenado.loc[index] = [score_final, emocion_final,
     parametros_optimizados]\n            params_entrenados = parametros_optimizados\n            \n        else:\n            \n
     result_op = optimizacion(row, params_entrenados)\n            score_final, emocion_final, parametros_optimizados =
     obtener_sentimiento(row, result_op)\n            df_entrenado.loc[index] = [score_final, emocion_final, parametros_optimizados]\n
     params_entrenados = parametros_optimizados\n        \n        if df_entrenado.iloc[index][\'Score\'] == df.iloc[index][\'Score\']
     :\n            accuracy_list.append(1)\n    \n    accuracy_sum = sum(accuracy_list)\n    accuracy_train =
     accuracy_sum/total_filas\n    \n    print(\'\')\n    print(\'\')\n    print(f\'Los parametros iniciales eran: {params_ini}\')\n
     print(\'\')\n    print(f\'Los parametros optimizados son: {params_entrenados}\')\n    print(\'\')\n    print(f\'Durante el proceso
     de entrenamiento se ha obtenido una eficacia de {accuracy_train}\')\n
     print(\'_____________________________________________________________________________\')\n\n    return params_entrenados,
     accuracy_train\n    '


"""
parametros = 10
params = []
for i in range(parametros):
    params.append(0.1)

train_iterativo(df_train, params)
"""

     '\nparametros = 10\nparams = []\nfor i in range(parametros):\n    params.append(0.1)\n\ntrain_iterativo(df_train, params)\n'


# TRAIN PONDERADO

def train_ponderado(df, params_ini):

    #print('Comenzando entrenamiento')
```

```
        total_filas = len(df)

        # PORCENTAJE COMENTADO
        #filas_recorridas = 0
        #paso_porcentaje = total_filas // 10

        df_entrenado = pd.DataFrame(columns = ['Score', 'Sentiment', 'Optimal_parameters', 'Error_coste'])

        parametros_buenos = []

        for index, row in df.iterrows():

            # PORCENTAJE COMENTADO
            #filas_recorridas += 1
            #if filas_recorridas % paso_porcentaje == 0:
                #porcentaje = (filas_recorridas//paso_porcentaje)*10
                #print(f"Progreso: {porcentaje}%")

            result_op = optimizacion(row, params_ini)
            score_final, emocion_final, parametros_optimizados, error_coste = obtener_sentimiento(row, result_op)
            df_entrenado.loc[index] = [score_final, emocion_final, parametros_optimizados, error_coste]

        filas_buenas = df_entrenado[(df_entrenado['Error_coste'] == 0) | (df_entrenado['Error_coste'] == 1)]

        #filas_buenas = df_entrenado[df_entrenado['Error_coste'] == 0]

        parametros_buenos = filas_buenas['Optimal_parameters'].tolist()
        params_entrenados = [(sum(elementos) / len(elementos)) for elementos in zip(*parametros_buenos)]

        accuracy_sum = len(filas_buenas)
        accuracy_train = accuracy_sum/total_filas

        error_medio = (df_entrenado['Error_coste'].sum())/total_filas

        # PORCENTAJE COMENTADO
        #print("Progreso: 100.0%")
        #print('')
        #print('')
        #print(f'Los parametros iniciales eran: {params_ini}')
        #print('')
        #print(f'Los parametros optimizados son:{params_entrenados}')
        #print('')
        #print(f'El error medio de la funcion de coste es: {error_medio}')
        #print('')
        #print(f'Durante el proceso de entrenamiento se ha obtenido una eficacia de {accuracy_train}')
        #print('_____________________________________________________________________________')

        return params_entrenados, accuracy_train, error_medio


    parametros = 10
    params = []
    for i in range(parametros):
        params.append(0.7)

    #params_ponderados = train_ponderado(df_train,params)[0]


    def modelo(df, params_ini, epoch):

        print('Comenzando entrenamiento')

        df_modelo = pd.DataFrame(columns = ['Epoch','Params_optimizados', 'Accuracy', 'Error_medio'])

        for epoca in range(1,epoch+1):

            if epoca == 1:
                params_entrenados, accuracy_train, error_medio = train_ponderado(df, params_ini)
                df_modelo.loc[epoca-1] = [epoca, params_entrenados, accuracy_train, error_medio]
            else:
                params_entrenados, accuracy_train, error_medio = train_ponderado(df, df_modelo.iloc[epoca-2,1])
                df_modelo.loc[epoca-1] = [epoca, params_entrenados, accuracy_train, error_medio]

            print(f'Epoch = {epoca}')

        print('Entrenamiento terminado')

        return df_modelo


    df_modelo = modelo(df_train, params, 5)
```

```
Comenzando entrenamiento
Epoch = 1
Epoch = 2
Epoch = 3
Epoch = 4
Epoch = 5
Entrenamiento terminado
```

```
#import os

#directorio = os.getcwd()
#directorio

#nombre_archivo = f'{directorio}df_modelo.txt'

#df_modelo.to_csv(nombre_archivo, sep = ' ')
```

Leemos y ploteamos

```
import matplotlib.pyplot as plt

# Obtener las columnas que deseas plotear
epocas = df_modelo.iloc[:, 0]
accuracy = df_modelo.iloc[:, 2]
error = df_modelo.iloc[:, 3]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.plot(epocas, accuracy, marker='o', linestyle='-', color='orange')
ax1.set_xlabel('Epocas')
ax1.set_ylabel('Accuracy_train')
ax1.set_title('Gráfico 1: Epocas vs. Accuracy_train')

ax2.plot(epocas, error, marker='o', linestyle='-', color='purple')
ax2.set_xlabel('Epocas')
ax2.set_ylabel('Error_train')
ax2.set_title('Gráfico 2: Epocas vs. Error_train')

