



Facultad de Ciencias Económicas y Empresariales

Los Grandes Modelos del Lenguaje basados en Transformers: revisión y aplicación práctica con ChatGPT

Autora: Marina Palomino Bravo

Clave: 201809830

Director: Carlos Martínez de Ibarreta Zorita

MADRID | JUNIO 2023

RESUMEN

En este Trabajo Fin de Grado se estudia la evolución que ha tomado el Procesamiento del Lenguaje Natural (NLP) para ofrecer una mejor comprensión de las arquitecturas *Transformer*. Se repasan conceptos como *tokenización*, *bag-of-words*, *one-hot encoding* y técnicas clásicas de procesamiento de texto. Luego, se explora la evolución hacia los modelos *Transformer*, incluyendo *word embeddings*, redes neuronales, y la arquitectura *Encoder-Decoder*. El trabajo destaca la importancia del *Prompt Engineering*, una disciplina en fase de investigación, que permite un mayor control sobre los resultados generados por los modelos de lenguaje. Se presentan los resultados obtenidos al optimizar *prompts* utilizando el modelo *ChatGPT*. Se concluye que no existen *prompts* universales que garanticen las mejores respuestas, sino que el proceso de refinamiento iterativo es fundamental. Para tareas de síntesis, inferencia de sentimiento y ampliación de texto se generaban respuestas de mayor calidad cuando se pedía al modelo que tomara un rol específico o se centrara en los aspectos clave del texto. La tarea de generación de código en R desvelaba que con 4 entradas el modelo era capaz de generar el código deseado. No obstante, el número de interacciones con el modelo variarán en función de la complejidad de la tarea.

Palabras Clave: *NLP, redes neuronales, embedding, Transformer, Prompt Engineering, ChatGPT.*

ABSTRACT

This Final Degree Project studies the evolution of Natural Language Processing (NLP) in order to offer a better understanding of Transformer architectures. Concepts such as tokenisation, bag-of-words, one-hot encoding and classical text processing techniques are reviewed. Then, the evolution towards Transformer models is explored, including word embeddings, neural networks, and the Encoder-Decoder architecture. The paper highlights the importance of Prompt Engineering, a discipline in the research phase, which allows greater control over the results generated by language models. The results obtained by optimising prompts using ChatGPT are presented. It is concluded that there are no universal prompts that guarantee the best answers, but that the process of iterative refinement is fundamental. For summarisation, sentiment inference and text expanding tasks, higher quality responses were generated when the model was asked to take a specific role or focus on key aspects of the text. The code generation task in R revealed that with 4 inputs the model was able to generate the desired code. However, the number of interactions with the model will vary depending on the complexity of the task.

Keywords: *NLP, redes neuronales, embedding, Transformer, Prompt Engineering, ChatGPT.*

Índice de Contenido

1. Introducción	4
1.1. Contexto	4
1.2. Objetivos	5
1.3. Metodología y estructura	6
1.3.1. Metodología	6
1.3.2. Estructura	6
1.4. Antecedentes y motivación.....	7
2. Evolución del Procesamiento del Lenguaje Natural	9
2.1. Fundamentos técnicos del sistema clásico	10
2.1.1. Tokenización, Bag-of-Words y One-Hot-Encoding	10
2.1.2. Técnicas de reducción de texto.....	13
2.2. Procesamiento del Lenguaje Natural basado en Deep Learning	14
2.2.1. Redes Neuronales	14
2.2.2. Word Embeddings y Continuous Bag-of-Words (CBOW).....	20
2.2.3. Redes Neuronales Convolucionales y Recurrentes	23
2.2.4. Modelo Sequence-to-Sequence en el marco Encoder-Decoder	29
3. Transformers	32
3.1. Elementos clave en la arquitectura del Transformer	32
3.2. Funcionamiento del modelo Transformer.....	38
3.3. Diseños de Transformers	39
4. Prompt Engineering: definición y aplicación práctica en ChatGPT.....	49
4.1. Introducción al Prompt Engineering.....	49
4.1.1. Ejemplos de <i>prompts</i> mal diseñados.....	51
4.1.2. Optimización de prompts para tareas de síntesis, inferencia y ampliación de texto.....	55
4.1.3. Prompt Engineering para la generación de código en R	62
5. Conclusiones	73
5.1. Conclusiones	73
5.2. Limitaciones y futuras líneas de investigación	75
6. Bibliografía	77
7. Anexos	83

1. Introducción

1.1. Contexto

Los grandes avances tecnológicos y la exhaustiva recopilación de datos a través de los medios digitales han propiciado un crecimiento sin precedentes de lo que actualmente se conoce como Procesamiento del Lenguaje Natural (Natural Language Processing o NLP, en inglés) (Patwardhan et al., 2023). El NLP en *Deep Learning* es una rama del aprendizaje automático que estudia las interacciones entre máquinas y humanos a través del lenguaje natural. Básicamente se enfoca en el procesamiento del lenguaje humano utilizando técnicas de redes neuronales profundas para entender el significado de palabras, frases o documentos. De esta forma, es capaz de desarrollar aplicaciones de traducción automática, generación de texto o sistemas de recomendación, entre otras aplicaciones (Zhou et al., 2020).

En el marco del procesamiento de datos, el texto en su forma natural no tiene una estructura definida. Entendiendo como “no estructurado” toda información que no se encuentre organizada en una tabla o formato específico (Eberendu, 2016). El texto tiene una estructura lingüística entendible para los seres humanos, pero no para las máquinas. Sin embargo, a menudo el texto puede ser estructurado mediante técnicas de procesamiento de lenguaje natural, lo que permite organizarlo y extraer información valiosa cuando esta información es utilizada por algoritmos de Machine Learning y redes neuronales.

El lenguaje humano se caracteriza por ser extremadamente complejo, variado y sutil en su expresión. Es un sistema de comunicación altamente adaptable y flexible, capaz de transmitir información de manera eficiente y precisa, pero también de transmitir emociones, matices, ironía, sarcasmo, entre otros elementos subjetivos. El lenguaje humano es complicado de entender por las máquinas no solo porque las reglas gramaticales y semánticas no siempre son explícitas o fáciles de definir, sino por la ambigüedad propia del lenguaje, la cual puede surgir por la polisemia (palabras que tienen varios significados) y la homonimia (palabras que suenan igual pero tienen diferentes significados). Asimismo, el lenguaje se basa en el conocimiento y la comprensión del mundo y la cultura, lo que dificulta aún más la tarea de las máquinas para procesar y entender el lenguaje natural (Open AI, 2023a).

A pesar de estos desafíos, el procesamiento de lenguaje natural y la inteligencia artificial han avanzado significativamente en las últimas décadas, y actualmente existen técnicas y algoritmos que permiten a las máquinas comprender y generar lenguaje humano con una mayor precisión.

La comunicación nos define como personas, pero gracias a la Inteligencia Artificial, ha sido posible trasladar esta característica a las máquinas, permitiendo así automatizar actividades que antes se relacionaban estrictamente con los seres humanos. De hecho, ya estamos viendo cómo el NLP tiene cabida en industrias como la educación, la medicina, el *marketing* o el servicio al cliente, al agilizar tareas y permitir una comunicación más eficiente. El Procesamiento del Lenguaje Natural es actualmente un área importante de investigación que está mostrando grandes avances en muy poco tiempo (Patwardhan et al., 2023).

1.2. Objetivos

El objetivo de este trabajo es revisar la literatura sobre los modelos de *Deep Learning* basados en *Transformers* para la elaboración de un resumen descriptivo de los principales modelos, así como la demostración práctica de la disciplina *Prompt Engineering* a través del modelo conversacional *ChatGPT*. En particular, se desea:

- Analizar la evolución del Procesamiento Natural del Lenguaje, desde las herramientas del sistema clásico hasta las últimas técnicas de aprendizaje profundo.
- Estudiar el funcionamiento de la arquitectura *Transformer* y describir los principales diseños de modelos basados en *Transformers* existentes hasta la fecha.
- Explicar las bases del *Prompt Engineering* y evaluar el proceso de optimización de *prompts* comparando los resultados obtenidos por *ChatGPT* para tareas de síntesis, inferencia y ampliación de texto, así como para la generación de código en el entorno de programación R.

1.3. Metodología y estructura

1.3.1. Metodología

En primer lugar se ha llevado a cabo una exploración del estado del arte del origen del Deep Learning, los modelos de Procesamiento del Lenguaje Natural (PLN) y el funcionamiento de las Redes Transformer. Para ello se han utilizado fuentes de información tales como artículos e investigaciones científicas, recopilados de bases de datos como *ArXiv*, *Web of Science* o *Google Scholar*.

A continuación, se realizó una aplicación práctica para mostrar la funcionalidad del *Prompt Engineering*, una de las últimas disciplinas que han surgido relacionadas con los grandes modelos del lenguaje. En concreto, se han implementado distintos casos prácticos donde se optimizan las interacciones con un modelo conversacional para que este lleve a cabo las tareas solicitadas. Para ello se ha utilizado el modelo pre-entrenado *ChatGPT*, de la empresa OpenAI.

1.3.2. Estructura

Este trabajo comienza introduciendo la situación en la que se enmarca el procesamiento del lenguaje natural, así como los antecedentes a esta investigación que encontramos en la literatura. A continuación, se exponen los distintos capítulos que aborda el trabajo entorno a los objetivos mencionados.

En el Capítulo 2, se investiga el la evolución que ha seguido el campo del Procesamiento del Lenguaje Natural desde las primeras técnicas del sistema clásico hasta la aparición del *Deep Learning*.

Posteriormente, en el Capítulo 3 se estudiarán los componentes clave y el funcionamiento de la arquitectura *Transformer*. Del mismo modo, se revisarán los principales diseños existentes hasta la fecha.

En el Capítulo 4 se llevará a cabo una demostración práctica de la disciplina *Prompt Engineering*. En concreto, se hará una breve introducción al tema para más adelante ejemplificar su uso a través de 4 tareas distintas (síntesis, inferencia, ampliación y codificación de texto), utilizando como modelo generativo del lenguaje *ChatGPT*.

Por último, se finalizará con un apartado de conclusiones que recogerá los principales resultados obtenidos como respuesta a los objetivos planteados en el inicio del trabajo.

1.4. Antecedentes y motivación

A pesar de los recientes y rápidos avances en el Procesamiento del Lenguaje Natural, las primeras investigaciones relacionadas con este campo datan hace décadas. En los años 50, Warren Weaver sentó las bases para el campo del NLP al abordar temas fundamentales como la traducción automática y la comunicación entre humanos y máquinas. Propuso la idea de que los lenguajes humanos pueden describirse en términos de reglas y estructuras lógicas, y que estas reglas podrían ser codificadas en algoritmos para permitir la traducción automática (Manaris, 1998 y Lennon, 2014). Desde entonces, numerosos estudios han sido publicados.

Las investigaciones más antiguas sobre los orígenes del NLP se relacionan con el uso de reglas y técnicas basadas en aprendizaje automático (Wallach, 2006; Antonellis & Gallopoulos, 2006; Qader et al., 2019; y Mikolov et al., 2013a), para más adelante evolucionar hacia el aprendizaje profundo con la aparición de las redes neuronales (Horchreiter & Schmidhuber, 1997; Abiodun et al., 2019; Kim, 2014; Dos Santos y Gatti, 2014; Schuster & Paliwal, 1997 y Cho et al., 2014). Zhou et al. (2020), Khurana et al. (2017), Young et al. (2018) y Manaris (1998) hacen una revisión sobre el estado del arte del Procesamiento del Lenguaje Natural con el objetivo de recopilar los principales avances en este campo.

La última gran revolución en el campo de la IA y en concreto, de la generación de texto automática, ha sido protagonizada por los modelos pre-entrenados *Transformers* (Vaswani et al., 2017) y los Grandes Modelos del Lenguaje (Zhao et al., 2023). Desde su publicación en 2017, hemos visto aparecer una infinidad de diseños *Transformers* como BERT (Devlin et al., 2018), XNET (Yang et al., 2019), RoBERTa (Liu et al., 2019) o GPT-3 (Brown et al., 2020). A pesar de su novedad, la literatura ya muestra algunas recopilaciones de los principales modelos *Transformers* en NLP. Por ejemplo, Patwardhan et al. (2023) revisan las aplicaciones de los modelos *Transformers* de acceso público, con especial foco en aquellos relacionados con el lenguaje. Von der Mosel et al. (2022) revisan en su estudio la validez de los transformadores dentro del campo de la ingeniería de software. Por otro lado, se han escrito libros especializados que describen

de forma técnica y detallada el funcionamiento de estos modelos, dirigidos a científicos de datos para que puedan desarrollar sus propias aplicaciones (Tunstall et al., 2022; Rothman & Gulli, 2022).

La Inteligencia Artificial comprende un conjunto de sistemas computacionales que están en constante cambio, lo que hace necesaria una revisión continua de los modelos que van apareciendo. En línea con los antecedentes, este trabajo pretende ofrecer una revisión de las técnicas de NLP existentes hasta la fecha, de forma comprensible y actualizada con el propósito de ofrecer una imagen del progreso que ha tomado este campo en el tiempo.

Asimismo, este trabajo busca explorar una de las nuevas disciplinas dentro de la IA, el “Prompt Engineering”. La novedad de esta disciplina se traduce en una escasa e insuficiente literatura sobre el tema hasta la fecha. Polak y Morgan (2023) proponen el método *ChatExtract*, un conjunto de *prompts* aplicados a *ChatGPT* para automatizar una extracción de datos de artículos de investigación. Zhou et al. (2022) proponen *Automatic Prompt Engineer* (APE) para la generación y selección automática de instrucciones por un LLM (Large Language Model) en comparación con las instrucciones generadas por humanos. White et al., (2023) describen un catálogo de patrones de ingeniería de *prompts* que se han aplicado con éxito para resolver problemas generales al conversar con un LLM. Enfocado a la generación de código encontramos que Lui et al. (2023) realizan experimentos utilizando el conjunto de datos de *CodeXGlue* para evaluar las capacidades de *ChatGPT* para dos tareas de generación de texto a código y de código a código, en lenguaje de programación Java.

A diferencia de las investigaciones previas que se centran en la calidad de los resultados generados, este trabajo pretende ilustrar el proceso manual de optimización de *prompts* que sigue un usuario cuando conversa con *ChatGPT* hasta conseguir el resultado deseado. Asimismo, pone foco en las tareas de síntesis, inferencia y expansión de texto, así como la generación de código en el lenguaje de programación R. Estas tareas son algunas de las más comunes en cuanto a procesamiento de texto se refiere (Liu et al., 2023), de ahí que se hayan seleccionado para ejemplificar el uso de estos modelos pre-entrenados.

En su conjunto, este trabajo busca ofrecer una visión global de la evolución que ha seguido el Procesamiento Natural del Lenguaje, teórica y práctica, de forma que cualquier

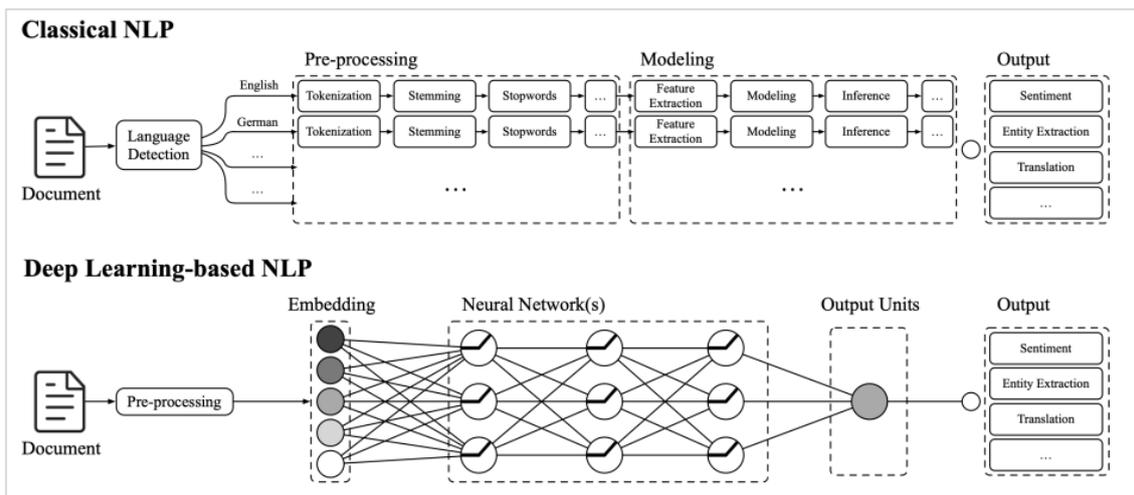
persona pueda comprender su funcionamiento y aplicación, especialmente en un tiempo en el que la Inteligencia Artificial empieza a tener cabida en cualquier actividad de la vida cotidiana.

2. Evolución del Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural (NLP) ha experimentado una evolución significativa en las últimas décadas, impulsada por la revolución tecnológica y los avances en el aprendizaje automático y la inteligencia artificial.

En este capítulo se exponen los progresos más importantes, desde la división de texto en variables para ser utilizadas como *input* de un modelo de aprendizaje automático, hasta el uso de redes neuronales que simulan la cognición humana y son capaces de entender contexto. En concreto, se presentarán técnicas simples de representación de texto como el *Bag-of-Words* y *One-Hot Encoding*, hasta técnicas más sofisticadas como *Embeddings* y Redes Neuronales Recurrentes y Convolucionales. Estas técnicas han permitido el desarrollo de modelos de NLP cada vez más precisos y efectivos para una amplia variedad de aplicaciones.

Figura 1. Comparación del funcionamiento clásico de NLP versus NLP basado en Deep Learning.



Nota. Adaptado de “A Taxonomy for Deep Learning in Natural Language Processing”, por S. Landolt, M. Söllner, & T. Wambsganb, 2021.

Una revisión de los progresos logrados en materia de procesamiento del lenguaje natural es esencial para la comprensión del funcionamiento de los modelos *Transformers*, la última gran revolución en el campo del aprendizaje profundo y la IA, y que será explicado en profundidad en el siguiente capítulo.

2.1. Fundamentos técnicos del sistema clásico

Durante décadas, el enfoque predominante en el procesamiento de lenguaje natural previo a la aparición del *Deep Learning* (aprendizaje profundo) se basaba en modelos superficiales de *Machine Learning* (aprendizaje automático) como la regresión logística o el SVM (*Support Vector Machine*) (Landolt et al., 2021). En concreto, los datos primero eran preprocesados mediante el uso de técnicas de estadística y procesamiento simbólico como la “tokenización”, lematización o eliminación de palabras vacías. Luego, las características o variables resultantes se utilizaban como entrada para entrenar un modelo, que dependiendo de la aplicación a la que esté enfocado, generaría una salida de tipo traducción, análisis de sentimiento o recuperación de información, entre otras.

Estos modelos se entrenaban con variables dispersas, “artesanales” y de grandes dimensiones, lo que podía resultar en una baja eficiencia en el procesamiento de grandes cantidades de datos (Young et al., 2018). Más adelante veremos cómo estos métodos de enfoque clásico han sido complementados por técnicas de *Deep Learning*, que han demostrado ser muy eficaces en tareas como el reconocimiento de voz, la traducción automática y la generación de texto.

2.1.1. Tokenización, Bag-of-Words y One-Hot-Encoding

El uso de texto como dato o entrada de cualquier algoritmo de aprendizaje automático requiere un procesamiento previo. Esta fase viene caracterizada por lo que se conoce en NLP como “tokenización”, el proceso de dividir un texto en unidades más pequeñas llamadas “tokens”, eliminando los espacios en blanco y los saltos de línea. Los *tokens* son básicamente palabras, frases, números o símbolos individuales, que se utilizan como elementos para la entrada posterior en un modelo. La tokenización también puede incluir otras transformaciones en el texto, como la eliminación de caracteres especiales o la conversión a minúsculas para normalizar el texto, técnicas que se mencionarán más adelante en este capítulo (Open AI, 2023b).

Este es un paso importante en muchos sistemas de NLP, ya que permite a los modelos de lenguaje analizar el texto en términos de las unidades semánticas más pequeñas, lo que a su vez facilita el análisis y la comprensión del texto por parte de las máquinas.

