



Universidad de Valladolid

Escuela de Ingeniería Informática de Valladolid

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención Ingeniería del Software

Modelo especialista en inversión financiera mediante NLP y RAG

Autor:

D. David Fernández Hermoso

Tutores:

D. Benjamín Sahelices Fernández

D. Fernando Adolfo Tejerina Gaite

Dedicatoria

Quiero agradecer a las personas que han hecho posible que nunca tire la toalla y pese a cualquier adversidad no me rinda nunca, en primer lugar, a mis padres:

Juan María y Azucena, que me han ofrecido su apoyo incondicional y han hecho que siempre muestre la mejor versión de mí mismo. Sin vosotros no hubiese llegado hasta donde estoy hoy.

A mi hermano Juan, mi referente en la vida y un segundo padre para mí, gracias por apoyarme en momentos duros y pase lo que pase, siempre estar ahí para mí.

A mi abuela Tina, la cual nos dejó hace unos años y no ha podido ver como su nieto “se hacía mayor” como ella decía. Gracias por enseñarme a esforzarme por las cosas que me gustan y me importan, siempre te llevaré en el corazón.

A todos y cada uno de vosotros, os doy las gracias de corazón por servirme vuestro apoyo y vuestra enseñanza.

Resumen

El trabajo desarrollado para este proyecto consta de la elaboración de un modelo especialista en inversión financiera usando métodos de NLP y RAG. Se tienen como referencia 21 cartas redactadas por el TCI hacia las compañías de las cuales es inversor activo y el objetivo es que el modelo desarrollado sea capaz de responder consultas propuestas por el usuario a cerca de los temas recogidos en dichas cartas. Este modelo podría ser la base un proyecto mayor en el que el ámbito de estudio sea mucho más amplio.

Abstract

The work developed for this Project involves creating a specialized financial investment model using NLP and RAG methods. 21 letters written by TCI to companies in which is it an active investor are used as reference, and the goal is for the developed model to be able to respond to user queries regarding the topics covered in these letters. This model could be the basis for a larger Project with a much broader scope of study.

Índice

Capítulo 1 – Introducción	12
1.1 Objetivos	12
Capítulo 2 – Contexto	13
2.1 Descripción del problema financiero	13
2.2 Introducción a las Redes neuronales	13
2.2.1 Concepto y evolución de las Redes Neuronales	14
2.2.2 Arquitectura de las Redes Neuronales	15
2.2.3 Entrenamiento de Redes Neuronales	20
2.2.4 Tipos principales de redes neuronales	30
2.3 Modelos de Lenguaje	34
2.3.1 Tipos de modelos de lenguaje	34
2.3.2 Entrenamiento de modelos de lenguaje	35
2.4 RAG (<i>Retrieval-Augmented Generation</i>).....	41
2.4.1 Componentes principales	41
2.4.2 Modelos de RAG	42
2.5 Tecnología utilizada	43
2.5.1 Anaconda	43
2.5.2 Python.....	43
2.5.3 CUDA (Compute Unified Device Architecture)	43
2.5.4 PyTorch	44
2.5.5 Jupyterlab	44
2.5.6 Notebook	44
2.5.7 Tensorboard	44
Capítulo 3 – Metodología	45
3.1 LLM Utilizados	46
3.2 Preparación de los datos para el entrenamiento	49
3.3 Aplicación de RAG	50
Capítulo 4 – Experimentación y resultados.....	52
4.1 Implementación de redes neuronales y modelo base	52

4.1.1 Clasificación de imágenes con una red FNN	52
4.1.2 Clasificación de imágenes con una red CNN	56
4.1.3 <i>Fine-tuning</i> de un LLM basado en Transformers	58
4.2 RAG	62
4.2.1 Modelo previo de RAG sin <i>fine-tuning</i>	62
4.2.2 Modelo completo (RAG + <i>Fine-tuning</i>)	65
4.3 Experimentos	71
4.4 Resultados	78
Capítulo 5 – Conclusiones	81
Bibliografía y Webgrafía	83
Anexos	89
Anexo A – Manual de instalación del entorno de estudio	91
Anexo B – Archivos adjuntos	95
Anexo C – Preguntas recogidas en las diferentes pruebas	98

Lista de figuras

Figura 1: Modelo del perceptrón	14
Figura 2: Comparación entre una neurona biológica y un perceptrón [5]	14
Figura 3: Adición del sesgo al perceptrón.....	14
Figura 4: Ecuación de la sigmoide	15
Figura 5: Entrada ponderada de una neurona.....	15
Figura 6: Función sigmoide.....	16
Figura 7: Ecuación de la tangente hiperbólica	16
Figura 8: Función tangente hiperbólica.....	17
Figura 9: Ecuación de ReLU	17
Figura 10: Función ReLU	17
Figura 11: Ecuación de Leaky ReLU y Parametric ReLU.....	18
Figura 12: Función Leaky ReLU	18
Figura 13: Función Parametric ReLU.....	18
Figura 14: Ecuación de SiLU.....	19
Figura 15: Función SiLU	19
Figura 16: Ecuación aproximada de GeLU.....	19
Figura 17: Función GeLU	20
Figura 18: Función cuadrática de coste	20
Figura 19: Función de entropía cruzada.....	21
Figura 20: Cálculo de la entrada ponderada.....	23
Figura 21: Ecuación de error en la capa de salida	23
Figura 22: Ecuación de retropropagación del error	23
Figura 23: Ecuación de la derivada del coste respecto a los sesgos.....	23
Figura 24: Ecuación de la derivada del coste respecto a los pesos	23
Figura 25: Regla de actualización en DG.....	24
Figura 26: Regla de actualización en SGD.....	25
Figura 27: Cálculo del momento.....	25
Figura 28: Cálculo del RMSprop.....	25
Figura 29: Inicialización del momento	26
Figura 30: Inicialización de RMSprop	26
Figura 31: Fórmula de Adam	26
Figura 32: Fórmula de AdamW.....	26
Figura 33: Fórmula de la convolución.....	30
Figura 34: Arquitectura de los Transformers [19]	33
Figura 35: Prompt zero-shot learning	39
Figura 36: Prompt one-shot learning	39
Figura 37: Prompt few-shot learning	40
Figura 38: Sistema RAG en el que se usa DPR y BART [30]	41
Figura 39: Probabilidad calculada por el retriever en RAG.....	42
Figura 40: Probabilidad calculada por el generador en RAG.....	42

Figura 41: RAG-Sequence	43
Figura 42: RAG-Token	43
Figura 43: Ranking Embedding Leaderboard	46
Figura 44: config.json de multilingual-e5-large-instruct	47
Figura 45: config.json de TinyLlama-1.1B-Chat-v1.0	48
Figura 46: Cálculo de los top-k documentos	51
Figura 47: Eval_loss 3 tasas	73
Figura 48: Train_loss 3 tasas	73
Figura 49: Eval_loss para las 3 pruebas con Optuna	77
Figura 50: Train_loss para las 3 pruebas con Optuna	77
Figura 51: Tasa de aprendizaje para las 3 pruebas con Optuna	77
Figura 52: Eval_loss del modelo final	78
Figura 53: Train_loss del modelo final	78
Figura 54: Tasa de aprendizaje del modelo final	79

Capítulo 1 – Introducción

Dentro de este trabajo se explicará cómo crear un modelo especialista en inversión financiera usando técnicas de aprendizaje automático y de procesamiento de lenguaje natural (NLP). Para ello se explican conceptos clave como las redes neuronales, los modelos de lenguaje y el mecanismo de *Retrieval-Augmented Generation* (RAG).

Concretamente se estudiará un caso de uso relacionado con *The Children's Investment (TCI) Fund Management*, una empresa gestora de fondos de cobertura hacia otras compañías fundada en 2003 por Chris Hohn. TCI exige a las compañías de las cuales son accionistas ciertos cambios relacionados con políticas medioambientales, como la reducción de emisiones de gases de efecto invernadero, la deforestación, etc. Estos cambios los solicita a través de cartas.

El modelo desarrollado, por tanto, tiene como objetivo extraer información precisa de dichas cartas, teniendo que ser capaz de responder a las motivaciones y requerimientos concretos que precisa TCI sobre las compañías para ayudar a entender el fin del comportamiento de los fondos de inversión.

1.1 Objetivos

Aunque en el caso que se desarrolla en este documento solamente hay 21 cartas de ejemplo, que, a priori, se podrían estudiar manualmente sin necesidad de una herramienta, el propósito principal es poder llegar a desarrollar una fase beta de un modelo especialista en inversión financiera el cual tiene como objetivo poder resolver dudas sobre documentos técnicos pertenecientes a este campo. Para poder cumplir este objetivo se establecen una serie de pasos a seguir:

1. Obtención del dataset de entrenamiento
2. Aprender a manejar LLMs
3. *Fine-tuning*
4. Aplicación de RAG
5. Evaluación del sistema RAG creado

Capítulo 2 – Contexto

A lo largo de este capítulo se explican los conceptos básicos sobre los que se desarrolla el trabajo, tales como las redes neuronales, los modelos de lenguaje o el método RAG, entre otros. En cada apartado se explicarán los fundamentos en los que se basan, sus características, etc.

2.1 Descripción del problema financiero

En los últimos años ha aumentado la presencia de inversores institucionales (fondos de inversión, fondos de cobertura, fondos de pensiones, fondos soberanos, etc.) en el capital de las empresas. Alrededor del 70% de empresas cotizadas en USA tienen como accionistas a estos inversores, los tres gestores de fondos por excelencia (*BlackRock*, *Vanguard* y *State Street*) controlan más del 25% de las acciones del S&P 500 (*Standard & Poor's 500 Index*), uno de los índices bursátiles principales presentes en la bolsa de Nueva York, y de las bolsas pertenecientes a la comunidad europea. En España, estos inversores tienen presencia en 77 empresas (dato extraído de 2023), y en 23 de ellas con más de un 5% [1].

Normalmente los inversores institucionales no son los principales accionistas de las empresas, pero no quiere decir que no sean relevantes, especialmente en los últimos años donde han adquirido un papel protagonista en la toma de decisiones y en las políticas que rigen a las empresas. En el caso que compete a este trabajo, TCI (fondo de cobertura), plantea cambios y propuestas a las empresas en las que invierte el capital, relacionados con las políticas medioambientales a través de cartas, instando a estas a aceptarlas ya que de lo contrario amenazan con votar en contra y presentar resoluciones de desaprobación en las juntas de accionistas, llegando incluso a evaluar la posible desvinculación (y, por tanto, la retirada de capital) con la empresa. TCI sostiene que, sin dichos cambios, la rentabilidad y sostenibilidad de las empresas decaerán a largo plazo, lo que principalmente se traduce en pérdidas monetarias para ellos [2].

2.2 Introducción a las Redes neuronales

Durante todo este apartado se usa en gran parte la referencia [3].

Las redes neuronales artificiales (RNA), son modelos de aprendizaje automático que simulan el comportamiento de un cerebro humano, inspirado en la estructura de este. Las RNA están compuestas por unidades llamadas neuronas artificiales, las cuales procesan información imitando el comportamiento de las neuronas biológicas. El objetivo principal de estas redes es reconocer patrones, tomar decisiones basadas en los datos de entrada y llegar a conclusiones [4].

Las redes neuronales han revolucionado el campo del aprendizaje automático gracias a la capacidad de modelar relaciones complejas, aprender grandes volúmenes de datos y generalizar a nuevos ejemplos. Se utilizan en una amplia variedad de aplicaciones como reconocimiento de imágenes, procesamiento de lenguaje natural, diagnósticos médicos, etc.

2.2.1 Concepto y evolución de las Redes Neuronales

El desarrollo de las redes neuronales artificiales tiene sus raíces en torno a los años 50 y 60 cuando Frank Rosenblatt desarrolló el perceptrón, un modelo matemático inspirado en las neuronas biológicas. El modelo recibe unos datos de entrada, cada uno con su peso correspondiente, y al realizar la suma ponderada se obtiene una salida binaria (0 o 1):

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq threshold \\ 1 & \text{if } \sum_j w_j x_j > threshold \end{cases}$$

Figura 1: Modelo del perceptrón

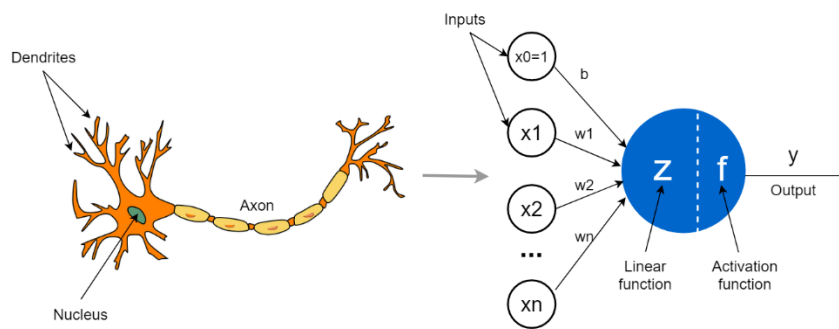


Figura 2: Comparación entre una neurona biológica y un perceptrón [5]

Inicialmente, los perceptrones podían simular puertas lógicas como NAND u OR, lo que permitía realizar operaciones básicas de clasificación. Sin embargo, tenían una limitación importante, no eran capaces de resolver problemas no lineales (un ejemplo de ello sería la puerta XOR). Esto derivó al desarrollo de perceptrones multicapa (MLP) que introdujeron capas ocultas y permitieron que las redes neuronales aprendiesen relaciones más complejas entre los datos.

Como podemos ver en la “Figura 1” se utiliza un umbral fijo lo cual puede llegar a ser poco productivo a la hora de crear algo más complejo. Por ello, esta función evolucionó y se introdujo el concepto de sesgo (bias), permitiendo una mayor flexibilidad y que se pueda ajustar la salida de la neurona sin depender exclusivamente de las entradas. Así queda la nueva ecuación:

$$output = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Figura 3: Adición del sesgo al perceptrón

Aun con esta modificación el perceptrón seguía siendo una capa muy básica, si se desean resolver problemas más complejos se han de utilizar algoritmos de aprendizaje automático lo cual permite que se resuelvan estos problemas sin intervención del programador. La idea es que nuestra red de perceptrones sea capaz de aprender los pesos y los sesgos, para ello se podrían introducir pequeños cambios para ajustar las salidas, dichos cambios harían posible el aprendizaje, pero cuando se ajusta una salida se producen cambios en otras salidas y es algo realmente difícil de controlar.

Todo este problema se puede resolver introduciendo lo que se llaman neuronas sigmoideas (también llamadas logísticas), son similares a los perceptrones, pero un pequeño cambio en los pesos o sesgos de estas solamente causan un pequeño cambio en las salidas. Como diferencia con los perceptrones, las neuronas generan una salida continua entre $[0,1]$ en lugar de los valores estrictamente binarios. La función de activación sigmoide se define como:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Figura 4: Ecuación de la sigmoide

Donde z es la entrada ponderada de la neurona, calculada como:

$$z \equiv w \cdot x + b$$

Figura 5: Entrada ponderada de una neurona

Donde:

- **w:** son los pesos de la red.
- **x:** es la entrada.
- **b:** es el sesgo.

2.2.2 Arquitectura de las Redes Neuronales

Las MLP siguen la siguiente estructura: [6]

1. **Capa de entrada (*Input layer*):** Esta capa se encarga de recibir los datos iniciales y se pasan a la siguiente capa sin aplicar ninguna transformación sobre ellos.
2. **Capas ocultas (*Hidden layers*):** Pueden existir una o más capas ocultas. Procesan la información detectando patrones más complejos, cada neurona aplica una transformación a la información recibida de la capa anterior.
3. **Capa de salida (*Output layer*):** Genera la predicción o el resultado final. El número de neuronas que hay en esta capa normalmente se elige según la tarea a realizar (Por ejemplo, si queremos hacer una clasificación binaria, solamente habrá 2 neuronas).
4. **Pesos (*Weights*):** Una neurona puede estar conectada con una o más neuronas de las capas adyacentes, cada conexión tiene un peso asociado.
5. **Neuronas de sesgo (*Bias Neurons*):** Son la implementación del término de sesgo ya definido previamente. Estas neuronas se pueden incluir en cualquier capa salvo en la de entrada.

El diseño de las capas ocultas determina la capacidad de la red para representar patrones complejos. Sin embargo, si una red posee demasiadas capas o neuronas puede volverse ineficiente o producirse un

¹ Desde ahora en adelante, siempre que aparezca “ z ” en esta sección, se hace referencia a esta ecuación.

sobreajuste a los datos de entrenamiento. Existen diferentes heurísticas para optimizar su comportamiento.

Funciones de activación

Dentro de la arquitectura también podemos incluir como componente “especial” a las funciones de activación. Cada neurona de las capas ocultas y de la capa de salida aplican una función de activación a la suma ponderada de las entradas. Las funciones de activación introducen no linealidad en la red, permitiendo así que esta aprenda patrones complejos en los datos. Entre las más comunes podemos encontrar: [7] [8]

- **Función sigmoide (Ver apartado 2.2.1 Concepto y evolución de las Redes Neuronales):** Si bien la función sigmoide fue una de las primeras en utilizarse, actualmente su uso es limitado debido a problemas como el desvanecimiento del gradiente en parte provocado por que su salida no es centrada en 0:

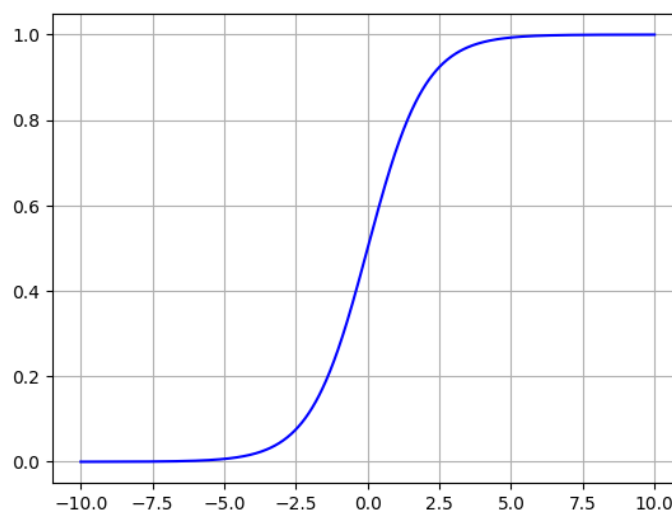


Figura 6: Función sigmoide

- **Tanh (Hyperbolic tangent):** Para resolver el problema de la función sigmoide se introdujo la función tanh la cual sí está centrada en 0. La salida de esta función toma valores entre [-1,1] y viene dada por:

$$\tanh(z) \equiv \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Figura 7: Ecuación de la tangente hiperbólica

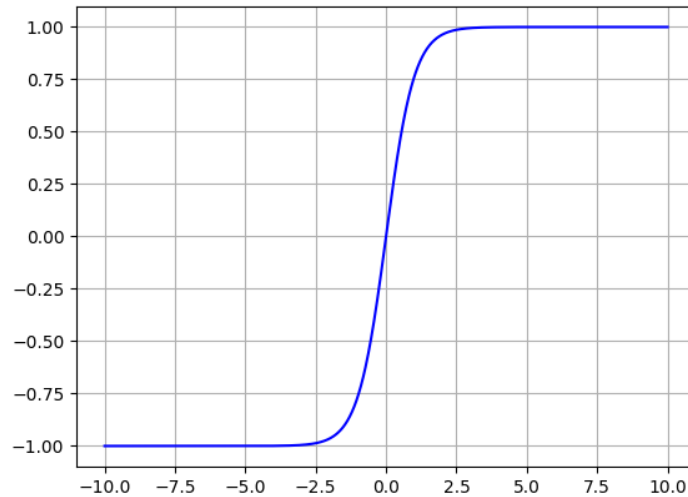


Figura 8: Función tangente hiperbólica

Sin embargo, aunque sea capaz de mitigar el problema de la función sigmoide, sigue sin poder resolverlo del todo y por tanto ambas funciones siguen siendo susceptibles al problema del desvanecimiento del gradiente (Este concepto se desarrolla en el apartado 2.3.4.4).

- **ReLU (Rectified Linear Unit):** Ayuda a evitar el desvanecimiento del gradiente al no comprimir tanto el rango de valores como lo hacen las funciones sigmoide o tanh y además requiere de menor coste computacional. Su ecuación es:

$$f(z) = \max(0, z)$$

Figura 9: Ecuación de ReLU

La función ReLU solo activa la neurona si la entrada es positiva, si es negativa, su salida es 0. Debido a ello algunos gradientes pueden morir durante el entrenamiento (en las regiones donde $x < 0$).

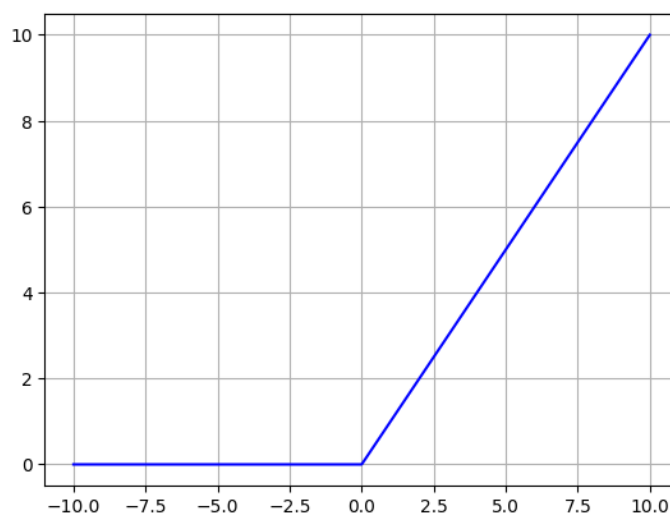


Figura 10: Función ReLU

- **Leaky ReLU:** Modifica la función ReLU original para permitir valores negativos (pequeños) en lugar de convertirlos directamente en 0. Se usa un hiperparámetro (con un valor normalmente entre el rango 0.01-0.3) para multiplicar los valores negativos, con esto se consigue reducir todavía más el problema de desvanecimiento del gradiente. Su función es:

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$$

Figura 11: Ecuación de Leaky ReLU y Parametric ReLU

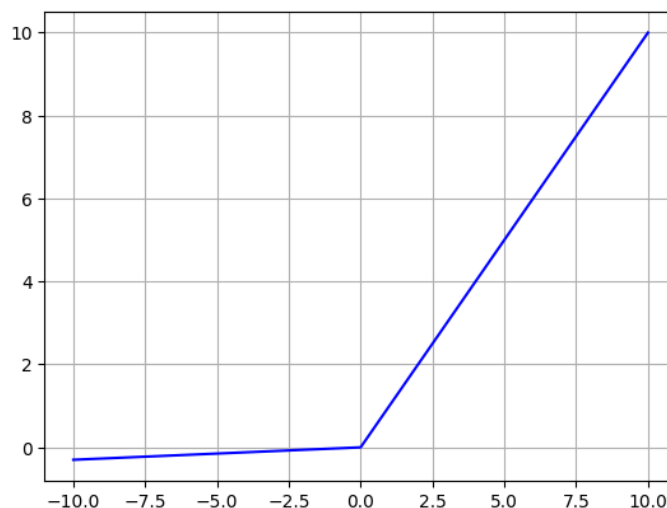


Figura 12: Función Leaky ReLU

Una variante de esta función es la Parametric ReLU (PReLU). En lugar de multiplicar los valores negativos por una hiperparámetro se utiliza un parámetro aprendible (como si de los pesos o los sesgos se tratase), mejora la precisión y la convergencia, pero es más costoso de llevar a cabo. Comparte la misma ecuación de la función Leaky ReLU.

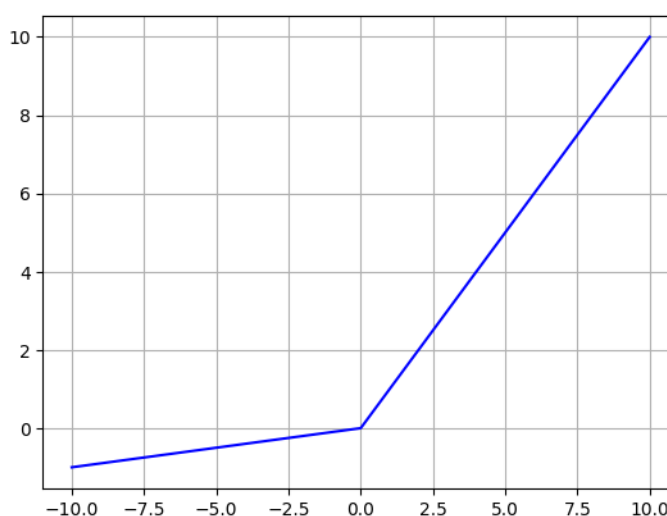


Figura 13: Función Parametric ReLU

Otras variantes, las cuales se utiliza en los modelos seleccionados para el proyecto son:

- **SiLU**, en la cual se multiplica la entrada por la función sigmoide. En lugar de tener una curva tan pronunciada en cero, SiLU permite suavizar dicha curva lo que significa que al cambiar el valor de la entrada su salida no cambia tan bruscamente [9]. Su función se describe como:

$$f(z) = z * \sigma(z)$$

Figura 14: Ecuación de SiLU

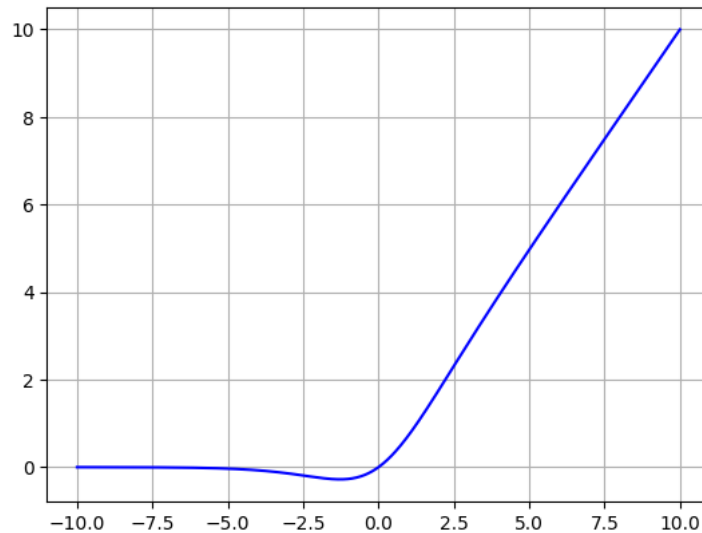


Figura 15: Función SiLU

- **GeLU**, en la cual se multiplica la entrada por la función de distribución acumulativa gaussiana estándar, tiene la misma motivación que SiLU, solo que en esta función suaviza menos la curva [10]. Su función viene dada por:

$$f(z) = z * \phi(z)$$

Esta ecuación se puede aproximar a:

$$f(z) = z * \sigma(1.702z)$$

Figura 16: Ecuación aproximada de GeLU

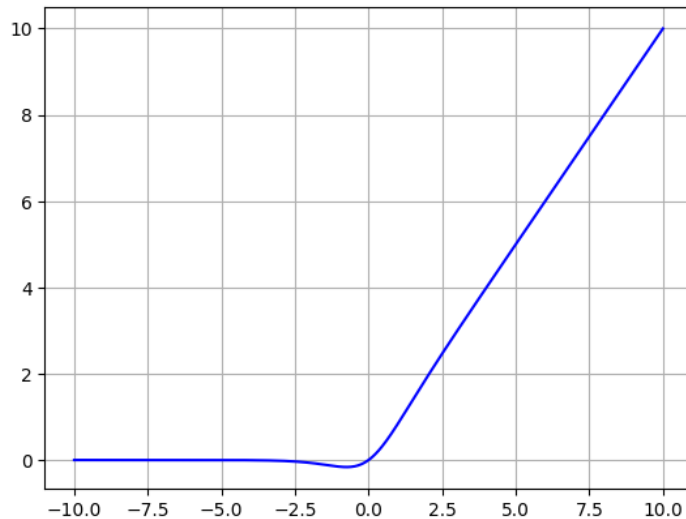


Figura 17: Función GeLU

Gracias a dichas funciones, las redes neuronales modernas se pueden entrenar de una manera más eficiente siempre y cuando se escoja adecuadamente la mejor función que se adapte a la naturaleza del problema o tarea a resolver. Cabe decir que existen más funciones de activación, pero son mucho más complejas de lo que se necesita explicar en este trabajo.