# Ajustar los espacios entre subplots
plt.tight_layout()
plt.show()

#plt.scatter(epocas, accuracy)
#plt.xlabel('Epocas')
#plt.ylabel('Accuracy_train')
#plt.title('Gráfico de dispersión: Epoch vs. Accuracy_train')
#plt.show()
```

Gráfico 1: Epocas vs. Accuracy_train Gráfico 2: Epocas vs. Error_train

Creamos el proceso de test con los parametros mas optimos

```
def test(df_test, df_modelo, linea):

    #print('Comenzando testeo')

    # Elegimos los parametros optimos
    #indice_optimo = df_modelo['Accuracy'].idxmax()
    #params_optimos = df_modelo.iloc[indice_optimo,1]
    params_optimos = df_modelo.iloc[linea,1]


    lista_errores = []
    accuracy = 0

    for index, row in df_test.iterrows():

        # Calculamos el error para cada frase
        funcion_coste_buena = ansatz_plus(row)[0]
        error_test = funcion_coste_buena(params_optimos)
        lista_errores.append(error_test)
        #if 0 <= error_test <= 1:
        if error_test == 0 :
            accuracy += 1

    accuracy_test = accuracy/len(df_test)

    error_f_test = sum(lista_errores)/len(lista_errores)

    #print('Testeo finalizado')

    #print(f'Accuracy_testeo: {accuracy_test}')
    #print(f'Error medio de testeo:{error_f_test}')

    return accuracy_test, error_f_test



def modelo_test(df_test, df_modelo):


    epocas = range(df_modelo.shape[0])
    accuracia = []
    erroria = []
    for i in epocas:
        acc, err = test(df_test, df_modelo, i)
        accuracia.append(acc)
        erroria.append(err)

    return accuracia, erroria


accuracia, erroria = modelo_test(df_test, df_modelo)

epocass = range(1, len(df_modelo)+1)

accuracy = accuracia
error = erroria

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.plot(epocass, accuracy, marker='o', linestyle='-', color='orange')
ax1.set_xlabel('Epocas')
ax1.set_ylabel('Accuracy_test')
ax1.set_title('Gráfico 1: Epocas vs. Accuracy_test')

ax2.plot(epocass, error, marker='o', linestyle='-', color='purple')
ax2.set_xlabel('Epocas')
ax2.set_ylabel('Error_test')
ax2.set_title('Gráfico 2: Epocas vs. Error_test')

# Ajustar los espacios entre subplots
plt.tight_layout()
plt.show()
```

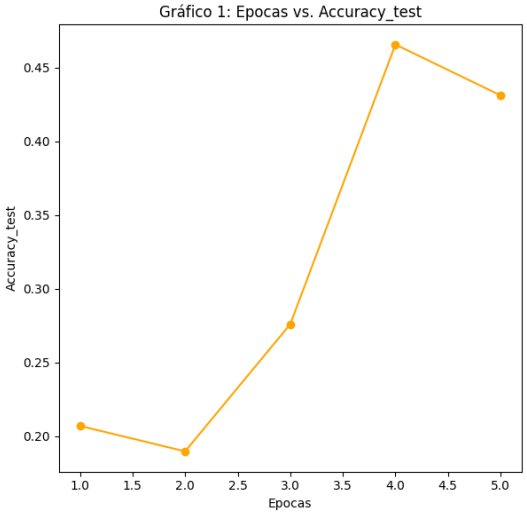


Gráfico 1: Epocas vs. Accuracy_test
Accuracy_test
0.45
0.40
0.35
0.30
0.25
0.20
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epocas

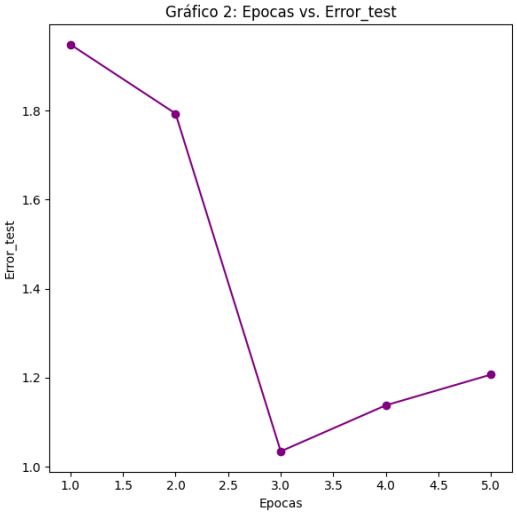


Gráfico 2: Epocas vs. Error_test
Error_test
1.8
1.6
1.4
1.2
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epocas

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import *
from sklearn.decomposition import *
from sklearn.model_selection import *

from qiskit import *
from qiskit.opflow import *
from qiskit.utils import *
from qiskit.circuit import *
from qiskit.circuit.library import *
from qiskit.opflow.primitive_ops import *
from qiskit.algorithms.optimizers import *

from qiskit_machine_learning.neural_networks import *
from qiskit_machine_learning.algorithms.classifiers import *
from qiskit_machine_learning.algorithms.regressors import *
from qiskit_machine_learning.datasets import *

from typing import Union

from qiskit_machine_learning.exceptions import QiskitMachineLearningError
```

```
/tmp/ipykernel_60/2596944044.py:10: DeprecationWarning: The ``qiskit.opflow`` module is deprecated as of qiskit-terra 0.24.0. It wi
  from qiskit.opflow import *
```