En concreto, uno de los enfoques comúnmente utilizados para la representación de características tomadas del texto, anterior a la explosión del aprendizaje profundo, se

conoce como *Bag-of-Words* (BOW). Este modelo representa el texto como un conjunto de palabras (*tokens*) sueltas e independientes, ignorando cualquier estructura sintáctica. Es decir, no tiene en cuenta la gramática o el orden de las palabras en la oración (Lauriola et al., 2022). A través de esta técnica se construye un "saco de palabras" (de ahí su nombre), que representa todas las palabras únicas en una colección de documentos o *corpus*. Este modelo es a menudo considerado como un modelo de espacio vectorial, ya que el *corpus* de n documentos queda representado por una matriz llamada *Document Term Matrix* (DTM) de n columnas y m filas, donde n es el número de documentos del *corpus* y m es el número de términos. Cada elemento de la matriz corresponde con algún método de medición de la importancia del término i con respecto al documento y a la colección completa. De esta forma se transforma el texto en un formato comprensible por las máquinas (Antonellis & Gallopoulos, 2006).

La forma de representación más simple consiste en asignar un vector a cada *token*. Un vector con tantas posiciones como palabras identificadas haya, de forma que cuando se quiera representar una palabra en concreto, su posición en el vector vendrá representada por un 1, marcando el resto de las palabras con un 0. A esta representación se le llama *One-Hot Encoding* (Torfi et al., 2020). Recordamos que las palabras no están ordenadas, cada una es independiente y ocupa su propia dimensión. Y por tanto, cada palabra mantiene la misma distancia con el resto en el texto.

El enfoque *Bag-of-Words* se puede calcular como una suma de vectores *one-hot*. De forma que el vector de características resultante para cada documento o frase en el *corpus* representa la presencia o ausencia de cada palabra en el documento (Lauriola et al., 2022 y Qader et al., 2019).

Por ejemplo, si tenemos tres oraciones:

1. "La abuela hizo una tarta de cumpleaños"
2. "La prima celebró su cumpleaños en el pueblo"
3. "Pasamos un cumpleaños increíble en la casa del pueblo"

El saco de palabras sería:

['la', 'abuela', 'hizo', 'una', 'tarta', 'de', 'cumpleaños', 'prima', 'celebró', 'su', 'en', 'el', 'pueblo', 'pasamos', 'un', 'increíble', 'casa', 'del']

Y bajo un enfoque *one-hot encoding*, la palabra “cumpleaños” quedaría representada de la siguiente forma en cada oración:

1. [0, 0, 0, 0, 0, 0, 1]
2. [0, 0, 0, 0, 1, 0, 0, 0]
3. [0, 0, 1, 0, 0, 0, 0, 0]

Otra solución para representar la información de un documento de texto, bajo el enfoque de BOW, consiste en contar la frecuencia de cada palabra en un documento y utilizar esos recuentos como características para representarlo. De esta forma, cada vector representa la frecuencia con la que un determinado término o *token* aparece en el documento. Este método de representación se conoce como *Term Frequency Representation*.

Por ejemplo, si tenemos el siguiente documento de texto:

"María cocina platos salados. Javier cocina platos dulces."

La representación de frecuencia de términos de este documento sería:

{'María': 1, 'cocina': 2, 'platos': 2, 'salados': 1, 'Javier': 1, 'dulces': 1}

Esto significa que la palabra "cocina" aparece dos veces, "salados" aparece una vez, y así sucesivamente, en el total del documento.

Esta técnica es útil porque permite representar documentos de diferentes longitudes en vectores de características de la misma dimensión, lo que facilita la comparación y el análisis de múltiples documentos. Sin embargo, la representación de la frecuencia de los términos no tiene en cuenta la importancia relativa de las palabras en el documento, lo que puede llevar a una sobrerrepresentación de palabras comunes y a una subrepresentación de palabras menos frecuentes pero más importantes (Open AI, 2023c). Este problema quedaría resuelto dividiendo la frecuencia de cada palabra por el total del documento.

Asimismo, un término que aparece en todos los documentos del *corpus* no aporta apenas información. Para ello, existe una tercera técnica de representación: *Term Frequency-Inverse Document Representation*, a través de la cual la importancia de una

palabra o término no viene dada simplemente por su frecuencia sino por cómo de común es el término en todo el conjunto del corpus. Dado un documento d y un término t , la TF-IDF para dicho documento y término corresponde con la multiplicación de las dos variables (Zhang et al., 2015):

$$\text{TFIDF}(d,t) = \text{TF}(d,t) \times \text{IDF}(t)$$

En resumen, el enfoque Bag-of-Words tiene la ventaja de ser simple y rápido de implementar, con muchas aplicaciones en tareas de clasificación y recuperación de información (Qader et al., 2019), pero no tiene en cuenta cómo se relacionan las palabras en el documento u oración, lo que puede llevar a una pérdida de información semántica y contextual, y dificultar la detección de sutilezas lingüísticas.

2.1.2. Técnicas de reducción de texto

Es importante preprocesar los datos antes de crear la matriz de términos. Esto se debe a que el procesamiento de texto puede ser muy costoso computacionalmente si hay un gran número de palabras en un corpus. Hay varias técnicas de reducción de texto que se utilizan comúnmente en el procesamiento de lenguaje y que ayudan no solo a reducir la dimensionalidad del vocabulario, sino también a mejorar la precisión de los modelos NLP. Entre ellas destacan: (Open AI, 2013d).

1. Stemming. Un proceso que implica eliminar los sufijos de las palabras para reducirlas a su raíz o "stem". Por ejemplo, las palabras "amigos", "amistoso" y "amistad" se reducen a la raíz "amig".

2. Lematización. Un proceso similar al stemming, pero en lugar de simplemente eliminar los sufijos, se reducen las palabras a su forma base o "lema", tal y como aparecería en el diccionario. Por ejemplo, las palabras "corriendo", "corre", y "corrió" se lematizan a "correr". La lematización es más precisa que el stemming, pero también es más costosa computacionalmente.

3. Eliminación de *Stopwords* o palabras vacías: son palabras muy comunes como artículos o preposiciones ("el", "a", ...) que se eliminan del texto ya que no aportan mucho valor semántico.

5. Convertir todos los términos a minúscula para que sean identificados como el mismo token, en el caso de no ser necesario conservar nombres propios.

6. Otras posibles técnicas: extracción de entidades nombradas (como nombres de personas, organizaciones, lugares, etc.), eliminar puntuación, números, espacios, caracteres especiales...

2.2. Procesamiento del Lenguaje Natural basado en Deep Learning

Las técnicas de aprendizaje automático no son lo suficientemente sofisticadas como para manejar la complejidad inherente del lenguaje natural, el procesamiento de una gran cantidad de datos y la variabilidad en el uso del lenguaje en diferentes contextos y dominios.

A inicios de los años 2000, los modelos de lenguaje neuronal comenzaron a reemplazar a los modelos basados en estadísticas. El primer modelo importante fue el modelo de lenguaje neuronal de Bengio et al. (2000), que utilizó redes neuronales para modelar la probabilidad de una secuencia de palabras. Más tarde, en 2008, Collobert y Weston demostraron la eficacia de los *embeddings* preentrenados en la tarea de etiquetado de secuencias de texto, lo que abrió el camino para futuras investigaciones sobre el uso de *embeddings* en el procesamiento del lenguaje natural. Desde entonces, los *embeddings* preentrenados han demostrado ser una herramienta muy valiosa en la mayoría de las tareas de NLP.

El surgimiento del aprendizaje profundo o *Deep Learning* ha permitido el desarrollo de modelos de NLP muy poderosos que pueden manejar grandes cantidades de datos y aprender patrones complejos en el lenguaje natural. Estos modelos han demostrado ser muy efectivos en tareas como la traducción automática, la generación de texto y el análisis de sentimientos, entre otras (Khurana et al., 2017).

2.2.1. Redes Neuronales

Las redes neuronales profundas son un conjunto de algoritmos de aprendizaje automático especialmente potentes que se inspiran en el cerebro humano para procesar información, a través de las cuales se pueden modelar comportamientos inteligentes. Estas redes están compuestas por múltiples capas de neuronas que permiten el aprendizaje de patrones complejos en los datos (Goldberg, 2016).

En el contexto de NLP, las redes neuronales profundas se utilizan para entender el lenguaje humano. Por ejemplo, se pueden utilizar para la clasificación de texto, el análisis

de sentimientos, la generación de texto, la traducción automática y muchas otras tareas relacionadas con el procesamiento del lenguaje natural. Una de las principales ventajas de utilizar redes neuronales profundas para estas tareas es que los modelos son capaces de aprender automáticamente representaciones semánticas del lenguaje, lo que les permite entender el significado de las palabras y frases en un contexto específico. Además, las redes neuronales profundas pueden aprender de grandes cantidades de datos, mejorando así su precisión a medida que reciben más información (Open AI, 2023e).

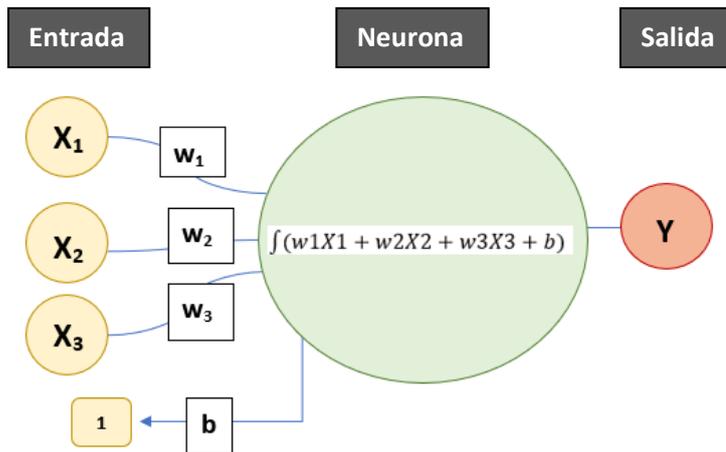
Funcionamiento de una Neurona

Las RN están formadas por neuronas (nodos), que son las unidades básicas de procesamiento. Similar a una neurona biológica estas neuronas tienen unas conexiones de entrada desde las que reciben unos estímulos externos (valores de entrada). Con estos valores las neuronas realizarán un cálculo interno y generan un valor de salida, como si de una función matemática se tratara (Goldberg, 2016).

Internamente la neurona utiliza todos los valores de entrada para realizar una suma ponderada de ellos. La ponderación viene dada por el peso que se le asigna a cada una de las conexiones de entrada. Es decir, cada conexión que llega a la neurona tendrá asociado un valor que servirá para definir la intensidad con la que cada valor de entrada afecta a la neurona. A estos pesos se les conoce como los “parámetros” del modelo.

De hecho, la neurona funciona de forma similar a un modelo de regresión lineal, donde unas variables de entrada definen una recta o hiperplano cuya inclinación varía según los parámetros o pesos asignados. Asimismo, en la neurona hay un término independiente llamado “sesgo” que permite mover la función. Se representa como otra variable de entrada, pero siempre está asignada al valor de 1 (Kröse & Van der Smagt, 1996).

Figura 2. Representación del funcionamiento de una neurona.



Nota. Elaboración propia.

No obstante, existe una última componente que diferencia una neurona de una simple regresión, y esa es la “función de activación”. El valor de salida pasa por la llamada función de activación, cuyo cometido es añadir deformaciones no lineales, distorsionando el resultado. Existen distintas funciones de activación, cada una con sus propias características y por tanto más adecuada para una tarea específica. Entre ellas encontramos la función de activación escalonada, sigmoidea, tangente, ReLU, etc (Ramachandran et al., 2017).

El motivo de incluir esta función reside en la incapacidad de una única neurona de resolver problemas no lineales. En cambio, las redes neuronales, entendidas como un conjunto de neuronas conectadas de forma secuencial y concatenando diferentes operaciones matemáticas, son capaces de desarrollar soluciones muy complejas. Mediante la función de activación, por tanto, se pueden encadenar de forma efectiva la computación de varias neuronas al mismo tiempo y resolver problemas no lineales más complejos.

En una red neuronal, aquellas neuronas o nodos que se encuentren en la misma “capa”, recibirán la misma información que ha sido previamente procesada en la capa anterior. Los cálculos que realizan se trasladan a la capa siguiente, y así sucesivamente recorren todas las capas que componen la neurona. Según la información va pasando por cada una de las capas, se va elaborando conocimiento más complejo, abstracto e interesante. Cuantas más capas añadimos, más complejo el conocimiento elaborado. Esta profundidad en la cantidad de capas es lo que da nombre al Deep Learning (Goldberg,

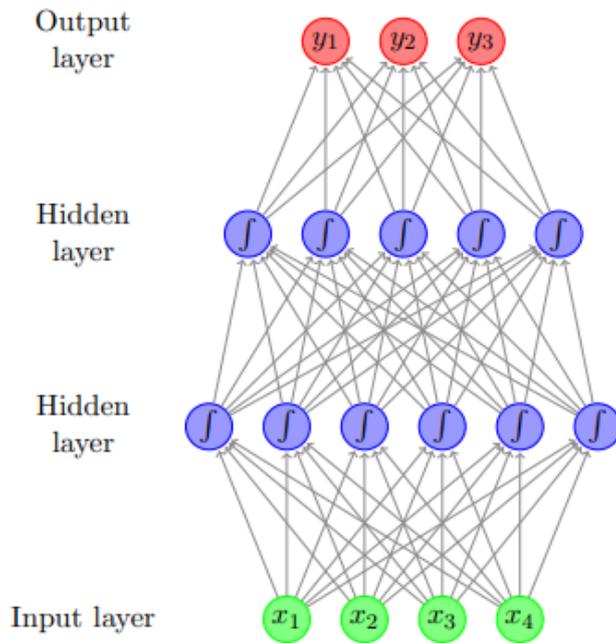
2016).

Multilayer Perceptron (MLP)

En concreto, el término Multilayer Perceptron (MLP) se refiere a un tipo de red neuronal artificial (ANN) alimentada hacia adelante (*feedforward*), que consta de múltiples capas de neuronas, cada una de las cuales está completamente conectada a la capa siguiente (Gardner & Dorling, 1998). La información fluye en una sola dirección, desde la entrada pasando a través de una serie de capas ocultas y hasta la salida, sin que haya ciclos o retroalimentación en la red. En un MLP, las neuronas en cada capa oculta usan una función de activación no lineal para transformar la entrada de la capa anterior. La salida de la última capa del MLP es la salida de la red.

El MLP Multilayer Perceptron se remonta a la década de 1940, cuando Warren McCulloch y Walter Pitts propusieron un modelo de neurona artificial que imitaba el comportamiento de las neuronas biológicas. Sin embargo, este modelo se limitaba a la clasificación lineal y no podía resolver problemas no lineales, como se explica en el apartado anterior. Fue en la década de 1980 cuando el MLP fue desarrollado como una forma de extender el modelo de perceptrón para resolver problemas más complejos. En estos años, autores como Paul Werbos, David Rumelhart, o Geoffrey Hinton comenzaron a desarrollar el concepto de “retropropagación del error” como método para entrenar las redes neuronales multicapa (Kröse & Van der Smagt, 1996).

Figura 3. Red Neuronal feed-forward con dos capas ocultas.



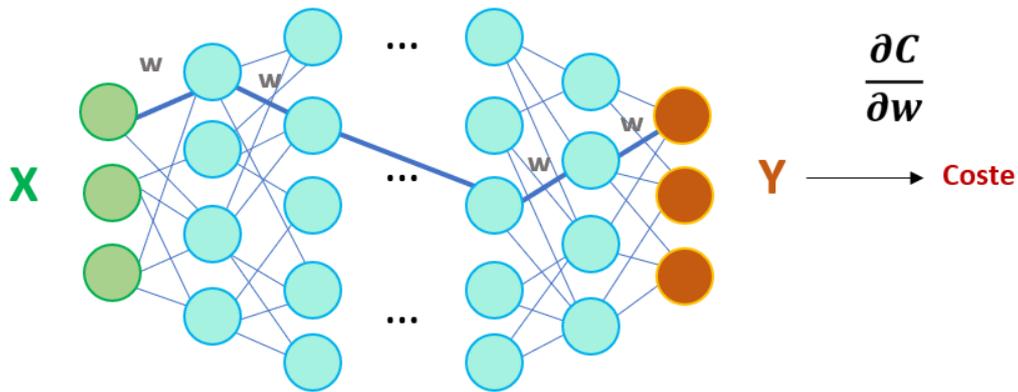
Nota. Adaptado de “A primer on neural network models for natural language processing” (p.345-420), por Y. Goldberg, 2016, *Journal of Artificial Intelligence Research*, 57.

Una de las características principales de las redes neuronales es que aprenden por sí solas a partir de los datos, de forma que no es necesaria la intervención humana en el ajuste de los parámetros de entrada. Esto se consigue a través de dos técnicas esenciales en el entrenamiento de redes neuronales: la Retropropagación del error y el Descenso del gradiente.

Funcionamiento de Backpropagation y Descenso del Gradiente

El algoritmo de *Backpropagation* o “Retropropagación del error” es un algoritmo de aprendizaje que se utiliza para propagar el error a través de la red neuronal y calcular los gradientes de una función de coste con respecto a los parámetros de la red, lo que es necesario para el Descenso del Gradiente. En concreto, *Backpropagation* se utiliza para retroceder a través de la red neuronal, calculando la contribución de cada neurona en el error total de la red. A continuación, se utiliza el Descenso del Gradiente, un algoritmo de optimización utilizado para ajustar los parámetros de la red neuronal y minimizar así la llamada función de coste, pérdida u objetivo (Lipton et al., 2015).

Figura 4. Representación del proceso de *Backpropagation* y el *Descenso del gradiente*.



Nota. Elaboración propia. *w* hace referencia a los pesos o parámetros que se optimizan y *C* al coste o error de predicción. La derivada representa cuanto varía la predicción cuando cambian los parámetros. Fuente: elaboración propia.

El proceso funciona de la siguiente manera (Lipton et al., 2015; Kröse & Van der Smagt, 1996).

1. Propagación hacia adelante: Se toman los datos de entrada y se propagan a través de la red neuronal, capa por capa, hasta llegar a la capa de salida. Cada neurona en la red realiza una suma ponderada de las entradas, aplica una función de activación y produce una salida.
2. Cálculo del error: Se compara la salida obtenida por la red neuronal con el valor objetivo deseado. Se calcula una medida de error o pérdida utilizando una función de coste, que evalúa qué tan bien se está prediciendo el resultado.
3. Retropropagación del error: el algoritmo de *backpropagation* calcula cuánta responsabilidad o “culpa” tiene cada neurona en el resultado final. Este análisis se hace hacia atrás, y según la implicación de cada neurona en el resultado final, se le asignará un porcentaje de responsabilidad o error a cada neurona (cadena de responsabilidades).

Una vez que se calcula el error en la capa de salida, se propaga hacia atrás a través de la red neuronal. Los gradientes de error se calculan para cada peso y se propagan en sentido inverso hasta la capa de entrada. Los gradientes son vectores que contienen las derivadas parciales de una función con respecto a cada una de las variables independientes, y por tanto indican la dirección y magnitud de mayor crecimiento de la función, que en este caso es la función de coste o pérdida.

4. Ajuste de los pesos: Utilizando los gradientes de error calculados, se actualizan los pesos de las conexiones en la red neuronal mediante un algoritmo de optimización, el descenso del gradiente. El algoritmo del descenso del gradiente ajusta los pesos de la red neuronal en la dirección opuesta al gradiente, de modo que la función de coste se minimice y se encuentren los pesos que reduzcan el error de predicción.

En una regresión lineal es muy fácil aplicar el descenso del gradiente porque los parámetros afectan directamente al resultado final. Sin embargo, en una red neuronal la información pasa por varias capas neuronales, de forma que, al variar algún parámetro, no es posible saber cómo dicho cambio puede influir en el resultado final por las distintas conexiones que participan en la cadena de responsabilidades

5. Repetición del proceso: Los pasos anteriores se repiten iterativamente para varios ejemplos de entrenamiento hasta que el modelo de red neuronal converge y los pesos se ajustan de manera óptima para realizar predicciones precisas.

En resumen, el objetivo final de ambas técnicas es encontrar los valores óptimos para los pesos y sesgos de la red neuronal para que la salida de la red se aproxime lo más posible a la salida deseada.