2.2.3 Entrenamiento de Redes Neuronales

Para que una red neuronal aprenda a realizar una tarea correctamente, es necesario un proceso de entrenamiento en el que se ajustan los pesos y sesgos de cada neurona con el objetivo de minimizar el error en las predicciones. De aquí en adelante se toma la función sigmoide como ejemplo para la función de activación.

Función de pérdida

El entrenamiento de una red neuronal se basa en la definición de una función de coste (también llamada función de pérdida o función objetivo), que mide la diferencia entre la salida deseada y la salida real de nuestro modelo. Podemos distinguir los siguientes tipos:

- Función cuadrática de coste

La fórmula de la función cuadrática de coste es:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

Figura 18: Función cuadrática de coste

Donde:

- **w**: colección de todos los pesos en la red.
- **b**: todos los sesgos.
- **n**: número total de entradas.

- **x**: entrada de la red neuronal.
- **y(x)**: resultado deseado.
- **a**: vector de salidas.
- $\|v\|$: notación que denota la longitud usual de un vector v .

Por ejemplo, en el caso del reconocimiento de dígitos, si una imagen de entrenamiento representa un 6, el vector de salida deseado será:

$$y(x) = (0,0,0,0,0,0,1,0,0,0)^T$$

Esta función es siempre positiva, cuando se aproxima a 0 significa que $y(x)$ es aproximadamente igual a la salida “a” para todas las entradas “x”. Nuestro algoritmo habrá hecho un buen trabajo si conseguimos que $C(w,b) \approx 0$.

- Entropía cruzada (Cross-Entropy)

Cuando una red neuronal comete errores graves su aprendizaje se ralentiza, estos errores ocurren cuando las salidas de la función sigmoide son cercanas a 0 o 1. En estas regiones la curva de la función es plana, lo cual significa que sus derivadas son pequeñas y por ende producen gradientes pequeños. En el siguiente apartado se explica por qué tener gradientes pequeños produce esta ralentización de aprendizaje.

Para solventar este problema de ralentización se reemplaza la función cuadrática de coste por una nueva función de coste conocida como entropía cruzada. La fórmula de la entropía cruzada es:

$$C = -\frac{1}{n} \sum [y \cdot \log(a) + (1 - y) \cdot \log(1 - a)]$$

Figura 19: Función de entropía cruzada

Donde:

- **y**: es la salida deseada de la muestra (suele ser 0 o 1).
- **a**: es la salida real de la neurona.
- **n**: es el número total de ejemplos.

A simple vista no parece ser una función de coste, pero tiene 2 propiedades que hacen razonable interpretarla como tal:

1. La función es positiva. Todos los términos en la suma son negativos y delante del sumatorio tenemos un signo negativo, por lo que la expresión es $C > 0$.
2. Si la salida real de una neurona es similar a la deseada para todas las entradas, entonces esta función será cercana a 0. Esto se llega a conseguir a través del entrenamiento.

$$(\forall x \in X, a(x) \approx y(x)) \Rightarrow C \rightarrow 0$$

Al realizar las derivadas parciales respecto a los pesos y los sesgos en ambas funciones, se puede apreciar que, en el caso de la entropía cruzada cuanto mayor sea el error en la salida de la neurona, más rápido será su aprendizaje. Lo más relevante es que desaparece el término $\sigma'(z)$ presente en la función cuadrática, lo que evita la ralentización de aprendizaje provocada por los gradientes pequeños.

Cuando las neuronas de salidas sean neuronas sigmoideas, usar la función de entropía cruzada es siempre mejor que usar la cuadrática. Al formar una red neuronal, normalmente se establecen los pesos y los sesgos de manera aleatoria y para evitar la ralentización de aprendizaje se ha de usar esta función.

Algoritmo de retropropagación (Backpropagation)

Siguiendo la explicación, tenemos claro que el objetivo principal es minimizar la función de coste. Para ello usamos este algoritmo cuyo objetivo principal es calcular las derivadas parciales de la función de coste con respecto a cualquier peso y sesgo de la red neuronal. Para calcular las derivadas parciales con este algoritmo se introduce un término intermedio δ_i , es el vector de errores asociados en la capa i . La retropropagación siempre nos da el procedimiento para calcular dichos errores para cada capa y luego poder relacionarlos con las derivadas parciales.

Antes de comenzar a explicar la retropropagación se han de hacer 2 suposiciones para que el algoritmo funcione:

1. La función de coste se puede escribir como un promedio:

$$C = \frac{1}{n} \sum_x C_x$$

La razón de hacer esta suposición es que el algoritmo nos permite calcular las derivadas parciales para un solo ejemplo de entrenamiento “ x ” y, por tanto, para obtener las derivadas de la función de coste C , basta con promediar las derivadas obtenidas para cada ejemplo individual.

2. La función de coste también se puede escribir como una función de las salidas de la red neuronal:

$$C = C(a)$$

El valor real de la salida, es decir, “ y ” es fijo y durante el entrenamiento no cambia, lo único que se puede modificar en el entrenamiento para minimizar el error es la salida y por tanto se considera que C solo depende de las salidas.

El algoritmo consta de 4 pasos principales (se incluyen las 4 ecuaciones fundamentales del backpropagation):

1. Propagación hacia adelante (*Forward pass*)

Para un dato de entrada (x), se calculan las activaciones a^l en cada capa de la red. Para ello se calcula la entrada ponderada y después se aplica la función de activación:

$$z^l = w^l a^{l-1} + b^l$$

Figura 20: Cálculo de la entrada ponderada

$$a^l = \sigma(z^l)$$

2. Cálculo del error en la capa de salida (δ^L)

Se calcula el error en la capa de salida utilizando la 1ª ecuación fundamental:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Figura 21: Ecuación de error en la capa de salida

3. Retropropagación del error (δ^l)

Se calculan los errores δ^l para cada capa oculta usando la 2ª ecuación fundamental:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

Figura 22: Ecuación de retropropagación del error

4. Cálculo de gradientes de los pesos y los sesgos

Una vez calculados los errores se calculan los gradientes correspondientes a los pesos y los sesgos utilizando para ello las dos últimas ecuaciones fundamentales:

$$\frac{\partial C}{\partial b^l} = \delta^l$$

Figura 23: Ecuación de la derivada del coste respecto a los sesgos

$$\frac{\partial C}{\partial w_{jk}^l} = a_j^{l-1} \delta_k^l$$

Figura 24: Ecuación de la derivada del coste respecto a los pesos

Sin retro propagación el cálculo de los gradientes sería largo e ineficiente ya que se tendrían que repetir los cálculos para cada parámetro de forma independiente. Usar este algoritmo tiene una serie de ventajas, alguna puede ser:

- **Reutilización de cálculos comunes**
- **Eficiencia:** usarlo implica acelerar el aprendizaje mayormente provocado por la actualización de pesos en función del error.

- **Escalabilidad y generalización:** el algoritmo escala eficientemente con conjuntos de datos mayores y redes más complejas, también se ayuda a que los modelos generalicen mejor frente a nuevos datos.

Algoritmos de optimización

Como hemos explicado, el objetivo es siempre minimizar al máximo la función de coste, para ello, después de obtener los gradientes hay que actualizar los parámetros de la red neuronal, lo cual se consigue gracias a los algoritmos de optimización.

- Descenso del gradiente (Gradient Descent)

Uno de los más utilizados es el descenso del gradiente, el cual ajusta los pesos y los sesgos de la red neuronal para reducir el error en cada iteración realizada. El descenso del gradiente sigue estos pasos:

1. Calcula el gradiente (la dirección de mayor descenso) de la función de coste respecto a los pesos y los sesgos.
2. Actualiza los pesos en pequeñas proporciones controladas por la tasa de aprendizaje (η), un valor pequeño y positivo. Se necesita que este valor sea lo suficientemente pequeño como para que la aproximación sea correcta, pero no demasiado porque si no los cambios también serán pequeños y el algoritmo se desarrollaría demasiado lento.
3. Se repite el proceso iterativamente hasta reducir lo máximo posible la función de coste.

Regla de actualización:

$$v \leftarrow v - \eta \nabla C$$

Figura 25: Regla de actualización en DG

Donde:

- **v:** son los pesos y sesgos
- **∇C :** es el gradiente de la función de coste.
- **η :** es la tasa de aprendizaje.

Al hacer esto una y otra vez seguiremos disminuyendo C hasta conseguir el mínimo. Hay una pequeña restricción, $\|v\| = \epsilon$ donde $\epsilon > 0$ y es un valor fijo, para garantizar que los pasos de cada iteración son pequeños y controlados. Si el número de entradas es muy alto puede tomar mucho tiempo y el aprendizaje es muy lento. Para ello se introduce el término llamado descenso de gradiente estocástico.

- Descenso de Gradiente Estocástico (SGD - Stochastic Gradient Descent)

En lugar de actualizar los pesos usando el conjunto entero de entrenamiento, se actualizan mediante pequeños subconjuntos denominados mini-lote (mini-batches). El proceso de entrenamiento con SGD se repite en ciclos denominados épocas. En cada época:

1. Se “barajan” los datos aleatoriamente (para evitar la repetición de patrones).

2. Se divide el conjunto de entrenamiento en mini-lotes.
3. Se actualizan los pesos y los sesgos para cada mini-lote, las cuales vienen dadas por la siguiente regla:

$$v_{i+1} = v_i - \eta \nabla C(v_i)$$

Figura 26: Regla de actualización en SGD

Es la misma idea que en el descenso del gradiente solo que por épocas.

El quid de la cuestión es repetir el SGD con diferentes mini-lotes hasta agotar todas las entradas posibles, una vez se llega a este punto se dice que se ha completado una época de entrenamiento. En este punto se han de reorganizar las entradas y crear mini-lotes distintos, es decir, empezar otra época. El SGD permite que nuestra red neuronal tenga un aprendizaje más rápido y una mejor generalización frente a nuevos datos.

- Adam y AdamW

Son variantes de SGD pues siguen con la idea de ajustar los parámetros en cada paso que da el algoritmo. Adam regula la tasa de aprendizaje para cada parámetro mediante dos cálculos principales [11]:

1. **Momentum (momento):** es un promedio móvil exponencial de los anteriores gradientes calculados. Se calcula mediante:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla C_t$$

Figura 27: Cálculo del momento

Donde:

- **m_t :** es el momento en el intervalo de tiempo t.
 - **β_1 :** es un hiperparámetro para controlar la ponderación que recibe el gradiente anterior.
 - **∇C_t :** es el gradiente actual.
2. **RMSprop (Root Mean Square Propagation):** es un promedio móvil exponencial de los gradientes al cuadrado. Se calcula mediante:

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla C_t)^2$$

Figura 28: Cálculo del RMSprop

Donde:

- **v_t :** es el RMSprop en el intervalo de tiempo t.
- **β_2 :** es un hiperparámetro para controlar la ponderación que recibe el gradiente al cuadrado anterior.

Tanto el momento como RMSprop comienzan con valor igual a 0 (cuando $t=0$), para ello hay que realizar una corrección en los sesgos mediante las siguiente formula de actualización:

$$m'_t = \frac{m_t}{(1 - \beta_1^t)}$$

Figura 29: Inicialización del momento

$$v'_t = \frac{v_t}{(1 - \beta_2^t)}$$

Figura 30: Inicialización de RMSprop

Una vez hechas las correcciones se puede escribir en su totalidad la fórmula de Adam:

$$\theta_{t+1} = \theta_t - \eta \frac{m'_t}{\sqrt{v'_t} + \epsilon}$$

Figura 31: Fórmula de Adam

Donde:

- **θ_t :** es un parámetro (peso o sesgo) en el intervalo de tiempo t.
- **η :** es la tasa de aprendizaje
- **ϵ :** es un número con valor muy reducido, se usa para evitar la división por cero en la fórmula.

Adam es un buen algoritmo de optimización, no obstante, si se le quiere aplicar la regularización L2 (Término que se explica más adelante en este apartado), se mezcla con las tasas de aprendizaje y puede provocar errores, para paliar dicho fallo se introduce su variante AdamW, el cual separa el término de regularización y en vez de aplicarlo directamente en la función de pérdida, lo aplica en las actualizaciones de los parámetros:

$$\theta_{t+1} = \theta_t - \eta \left(\frac{m'_t}{\sqrt{v'_t} + \epsilon} + \lambda + \theta_t \right)$$

Figura 32: Fórmula de AdamW

Donde:

- **λ :** es el término de regularización.

Selección de hiperparámetros

El rendimiento de nuestra red neuronal depende de varios factores, su arquitectura y los datos e hiperparámetros utilizados para el entrenamiento. No hay que confundirlos con los parámetros internos (pesos y sesgos), los hiperparámetros no se aprenden a medida que se entrena, se han de ajustar antes. Este ajuste puede ser manual o usando estrategias de optimización. Los hiperparámetros principales son:

- **Tasa de aprendizaje (*learning rate*):** Determina con qué rapidez ajusta nuestra red los parámetros internos (pesos y sesgos). Si la tasa es baja, esta actualización es lenta y puede tardar

mucho en alcanzar la convergencia. Por el contrario, si es demasiado alta, el modelo puede hacer una corrección excesiva de errores y provocar que diverja.²

- **Tamaño del mini-lote:** Determina cuántos ejemplos de entrenamiento se utilizan en cada iteración del entrenamiento. Un tamaño de mini-lote pequeño actualiza el modelo con mayor frecuencia lo que genera actualizaciones “ruidosas. Si el tamaño del mini-lote es grande el modelo se actualiza más lento, pero esto ofrece actualizaciones más estables (también es más fácil que se produzca un sobreajuste).
- **Número de épocas:** Determina cuántas veces se recorre el conjunto de entrenamiento durante este. Si hay pocas épocas el modelo puede que no aprenda lo suficiente (lo que se conoce como *underfitting*). Por el contrario, si hay muchas épocas, la red puede llegar al sobreajuste (*overfitting*).
- **Número de capas y neuronas:** Definen la arquitectura de la red neuronal. Las redes poco profundas pueden sufrir *underfitting* al no saber captar relaciones complejas. Las redes muy profundas se pueden ver afectadas por el desvanecimiento del gradiente.
- **Funciones de activación:** Ya han sido explicadas anteriormente.
- **Técnicas de regularización** (Se explican en el siguiente apartado).

La elección de los hiperparámetros es de elección personal, no hay ninguna regla establecida que defina que valores son los óptimos, normalmente su elección se lleva a cabo mediante un proceso de experimentación. Las estrategias más comunes son:

- **Experimentación manual:** Consiste en probar distintas combinaciones de valores y evaluar su impacto en el rendimiento de la red neuronal. Es una buena forma de conocer el comportamiento de las redes neuronales, pero puede llegar a ser muy lento.
- **Búsqueda en cuadrícula (*grid search*):** Se prueban todas las combinaciones posibles de los conjuntos de valores para cada hiperparámetro, estas combinaciones forman cuadrículas, de ahí el nombre de este método. Se automatiza el proceso de elección por lo que es más eficaz que la experimentación manual, pero conlleva aumentar la complejidad computacional.

Problemas que surgen en el entrenamiento

Durante el proceso de entrenamiento pueden aparecer varios problemas debido a la forma en la que se realiza este. Los problemas más comunes son:

² Puede incluso provocar la explosión del gradiente, término que se explica en el siguiente apartado.

- Problema del sobreajuste (overfitting)

El sobreajuste se produce cuando una red neuronal aprende tan bien los detalles del conjunto de entrenamiento que tiene un mal desempeño para datos nuevos. La red memoriza todo el conjunto incluyendo las partes irrelevantes en lugar de aprender patrones generales que es lo que realmente importa.³ Las principales causas de que se produzca el sobreajuste son:

- **Red neuronal compleja:** Hay demasiadas capas, neuronas y parámetros, la red tiene una gran capacidad de aprendizaje, pero puede ocasionar que se memoricen patrones innecesarios y provoque sobreajuste.
- **Conjunto de datos de entrenamiento pequeño:** Por el contrario, si se tienen pocos ejemplos de entrenamiento, puede ser que la red se centre en patrones innecesarios que no le permitan después generalizar.
- **Proceso de entrenamiento prologando:** Si entrenamos durante mucho tiempo a una red existe un mayor riesgo de que aprenda detalles o patrones específicos del conjunto de entrenamiento y que después no sepa generalizar.

Para detectar si nuestra red tiene problemas de sobreajuste nos hemos de fijar en los conjuntos de entrenamiento y validación. Si la precisión del conjunto de entrenamiento mejora y la precisión del conjunto de validación se estanca o empeora entonces nuestra red se está sobreajustando. Existen varias técnicas para evitar el sobreajuste:

- **Aumento de datos (*Data augmentation*):** Se crean nuevas variaciones de los datos de entrenamiento. Esto aumenta el tamaño del conjunto de entrenamiento y ayuda a la red a poder generalizar mejor. Ya que manualmente el obtener nuevos datos puede ser un proceso costoso, normalmente se realiza de manera artificial, por ejemplo, si estamos trabajando en clasificación de imágenes, con rotar o cambiar algunos tonos de la imagen ya estamos creando una nueva variación de esta.
- **Regularización:** Se añade un término a la función de coste que penaliza a los pesos grandes. Este método reduce la complejidad del modelo y así evita que se ajuste a detalles específicos. La más usada es la regularización L2 (también conocida como *weight decay*), la cual penaliza a los pesos más grandes mediante un término conocido como término de regularización añadido a la función de coste.
- **Abandono (*Dropout*):** En lugar de modificar la función de coste, se modifica directamente la red. Consiste en “apagar” aleatoriamente neuronas de las capas ocultas en cada iteración del entrenamiento, lo cual provoca que cada vez se entrene con una red distinta. Aunque cada versión

³ Por el contrario, el *underfitting* ocurre cuando la red no consigue aprender los suficientes patrones del conjunto de datos, generalmente por ser demasiado simple. Aunque es menos común que el sobreajuste es importante nombrar el concepto para entender los problemas que pueden surgir.

de la red puede sobreajustarse a una parte del conjunto de datos, al terminar el entrenamiento y usar la red completa (sin aplicar *dropout*), se obtiene una especie de promedio de los efectos aprendidos por las diferentes versiones. Esto ayuda a reducir el sobreajuste global de la red y mejorar la capacidad de generalización.

- **Detención temprana (*Early stopping*):** Se detiene el entrenamiento antes de que la red tenga la posibilidad de sobreajustarse, se lleva a cabo mediante la monitorización del conjunto de validación, deteniendo el entrenamiento cuando este deje de mejorar.

- Desvanecimiento y explosión del gradiente

Este problema ocurre al usar la retropropagación, los gradientes se calculan mediante las derivadas parciales en las cuales aparecen multiplicaciones, debido al término introducido por la función sigmoide y la inicialización escogida de los parámetros, este producto puede disminuir (desvanecimiento) o crecer (explosión) a medida que se retrocede en las capas ocultas. En redes neuronales profundas formadas por muchas capas este problema crece debido a que el número de operaciones a realizar aumenta.

- **Desvanecimiento del gradiente (*Vanishing gradient*):** Los gradientes se vuelven extremadamente pequeños en las capas cercanas a la entrada lo que provoca que el aprendizaje sea muy lento o incluso nulo. Esto ocurre con mayor frecuencia al usar funciones de activación como la sigmoide o la tangente hiperbólica.
Como consecuencia de esta explicación, las capas cercanas a la entrada aprenden a menor velocidad que las capas cercanas a la salida, debido a que reciben gradientes más pequeños. Todo esto puede provocar que las capas cercanas a la salida sufran de sobreajuste.
- **Explosión del gradiente (*Exploding gradient*):** Los gradientes pueden crecer exponencialmente a medida que ocurre la retropropagación, lo cual provoca actualizaciones inestables de los pesos y sesgos, haciendo que el modelo diverja. Normalmente es menos común que el desvanecimiento.

Existen varias técnicas para evitar estos dos problemas:

- **Inicialización adecuada de los pesos:** Una inicialización inapropiada de los pesos puede producir tanto el desvanecimiento como la explosión del gradiente, dificultando así el entrenamiento de la red. Para solventar dichos problemas se pueden usar diferentes métodos de inicialización como LeCun, Xavier o He.
- **Funciones de activación alternativas:** Funciones como ReLU o cualquiera de sus variantes ayudan a evitar el desvanecimiento del gradiente (Ver el apartado **Dentro de la arquitectura** también podemos incluir como componente “especial” a las funciones de activación).
- **Regularización y *dropout*:** Ambas técnicas evitan indirectamente ambos problemas al mejorar el aprendizaje de la red neuronal.

Existen más técnicas que ayudan a reducir ambos problemas, pero estas son las principales.

Para acabar esta sección vamos a desglosar el ciclo completo de entrenamiento de una red neuronal:

1. Inicialización de pesos y sesgos, generalmente de forma aleatoria.
2. Aplicación del algoritmo de retropropagación para calcular el gradiente del error respecto a los parámetros de la red.
3. Uso de algoritmos de optimización para actualizar los pesos y sesgos de la red basándose en los gradientes calculados.
4. Repetición del proceso durante varias épocas hasta minimizar lo máximo posible la función de coste o alcanzar el valor predefinido.

2.2.4 Tipos principales de redes neuronales

A continuación, se describen los tipos de redes neuronales más comunes hoy en día. En la mayoría de este subapartado se ha utilizado de referencia [12].

Redes neuronales prealimentadas (Feedforward Neural Network - FNN)

Las FNN también denominadas como MLP⁴ tienen como objetivo aproximar una función, de modo que la información fluya solo hacia adelante (de ahí su nombre). No existe ningún tipo de retroalimentación en la red, es decir, las neuronas no forman ciclos entre las capas o entre ellas mismas.

En una FNN, cada neurona normalmente⁵ está conectada a todas las neuronas de la capa adyacente, permitiendo así que la red posea toda la información posible. Estas redes están formadas por una capa de entrada, una o más capas ocultas y finalmente, una capa de salida.

Estas redes sirven como base para el desarrollo de muchas otras. Algunas de las aplicaciones de este tipo de redes son las tareas de reconocimiento visual y de voz o el procesamiento de lenguaje natural, entre otras muchas.

Redes neuronales convolucionales (Convolutional Neural Network - CNN)

Las CNN solucionan el problema de estructura espacial que tienen las FNN, al utilizarse filtros estas redes son capaces de detectar las relaciones espaciales. Estos filtros se basan en la operación denominada convolución y se denominan filtros convolucionales [13]:

$$s(t) = (x * w)(t) = \int x(a)w(t - a)da$$

Figura 33: Fórmula de la convolución

Donde:

- x es la entrada.

⁴ El perceptrón simple a veces es considerado una FNN, aunque esté formado solamente por una capa.

⁵ Cuando esto ocurre se trata específicamente de una red FCN (*Fully connected network*), una variante de las FNN.

- w es el filtro convolucional (también denominado *kernel*).
- t es la distancia que se desplaza.
- $s(t)$ es la salida de la función y se denomina mapa de características.

Las redes CNN están formadas por:

- **Capas convolucionales:** Se encargan de extraer las características locales de la entrada, comúnmente imágenes, como los bordes, colores, etc. Se realiza a través de los *kernels* y esto genera el mapa de características.
- **Capas de agrupación (*pooling layers*):** Simplifican la salida de las capas convolucionales, es decir, reducen la dimensionalidad del mapa de características. Dicha agrupación permite que la representación obtenida no varíe frente a pequeños cambios o traslaciones de la entrada (por ejemplo, rotar la imagen). Existen dos tipos principales de agrupación:
 - **Max pooling:** Selecciona el máximo valor dentro de una región específica del mapa de características. Así, se recuperan las características más importantes.
 - **Average pooling:** Calcula el valor promedio de una región específica del mapa de características. Puede ser útil para reducir el ruido, pero se pueden perder características importantes.
- **Capas totalmente conectadas (*fully connected layers*):** En las CNN se suelen incluir una o más de estas capas, las cuales conectan cada neurona de las *pooling layers* con la capa de salida permitiendo realizar una clasificación.