Quantum Neural Networks for QNLP sentiment analysis

```
import pandas as pd
from itertools import product

# Definir las palabras para cada columna
columna1 = ["I", "we"]
columna2 = ["buy", "sell", "exchange", "use"]
columna3 = ['few', 'some', 'quite']
columna4 = ["bitcoin", "ethereum", "Google", "Apple"]
columna5 = ['yesterday', 'today', 'tomorrow']

# Generar todas las combinaciones posibles
combinaciones = list(product(columna1, columna2, columna3, columna4, columna5))

# Unir las palabras de cada combinación en una sola cadena
combinaciones = [" ".join(combinacion) for combinacion in combinaciones]

# Crear el DataFrame con una sola columna
df = pd.DataFrame(combinaciones, columns=["Text"])


# Recordemos que en analisis de sentimiento 0 es negativo y 1 positivo.

# ASIGNAMOS UN SCORE FECHA 18/07/2023
# Si baja en bolsa es 0 si sube es 1

# bitcoin = 1
# ethereum = 0
# google = 0
# apple = 1

# La puntuacion de los verbos esta sujeta al estado de las acciones (suma binaria):

# buy google = x 0 = 1
# buy bitcoin = x 1 = 0
# buy = 1

# sell google = x 0 = 0
# sell bitcoin = x 1 = 1
# sell = 0

# exchange google = x 0 = 0
# exchange bitcoin = x 1 = 1
# exchange = 0

# use google = x 0 = 0
# use bitcoin = x 1 = 1
# use = 0
```

```
# AJUSTAR PUNTUACIONES

# Cuanta mas gente lo haga mejor.
ss = {'I': 1, 'we': 2}

# Valores determinados por suma binaria con cds.
vv = {'buy': 1, 'sell': 0, 'exchange': 0, 'use': 0}

# Cuanto más mejor
mm = {'few': 1, 'some': 2, 'quite': 3}

# Valores variables por la bolsa, si baja 0 si sube 1.
cds ={'bitcoin': 1, 'ethereum': 0, 'Google': 0, 'Apple': 1}

# Cuanto mas tarde en el tiempo peor, mas puede variar.
tt = {'yesterday': 2, 'today': 1 , 'tomorrow': 0}


# Creamos una lista de diccionarios para el proceso iterativo:
diccionarios = [ss, vv, mm, cds, tt]


# EL VERBO Y EL CDS SE SUMAN BINARIAMENTE PARA DETERMINAR SI EL SUJETO Y LA CNATIDAD SUMAN O RESTAN.

# EL TIEMPO SE SUMA INDEPENDIENTEMENTE DEL RESTO.

# Minimo positivo (vv + cds = 1): 2 (1+1+0)
# Maximo positivo (vv + cds = 1): 7 (2+3+2)

# Minimo negativo (vv + cds = 0): -5 (-2-3+0)
# Maximo negativo (vv + cds = 0): 0 (-1-1+2)

# DIVIDIMOS LAS ETIQUETAS EN 4 PUES TENEMOS LAS HORQUILLAS [-5,0] y [2,7]:
# [[-5,-3],[-2,0],[2,4],[5,7]]

def etiquetar_frase(frase):

    '''
    Funcion para generar añadir labels al dataframe en base a su puntuación.

    args*
    frase: frase a labelear del dataframe
    '''

    lista_frase = frase.split()
    puntuacion_palabras = []

    for palabra, diccionario in zip(lista_frase, diccionarios):
        puntuacion_palabras.append(diccionario[palabra])



    p_ss = puntuacion_palabras[0]
    p_vv = puntuacion_palabras[1]
    p_mm = puntuacion_palabras[2]
    p_cds = puntuacion_palabras[3]
    p_tt = puntuacion_palabras[4]

    suma = 0

    if p_vv ^ p_cds == 1:
        suma = p_ss + p_mm + p_tt
    else:
        suma = -p_ss -p_mm + p_tt

    score = 0

    if -5 <= suma <= -3:
        score = 0
    if -2 <= suma <= 0:
        score = 1
    if 2 <= suma <= 4:
        score = 2
    if 5 <= suma <= 7:
        score = 3

    return score, puntuacion_palabras


df_bueno = pd.DataFrame(columns = ['Score', 'Features'])
```

```
df_bueno['Score'] = df['Text'].apply(lambda x: etiquetar_frase(x)[0])
df_bueno['Features'] = df['Text'].apply(lambda x: etiquetar_frase(x)[1])

# Barajar las filas del DataFrame
df_barajado = df_bueno.sample(frac=1, random_state=42).reset_index(drop=True)

# Mostrar el DataFrame barajado
df_bueno = df_barajado

print(df_bueno)
```

```
     Score         Features
0        2  [1, 0, 1, 1, 2]
1        0  [2, 1, 2, 1, 1]
2        1  [2, 0, 1, 0, 1]
3        1  [1, 0, 1, 0, 2]
4        2  [2, 0, 1, 1, 1]
..     ...              ...
283      0  [2, 0, 1, 0, 0]
284      2  [1, 0, 3, 1, 0]
285      3  [1, 0, 3, 1, 1]
286      1  [2, 0, 2, 0, 2]
287      1  [1, 0, 3, 0, 2]

[288 rows x 2 columns]
```

```
def transform_score(score):
    if score in [2, 3]:
        return 1
    elif score in [0, 1]:
        return 0
    else:
        return score


df_bueno['Score'] = df_bueno['Score'].apply(transform_score)


total = df_bueno.shape[0]
var_a = int(total * 0.8)
#var_b = int(total * 0.1)
#var_c = var_a + var_b

df_train = df_bueno[:var_a]
df_test = df_bueno[var_a:]
df_test = df_test.reset_index(drop=True)


train_labels = np.array(df_train['Score'].tolist())
train_features = np.array(df_train['Features'].tolist())

test_labels = np.array(df_test['Score'].tolist())
test_features = np.array(df_test['Features'].tolist())
```

Generic OpflowQNN classification

We are going to use the generic `OpflowQNN` class to classify this data points. It is always required to define an observable and a parametrized circuit. They are the key elements of the expectation value to be measured (and the way to optimize the variables).