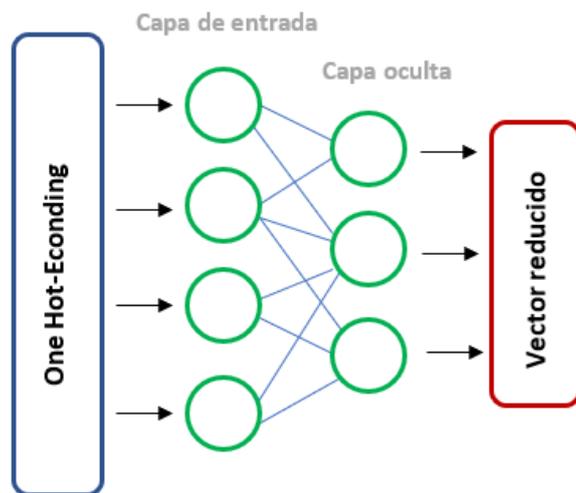
2.2.2. Word Embeddings y Continuous Bag-of-Words (CBOW)

Como seres humanos somos capaces de entender que hay palabras que están más cerca conceptualmente unas de otras. En las técnicas de representación de *Bag-of-Words* y *One Hot-Encoding* surgía el problema de que los términos tenían representaciones ortogonales, es decir, eran equidistantes y toda información sobre el contexto se perdía. No obstante, el orden en el que aparecen las palabras en el texto es una información que es interesante codificar para los algoritmos de NLP.

La aparición de los *embeddings* supone el primer punto de encuentro entre el Procesamiento del Lenguaje Natural clásico y el Aprendizaje Profundo, como mecanismo de reducción de la dimensionalidad del texto que permite capturar el significado de las palabras en función del contexto. Los orígenes del concepto de *embedding* se remonta a la década de 1990. Hinton y McClelland hablaban de representaciones distribuidas en el aprendizaje automático (1984), pero no fue hasta el surgimiento de técnicas de aprendizaje profundo que los *embeddings* se popularizaron ampliamente en el procesamiento del lenguaje natural (Bengio et al., 2000 y Mikolov et al., 2013a).

En concreto, “embedding” hace referencia al proceso que ocurre en la primera capa de una red neuronal, donde la neurona tiene que aprender a comprimir vectores *One Hot-Encoding* a una representación más compacta y sofisticada que le permita resolver su tarea. Los *embeddings* son representaciones vectoriales de las palabras, donde cada dimensión del vector representa un aspecto semántico o sintáctico de la palabra (Mikolov et al., 2013a). Gracias a esta transformación, las palabras son representadas numéricamente, lo que permite operar matemáticamente con ellas.

Figura 5. Reducción de dimensionalidad del vector de palabras de entrada.



Nota. Elaboración propia.

Como bien se muestra en la imagen, si por ejemplo tenemos un texto con 10 mil palabras, la entrada de la red correspondería con vectores de tamaño 10 mil según una representación *One-Hot Encoding*. Pero una vez la entrada es procesada por la primera capa neuronal, la salida resultante es un vector de dimensionalidad reducida.

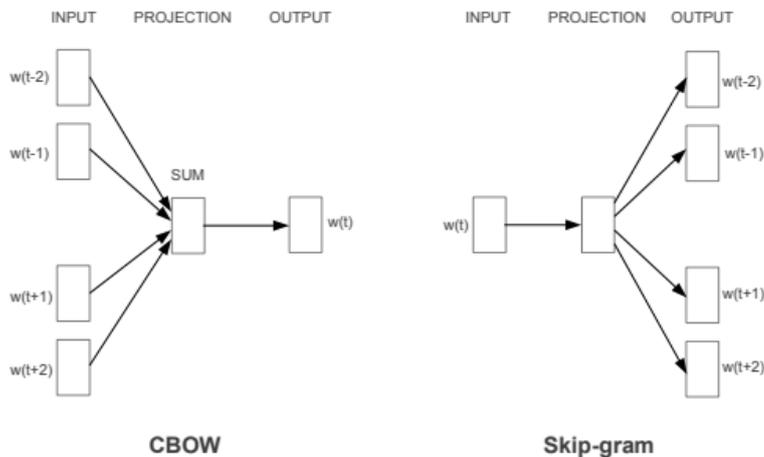
Con un *embedding* podemos dejar que sea una red neuronal la que aprenda una codificación más inteligente para resolver la tarea objetivo. Incluso, la red neuronal puede partir de *embeddings* preentrenados, de forma que el conocimiento ya ha sido aprendido y solo es necesario transferirlo a una nueva tarea (Mikolov et al., 2013b). Para que el *embedding* sea lo suficientemente universal para poder ser aplicado a distintos tipos de tareas, este debe ser entrenado con grandes cantidades de datos.

Si bien Collobert y Weston (2008) fueron pioneros en la investigación de la utilidad de los *embeddings* preentrenados en el procesamiento del lenguaje natural, la inmensa

popularización se debió posiblemente a Mikolov et al., un equipo de investigación de Google quienes presentaron los modelos de CBOW y Skip-gram bajo el *framework* Word2Vec en las populares publicaciones "Distributed Representations of Words and Phrases and Their Compositionality" y "Efficient Estimation of Word Representation in Vector Space" en el año 2013. La idea principal detrás de Word2Vec es que las palabras que aparecen en contextos similares en el texto tendrán *embeddings* similares. Además, el algoritmo es capaz de aprender relaciones semánticas entre las palabras, por ejemplo, que "rey" está relacionado con "reina" de la misma manera que "hombre" está relacionado con "mujer". Ambos modelos, CBOW y Skip-gram, son utilizados para aprender *embeddings* de palabras, y han demostrado ser muy efectivos en diversas tareas de NLP.

En el modelo CBOW, el objetivo es predecir una palabra dada su ventana de contexto, es decir, las palabras que la rodean en un corpus de texto (Torfi et al., 2020). El modelo utiliza una red neuronal de una sola capa oculta para aprender los *embeddings* de palabras, y la capa oculta actúa como la representación vectorial de la palabra en el espacio de *embeddings*. El otro modelo presentado en el paper es Skip-gram, que en lugar de predecir una palabra dada su contexto, se enfoca en predecir las palabras del contexto a partir de la palabra central (Mikolov et al., 2013a y Mikolov et al., 2013b).

Figura 6. Funcionamiento de los modelos CBOW y Skip-gram.



Nota. Adaptado de "Distributed Representations of Words and Phrases and their Compositionality", por Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J., 2013.

En concreto el paper "Distributed Representations of Words and Phrases and Their Compositionality" (Mikolov et al., 2013b) presenta un nuevo algoritmo para entrenar *embeddings* de palabras como alternativa a la técnica "Hierarchical Softmax". La función *softmax* se utiliza para normalizar los valores de salida en un vector de probabilidades, donde la suma de todas las probabilidades es igual a 1. En concreto, *softmax* se utiliza para calcular la probabilidad de una palabra objetivo dada una palabra de contexto. Sin embargo, el cálculo del *softmax* puede ser computacionalmente costoso, especialmente cuando hay un gran número de palabras en el vocabulario. Por esta razón, se propone una técnica llamada "Negative Sampling" como una alternativa más eficiente en términos de velocidad y escalabilidad. En *negative sampling*, en lugar de calcular las probabilidades de todas las palabras en el vocabulario, se eligen un subconjunto pequeño de palabras no objetivo, también conocidas como palabras negativas. Estas palabras negativas se muestrean de acuerdo con alguna distribución, como la frecuencia de las palabras. El objetivo es que estas palabras negativas tengan menos probabilidades de ser elegidas en comparación con las palabras objetivo reales. Durante el entrenamiento, en lugar de maximizar la probabilidad de todas las palabras objetivo en el vocabulario, se ajustan los parámetros del modelo para maximizar la probabilidad de las palabras objetivo y minimizar la probabilidad de las palabras negativas. Esto ayuda a enfocar el aprendizaje en la discriminación entre palabras objetivo y palabras negativas, en lugar de considerar todas las palabras del vocabulario (Open AI, 2023f).

2.2.3. Redes Neuronales Convolucionales y Recurrentes

Las redes neuronales recurrentes (RNNs) se introdujeron en los años 80 (Hopfield, 1982) como una extensión de las redes neuronales para procesar secuencias de datos, como el lenguaje natural. Sin embargo, las RNNs tardaron varios años en ser ampliamente adoptadas debido a los desafíos de entrenamiento y la complejidad computacional involucrada. Las redes neuronales convolucionales (CNNs), por su parte, se popularizaron en el campo de la visión a finales de los años 90 (Lecun et al., 1998) como una forma de extraer características útiles de imágenes. Fue más tarde, en la década de 2010, que las CNNs comenzaron a utilizarse en el procesamiento del lenguaje natural como una forma de modelar secuencias de texto (Kim, 2014). En particular, se descubrió que las capas de convolución podían actuar como extractores de características que podían identificar patrones significativos en los datos de texto, útiles en tareas como la clasificación de texto, la generación de texto y la traducción automática. Las CNNs

también han demostrado ser efectivas en tareas que involucran textos cortos, como la clasificación de sentimientos en redes sociales y la detección de spam en correos electrónicos.

Redes Neuronales Recurrentes

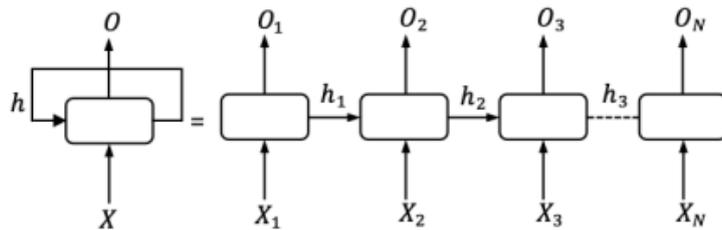
En general, las redes neuronales *feedforward* no tienen memoria explícita. La información fluye a través de la red desde la capa de entrada hasta la capa de salida, y no hay una retroalimentación directa en la red que permita "recordar" estados anteriores o patrones que se han presentado anteriormente. Hasta ahora la red neuronal básica tomaba como input la primera palabra de una secuencia de texto y esta era posteriormente procesada multiplicándose capa tras capa con los parámetros aprendidos de la red. No obstante, es probable que dicha palabra esté relacionada semántica y sintácticamente con el resto de las palabras de la secuencia.

Las redes neuronales recurrentes (RNN) son una variante de las redes feedforward (FNN) que tienen conexiones de retroalimentación y que pueden mantener una especie de "memoria" de los estados anteriores. Las redes neuronales recurrentes son básicamente alineaciones de secuencias de FNNs donde la salida de cada FNN es añadida a la entrada de la siguiente (Horchreiter & Schmidhuber, 1997). De esta forma, la información procesada por la red al tomar una palabra como input es utilizada para procesar entradas futuras de las palabras siguientes de la secuencia.

La red neuronal realiza el mismo cálculo sobre cada token o palabra de la secuencia de texto y cada procesamiento depende de los cálculos y resultados anteriores (Young et al., 2018). Son las capas ocultas de la red las que actúan como unidades de memoria (Horchreiter & Schmidhuber, 1997). En cada instante temporal se hacen predicciones y se utiliza la información de la capa oculta actual como entrada del siguiente instante temporal. Este proceso de conectar el procesamiento del output anterior con el input del procesamiento actual es lo que da el nombre de "recurrentes" (Torfi et al., 2020).

Si observamos la frase "Queen era una banda de rock; algunos la consideran una de las mejores de la historia". Para un humano es fácil identificar "la" con "banda", aunque "banda" aparezca tres palabras después. Lo que pretenden las RNNs es precisamente recordar palabras que ya han sido procesadas y poder asociarlas con palabras que se procesarán después.

Figura 7. Red Neuronal Recurrente (RNN).



Nota. X indica la entrada, N los instantes temporales, h las capas ocultas y O la salida. Adaptado de "Natural Language Processing advancements by deep learning: A survey", por Torfi, A., Shirvani, R. A., Keneshloo, Y., Tavaf, N., & Fox, E. A., 2020.

Aunque las RNNs son muy útiles en muchas aplicaciones, también tienen algunas desventajas, entre las que se incluyen:

- Desvanecimiento del gradiente y dificultad para aprender dependencias a largo plazo: En las RNNs tradicionales, el gradiente puede desvanecerse o explotar a medida que se retropropaga a través de varias capas, lo que puede dificultar el aprendizaje de patrones a largo plazo (Lipton et al., 2015 y Hochreiter & Schmidhuber, 1997).
- Dificultad para manejar datos de longitud variable: Las RNNs requieren que todas las secuencias tengan la misma longitud, lo que puede ser difícil de manejar cuando se trabaja con datos de longitud variable (Vaswani et al., 2017).
- Memoria limitada a corto plazo: Las RNNs tienen una memoria limitada a corto plazo, lo que significa que solo pueden recordar información de las últimas etapas de la secuencia (Hochreiter & Schmidhuber, 1997).

En 1997 se presentaron dos tipos de arquitecturas RNN que pretendían resolver estos problemas en el aprendizaje de secuencias: Hochreiter y Schmidhuber presentaron la *Long Short-Term Memory*, y Schuster y Paliwal (1997) introdujeron las Redes neuronales recurrentes bidireccionales. Estos modelos de redes no son mutuamente excluyentes y han ido evolucionando a lo largo de los años.

El problema que se intentaba resolver con la arquitectura *Long Short-Term Memory* (LSTM) era el de la "desaparición del gradiente", que es un problema común en las RNNs tradicionales. A medida que se retropropaga el gradiente a través de capas anteriores en

la red, se multiplican las derivadas parciales correspondientes a cada capa, lo que puede dar lugar a un gradiente que se vuelve exponencialmente pequeño (Lipton et al, 2015).

Otra razón por la que ocurre la desaparición del gradiente en las RNNs es que la función de activación que se utiliza en las capas ocultas, como la función sigmoide o la tangente hiperbólica, tiene gradientes que se vuelven muy pequeños cuando los valores de entrada están muy lejos del rango lineal de la función. Esto significa que la retropropagación del gradiente se vuelve menos efectiva a medida que las entradas se alejan del rango lineal de la función de activación (Horchreiter & Schmidhuber, 1997).

Para resolver este problema, los autores propusieron una arquitectura de red neuronal que utiliza una serie de "puertas" (gateways) que regulan el flujo de información en la red. Estas puertas permiten que la red memorice información a largo plazo y eviten que la información menos relevante sea olvidada. En concreto, la arquitectura LSTM consta de tres puertas: la puerta de entrada, la puerta de olvido y la puerta de salida. Estas puertas permiten que la red decida qué información debe pasar a través de la red y qué información debe ser ignorada o eliminada (Horchreiter & Schmidhuber, 1997).

Si bien es cierto que las LSTMs resolvieron el problema de la pérdida de memoria a largo plazo de las RNNs, las LSTMs aún tienen algunas limitaciones en términos de cómputo, ya que su cálculo secuencial es costoso y no puede ser fácilmente paralelizado, lo que puede limitar su capacidad para procesar grandes conjuntos de datos.

Por otro lado, las redes recurrentes bidireccionales (BRNNs) procesan la secuencia de entrada en dos direcciones: hacia adelante y hacia atrás (Schuster & Paliwal, 1997). Es decir, se utiliza información tanto del futuro como del pasado para determinar la salida en cualquier punto de la secuencia, en contraste con las redes anteriores en las que solo la entrada pasada puede afectar la salida.

En una BRNN, se utilizan dos RNNs idénticas, una que procesa la secuencia de entrada en sentido directo y otra que procesa la secuencia de entrada en sentido inverso. La salida de cada RNN se concatena en cada paso de tiempo para obtener una representación combinada de la entrada en ese punto en el tiempo. Esta representación combinada se utiliza como entrada para las capas de salida de la red.

La ventaja de utilizar una BRNN es que puede capturar tanto la información contextual anterior como la posterior a cada punto en la secuencia de entrada. Esto puede

ser especialmente útil en tareas de procesamiento de lenguaje natural, como el reconocimiento de voz o la traducción automática, donde la información contextual anterior y posterior puede ser importante para la interpretación correcta de la entrada. No obstante, las BRNNs es que requieren más recursos computacionales que las RNNs tradicionales debido a la necesidad de procesar la secuencia de entrada en dos direcciones. Además, en algunos casos, la información contextual anterior y posterior puede no ser igualmente relevante para todas las tareas, lo que puede limitar la eficacia de las BRNNs en ciertos casos.

Cabe mencionar también en este punto las GRU (*Gated Recurrent Units*), un tipo de arquitectura de redes neuronales recurrentes que fueron propuestas por Kyunghyun Cho, et al., en 2014 como variante de las redes LSTM para abordar el problema del desvanecimiento de gradientes en las RNN estándar. Aunque las GRU y las LSTM comparten el objetivo de resolver este problema, hay algunas diferencias clave entre ellas. Las GRU son más simples en términos de estructura y tienen menos compuertas. Se componen de una "puerta de actualización" que determina cuánta información nueva se debe pasar en cada paso de tiempo, y una "puerta de reinicio" que controla cuánta información pasada se debe olvidar. Las GRU combinan estas dos puertas en una sola unidad, lo que reduce la complejidad en comparación con las LSTM (Cho et al., 2014).

Redes Neuronales Convolucionales

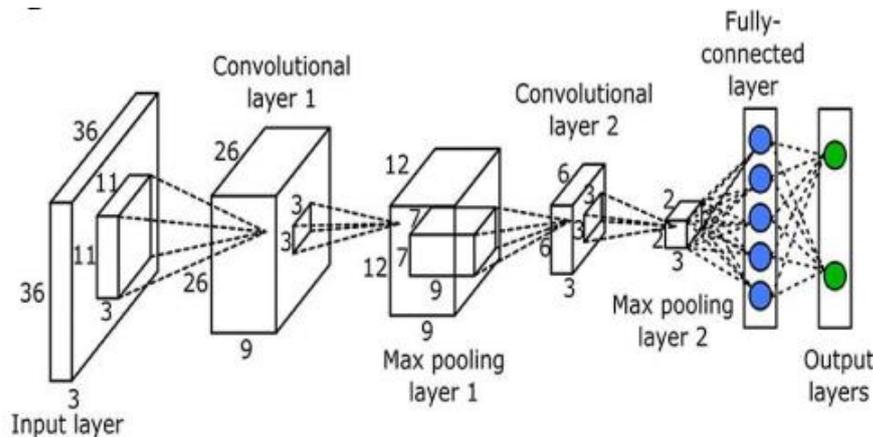
Una red neuronal convolucional (CNN) es un tipo de red neuronal profunda de tipo *feedforward* que simula la corteza visual humana. Este tipo de redes se utiliza principalmente para el procesamiento de imágenes y vídeos (Torfi et al., 2020).

Las CNN son capaces de identificar patrones y características en una imagen a través de un proceso de convolución, de ahí su nombre, en el que se aplican filtros para extraer información relevante. Las capas de una CNN suelen incluir capas de convolución, capas de agrupación (pooling) y capas totalmente conectadas. Las capas de convolución aplican filtros a la imagen de entrada para extraer características, mientras que las capas de agrupación reducen la dimensionalidad de la salida de las capas de convolución. Las capas totalmente conectadas son similares a las capas de una red neuronal estándar y se utilizan para clasificar la imagen final (Abiodun et al., 2019).

Las redes neuronales convolucionales suelen emplearse en situaciones en las que los

datos de entrada están representados de forma bidimensional o tridimensional. La imagen que la red recibe como input está representada como una matriz de valores numéricos. Sobre estos valores se realizan operaciones matemáticas para extraer características de la imagen y representar el contenido clave en el espacio de salida de la red. Las características extraídas en el proceso pueden utilizarse para otro tipo de tarea o análisis. (Torfi et al., 2020).

Figura 8. Arquitectura de una Red Neuronal Convolutiva (CNN).



Nota. Adaptado de "Comprehensive review of artificial neural network applications to pattern recognition", por Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., ... & Kiru, M. U., 2019.

Las CNN han demostrado ser muy eficaces en tareas de clasificación y reconocimiento de objetos en imágenes y vídeos, y son ampliamente utilizadas en aplicaciones de visión artificial, como la detección de objetos en tiempo real, la clasificación de imágenes médicas y la identificación de rostros. Asimismo, las CNNs se utilizan a menudo también para tareas de procesamiento natural del lenguaje, en cuyo caso, las entradas son frases o documentos representados como matrices donde cada fila de la matriz está asociada a un token. Las arquitecturas de CNN han sido utilizadas en varias tareas de clasificación, como el análisis de sentimientos y la categorización de temas (Kim, 2014; Dos Santos & Gatti, 2014; y Johnson & Zhang, 2015).