Las propiedades clave de este tipo de redes son:

- **Conectividad dispersa (*sparse connectivity*):** Esta idea consiste en hacer los *kernels* más pequeños que la propia entrada. Básicamente si tenemos una imagen la cual tiene, por ejemplo, un millón de píxeles, podemos seguir detectando las características locales con un kernel que ocupe cientos de píxeles. De esta manera la red tiene que usar menos parámetros y por tanto se reduce la memoria usada.
- **Compartición de parámetros:** En las CNN, cuando se aplica la operación de convolución se comparten los parámetros para cada *kernel*, lo cual, junto a la propiedad anterior, hace que se reduzca incluso más el número de parámetros a utilizar.

No obstante, estas redes no son perfectas y presentan algunas limitaciones, como:

- **Dificultad de diseño:** Este tipo de redes están formadas por muchos hiperparámetros y encontrar la combinación correcta puede ser difícil, para ello se usan las técnicas ya vistas anteriormente.

- **Eficiencia computacional y consumo de memoria:** Las CNN pueden ser muy costosas en términos de tiempo de entrenamiento y de utilización de memoria. El uso de GPUs es esencial para acelerar el proceso de entrenamiento. Además, para optimizar el uso de memoria se recomienda maximizar el trabajo de las *pooling layers* y/o usar arquitecturas más eficientes de las CNN como pueden ser LeNet-5, AlexNet, VGG o ResNet. [14]

Redes neuronales recurrentes (Recurrent Neural Network - RNN)

Las RNN a diferencia de las redes tradicionales, tienen conexiones recurrentes que permiten mantener la información de estados previos. El comportamiento de las neuronas no solo se debe a las activaciones de las capas previas, puede estar determinada por su propia activación en un momento anterior.

Estas redes sufren del desvanecimiento del gradiente, el problema incluso empeora en este tipo de redes porque los gradientes no solamente se propagan hacia atrás de las capas, sino que también lo hacen a través del tiempo. Para mitigar este problema surge la variante denominada LSTM (*Long Short-Term Memory*) la cual es capaz de aprender dependencias a largo plazo.

Las RNN se usan principalmente para procesar datos secuenciales, como texto, audio o series temporales.

Arquitecturas *encoder-decoder*

Están formadas por dos módulos [15]:

- **Codificador (*encoder*):** se encarga de procesar la entrada y de generar una representación del espacio latente, es decir transforma los datos de entrada en uno o varios vectores numéricos, lo que se conoce como *embeddings*. Están formados por una capa de *self-attention*⁶, la cual permite que el codificador se centre en las partes importantes que producen el contexto, y por una red FNN que permite procesar la información y capturar relaciones y patrones entre los datos.
- **Decodificador (*decoder*):** utiliza la representación generada por el codificador para generar una salida, esta salida puede ser una reconstrucción de la entrada o una secuencia distinta (generalmente esta última). Comparten las dos capas que forman a los codificadores con la salvedad, que entre estas dos hay una capa de atención *encoder-decoder*, la cual permite que el decodificador se centre en las partes más relevantes de la entrada.

Dentro de esta arquitectura podemos distinguir dos tipos fundamentales de redes neuronales:

⁶ *Self-attention* es un mecanismo de atención mediante el cual cada elemento (*token*) de una secuencia se pondera según la importancia de cada uno en relación con los demás.

1. Transformers

El mecanismo de self-attention puede limitar la capacidad del modelo de aprender relaciones, pues podría prestar excesiva atención al token a estudiar (se pondera mucho que al resto de la secuencia). [16] [17] [18]

Para evitar esta limitación, los Transformers utilizan el mecanismo multi-head attention, el cual aplica múltiples mecanismos self-attention procesando la sentencia simultáneamente y obteniendo así mejores representaciones de las relaciones de cada token. Cada bloque del Transformer también contiene una red FNN para procesar los datos.

Los Transformers están diseñados originalmente como un modelo encoder-decoder, aunque en la actualidad hay modelos que solamente utilizan una parte, es decir, el codificador o el decodificador, denominándose encoder-only y decoder-only, respectivamente.

Encoder-only

Al solo tener la parte del encoder el modelo no se centra en generar texto, su tarea es entender los datos de entrada. Por ello las tareas principales que desempeña son clasificación de texto y extracción de información relevante.

Decoder-only

En el ejemplo de decoder-only al no tener la parte del encoder, directamente se intenta generar texto mediante la predicción del siguiente token. Para ello se usa una variante de self-attention conocida como masked self-attention, aquí no se tienen en cuenta todos los tokens de la secuencia, solo se miran los que vienen por detrás del token principal enmascarando los demás. Por ejemplo, en la secuencia:

“La casa de Pedro es muy grande”

Si el token principal es “Pedro” solamente se ponderan los tokens “La”, “casa” y “de”; cuando usábamos self-attention los que venían detrás de “Pedro” también se tenían en cuenta, pero aquí se ocultan mediante una máscara.

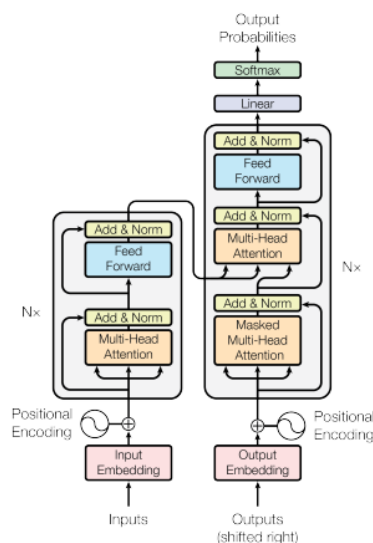


Figura 34: Arquitectura de los Transformers [19]

2. Autoencoders

Estas redes son el ejemplo de que la salida es una reconstrucción de la entrada, es decir, la entrada y la salida son las mismas, solo que se realizan algunas transformaciones en el proceso. Alguno de los usos principales de estas redes es [20]:

- Compresión de información (imagen y audio)
- Eliminación de ruido en datos (*denoising*)
- Reducción de la dimensionalidad

entre otros muchos. Se entrenan mediante aprendizaje no supervisado (Este concepto se explicará más adelante en este documento). Existen muchos más tipos de redes, pero considero que los explicados son los más relevantes para este proyecto.

2.3 Modelos de Lenguaje

Un modelo de lenguaje es un tipo de modelo estadístico o de aprendizaje automático para procesar lenguaje natural (PLN). Su objetivo principal es comprender, generar o clasificar texto y se usan para diversas tareas como la traducción, la clasificación o la generación de texto entre muchas otras. Los modelos se entrenan con grandes cantidades de datos para poder aprender patrones lingüísticos y relaciones semánticas [21] [22].

2.3.1 Tipos de modelos de lenguaje

Dentro de los modelos de lenguaje se pueden distinguir dos tipos principales, el modelado estadístico y el modelado neuronal.

Modelos estadísticos

El propósito de estos modelos es predecir la probabilidad que tiene una palabra de aparecer después de una secuencia, es decir, qué probabilidad tiene una palabra de ser la siguiente en generarse dado un contexto. Existen los siguientes tipos:

- **N-gramas:** Son secuencias de n elementos de una muestra de texto, este modelo analiza la probabilidad de la siguiente palabra basándose en las $(n-1)$ palabras anteriores. Por ejemplo, un modelo con $n = 5$ predecirá la siguiente palabra en base a las 4 anteriores. Puede tener cierta limitación ya que, si aumentamos el valor de n , el modelo puede que no cubra todas las secuencias posibles y dificulte el proceso de entrenamiento.
- **Modelos exponenciales:** En estos modelos se mezcla la predicción basándose en palabras anteriores (al igual que los n -gramas) junto a otras características del texto, como puede ser la aparición de ciertas palabras o patrones. Cada una de las características tiene un peso asignado y se combinan mediante una función exponencial para calcular la probabilidad. Son más flexibles que los n -gramas, pero son más costos computacionalmente hablando

- **Modelos *skip-gram*:** Se utilizan principalmente en el caso de generación de *word embeddings*. Dada una palabra objetivo predicen las palabras contexto (palabras cercanas a esta) dentro de una ventana con un tamaño previamente definido, por ejemplo:

“La casa de Pedro es muy grande”

Si nuestra palabra objetivo es “Pedro” y el tamaño de ventana es de 2, el modelo predice que las palabras: “casa”, “de”, “es”, “muy” suelen aparecer junto a la palabra objetivo y esto permite generar embeddings que representan estas relaciones. Puede recordar al mecanismo de *self-attention* definido anteriormente, pero con la salvedad de que aquí se utiliza una ventana para fijar el número de *tokens* a observar.

Modelos neuronales

Estos modelos son más avanzados ya que utilizan redes neuronales como base, lo cual permite un modelado más preciso al tener la capacidad de capturar patrones más complejos y una mejor generación de embeddings. Se dividen en:

- **Redes neuronales recurrentes (RNN):** La explicación más detallada de estas redes se ha desarrollado previamente, pero recordando la explicación, estas redes están diseñadas para datos secuenciales, lo que las hace idóneas para el modelado de lenguaje.
- **Modelos basados en Transformers:** También se han explicado previamente. La idea del mecanismo de atención es clave para la modelación de lenguaje, pues captura las relaciones semánticas de cada palabra de la secuencia paralelamente. Dentro de estos se encuentran los famosos *Large Language Model* (LLM), estos modelos se caracterizan por su gran número de parámetros (normalmente están formados por miles de millones) y su desempeño en una gran variación de tareas. El entrenamiento de estos modelos puede llegar a ser muy costoso computacionalmente (se suelen utilizar varias GPUs) ya que normalmente suele estar formado por varias etapas: un preentrenamiento no supervisado con una gran cantidad de datos y un posterior *fine-tuning* dedicado a especializar al modelo en tareas específicas.

2.3.2 Entrenamiento de modelos de lenguaje

Antes de explicar el proceso de entrenamiento de un modelo de lenguaje, hay que diferenciar las principales técnicas de aprendizaje [23] [24] [25]:

1. Aprendizaje no supervisado

El aprendizaje no supervisado emplea algoritmos capaces de analizar conjuntos de datos no etiquetados, es decir, sin información que facilite su contexto. Los algoritmos identifican patrones, relaciones o agrupaciones (clústeres) dentro de los datos, sin intervención humana. Se utilizan para dos tareas:

- **Agrupación en clústeres:** Se utilizan para procesar conjuntos de datos no etiquetados basándose en sus similitudes o diferencias. Se pueden clasificar en varios tipos:
 1. Agrupación excluyente: un punto de datos solamente puede existir en un clúster.
 2. Agrupación superpuesta: permite que los puntos de datos pertenezcan a varios clústeres teniendo diferentes grados de pertenencia en cada uno.
 3. Agrupación jerárquica: se crean grupos de forma que los datos similares están dentro del mismo y los diferentes están en otros grupos. Los clústeres más pequeños se fusionan en cada iteración hasta obtener una sola raíz.
Hay dos subtipos: agrupaciones aglomerativas, donde los elementos se unen de abajo hacia arriba y, las agrupaciones divisivas, que toman un enfoque contrario, de arriba hacia abajo (en este caso se comienza con un solo clúster y se va dividiendo en conjuntos más pequeños).
 4. Agrupación probabilística: las agrupaciones se crean en función de la probabilidad de pertenecer a una distribución determinada. Uno de los modelos más utilizados es el modelo de mezcla Gaussiana (GMM).
- **Regla de asociación:** Es un mecanismo basado en reglas que permite descubrir relaciones entre las características de un conjunto de datos. Un ejemplo de su uso son las cestas de la compra en los comercios online, permite a las empresas conocer qué relación existen entre unos productos y otros, estableciendo patrones. Un algoritmo muy conocido que usa las reglas de asociación es el algoritmo Apriori.

2. Aprendizaje supervisado

Como contraparte al aprendizaje no supervisado, aquí el modelo aprende mediante conjuntos de datos etiquetados. Estos datos están formados por las entradas y salidas deseadas, de forma que el algoritmo de aprendizaje tiene que ser capaz de encontrar los patrones y las relaciones para comprender como llegar a esas salidas. En entrenamientos a gran escala puede llegar a ser difícil mantener el uso de este algoritmo por falta de datos etiquetados (es muy difícil que ocurra, pero llegados a cierto punto puede ocurrir). Principalmente se usan en dos tareas:

- **Clasificación:** A través de los datos que se tiene, el algoritmo debe de ser capaz de deducir a qué categoría pertenece una nueva entrada. Un clasificador conocido es Naive Bayes el cual mide la probabilidad de pertenencia a una categoría para cada entrada del conjunto existente y al recibir un dato nuevo utiliza estos conocimientos para clasificarlo.
- **Regresión:** El algoritmo predice la variable dependiente (la nueva entrada de datos) a partir de una o más variables independientes (los datos existentes). Existen dos tipos, lineal (sencilla o múltiple, depende del número de variables independientes) y logística.

3. Técnicas mixtas

Definimos como técnica mixta a cualquier aprendizaje que mezcla conceptos de los aprendizajes supervisado y no supervisado.

- Aprendizaje semi supervisado

El aprendizaje semi supervisado se caracteriza por fusionar los conceptos del aprendizaje supervisado y no supervisado. Para el entrenamiento se utilizan tanto pares de datos etiquetados como no etiquetados, mediante esta fusión los algoritmos son capaces de aprender a etiquetar datos [26].

- Aprendizaje auto supervisado

Nace del concepto de auto entrenamiento donde primero se entrena un modelo con datos etiquetados, después se le pasan datos no etiquetados y el modelo genera una especie de pseudoetiquetas mediante las cuales refina el modelo a través de un proceso de iteración [27].

De esta idea surge el aprendizaje auto supervisado, donde no existe interacción humana, es decir, los datos no son etiquetados. Aquí, el modelo entrena gracias a las señales de supervisión que él mismo genera, lo cual se consigue mediante la creación de tareas de entrenamiento a partir de los datos no etiquetados (este proceso es similar a las pseudoetiquetas).

Una vez explicados los tipos de aprendizaje existentes, veamos como es el proceso de entrenamiento de un modelo de lenguaje. Surgen dos principales ideas:

Preentrenamiento

Como primera tarea, se han de optimizar los parámetros en un proceso denominado preentrenamiento, no se presupone qué tareas específicas va a realizar el modelo, simplemente se entrena en un ámbito general.

En el preentrenamiento se utiliza cualquiera de los aprendizajes que acabamos de definir, aunque, hoy día, la mayoría de los modelos de lenguaje dedicados a NLP se basan en el aprendizaje auto supervisado. En muchas ocasiones el entrenamiento de un modelo de lenguaje simplemente se basa en esta fase, obteniendo así un modelo de lenguaje “base” con un gran rango de tareas.

No obstante, existe un problema al entrenar los modelos de lenguaje, puede ser que una vez finalizada la etapa de preentrenamiento no se comporten siguiendo el raciocinio humano, es decir, pueden llegar a ser partidarios de ciertas posturas frente a otras (lo cual no es una buena práctica porque siempre se busca que el modelo sea imparcial) e incluso llegar a ser peligrosos o dañinos, si se le hacen cuestiones sobre temas delicados, el modelo puede llegar a dar una respuesta inadecuada y atentar contra la ética humana. En estos casos la siguiente fase del entrenamiento es totalmente necesaria.

Post entrenamiento

Una vez tenemos nuestro modelo preentrenado hay que transformarlo para que se focalice en ciertas tareas, en este proceso se han de reajustar los parámetros mediante un proceso conocido como *fine-*

tuning (ajuste fino), ya sea utilizando datos etiquetados (*supervised fine-tuning*) o facilitando al modelo descripciones de las tareas a realizar (*prompt engineering*). En el caso que hemos comentado donde el preentrenamiento puede derivar en un mal comportamiento del modelo, se introduce el concepto definido como RLHF (*Reinforcement Learning from Human Feedback*). A continuación, se explican los tres métodos [27]:

1. *Supervised Fine-tuning* (Ajuste fino supervisado)

En este método se utilizan datos etiquetados de forma que el modelo es capaz de aprender a justar los parámetros de la red para predecir *tokens* lo más parecidos posibles a los datos de entrada. Las técnicas más comunes son [28]:

- ***Fine-tuning* para una tarea específica:** Es el caso más común, el modelo se entrena solamente para desempeñar una tarea específica.
- **Aprendizaje multi-tarea:** En este caso se busca que el modelo a entrenar sea capaz de realizar varias tareas específicas, se consigue aprovechando las características compartidas y las diferencias entre las tareas.
- **Transferencia de aprendizaje (*Transfer learning*):** Cuando se trabaja con un conjunto de datos limitado esta técnica resulta muy útil porque se parte de la idea del modelo base (modelo ya preentrenado) en la que ya se tienen unos conocimientos previos y con el conjunto de datos que se tiene se realiza la tarea de *fine-tuning*. Debido a esto, el proceso de post entrenamiento se reduce.
- ***Fine-tuning* mediante instrucciones:** Este enfoque utiliza instrucciones como entrada, al modelo se le proporcionan pares o tuplas de entrada-salida deseada. Por ejemplo:

“Tengo 5 euros en la cartera, pero me he comprado una piruleta que me ha costado 20 céntimos. ¿cuánto me queda?” – Entrada

“Te quedan 4 euros y 80 céntimos” – Salida deseada

De esta forma, al usar muchas tuplas el modelo aprenderá a captar el seguimiento de instrucciones y llegará un punto en el que será capaz de responder a una pregunta que no haya visto antes. Esta idea entra en el campo del *prompt engineering*.

2. *Prompt engineering* (Ingeniería de instrucciones)

Se define por *prompt* a una instrucción que se facilita al modelo para que este ejecute una tarea específica, por tanto, la ingeniería de instrucciones tiene como objetivo el diseño y el uso efectivo de estos *prompts*.

Esta idea se puede aplicar directamente sobre un modelo preentrenado sin aplicar *fine-tuning* de por medio, no obstante, los modelos base son capaces de completar textos con una indicación inicial, pero no ideales para seguir instrucciones. Los modelos optimizados para instrucciones son versiones de estos

a los cuales se les aplica la técnica de *fine-tuning* por instrucciones, los cuales son más adecuados para el *prompting*.

Para explicar la ingeniería de instrucciones existe un método relacionado, el *in-context learning*, consiste en añadir un contexto mientras se realiza el *prompting*, generalmente este contexto son demostraciones de cómo se tiene que resolver la tarea que nosotros le pedimos al modelo. Dentro de este método tenemos tres aplicaciones:

- **Zero-shot learning:** Como su propio nombre indica, no hay un proceso explícito de aprendizaje, simplemente se proporciona al modelo un *prompt* sin la salida deseada para que ver qué responde. Esta explicación da sentido al apartado de *fine-tuning* mediante instrucciones pues su objetivo final es llegar a este punto, poder resolver nuevos problemas no vistos durante la fase de entrenamiento. Un ejemplo sería:

```
Eres un asistente especializado en gramática y traducción. Se te va a dar una oración y la tienes que traducir al inglés.  
  
Entrada: Juan ha tenido un examen muy difícil de matemáticas y está un poco triste.  
  
Salida: ____
```

Figura 35: Prompt zero-shot learning

- **One-shot learning:** A diferencia del anterior, en este caso solamente se introduce un ejemplo correcto de entrada y salida deseada:

```
Eres un asistente especializado en gramática y traducción. Se te va a dar una oración y la tienes que traducir al inglés.  
  
Entrada: Juan ha tenido un examen muy difícil de matemáticas y está un poco triste.  
  
Salida: Juan had a very difficult math examen and is a little bit sad.
```

Figura 36: Prompt one-shot learning

- **Few-shot learning:** En este método se utilizan varios ejemplos de entrada y salida deseada haciendo que el modelo de lenguaje aprenda los patrones y sea capaz de elaborar una respuesta mejor que con el caso de *one-shot*. Está claro que cuantos más ejemplos se introduzcan mejor generalizará el modelo, pero todo depende de la potencia de este, en ocasiones con introducir un grupo pequeño de ejemplos es suficiente para que el modelo aprenda correctamente. Un ejemplo:

<p>1ª Prueba</p> <p>Eres un asistente especializado en gramática y traducción. Se te va a dar una oración y la tienes que traducir al inglés.</p> <p>Entrada: Juan ha tenido un examen muy difícil de matemáticas y está un poco triste.</p> <p>Salida: Juan had a very difficult math examen and is a little bit sad.</p>
<p>2ª Prueba</p> <p>Eres un asistente especializado en gramática y traducción. Se te va a dar una oración y la tienes que traducir al inglés.</p> <p>Entrada: Leo Messi es el mejor jugador de fútbol de la historia.</p> <p>Salida: Leo Messi is the best football player in history.</p>

Figura 37: Prompt few-shot learning

3. RLHF

Este proceso a menudo se describe como aprendizaje por refuerzo, en el cual existe una interacción con *feedback* humano, consiguiendo que el modelo ajuste sus respuestas mediante un sistema de recompensas. Las técnicas más comunes de este método son:

- **Reward modeling (Modelado de recompensas):** Se define como el proceso de entrenar un modelo conocido como *reward model* para que este sea capaz de aprender a predecir las recompensas establecidas por evaluadores humanos y luego maximizarlas.

Para entrenar esta técnica se utiliza la clasificación comparativa, el modelo genera varias respuestas y los evaluadores las califican según su criterio, una vez obtenidas las respuestas con su recompensa asociada el modelo comienza el proceso de entrenamiento. La retroalimentación se puede obtener de varias formas:

- **Pairwise ranking:** Se proporciona dos salidas y los evaluadores eligen cuál es la óptima.
 - **Rating:** Los evaluadores establecen una puntuación a cada salida, normalmente la puntuación es un valor dentro de un rango numérico, pero también puede ser una puntuación binaria, por ejemplo, “sí-no”.
 - **Listwise ranking:** Se les muestra a los evaluadores una lista de posibles salidas ante una entrada y estos han de establecer un orden de puntuación.
- **Proximal policy optimization – PPO:** Es un algoritmo iterativo que actualiza el criterio o política del modelo sobre cómo maximiza las recompensas.

Se introduce un término de penalización para que la política actual (la nueva) no diste mucho de la política de referencia. Con esto conseguimos que el algoritmo no se aleje de la región de

confianza en la que se encontraba, en la cual se sabía que el modelo tenía un correcto comportamiento.

2.4 RAG (Retrieval-Augmented Generation)

RAG es una técnica dentro del campo de la inteligencia artificial que mezcla el uso de memoria paramétrica y no paramétrica. Antes de profundizar en RAG debemos diferenciar muy bien los dos tipos de memoria [29]:

- **Memoria paramétrica:** Se define como el conocimiento que reside en el propio modelo. Durante la época de entrenamiento se guarda dicho conocimiento en los pesos y no se precisa de ninguna fuente externa para la obtención de información, no obstante, tiene una serie de problemas como el aumento o modificación de su propia memoria que puede derivar en una pérdida de antigua información frente a nuevas entradas. También hay que tener en cuenta que cada vez que se quiera ampliar conocimiento el modelo ha de ser reentrenado lo que implica costes computacionales elevados.
- **Memoria no paramétrica:** Se usa principalmente en sistemas de recuperación, al tener la información guardada en fuentes externas, típicamente en bases de datos, permite que el sistema recupere la información deseada cuando se realiza una consulta.

Podríamos decir, por tanto, que RAG tiene como objetivo recuperar información mediante el uso de un recuperador (*retriever*), el cual accede a fuentes de datos externas (memoria no paramétrica) y generar respuestas a través de un modelo generador (*generator*) usando memoria paramétrica. Aquí un ejemplo de un sistema que usa RAG [30]:

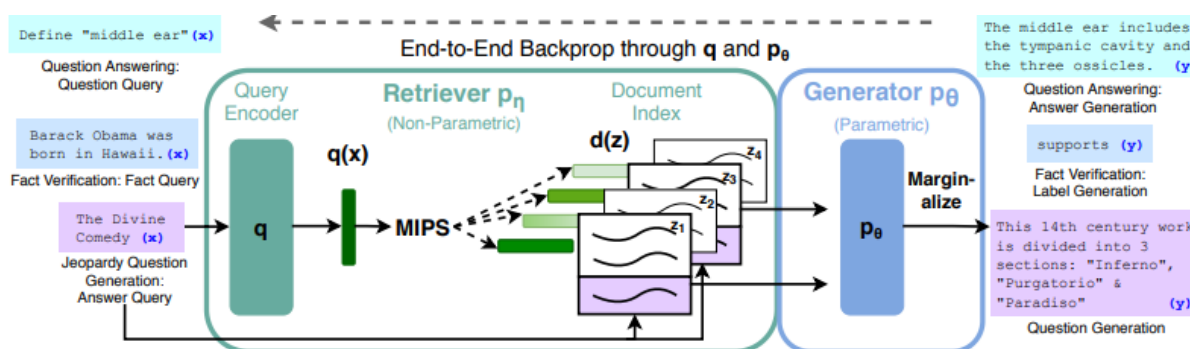


Figura 38: Sistema RAG en el que se usa DPR y BART [30]

2.4.1 Componentes principales

Como se puede ver en la “Figura 38” un sistema RAG está formado por dos componentes: el *retriever* y el *generator*. A continuación, se explican en profundidad.

Retriever (Recuperador)

La idea principal de un *retriever* es calcular los top-k documentos más relevantes para una consulta, para ello, el *retriever* se encarga de generar una representación de dicha consulta, así como representaciones para cada uno de los documentos (z_i) de los que obtiene información. Estas representaciones se obtienen a partir de *encoders*, recordemos que el papel de estos es transformar una entrada en *embeddings* pero ¿por qué recordar este concepto ahora? Muy sencillo, los *retrievers* calculan la probabilidad de que un documento z_i sea relevante para la consulta realizada mediante el producto escalar de las representaciones, lo que se consigue con este producto es mirar si los *embeddings* apuntan a la misma dirección, o, en otras palabras, que sean cercanos en el espacio vectorial. Dicha probabilidad viene dada por:

$$p_{\eta}(z|x) \propto \exp(d(z)^T q(x))$$

Figura 39: Probabilidad calculada por el retriever en RAG

Donde:

- μ son los parámetros del *retriever*.
- x es la entrada.
- z es un documento.
- $d(z)$ es el *embedding* del documento.
- $q(x)$ es el *embedding* de la consulta.