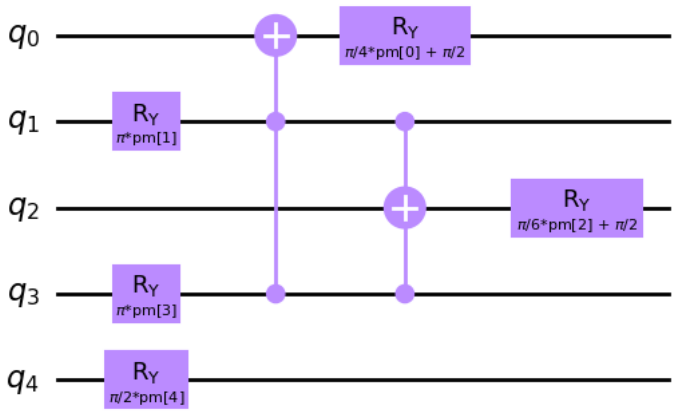
Simple observable (I)

For that purpose, we construct a parametrized circuit combining a feature map (converting classical data into quantum parametrized data), an ansatz (parametrized circuit) and a simply observable (to perform the expectation value):

Constructing the Feature Map

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit import ParameterVector
import numpy as np

mapa = QuantumCircuit(5)

p_fm = ParameterVector('pm', length=5)

mapa.ry(p_fm[1]*np.pi, 1)
mapa.ry(p_fm[3]*np.pi, 3)

mapa.ccx(1, 3, 0)
mapa.ccx(1, 3, 2)
```

```
mapa.ry(((p_fm[0]*np.pi/4)+np.pi/2), 0)
mapa.ry(((p_fm[2]*np.pi/6)+np.pi/2), 2)

mapa.ry(p_fm[4]*(np.pi/2), 4)

mapa.draw()
```



Now we can check if is a good feature map or not

To check that exist expressions depending of the problem we want to solve. In case of a quantum classifier we have:

$$\langle 0|F^{+}(x_1)F(x_2)|0\rangle \approx 0$$

For elements of the same class:

$$\langle 0|F^{+}(x_1)F(x_2)|0\rangle \approx 1$$

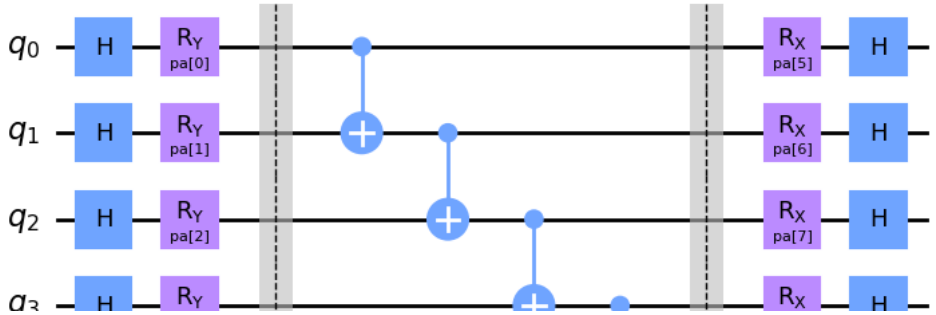
for elements of different class. In addition we can calculate the error in the following way:

$$Error(F) = -\sum_{i,j} y_i y_j \langle 0|F^{+}(x_i)F(x_j)|0\rangle$$

Thus if elements belong to different classes the error grows. Then the error is bigger as lower is the *overlap*

Constructing the ansatz

```
params_ansatz = ParameterVector('pa', length=10)

ansatz = QuantumCircuit(5)

ansatz.h([0,1,2,3,4])
ansatz.ry(params_ansatz[0], 0)
ansatz.ry(params_ansatz[1], 1)
ansatz.ry(params_ansatz[2], 2)
ansatz.ry(params_ansatz[3], 3)
ansatz.ry(params_ansatz[4], 4)
ansatz.barrier()
ansatz.cx(0,1)
ansatz.cx(1,2)
ansatz.cx(2,3)
ansatz.cx(3,4)
ansatz.barrier()
ansatz.rx(params_ansatz[5], 0)
ansatz.rx(params_ansatz[6], 1)
ansatz.rx(params_ansatz[7], 2)
ansatz.rx(params_ansatz[8], 3)
ansatz.rx(params_ansatz[9], 4)
ansatz.h([0,1,2,3,4])

ansatz.draw()
```




```
# creating the quantum circuit
dimension = 5
qcirc = QuantumCircuit(dimension)
qcirc.append(mapa, range(dimension))
qcirc.append(ansatz, range(dimension))

qcirc_op = StateFn(qcirc)
H = StateFn(PauliSumOp.from_list([('I', 1.0)]))
final_op = ~H @ qcirc_op
```

```
/tmp/ipykernel_60/3690401014.py:7: DeprecationWarning: The class ``qiskit.opflow.state_fns.circuit_state_fn.CircuitStateFn`` is dep
  qcirc_op = StateFn(qcirc)
/tmp/ipykernel_60/3690401014.py:8: DeprecationWarning: The class ``qiskit.opflow.state_fns.operator_state_fn.OperatorStateFn`` is d
  H = StateFn(PauliSumOp.from_list([('I', 1.0)]))
```

```
# selecting the method to calculate the expectation value
exp_val = AerPauliExpectation()

# selecting the gradient to use
grad = Gradient()

# defining the QuantumInstance
qi = QuantumInstance(Aer.get_backend('qasm_simulator'))

# defining the Quantum Neural Network
qnn = OpflowQNN(operator=final_op, input_params=mapa.parameters, weight_params=ansatz.parameters, exp_val=exp_val, gradient=grad, quantum
```

```
/tmp/ipykernel_60/880638602.py:2: DeprecationWarning: The class ``qiskit.opflow.expectations.aer_pauli_expectation.AerPauliExpectat
  exp_val = AerPauliExpectation()
/tmp/ipykernel_60/880638602.py:5: DeprecationWarning: The class ``qiskit.opflow.gradients.gradient.Gradient`` is deprecated as of q
  grad = Gradient()
/tmp/ipykernel_60/880638602.py:8: DeprecationWarning: The class ``qiskit.utils.quantum_instance.QuantumInstance`` is deprecated as
  qi = QuantumInstance(Aer.get_backend('qasm_simulator'))
```