Figura 9. Red Neuronal Convolutiva para la representación de palabras (embedding) en una tarea de NLP.

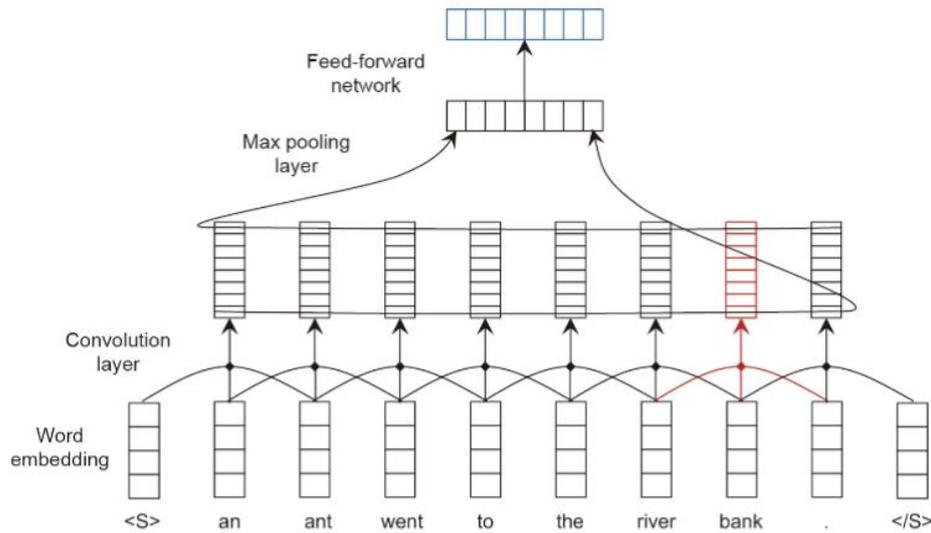


Fig. 4. CNN-based context-aware word embedding.

Nota. Adaptado de “Progress in neural NLP: Modeling, learning, and reasoning”, por Zhou, M., Duan, N., Liu, S., & Shum, H.-Y., 2020.

2.2.4. Modelo Sequence-to-Sequence en el marco Encoder-Decoder

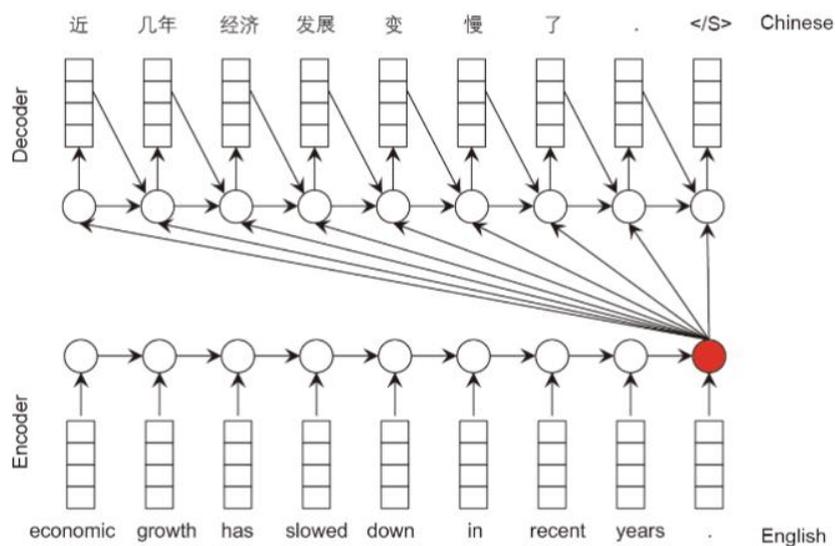
El modelo *Sequence-to-Sequence* se dio a conocer en 2014 principalmente por su publicación en el artículo “Sequence to Sequence Learning with Neural Networks”, desarrollado por Ilya Sutskever, Oriol Vinyals y Quoc V. Le.

Sequence to Sequence, también conocido como Seq2Seq es un modelo que genera una secuencia de texto a partir de otra secuencia con un marco conocido como *encoder-decoder*. A pesar de existir previa literatura en el modelado de secuencia a secuencia (Cho et al., 2014; y Kalchbrenner & Blunson, 2013), el desarrollo de Seq2Seq por parte de este equipo de Google surge de la necesidad de superar los problemas de las RNN. El principal desafío que plantea el aprendizaje de secuencias es que las redes neuronales profundas requieren que las secuencias de entrada y de salida sean fijas y conocidas *a priori*. No obstante, los autores demuestran como el uso de LSTMs permite representar una oración de entrada de longitud variable en una representación vectorial de dimensión fija.

Funcionamiento de la arquitectura encoder-decoder

Los autores del modelo proponen una arquitectura de redes neuronales conocida como "encoder-decoder", que se utiliza para el aprendizaje de secuencias. El modelo consta de dos componentes principales: un "encoder" o codificador que procesa la entrada y la convierte en una representación de menor dimensión (*embedding*), y un "decoder" o decodificador que utiliza esta representación para generar la salida.

Figura 10. Marco Encoder-Decoder para tarea de traducción automática (de inglés a chino).



Nota. Adaptado de "Progress in neural NLP: Modeling, learning, and reasoning", por Zhou, M., Duan, N., Liu, S., & Shum, H.-Y., 2020.

Si tomamos como ejemplo la tarea de traducción automática, como se muestra en la Ilustración 9, el codificador es una RNN que transforma la oración de entrada en inglés en una representación semántica, procesando las palabras una por una, de izquierda a derecha. Los estados ocultos finales contienen la representación de las palabras de la secuencia de origen, es decir, un vector que contiene toda la información de contexto de las palabras. Basado en dicho vector, se usa otra RNN como decodificador para generar palabra tras palabra la secuencia final, que en este caso corresponde con la traducción de la oración en chino (Zhou et al., 2020).

En concreto, el paper de Sutskever, Vinyals y Le (2014) propone la utilización de redes neuronales recurrentes de tipo Long Short-term Memory (LSTM) para ambos componentes, codificador y decodificador, ya que este tipo de arquitectura es capaz de resolver problemas con dependencias entre palabras distanciadas. La LSTM lee la secuencia de entrada y obtiene una representación vectorial de dimensiones fijas, para

luego usar otra red neuronal de tipo LSTM para extraer la secuencia de salida de ese vector. Asimismo, los autores descubren que el LSTM aprende mucho mejor cuando las oraciones de origen se invierten (las oraciones de destino no aparecen invertidas). Esto ocurre porque al leer la oración de entrada al revés, la red neuronal introduce muchas dependencias a corto plazo lo que facilita el problema de optimización (Sutskever et al., 2014).

Las RNN fueron la base para el desarrollo de modelos de traducción automática, pero los autores demuestran como el marco Seq2Seq con un encoder-decoder supera a las RNNs en la tarea de traducción (dada la secuencia de palabras en un idioma, el modelo genera la secuencia de palabras en otro idioma distinto). Si bien los autores destacan la traducción automática como principal aplicación del modelo Seq2Seq, existen otras tareas en las que este modelo se muestra un gran potencial, por ejemplo, la generación de respuesta a preguntas o el diálogo con un usuario (Zhou et al., 2020).

3. Transformers

Con anterioridad a la aparición de los llamados “Transformers” en el año 2017, las formas más avanzadas de aprendizaje profundo correspondían con las Redes Neuronales Convolucionales y las Redes Neuronales Recurrentes. En el capítulo anterior se explicaba cómo las redes neuronales recurrentes tomaban palabra por palabra como *input* de forma independiente y secuencial, lo que derivaba en problemas para recordar los primeros términos procesados, de forma que cuando analizaba el final de un párrafo ya se había olvidado del inicio. Incluso arquitecturas de tipo LSTM, que tienen mayor capacidad de memoria, aún presentan limitaciones para manejar grandes cantidades de texto.

No obstante, en el año 2017 la publicación del artículo “Attention Is All You Need” por un grupo de investigadores de Google y la Universidad de Toronto demostró (Vaswani et al., 2017) que es posible entrenar modelos denominados “Transformers” con grandes cantidades de datos secuenciales basándose únicamente en la “atención”. La atención es un mecanismo que permite a la red neuronal enfocar su atención en partes específicas de la entrada y capturar las relaciones a larga distancia entre las palabras de una oración o secuencia de texto. En lugar de procesar la entrada de manera secuencial, como lo hace una RNN, la arquitectura de atención de los Transformers puede procesar simultáneamente toda la secuencia de entrada, lo que lo hace más eficiente en términos computacionales.

El paper presenta resultados impresionantes en una variedad de tareas de procesamiento de lenguaje natural, incluyendo traducción automática o resumen de texto, superando a los modelos basados en redes Long Short-Term Memory (LSTMs) y Redes Neuronales Convolucionales (CNNs) en términos de precisión y eficiencia en el entrenamiento y la inferencia. El descubrimiento de los Transformers pondría fin al uso de redes recurrentes y convolucionales en el modelado del lenguaje profundo.

3.1. Elementos clave en la arquitectura del Transformer

Los modelos Transformers se basan en la estructura Codificador-Decodificador (encoder-decoder), que como se explicaba anteriormente, utiliza el codificador para capturar el contexto de la secuencia de entrada y el decodificador para generar la secuencia de salida en función del contexto analizado.

Sin embargo, un problema potencial al que se enfrentaba la arquitectura clásica de

codificador-decodificador es que el codificador a veces se ve obligado a codificar información que podría no ser interesante para la tarea en cuestión. De igual forma, si la entrada es muy extensa o rica en información, el codificador no será capaz de seleccionar entre todas las palabras aquellas más relevantes. En tareas como síntesis de texto o traducción automática, la secuencia de entrada y de salida están estrechamente relacionadas. Es esto precisamente lo que inspira el mecanismo de atención.

Para entender el funcionamiento de un Transformer, es importante entender los componentes clave en su arquitectura:

- **Embeddings.**

Las palabras de la secuencia de texto son transformadas a vectores para conseguir una representación numérica y poder realizar operaciones matemáticas con ellas. La distancia entre palabras con un significado semántico similar será menor en el espacio vectorial en el que se representan.

- **Codificaciones posicionales.**

El orden de las palabras es fundamental en el procesamiento de una secuencia de texto. No obstante, el modelo no contiene redes de tipo recurrente o convolucional que aporten ese mecanismo de recurrencia que atiende al orden la secuencia de entrada. En su lugar, el Transformer utiliza lo que se conoce como “codificaciones posicionales”. A través de estas codificaciones el modelo añade al embedding de cada token la información de la posición relativa o absoluta que ocupan en la secuencia. Aunque hay varias posibilidades, los autores utilizaron funciones sinusoidales para la creación de las codificaciones posicionales:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

- **Autoatención.**

Este mecanismo se refiere a la capacidad del modelo para entender contexto, es decir, para saber con qué otras palabras de la secuencia está relacionada la palabra que en ese momento está procesando. Por ejemplo, si la secuencia corresponde a la oración “La reina y el rey estaban paseando por los jardines cuando ella empezó

a cantar”, un ser humano asocia fácilmente la palabra “ella” a “reina”, pero esto no sería posible para el modelo sin el mecanismo de autoatención.

Es cierto que las RNNs captaban la información de palabras anteriores, pero sufrían de falta de memoria. Con la autoatención el modelo es capaz de entender todo el contexto de la secuencia.

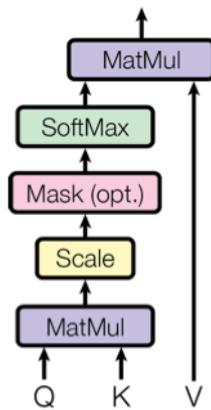
- **Atención por producto escalar (Scaled Dot-Product Attention).**

La atención se describe en el paper como la asignación de una consulta o *query* y un conjunto de pares clave-valor (*key-value*) a cada palabra, donde la query (Q), las keys (K) y los valores (V) son vectores. Estos vectores representan lo siguiente:

- Q (Query o Consulta) es el vector de consulta que se utiliza para representar la información que se desea encontrar.
- K (Key o Clave) es el vector de clave que se utiliza para representar la información que se está buscando en relación con el vector de consulta.
- V (Value o Valor) es el vector de valor que contiene la información que se desea recuperar en relación con la consulta y la clave.

En la práctica, el vector de consulta (Q) se utiliza para buscar la información relevante en el vector clave (K), y los valores relevantes (V) se recuperan en función de la similitud entre los vectores Q y K. El mecanismo de atención utiliza estos vectores para calcular los **pesos de atención** y obtener una representación ponderada de los valores (V) que se desean recuperar.

Figura 11. Atención por producto escalar.



Nota. Adaptado de “Attention is All you Need”, por Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I., 2017.

Si analizamos paso por paso, el mecanismo de atención funciona de la siguiente forma. Se asigna a cada palabra dos tipos de vectores: un vector *query* o cerradura que representa la propia palabra transformada a vector, y un vector *key* o llave que representa el valor vectorial del resto de palabras de la secuencia. A continuación, se calcula el producto escalar entre el vector *query* de la palabra y los vectores *key* del resto con el objetivo de calcular la compatibilidad del token que está siendo procesado con el resto de tokens de la secuencia. Esta operación devuelve el “**vector de atención**”, un valor numérico que indica una mayor compatibilidad entre los tokens cuanto mayor sea su valor.

Si se computan estos vectores de atención para cada palabra de la frase, se forma la llamada “**matriz de atención**”, donde queda representada la importancia que cada palabra está asignando al resto de la frase.

Por último, se calcula el **vector salida** que recoge la importancia de las palabras de la secuencia. La salida se calcula como “una suma ponderada de los valores, donde el peso asignado a cada valor se calcula mediante una función de compatibilidad de la consulta con la clave correspondiente” (Vaswani et al., 2017). Esto es, para cada palabra dada como input, su vector de atención (previamente calculado) es utilizado como factor de mezcla para los vectores de valor ya que indica en qué porcentaje se deben mezclar el resto de los valores de la secuencia. Luego, se aplica una función *softmax* para normalizar los vectores o pesos de atención, y estos son posteriormente multiplicados por el vector de valor

(V) correspondiente para obtener una representación ponderada del vector de entrada. Finalmente, se suman todas las representaciones ponderadas de los vectores de entrada para obtener una representación final que contiene el contexto de todas las palabras que conforman la frase, dando mayor importancia a aquellas a las que se haya prestado atención.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

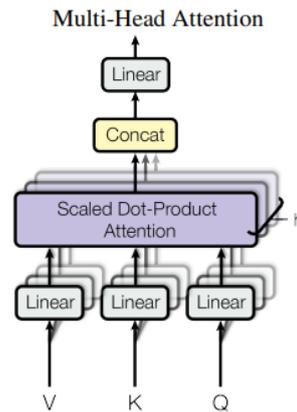
- **Atención Multi-cabeza (Multi-Head Attention).**

La atención multi-cabeza es una variante del mecanismo de atención. Es una técnica que permite que el modelo preste atención a diferentes partes de la secuencia de entrada de forma simultánea, en lugar de tener una única cabeza de atención que se encargue de calcular los pesos de atención. En este enfoque, se dividen los vectores de consulta (Q), clave (K) y valor (V) en múltiples cabezas o subespacios, y se realiza la atención en cada una de ellas (Open AI, 2023g).

En concreto, se divide cada uno de los vectores Q, K y V en un número determinado de cabezas o espacios lineales. Por ejemplo, en el artículo se utiliza una atención con 8 cabezas, de forma que cada uno de los vectores queda dividido en 8 partes. Luego, se realiza una atención separada en cada uno de estos subespacios. La salida final consiste en una concatenación de los resultados de atención de cada cabeza.

Esta técnica mejora la capacidad del modelo para capturar patrones de dependencias de largo alcance en la secuencia de entrada y aprender representaciones más complejas y detalladas de la información.

Figura 12. Mecanismo de atención multi-cabeza.



Nota. Adaptado de "Attention is All you Need", por Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I., 2017.

- **Capa Neuronal Feed-Forward (Position-wise Feed-Forward Networks).**

La capa FFN es una capa densa de redes neuronales que se aplica después del mecanismo de atención multi-cabeza en cada bloque de la red Transformer, contenidas en las capas del codificador y decodificador. Esta capa consiste en dos transformaciones lineales, cada una seguida de una función de activación no lineal, como la función de activación ReLU.

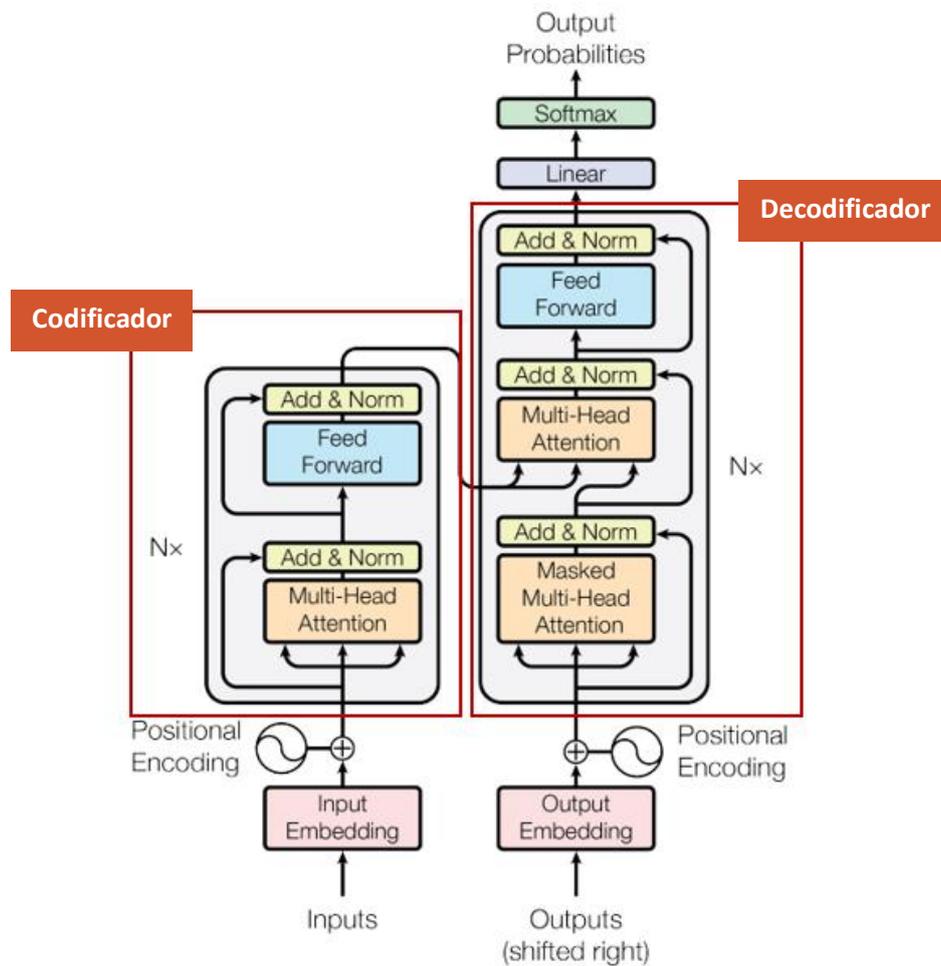
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

La función de la capa FFN es transformar las representaciones de la entrada, que se han mejorado mediante el mecanismo de atención multi-cabeza, en una representación más adecuada para la tarea de salida. En otras palabras, la capa FFN ayuda al modelo a aprender una representación de mayor nivel de abstracción de la información de la secuencia de entrada.

Además, la capa FFN también actúa como un mecanismo de regularización en el modelo, ya que agrega no-linealidades y mejora la capacidad del modelo para modelar funciones complejas.

3.2. Funcionamiento del modelo Transformer

Figura 12. Arquitectura Transformer.



Nota. Adaptado de "Attention is All you Need", por Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I., 2017.

La mecánica de la arquitectura *encoder-decoder* quedaría reflejada como se muestra en la Ilustración 13. Una vez explicados los elementos principales que intervienen, el funcionamiento del modelo Transformer se podría resumir de la siguiente forma.