Generator (Generador)

La tarea del generador, básicamente, es generar, valga la redundancia, una respuesta dada una consulta x y los z documentos recuperados, los cuales se añaden como contexto adicional a la consulta. Se puede explicar como la probabilidad de qué se genere una secuencia de salida dada la entrada (consulta) y un documento relacionado, su ecuación viene dada por:

$$p_{\theta}(y_i|x, z, y_{1:i-1})$$

Figura 40: Probabilidad calculada por el generador en RAG

Donde:

- Θ son los parámetros del generador.
- y_i el token i de la salida.
- $y_{1:i-1}$ son los tokens de la salida ya generados previamente.

2.4.2 Modelos de RAG

Existen dos formas de marginalizar los documentos recuperados, por marginalizar se entiende aplicar una distribución marginal sobre las probabilidades asociadas a dichos documentos. De esta idea nacen los dos modelos de RAG:

1. RAG-Sequence

En *RAG-Sequence* cada documento se utiliza para generar una secuencia. Se buscan los documentos más relevantes para la entrada (1ª probabilidad) y una vez recogidos se mira cómo influye cada uno junto

con la propia entrada para poder generar la secuencia de salida (2ª probabilidad). Ambas probabilidades se marginalizan quedando:

$$p_{RAG-Sequence}(y|x) \approx \sum p_{\eta}(z|x) p_{\theta}(y|x, z) = \sum p_{\eta}(z|x) \prod_i^N p_{\theta}(y_i|x, z, y_{1:i-1})$$

Figura 41: RAG-Sequence

En los sumatorios se estudian solamente los top-k documentos más relevantes para la entrada x, lo que se define como:

$$z \in \text{topk}(p(\cdot|x))$$

2. RAG-Token

Al contrario que en *RAG-Sequence*, para generar la secuencia se va mirando cada uno de los *tokens*, es decir por cada uno se puede usar un documento distinto. Aquí el generador antes de marginalizar produce una distribución para el siguiente token para cada uno de los documentos recuperados:

$$p_{RAG-Sequence}(y|x) \approx \prod_i^N \sum_{z \in \text{top-k}(p(\cdot|x))} p_{\eta}(z|x) p_{\theta}(y_i|x, z, y_{1:i-1})$$

Figura 42: RAG-Token

2.5 Tecnología utilizada

Durante el desarrollo de este proyecto se han usado diferentes herramientas para construir el entorno de trabajo, se enumeran y explican a continuación.

2.5.1 Anaconda

Es una plataforma de código abierto para crear modelos de ciencia de datos o aprendizaje automático en Python o R. Tiene varios paquetes y herramientas preinstaladas y facilita la gestión de entornos virtuales y dependencias de paquetes a través de conda, mediante el uso de esta herramienta de línea de comandos se ha podido instalar un entorno para la ejecución del proyecto. [31] [32]

2.5.2 Python

Python es un lenguaje de programación interpretado de tipado dinámico, es multiparadigma ya que admite varias tareas como la orientación a objetos, la programación funcional o la programación procedimental. Python se caracteriza por tener una sintaxis sencilla, pero a su vez tiene una gran potencia. Este lenguaje también es portable, es decir, funciona en distintos sistemas operativos [33]. En este proyecto se utiliza la versión de Python 3.13.3.

2.5.3 CUDA (Compute Unified Device Architecture)

Es una plataforma de computación paralela desarrollada por NVIDIA que permite usar la GPU para acelerar tareas computacionales de alto rendimiento, como puede ser el entrenamiento de modelos de deep learning [34]. La versión de CUDA instalada en el entorno de trabajo es 12.8.

2.5.4 PyTorch

Es una librería dedicada a la creación de modelos de deep learning, una rama del aprendizaje automático. Está escrito principalmente en Python y por lo tanto se caracteriza por su enfoque de programación dinámica [35] [36].

Al usar algoritmos de aprendizaje automático se necesita que los datos estén representados de manera numérica, en PyTorch se consigue a través de los llamados tensores. Un tensor es similar a un array multidimensional y se utilizan para codificar las entradas, salidas y parámetros de un modelo, son similares a los ndarrays de NumPy con la salvedad de que los tensores se pueden ejecutar en GPU [37].

2.5.5 Jupyterlab

Es una aplicación de creación y edición de notebooks extensible y con muchas funciones, es parte del proyecto Jupyter. Es un entorno de desarrollo interactivo basado en web que permite trabajar con notebooks, código en vivo, visualización de datos, etc. Es una versión mejorada de Jupyter Notebook [38].

2.5.6 Notebook

Es un documento que combina código ejecutable, comentarios explicativos, visualización y otros elementos multimedia en una misma interfaz. Proporciona un entorno flexible e interactivo para el análisis de datos, la visualización y el desarrollo de prototipos de código [39].

2.5.7 Tensorboard

Tensorboard es un kit de herramientas que permite la visualización de pruebas para nuestro trabajo, se permite visualizar métricas como la pérdida en el entrenamiento y en la validación, visualizar la evolución de hiperparámetros como la tasa de aprendizaje, etc. [40]

Capítulo 3 – Metodología

Antes de explicar la metodología del proyecto, se desarrollan las principales características del entorno. El desarrollo de este trabajo se ha llevado en un entorno de desarrollo local con las siguientes especificaciones:

- **Sistema operativo:** Microsoft Windows 11 Home
- **Procesador:** Intel Core i7-12700K (20 CPUs), 3.6 GHz
- **Memoria RAM:** 32 GB RAM
- **Almacenamiento:** Disco HDD 1TB
- **GPU:** Nvidia GeForce RTX 4070, 12 GB VRAM

Como se comenta en el apartado anterior, el proyecto se ejecuta en un entorno de conda con Python 3.13.3. Algunas de las librerías más importantes para el desarrollo del trabajo son:

- datasets v-3.5.1 para la creación de Datasets.
- faiss-cpu v-1.11.0 para el uso de FAISS de langchain.
- langchain v-0.3.25 para tareas como la creación del índice FAISS para la recuperación y almacenamiento de *embeddings* y el uso del divisor (*splitter*) de texto.
- langchain-huggingface v-0.1.2 para el uso del modelo de *embeddings*.
- ocrmypdf v-16.10.1⁷ para aplicar OCR sobre los PDFs.
- optuna v-4.3.0 para la prueba de ajuste de los hiperparámetros del modelo.
- PyMuPDF v-1.25.5 para extraer el texto de los PDFs con OCR ya aplicado.
- PyTorch para el uso de tensores, se divide en tres (PyTorch se compila con CUDA 12.8):
 - torch, v-2.7.0+cu128
 - torchaudio v-2.7.0+cu128
 - torchvision, v-0.22.0+cu128
- sentence-transformers v-4.1.0 para medir la similitud de los *embeddings*.
- tqdm v-4.67.1 para visualizar barras de progreso en el proceso de OCR.

⁷ Para usar ocrmypdf hay que instalar en el entorno Tesseract y Ghostscript, se explica en

- transformers v-4.51.3 para crear pipelines, el uso del Tokenizer, el uso de la clase Trainer para el *fine-tuning* del modelo y el propio uso del LLM principal.

Para el visionado de resultados se utilizan las siguientes librerías:

- matplotlib v-3.10.3 para generar gráficos.
- numpy v-2.2.5 para la utilización de operaciones matemáticas.
- tensorboard v-2.19.0⁸ para el visionado de resultados de distintas pruebas.

3.1 LLM Utilizados

Para la selección de los modelos en el sistema RAG diseñado se han utilizado dos *benchmarks* reconocidos y públicos, ambos de Hugging Face:

- El Open LLM Leaderboard, que se encarga de evaluar modelos de lenguaje abiertos en tareas como la comprensión, la generación de texto, razonamiento de un contexto largo, etc. [41]
- El MTEB Leaderboard (*Massive Text Embedding Benchmark*), que compara modelos de lenguaje para generación de embeddings midiendo tareas como clasificación (incluyendo por pares y multi-etiqueta), reordenamiento, semejanza semántica textual (STS), etc. [42]

Debido a las limitaciones de hardware del entorno donde se ha realizado el proyecto, en particular a la cantidad de VRAM disponible en la GPU, no ha sido posible ejecutar modelos de gran tamaño (a partir de 2-3B de parámetros), ya que, en conjunto con el resto de la aplicación se sobrepasa la cantidad de memoria disponible. Por este motivo, el estudio de los modelos se ha reducido a modelos más ligeros, en torno a 1B de parámetros. Utilizando los *benchmarks*, se han seleccionado:

- Un modelo de lenguaje para generación de embeddings, “*multilingual-e5-large-instruct*”, se corresponde con el cuarto lugar de la siguiente tabla:

Rank	Model	Zero-shot	Memory	Number of Embeddings	Max Tokens	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)	Mean (Task)
1	[gemini-e-99%	Unknown	Unknown	3072	2048	68,37	59,59	79,28	71,82	54,59	5,18	29,16	83,63	65,58	67,71	79,4		
2	[Linq-Emt-99%	13563	7B	4096	32768	61,47	54,14	70,34	62,24	50,6	0,94	24,77	80,43	64,37	58,69	74,86		
3	[gte-Qwei-NA	29040	7B	3584	32768	62,51	55,93	73,92	61,55	52,77	4,94	25,48	85,13	65,55	60,08	73,98		
4	[multiling-99%	1068	560M	1024	514	63,22	55,08	80,13	64,94	50,75	-0,4	22,91	80,86	62,61	57,12	76,81		
5	[SFR-Embi-96%	13563	7B	4096	32768	60,9	53,92	70	60,02	51,84	0,16	24,55	80,29	64,19	59,44	74,79		
6	[GritLM-7199%	13813	7B	4096	4096	60,92	53,74	70,53	61,83	49,75	3,45	22,77	79,94	63,78	58,31	73,33		
7	[text-mult-99%	Unknown	Unknown	768	2048	62,16	54,25	70,73	64,64	47,84	4,08	22,8	81,14	61,22	59,68	76,11		
8	[GritLM-8199%	89079	57B	4096	4096	60,49	53,31	68,17	61,55	50,16	2,44	24,43	79,73	62,61	57,54	73,16		
9	[e5-mistr-99%	13563	7B	4096	32768	60,25	53,08	70,58	60,31	50,57	-0,62	22,2	81,12	63,82	55,75	74,02		
10	[Cohe-e-NA	Unknown	Unknown	1024	Unknown	61,12	53,23	70,5	62,95	46,89	-1,89	22,74	79,88	64,07	59,16	74,8		
11	[gte-Qwei-NA	6776	1B	8960	32768	59,45	52,69	62,51	58,32	52,05	0,74	24,02	81,58	62,58	60,78	71,61		
12	[bilineal-98%	2136	559M	1024	514	60,96	52,92	73,55	62,77	46,49	-3,04	22,36	79,83	61,42	55,1	77,81		

Figura 43: Ranking Embedding Leaderboard

⁸ Para la utilización de esta librería se ha creado un segundo entorno, se explica en el “Anexo A”.

Este modelo se utiliza para generar los embeddings de los textos a analizar, construir un índice FAISS (*Facebook AI Similarity Search*) el cual permite realizar búsquedas eficientes, y crear un *retriever* que sirve para recuperar los fragmentos de texto más importantes ante una consultada dada. Debido a estas características nuestro modelo está optimizado para tareas de recuperación y STS.

- Un modelo de lenguaje, que utilizamos tanto para generar pares pregunta-respuesta (QA) como para su posterior *fine-tuning* mediante dichos pares, “*TinyLlama-1.1B-Chat-v1.0*”. A pesar de su tamaño reducido (1.1B de parámetros), es adecuado para tareas de generación de texto y muy útil para ejecutarse en entornos con restricciones de hardware [43].

La elección de este modelo se basa en el equilibrio entre rendimiento y eficiencia, así como su gran popularidad dentro de la comunidad. Aunque en el leaderboard hay otros modelos con puntuaciones superiores, estas características han sido determinantes en la elección del modelo.

Para averiguar qué tipo de arquitecturas siguen estos modelos podemos ver su archivo *config.json* [44] [45] [46]:

1. *Embedding model*

El archivo json contiene la siguiente información:

```
{
  "_name_or_path": "tmp/",
  "architectures": [
    "XLNetModel"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "eos_token_id": 2,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-05,
  "max_position_embeddings": 514,
  "model_type": "xlm-roberta",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "output_past": true,
  "pad_token_id": 1,
  "position_embedding_type": "absolute",
  "torch_dtype": "float16",
  "transformers_version": "4.36.0",
  "type_vocab_size": 1,
  "use_cache": true,
  "vocab_size": 250002
}
```

Figura 44: *config.json* de *multilingual-e5-large-instruct*

Como se puede observar nuestro modelo de *embeddings* sigue la arquitectura de “xlm-Roberta”, para especificar la arquitectura exacta deberemos mirar el *config.json* de este modelo.

```
"architectures": [
  "XLNetForMaskedLM"
],
```

Con ver este trozo ya sabemos que se rige por *Masked Language Modeling* (Modelado de lenguaje enmascarado), el cual se centra en la idea de enmascarar *tokens* de la secuencia de entrada para que así el modelo pueda aprender a predecirlos, para esta predicción el modelo sigue un contexto bidireccional, en la que la predicción del *token* se basa tanto en los *tokens* anteriores como en los siguientes [47]. Estos modelos están pensados para seguir la arquitectura *encoder-only* (explicada anteriormente en el documento). Por tanto, “*multilingual-e5-large-instruct*” es un modelo basado en Transformers que sigue la arquitectura *encoder-only*.

2. Generator model

El archivo json en este caso contiene la siguiente información:

```
{
  "architectures": [
    "LlamaForCausalLM"
  ],
  "attention_bias": false,
  "bos_token_id": 1,
  "eos_token_id": 2,
  "hidden_act": "silu",
  "hidden_size": 2048,
  "initializer_range": 0.02,
  "intermediate_size": 5632,
  "max_position_embeddings": 2048,
  "model_type": "llama",
  "num_attention_heads": 32,
  "num_hidden_layers": 22,
  "num_key_value_heads": 4,
  "pretraining_tp": 1,
  "rms_norm_eps": 1e-05,
  "rope_scaling": null,
  "rope_theta": 10000.0,
  "tie_word_embeddings": false,
  "torch_dtype": "bfloat16",
  "transformers_version": "4.35.0",
  "use_cache": true,
  "vocab_size": 32000
}
```

Figura 45: *config.json* de *TinyLlama-1.1B-Chat-v1.0*

Este LLM en contraparte, usa *Causal Language Modeling* (Modelado de lenguaje causal), el cual sigue un contexto unidireccional, de izquierda a derecha concretamente, es decir, lo que hace es predecir el siguiente *token* teniendo en cuenta los que vengas por detrás, sin ver los siguientes [47]. ¿A qué recuerda esta idea? Efectivamente, a la arquitectura *decoder-only* (también explicada). “*TinyLlama-1.1B-Chat-v1.0*” (que se basa en modelos Llama, concretamente en Llama2) es, por tanto, un modelo basado en Transformers con una arquitectura *decoder-only*.

Algunos aspectos importantes que se pueden destacar de ambos archivos:

- “**hidden_act**”: especifica cual es la función de activación que se usa.
- “**intermediate_size**”: es el tamaño de la red FNN de cada bloque Transformer.
- “**max_position_embeddings**”: es la longitud máxima de *tokens* que el modelo puede manejar.

- **"num_attention_heads"**: es el número de “cabezas” del mecanismo *multi-head attention*.
- **"num_hidden_layers"**: es el número de capas formadas por bloques Transformer completos por las que atraviesa cada *token*.

3.2 Preparación de los datos para el entrenamiento

La preparación del dataset de entrenamiento comienza con la recuperación de los PDFs de estudio, a estos se les aplica una técnica conocida como OCR (*Optical Character Recognition*) Reconocimiento óptico de caracteres, se usa para procesar imágenes a texto interpretable por máquinas, las imágenes generalmente son documentos escaneados [48]. En el caso de este proyecto la mayoría de las cartas son escaneadas (algunas no y no necesitan de OCR, pero se controla mediante una opción para evitar trabajo innecesario) y por ende necesitan la aplicación de este proceso.

Una vez tenemos los PDFs con texto manejable, lo siguiente es dividir el texto de estos en fragmentos manejables (*chunks*) para su procesamiento, concretamente para *embeddings* y para la generación de pares de pregunta-respuesta. Se realiza mediante el uso de un divisor (*splitter*) de texto. Por último, para la creación del dataset de entrenamiento se siguen unos pasos:

1. Se crea la plantilla del dataset con un campo “context”, el cual contiene cada uno de los *chunks* divididos anteriormente. Es decir, por cada *chunk* distinto hay una fila distinta en el dataset. Da como resultado:

```
“context”: chunk1
“context”: chunk2
...
```

2. Se utiliza el modelo que actúa como *generator* para que, dado el contexto, genere un par pregunta-respuesta para ese contexto, en caso de que el modelo no sea capaz de generar ningún par, se queda como una entrada vacía (Puede parecer una práctica inútil el incluir un campo vacío, pero a la hora de mirar resultados ayuda a distinguir que pruebas son mejores o peores). Al final el dataset tiene el siguiente formato:

```
“context”: chunk1, “question”: consulta1, “answer”: respuesta1
“context”: chunk2, “question”: consulta2, “answer”: respuesta2
...
```

3. Ahora solamente queda filtrar las respuestas vacías o incorrectas y eliminar el contexto para que el modelo solamente vea pares pregunta-respuesta y aprenda relaciones y patrones sobre ellos. Para eliminar el contexto simplemente se usa un *prompt* para que el dataset final quede así:

Question: consulta1

Answer: respuesta1

Question: consulta2

Answer: respuesta2

3.3 Aplicación de RAG

Una vez se generan los pares qa y se aplica *fine-tuning* al modelo, es hora de realizar consultas al modelo y aplicar la técnica de RAG. Recordando su nombre completo dividimos en dos pasos esta técnica:

1. Retrieval (Recuperación)

Para la parte de recuperación se usa el modelo de *embedding*. Primero se convierten los *chunks* que habíamos obtenido del *splitter* a *embeddings*, se crea el índice FAISS y se asocian esos *embeddings* a los *chunks* originales, para saber cuál corresponde a cada uno.

Una vez se realizan dichas operaciones se crea el *retriever*, el encargado de convertir nuestra consulta a un *embedding* y comparar la posición en el espacio vectorial con los *embeddings* almacenados en FAISS, para así obtener los documentos más relevantes asociados a la consulta.

2. Augmented Generation (Generación aumentada)

El siguiente paso es cargar nuestro LLM y el tokenizador, los cuales nos permiten crear un pipeline para inferencia. Se crea un prompt que contiene unas leves instrucciones del comportamiento deseado que debe tener nuestro modelo y, además, se le pasa el contexto, el cual es un chunk recuperado usando el *retriever*, y la consulta en sí. Gracias al *pipeline* que se ha creado con el modelo y el tokenizador podemos generar la respuesta a través del *prompt*:

- Se tokeniza el *prompt* para que el LLM sea capaz de entender la entrada.
- El LLM genera la respuesta
- Se procesan los *tokens* a lenguaje natural, NLP

El *retriever* tiene establecido un límite de recuperación de 15 documentos, si ponemos más se exceden los *tokens* máximos que puede manejar el LLM, al menos con las pruebas que se han realizado para este trabajo. Para establecer el número de documentos, un *token* equivale a 4 caracteres en lengua inglesa [49]. Si tenemos en cuenta que los fragmentos tienen como máximo 512 caracteres que son unos 128

tokens, el *prompt* tiene unos 500 caracteres lo cual son 125 tokens, la pregunta más larga que se ha probado es de 107 caracteres que son 27 tokens y que el LLM tiene la capacidad de generar como máximo 500 nuevos *tokens*, para el cálculo de cuantos top-k documentos máximos se pueden recuperar, se puede utilizar esta fórmula:

$$\text{maxTokens} = \text{tokensSalida} + \text{prompt} + \text{consulta} + k * \text{tokensContexto}$$

$$k = \frac{\text{maxTokens} - (\text{tokensSalida} + \text{prompt} + \text{consulta})}{\text{tokensContexto}}$$

Figura 46: Cálculo de los top-k documentos

En el peor caso donde cada una de las variables tenga el valor máximo, k es $k \approx 11$, no obstante, como se ha explicado, tras realizar pruebas el máximo permitido es de $k = 15$.

Una vez generadas las respuestas, se aplica una función que convierte en tensores tanto a la consulta como a cada una de las respuestas, que compara la similitud en el espacio vectorial entre estas. También se le añade una pequeña puntuación a la longitud de la respuesta, para que no tenga en cuenta a preguntas muy similares, pero quizá demasiado cortas.

El modelo de RAG que se usa en este proyecto, como se puede observar, es una aproximación de *RAG-Sequence*, en nuestro modelo no se sigue la distribución marginal propia de RAG ni se fusionan las respuestas en una sola, pero la base es la misma. En el apartado **¡Error! No se encuentra el origen de la referencia.** se explica en profundidad este concepto.

Capítulo 4 – Experimentación y resultados

A lo largo de este capítulo se muestran todas las pruebas y resultados que se han ido elaborando hasta llegar al punto final donde se ha obtenido el modelo deseado. Se explica parte del código de cada prueba y también los pasos que se han tomado hasta llegar a la versión final.

4.1 Implementación de redes neuronales y modelo base

Este apartado está dedicado a la experimentación previa al uso del método RAG. En un primer instante se realizaron pruebas de creación de redes neuronales para tareas de clasificación y para acabar se obtiene un modelo base al que se le ha aplicado *fine-tuning* del que partimos para la aplicación posterior de RAG.

4.1.1 Clasificación de imágenes con una red FNN

Como primer ejemplo para familiarizarse con las redes neuronales se construye una red FNN con capas *fully connected* para clasificar imágenes del dataset MNIST. Dentro del código las secciones más importantes son:

1. Preprocesamiento

Se define una transformación para convertir las imágenes a tensores y normalizarlas en un rango de valores entre -1 y 1. Luego se descargan los conjuntos de datos MNIST de entrenamiento y validación, aplicando la transformación previa. Finalmente se crean los DataLoaders que cargan los datos en lotes de 64 imágenes para entrenamiento y validación.

```
transform = transforms.Compose([
    transforms.ToTensor(),
    # Se normaliza entre -1 y 1
    transforms.Normalize((0.5,), (0.5,))
])

# Descargamos el dataset de MNIST
train_dataset = torchvision.datasets.MNIST(root="./data", train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.MNIST(root="./data", train=False, transform=transform, download=True)

# Se crean DataLoaders para cargar nuestros datos en lotes
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=True)
```

2. Arquitectura de la red neuronal

Se define una red neuronal *fully connected* con la siguiente arquitectura:

- Flatten: Convierte la imagen de 28x28 píxeles en un vector de 784 elementos.
- Capa totalmente conectada (fc1): De 784 neuronas a 128.
- Función de activación ReLU: Introduce no linealidad en la red.
- Capa totalmente conectada (fc2): Reduce la dimensión de 128 a 10 (una neurona por cada dígito del 0 al 9).

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten() # Convertir 28x28 en un vector de 784
        self.fc1 = nn.Linear(28*28, 128) # Primera capa (Se pasa de 784 a 128)
        self.relu = nn.ReLU() # Función de activación ReLU
        self.fc2 = nn.Linear(128, 10) # Segunda capa (Se pasa de 128 a 10, números del 0 al 9)

    # Función para que se cumpla la propiedad de propagación hacia adelante
    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

3. Función de pérdida y algoritmo de optimización

Se define la función de pérdida (*Cross-Entropy*) y el algoritmo de optimización (SGD, en este caso con una tasa de aprendizaje de 0.01).

```
lossfunc = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

4. Función de entrenamiento

Primero se crea la función de entrenamiento que activa el modo de entrenamiento del modelo, después iteramos por cada época y cada lote de imágenes (movemos los datos a la GPU), los pasos de cada iteración son los siguientes:

1. Reiniciar los gradientes antes de cada actualización.
2. Generar predicciones con el modelo.
3. Calcular la pérdida comparando las predicciones con las etiquetas reales.
4. Retropropagación para calcular los gradientes.
5. Actualización de los pesos de la red neuronal.

Finalmente, se muestra la pérdida promedio de cada época

```
def train(model, dataloader, optimizer, lossfunc, num_epochs=5):
    # Modo de entrenamiento
    model.train()

    for epoch in range(num_epochs):
        epoch_loss = 0
        for images, labels in tqdm(dataloader, desc=f"Epoch {epoch+1}/{num_epochs}"):
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad() # Reiniciar gradientes
            outputs = model(images) # Hacer predicciones de las imagenes
            loss = lossfunc(outputs, labels) # Calcular la pérdida

            loss.backward() # Retropropagación
            optimizer.step() # Se actualizan los pesos

            epoch_loss += loss.item() # Acumulamos la pérdida

    # Pérdida promedio de cada época
    avg_loss = epoch_loss / len(dataloader)
    print(f"Epoch {epoch+1} | Loss: {avg_loss:.4f}")
```

5. Función de evaluación

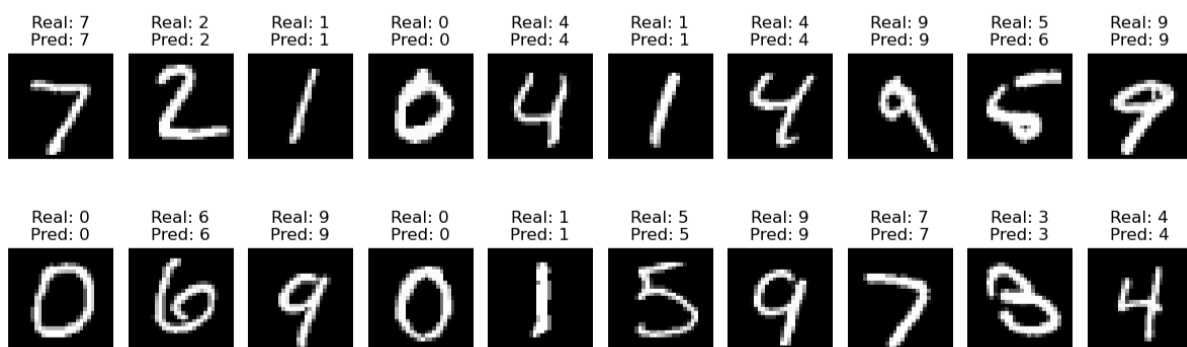
Dentro de esta función, se configura el modelo para que active el modo evaluación y con `with torch.no_grad()` se desactiva el cálculo de gradientes para ahorrar memoria. Se obtienen las predicciones del modelo, se calcula el total de muestras y el número de aciertos y finalmente se muestra el porcentaje de acierto junto con algunos ejemplos de imágenes en los que se muestra el valor real y el predicho por el modelo.