```
# defining input and weights entries
qnn_input = train_features[3]
qnn_weights = np.random.rand(qnn.num_weights)

# executing the neural network (this is what fit() function does step by step)
qnn.forward(qnn_input, qnn_weights)
qnn.backward(qnn_input, qnn_weights)
```

```
(None,
 array([[[-1.11022302e-16,  0.00000000e+00, -5.55111512e-17,
           0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
           0.00000000e+00,  0.00000000e+00, -5.55111512e-17,
           0.00000000e+00]]]))
```

```
# executing the neural network classifier
qnn_classifier = NeuralNetworkClassifier(qnn, optimizer=COBYLA())
qnn_classifier.fit(train_features, train_labels)
qnn_classifier.score(test_features, test_labels)
```

```
0.603448275862069
```

YZ observables

Now, we have tested some observables to see how to improve the obtained score. Let's try with Y and Z:

```
H = StateFn(PauliSumOp.from_list([('Y', 1.0), ('Z', 1.0)]))

final_op = ~H @ qcirc_op
```

```
/tmp/ipykernel_82/3230254597.py:1: DeprecationWarning: The class ``qiskit.opflow.state_fns.operator_state_fn.OperatorStateFn`` is d
  H = StateFn(PauliSumOp.from_list([('Y', 1.0), ('Z', 1.0)]))
```

```
# defining the Quantum Neural Network
qnn = OpflowQNN(operator=final_op, input_params=mapa.parameters, weight_params=ansatz.parameters, exp_val=exp_val, gradient=grad, quantum

# executing the neural network classifier
qnn_classifier = NeuralNetworkClassifier(qnn, optimizer=COBYLA())
qnn_classifier.fit(train_features, train_labels)
qnn_classifier.score(test_features, test_labels)
```

```
0.3620689655172414
```

Specific *Variational Quantum Classifier* Qiskit function

Qiskit allows us to use an specific *Variational Quantum Classifier* to perform the same task as above. This encapsulated formula is a special variant of a `NeuralNetworkClassifier` with a `CircuitQNN`.

A `CircuitQNN` is a parametrized `QuantumCircuit` where a post-function can be applied to the obtained examples to do a binary or multi-class classification.

`VQC` uses a `CircuitQNN` with a parity formula that maps from the samples to the classification. It uses by default the *Cross Entropy Loss* function as a cost function.

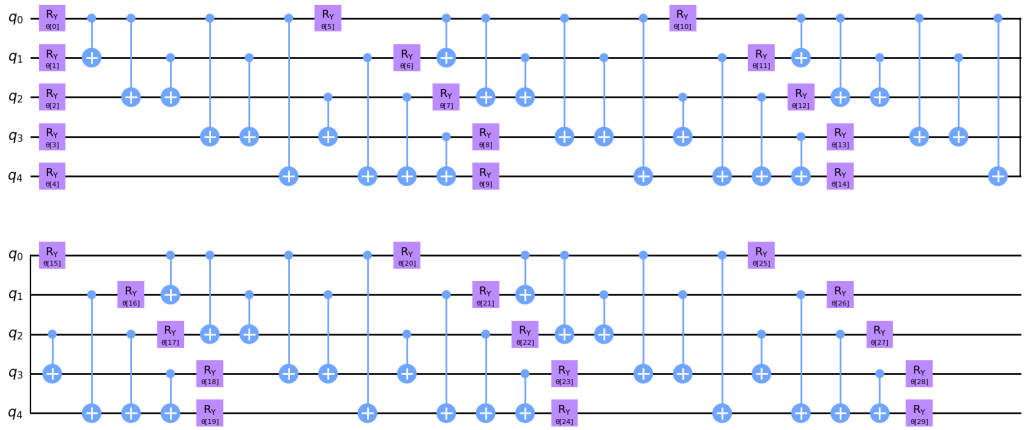
The *Variational Quantum Classifier* is an encapsulated formula, but the way to proceed follows the same approach as above: feature map, ansatz and observable:

```
num_samples = len(train_features)
y_one_hot = np.zeros((num_samples, 5))
for i in range(num_samples):
    y_one_hot[i, train_labels[i]] = 1

# preparing the QuantumInstance as per any execution
quantum_instance = QuantumInstance(Aer.get_backend('qasm_simulator'))

#preparing feature_map (convert classical data to quantum data) and ansatz (parametrized circuit)
feature_map = PauliFeatureMap(feature_dimension=dimension, reps=3, entanglement='full', alpha=1.5, paulis=['X','Y'])
ansatz = RealAmplitudes(num_qubits=dimension, entanglement='full', reps=5)
```

```
/tmp/ipykernel_82/3476833736.py:2: DeprecationWarning: The class ``qiskit.utils.quantum_instance.QuantumInstance`` is deprecated as
  quantum_instance = QuantumInstance(Aer.get_backend('qasm_simulator'))
```

```
ansatz.draw('mpl')
```


```
ansatz.decompose().draw('mpl')
```



```
# defining the Variational Quantum Classifier
vqc = VQC(feature_map=mapa, ansatz=ansatz, loss='cross_entropy', optimizer=COBYLA(), quantum_instance=quantum_instance)
```

```
/tmp/ipykernel_82/3464363397.py:2: DeprecationWarning: The quantum_instance argument is deprecated as of version 0.5.0 and will be
  vqc = VQC(feature_map=mapa, ansatz=ansatz, loss='cross_entropy', optimizer=COBYLA(), quantum_instance=quantum_instance)
```

```
# fitting the data
vqc.fit(train_features, y_one_hot)

# getting the score
vqc.score(train_features, y_one_hot)
```

```
0.9521739130434783
```

Using Lambeq model to obtain quantum results

Constructing dataframe

Due to the limitations of **lambeq** (quantum software which will be explain in detail) we can't use too complex dataset so we construct one using features relationed with finance. Our proceed consist in generate a shuffled dataframe with sentences scored with 0 and 1 due to his sentiment intention and trying to obtain results with quantum and classical processes in order to discern if there is any improvement in the quantum process.