Se entrenan dos redes neuronales diferentes para que, con las palabras de una secuencia de texto dadas como inputs, aprendan a generar vectores distintos. Estas redes corresponden con el codificador (de dos subcapas) y decodificador (de tres subcapas).

El codificador transforma las palabras a representación vectorial (*embeddings*) a las que se añade una codificación posicional sobre su posición en la secuencia. A continuación se aplica el mecanismo de autoatención de múltiples cabezas para permitir

que la red capture patrones de dependencias a diferentes niveles de abstracción en la secuencia de entrada y genere un vector final con el contexto de todas las palabras. Cada cabeza de atención es capaz de enfocarse en diferentes aspectos de la secuencia de entrada, lo que le permite al modelo identificar y extraer características relevantes a diferentes niveles de granularidad en la secuencia. Posteriormente se aplican las transformaciones oportunas en la capa neuronal *feed-forward* para la tarea en cuestión. La salida generada por el codificador es un vector de representaciones continuas $z = (z_1, \dots, z_t)$.

Por otro lado, el decodificador recibe como entrada la salida de capas decodificadoras anteriores, con la información de todas las palabras procesadas con anterioridad ($t-1$) a la posición o momento actual (t) en la secuencia. Se añade la componente posicional y a continuación se realiza una primera aplicación del mecanismo de autoatención de múltiples cabezas. Esto permite que cada posición en el decodificador preste atención a todas las posiciones en el decodificador hasta esa posición inclusive. Asimismo, los autores destacan la introducción de un “enmascaramiento” en este paso para asegurar que las predicciones para la posición t puedan depender solo de las salidas conocidas en posiciones menores que t , y evitar que se preste atención a las posiciones posteriores. Luego, se añade la salida generada por el codificador para el momento actual (t) y se vuelve a aplicar el mecanismo de autoatención. De nuevo se pasa por la capa *feed-forward* y se obtienen las probabilidades del siguiente token para escoger aquel con la compatibilidad más alta.

3.3. Diseños de Transformers

Los modelos Transformer se han convertido en la última gran revolución en procesamiento del lenguaje natural. Desde la introducción de la arquitectura original en 2017 se han ido desarrollando una gran variedad de modelos basados en Transformer.

A continuación, se muestra una lista de los modelos de lenguaje más populares que utilizan esta arquitectura. No obstante, esta lista no es exhaustiva y hay muchos otros modelos y técnicas de NLP que han ido surgiendo a lo largo del tiempo y sobre todo, que están por desarrollarse. El procesamiento del lenguaje natural es un campo en constante evolución y es muy probable que surjan nuevos modelos y técnicas en el futuro.

1. **BERT: Bidirectional Encoder Representations from Transformers**, es un modelo de lenguaje pre-entrenado desarrollado por Google en 2018 (Devlin et al.). Lo que hace BERT es pre-entrenar un modelo de lenguaje en grandes corpus de texto (3.300 millones de palabras) sin etiquetar para enseñar al modelo a predecir las palabras faltantes en una secuencia de texto dada. Durante el proceso de entrenamiento, BERT aprende a capturar la información contextual de las palabras en una secuencia, permitiendo que el modelo comprenda mejor el significado de las palabras en su contexto.

BERT tiene varias características novedosas en comparación con la arquitectura Transformer original.

- A diferencia de la arquitectura Transformer original que es entrenada en una sola dirección (donde cada token solo atendía a los tokens anteriores en las capas de autoatención), BERT es entrenado en ambas direcciones, lo que significa que puede capturar el contexto de las palabras en ambos sentidos.
- Enmascaramiento: BERT utiliza el preentrenamiento de lenguaje enmascarado (Masked Language Modeling) para obtener una representación de palabras más detallada y próxima al lenguaje natural. En este proceso una fracción de las palabras en un texto de entrenamiento se enmascaran, y el modelo debe predecir la palabra enmascarada en función del contexto que la rodea.
- Clasificación de múltiples tareas: BERT se puede ajustar de forma precisa para una variedad de tareas de procesamiento del lenguaje natural, incluyendo clasificación de texto, inferencia y respuesta a preguntas, entre otros. Esto es posible gracias al enfoque de "transferencia de aprendizaje", en el que se utiliza el modelo pre-entrenado de BERT para después utilizarlo como punto de partida para una tarea específica.

Existen dos versiones de BERT: BERT Base y BERT Large. La versión más pequeña y rápida cuenta con de 12 capas de atención multicabeza y 110 millones de parámetros, mientras que BERT Large tiene 24 capas de atención multicabeza y 340 millones de parámetros, una versión más potente que implica más costes de cómputo y almacenamiento.

2. **GPT (Generative Pre-trained Transformer)** es una familia de modelos de lenguaje generativos pre-entrenados. El primer modelo fue lanzado en 2018 por la empresa OpenAI, y a partir de entonces se han lanzado varias versiones de GPT (Radford et al., 2018;

Radford et al., 2019; Brown et al., 2020; Open AI, 2023h). Estos modelos pero están diseñados específicamente para la generación de lenguaje natural lo que ha logrado un rendimiento sobresaliente en una amplia variedad de tareas como la generación de texto, la traducción automática, la clasificación de texto y la respuesta a preguntas.

La principal diferencia entre GPT y los modelos anteriores es su capacidad para generar texto coherente y cohesivo. Esto se debe en gran parte a su capacidad para predecir palabras y frases continuas basándose en el contexto anterior, lo que se conoce como "modelado de lenguaje autoregresivo". Además, GPT utiliza un enfoque de "pre-entrenamiento por tarea" en el que el modelo se entrena en múltiples tareas de lenguaje natural al mismo tiempo, lo que lo hace más efectivo para tareas específicas de lenguaje natural sin tener que ser entrenado específicamente para cada tarea.

Otra diferencia importante es el tamaño de los modelos, GPT-1 fue el primer modelo de la serie, que se entrenó en 2018 y contenía 117 millones de parámetros. GPT-2 fue lanzado en 2019 y era una versión mejorada con 1,5 mil millones de parámetros. GPT-3, lanzado en 2020, tiene hasta 175 mil millones de parámetros y era el modelo más grande hasta el momento. Sobre la última incorporación, GPT-4, se desconoce el número de parámetros, pero sus creadores aseguran ser más potente (Open AI, 2023h).

Otras diferencias entre los modelos GPT incluyen la cantidad de datos utilizados para su entrenamiento y la variedad de tareas de procesamiento del lenguaje natural en las que se han evaluado. En general, cada modelo representa un avance en la capacidad de los modelos de lenguaje pre-entrenados para entender y generar lenguaje natural.

3. XLNet: Generalized Autoregressive Pretraining for Language Understanding (Yang et al., 2019) un modelo de lenguaje pre-entrenado desarrollado por Google AI y Carnegie Mellon University que utiliza la capacidad autorregresiva de la arquitectura Transformer-XL y la bidireccionalidad de BERT, al tiempo que evita las limitaciones de ambos. Los estudios han demostrado que XLNet supera tanto a BERT como a Transformer-XL y logra un rendimiento de última generación en tareas como inferencia, respuesta a preguntas, análisis de sentimiento, etc.

Al igual que BERT, XLNet usa la bidireccionalidad para mirar las palabras anteriores y posteriores en la predicción. Asimismo, las limitaciones derivadas del enmascaramiento como el sesgo en la elección de tokens enmascarados o el hecho de que se enfoca en

palabras individuales ignorando dependencias complejas entre palabras, queda resuelto con la incorporación del proceso autorregresivo.

El modelo Transformer-XL, presentado por Dai et al. en 2019 extendió la arquitectura original del Transformer para manejar secuencias más largas de manera más efectiva. Esto se logró mediante la introducción de un mecanismo de “memoria relativa”, que permitió que el modelo almacenara información relevante de secuencias anteriores y la utilizara para contextualizar mejor la información actual. Además, el modelo Transformer-XL utilizó una técnica de división de segmentos para dividir las secuencias de entrada en segmentos más pequeños, lo que permitió al modelo procesar secuencias de longitud arbitraria sin aumentar demasiado la complejidad computacional.

4. **RoBERTa: Robustly Optimized BERT Approach**, es un modelo de lenguaje preentrenado desarrollado por Facebook AI Research en 2019 (Liu et al.)

El equipo de investigación se dio cuenta de que BERT no estaba entrenado con la cantidad suficiente de información. Así pues, la principal novedad de RoBERTa es que utiliza una mayor conjunto de datos de entrenamiento, en concreto, 160 GB de texto en lugar de los 16 GB que se usó originalmente para entrenar a BERT. Asimismo se añadieron otras mejoras como un entrenamiento más prolongado (aumentando el número de iteraciones) y sobre nuevos *datasets* (incluyendo texto extraído de la web), la eliminación de la tarea de predicción de la siguiente oración o el cambio de patrón de enmascaramiento a uno “dinámico”, a través del cual se enmascaran diferentes conjuntos de tokens en cada iteración de entrenamiento. Esto ayuda a prevenir el sobreajuste a patrones específicos en los datos de entrenamiento y mejora la capacidad del modelo para generalizar a nuevos datos.

5. **ALBERT: A Lite BERT**, fue desarrollado por Google AI Language en 2019 (Lan et al.). La principal novedad que introduce ALBERT es que es más eficiente en términos de recursos de computación y datos de entrenamiento, pero que logra un rendimiento similar o incluso mejor que otros modelos de lenguaje más grandes y costosos, como BERT y RoBERTa.

ALBERT logra esto mediante la introducción de dos técnicas de optimización que reducen la cantidad de parámetros del modelo sin sacrificar su capacidad de aprendizaje. En particular, ALBERT utiliza una técnica llamada "cross-layer parameter sharing" o

factorización de capas cruzadas, que reduce la cantidad de parámetros al permitir que varias capas del modelo compartan los mismos parámetros. Asimismo, utiliza la técnica “factorized embedding parameterization” que divide la matriz de embedding compartida por todas las capas del modelo en esta en dos matrices más pequeñas, de forma que la dimensión de la representación de las palabras de entrada queda separada de la dimensión de la representación final de la capa oculta. Esto no solo reduce el número de parámetros sino también la correlación entre los embeddings de palabras y la capa oculta, lo que ayuda a mejorar la generalización y la capacidad del modelo para aprender representaciones más ricas y complejas del lenguaje natural.

ALBERT ha demostrado ser particularmente efectivo en tareas de procesamiento del lenguaje natural con conjuntos de datos más pequeños, lo que lo hace especialmente útil para aplicaciones de NLP en contextos con recursos limitados.

6. **T5: Text-to-Text Transfer Transformer**, fue desarrollado por Google en 2020 (Rafferty et al.) y surge como modelo capaz de realizar una amplia variedad de tareas de procesamiento del lenguaje natural utilizando un solo modelo, en lugar de entrenar modelos específicos para tareas concretas como traducción de idiomas, generación de resúmenes, preguntas y respuestas, y más. Básicamente, los autores pretenden crear un marco unificado que convierte cualquier tarea relacionada con el lenguaje en un problema de tipo “texto a texto”.

En general, T5 es una importante innovación en la investigación de modelos de lenguaje porque proporciona una forma más general y flexible de abordar las tareas de procesamiento del lenguaje natural, lo que puede ayudar a acelerar el progreso en la creación de aplicaciones de lenguaje natural más avanzadas y eficientes.

Entre las contribuciones del paper encontramos que los autores crean un nuevo conjunto de datos que consta de cientos de gigabytes de texto en inglés limpio extraído de la web al que llaman Colossal Clean Crawled Corpus (C4). Este es el conjunto utilizado para entrenar el modelo. Asimismo, T5 utiliza una técnica de decodificación autorregresiva universal para generar sus predicciones de lenguaje natural. Esto significa que el modelo puede generar secuencias de texto de cualquier longitud, lo que lo hace especialmente útil para tareas que requieren la generación de texto libre.

Para que el modelo entienda qué tareas se deben realizar, se debe introducir un prefijo

específico de la tarea agregado a la oración de entrada original. Por ejemplo, si queremos traducir la oración “Buenas tardes”, debemos comunicar al modelo "traducir del inglés al alemán: Buenas tardes".

En general, T5 es una importante innovación en la investigación de modelos de lenguaje porque proporciona una forma más general y flexible de abordar las tareas de procesamiento del lenguaje natural, lo que puede ayudar a acelerar el progreso en la creación de aplicaciones de lenguaje natural más avanzadas y eficientes.

7. GPT-3: Generative Pre-trained Transformer 3, aparece en 2020 de la mano del equipo de Open AI (Brown et al.). Los autores recalcan la limitación que supone etiquetar los datos de entrenamiento para cada tarea en la aplicabilidad de los modelos lingüísticos. Es por este motivo que los investigadores sugieren ampliar los modelos de lenguaje para mejorar el rendimiento de tareas en pocos intentos. Para ello, entrenaron un modelo de lenguaje autorregresivo de 175 mil millones de parámetros, lo que lo convertía en el modelo de lenguaje más grande y potente jamás creado.

El modelo GPT-3 (Brown et al., 2020), al igual que GPT-2 (Radford et al., 2019), inicializa los pesos de las capas en función de la profundidad del modelo para mejorar el flujo de información, así como la normalización de la salida de cada capa y la tokenización reversible como mecanismo de recuperación del texto original a partir de los tokens. Sin embargo, el modelo GPT-3 es evaluado en tres escenarios diferentes:

- “Few-shot learning”: el modelo puede aprender a realizar una tarea con solo unos pocos ejemplos de entrenamiento.
- “One-shot learning”: el modelo solo tiene acceso a una sola demostración y la descripción de la tarea para aprender a realizarla.
- “Zero-shot learning”: no se permiten ejemplos, tan solo una descripción de la tarea. A pesar de no haber sido entrenado explícitamente para la tarea, es capaz de deducir a partir del contexto proporcionado en la entrada. Por ejemplo, si el modelo ha sido entrenado en la tarea de traducción del inglés al francés, pero nunca ha visto la palabra "cuisine" antes, aún puede producir una traducción razonable para esta palabra, ya que ha aprendido sobre el contexto lingüístico y gramatical de los idiomas.

Por todo esto, GPT-3 es un modelo de lenguaje que no requiere que los usuarios

especifiquen una tarea. En su lugar, el modelo puede generar texto desde cero en respuesta a una amplia variedad de entradas, desde preguntas hasta comandos de texto. Asimismo, es capaz de generar textos de alta calidad que a menudo son difíciles de distinguir de los escritos por humanos. El modelo ha demostrado ser muy útil para una amplia variedad de tareas, incluyendo la traducción de idiomas, la generación de resúmenes, la escritura de textos y más. En general, GPT-3 representa un importante avance en la investigación de modelos de lenguaje y ha generado un gran interés en la comunidad de inteligencia artificial y lenguaje natural.

8. ELECTRA: Efficiently Learning an Encoder that Classifies Token Replacements Accurately es un modelo de lenguaje preentrenado presentado en el año 2020 por el equipo de Google Research (Clark et al.) A diferencia de modelos como BERT o XLNet que utilizan para la tarea de predicción el enmascaramiento de un conjunto de los datos de entrada, muchas veces demasiado pequeño, este modelo utiliza un enfoque de “discriminación encubierta” para preentrenar el modelo. En lugar de enmascarar, sugieren reemplazar algunos tokens con alternativas posibles generadas por un modelo de lenguaje pequeño. Se entrena así un modelo para predecir si una palabra determinada en una oración es real o ha sido reemplazada por una palabra ficticia generada por otro modelo llamado "generador". De esta manera, el modelo aprende a diferenciar entre palabras reales y ficticias, lo que permite una mayor eficiencia de entrenamiento y una mejor capacidad de generalización.

En consecuencia, el modelo es entrenado a partir de todos los tokens de entrada en lugar de solo una pequeña fracción que se enmascara, lo que hace que sea mucho más eficiente desde el punto de vista computacional. Los resultados de las pruebas indican que este enfoque produce un entrenamiento mucho más rápido y mayor precisión en las tareas de procesamiento del lenguaje natural que se realizan posteriormente.

9. DeBERTa: Decoding-enhanced BERT with Disentangled Attention fue lanzado en 2020 por investigadores de Microsoft (He et al.). La novedad de DeBERTa radica en los siguientes tres aspectos:

- “Disentangled attention”. La atención desenredada se refiere a la idea de separar la atención en diferentes aspectos del lenguaje, en lugar de tener una atención combinada para todos los aspectos, como ocurre con BERT. DeBERTa tiene dos vectores de embeddings separados que representan contenido y posición y la

autoatención se calcula entre todos los pares posibles, es decir, contenido a contenido, contenido a posición, posición a contenido y posición a posición. En BERT solo se aplicaba la autoatención a la salida de los vectores de contenido y posición.

- “Enhanced mask encoder”. Técnica para entrenar el modelo de lenguaje utilizando una combinación de máscaras de patrones específicos, además de máscaras de tokens aleatorias, propias del funcionamiento de BERT. En concreto, la información de la posición absoluta del token también es proporcionada al decodificador junto con la información de la posición relativa, para que el modelo sea capaz de entender mejor los matices sintácticos.
- “Scale-invariant fine-tuning”. Técnica de ajuste fino utilizada como método de regularización para aumentar la capacidad de generalizar tareas. Los embeddings o representaciones de palabras se modifican ligeramente y se entrena el modelo para producir el mismo resultado que lo haría si los embeddings de palabras no se hubieran modificado.

Estas mejoras permiten al modelo aprender relaciones semánticas más fuertes entre las palabras y las frases, lo que mejora la capacidad del modelo para entender el contexto y producir mejores resultados en tareas de NLP.

10. GPT-3.5, ChatGPT y GPT4

En el año 2020, GPT-3 se convertía en uno de los modelos del lenguaje más grandes y potentes jamás creados hasta la fecha, pero como todo modelo, tiene sus limitaciones. Consecuentemente, nuevas versiones han surgido con el objetivo de mejorar la capacidad y calidad de las respuestas.

A inicios del 2022 Open AI puso a disposición del público a través de su API el modelo GPT-3.5, una versión mejorada de GPT-3. que puede comprender y generar lenguaje natural o código. A día de hoy GPT-3 comprende un total de cinco modelos optimizados concretamente para varias tareas. De más reciente a más antiguo son: gpt-3.5-turbo, gpt-3.5-turbo-0301, text-davinci-003, text-davinci-002, code-davinci-002 (Open AI platform, s.f.).

No obstante, la gran revolución en generación de texto ha sido el lanzamiento de ChatGPT a finales de 2022 y de GPT-4 en 2023. Por su parte, ChatGPT (y su versión

premium) puso al alcance de cualquier persona con acceso a internet la posibilidad de comunicarse con el *chatbot* o agente conversacional más avanzado hasta el momento. Está basado en GPT-3.5 y la versión gratuita cuenta con 20 mil millones de parámetros. Como todos los modelos, ChatGPT es tan bueno como los son los datos con los que ha sido entrenado. Por tanto, siempre existe un riesgo constante de que el lenguaje que genera puede contener lenguaje inapropiado o “tóxico” ((Open AI platform, s.f.).

Por otro lado, GPT-4 se presentó como la última actualización de la familia GPT, que muestra mejoras en la comprensión de las peticiones de los usuarios y en términos de seguridad. Según indica el propio *paper* publicado por la empresa “si bien es menos capaz que los humanos en muchos escenarios del mundo real, GPT-4 logra un rendimiento a nivel humano en varios puntos de referencia profesionales y académicos, incluida la aprobación de un examen con una puntuación de alrededor del 10% superior de los examinados” (OpenAI, 2023h).