```
def evaluate(model, dataloader, device, num_images_to_show=10, images_per_row=10):  
    # Modo de evaluación  
    model.eval()
```

```
    with torch.no_grad():  
        for images, labels in dataloader:  
            images, labels = images.to(device), labels.to(device)  
  
            # Se hacen predicciones  
            outputs = model(images)  
            # Se obtiene la clase más probable para cada imagen  
            _, predicted = torch.max(outputs, 1)  
  
            total += labels.size(0)  
            correct += (predicted == labels).sum().item()
```

6. Resultados

Una pequeña muestra de los resultados:



4.1.2 Clasificación de imágenes con una red CNN

Es el mismo ejemplo que el anterior con la salvedad de que en este caso he utilizado una red CNN en vez de una FNN. Aquí el cambio importante en el código:

1. Arquitectura

Se cambia completamente la arquitectura para formar la nueva red CNN.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        # Capas convolucionales
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)

        # Capa de pooling, concretamente max_pooling
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Capas fully connected
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)

        # Función para que se cumpla la propiedad de propagación hacia adelante
    def forward(self, x):
        x = self.pool(func.relu(self.conv1(x))) # 1ª Conv + ReLU + Pooling
        x = self.pool(func.relu(self.conv2(x))) # 2ª Conv + ReLU + Pooling
        x = torch.flatten(x, start_dim=1) # Operación para conectar la convolucional y la fc
        x = func.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

2. Comparación de resultados

Si comparamos los resultados de una y otra, vemos que la CNN es más efectiva para MNIST, la razón principal de esto es que las imágenes tienen patrones espaciales importantes como pueden ser los bordes, las texturas, etc. que una FNN no aprovecha para nada, pero una CNN sí lo hace. La red FNN tiene menor precisión que la CNN, la razón es que esta es capaz de capturar las relaciones espaciales de las imágenes como bien se ha explicado antes en este documento.

En este caso el porcentaje de precisión no dista demasiado uno de otro, 95 frente a 100 (las pruebas de la red CNN me suelen dar entre 98-100%) pero aun así se mantiene que las CNN en estos casos son más efectivas.

4.1.3 *Fine-tuning* de un LLM basado en Transformers

En este ejemplo se utiliza como modelo GPT-2, un modelo basado en Transformers, concretamente en la arquitectura *decoder-only* [50] y para su entrenamiento utiliza el dataset “*wikitext-2-raw-v1*”, obtenido desde Hugging Face [51].

1. Carga del modelo

Se carga el modelo y el tokenizer, como se indica en el propio código se ha de definir el *token* de *padding* porque GPT-2 no tiene uno definido por defecto y en algunas funciones necesita uno.

```
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
# Se ha de definir un token de padding para que no de error
tokenizer.pad_token = tokenizer.eos_token

model = GPT2LMHeadModel.from_pretrained("gpt2")
model = model.to(device)
```

2. Carga del dataset

Primero se carga el dataset y se crea la clase WikiTextDataset para poder transformar el texto del dataset en *tokens* (función *init*), devolver la longitud del dataset (función *len*) y poder devolver los ids y de los tokens además de su máscara de atención (función *getitem*).

Esta clase permite crear un dataset manejable por *DataLoader*, que al final es el objeto que se usa para el entrenamiento. Quizás crear dicha clase no sea la solución óptima pues no se precisa de una logística avanzada, pero como se trata de una simple prueba de desarrollo, basta con ello.

```
# Filtrar textos no vacíos
train_texts = [item["text"] for item in dataset["train"] if item["text"].strip()]

dataset = WikiTextDataset(train_texts, tokenizer)
dataloader = DataLoader(dataset, batch_size=16, shuffle=True)
```

3. Función de entrenamiento y algoritmo de optimización

Se inicia el algoritmo de optimización, en este caso AdamW y se inicia el entrenamiento (**Captura 1**). El entrenamiento se realiza mediante la función definida, la cual recibe el DataLoader que hemos creado anteriormente. Activa el modo de entrenamiento y se itera por épocas y por lotes (dentro de cada época), dentro de las iteraciones, tal y como se indica en el propio código, GPT-2 calcula internamente la función de pérdida de entropía cruzada, *cross-entropy* (por lo que no hay que definirla), y se guarda (**Captura 2**). Se aplica el algoritmo de retropropagación y finalmente se mide la pérdida promedio en cada época.

Captura 1

```
optimizer = AdamW(model.parameters(), lr=5e-5)
train(model, dataloader, optimizer, num_epochs=3)
```

Captura 2

```
model.train()

for epoch in range(num_epochs):
    epoch_loss = 0
    for batch in tqdm(dataloader, desc=f"Época {epoch+1}/{num_epochs}"):
        input_ids, attention_mask = [b.to(device) for b in batch]
        labels = input_ids.clone() # Las etiquetas son iguales a las entradas

        # GPT-2 ya devuelve la pérdida, cuando se hacen las 2 primeras llamadas
        # se calcula el crossentropy internamente
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        # Se acumula la pérdida del mini-batch a la total de la época
        epoch_loss += loss.item()

    optimizer.zero_grad() # Reiniciar gradientes
    loss.backward() # Backpropagation
    optimizer.step() # Actualizar pesos
```

4. Consultas al modelo y generación de texto

Definimos la consulta que queremos realizar al modelo, la cual tokenizamos para que el modelo sea capaz de entenderla y llamamos a la función definida para generar la respuesta. Aquí el modelo se pone en modo evaluación y se desactiva el cálculo de gradientes, pues no se está entrenando el modelo, la salida se genera y se decodifica para transformarla de nuevo en lenguaje natural.

```
model.eval()
inputs = tokenizer(question, return_tensors="pt").to(device)

with torch.no_grad(): #Durante la generacion de texto no se calculan gradientes
    outputs = model.generate(
        inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        max_length=max_length,
        temperature=temperature,
        top_k=top_k,
        top_p=top_p,
        do_sample=do_sample,
        eos_token_id=tokenizer.eos_token_id,
        pad_token_id=tokenizer.eos_token_id
    )

    generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return generated_text

# Consulta a realizar
question = "What can you tell me about Valkyria Chronicles III?"

# Tokenizar la consulta
inputs = tokenizer(question, return_tensors="pt").to(device)

generated_text = generate_text(model, tokenizer, question, max_length=100)
```

5. Resultados

Como se puede observar, la respuesta no es muy buena, aunque va encaminada y el modelo distingue que efectivamente se trata de un videojuego, si observamos los valores de la pérdida durante el entrenamiento, lo que comúnmente se conoce como *train_loss*, se puede ver claramente que, aunque bajen, siguen siendo valores muy altos y quiere decir que el modelo no se está ajustando como debería a los ejemplos de entrenamiento [52].

Al generar preguntas, este modelo tiene dos vertientes: o se inventa completamente la respuesta o da una respuesta encaminada, aunque no correcta del todo, por tanto, el proceso de *fine-tuning* debe ajustarse mucho más, así como incluir técnicas de recuperación como RAG para una mayor precisión.

```
Época 1/3: 100% | 1486/1486 [02:06<00:00, 11.74it/s]
Época 1 | Pérdida: 2.6046
Época 2/3: 100% | 1486/1486 [02:04<00:00, 11.90it/s]
Época 2 | Pérdida: 2.3187
Época 3/3: 100% | 1486/1486 [02:05<00:00, 11.87it/s]
Época 3 | Pérdida: 2.1474
```

“Texto Generado (Respuesta):

Valkyria Chronicles III is an original video game developed and published by Square Enix . It is based on the true story of a girl who joins the fighting guild of an aristocratic family to fight for the Kingdom of Heaven . She ends up dying in the end , and thus inherits the title of Hero of Earth from her mother . As such , Hero is named after her after the game 's protagonist , Raiden , who is a legendary character in Final Fantasy VII .”

4.2 RAG

Una vez exploradas las bases de las redes neuronales y la técnica de *fine-tuning*, comenzamos a probar la implementación de RAG. La idea es realizar *fine-tuning* a un modelo mediante pares pregunta-respuesta que él mismo ha generado de la base de conocimiento, y que sea capaz de responder a las consultas del usuario mediante el uso de RAG, usando el *retriever* para recuperar los top-k documentos asociados a la consulta y elegir la mejor respuesta posible.

4.2.1 Modelo previo de RAG sin *fine-tuning*

En este ejemplo se ha desarrollado un modelo que se apoya en RAG a la hora de generar respuestas, su base de recuperación es el dataset “*databricks-dolly-15k*” [53]. No obstante, no se ha aplicado *fine-tuning* al modelo, es una primera toma de contacto con RAG. A continuación, se muestran los fragmentos más importantes del código.

1. Implementación del *retriever* y el LLM

El Dataset que se carga es un diccionario que contiene entradas del tipo:

```
{“instruction”: pregunta1, “context”: contexto1, “response”: respuesta1, “category”: tipoGen1}
```

Solamente nos interesa quedarnos con la parte de la pregunta y la respuesta, pues hay entradas en el Dataset original que no tienen un contexto proporcionado y así evitamos fallos a la hora de generar nuestra respuesta final, por ello se formatea el Dataset para quedarnos con estos dos campos y a partir de ahí se aplica el *splitter* para dividir los fragmentos en *chunks* y se inician el modelo de *embeddings*, el LLM, el Tokenizer y también se crea el pipeline de inferencia. En este ejemplo se usa como modelo de *embeddings* el “*sentence-transformers/all-MiniLM-L6-v2*”, todavía se estaba experimentando que modelo usar y es uno con mucha popularidad entre la comunidad de desarrollo [54].

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=150)
formatted_data = text_splitter.split_documents(formatted_data)

# Inicializamos el embedding model y creamos el indice FAISS y el retriever para la recuperación
e_model_name = "sentence-transformers/all-MiniLM-l6-v2"
embedding_model = HuggingFaceEmbeddings(model_name=e_model_name)

vectorstore = FAISS.from_documents(formatted_data, embedding_model)
retriever = vectorstore.as_retriever(search_kwargs={"k": 4})

# Inicializamos el LLM y el tokenizer
model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

model = AutoModelForCausalLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Se crea el pipeline de inferencia
gen_pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    device_map="auto"
)
```

3. Consultas mediante el uso de RAG

Se formula la consulta a realizar y se llama al *retriever* para que recupere los top-k documentos asociados a dicha consulta, todos ellos se unen en una variable *context*, la cual, junto a la consulta original, se usa para construir un *prompt* para que el modelo tenga claro la instrucción que debe seguir para generar la respuesta. Se tokeniza el *prompt* para que el modelo lo procese y genere una respuesta y finalmente se elimina de dicha respuesta toda la parte del contexto, pues no nos interesa.

```
# Consulta que queremos realizar
question = "What can you tell me about The Children's Investment Fund Management (TCI)?"

# Nos devuelve los top-k fragmentos de texto asociados a la pregunta, en nuestro código k=4
docs_retrieved = retriever.invoke(question)

# Se crea el contexto para la creación del prompt
context = " ".join([doc.page_content for doc in docs_retrieved])

# Se crea el prompt con un estilo de ChatBot
prompt = [
    {
        "role": "system",
        "content": "Answer the following question using the information given to you in the context. You have to respond in English"
    },
    {"role": "user", "content": f"Contexto: {context}\nPregunta: {question}"}
]

# Se tokeniza el prompt para que lo procese el modelo
tokenized_prompt = tokenizer.apply_chat_template(prompt, tokenize=False, add_generation_prompt=True)

# Generamos la respuesta
outputs = gen_pipeline(tokenized_prompt, max_new_tokens=256, do_sample=True, top_p=0.9)

# Una vez generada, se le elimina la parte del prompt del inicio para quedarnos solo con la respuesta
answer = outputs[0]["generated_text"][len(tokenized_prompt):].strip()
```

4. Resultados

A la consulta que aparece en el código, el modelo da la siguiente respuesta:

Respuesta: Thomas Jefferson (1743-1826) is an American statesman, philosopher, architect, and Founding Father who served as the third president of the United States from 1801 to 1809. He was the principal author of the Declaration of Independence, which declared the United States' independence from Great Britain.

Si consultamos el Dataset original [53], vemos que la entrada para esa pregunta concretamente es mucho más larga, pero la idea principal la recoge y la resume, por lo tanto, el modelo está realizando buen trabajo. Ante la pregunta: “What can you tell me about The Children's Investment Fund Management (TCI)?” El modelo ha respondido lo siguiente:

En el contexto, TCI es una fundación de inversión con sede en Londres y sede de administración en las Islas Canarias que busca apoyar la educación y la prevención del deterioro mental de la infancia en países ricos y bajos rendimientos. Su misión es aumentar el conocimiento sobre los medios de promover la educación y la prevención del deterioro mental en los niños y niñas de América Latina y el Caribe, y apoyar a los educadores, las autoridades y los estados para que sean capaces de realizar esos objetivos en cada uno de los países. La fundación utiliza la herramienta "iniciativa de la escuela", que utiliza un modelo innovador y efectivo para mejorar el desempeño escolar de niños en el área de las ciencias y el conocimiento acumulado en ciencias aplicadas. Esta iniciativa ayuda a que los alumnos sean más aptos para tomar exámenes, a aprender y a mantenerse en curso con más facilidad. TCI también lleva

Claramente se ha producido una alucinación, término que en inteligencia artificial viene a decir que el LLM produce una respuesta inventada, que puede producirse por varios factores [55], en nuestro caso el modelo ha confundido conceptos y se pensaba que estábamos hablando del CIFF (*Children's Investment Fund Foundation*), también se puede observar que el modelo ha llegado al límite de *tokens* máximos a generar y ha dejado una frase incompleta, lo cual no es una buena práctica. Para evitar que el modelo produzca alucinaciones el *fine-tuning* es un paso importante ya que focaliza al modelo en responder sobre temas exactos y de los que tiene cierto conocimiento.

4.2.2 Modelo completo (RAG + *Fine-tuning*)

Tras aprender cómo aplicar RAG y *fine-tuning*, es hora de ponerlo a prueba mediante la creación del modelo especialista, el cual era el objetivo principal del desarrollo de este trabajo. A continuación, se muestran los fragmentos más importantes del código desarrollado. El LLM utilizado para el desarrollo de este modelo es “*TinyLlama-1.1B-Chat-v1.0*” (se utiliza como *generator* para los pares pregunta-respuesta y en sí mismo es el modelo al que aplicamos *fine-tuning* para que luego genere las respuestas ante nuestras consultas) y el modelo de *embeddings* que utilizamos para crear el *retriever* es “*multilingual-e5-large-instruct*”. Para la mayor parte de este código se han utilizado como guía de desarrollo las siguientes fuentes [56] [57].

1. Recuperación de datos mediante OCR

Tanto la técnica en sí, como la aplicación de OCR se ha detallado en el apartado de “Metodología”. Lo que tenemos que definir básicamente es la ruta donde guardamos los PDFs de estudio y donde queremos que se guarden los nuevos a los que se les ha aplicado OCR, una vez aplicamos el OCR aprovechamos para guardar en una lista todos los textos completos para el uso posterior del *splitter* para obtener *chunks* procesables.

```
# Aplicación de OCR
for filename in tqdm(os.listdir(pdf_folder)):
    if filename.endswith(".pdf"):
        input_path = os.path.join(pdf_folder, filename)
        output_path = os.path.join(ocr_folder, filename)

        # skip_text hace que solo se aplique OCR si el pdf no contiene texto
        ocrmypdf.ocr(input_path, output_path, skip_text=True)

        # Se extrae el texto de los PDFs
        with fitz.open(output_path) as doc:
            text = ""
            for page in doc:
                text += page.get_text()
            ocr_texts.append(text)

# División del texto en fragmentos (chunks) para embeddings y la generacion de qa
# Tiene como tamaño máximo 512 caracteres y para no perder contexto, un overlap de 64
splitter = RecursiveCharacterTextSplitter(chunk_size=512, chunk_overlap=64)
# Lista que guarda los chunks
all_chunks = []
for text in ocr_texts:
    all_chunks.extend(splitter.split_text(text))
```

2. Creación del pipeline que actúa como *generator* en el modelo de RAG

Creamos un pipeline al que pasamos el LLM que actúa como el generador dentro del modelo de RAG. La elección del valor de “temperature” se explica en el apartado de resultados, al final de esta sección.

```
model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

generator = pipeline(
    "text-generation",
    model=AutoModelForCausalLM.from_pretrained(model_name, device_map="auto"),
    tokenizer=AutoTokenizer.from_pretrained(model_name),
    max_new_tokens=300,
    temperature=0.7,
    do_sample=True
)
```

4. Generación de pares pregunta-respuesta

Lo primero que tenemos es un dataset con un campo “context” que guarda cada uno de los *chunks* generados previamente (**Captura 1**). Para la generación de los pares utilizamos es una función en la que creamos un *prompt* (**Captura 2**) incluyendo las instrucciones que debe seguir el generador junto a dicho “context” para que el generador sea capaz de producir los pares deseados. El *prompt* como se puede observar no dicta que a “Question:” le haya de seguir “Answer:” pero el modelo es totalmente capaz de deducir su función y generarlo, de hecho, se realizó una prueba indicando también “Answer:” y los resultados fueron peores, aquí algún ejemplo:

```
{"context": "Aena owns a large amount of surplus land and solar is very efficient in Spain, so Aena has the \npotential to significantly reduce its electricity costs if it moves to an all renewable strategy for \nelectricity procurement. \n\nAena's commitment to move to 70pc self generation of electricity by 2030 is very good in \nprinciple but is too long term to allow any measure of accountability and does not work on \nthe urgent timetable the world needs. It also unnecessarily delays the significant financial", "question": "", "answer": "How does Aena's commitment to move to 70pc self generation of electricity by 2030 work on the urgent timetable the world needs?"}
```

```
{"context": "benefits of moving to self generation. We propose an additional target of 30-40pc of \nelectricity production to be self generated by 2025 with annual targets for each of the next \n5 years. \n\nAena's commitment to source all their electricity from existing renewable sources really does \nnothing to drive more renewable energy production and I discount this commitment. \n\n(iii) \nMinimise emissions of aircraft in landing and takeoff", "question": "", "answer": "Benefit: \nAena's commitment to source all their electricity from existing renewable sources would help drive more renewable energy production."}
```

Como se puede observar, el campo “question” siempre aparece vacío y el campo “answer” puede ser que contenga la pregunta o la respuesta, pero tampoco las dos.

Una vez se obtienen las respuestas generadas, se separan los campos “question” y “answer” del “context” para poder así formar el Dataset deseado de preguntas-respuestas (**Captura 3**). Una vez se obtiene el Dataset se pasa a formato JSON porque hay que formatear el conjunto algo más.

Captura 1

```
# Se crea la primera versión del dataset que solo contiene los chunks
ds = Dataset.from_dict({"context": all_chunks})
```

Captura 2

```
prompt = [
    (
        "You are a helpful assistant. Given the following context, "
        "generate a question and its corresponding answer.\n\n"
        f"Context:\n{context}\n\n"
        "Question:"
    )
    for context in batch["context"]
]
```

```
# Se generan los pares a partir del prompt usando el generador
outputs = generator(prompt, return_full_text=True)
```

Captura 3

```
for output in outputs:

    text = output[0]["generated_text"]

    # Esta condición sirve para separar la pregunta y la respuesta, para poder crear el diccionario correctamente
    # En caso negativo, simplemente se dejan los campos vacíos para evitar fallos
    if "Question:" in text and "Answer:" in text:
        question = text.split("Question:")[1].split("Answer:")[0].strip()
        answer = text.split("Answer:")[1].strip()

    else:
        question=""
        answer=""

    results.append({"question": question, "answer": answer})

return{
    "question": [r["question"] for r in results],
    "answer": [r["answer"] for r in results]
}
```

5. Proceso de *fine-tuning*

Lo primero que se hace es recuperar el JSON que hemos guardado anteriormente el cual contiene el Dataset, para cada par se comprueba que los campos no estén vacíos ni que existan malos patrones dentro de la respuesta. Después se formatean los pares para que aparezcan en una sola entrada de un diccionario dentro de la clave “text”. Se separa el 10% del Dataset para formar el conjunto de validación, el restante es el conjunto de entrenamiento (**Captura 1**).

Una vez tenemos el Dataset bien formateado y dividido en los dos conjuntos, pasamos a implementar el modelo y el tokenizador para tokenizar, valga la redundancia, el Dataset de manera que el modelo pueda procesarlo. Se utiliza la clase Trainer [58] para realizar el *fine-tuning* del modelo usando el Dataset y se guarda nuestro modelo *fine-tuned* (**Captura 2**). La elección del valor de “training_args” se explica en el apartado de resultados, al final de esta sección.

Captura 1

```
data = [e for e in data if is_valid(e)]

# Se formatean los datos para el fine-tuning
formatted_data = [
    {
        "text": f"### Question: {i['question']}\n\n### Answer: {i['answer']}"
    }
    for i in data
]

# Creamos el dataset final que va a servir para fine-tuning
dataset = Dataset.from_list(formatted_data)
# Se separa el 10% del conjunto para validación
dataset = dataset.train_test_split(test_size=0.1)
```

Captura 2

```
training_args = TrainingArguments(
    output_dir="./tinyllama-finetuned",
    per_device_train_batch_size=6,
    per_device_eval_batch_size=6,
    num_train_epochs=6,
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs/modeloFinal",
    report_to="tensorboard",
    logging_steps=10,
    save_total_limit=2,
    fp16=True,
)

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized["train"],
    eval_dataset=tokenized["test"],
    tokenizer=tokenizer
)
```

6. Creación del *retriever* para el uso de RAG

En este paso se crea el índice FAISS y el *retriever*, el cual utiliza dicho índice para la búsqueda de documentos mediante el uso de *embeddings*.

```
e_model_name = "intfloat/multilingual-e5-large-instruct"
embedding_model = HuggingFaceEmbeddings(model_name=e_model_name)

# Se crea el índice FAISS y el retriever
vectorstore = FAISS.from_texts(all_chunks, embedding_model)
retriever = vectorstore.as_retriever()
```

7. Creación del *pipeline* de inferencia

Mediante la carga del modelo *fine-tuned* y el tokenizador, se crea el *pipeline* de inferencia que nos va a permitir en un futuro próximo generar las respuestas a las consultas que realicemos al modelo.

```
model_path = "./llama-finetuned2"
model = AutoModelForCausalLM.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)

# Creación de la pipeline de inferencia
qa_pipeline = pipeline("text-generation", model=model, tokenizer=tokenizer)
```

8. Creación del *pipeline* de inferencia

Lo primero que hacemos es formular la consulta que queremos realizar, para generar la respuesta se utiliza una función que se encarga de recuperar los top-k documentos más relevantes para esa pregunta, mediante el uso del *retriever*, y generar una respuesta para cada documento ayudándose de un *prompt* (**Captura 1**). Dentro de esta función se aplica una limpieza a cada respuesta obtenida para eliminar tabulaciones, espacios innecesarios, saltos de línea que no aplican, etc. Una vez se tienen las respuestas se llama a otra función que compara la similitud semántica entre la pregunta realizada y la respuesta generada por nuestro modelo, y devuelve la mejor candidata de entre todas las posibles (**Captura 2**). Toda esta idea se explica en detalle en el apartado de “Metodología”.

Captura 1

```
docs = retriever.get_relevant_documents(question)

answers = []

# Para cada documento se genera un prompt que contiene las instrucciones específicas de comportamiento
# del modelo, el documento recuperado y la consulta
for i, doc in enumerate(docs[:k]):
    context = doc.page_content.strip()

    prompt = (
        f"You are an assistant specialized in analyzing letters written by the investment fund"
        f"TCI (The Children's Investment Fund) to various companies.\n"
        f"The following are excerpts from those letters. Your task is to carefully analyze the content"
        f"and answer the question based strictly on the information contained in them.\n\n"
        f"Do not repeat the same entities or phrases. Provide a concise, non-redundant answer.\n\n"
        f"Context:\n{context}\n\n"
        f"### Question: {question}\n\n"
        f"### Answer:"
    )

    # Se genera la respuesta usando el pipeline de inferencia que se ha creado antes, se le añade la misma
    # temperatura que tenía cuando generó los pares qa
    output = qa_pipeline(prompt, max_new_tokens=500, do_sample=True, top_p=0.7)[0]["generated_text"]
```

Captura 2

```
question_emb = embedding_model.embed_query(question)
question_emb = torch.tensor(question_emb)

score = []

for r in results:
    answer = r["answer"]
    # Al igual que con la pregunta, se crean embeddings de las respuestas
    answer_emb = embedding_model.embed_query(answer)
    answer_emb = torch.tensor(answer_emb)

    # Se calcula la similitud en el espacio vectorial entre pregunta y respuesta
    sim_score = util.cos_sim(question_emb, answer_emb).item()

    # Se normaliza la longitud de la respuesta, se elige 2000 porque tenemos establecido que el generador tenga max_new_tokens=500
    # y por la regla de que 1 token = 4 caracteres, es decir, como máximo una respuesta tendrá 2000 caracteres
    norm_length = min(len(answer)/2000, 1.0)

    # Se calcula la puntuación final teniendo en cuenta la ponderación de la longitud y se guarda junto con la respuesta
    combined_score = (1- length_weight) * sim_score + length_weight * norm_length
    score.append((combined_score, r))

# Devuelve la mejor según la puntuación máxima
best = max(score, key=lambda x: x[0])[1]
return best
```

4.3 Experimentos

En este apartado se va a explicar cómo ha sido el proceso de selección de los parámetros del modelo mediante diversas pruebas para su correcto funcionamiento.