```
import warnings
warnings.filterwarnings("ignore")

import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"


import numpy as np
import pandas as pd
from itertools import product


import pandas as pd
from itertools import product

# Definir las palabras para cada columna
columna1 = ["I", "we"]
columna2 = ["buy", "sell", "exchange", "use"]
columna3 = ['few', 'some', 'quite']
columna4 = ["bitcoin", "ethereum", "Google", "Apple"]
columna5 = ['yesterday', 'today', 'tomorrow']

# Generar todas las combinaciones posibles
combinaciones = list(product(columna1, columna2, columna3, columna4, columna5))

# Unir las palabras de cada combinación en una sola cadena
combinaciones = [" ".join(combinacion) for combinacion in combinaciones]

# Crear el DataFrame con una sola columna
df = pd.DataFrame(combinaciones, columns=["Text"])


# Cuanta mas gente lo haga mejor.
ss = {'I': 1, 'we': 2}

# Valores determinados por suma binaria con cds.
vv = {'buy': 1, 'sell': 0, 'exchange': 0, 'use': 0}

# Cuanto más mejor
mm = {'few': 1, 'some': 2, 'quite': 3}

# Valores variables por la bolsa, si baja 0 si sube 1.
cds ={'bitcoin': 1, 'ethereum': 0, 'Google': 0, 'Apple': 1}

# Cuanto mas tarde en el tiempo peor, mas puede variar.
tt = {'yesterday': 2, 'today': 1 , 'tomorrow': 0}


# Creamos una lista de diccionarios para el proceso iterativo:
diccionarios = [ss, vv, mm, cds, tt]


def etiquetar_frase(frase):

    '''
    Funcion para generar añadir labels al dataframe en base a su puntuación.

    args*
    frase: frase a labelear del dataframe
    '''

    lista_frase = frase.split()
    puntuacion_palabras = []

    for palabra, diccionario in zip(lista_frase, diccionarios):
        puntuacion_palabras.append(diccionario[palabra])
```

```
        p_ss = puntuacion_palabras[0]
        p_vv = puntuacion_palabras[1]
        p_mm = puntuacion_palabras[2]
        p_cds = puntuacion_palabras[3]
        p_tt = puntuacion_palabras[4]

        suma = 0

        if p_vv ^ p_cds == 1:
            suma = p_ss + p_mm + p_tt
        else:
            suma = -p_ss -p_mm + p_tt

        score = 0

        if -5 <= suma <= -3:
            score = 0
        if -2 <= suma <= 0:
            score = 1
        if 2 <= suma <= 4:
            score = 2
        if 5 <= suma <= 7:
            score = 3

        return score


    df['Score'] = df['Text'].apply(etiquetar_frase)

    # Obtener el orden de las columnas permutadas
    columnas_permutadas = df.columns[::-1]

    # Permutar las columnas del DataFrame
    df_permutado = df.reindex(columns=columnas_permutadas)
    df = df_permutado

    print(df)

              Score                          Text
         0        1   I buy few bitcoin yesterday
         1        1       I buy few bitcoin today
         2        1    I buy few bitcoin tomorrow
         3        2  I buy few ethereum yesterday
         4        2      I buy few ethereum today
         ..     ...                           ...
         283      0     we use quite Google today
         284      0  we use quite Google tomorrow
         285      3  we use quite Apple yesterday
         286      3      we use quite Apple today
         287      3   we use quite Apple tomorrow

         [288 rows x 2 columns]


    def transform_score(score):
        if score in [2, 3]:
            return 1
        elif score in [0, 1]:
            return 0
        else:
            return score


    df_bueno = df
    df_bueno['Score'] = df_bueno['Score'].apply(transform_score)


    total = df_bueno.shape[0]
    var_a = int(total * 0.8)
    var_b = int(total * 0.1)

    df_train = df_bueno[:var_a]
    df_test = df_bueno[var_a: var_a + var_b]
    df_dev = df_bueno[var_a+var_b:]

    df_test = df_test.reset_index(drop=True)
    df_dev = df_dev.reset_index(drop=True)

    print(df_train.shape)
    print(df_dev.shape)
    print(df_test.shape)

         (230, 2)
         (30, 2)
         (28, 2)
```

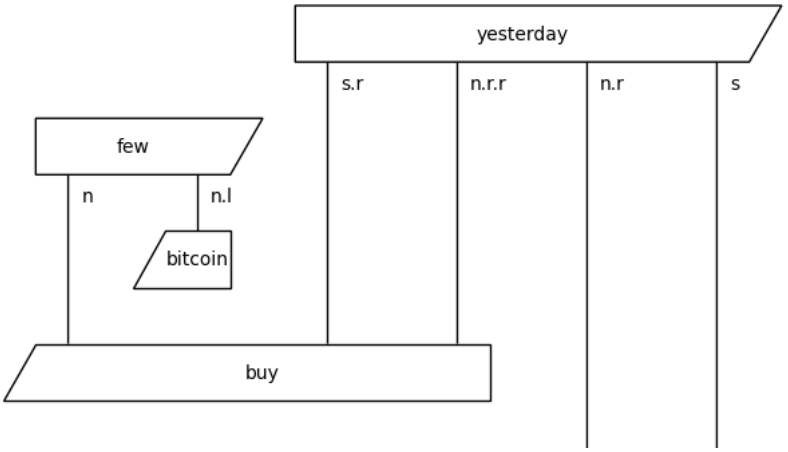
```
import csv


ruta_archivo = os.getcwd()

#ruta_archivo = 'C:/Users/bufal/Desktop/'

nombre_archivos = ['df_train', 'df_test', 'df_dev']


for name in nombre_archivos:
    nombre_archivo = f'{ruta_archivo}/{name}.txt'
    eval(name).to_csv(nombre_archivo, sep = ' ', quoting=csv.QUOTE_NONE, header=False, index=False, escapechar=' ')