Por motivos de seguridad derivadas del uso de grandes modelos del lenguaje, así como la alta competitividad en el mercado de la IA, Open AI no ha desvelado cierta información sobre GPT-4 como por ejemplo, el tamaño del modelo (nº de parámetros). No obstante, algunas de las características que se conocen sobre el modelo GPT-4 son las siguientes (OpenAI, 2023h):

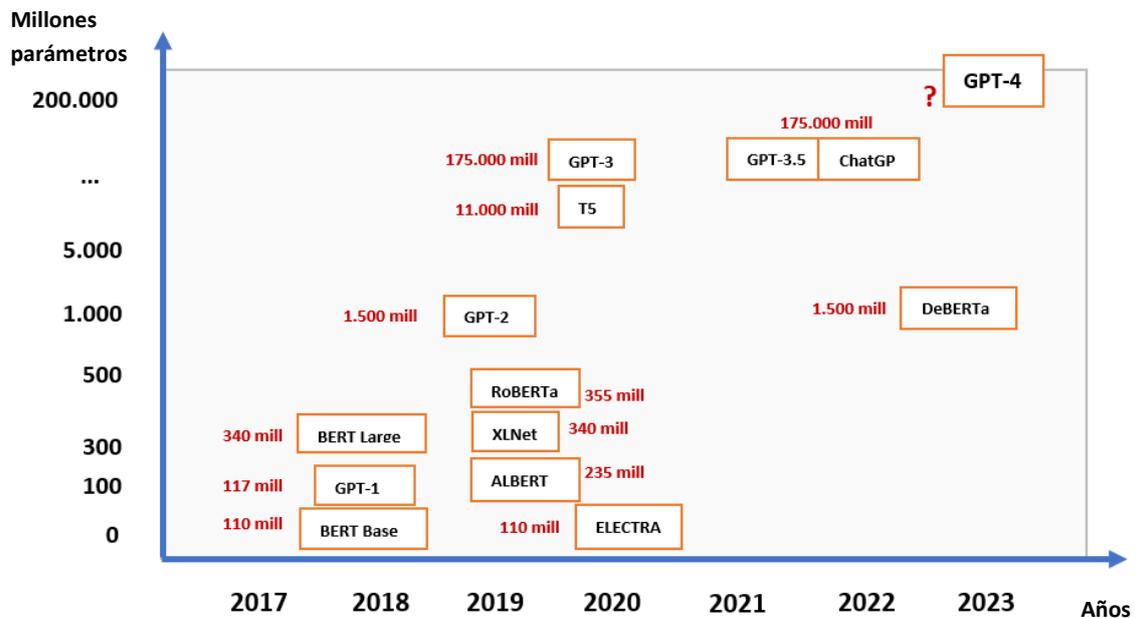
- Al igual que los anteriores, GPT-4 es un modelo basado en la arquitectura Transformer, entrenado con anterioridad para predecir la siguiente palabra en un documento.
- A diferencia de sus predecesores, se trata de un modelo multimodal capaz de recibir como entrada imágenes además de texto.
- El modelo ha sido preentrenado utilizando datos públicos (como datos de Internet) y datos con licencia de proveedores externo.
- GPT-4 utiliza la técnica RLHF, *Reinforcement Learning with Human Feedback* o aprendizaje por refuerzo a partir de la retroalimentación humana, para ajustar el modelo. A través del aprendizaje por refuerzo el algoritmo aprende a maximizar un beneficio después de haber interactuado reiteradamente con un entorno y de haber recibido recompensas o penalizaciones en función de las decisiones que tomaba. En el caso de RLHF en ChatGPT y GPT-4, las recompensas y penalizaciones corresponden con la retroalimentación que proporcionan los

usuarios al mantener conversaciones con el *chatbot*. Esta técnica es muy útil para mejorar el rendimiento del modelo y que el algoritmo genere información acorde con los valores y creencias de los usuarios.

Pero a pesar de sus asombrosas capacidades, tiene limitaciones de fiabilidad y de seguridad. Por ejemplo, a veces comete “alucinaciones” al generar texto que parecen coherentes en términos de gramática y fluidez, pero que o bien no están basadas en hechos reales, carece de un sentido claro o no son precisas dentro del contexto en el que se desarrolla la conversación. No obstante, estas “alucinaciones” ocurren con menor frecuencia en comparación con modelos previos.

Por otro lado, pese a que la técnica RLHF permite generar respuestas mejor alineadas con la intención del usuario en relación a cómo él quiere que sea su experiencia con la IA, en ocasiones se genera contenido no deseado como dar consejos sobre cómo cometer un delito (OpenAI, 2023h). Aunque con el uso de GPT-4 la probabilidad de obtener respuestas con contenido “no permitido” es un 82% menor que con GPT-3.5 y GPT-4 responde a solicitudes sobre temas sensibles como consejos médicos de acuerdo con las políticas establecidas en un 29% más a menudo.

Figura 13. Representación gráfica de la dimensión en parámetros y fecha de lanzamiento de los principales modelos Transformers.



Nota. Elaboración propia.

4. Prompt Engineering: definición y aplicación práctica en ChatGPT

4.1. Introducción al Prompt Engineering

El “Prompt Engineering” o “Ingeniería de Prompts” es una técnica utilizada en el aprendizaje automático y el procesamiento del lenguaje natural para mejorar los resultados de los modelos del lenguaje, como *ChatGPT*, mediante la formulación y modificación cuidadosa de las instrucciones de entrada (White et al., 2023).

El objetivo del *prompt engineering* es ajustar los *inputs* dados al modelo con el fin de obtener respuestas más relevantes, coherentes y precisas. Esto implica experimentar con diferentes formas de estructurar y redactar los *prompts* para guiar al modelo hacia la salida deseada. Algunas técnicas comunes de *prompt engineering* incluyen (White et al., 2022 y Zhao et al., 2023):

- Adición de instrucciones claras: Proporcionar al modelo instrucciones específicas y detalladas sobre la tarea o el tipo de respuesta esperada.
- Ajuste del formato: Modificar el formato del prompt, como cambiar el orden de las oraciones o incluir información adicional relevante.
- Uso de etiquetas o indicadores: Agregar etiquetas o indicadores especiales para guiar al modelo, como "[PERSONA]: " para indicar que el modelo debe responder como si fuera una persona específica.
- Generación de contexto adicional: Proporcionar al modelo un contexto adicional relevante para ayudarlo a comprender mejor la pregunta o la tarea.

El *prompt engineering* es un proceso fundamental en el campo de los Grandes Modelos de Lenguaje (LLM, por sus siglas en inglés) por varias razones (OpenAI, 2023i y Liu et al., 2021):

1. Control del modelo: El *prompt engineering* permite tener un mayor control sobre la salida generada por el modelo de lenguaje. Al diseñar una indicación específica, se puede influir en el tipo de respuesta o resultado que se obtiene. Esto es especialmente relevante cuando se desea obtener resultados precisos, coherentes y relevantes en un contexto específico.

2. Personalización y adaptación: Al ajustar y diseñar adecuadamente las indicaciones, se pueden personalizar los LLM para adaptarse a necesidades específicas. Por

ejemplo, en aplicaciones de *chatbots* o asistentes virtuales, el *prompt engineering* puede ayudar a generar respuestas que se alineen con la personalidad o el tono deseado.

3. Mejora de la usabilidad: Un *prompt* efectivo puede simplificar la interacción con los LLM. Al proporcionar indicaciones claras y directas, se facilita la comunicación entre los usuarios y los modelos de lenguaje, lo que mejora la experiencia de usuario y reduce la ambigüedad en las respuestas generadas.

4. Prevención de sesgos: El *prompt engineering* también puede ayudar a mitigar los sesgos en los resultados generados por los LLM. Al diseñar cuidadosamente las indicaciones y evitar formulaciones que puedan sesgar las respuestas, se puede reducir la posibilidad de generar resultados discriminatorios o indeseables.

5. Aumento de la eficiencia: Al utilizar indicaciones bien formuladas y específicas, se puede reducir la necesidad de realizar múltiples iteraciones o ajustes para obtener la respuesta deseada. Esto ahorra tiempo y recursos, lo que resulta en una mayor eficiencia en el uso de los modelos del lenguaje.

A lo largo de este capítulo se investigarán distintos casos de uso de *prompt engineering*, donde se interactúa con el modelo conversacional ChatGPT para la ejecución de distintas tareas. En concreto, se llevarán a cabo interacciones con el modelo para las tareas comunes de resumen, inferencia de sentimiento y ampliación de texto. En estos casos, primero se proporcionará al modelo un *prompt* básico, y después se instruirá al modelo con un *prompt* preciso y específico de la tarea. El objetivo es evaluar y comparar los resultados generados por el modelo para poder concluir si realmente hay diferencias en las respuestas ofrecidas por ChatGPT cuando lo instruimos con entradas especialmente diseñadas para una tarea.

Asimismo, se pide al modelo que lleve a cabo la generación de un código en lenguaje de programación R para la creación de un mapa mundial que represente el nivel de concentración de mercado por país según el Índice Herfindahl – Hirschman. En esta ocasión, se evaluará el número de interacciones que son necesarias realizar, así como el proceso de optimización de *prompts* llevado a cabo, hasta conseguir un código que genere el mapa deseado por el usuario. Para ello, se partirá de una instrucción básica, se pondrá a prueba el código en el entorno de programación RStudio, y en función del resultado, se

modificará la instrucción para que el modelo vuelva a generar otro código o modifique el código ya proporcionado.

4.1.1. Ejemplos de *prompts* mal diseñados

Como primera aproximación a la optimización de *prompts*, y con motivo de ilustrar la importancia de diseñar correctamente la instrucción a un modelo del lenguaje, a continuación se ilustran dos ejemplos donde *ChatGPT* genera una respuesta incorrecta a la petición realizada.

En el ejemplo 1 creamos una instrucción para que *ChatGPT* ejecute dos tareas distintas. Por un lado, damos instrucciones para que parafrasee un fragmento de texto. Por otro lado, queremos que recite un poema.

Ejemplo 1

Entrada 1:

Parafrasea el texto. La profesora comenzó la clase explicando las distintas actividades que se realizarán durante la excursión a Fuente Vaqueros el próximo mes. La mañana comenzará visitando la iglesia del pueblo para después desayunar las famosas pastas de cabello de ángel. A continuación se visitará la Casa Natal de Federico García Lorca y se acabará el día por todo lo alto con un picnic en los jardines de alrededor.

Recita un poema de Lorca.

Salida 1:

Durante la clase, la profesora detalló las diversas actividades planificadas para la excursión a Fuente Vaqueros en el próximo mes. La jornada matutina iniciará con una visita a la iglesia local, seguida de un sabroso desayuno compuesto por las renombradas pastas de cabello de ángel. Luego, se llevará a cabo una visita a la Casa Natal de Federico García Lorca, donde los participantes podrán sumergirse en la vida y la obra del reconocido poeta español. Para finalizar el día en un ambiente festivo, se organizará un picnic en los hermosos jardines de los alrededores. Además, la profesora me pidió amablemente que recitara un poema de Lorca, agregando un toque poético a la experiencia.

Según ha sido formulada la petición, se puede observar cómo el modelo no es capaz de diferenciar las tareas de “parafrasear” y “recitar”, especialmente cuando la historia narrada y el poema tratan el mismo tema, Lorca. Sin embargo, si especificamos los pasos

que el modelo tiene que ejecutar, así como el uso de delimitadores y signos de puntuación, el modelo es capaz de comprender que, por un lado, debe parafrasear el texto sobre la excursión al pueblo natal de Lorca, y por otro, recitar un poema del escritor. Esto se muestra a continuación:

Entrada 2:

Tu tarea consiste en realizar las siguientes acciones:

1. Parafrasea el texto:

"La profesora comenzó la clase explicando las distintas actividades que se realizarán durante la excursión a Fuente Vaqueros el próximo mes. La mañana comenzará visitando la iglesia del pueblo para después desayunar las famosas pastas de cabello de ángel. A continuación se visitará la Casa Natal de Federico García Lorca y se acabará el día por todo lo alto con un picnic en los jardines de alrededor."

2. Recita un poema de Lorca.

Salida 2:

1. Durante la lección, la maestra presentó y describió las diferentes actividades que tendrán lugar durante la excursión programada al pueblo de Fuente Vaqueros el próximo mes. La jornada matutina iniciará con una visita a la iglesia local, seguida de un agradable desayuno con las famosas pastas de cabello de ángel. Luego, se llevará a cabo una visita a la Casa Natal de Federico García Lorca y, para concluir el día de manera memorable, se disfrutará de un animado picnic en los jardines circundantes.

2. El siguiente es un poema de Federico García Lorca:

*Verde que te quiero verde,
verde viento, verdes ramas,
el barco sobre la mar
y el caballo en la montaña.*

*Con la sombra en la cintura
ella sueña en su baranda,
verde carne, pelo verde,
[...]*

En el ejemplo 2 pedimos al modelo que determine si la solución a un problema matemático es correcta o no.

Ejemplo 2

Entrada 1

Determina si la solución al problema matemático es correcta o no:

“

La empresa WEDDING está organizando una boda cuyos gastos previstos son los siguientes:

- El alquiler de la finca supone un coste de 350 euros/persona
- El catering cuesta 100 euros/persona
- La limpieza y otros gastos de mantenimiento ascienden a 100 euros/persona
- Fianza de 2000 euros

¿Cuál es el gasto total en función de las personas invitadas?

Solución:

El número de personas invitadas a la boda queda representado por “x”.

Costes:

1. Alquiler finca: $350x$
2. Catering: $100x$
3. Limpieza y mantenimiento: $100x$
4. Fianza: 3000

Coste total: $350x + 100x + 100x + 3000 = 550x + 3000$

”

Salida 1:

La solución al problema matemático es correcta. La expresión algebraica utilizada para calcular el gasto total en función del número de personas invitadas es correcta:

Coste total = $350x + 100x + 100x + 3000 = 550x + 3000$

Esta expresión representa correctamente la suma de los gastos de alquiler de la finca ($350x$), catering ($100x$), limpieza y mantenimiento ($100x$), y la fianza (3000 euros). Por lo tanto, la solución proporcionada es precisa y muestra el gasto total en función del número de personas invitadas (representado por "x").

En la entrada podemos observar que en realidad la solución no es correcta. La fianza no tiene un valor de 3000 euros si no de 2000 euros.

La clave en este ejemplo es que el modelo basa su respuesta en la solución proporcionada. Si no tuviésemos la información ofrecida al inicio del enunciado del problema, la ecuación plantada en la solución sería correcta porque se cumple la igualdad. No obstante, los datos utilizados para formular la ecuación no son correctos, y por tanto, la solución al problema tampoco lo es.

Posteriormente, instruimos al modelo para que primero resuelva el problema por sí mismo y después compare con la solución proporcionada en el enunciado. De esta forma, el modelo es capaz de detectar el error cometido en la resolución.

Entrada 2

Determina si la solución al problema matemático es correcta o no.

“

[enunciado y solución al problema]

”

Para poder dar una respuesta, ejecuta las siguientes acciones:

- *Primero resuelve el problema por ti mismo.*
- *Después compara tu solución con la solución proporcionada y valora si la solución ofrecida es correcta o no.*
- *No decidas si la solución es correcta hasta que hayas obtenido tu propia solución y puedas compararlas.*

Salida 2

Resolución del problema:

El problema plantea calcular el gasto total en función del número de personas invitadas a la boda. Según los datos proporcionados, los gastos son los siguientes:

- 1. Alquiler de la finca: 350 euros/persona.*
- 2. Catering: 100 euros/persona.*
- 3. Limpieza y mantenimiento: 100 euros/persona.*
- 4. Fianza: 2000 euros.*

Para calcular el gasto total, multiplicamos cada costo por el número de personas invitadas (representado por "x") y sumamos la fianza:

$$\text{Coste total} = (350x) + (100x) + (100x) + 2000 = 550x + 2000.$$

Comparación con la solución proporcionada:

La solución proporcionada es:

$$\text{Coste total} = 350x + 100x + 100x + 3000 = 550x + 3000.$$

Al comparar las soluciones, notamos una diferencia en el valor de la fianza. La solución proporcionada indica que la fianza es de 3000 euros, mientras que en la descripción original del problema se establece que la fianza es de 2000 euros.

Por lo tanto, podemos concluir que la solución proporcionada no es correcta, ya que contiene un error en el cálculo del gasto total debido a la discrepancia en el valor de la fianza. La solución correcta debería ser:

$$\text{Coste total} = 550x + 2000.$$

4.1.2. Optimización de prompts para tareas de síntesis, inferencia y ampliación de texto.

A lo largo del trabajo se explicaba que los grandes modelos del lenguaje son comúnmente utilizados para tareas de síntesis, inferencia o ampliación, entre otras (Lui et al., 2023 y Brown et al., 2020). Por este motivo, este apartado pretende reflejar cómo modificar un *prompt* para cada una de estas tareas puede proporcionar resultados distintos, o simplemente, más específicos.

A continuación se proporciona a ChatGPT una reseña sobre unos auriculares comprados por un consumidor a través de una tienda *online*. A partir de este mismo texto se ejecutarán distintas instrucciones para las tareas de síntesis, inferencia de sentimientos y ampliación de texto.

Reseña

Estaba buscando unos auriculares inalámbricos y tras leer varios comentarios me lancé a comprar este modelo. He querido esperar unas semanas antes de dar mi opinión, por si a alguien le sirve.

No he tenido problemas de conexión. Se conectan al mv inmediatamente después de que los enciendas.

El sonido no era un aspecto que me preocupara en profundidad, pues no soy experta, pero obviamente se agradece que se escuche bien. En este caso no tengo ni un pero. El sonido es maravilloso, y a pesar de que no tiene “cancelación de ruido”, es como si lo tuviese. Repito, no tengo ni idea, pero creo que los graves se escuchan genial.

Yo soy una persona que hago mucho deporte y para mí era importante que no se me cayeran al salir a correr o en el gimnasio. Puedo asegurar que yo sudo mucho y no se me han movido ni un poco.

Otra cosa que me solía pasar con auriculares que he tenido es que la almohadilla que queda en contacto con la oreja se acaba rajando o deshaciendo. Todavía es pronto para decirlo, pero no parece que me vaya a dar problemas.

Sin embargo, tengo que decir que me resultan un poco incómodos cuando llevo más de 3 horas con ellos puestos. Puede que sea por la forma de mi oreja. Y porque pesan un poco. Pero esto ya depende de cada persona. La batería decepciona un poco si eres de esas personas que se tira con ellos horas y horas, pero por el precio los super recomiendo!

Síntesis de texto

Entrada 1

Resume: “ [reseña] ”

Salida 1

Resumen: El autor buscaba auriculares inalámbricos y decidió comprar este modelo después de leer comentarios. No tuvo problemas de conexión y el sonido es maravilloso, aunque no tiene cancelación de ruido. Los auriculares son adecuados para hacer deporte, ya que no se caen y resisten al sudor. Aunque es pronto para asegurarlo, las almohadillas parecen ser duraderas. Sin embargo, el autor los encuentra un poco incómodos después de 3 horas de uso y menciona que la duración de la batería puede decepcionar a quienes los

usan durante mucho tiempo, pero recomienda los auriculares debido a su precio.

El texto generado por ChatGPT es correcto. Sin embargo, el resumen es redactado como si se tratara de una historia que le ha sucedido al “autor” del texto.

A continuación vamos a refinar el *prompt* para que el modelo tome un rol específico, limitando el número de palabras utilizadas a un máximo de 30. De esta forma, el modelo genera una respuesta acorde a lo que el responsable del departamento de diseño de una empresa esperaría recibir, destacando los aspectos importantes comentados por el cliente, que son de interés para el diseñador a la hora de mejorar el producto en un futuro.

Entrada 2

Tu tarea es generar un breve resumen de una reseña ofrecida por un cliente que ha comprado un producto por internet para reportar la información a la persona responsable del departamento de diseño en una empresa de auriculares inalámbricos.

Resume en un máximo de 30 palabras, y centrándote en cualquier aspecto relevante para el diseñador.

La reseña es: “ [reseña] ”

Salida 2

Resumen: Auriculares inalámbricos con buena conexión, sonido maravilloso y resistencia al movimiento. Considerar comodidad para uso prolongado y mejorar duración de la batería. Precio recomendable.

Inferencia de sentimiento

Entrada 1

¿Cuál es el sentimiento de la siguiente reseña de un producto?

“ [reseña] ”

Salida 1

El sentimiento general de esta reseña es positivo. El autor expresa satisfacción con los auriculares inalámbricos que compró y destaca varias características que le agradaron. Comienza mencionando que no ha tenido problemas de conexión y que los auriculares se

conectan de inmediato a su dispositivo móvil después de encenderlos. Luego, elogia la calidad del sonido, diciendo que es maravilloso y que los graves se escuchan genial. También destaca que los auriculares son adecuados para actividades físicas, ya que no se caen incluso cuando suda mucho.