1. Parámetro *temperature* para el pipeline “generator”

Este parámetro se usa en el *pipeline* que actúa como generador de pares pregunta-respuesta. Para ello se han ejecutado pruebas con 4 valores distintos: 0.3, 0.4, 0.5 y 0.7. Son valores por debajo de 1, lo cual significa que el modelo es más conservativo en cuanto a sus respuestas, no es tan creativo [59]. Dicho comportamiento es el que deseamos, pues nuestro modelo se encarga de la inversión financiera, no queremos ningún tipo de alucinación ni texto mal formado.

Para evaluar qué tan buenas han sido las pruebas me apoyo en un código especialmente diseñado para comprobar mediante tensores y su posición vectorial qué similitud guardan el contexto, la pregunta y la respuesta, todos entre sí, es decir contexto-pregunta, contexto-respuesta y pregunta-respuesta (En el **Anexo C** se explica en detalle). Estos han sido los resultados:

```
qa_pairs-0.3.json: Puntuación total=0.90, Promedio C-Q:0.87, Promedio Q-A=0.91, Promedio C-A=0.91
qa_pairs-0.4.json: Puntuación total=0.89, Promedio C-Q:0.87, Promedio Q-A=0.91, Promedio C-A=0.91
qa_pairs-0.5.json: Puntuación total=0.89, Promedio C-Q:0.87, Promedio Q-A=0.91, Promedio C-A=0.91
qa_pairs-0.7.json: Puntuación total=0.89, Promedio C-Q:0.87, Promedio Q-A=0.90, Promedio C-A=0.90
```

Como se puede observar, las puntuaciones para todas las temperaturas son prácticamente iguales menos para el valor $t = 0.3$ que tiene 0.01 más, lo lógico sería usar este valor, pero tras revisar algunos ejemplos manualmente, los valores 0.3, 0.4 y 0.5 contienen mucha repetición en sus respuestas, con repetición me refiero a qué en la misma respuesta se hace una pequeña introducción que es prácticamente igual a la pregunta, por tanto, se ha usado $t = 0.7$ que es el mejor valor.

2. Parámetro *max_length* para en la función “tokenize” en el apartado de *fine-tuning*

Este parámetro mide la longitud máxima que se puede manejar de texto tokenizado, para medir el valor de este parámetro nos hemos ayudado de un código “momentáneo” para comprobar cual es la máxima longitud de *tokens* que contiene una entrada:

```
lengths = [len(tokenizer(example["text"])["input_ids"]) for example in formatted_data]

print(f"Máximo tokens en un ejemplo: {max(lengths)}")
print(f"Promedio tokens en ejemplos: {sum(lengths) / len(lengths):.2f}")
print(f"Número de ejemplos mayores que 256 tokens: {sum(1 > 256 for l in lengths)} de {len(lengths)}")

Máximo tokens en un ejemplo: 308
Promedio tokens en ejemplos: 130.15
Número de ejemplos mayores que 256 tokens: 25 de 300
```

Por tanto, el valor que se establece es de 308 *tokens* para evitar errores.

3. Tamaño de mini-lote y número de épocas para la clase Trainer en el apartado de *fine-tuning*

Para medir el número de épocas idóneo, se han ejecutado dos pruebas: una con tres épocas y otra con seis (al ejecutar la de seis y fijarme en su valor de *train_loss* no tenía mucho sentido seguir probando más épocas):

- 3 épocas:

Epoch	Training Loss	Validation Loss
1	0.756400	0.735300
2	0.308900	0.765367
3	0.114900	0.848322

- 6 épocas:

Epoch	Training Loss	Validation Loss
1	0.795600	0.711181
2	0.325200	0.740867
3	0.145100	0.817520
4	0.082100	0.890066
5	0.054100	0.947150
6	0.033300	0.995089

Como nos interesa que nuestro modelo sea experto en las cartas de estudio, nos interesa que el “train_loss” baje lo máximo posible y, por tanto, cuantas más épocas mejor. Por ello el estudio se realiza con seis épocas. En cuanto al tamaño de lote, cuanto mayor valor tengamos, mayor paralelismo se produce y por tanto el entrenamiento es más rápido. Se han ido haciendo pruebas hasta que el hardware del entorno de desarrollo no ha dado más de sí y se producían errores de memoria. El valor máximo recogido ha sido de seis. Por tanto, queda así:

```
training_args = TrainingArguments(  
    output_dir="./tinylama-finetuned",  
    per_device_train_batch_size=6,  
    per_device_eval_batch_size=6,  
    num_train_epochs=6,
```


4. Tasa de aprendizaje, β_1 , β_2 , ϵ y decaimiento de pesos (*weight_decay*) de la clase Trainer

El hiperparámetro más importante de los mencionados a la hora de realizar *fine-tuning* es la tasa de aprendizaje, en una primera instancia se han usado 3 valores para ver cómo se comportaban: $1e-5$, $5e-5$ (el valor por defecto de la clase Trainer) y $1e-4$. Estos han sido los resultados obtenidos⁹:

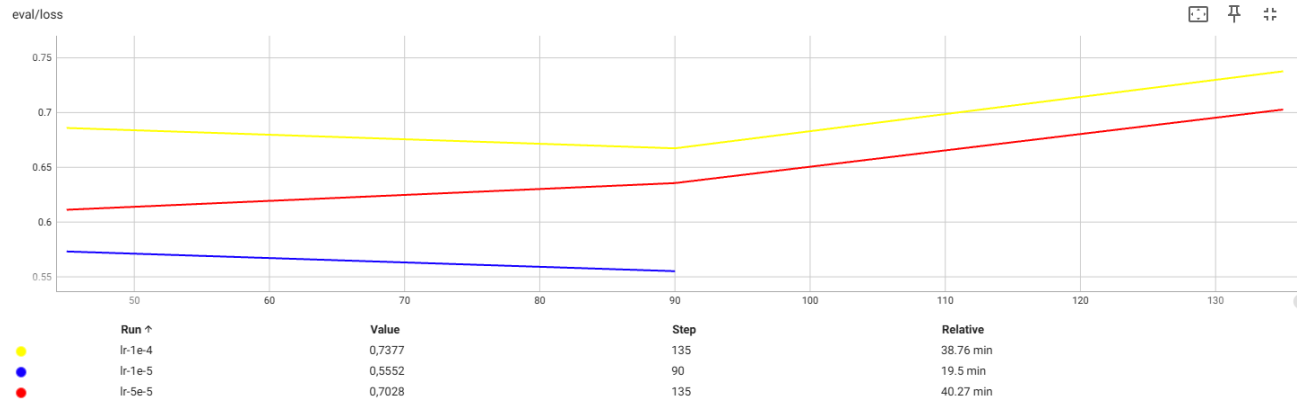


Figura 47: Eval_loss 3 tasas

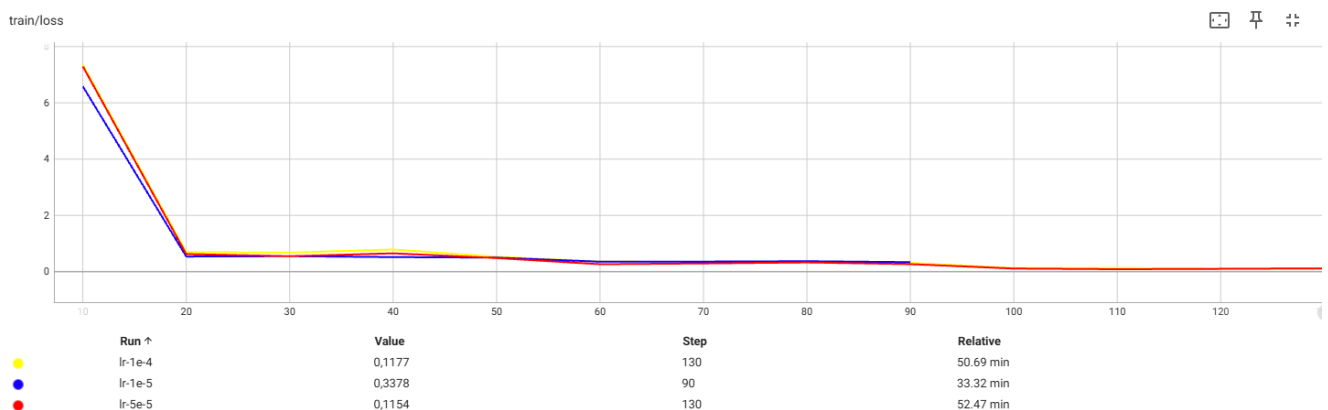


Figura 48: Train_loss 3 tasas

Como se puede observar, la pérdida para el conjunto de validación es un valor que sube, por más que pueda bajar un poco en las primeras etapas, acaba subiendo, lo cual quiere decir que nuestro modelo no generaliza bien datos que no haya visto durante el entrenamiento. Para el conjunto de entrenamiento, sin embargo, se observa una bajada casi hasta 0, lo cual quiere decir que está aprendiendo el conjunto de entrenamiento, casi memorizándolo. La diferencia entre ambos valores induce que nuestro modelo se está sobreajustando, en un primer momento se pensó que no era buena práctica y de ahí se produce el siguiente estudio.

Para que no se produzca sobreajuste me he ayudado del código que usa Optuna (En el **Anexo C** se explica en detalle) para la búsqueda de hiperparámetros, en el cual se puede especificar en qué métrica se centra el modelo y si se quiere maximizar o minimizar. En nuestro caso queremos que “*eval_loss*” se minimice para que no se produzca un sobreajuste.

⁹ Cabe decir que para $\lambda=1e-5$ solamente se ejecutaron 2 épocas porque los otros dos valores se ejecutaron primero y tras ver las gráficas, la idea era bastante clara y no se necesitaban más para corroborar lo que se pensaba.

También se tienen en cuenta otros hiperparámetros además de la tasa de aprendizaje, sabemos que nuestro modelo usa AdamW como algoritmo de optimización, el cual tiene β_1 , β_2 y ϵ . También recordemos que AdamW separaba la regularización L2 en un término para evitar los problemas de Adam, así que por tanto también se ha medido el decaimiento del peso (*weight decay*). Así quedan los rangos de valores que se han medido para estos hiperparámetros:

```
def optuna_hp(trial):
    """
    Función que establece un rango de valores para los hiperparámetros
    La "gracia" de Optuna es que permite probar cual es la mejor combinación de
    varios hiperparámetros y para ello se ha definir el rango de valores de estos

    Devuelve:
        La colección del rango de valores
    """
    return{
        "learning_rate": trial.suggest_float("learning_rate", 1e-8, 5e-5, log=True),
        "num_train_epochs": 2, # Valor fijo para que todos los intentos sean igual
        "adam_beta1": trial.suggest_float("adam_beta1", 0.9, 0.95),
        "adam_beta2": trial.suggest_float("adam_beta2", 0.98, 0.999),
        "adam_epsilon": trial.suggest_float("adam_epsilon", 1e-8, 1e-7, log=True),
        "weight_decay": trial.suggest_float("weight_decay", 0.0, 0.3)
    }
```

Por ejemplo, en la tasa de aprendizaje se ha puesto como valor máximo el valor por defecto que usaba la clase Trainer, porque tras la prueba anterior sabemos que un valor mayor es peor. En total se han ejecutado cinco intentos y estos han sido los resultados:

- Intento 1

Epoch	Training Loss	Validation Loss
1	0.528500	0.593350
2	0.273100	0.594866

Valores: Trial 0 finished with value: 0.5948659777641296 and parameters: {'learning_rate': 1.593197067858619e-05, 'adam_beta1': 0.9015699597048037, 'adam_beta2': 0.9931747393594664, 'adam_epsilon': 3.075748753032777e-08, 'weight_decay': 0.026394690076103743}.

- Intento 2

Epoch	Training Loss	Validation Loss
1	0.653500	0.693242
2	0.547200	0.654250

Valores: Trial 1 finished with value: 0.65425044298172 and parameters: {'learning_rate': 7.776815254439745e-07, 'adam_beta1': 0.9302826482942208, 'adam_beta2': 0.9907111923434436, 'adam_epsilon': 2.497371884703282e-08, 'weight_decay': 0.27991574724845913}.

- Intento 3

Epoch	Training Loss	Validation Loss
1	0.676200	0.710568
2	0.562800	0.666342

Valores: Trial 2 finished with value: 0.6663417816162109 and parameters: {'learning_rate': 6.301601699103278e-07, 'adam_beta1': 0.9221027473310615, 'adam_beta2': 0.9945867497055327, 'adam_epsilon': 2.0829953874796983e-08, 'weight_decay': 0.1676024185554322}.

- Intento 4

Epoch	Training Loss	Validation Loss
1	0.532200	0.594812
2	0.272200	0.595172

Valores: Trial 3 finished with value: 0.5951718091964722 and parameters: {'learning_rate': 1.6431260325572777e-05, 'adam_beta1': 0.9112318024994315, 'adam_beta2': 0.9903309250333571, 'adam_epsilon': 9.22302940754137e-08, 'weight_decay': 0.2817152730306327}.

- Intento 5

Epoch	Training Loss	Validation Loss
1	11.703000	9.944861
2	11.037000	9.887946

Valores: Trial 4 finished with value: 9.887946128845215 and parameters: {'learning_rate': 1.8274119730270958e-08, 'adam_beta1': 0.9105171556533699, 'adam_beta2': 0.980973877294153, 'adam_epsilon': 3.76519523058644e-08, 'weight_decay': 0.18197466861939804}.

Se pueden observar cuatro intentos más o menos factibles (Todos menos el último), aunque en dos de ellos (El 1 y el 4) comienza a subir la “eval_loss” y por tanto se descartan. Se ha realizado un promedio de los intentos 2 y 3 y se ha ejecutado un entrenamiento con dichos hiperparámetros y seis épocas. Estos han sido los resultados:

Epoch	Training Loss	Validation Loss
1	0.672600	0.649996
2	0.538300	0.592610
3	0.510300	0.577208
4	0.503000	0.569812
5	0.429900	0.566391
6	0.442800	0.565616

Los resultados no han sido del todo buenos, pues en torno a la cuarta época las pérdidas comienzan a estancarse e incluso en la última época se produce una subida. Si revisamos los argumentos de la clase Trainer [58], vemos que hay uno muy interesante, “lr_scheduler”. Para evitar que se produzca el sobreajuste se han probado dos tipos (todo esto siguiendo con la configuración de hiperparámetros que teníamos):

- “reduce_lr_on_plateau”

Epoch	Training Loss	Validation Loss
1	0.684400	0.617350
2	0.534400	0.560123
3	0.478300	0.537089
4	0.455600	0.527109
5	0.408400	0.522338
6	0.383600	0.526027

- “cosine”

Epoch	Training Loss	Validation Loss
1	0.665900	0.619940
2	0.542900	0.560486
3	0.526200	0.542012
4	0.464100	0.534744
5	0.475700	0.532094
6	0.436900	0.531652

Para verlo mejor nos apoyamos en las gráficas que nos proporciona el propio Tensorboard:

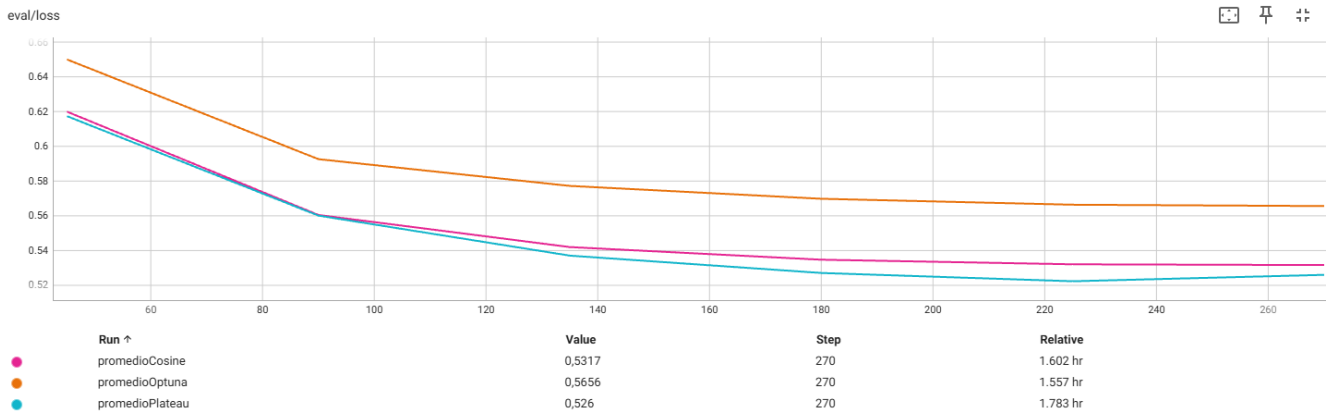


Figura 49: Eval_loss para las 3 pruebas con Optuna

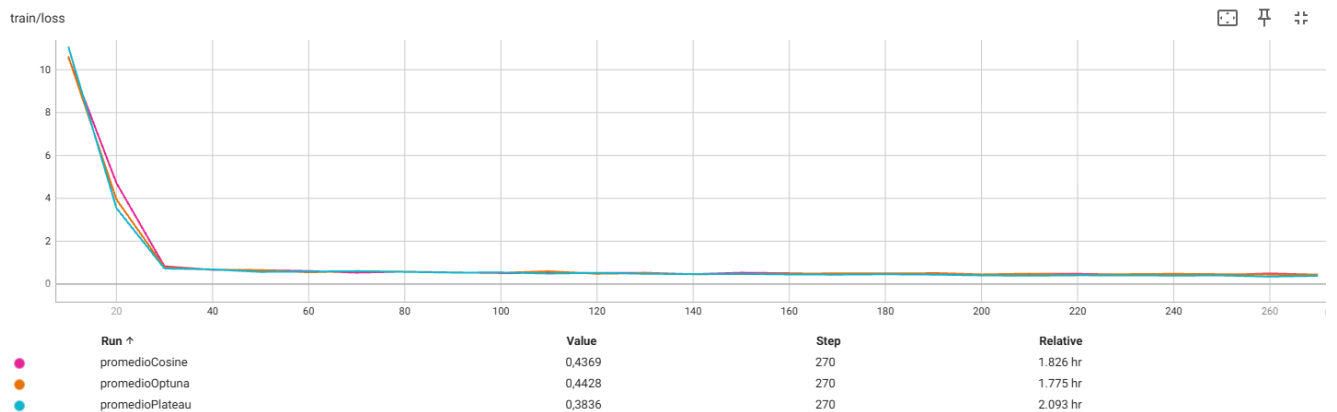


Figura 50: Train_loss para las 3 pruebas con Optuna

Si nos fijamos en la gráfica de la tasa de aprendizaje:

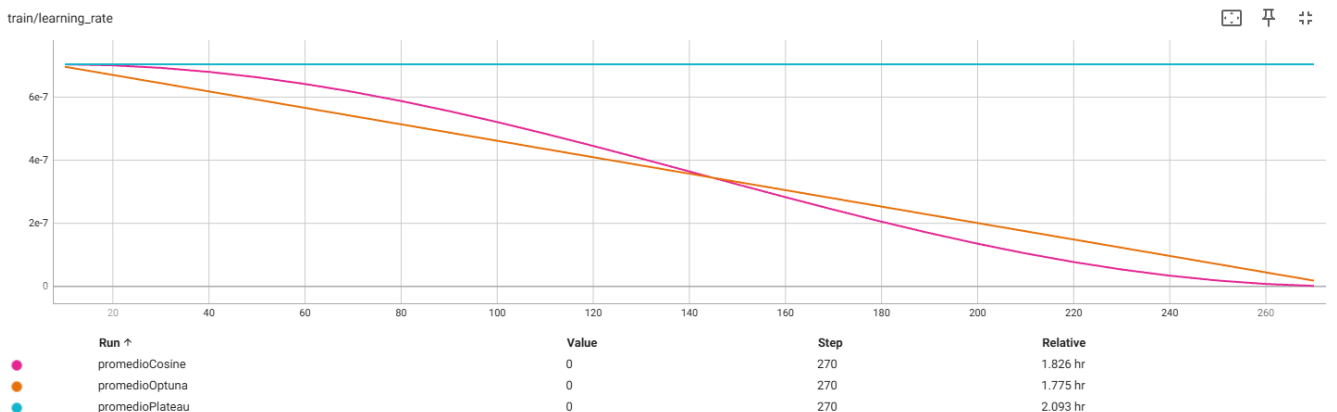


Figura 51: Tasa de aprendizaje para las 3 pruebas con Optuna

Podemos observar que la técnica “*reduce_lr_on_plateau*” no cambia su tasa, se debe a que el modelo no está detectando un empeoramiento en la métrica que se le ha pedido medir, en este caso “*eval_loss*”. El deterioro comienza en la última época, pero por eso precisamente no le da tiempo a aplicar su método. El mejor resultado, a la vista está, es el obtenido mediante el uso de *cosine* el cual permite que tanto *train_loss* como *eval_loss* bajen e incluso se estabilicen.

4.4 Resultados

Por último, solo nos queda probar como responde el modelo frente a las consultas que realicemos, para ello se han recogido muestras de los resultados, en el **Anexo C** se pueden ver con claridad. Si nos fijamos en la prueba que se ha realizado con el modelo generado mediante el $lr_scheduler = cosine$, hay algunas respuestas que no tienen mucho sentido, lo que ha producido el replantearse sobre la idea de que sobreajuste no es bueno para nuestro objetivo.

Para ello se han usado las pruebas donde no se ajustaban hiperparámetros (se dejaban por defecto) y se producía un ligero sobreajuste. Como se puede observar en el anexo, las preguntas son mucho más consistentes y guardan mayor relación con la pregunta y el contexto. También se puede observar que el modelo es capaz de responder mucho mejor si se entrena durante más épocas. Por lo tanto, la elección final de hiperparámetros, es dejar los valores por defecto que ya usa la clase Trainer ya que han producido un resultado óptimo. No interesa aumentar más la tasa de aprendizaje porque se puede correr el riesgo de que el modelo memorice tal cual los ejemplos de entrenamiento y sus respuestas sean iguales.

El modelo con seis épocas se volvió a construir para poder guardar los logs del entrenamiento y poder obtener las gráficas con Tensorboard:

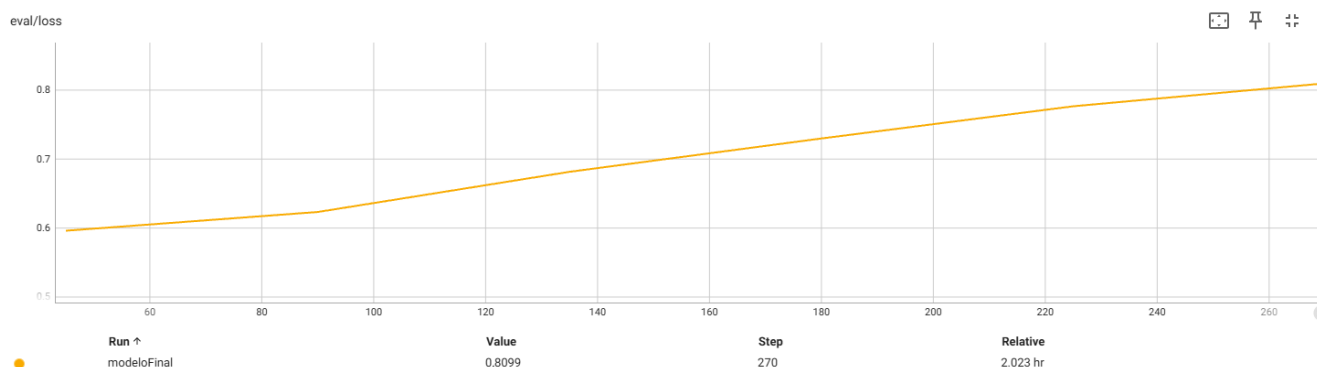


Figura 52: Eval_loss del modelo final

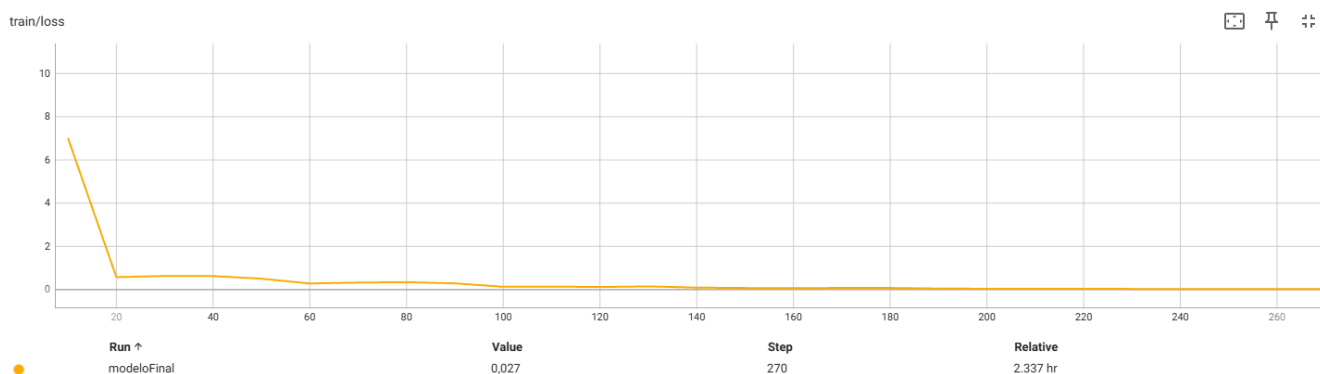


Figura 53: Train_loss del modelo final

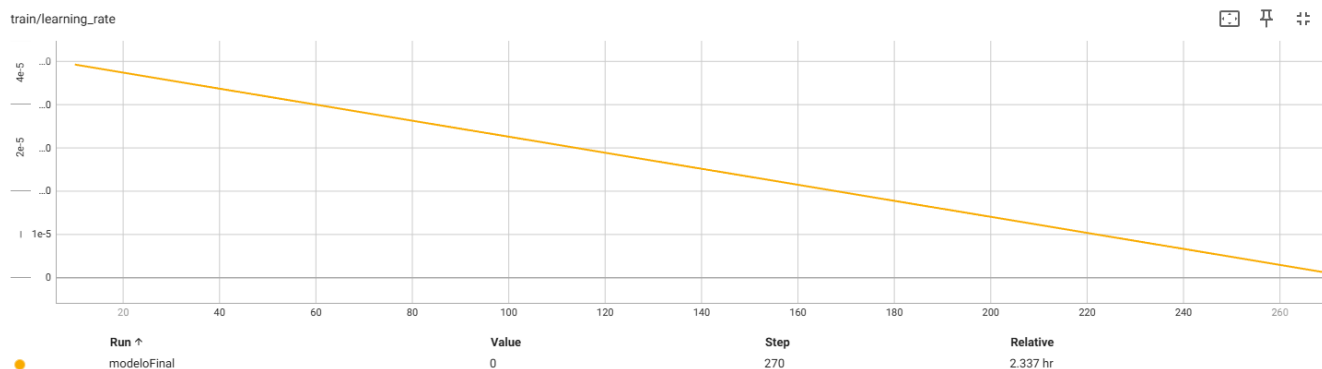


Figura 54: Tasa de aprendizaje del modelo final

La prueba final que aparece en el anexo se ha realizado con este modelo e incrementando el número de top-k documentos que recupera el *retriever* para ver si ayudaba al modelo a refinar su respuesta. Viendo los resultados, claramente mejora el uso de un mayor valor para k, el modelo tiene una mayor fuente de información y es capaz de generar respuestas mucho más precisas.