# We read and save dataframes in 3 varaibles

def read_data(filename):
    labels, sentences = [], []
    with open(filename) as f:
        for line in f:
            t = int(line[0])
            labels.append([t, 1-t])
            sentences.append(line[1:].strip())
    return labels, sentences

train_labels, train_data = read_data(f'{ruta_archivo}/df_train.txt')
dev_labels, dev_data = read_data(f'{ruta_archivo}/df_dev.txt')
test_labels, test_data = read_data(f'{ruta_archivo}/df_test.txt')
```

Creating diagrams

```
# Skip the headers

TESTING = int(os.environ.get('TEST_NOTEBOOKS', '0'))

if TESTING:
    train_labels, train_data = train_labels[:2], train_data[:2]
    dev_labels, dev_data = dev_labels[:2], dev_data[:2]
    test_labels, test_data = test_labels[:2], test_data[:2]


# We turn sentences into diagrams using lambeq library

from lambeq import BobcatParser

parser = BobcatParser(verbose='text')

raw_train_diagrams = parser.sentences2diagrams(train_data)
raw_dev_diagrams = parser.sentences2diagrams(dev_data)
raw_test_diagrams = parser.sentences2diagrams(test_data)
```

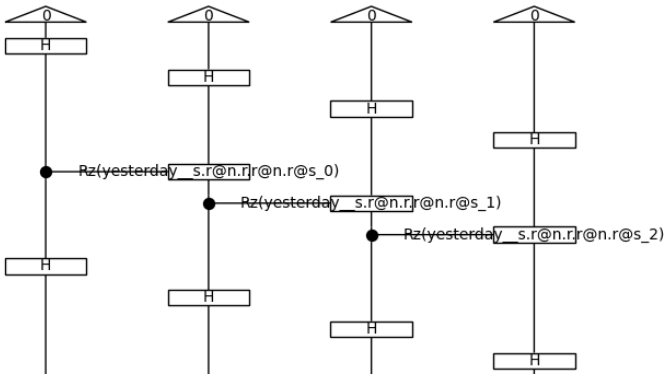
```
     Tagging sentences.
     Parsing tagged sentences.
     Turning parse trees to diagrams.
     Tagging sentences.
     Parsing tagged sentences.
     Turning parse trees to diagrams.
     Tagging sentences.
     Parsing tagged sentences.
     Turning parse trees to diagrams.
```

```
# As we will see is important to remove cups to reduce the depth of circuits

from lambeq import remove_cups

train_diagrams = [remove_cups(diagram) for diagram in raw_train_diagrams]
dev_diagrams = [remove_cups(diagram) for diagram in raw_dev_diagrams]
test_diagrams = [remove_cups(diagram) for diagram in raw_test_diagrams]

train_diagrams[0].draw()
```



Creating circuits

```
# Now turn diagrams into quantum circuits

from lambeq import AtomicType, IQPAnsatz

ansatz = IQPAnsatz({AtomicType.NOUN: 1, AtomicType.SENTENCE: 1},
                   n_layers=1, n_single_qubit_params=3)

train_circuits = [ansatz(diagram) for diagram in train_diagrams]
dev_circuits =  [ansatz(diagram) for diagram in dev_diagrams]
test_circuits = [ansatz(diagram) for diagram in test_diagrams]

train_circuits[0].draw(figsize=(9, 12))
```



Setting quantum model

```
# Setting backend config

from pytket.extensions.qiskit import AerBackend
from lambeq import TketModel

all_circuits = train_circuits+dev_circuits+test_circuits

backend = AerBackend()
backend_config = {
    'backend': backend,
    'compilation': backend.default_compilation_pass(2),
    'shots': 8192
}
model = TketModel.from_diagrams(all_circuits, backend_config=backend_config)
```

```
from lambeq import BinaryCrossEntropyLoss

# Using the builtin binary cross-entropy error from lambeq
bce = BinaryCrossEntropyLoss()

acc = lambda y_hat, y: np.sum(np.round(y_hat) == y) / len(y) / 2  # half due to double-counting
```

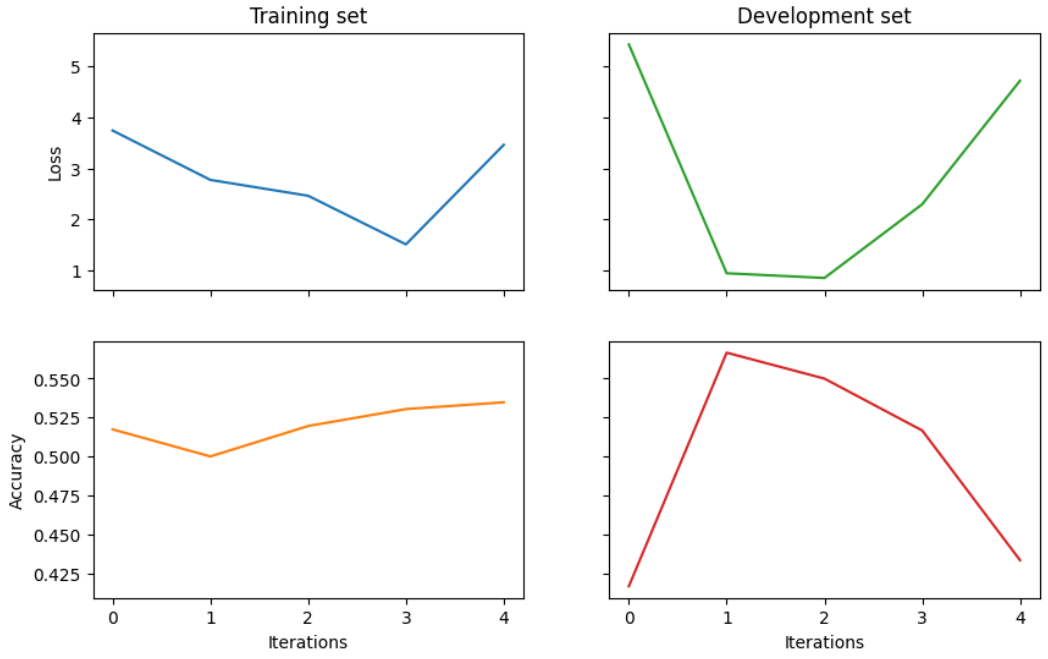
```
# FEATURES OF QUANTUM PROCESS