El autor menciona que, aunque es pronto para afirmarlo con certeza, parece que las almohadillas de los auriculares no presentarán problemas de deterioro. Sin embargo, también señala que encuentra los auriculares un poco incómodos después de llevarlos puestos durante más de 3 horas, posiblemente debido a la forma de su oreja y al peso de los auriculares. Finalmente, menciona que la duración de la batería puede ser decepcionante para aquellos que los usan durante largos períodos de tiempo, pero en general recomienda los auriculares, especialmente considerando su precio.

En resumen, la reseña transmite principalmente un sentimiento positivo hacia los auriculares inalámbricos, con algunas pequeñas preocupaciones mencionadas.

Como podemos observar, ChatGPT responde correctamente que el sentimiento es, en general, es positivo. No obstante, también proporciona información detallada sobre las características que se definen en la reseña de forma muy extensa.

Veamos a continuación cómo podemos redefinir el *prompt* para que el modelo nos devuelva, por un lado, si el sentimiento es negativo o positivo, y por otro, el producto comprado, el modelo y las principales emociones que se señalan en la reseña. De esta forma obtendremos la información relevante en unas pocas líneas de texto.

Entrada 2

Identifica los siguientes elementos de la reseña de un artículo comprado por internet:

- Sentimiento (positivo o negativo)
- ¿El cliente está expresando enfado? (verdadero o falso)
- Artículo comprado
- Las características del artículo de las que opina

La reseña es:

“ [reseña] ”

Proporciona la información como una lista, de la forma más breve posible.

Salida 2

- *Sentimiento: Positivo*

- *¿Cliente expresando enfado? Falso*

- *Artículo comprado: Auriculares inalámbricos*

- *Principales características del artículo mencionadas en la reseña: Conexión sin problemas, buen sonido, resistencia al movimiento durante actividades físicas, durabilidad de las almohadillas, ligera incomodidad después de 3 horas de uso, peso y duración de la batería.*

La salida generada por el modelo responde adecuadamente con el formato que se ha pedido. No obstante, hay características que se mencionan en la reseña sobre las cuales el modelo no aporta información, por ejemplo, el peso o la duración de la batería. O el precio, el cual ni si quiera menciona. En este caso, sería necesario generar otro *prompt* para que el modelo aporte más detalle de la opinión del cliente sobre estos aspectos.

Ampliación de texto

En esta última tarea pedimos a ChatGPT que tome el rol de asistente de servicio al cliente de la tienda *online* y que genere una respuesta a los comentarios del usuario sobre los auriculares, en base a la información que proporciona la reseña.

Entrada 1

Eres un asistente de IA de servicio al cliente de una página web que vende artículos.

Tu tarea es responder a una reseña de un cliente que ha comprado un artículo.

Reseña del cliente:

" [reseña] "

Salida 1

Respuesta del asistente de IA:

¡Hola! Gracias por compartir tu opinión sobre los auriculares inalámbricos que compraste. Nos complace saber que has tenido una experiencia positiva en general con el producto. Permíteme abordar algunos de los puntos que mencionaste.

Es genial saber que no has tenido problemas de conexión y que los auriculares se conectan rápidamente a tu teléfono móvil. La calidad de sonido es un aspecto importante, y nos complace que estés satisfecho con el rendimiento de los auriculares en ese aspecto, incluso sin tener cancelación de ruido. Además, es bueno saber que percibes que los graves se escuchan genial.

Nos alegra saber que los auriculares se mantienen en su lugar durante tus actividades deportivas, incluso cuando sudas mucho. La resistencia y la durabilidad son características clave que buscamos en nuestros productos, y nos complace que los auriculares hayan cumplido tus expectativas hasta ahora.

Entendemos que cada persona tiene diferentes formas de orejas y preferencias de comodidad. Es posible que los auriculares no sean tan cómodos para un uso prolongado en tu caso específico debido a la forma de tu oreja y su peso. Sin embargo, agradecemos tus comentarios al respecto, ya que nos ayuda a mejorar y brindar una experiencia más satisfactoria a nuestros clientes en el futuro.

En cuanto a la duración de la batería, comprendemos que puede resultar decepcionante si se utilizan durante largos períodos de tiempo. Ten en cuenta que la duración de la batería puede variar según el volumen y el uso general de los auriculares. No obstante, nos alegra que aún así los recomiendes debido a su relación calidad-precio.

Agradecemos sinceramente tus comentarios y esperamos que disfrutes aún más de tus auriculares inalámbricos. Si tienes alguna otra pregunta o necesitas asistencia adicional, no dudes en contactarnos. ¡Estando encantados de ayudarte!

Observamos que la respuesta tiene un tono informal y es demasiado extensa. Prácticamente comenta cada uno de los aspectos que menciona en la reseña. En el siguiente *prompt* daremos instrucciones para que identifique en primer lugar el sentimiento de la reseña, y en base a ese sentimiento genere una respuesta más concisa que se centre en los aspectos relevantes del comentario del cliente. Asimismo, pediremos que conteste en un tono formal y se asegure de indicar que se trata de un asistente de IA.

Entrada 2

Eres un asistente de IA de servicio al cliente de una página web que vende artículos.

Tu tarea es responder a una reseña de un cliente que ha comprado un artículo.

Identifica el sentimiento de la reseña para generar la respuesta. Si el sentimiento es positivo o neutral, agradézcales por su revisión. Si el sentimiento es negativo, pida disculpas y sugiera que se comuniquen con el servicio de atención al cliente.

Asegúrate de usar detalles específicos de la revisión.

Escribe en un tono conciso y profesional.

Acaba la respuesta indicando que eres un "Asistente de IA de servicio al cliente".

Reseña del cliente:

" [reseña] "

Salida 2

Estimado cliente,

Gracias por tomarse el tiempo de compartir su experiencia con nosotros. Nos complace saber que nuestros auriculares inalámbricos han cumplido con sus expectativas en términos de conectividad y calidad de sonido. Valoramos sus comentarios positivos sobre el sonido maravilloso y la sensación de cancelación de ruido. También nos alegra saber que nuestros auriculares se han mantenido en su lugar durante sus actividades deportivas y que no ha experimentado problemas con las almohadillas.

Lamentamos escuchar que los auriculares pueden resultar un poco incómodos después de llevarlos puestos durante más de 3 horas, así como que puedan sentirse pesados. Entendemos que la comodidad es importante, y apreciamos su perspectiva sobre este tema. Además, tomamos nota de su comentario sobre la duración de la batería y lo tendremos en cuenta para futuras mejoras.

Si tiene alguna otra inquietud o pregunta, no dude en comunicarse con nuestro servicio de atención al cliente. Estaremos encantados de ayudarle y brindarle una solución adecuada.

Gracias una vez más por su revisión detallada. Apreciamos su apoyo y confianza en nuestros productos.

Atentamente,

Asistente de IA de servicio al cliente

4.1.3. Prompt Engineering para la generación de código en R

El papel que juegan los modelos del lenguaje en la generación de código ha cobrado importancia en los últimos años con la aparición de los modelos del lenguaje a gran escala. Estos modelos son capaces de generar código automáticamente en función de los requisitos de los desarrolladores. Puede reducir los esfuerzos de codificación repetitivos y mejorar la productividad del desarrollo de software (Lui et al., 2023 y Brown et al., 2020). Para ello, los desarrolladores pueden expresar sus requisitos como descripciones en lenguaje natural, lo que permite a los desarrolladores especificar sus necesidades de forma intuitiva y comunicárselo al modelo para que genere el código deseado.

A continuación se pone a prueba ChatGPT para la generación de código en el entorno de programación R para una tarea específica. En concreto, se pretende crear un mapa mundial que represente la puntuación que los países han obtenido en el Índice de concentración de mercado de Herfindahl e Hirschman (IHH). En economía, este índice es conocido por medir la competencia de un mercado, cuyo valor se mueve en un rango entre 0 y 1. Un valor alto expresa un mercado muy concentrado y cercano al monopolio, mientras que un valor bajo indica un mercado en situación próxima a la competencia perfecta (Banco Mundial, s.f.)

Para dicha tarea, se ha creado un archivo *Excel* con los datos del índice para un listado de países. El objetivo es conseguir que ChatGPT genere un código en R que importe los datos de este archivo y represente en un mapa mundial la puntuación obtenida por cada país en una escala de color degradado.

Entrada 1

Tengo un conjunto de datos sobre el índice de concentración de mercado de algunos países. Tu tarea es proporcionarme un código de R para representar en un mapa mundial dicha puntuación en una escala de color degradado.

Salida 1

¡Claro! Aquí tienes un ejemplo de código en R que puedes usar para representar el índice de concentración de mercado en un mapa mundial utilizando una escala de color

degradado:

```
``R

# Instalar y cargar paquetes necesarios

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

library(maps)

library(mapdata)

library(ggplot2)

# Crear un dataframe de ejemplo con los datos del índice de concentración de mercado

datos <- data.frame(

  Pais = c("Argentina", "Brasil", "Chile", "China", "Estados Unidos", "India", "Japón",
"Rusia"),

  Indice = c(0.35, 0.42, 0.28, 0.65, 0.5, 0.33, 0.6, 0.55)

)

# Cargar los datos de los mapas mundiales

mapa_mundial <- map_data("world")

# Unir los datos del mapa con los datos del índice de concentración

mapa_datos <- merge(mapa_mundial, datos, by.x = "region", by.y = "Pais", all.x =
TRUE)

# Crear el gráfico utilizando ggplot2
```

```

ggplot() +

  geom_polygon(data = mapa_datos, aes(x = long, y = lat, group = group, fill = Indice),

    color = "black", size = 0.2) +

  scale_fill_gradient(low = "green", high = "red", name = "Índice de concentración") +

  theme_void()

# Mostrar el gráfico

...

[explicación del código generado]

```

Como se puede observar, el código generado crea su propio conjunto de datos. No obstante, lo que queremos es que importe los datos de un archivo *Excel* ya creado previamente. El siguiente *prompt* consiste en una modificación del *prompt* inicial para proporcionar más datos al modelo y poder así poder obtener el código deseado. En concreto, se indica al modelo que debe importar los datos de un Excel, el formato del archivo, las columnas que contiene y su tarea de generar un mapa. En cuanto al mapa, se especifican los colores que queremos utilizar en función de la puntuación obtenida.

Entrada 2

Tu tarea es generar un código de R que pinte un mapa mundial a partir de unos datos importados de un archivo excel en formato xlsx llamado "HHIndex". A continuación te proporciono más información:

Este archivo contiene una columna con los datos de los países (llamada "País") y otra columna con la puntuación en el índice de concentración de mercado (llamada "Puntuación").

El mapa generado tiene que representar los países en una escala de color degradado en función de su puntuación. Las puntuaciones se mueven en un rango entre el 0 y el 1. Un valor de 0 corresponde con el color blanco. Un valor de 1 se corresponde con el color rojo. Si no hay dato para algún país, aparecerá representado en color gris.

En color gris también aparecerán las líneas que delimitan los países.

Salida 2

Claro, aquí tienes un código de R que importará los datos de un archivo Excel en formato xlsx llamado "HHIndex" y generará un mapa mundial con una escala de color degradado basada en la puntuación del índice de concentración de mercado:

```
``R

# Instalar y cargar paquetes necesarios

install.packages("readxl")

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

library(readxl)

library(maps)

library(mapdata)

library(ggplot2)

# Ruta al archivo Excel

ruta_archivo <- "ruta/al/archivo_excel.xlsx"

# Leer datos del archivo Excel

datos <- read_excel(ruta_archivo, sheet = "HHIndex")

# Cargar los datos de los mapas mundiales

mapa_mundial <- map_data("world")
```

```

# Unir los datos del mapa con los datos del índice de concentración

mapa_datos <- merge(mapa_mundial, datos, by.x = "region", by.y = "País", all.x =
TRUE)

# Crear el gráfico utilizando ggplot2

ggplot() +

  geom_polygon(data = mapa_datos, aes(x = long, y = lat, group = group, fill =
Puntuación),

              color = "gray60", size = 0.2) +

  scale_fill_gradient(low = "white", high = "red", name = "Puntuación del Índice de
Concentración",

                    na.value = "gray70", limits = c(0, 1)) +

  theme_void()

# Mostrar el gráfico

...

[explicación del código generado]

```

El código proporcionado por el modelo genera un error. El modelo está asumiendo que la columna “País” del conjunto “datos” contiene los mismos nombres que la columna “región” del conjunto “mapa_mundial”. No obstante, Estados Unidos aparece como “United States” en un conjunto y como “USA” en el otro conjunto. Estas diferencias hacen que a la hora de unir los dos conjuntos de datos, haya países a los que no se les asigne las coordenadas necesarias para que queden representados en el mapa (ver anexos para más información sobre la explicación del código).

Posteriormente, en vez de modificar el *prompt* inicial y que el modelo genere un

código de nuevo, comunicamos a *ChatGPT* el error que produce el código y pedimos que lo modifique.

Entrada 3

El código proporcionado no representa correctamente todos los países porque la nomenclatura de los países de la columna "País" del conjunto "datos" no corresponden siempre con la nomenclatura utilizada en la columna "region" del conjunto "mapa_mundial".

Para solucionar esto modifica el código para que, en primer lugar, añada una columna con los códigos ISO de los países tanto al conjunto "datos" como al conjunto "mapa_mundial", y después una estos dos conjuntos utilizando los códigos ISO.

Salida 3

Mis disculpas por la confusión anterior. Aquí tienes una versión modificada del código que agrega una columna con los códigos ISO de los países y luego une los conjuntos de datos utilizando esos códigos:

```
``R

# Instalar y cargar paquetes necesarios

install.packages("readxl")

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

library(readxl)

library(maps)

library(mapdata)

library(ggplot2)

# Ruta al archivo Excel

ruta_archivo <- "ruta/al/archivo_excel.xlsx"
```

```

# Leer datos del archivo Excel

datos <- read_excel(ruta_archivo, sheet = "HHIndex")

# Cargar los datos de los mapas mundiales

mapa_mundial <- map_data("world")

# Obtener los códigos ISO de los países en el conjunto "datos"

library(countrycode)

datos$ISO <- countrycode(sourcevar = datos$País, origin = "country.name", destination
= "iso2c")

# Obtener los códigos ISO de los países en el conjunto "mapa_mundial"

mapa_mundial$ISO <- countrycode(sourcevar = mapa_mundial$region, origin =
"country.name", destination = "iso2c")

# Unir los datos del mapa con los datos del índice de concentración utilizando los
códigos ISO

mapa_datos <- merge(mapa_mundial, datos, by = "ISO", all.x = TRUE)

# Crear el gráfico utilizando ggplot2

ggplot() +

  geom_polygon(data = mapa_datos, aes(x = long, y = lat, group = group, fill =
Puntuación),

    color = "gray60", size = 0.2) +

  scale_fill_gradient(low = "white", high = "red", name = "Puntuación del Índice de

```

Concentración",

```
na.value = "gray70", limits = c(0, 1)) +
```

```
theme_void()
```

```
# Mostrar el gráfico
```

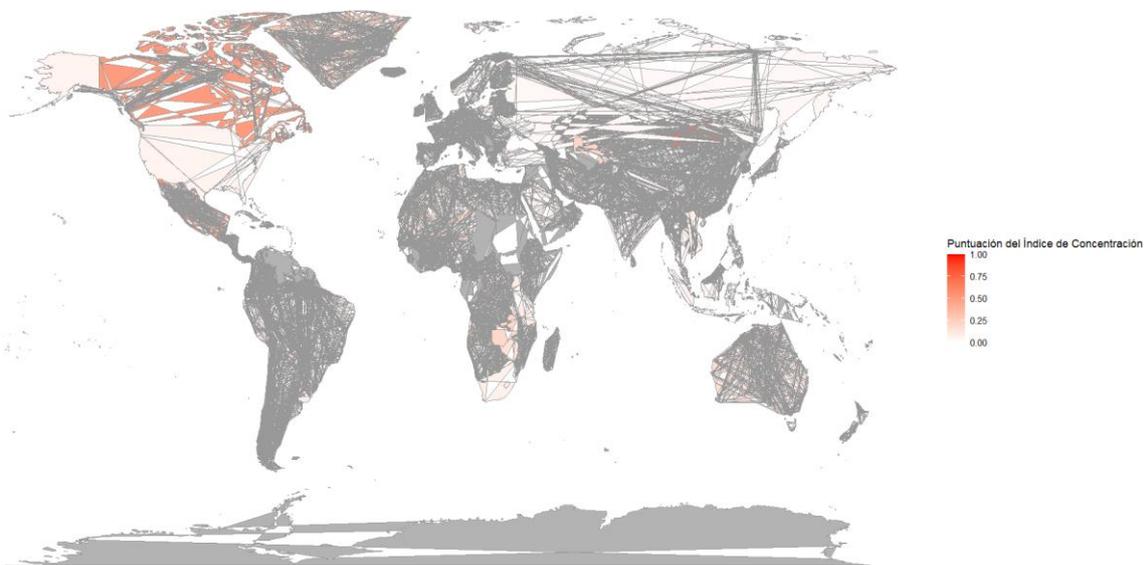
```
...
```

[explicación del código generado]

El modelo ha modificado el código anterior, y como bien hemos indicado en la instrucción, añade una columna a ambos conjuntos de datos para luego realizar una unión utilizando esta columna común.

Ponemos a prueba el código en el entorno de programación R para ver si el mapa generado es lo que buscamos. El resultado se muestra en la figura 14. A pesar de crear la escala de color correctamente en función de la puntuación, observamos que la representación del mapa queda distorsionada. Para intentar resolverlo, generamos otra instrucción para que el modelo modifique el código a uno que utilice la librería “ggthemes”. Esta librería es una extensión del paquete “ggplot2” que mejora la apariencia de los gráficos.

Figura 14. Mapa generado por ChatGPT con el prompt nº3.



Nota. Mapa generado con código en R a través del software Rstudio. Elaboración propia.

Entrada 4

¿Podrías modificar el código para que utilice la librería "ggthemes" para representar el mapa?

Salida 4

Por supuesto, aquí tienes el código modificado que utiliza la librería "ggthemes" para representar el mapa mundial:

```
``R

# Instalar y cargar paquetes necesarios

install.packages("readxl")

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

install.packages("ggthemes")

library(readxl)

library(maps)

library(mapdata)

library(ggplot2)

library(ggthemes)

# Ruta al archivo Excel

ruta_archivo <- "ruta/al/archivo_excel.xlsx"

# Leer datos del archivo Excel

datos <- read_excel(ruta_archivo, sheet = "HHIndex")
```

```

# Cargar los datos de los mapas mundiales

mapa_mundial <- map_data("world")

# Obtener los códigos ISO de los países en el conjunto "datos"

library(countrycode)

datos$ISO <- countrycode(sourcevar = datos$País, origin = "country.name", destination
= "iso2c")

# Obtener los códigos ISO de los países en el conjunto "mapa_mundial"

mapa_mundial$ISO <- countrycode(sourcevar = mapa_mundial$region, origin =
"country.name", destination = "iso2c")

# Unir los datos del mapa con los datos del índice de concentración utilizando los
códigos ISO

mapa_datos <- merge(mapa_mundial, datos, by = "ISO", all.x = TRUE)

# Crear el gráfico utilizando ggplot2 y ggthemes

ggplot() +

  geom_map(data = mapa_datos, map = mapa_datos, aes(x = long, y = lat, map_id =
region, fill = Puntuación),

  color = "gray60", size = 0.2) +

  scale_fill_gradient(low = "white", high = "red", name = "Puntuación del Índice de
Concentración",

  na.value = "gray70", limits = c(0, 1)) +

  theme_map() +

```

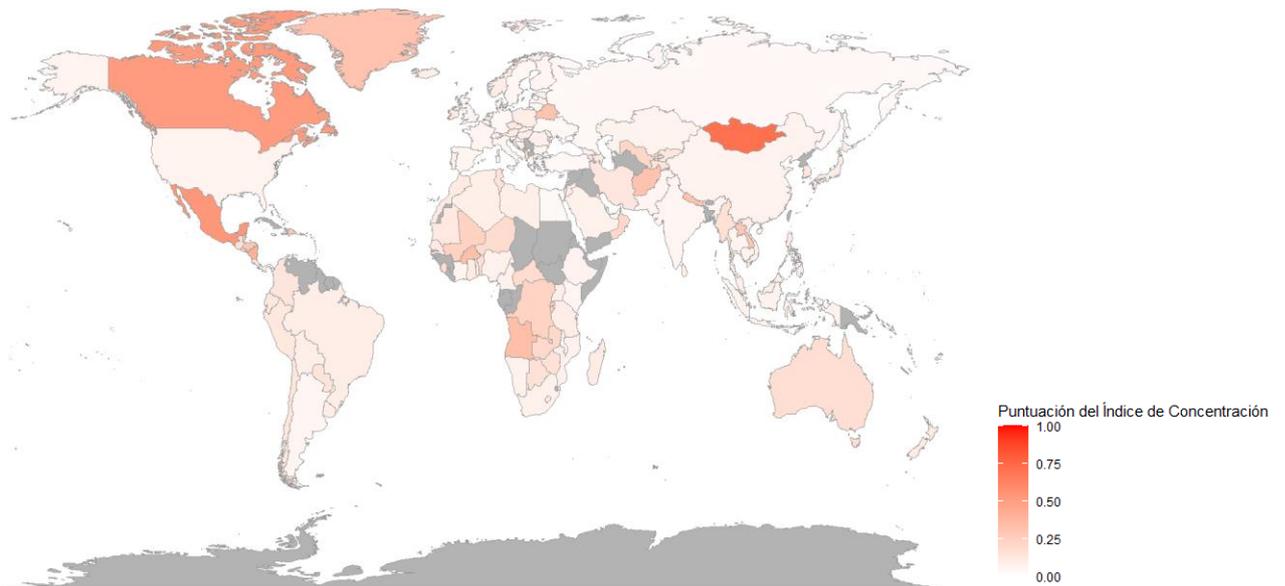
```
theme(legend.position = "right")
```

```
# Mostrar el gráfico
```

```
...
```

```
[explicación del código generado]
```

Figura 15. Mapa generado por ChatGPT con el prompt nº4.