Capítulo 5 – Conclusiones

Tras las pruebas y estudios realizados para el desarrollo de dicho trabajo se ha conseguido obtener la versión *beta* deseada del modelo especialista en inversión financiera. El modelo es capaz de recuperar información sobre las cartas enviadas por el TCI y responder ante las consultas del usuario.

En vistas al futuro, se puede pulir esta versión llegando a obtener un asistente capaz de resolver dudas en un ámbito más general del ámbito financiero. Como mejoras se pueden utilizar modelos (tanto de *embeddings* como el generador) más potentes que tengan mayor capacidad de recuperación y mayor número de *tokens* manejables. También se pueden aplicar filtros más potentes a la hora de realizar el *fine-tuning* que permitan un aprendizaje totalmente supervisado, así como la introducción de retroalimentación por parte de especialistas en el ámbito de estudio para que el modelo se ajuste a las necesidades requeridas.

En la parte de RAG se pueden implementar ambas vertientes, *RAG-Sequence* o *RAG-Token*, de manera que sigan el enfoque original y no sean una versión adaptada, como es el caso de este trabajo. Para ello se pueden implementar técnicas de RetrievalQA, para unir todas las preguntas generadas por los top-k documentos en una sola mediante las tres técnicas diferentes (*map_reduce*, *refine* y *stuff*) [60]. También se puede introducir un *reranker* para ayudar a RAG a evaluar la respuesta en vez de utilizar los tensores y su similitud en el espacio vectorial (esta técnica es una especie de *reranker*, pero cuando se habla de estos normalmente nos referimos a la utilización de otro LLM externo). Estas técnicas se probaron durante el desarrollo del estudio, pero debido al hardware del entorno de trabajo no se llegaron a desarrollar en su totalidad.

Por tanto, en resumen, para el desarrollo de una futura mejora lo ideal es utiliza un entorno formado por varias GPUs con una VRAM amplia que permitan el despliegue de modelos más potentes y de técnicas que requieren de estas características.

En la experiencia personal, el desarrollo de este trabajo ha permitido la adquisición de conocimientos sobre inteligencia artificial, un campo que durante el transcurso del grado siempre me ha gustado pero que no había explorado en su totalidad. La realización de este estudio me ha permitido poner a prueba las aptitudes adquiridas durante estos años y puedo decir que estoy muy contento con el resultado final.

Bibliografía y Webgrafía

- [1] R. V. Aguilera, K. A. Desender y M. LopezPuertas-Lamy, «From universal owners to owners of the universe? How the big three are reshaping corporate governance,» *Corporate Governance: An International Review*, vol. 33, nº 3, pp. 462-482, 2025.
- [2] TCI Fund Management, «ESG Investment Policy,» [En línea]. Available: <https://www.tcifund.com/ESG>. [Último acceso: 2025].
- [3] M. Nielsen, «Neural Networks and Deep Learning,» Diciembre 2019. [En línea]. Available: <http://neuralnetworksanddeeplearning.com/>. [Último acceso: 2025].
- [4] IBM, «¿Qué son las redes neuronales?,» [En línea]. Available: <https://www.ibm.com/es-es/think/topics/neural-networks>. [Último acceso: 2025].
- [5] R. Pramoditha, «The Concept of Artificial Neurons (Perceptrons) in Neural Networks,» 26 Diciembre 2021. [En línea]. Available: <https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc/>. [Último acceso: 2025].
- [6] N. Lunge, «A Deep Architecture: Multi-Layer Perceptron,» 24 Marzo 2024. [En línea]. Available: <https://medium.com/@nlunge786/a-deep-architecture-multi-layer-perceptron-164bc5ff3842>. [Último acceso: 2025].
- [7] M. Chaudhary, «Activation Functions: Sigmoid, Tanh, ReLU, Leaky ReLU, Softmax,» 28 Agosto 2020. [En línea]. Available: <https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-relu-softmax-50d3778dcea5>. [Último acceso: 2025].
- [8] J. C. Olamendy, «Understanding ReLU, LeakyReLU, and PReLU: A Comprehensive Guide,» 4 Diciembre 2023. [En línea]. Available: <https://medium.com/@juanc.olamendy/understanding-relu-leakyrelu-and-prelu-a-comprehensive-guide-20f2775d3d64>. [Último acceso: 2025].
- [9] A. K. Pandey, «SiLU (Sigmoid Linear Unit) activation function,» 6 Abril 2024. [En línea]. Available: <https://medium.com/@akp83540/silu-sigmoid-linear-unit-activation-function-d9b6845f0c81>. [Último acceso: 2025].
- [10] a. K. D.Hendrycks, «Gaussian Error Linear Units (GELUs),» arXiv 1606.08415, 2023. [En línea]. Available: <https://arxiv.org/abs/1606.08415>. [Último acceso: 2025].
- [11] A. Yassin, «Adam vs. AdamW: Understanding Weight Decay and Its Impact on Model Performance,» 8 Noviembre 2024. [En línea]. Available: <https://yassin01.medium.com/adam-vs-adamw-understanding-weight-decay-and-its-impact-on-model-performance-b7414f0af8a1>. [Último acceso: 2025].
- [12] GFG Team, «Types of Neural Networks,» 25 Febrero 2025. [En línea]. Available: <https://www.geeksforgeeks.org/types-of-neural-networks/>. [Último acceso: 2025].

- [13] Y. B. A. C. I. Goodfellow, «Deep Learning,» Cambridge, Massachusetts: MIT Press, 2016, pp. 326-334.
- [14] J. Jordan, «Common architectures in convolutional neural networks.,» 19 Abril 2018. [En línea]. Available: <https://www.jeremyjordan.me/convnet-architectures/>. [Último acceso: 2025].
- [15] GFG Team, «Encoder Decoder Models,» 2 Mayo 2025. [En línea]. Available: <https://www.geeksforgeeks.org/encoder-decoder-models/>. [Último acceso: 2025].
- [16] U. Ankit, «Transformer Neural Networks: A Step-by-Step Breakdown,» 24 Mayo 2024. [En línea]. Available: <https://builtin.com/artificial-intelligence/transformer-neural-network>. [Último acceso: 2025].
- [17] R. Singh, «Types of Transformer Model,» 18 Octubre 2024. [En línea]. Available: <https://medium.com/@RobuRishabh/types-of-transformer-model-1b52381fa719>. [Último acceso: 2025].
- [18] C. R. Wolfe, «Decoder-Only Transformers: The Workhorse of Generative LLMs,» 4 Marzo 2024. [En línea]. Available: <https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse>. [Último acceso: 2025].
- [19] N. S. N. P. J. U. L. J. A. N. G. L. K. a. I. P. A. Vaswani, «Attention Is All You Need,» arXiv 1706.03762, 2023. [En línea]. Available: <https://arxiv.org/abs/1706.03762>. [Último acceso: 2025].
- [20] P. S, «Unleashing the Power of Autoencoders: Applications and Use Cases,» 4 Diciembre 2023. [En línea]. Available: <https://www.analyticsvidhya.com/blog/2023/05/unleashing-the-power-of-autoencoders-applications-and-use-cases/#h-layman-explanation-of-autoencoders>. [Último acceso: 2025].
- [21] GFG Team, «What are Language Models in NLP?,» 7 Junio 2024. [En línea]. Available: <https://www.geeksforgeeks.org/what-are-language-models-in-nlp/>. [Último acceso: 2025].
- [22] «Qué es el modelado del lenguaje y su relación con la PLN,» 9 Noviembre 2022. [En línea]. Available: <https://blog.pangeanic.com/es/modelado-del-lenguaje-y-relacion-con-pln>. [Último acceso: 2025].
- [23] K. Pykes, «Introducción al aprendizaje no supervisado,» 25 Abril 2024. [En línea]. Available: <https://www.datacamp.com/es/blog/introduction-to-unsupervised-learning>. [Último acceso: 2025].
- [24] IBM, «¿Qué es el aprendizaje no supervisado?,» 23 Septiembre 2021. [En línea]. Available: <https://www.ibm.com/es-es/think/topics/unsupervised-learning>. [Último acceso: 2025].
- [25] IBM, «¿Qué es el aprendizaje supervisado?,» 28 Diciembre 2024. [En línea]. Available: <https://www.ibm.com/es-es/think/topics/supervised-learning>. [Último acceso: 2025].

- [26] K. Wakefield, «A guide to the types of machine learning algorithms and their applications,» [En línea]. Available: https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html#:~:text=There%20are%20four%20types%20of,%2Dsupervised%2C%20unsupervised%20and%20reinforcement.. [Último acceso: 2025].
- [27] T. Xiao y J. Zhu, «Foundations of Large Language Models,» arXiv 2501.09223, 2025. [En línea]. Available: <https://arxiv.org/pdf/2501.09223>. [Último acceso: 2025].
- [28] H. Chawre, «Fine-Tuning LLMs: Overview, Methods, and Best Practices,» 21 Noviembre 2023. [En línea]. Available: <https://www.turing.com/resources/finetuning-large-language-models>. [Último acceso: 2025].
- [29] N. Miranda, «Introduction to RAG: A Novel Fusion of Parametric and Non-Parametric Memory,» 5 Enero 2024. [En línea]. Available: https://medium.com/@nelson.miranda_40644/introduction-to-rag-a-novel-fusion-of-parametric-and-non-parametric-memory-8fa10d5e07d3. [Último acceso: 2025].
- [30] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel y D. Kiela, «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,» arXiv preprint 2005.11401, 2021. [En línea]. Available: <https://arxiv.org/abs/2005.11401>. [Último acceso: 2025].
- [31] Anaconda, «Anaconda Distribution,» [En línea]. Available: <https://www.anaconda.com/docs/getting-started/anaconda/main>. [Último acceso: 2025].
- [32] Conda, «Conda Documentation,» 2017. [En línea]. Available: <https://docs.conda.io/en/latest/>. [Último acceso: 2025].
- [33] Python Software Foundation, «General Python FAQ - What is Python?,» 9 Junio 2025. [En línea]. Available: <https://docs.python.org/3/faq/general.html#what-is-python>. [Último acceso: 2025].
- [34] NVIDIA Development Team, «Cuda Toolkit - Free Tools and Training,» [En línea]. Available: <https://developer.nvidia.com/cuda-toolkit>. [Último acceso: 2025].
- [35] PyTorch, «PyTorch documentation,» [En línea]. Available: <https://docs.pytorch.org/docs/stable/index.html#pytorch-documentation>. [Último acceso: 2025].
- [36] Nvidia, «PyTorch,» [En línea]. Available: <https://www.nvidia.com/en-us/glossary/pytorch/>. [Último acceso: 2025].
- [37] PyTorch, «Tensors,» [En línea]. Available: https://docs.pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html. [Último acceso: 2025].
- [38] Jupyter Development Team, «JupyterLab Documentation,» 21 Junio 2016. [En línea]. Available: <https://jupyterlab.readthedocs.io/en/latest/>. [Último acceso: 2025].

- [39] Jupyter Development Team, «Project Jupyter Documentation,» 2015. [En línea]. Available: <https://docs.jupyter.org/en/latest/>. [Último acceso: 2025].
- [40] TensorFlow, «TensorBoard: el kit de herramientas de visualización de TensorFlow,» [En línea]. Available: <https://www.tensorflow.org/tensorboard?hl=es-419>. [Último acceso: 2025].
- [41] Hugging Face, «Open LLM Leaderboard,» [En línea]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard. [Último acceso: 2025].
- [42] Hugging Face, «<https://huggingface.co/spaces/mteb/leaderboard>,» [En línea]. Available: <https://huggingface.co/spaces/mteb/leaderboard>. [Último acceso: 2025].
- [43] Ajay, «TinyLlama 1.1B – Size Doesn’t Matter,» 22 Mayo 2025. [En línea]. Available: <https://www.analyticsvidhya.com/blog/2024/01/tinyllama-b-size-doesnt-matter/#h-potential-use-cases-and-applications>. [Último acceso: 2025].
- [44] TinyLlama, «TinyLlama-1.1B,» Hugging Face, [En línea]. Available: <https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0>. [Último acceso: 2025].
- [45] L. Wang, «multilingual-e5-large-instruct,» Hugging Face, [En línea]. Available: <https://huggingface.co/intfloat/multilingual-e5-large-instruct>. [Último acceso: 2025].
- [46] Facebook AI community, «xlm-roberta-base,» Hugging Face, [En línea]. Available: <https://huggingface.co/FacebookAI/xlm-roberta-base>. [Último acceso: 2025].
- [47] T. Vykruta, «Understanding Causal LLM’s, Masked LLM’s, and Seq2Seq: A Guide to Language Model Training Approaches,» 30 Abril 2023. [En línea]. Available: https://medium.com/@tom_21755/understanding-causal-llms-masked-llm-s-and-seq2seq-a-guide-to-language-model-training-d4457bbd07fa. [Último acceso: 2025].
- [48] Amazon Web Services, «¿Qué es el reconocimiento óptico de caracteres (OCR)?,» [En línea]. Available: <https://aws.amazon.com/es/what-is/ocr/>. [Último acceso: 2025].
- [49] OpenAI Team, «What are tokens and how to count them?,» Enero 2025. [En línea]. Available: <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>.
- [50] Hugging Face, «GPT-2,» [En línea]. Available: https://huggingface.co/docs/transformers/model_doc/gpt2. [Último acceso: 2025].
- [51] Salesforce, «Wikitext,» Hugging Face, 4 Enero 2024. [En línea]. Available: <https://huggingface.co/datasets/Salesforce/wikitext>. [Último acceso: 2025].
- [52] Baeldung, «Training and Validation Loss in Deep Learning,» 27 Noviembre 2024. [En línea]. Available: <https://www.baeldung.com/cs/training-validation-loss-deep-learning>. [Último acceso: 2025].

- [53] Databricks, «databricks/databricks-dolly-15k,» Hugging Face, [En línea]. Available: <https://huggingface.co/datasets/databricks/databricks-dolly-15k/viewer/default/train?views%5B%5D=train&row=10>. [Último acceso: 2025].
- [54] Sentence Transformers, «sentence-transformers/all-MiniLM-L6-v2,» Hugging Face, [En línea]. Available: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>. [Último acceso: 2025].
- [55] T. Farnschläder, «AI Hallucination: A Guide With Examples,» 27 Enero 2025. [En línea]. Available: <https://www.datacamp.com/blog/ai-hallucination>. [Último acceso: 2025].
- [56] LangChain, «Build a Retrieval Augmented Generation (RAG) App: Part 1,» [En línea]. Available: <https://python.langchain.com/docs/tutorials/rag/>. [Último acceso: 2025].
- [57] A. Upadhyay, «Implementing RAG with Langchain and Hugging Face,» 16 Octubre 2023. [En línea]. Available: <https://medium.com/@akriti.upadhyay/implementing-rag-with-langchain-and-hugging-face-28e3ea66c5f7>.
- [58] Hugging Face, «Trainer,» [En línea]. Available: https://huggingface.co/docs/transformers/main_classes/trainer. [Último acceso: 2025].
- [59] iguazio, «What is LLM Temperature?,» [En línea]. Available: <https://www.iguazio.com/glossary/llm-temperature/>. [Último acceso: 2025].
- [60] LangChain, «RetrievalQA,» [En línea]. Available: https://python.langchain.com/api_reference/langchain/chains/langchain.chains.retrieval_qa.base.RetrievalQA.html. [Último acceso: 2025].

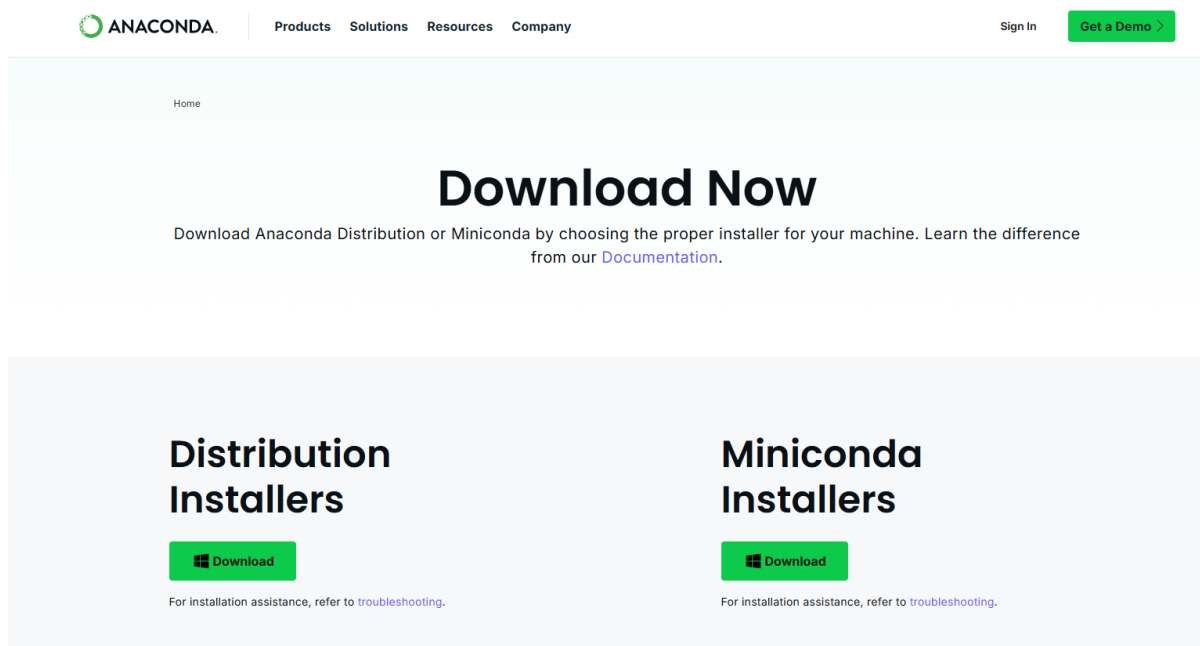
Anexos

Anexo A – Manual de instalación del entorno de estudio

A continuación, se explica en detalle los pasos seguidos para configurar el entorno de estudio y permitir así un adecuado desarrollo del trabajo.

1. Instalación de Anaconda

En la página oficial de Anaconda [31], podemos encontrar un apartado de descargas:



Se selecciona “*Distribution Installers*”, lo cual nos descarga un ejecutable que directamente instala Anaconda en nuestro equipo.

2. Creación y activación el entorno

Una vez tenemos instalado Anaconda, comprobamos que efectivamente nos ha instalado también Conda y Python, para ello abrimos en nuestro equipo Anaconda Prompt y ejecutamos estos comandos:

```
(base) C:\Users\David>conda --version
conda 24.11.3

(base) C:\Users\David>python --version
Python 3.12.7
```

Lo siguiente es crear nuestro entorno de trabajo (con una versión más reciente de Python) y una vez creado, activarlo, para ello ejecutamos lo siguiente:

```
(base) C:\Users\David>conda create -n tfg python=3.13.3
```

```
(base) C:\Users\David>conda activate tfg
(tfg) C:\Users\David>
```

A partir de aquí ya podemos trabajar en el entorno, pero antes de ponernos a trabajar necesitamos la interfaz sobre la que trabajar y el framework principal, PyTorch.

3. Instalación de PyTorch

En la página de PyTorch [35], hay una sección “Get Started” en la que podemos encontrar esto:

PyTorch Build	Stable (2.7.1)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Pip	LibTorch		Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.6	CUDA 12.8	ROCm 6.3
Run this Command:	pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu128			

Como se muestra en la imagen, he seleccionado las características específicas para mi equipo, en mi caso se soporta una versión de CUDA mayor, para comprobarlo hay que ejecutar el comando “*nvidia-smi*” en cmd:

```
NVIDIA-SMI 576.28              Driver Version: 576.28      CUDA Version: 12.9
```

Mi equipo soporta la versión 12.9, por tanto, para PyTorch escojo la mayor versión posible hasta la fecha de este estudio, la 12.8. Hay que ejecutar el comando que nos aparece en la página en nuestro entorno:

```
(tfg) C:\Users\David>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu128
```

Para ver si se ha instalado simplemente podemos ejecutar “*conda list torch*”.

4. Instalación e inicio de JupyterLab

Para instalar JupyterLab hemos de ejecutar en nuestro entorno el siguiente comando:

```
(tfg) C:\Users\David>conda install -c conda-forge jupyterlab
```

Para comprobar si se ha instalado, podemos probar a iniciarlo directamente con el comando *“jupyter lab”*.

5. Instalación de librerías y desarrollo del trabajo

Para instalar librerías podemos hacerlo directamente desde JupyterLab, si en la celda de un Notebook ejecutamos: *“pip install x”* donde x es el nombre de cualquier librería, se instalará directamente para nuestro entorno. Una vez tengamos todas las dependencias necesarias descargadas ya podremos comenzar a trabajar.