BATCH_SIZE =  20 # Due to dimensionality of dev and test dataframes we select 6 as batch size
EPOCHS = 5
SEED = 2


# Select the trainer and optimizer

from lambeq import QuantumTrainer, SPSAOptimizer

trainer = QuantumTrainer(
    model,
    loss_function=bce,
    epochs=EPOCHS,
    optimizer=SPSAOptimizer,
    optim_hyperparams={'a': 0.05, 'c': 0.06, 'A':0.01*EPOCHS},
    evaluate_functions={'acc': acc},
    evaluate_on_train=True,
    verbose = 'text',
    seed= SEED
)


from lambeq import Dataset

train_dataset = Dataset(
            train_circuits,
            train_labels,
            batch_size=BATCH_SIZE)

val_dataset = Dataset(dev_circuits, dev_labels, shuffle=False)


trainer.fit(train_dataset, val_dataset, logging_step=2)
```

```
Epoch 1:  train/loss: 3.7425   valid/loss: 5.4355   train/acc: 0.5174   valid/acc: 0.4167
Epoch 2:  train/loss: 2.7715   valid/loss: 0.9353   train/acc: 0.5000   valid/acc: 0.5667
Epoch 4:  train/loss: 1.5038   valid/loss: 2.2930   train/acc: 0.5304   valid/acc: 0.5167

Training completed!
```

Quantum Results

```
import matplotlib.pyplot as plt

fig, ((ax_tl, ax_tr), (ax_bl, ax_br)) = plt.subplots(2, 2, sharex=True, sharey='row', figsize=(10, 6))
ax_tl.set_title('Training set')
ax_tr.set_title('Development set')
ax_bl.set_xlabel('Iterations')
ax_br.set_xlabel('Iterations')
ax_bl.set_ylabel('Accuracy')
ax_tl.set_ylabel('Loss')

colours = iter(plt.rcParams['axes.prop_cycle'].by_key()['color'])
ax_tl.plot(trainer.train_epoch_costs, color=next(colours))
ax_bl.plot(trainer.train_results['acc'], color=next(colours))
ax_tr.plot(trainer.val_costs, color=next(colours))
ax_br.plot(trainer.val_results['acc'], color=next(colours))

test_acc = acc(model(test_circuits), test_labels)
print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.5
```



Using ROBERTA model to obtain classical results

Tokenizing data

```
import nltk
nltk.download('punkt')
example = train_data[0]
tokens = nltk.word_tokenize(example)
tokens[:]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\bufal\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
['I', 'buy', 'few', 'bitcoin', 'yesterday']
```

```
nltk.download('averaged_perceptron_tagger')
tagged = nltk.pos_tag(tokens)
tagged[:]
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\bufal\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[('I', 'PRP'),
 ('buy', 'VBP'),
 ('few', 'JJ'),
 ('bitcoin', 'NN'),
 ('yesterday', 'NN')]
```

```
nltk.download('maxent_ne_chunker')
nltk.download('words')
entities = nltk.chunk.ne_chunk(tagged)
entities.pprint()
```

```
(S I/PRP buy/VBP few/JJ bitcoin/NN yesterday/NN)
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\bufal\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\bufal\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
```

```
from transformers import AutoTokenizer

from transformers import AutoModelForSequenceClassification

from scipy.special import softmax
```

Importing ROBERTA model

```
# We make sure that there is no directory with the same name to avoid problems
# This step is really important !

import shutil
import os

MODEL = "cardiffnlp/twitter-roberta-base-sentiment"
if os.path.isdir(MODEL):
    shutil.rmtree(MODEL)
    print(f"El directorio '{MODEL}' ha sido eliminado correctamente.")
else:
    print(f"No se encontró un directorio con el nombre '{MODEL}'.")
```

```
El directorio 'cardiffnlp/twitter-roberta-base-sentiment' ha sido eliminado correctamente.
```

```
import torch

MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
model.save_pretrained(MODEL)
```

```
# Run for Roberta Model
encoded_text = tokenizer(example, return_tensors='pt')
output = model(**encoded_text)
scores = output[0][0].detach().numpy()
scores = softmax(scores)
scores_dict = {
    'roberta_neg': scores[0],
    'roberta_neu': scores[1],
    'roberta_pos': scores[2]
}

print(scores_dict)
```

```
{'roberta_neg': 0.021732017, 'roberta_neu': 0.7869983, 'roberta_pos': 0.19126976}
```

Create a function to apply and obtain results from all data

```
# Seleccionar cada frase de un dataframe y pasarle roberta
# solo nos quedamos con pos y comparamos como de cerca se queda de 0 y 1 del dataframe original:
# si el original es 1, el score devuelve directamente el porcentaje de acierto.
# si el original es 0, el score (s) es 1-s de acierto.
# con eso obtenemos el accuracy.

def roberta_model(df):
```

```
    accuracy = 0

    for puntuacion, frase in zip(df['Score'], df['Text']):
        enc_text = tokenizer(frase, return_tensors='pt')
        out = model(**enc_text)
        score = out[0][0].detach().numpy()
        score = softmax(score)
        pos = score[2]

        if puntuacion == 1:
            accuracy += pos
        else:
            accuracy += 1-pos

    final_score = accuracy/ df.shape[0]

    return final_score

print('For df_train:',roberta_model(df_train))

print('For df_dev:',roberta_model(df_dev))
```

```
 For df_train: 0.49875032368885436
 For df_dev: 0.4920505913595358
```