6. Instalación de Tensorboard

Como se ha comentado anteriormente en este documento, para el visionado de resultados en este trabajo, se ha utilizado, entre otras herramientas, Tensorboard, el cual necesita una versión de Python anterior a la 3.13 porque no son compatibles, para ello se ha tenido que instalar otro entorno:

```
(cleantfg-tb) C:\Users\David>python --version  
Python 3.12.11
```

Una vez estemos en el entorno adecuado se ha de ejecutar el comando:

“conda install -c conda-forge tensorboard”

Y para comprobar que funciona, podemos ejecutar:

```
(cleantfg-tb) F:\Proyectos TFG>tensorboard --logdir=logs  
TensorFlow installation not found - running with reduced feature set.  
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all  
TensorBoard 2.19.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Para ello tenemos que especificar el directorio donde hayamos guardado nuestros logs de la clase Trainer durante el *fine-tuning*. Una vez ejecutado el comando tendremos que ir a la dirección que se especifica: *“http://localhost:6006/”* y ya podremos iniciar nuestro estudio.

Anexo B – Archivos adjuntos

El código correspondiente a este proyecto se encuentra en mi OneDrive de la cuenta de la escuela:

- [TFG](#)

Dentro de este repositorio hay una jerarquía con diferentes directorios, se desglosa en:

1. **experimentación-previa:** durante la fase preliminar de pruebas, se realizaron diversas pruebas para tratar de entender como funcionaban los modelos de lenguaje y las técnicas de RAG y *fin-tuning*. En este directorio podemos encontrar los archivos:
 - **MNIST-FNN.ipynb:** es el notebook inicial, en él se construye una red FNN y se prueba la clasificación de imágenes sobre el dataset de *MNIST*.
 - **MNIST-CNN.ipynb:** es una variante del notebook inicial, se cambia el tipo de red por una CNN, pero la tarea sigue siendo exactamente la misma, la clasificación de imágenes sobre *MNIST*.
 - **LLM-Finetuning.ipynb:** es un notebook nuevo, en él se comienza la implementación de un LLM y además se prueba la técnica de *fine-tuning*. Para llevar a cabo el entrenamiento se utiliza el dataset de “*wikitext-2-raw-v1*”. En este caso el objetivo era probar a realizar consultas, tanto de datos contenidos en el dataset como datos nuevos, y ver como respondía con el uso del *fine-tuning*.
 - **RAG-SinFinetuning.ipynb:** en este notebook lo que se pretendía era entender el método RAG, para ello se usa un *retriever* y el modelo generador. El dataset que utilizamos como referencia es “*databricks/databricks-dolly-15k*”. En este caso la idea es realizar consultas sobre el dataset y que el *retriever* recupere la información relevante para la consulta, para que luego el generador produzca la respuesta.
2. **codigo-complementario:** dentro de este directorio se encuentran archivos utilizados como complementos al estudio realizado:
 - **BusquedaHiperparametros.ipynb:** en este archivo se utiliza Optuna para la búsqueda de hiperparámetros mediante la ejecución de varios intentos y un rango de valores.
 - **ComparacionPares.ipynb:** se utiliza para medir el valor de temperatura que se desea para la generación de pares pregunta-respuesta.
 - **Funciones.ipynb:** este notebook se ha utilizado para imprimir las gráficas de las funciones de activación para el apartado de redes neuronales.

3. **pares-qa:** se encuentran los pares pregunta-respuesta cada uno generado con una temperatura diferente. Cada archivo tiene el nombre qa_pairs-0.x.json donde x son los diferentes valores para la temperatura, en el caso de estudio de este trabajo son: 3,4,5 y 7.
4. **pdfs:** este directorio simplemente contiene las cartas redactas por el TCI, de las cuales se desea extraer información.
5. **logs:** aquí se encuentran los logs generados durante las pruebas de *fine-tuning* con distintos hiperparámetros, podemos encontrar:
 - **lr_1e-4:** tasa de aprendizaje de 1e-4.
 - **lr_1e-5:** tasa de aprendizaje de 1e-5.
 - **lr_5e-5:** tasa de aprendizaje de 5e-5.
 - **optuna:** prueba completa con Optuna, recoge cinco intentos.
 - **promedioOptuna:** promedio de los hiperparámetros de los dos mejores intentos.
 - **promedioPlateau:** con el mismo promedio anterior se introduce el parámetro `lr_scheduler = "reduce_lr_on_plateau"`.
 - **promedioCosine:** exactamente lo mismo, pero con `lr_scheduler = "cosine"`.
 - **modeloFinal:** registro de versión final del modelo desarrollado.
6. **tinylama-finetuned:** es el modelo con *fine-tuning* aplicado.
7. **RAG-Final.ipynb:** es la última versión de código que se ha obtenido durante el proceso de desarrollo.

Anexo C – Preguntas recogidas en las diferentes pruebas

En este anexo se recogen las distintas pruebas de generación realizadas con las diferentes versiones del modelo especialista buscado.

Prueba 1. Hiperparámetros ajustados, “lr_scheduler” = cosine, k=15

1. What aspects concern the investor (TCI Fund)?

The investor is concerned about the company's carbon emissions disclosure, which is significant to the company. The letter also states that TCI has a long-standing investment in Canadian Pacific which raises credibility and credibility for the disclosure. In summary, TCI is concerned about the company's emissions management strategy.

2. What are the reasons for the concerns that the TCI Fund has expressed?

The concern is that the company has not provided a detailed description and analysis of the effectiveness of the actions taken by TCI Fund Management Limited, including those relating to climate change risk. The company has also not disclosed any specific initiatives or plans for addressing climate change-related risks.

3. What average ownership percentage does TCI have in the contacted companies?

TCI owns around 8% of the following Canadian companies: 1. Canadian Pacific (CP): 7.3% 2. Suncor Energy (SU): 6.4% 3. Shell E&P Canada (SHP.K): 6.4% 4. Canadian Tire (CTI): 5.3% (a.k.a. TCI Fund Management Limited) 5. Imperial Oil (IMO.A): 4.6% 6. Enbridge (ENB.A): 3.7% 7. Baytex Energy (BTE.A): 3.4% 8. Cenovus Energy (CO.A): 3.2% 9. Enlink Energy (ELE.A): 2.9% 10. Imperial Oil (a.k.a. TCI Fund Management Limited)
Sources: 1. “Corporate Disclosure,” TCI Fund Management, retrieved January 2022. 2. “Shareholder Information,” Canadian Pacific Corp., retrieved January 2022.

4. What positive aspects does TCI highlight in its communications as achievements of the contacted companies?

The positive aspects of these companies that TCI has highlighted in its communications include, but are not limited to: 1. Promoting sustainability in telecom companies 2. Encouraging the adoption of digital roads and traffic management systems 3. Launching green bonds successfully

Conclusion: Based on the information presented, we can confirm that the contacted companies have been positively engaged with the TCI fund's sustainability approach, particularly in promoting the adoption of digital roads and traffic management systems, green bonds, and sustainable development.

5. What main areas for improvement does TCI identify in the communications about the contacted companies?

The communications about the contacted companies address the main areas identified by TCI. The communication should clearly demonstrate that the company has made significant progress on the areas identified by TCI. There should be a clear action plan for implementing the improvements identified.

6. What actions or strategies does TCI encourage the contacted companies to implement?

While TCI encourages the contacted companies to respond to CDP's Climate Change questionnaire, TCI encourages the contacted companies to respond to CDP's Climate Change questionnaire for

assessment, as all other TCI portfolio companies do. Therefore, TCI does not provide specific actions or strategies to the contacted companies for implementation.

7. Who TCI has written these letters to? List the companies

TCI has written these letters to any company within its investment mandate that is a subsidiary of a listed entity. Specifically, they are writing these letters to any company within their investment mandate that is a subsidiary of Charter Communications Inc., a communication services company.

8. What does the TCI Fund demand of the Spanish Minister of Ecological Transition in its communications?

The TCI Fund demands of the Spanish Minister of Ecological Transition the creation of GHG emissions reduction targets and GHG emissions reduction targets by Spanish energy companies. In response to the letter, Mr. Baring stated that the TCI had also asked the Spanish Minister to ensure that the Spanish Energy Regulator, the Competition Authority, the Agencia Española de Calidad del Espacio and the Agencia Espanola de Producción Audiovisual (AEPA) had been informed of the fund's view and of its concerns regarding GHG emissions and the effectiveness of any such policies implemented.

Charter Communications Response

Dear Chris,

We would appreciate your clarification of the TCI demand. Does the TCI Fund demand that Spanish energy companies have GHG emissions reduction targets and GHG emissions reduction targets by Spanish energy companies? If yes, please clarify which specific GHG emissions reduction targets are required. If the TCI Fund demands that Spanish energy companies have GHG emissions reduction targets and GHG emissions reduction targets by Spanish energy companies, please specify which type of targets is required.

Prueba 2. Hiperparámetros por defecto, 3 épocas y k=3

Se usó k=3 para comprobar rápidamente si era mejor con tres épocas o con seis. También se ejecutaron varias pruebas.

1. What aspects concern the investor (TCI Fund)?

1ª: The TCI Fund is particularly concerned about the company's greenhouse gas (GHG) emissions, which will have a material effect on a company's long-term profitability, sustainability and investor returns.

2ª: TCI Fund is invested in corporates which have a history of setting carbon reduction targets.

2. What are the reasons for the concerns that the TCI Fund has expressed?

1ª: The investor (TCI Fund) is concerned about the company's GHG emissions, particularly its climate change risk. The reason for this concern is that the company has set a target to reduce its GHG emissions by 30% by 2030.

I hope this helps! Let me know if you have any further questions.

2ª: TCI is concerned about the impact of climate change on a company's long-term profitability, sustainability and investor returns. Specifically, TCI believes that climate change-related risks, in particular a company's greenhouse gas (GHG) emissions, will have a material effect on a company's long-term profitability, sustainability and investor returns. These risks include regulation, taxation, competitive disadvantage, brand impairment, financial asset impairment and litigation.

3. What average ownership percentage does TCI have in the contacted companies?

1ª: As of the end of the second quarter of 2021, TCI owns around 3% of Canadian National.

2ª: TCI has a 15% average ownership shareholding in the contacted companies.

4. What positive aspects does TCI highlight in its communications as achievements of the contacted companies?

1ª: TCI is impressed to see the progress made by the companies in addressing climate change. TCI believes that the positive aspects should be considered in addition to the negative. We believe that the communication from TCI is a positive example for other companies.

2ª: TCI highlights the positives in its communications as achievements of the contacted companies, particularly concerning the environment and climate change.

5. What main areas for improvement does TCI identify in the communications about the contacted companies?

1ª: TCI identifies four areas for improvement in the same communications about the contacted companies:

1. Climate change disclosure
2. GHG emissions reduction targets
3. A low carbon transition plan
4. Company responsibilities for global warming TCI requires all its portfolio companies to issue a sustainability report, respond to CDP, and have GHG reduction goals. Charter must do the same to meet TCI's requirements.

2^a: The main areas for improvement identified by TCI in the same communications about the contacted companies are their ESG plans.

6. What actions or strategies does TCI encourage the contacted companies to implement?

1^a: TCI encourages the contacted companies to implement the following actions or strategies:

- a) Publicly disclose their emissions and have a credible plan for their reduction;
- b) Develop a sustainability strategy that aligns with TCI's values and targets, and implement it in the business and supply chain;
- c) Source low carbon energy through direct generation or power purchase agreements;
- d) De-carbonise transport fleets, e.g. Through electric vehicles.

In summary, TCI encourages the contacted companies to implement the following actions or strategies: publicly disclose their emissions and have a credible plan for their reduction, develop a sustainability strategy that aligns with TCI's values and targets, source low carbon energy through direct generation or power purchase agreements, and de-carbonise transport fleets, via electric vehicles.

2^a: TCI encourages the contacted companies to implement the following actions.

- 1. Introduce efficient energy management into buildings and factories.
- 2. Source low carbon energy through direct generation or power purchase agreements.
- 3. De-carbonise transport fleets, e.g. Through electric vehicles.
- 4. Offset emissions from corporate travel, e.g. Through afforestation.
- 5. Participate in community-based energy projects, e.g. Through the Gowanus Energy Project.
- 6. Reduce emissions from corporate events, e.g. Through event planning.
- 7. Publish an annual sustainability report on the company website.
- 8. Continue to invest in the company's sustainability efforts, e.g. Through employee involvement.
- 9. Overall, TCI encourages the contacted companies to implement measures that align with the TCFD disclosure recommendations.

7. Who TCI has written these letters to? List the companies

1^a: The letters addressed to are: Chris Hohn, Philip Green, and Alex Baring.

I hope this helps. Let me know if you have any further questions.

2^a: The persons addressed by the letters are Chris Hohn and Philip Green

8. What does the TCI Fund demand of the Spanish Minister of Ecological Transition in its communications?

1^a: The Spanish Minister of the Environment asks for urgent action to be taken to address climate change.

2^a: The Spanish Minister of the Environment demands that the "Climate Change Law Project" be imposed on the Spanish Government, which is the latest draft of the project.

Prueba 2. Hiperparámetros por defecto, 6 épocas y k=3

Se usó k=3 para comprobar rápidamente si era mejor con tres épocas o con seis. También se ejecutaron varias pruebas.

1. What aspects concern the investor (TCI Fund)?

1^a: The TCI Fund's investment approach and engagement consider the company's ESG factors, particularly climate-related risk, when assessing its investment opportunities.

2^a: The TCI Fund's investment process assesses a range of ESG factors, particularly climate change-related risks. We believe that climate change-related risks, in particular a company's greenhouse gas (GHG) emissions, will have a material effect on a company's long-term profitability, sustainability and investor returns. These risks include regulation, taxation, competitive disadvantage, brand impairment, financing, physical asset impairment and litigation.

2. What are the reasons for the concerns that the TCI Fund has expressed?

1^a: TCI believes that the main reason that concerns the investor (TCI Fund) is that it is not aligned with TCI's vision of a low-carbon investment environment. The TCI Fund's investment approach and engagement are inconsistent with TCI's commitment to reducing its carbon footprint and promoting sustainable investment practices.

2^a: TCI Fund's investment process assesses a range of ESG factors, particularly climate change risk. The firm believes that climate change-related risks, in particular a company's greenhouse gas (GHG) emissions, will have a material effect on a company's long-term profitability, sustainability and investor returns. These risks include regulation, taxation, competitive disadvantage, brand impairment, financing, physical asset impairment and litigation.

3. What average ownership percentage does TCI have in the contacted companies?

1^a: We own around 4% of the shares of TCI Fund Management Limited.

2^a: TCI has an average ownership percentage of 2% in the contacted companies.

4. What positive aspects does TCI highlight in its communications as achievements of the contacted companies?

1^a: We welcome the opportunity to meet with companies to discuss their ESG strategies and progress. TCI's communication with companies is positive, with a range of companies reporting increased annual commitment to ESG. We believe that these commitments are a testament to their companies' commitment to sustainability and value creation.

2^a: TCI provides positive aspects of its communications as achievements of the contacted companies. In its most recent communication, TCI highlighted the company's commitment to reducing its carbon emissions to net-zero by 2050. The communication stated that the company had set science-based targets to reduce its carbon emissions, which were aligned with international climate goals.

5. What main areas for improvement does TCI identify in the communications about the contacted companies?

1^a: TCI identifies the main areas for improvement in the communications about the contacted companies as:

1. Insufficient disclosure of GHG emissions and other climate-related information to investors.
2. Unawareness of the impact of the company's decisions on climate change.
3. Failure to implement a low carbon transition plan.
4. Inconsistency in reporting and disclosure of GHG emissions.
5. Refusal to disclose data and metrics used to evaluate the company's performance on climate change.

We believe that these issues need to be addressed by the companies to ensure proper disclosure and integration of their decisions related to climate change into their oversight frameworks.

2^a: TCI identifies two areas of improvement in the communications about the contacted companies:

1. Inclusion of climate change-related information in the communication about the company's ESG opportunities and risks.
2. Management of environmental impacts through the use of the Clean Skies for Tomorrow Coal Briefings.

6. What actions or strategies does TCI encourage the contacted companies to implement?

1^a: TCI encourages the contacted companies to take the following actions:

1. Implement stronger environmental and energy management systems to monitor and report on emissions.
2. Develop and implement a credible plan for reducing emissions, including setting targets and timelines.
3. Monitor and measure the progress towards reducing emissions, and adjust our investment strategy accordingly.
4. Engage with the company to ensure that the climate change risks and opportunities related to material ESG are being managed effectively.
5. Provide regular updates to TCI on the company's ESG progress, and engage with the company to encourage them to take action on our behalf.

Based on the context, what is TCI's approach to engaging with companies on ESG issues?

We engage with companies to encourage them to take action on our behalf, provide regular updates to TCI on their ESG progress, and engage with the company to manage the climate change risks and opportunities related to ESG.

2^a: TCI encourages the contacted companies to implement the following actions or strategies:

1. Change business processes to reduce the company's carbon footprint
2. Introduce efficient energy management into buildings and factories
3. Source low carbon energy through direct generation or power purchase agreements
4. De-carbonise transport fleets, e.g. Through electric vehicles
5. Offset emissions from corporate travel, e.g. Through afforestation

Overall, TCI's investment approach is one that seeks to create long-term value for shareholders while being mindful of the impact of our decisions on the environment, society and economics.

7. Who TCI has written these letters to? List the companies

1^a: The letters addressed to are:

1. Christopher Hohn, a shareholder in the company
2. Philip Green, a brand manager in the company
3. Jonathan Amouyal, a technology expert and investor in the company

I hope this helps!

Mejor respuesta:

The letters addressed to are the company's shareholders.

2^a: The letters addressed to are:

Chris Hohn, Philip Green, and Alex Baring.

8. What does the TCI Fund demand of the Spanish Minister of Ecological Transition in its communications?

1^a: The TCI Fund demands the Spanish Minister of Ecological Transition to support the adoption of low-carbon technologies by publicly disclosing a plan for doing so.

2^a: The TCI Fund demands the Spanish Minister of Ecological Transition to implement policies that will accelerate the transition towards a low-carbon economy. The TCI Fund requires the Spanish Government to take a more sustainable and responsible approach to industry growth and development, and to support the implementation of policies that will enable this to be achieved.

Prueba 4. Hiperparámetros por defecto, 6 épocas y k=15

1. What aspects concern the investor (TCI Fund)?

1^a: The TCI Fund is aware of the climate change crisis and its impact on the investment world. The company believes that climate change-related risks, in particular a company's greenhouse gas (GHG) emissions, will have a material effect on a company's long-term profitability, sustainability and investor returns. These risks include regulation, taxation, competitive disadvantage, brand impairment, financing, physical asset impairment and litigation.

2^a: The investor's (TCI) investment approach and engagement consider the company's ESG factors, particularly GHG emissions, when assessing its overall investment strategy. The investment approach and engagement consider the potential regulatory, taxation, competitive and brand impairment effects of climate change on a company's long-term profitability, sustainability and investor returns. The investment approach and engagement also consider the potential regulatory, taxation, competitive and brand impairment effects of climate change on a company's short-term profitability, sustainability and investor returns. The aim is to mitigate the risks associated with climate change and align our investment strategy with the Paris Agreement goals.

2. What are the reasons for the concerns that the TCI Fund has expressed?

1^a: The TCI Fund believes that the major risks that it faces are: 1. Climate change-related risks, in particular a company's greenhouse gas (GHG) emissions, which will have a material effect on a company's long-term profitability, sustainability and investor returns. 2. Regulation, taxation, competitive disadvantage, brand impairment, financing, physical asset impairment and litigation. 3. Climate change-related risks, in particular a company's greenhouse gas (GHG) emissions, will have a material effect on a company's long-term profitability, sustainability and investor returns.

2^a: The TCI Fund has expressed concerns that the Climate Change, Accountability and Financing (CACF) requirements of the Vinci Group may not be aligned with the TCFD recommendations. Specifically, the TCI Fund has expressed concerns that the CACF requirements of the Vinci Group may not be aligned with the TCFD recommendations because they do not align with the TCFD's goal of ensuring that the financial reporting industry is aligned with the goals of the Paris Agreement. The TCFD's goals are not aligned with the TCAA's mission, which is to promote fair and robust financial reporting practices that enable investors to understand, quantify and influence a company's exposure to climate change-related risks and the way it is managing those risks.

3. What average ownership percentage does TCI have in the contacted companies?

1^a: We believe that the average ownership percentage that TCI has in the contacted companies is 4%.

2^a: TCI's average ownership percentage in the contacted companies is 4.2%.

4. What positive aspects does TCI highlight in its communications as achievements of the contacted companies?

1^a: TCI believes that the positive aspects of its communications as achievements of the contacted companies are threefold: 1. Conducting regular communication with companies to raise awareness of their carbon reduction progress and to encourage them to take action. 2. Developing strong, meaningful and timely relationships with companies to ensure that they have a clear understanding of the risks and opportunities related to climate change. 3. Providing

guidance and support to companies on how they can achieve an “A grade” in the CDP Climate Change Program. 4. Implementing a comprehensive carbon emissions disclosure plan that includes regularly updating company websites and conducting annual carbon emissions assessments. 5. Supporting the transition of the economy to a low-carbon future by investing in energy-efficient technologies and infrastructure.

Overall, TCI's commitment to environmental stewardship is evident in its focus on employee health and wellness, as well as in its environmental and social sustainability goals, including reducing carbon emissions and promoting a more sustainable future.

2^a: TCI believes that the positive aspects of TCI's communications as achievements of the contacted companies are:

1. The companies' investment approach and engagement with TCI are positive. 2. TCI has provided regular updates to shareholders and investors about the companies' progress towards reducing their GHG emissions and managing their risks. 3. TCI's investment process assesses a range of ESG factors, particularly climate change risk. 4. The companies' GHG emissions reduction targets are aligned with the Paris Agreement. 5. TCI has received positive feedback from shareholders and investors about the companies' disclosure and engagement with TCI.

5. What main areas for improvement does TCI identify in the communications about the contacted companies?

1^a: TCI identifies the following main areas for improvement in the communications about the contacted companies:

1. Insufficient disclosure: The companies must disclose their GHG emissions, GHG reduction targets, and a low carbon transition plan in their CDP climate change reports.
2. Improved data collection: The companies must improve their data collection processes to ensure that they have accurate and timely information about their GHG emissions and other relevant environmental facts.
3. Univar: The companies must disclose their GHG emissions and other relevant environmental facts in their CDP climate change reports.
4. Walmart: The companies must improve their data collection processes to ensure that they have accurate and timely information about their GHG emissions and other relevant environmental facts.
5. Target setting: The companies must set clear and measurable targets for reducing their GHG emissions by 50% by 2025.
6. Communication: The companies must communicate the key takeaways from the CDP climate change program to their stakeholders, including customers, suppliers, and partners.
7. Engagement: The companies must engage with CDP to improve their disclosure and reporting forensics.

2^a: TCI identifies two main areas for improvement in the communications about the contacted companies: 1. Insufficient disclosure of carbon and other GHG emissions associated with the companies 2. Improved dialogue and engagement with the companies to encourage their adoption of sustainable approaches and behaviour 3. Notification and verification of carbon and other GHG emissions disclosure and engagement 4. Increased focus on developing new technologies that may have a positive impact on GHG emissions 5. Increased deployment of climate-related resources, such as improved lighting and HVAC systems, to manage the company's GHG emissions 6. Rising energy costs, particularly for lighting and HVAC systems 7. Improved energy management tools, such as energy-efficient lighting and HVAC systems, to manage the company's GHG emissions 8. Increased availability of clean energy, particularly for transportation and power generation 9. Reduction in truck rolls, particularly for urban delivery and logistics purposes 10. Increased focus on alternative fuels, such as biomass and advanced biofuels, to meet the company's GHG emissions targets 11. Improved air transportation, particularly for

long-haul flights 12. Airports and airports: Improved efficiency measures, such as improved lighting and HVAC systems, to manage the company's GHG emissions 13. Increased focus on sustainable aviation fuels, such as biofuels and advanced biofuels, to meet the company's GHG emissions targets 14. Airport emissions management systems: Improved energy efficiency measures, such as using LED lighting, implementing energy-efficient HVAC systems, and using renewable energy sources (e.g. Wind and solar power) to manage the company's GHG emissions 15. Increased focus on alternative fuels, such as biofuels and advanced biofuels, to meet the company's GHG emissions targets 16.

6. What actions or strategies does TCI encourage the contacted companies to implement?

1^a: TCI recommends that companies implement the following actions to reduce carbon emissions:

1. Change business processes to reduce the company's carbon footprint:
 - a. Implement energy efficiency measures, such as using LED lighting, insulating buildings, and using greener heating and cooling systems.
 - b. Reduce waste, such as recycling and composting.
 - c. Improve the use of renewable energy sources (e.g. Wind and solar power) through direct generation or power purchase agreements.
 - d. Optimise transportation, such as using electric vehicles, cycling, or public transport.
 - e. Invest in green innovation, such as renewable energy storage, smart grids, and energy-efficient machinery.
2. Introduce efficient energy management into buildings and factories:
 - a. Conduct an energy audit to identify areas of high energy consumption, such as lighting and heating.
 - b. Implement energy-efficient technologies, such as LED lighting, smart thermostats, and energy-efficient HVAC systems.
 - c. Support the transition of the building and factory industry to a low-carbon economy by investing in renewable energy and energy-efficient technologies.
 - d. Get involved with industry associations, such as the Building America Report of the United Association of Plumbers and Pipefitters, and the Energy Managers Association.
 - e. Learn from each other's experiences, such as through workshops and webinars.
3. Develop a low-carbon economy:
 - a. Conduct a low-carbon transition plan for the company, including prioritising emissions reductions and introducing policies to support sustainable practices.
 - b. Implement a carbon-trading strategy, such as setting targets for emissions reduction and engaging with regulators and politicians to support the adoption of carbon-neutral products and services.
 - c. Support the transition of the aviation industry to a low-carbon economy by investing in renewable energy and energy-efficient technologies.
 - d. Get involved with the UN Sustainable Development Goals, such as reducing emissions and promoting sustainable development.
 - e.

2^a: TCI recommends that the contacted companies implement the following actions: 1. Change business processes to reduce the company's carbon footprint:

- a. Implement energy efficiency measures, such as using LED lighting, insulating buildings, and using greener heating and cooling systems.
- b. Reduce waste, such as recycling and composting.
- c. Improve the use of renewable energy sources (e.g. Wind and solar power) through direct generation or power purchase agreements.
- d. Optimise transportation, such as using electric vehicles, cycling, or public transport.
- e. Invest in green innovation, such as renewable energy storage, smart grids, and energy-efficient machinery.

2. Introduce efficient energy management into buildings and factories:

- a. Conduct an energy audit to identify areas of high energy consumption, such as lighting and heating.
- b. Implement energy-efficient technologies, such as LED lighting, smart thermostats, and energy-efficient HVAC systems.
- c. Invest in energy-efficient building renovations, such as replacing old windows with energy-efficient ones.
- d. Monitor and report on energy consumption, such as using energy-efficient lighting and HVAC systems.
- e. Engage with suppliers and customers to encourage the adoption of energy-efficient technologies.

3. Source low carbon energy through direct generation or power purchase agreements:

- a. Develop renewable energy sources (e.g. Wind and solar power) through direct generation or power

purchase agreements. b. Consume carbon-neutral energy (e.g. Through buying carbon credits) through direct generation or power purchase agreements. c. Source low carbon energy through direct generation or power purchase agreements with renewable energy sources (e.g. Wind and solar power) or carbon-neutral energy sources (e.g. Through buying carbon credits). d. Consume carbon-neutral energy through direct generation or power purchase agreements with renewable energy sources (e.g. Wind and solar power) or carbon-neutral energy sources (e.g. Through buying carbon credits). e.

7. Who TCI has written these letters to? List the companies

1^a: The following are some of the companies that TCI has written letters to regarding their carbon emissions disclosure:

1. Aena SME S.A., Madrid, Spain
2. Airbus, Lille, France
3. Alphabet Inc., Mountain View, USA
4. BAE Systems, London, UK
5. Carbon Trust, London, UK
6. CDP, California, USA
7. Enaire SME S.A., Aeropuertos Españoles y Nacionales (Enaire), Madrid, Spain
8. Ferrovial, Milan, Italy
9. Hackett Group, Dublin, Ireland
10. IHG, London, UK
11. Philip Green, London, UK
12. The Children's Investment Fund, London, UK
13. Vinci, Rome, Italy
14. Xerox Corporation, Redwood City, USA
15. Zurich Airport, Switzerland

2^a: TCI has written these letters to various companies including but not limited to:

1. Alphabet Inc. 2. The Carbon Trust 3. CIMF 4. CDP 5. Aena 6. Airbus 7. Enaire 8. TCI Fund Management Limited 9. The Children's Investment Fund 10. Alphabet Classics 11. Safran 12. Vinci 13. Philip Green 14. Heathrow Airport 15. SBT! 16. London Stock Exchange 17. City Index 18. Euronext 19. London Stock Exchange (LSEW) 20.

8. What does the TCI Fund demand of the Spanish Minister of Ecological Transition in its communications?

1^a: The TCI Fund demands that the Spanish Minister of Ecological Transition provide regular updates to the TCI Fund management team regarding the progress made towards implementing the European Commission's Requirements for the Sustainability of Transportation Services (CTS) and related Criteria.

2^a: The TCI Fund demands the Spanish Minister of Ecological Transition to publish annual reports with regulation for the sourcing and management of energy across all buildings, processes, and transportation as part of the country's transition towards a low-carbon economy.