Nota. Mapa generado con código en R a través del software Rstudio. Elaboración propia.

Como podemos observar, el último código creado por el modelo pinta correctamente un mapa mundial donde la concentración de mercado de los países según el índice IHH queda representado a través de una escala de colores entre el blanco y el rojo, siendo el color gris aquellos países donde no hay datos de puntuación en el índice.

5. Conclusiones

5.1. Conclusiones

El presente trabajo ha reunido los **principales conceptos en torno al Procesamiento del Lenguaje Natural** a través de una **revisión exhaustiva de la literatura**. Este repaso por la historia del NLP ha sido necesario para poder entender la última gran revolución en el campo de la IA, las **redes Transformer**.

Concretamente, se han revisado los conceptos de *tokenización*, *bag-of-word* y *one-hot encoding*, que sumado a las técnicas de reducción de texto como el *stemming* o la lematización, integran el conjunto de herramientas empleadas por las máquinas en el **sistema clásico de procesamiento** de texto. Más adelante, como origen de los modelos *Transformer*, se estudia la **evolución del aprendizaje profundo** hacia algoritmos capaces de prestar atención al contexto en el que se produce el lenguaje. En este apartado se explican conceptos importantes como *word embeddings*, redes recurrentes y convolucionales, y la arquitectura *Encoder-Decoder*. Por último en esta revisión, se recopilan los **principales diseños Transformer** creados hasta la fecha, poniendo foco en las características que los diferencian.

Finalmente, el trabajo **proporciona un mejor entendimiento del Prompt Engineering**, una disciplina que aún se encuentra en fase de investigación por la comunidad científica. El *Prompt Engineering* es importante en el campo de los LLM porque permite tener un mayor control sobre los resultados generados, facilita la personalización, mejora la usabilidad, previene sesgos y aumenta la eficiencia en el uso de estos modelos de lenguaje. El *Prompt Engineering* funciona como un puente de unión y de traducción entre el humano y la máquina. Permite al modelo comprender las necesidades y expectativas que tiene el usuario cuando interactúa con él, beneficiándose de todo el potencial que ofrecen estos algoritmos.

No obstante, es una técnica que requiere experimentación y ajuste fino, ya que la forma en que se formula el *prompt* puede tener un impacto significativo en los resultados del modelo. Es importante probar y evaluar diferentes enfoques para encontrar el mejor *prompt* para cada tarea o problema específico.

A través del modelo pre-entrenado **ChatGPT**, se ilustra el proceso de optimización de *prompts* para las tareas comunes de síntesis, inferencia de sentimiento y ampliación de texto, así como la generación de código en lenguaje de programación **R** para una tarea específica. Esto nos permite comparar las respuestas generadas por ChatGPT y evaluar si la tarea para la cual ha sido instruido se lleva a cabo correctamente.

Los **resultados obtenidos** tomando como base una reseña de unos auriculares comprados por un cliente en una tienda online, son los siguientes:

- El uso de un *prompt* básico para la tarea de síntesis generaba un resumen con un tono narrativo. No obstante, si en la instrucción se indicaba que el modelo **tomara un rol específico** (el resumen iba a ser proporcionado al diseñador de los auriculares), el resumen se redactaba acorde a ese contexto concreto.
- La tarea de extracción del sentimiento del texto se ejecutaba correctamente con el uso de un *prompt* básico, pero la información proporcionada era muy extensa, casi parafraseando el texto original. Con un *prompt* más específico se conseguía que el modelo generara una respuesta **centrándose en las características principales** comentadas en la reseña.
- Cuando se usa un *prompt* básico para que el modelo que genere una respuesta al comentario como si fuese un asistente del servicio de atención al cliente, este lo hace con un tono informal, y de nuevo, comentando cada aspecto proporcionado en la reseña. Con un *prompt* más detallado, obtenemos una **respuesta formal, basada en el sentimiento inferido y comentando solo los aspectos relevantes** de la reseña.

Por otro lado, **la tarea de generación de código en R**, para la representación en un mapa mundial de la concentración de mercado de los países según el Índice HH, ha **requerido 3 modificaciones adicionales del *prompt* básico** para conseguir que el modelo genere el código deseado. En ocasiones ha sido necesario modificar el *prompt* original a uno más específico, en otras ocasiones simplemente se ha interactuado con el modelo para que directamente modificara el código ya proporcionado y corregir el error detectado. No obstante, el número de interacciones con el modelo variarán en función de la complejidad del resultado que se quiera obtener, así como del nivel de detalle requerido por el modelo para la generación del código.

Los resultados obtenidos muestran que **en general, no existen *prompts* o instrucciones que aseguren la mejor respuesta.** En su lugar, los mejores *prompts* se generan como resultado de un **proceso iterativo y continuo refinamiento**, donde se desarrollan distintas instrucciones hasta encontrar la entrada que proporcione la respuesta deseada para una tarea específica. No obstante, existen **algunas reglas básicas** que se recomiendan seguir para guiar al modelo hacia el resultado deseado y reducir la posibilidad de obtener respuestas irrelevantes o incorrectas.

1. Escribir instrucciones claras y específicas.

- Un mensaje claro no es lo mismo que un mensaje corto, porque en muchos casos, los mensajes más largos brindan más claridad y contexto para el modelo.
- Especificar una estructura o formato concreto de salida, por ejemplo, un listado.
- Pedir al modelo que ejecute una acción en caso de que se cumplan unas condiciones.
- Usar delimitadores y signos de puntuación para indicar claramente distintas partes de la entrada. Esto es muy útil para evitar instrucciones contradictorias.
- Mostrar ejemplos de resultados exitosos antes de pedir que ejecute una tarea. Esto es lo que se conoce como “Few-shot prompting”.

2. Darle al modelo tiempo para pensar.

- Especificar los pasos necesarios para completar una tarea.
- Instruir al modelo para que razone su propia solución antes de apresurarse a llegar a una conclusión.

5.2. Limitaciones y futuras líneas de investigación

A lo largo del trabajo se ha identificado como principales limitaciones las restricciones de tiempo y recursos. Con una mayor disponibilidad de tiempo y acceso público a recursos tanto bibliográficos como de modelos del lenguaje, se podría haber llevado a cabo una revisión más profunda y contrastada, así como se podría haber implementado más casos prácticos utilizando distintos modelos.

A raíz de estas limitaciones, se sugiere como futuras líneas de investigación no solo mantener una revisión actualizada de los avances que se vayan consiguiendo en el campo

del NLP, si no continuar la investigación del *Prompt Engineering* y poner en práctica esta disciplina para más tareas de las estudiadas en este trabajo. En concreto, la automatización de generación de código es un área que aún tiene mucho margen de investigación y la aplicación del *Prompt Engineering* puede ser muy beneficiosa en términos de eficiencia.

Asimismo, otras áreas de investigación pueden ir asociadas a estudiar el impacto que tienen estos modelos, y en general la Inteligencia Artificial, en la sociedad. En concreto, existen diversos debates en torno a los costes ambientales y económicos (por la cantidad de energía que consumen los algoritmos), en torno a la ética y escasa regulación, así como sería interesante realizar un estudio de la opinión del gran público ante la llegada inminente de la IA.

6. Bibliografía

- Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., ... & Kiru, M. U. (2019). Comprehensive review of artificial neural network applications to pattern recognition. *IEEE Access*, 7, 158820-158846.
<https://ieeexplore.ieee.org/abstract/document/8859190>
- Antonellis, I., & Gallopoulos, E. (2006). Exploring term-document matrices from matrix models in text mining. *En arXiv [cs.IR]*. <http://arxiv.org/abs/cs/0602076>
- Banco Mundial. (s.f). *Índice de HH sobre concentración económica de un mercado*. World Integrated Trade Solution (WITS).
<https://wits.worldbank.org/CountryProfile/es/Country/CHN/StartYear/2016/EndYear/2020/Indicator/HH-MKT-CNCNTRTN-NDX>
- Bengio, Y., Ducharme, R., & Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.
https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. *En arXiv [cs.CL]*.
<http://arxiv.org/abs/2005.14165>
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *En arXiv [cs.CL]*. <http://arxiv.org/abs/1406.1078>
- Clark, K., Luong, M.-T., Le, Q. V., & Manning, C. D. (2020). ELECTRA: Pre-training text encoders as discriminators rather than generators. *En arXiv [cs.CL]*.
<http://arxiv.org/abs/2003.10555>
- Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th international conference on Machine learning* (pp. 160-167).
<https://dl.acm.org/doi/abs/10.1145/1390156.1390177>
- Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. *En arXiv [cs.LG]*.
<http://arxiv.org/abs/1901.02860>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional Transformers for language understanding. *En arXiv [cs.CL]*.
<http://arxiv.org/abs/1810.04805>

- Dos Santos, C., & Gatti, M. (2014, August). Deep convolutional neural networks for sentiment analysis of short texts. *Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers*, [Conferencia], (pp. 69-78). <https://aclanthology.org/C14-1008.pdf>
- Eberendu, A. C. (2016). Unstructured Data: an overview of the data of Big Data. *International Journal of Computer Trends and Technology*, 38(1), 46-50.
- Gardner, M. W., & Dorling, S. R. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14–15), pp. 2627–2636. [https://doi.org/10.1016/s1352-2310\(97\)00447-0](https://doi.org/10.1016/s1352-2310(97)00447-0)
- Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57, 345-420. <https://www.jair.org/index.php/jair/article/view/11030>
- He, P., Liu, X., Gao, J., & Chen, W. (2020). DeBERTa: Decoding-enhanced BERT with disentangled attention. En *arXiv [cs.CL]*. <http://arxiv.org/abs/2006.03654>
- Hinton, G. E. (1984). Distributed representations.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), pp.1735-1780. <https://ieeexplore.ieee.org/abstract/document/6795963>
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8), pp. 2554-2558. <https://doi.org/10.1073/pnas.79.8.2554>
- https://www.researchgate.net/profile/Adanma-Eberendu/publication/309393428_Unstructured_Data_an_overview_of_the_data_of_Big_Data/links/5bc89b5c458515f7d9c65beb/Unstructured-Data-an-overview-of-the-data-of-Big-Data.pdf
- Johnson, R., & Zhang, T. (2015). Semi-supervised convolutional neural networks for text categorization via region embedding. En *arXiv [stat.ML]*. <http://arxiv.org/abs/1504.01255>
- Kalchbrenner, N., & Blunsom, P. (2013, octubre). Recurrent continuous translation models. *Proceedings of the 2013 conference on empirical methods in natural language processing*, [Conferencia], pp. 1700-1709. <https://aclanthology.org/D13-1176.pdf>
- Khurana, D., Koli, A., Khatter, K., & Singh, S. (2017). Natural language processing: State of the art, current trends and challenges. En *arXiv [cs.CL]*. <http://arxiv.org/abs/1708.05148>

- Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. En *arXiv [cs.CL]*. <http://arxiv.org/abs/1408.5882>
- Kröse, B., & Van der Smagt, P. (1996). An introduction to Neural Networks. <https://www.infor.uva.es/~teodoro/neuro-intro.pdf>
- Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. En *arXiv [cs.CL]*. <http://arxiv.org/abs/1909.11942>
- Landolt, S., Söllner, M., & Wambsganb, T. (2021). A Taxonomy for Deep Learning in Natural Language Processing. *Hawaii International Conference on System Sciences (HICSS)* [Conferencia]. <https://scholarspace.manoa.hawaii.edu/handle/10125/70741>
- Lauriola, I., Lavelli, A., & Aiolli, F. (2022). An introduction to Deep Learning in Natural Language Processing: Models, techniques, and tools. *Neurocomputing*, 470, pp. 443–456. <https://doi.org/10.1016/j.neucom.2021.05.103>
- Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE. Institute of Electrical and Electronics Engineers*, 86(11), pp. 2278–2324. <https://doi.org/10.1109/5.726791>
- Lennon, B. (2014). Machine translation: A tale of two cultures. *A companion to translation studies*, pp.133-146. <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118613504.ch10>
- Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. En *arXiv [cs.LG]*. <http://arxiv.org/abs/1506.00019>
- Liu, C., Bao, X., Zhang, H., Zhang, N., Hu, H., Zhang, X., & Yan, M. (2023). Improving ChatGPT Prompt for Code Generation. En *arXiv [cs.SE]*. <http://arxiv.org/abs/2305.08360>
- Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2021). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. En *arXiv [cs.CL]*. <http://arxiv.org/abs/2107.13586>
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. En *arXiv [cs.CL]*. <http://arxiv.org/abs/1907.11692>
- Manaris, B. (1998). Natural language processing: A human-computer interaction perspective. *Advances in Computers*, 47, pp. 1-66. <https://www.sciencedirect.com/science/article/abs/pii/S0065245808606658>
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. En *arXiv [cs.CL]*. <http://arxiv.org/abs/1301.3781>
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013b). Distributed

Representations of Words and Phrases and their Compositionality. *En arXiv [cs.CL]*.
https://papers.nips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

- OpenAI platform. (s/f). Models Overview. <https://platform.openai.com/docs/models>
- OpenAI. (2023a). *ChatGPT* [Large language model]. Prompt: ¿por qué se caracteriza el lenguaje humano y por qué es complicado de entender por las máquinas?
<https://chat.openai.com/chat>
- OpenAI. (2023b). *ChatGPT* [Large language model]. Prompt: ¿qué es la tokenización en Procesamiento del Lenguaje Natural? <https://chat.openai.com/chat>
- OpenAI. (2023c). *ChatGPT* [Large language model]. Prompt: ¿ qué es Term Frequency Representation en Bag of Words? <https://chat.openai.com/chat>
- OpenAI. (2023d). *ChatGPT* [Large language model]. Prompt: ¿qué técnicas de reducción de texto hay (stemming...)? Y ¿por qué es importante en el contexto del Procesamiento del Lenguaje Natural? <https://chat.openai.com/chat>
- OpenAI. (2023e). *ChatGPT* [Large language model]. Prompt: ¿qué características tienen las redes neuronales profundas en el campo del NLP? <https://chat.openai.com/chat>
- OpenAI. (2023f). *ChatGPT* [Large language model]. Prompt: ¿ qué dice el paper Distributed Representations of Words and Phrases and Their Compositionality de “softmax” y “negative sampling”? <https://chat.openai.com/chat>
- OpenAI. (2023g). *ChatGPT* [Large language model]. Prompt: ¿a quí se refiere el paper “Attention is All you Need” con “atención multicabeza”? <https://chat.openai.com/chat>
- OpenAI. (2023h). GPT-4 Technical Report. *En arXiv [cs.CL]*. <http://arxiv.org/abs/2303.08774>
- OpenAI. (2023i). *ChatGPT* [Large language model]. Prompt: ¿Por qué es importante el "Prompt Engineering" en el campo de los LLM? <https://chat.openai.com/chat>
- Patwardhan, N., Marrone, S., & Sansone, C. (2023). Transformers in the Real World: A Survey on NLP Applications. *Information*, 14(4), 242. <https://www.mdpi.com/2078-2489/14/4/242>
- Polak, M. P., & Morgan, D. (2023). Extracting Accurate Materials Data from Research Papers with Conversational Language Models and Prompt Engineering--Example of ChatGPT. *En arXiv [cs.CL]*. <https://arxiv.org/pdf/2303.05352.pdf>
- Qader, W. A., Ameen, M. M., & Ahmed, B. I. (2019). An overview of bag of words;Importance, implementation, applications, and challenges. *2019 International Engineering Conference (IEC)*, [Conferencia], pp. 200–204.
- Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training*.
<https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf>

- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8), 9. <https://life-extension.github.io/2020/05/27/GPT%E6%8A%80%E6%9C%AF%E5%88%9D%E6%8E%A2/language-models.pdf>
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. En *arXiv [cs.LG]*. <http://arxiv.org/abs/1910.10683>
- Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for Activation Functions. En *arXiv [cs.NE]*. <http://arxiv.org/abs/1710.05941>
- Rothman, D., & Gulli, A. (2022). *Transformers for Natural Language Processing: Build, train, and fine-tune deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, and GPT-3* (2a ed.). Packt Publishing. <https://books.google.at/books?id=u9FjEAAAQBAJ>
- Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11), pp. 2673-2681. <https://ieeexplore.ieee.org/abstract/document/650093>
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27. https://proceedings.neurips.cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html
- Torfi, A., Shirvani, R. A., Keneshloo, Y., Tavaf, N., & Fox, E. A. (2020). Natural Language Processing advancements by deep learning: A survey. En *arXiv [cs.CL]*. <http://arxiv.org/abs/2003.01200>
- Tunstall, L., Von Werra, L., & Wolf, T. (2022). *Natural language processing with transformers*. O'Reilly Media. <https://books.google.at/books?id=nzxbEAAAQBAJ>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. En *arXiv [cs.CL]*. <http://arxiv.org/abs/1706.03762>
- Von der Mosel, J., Trautsch, A., & Herbold, S. (2022). On the validity of pre-trained transformers for natural language processing in the software engineering domain. *IEEE transactions on software engineering*, 49(4), 1–1. <https://doi.org/10.1109/tse.2022.3178469>
- Wallach, H. M. (2006, June). Topic modeling: beyond bag-of-words. *Proceedings of the 23rd international conference on Machine learning*, [Conferencia], pp. 977-984. <https://dl.acm.org/doi/abs/10.1145/1143844.1143967>
- White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. En *arXiv [cs.SE]*. <http://arxiv.org/abs/2302.11382>

- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. En *arXiv [cs.CL]*. <http://arxiv.org/abs/1906.08237>
- Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE computational intelligence magazine*, 13(3), pp. 55–75. <https://doi.org/10.1109/mci.2018.2840738>
- Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. En *arXiv [cs.LG]*. <http://arxiv.org/abs/1509.01626>
- Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., ... Wen, J.-R. (2023). A survey of large language models. En *arXiv [cs.CL]*. <http://arxiv.org/abs/2303.18223>
- Zhou, M., Duan, N., Liu, S., & Shum, H.-Y. (2020). Progress in neural NLP: Modeling, learning, and reasoning. *Engineering*, 6(3), pp. 275–290. <https://doi.org/10.1016/j.eng.2019.12.014>
- Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2022). Large language models are human-level prompt engineers. En *arXiv [cs.LG]*. <http://arxiv.org/abs/2211.01910>

7. Anexos

Entrada 1

Tengo un conjunto de datos sobre el índice de concentración de mercado de algunos países. Tu tarea es proporcionarme un código de R para representar en un mapa mundial dicha puntuación en una escala de color degradado.

Salida 1

¡Claro! Aquí tienes un ejemplo de código en R que puedes usar para representar el índice de concentración de mercado en un mapa mundial utilizando una escala de color degradado:

```
``R

# Instalar y cargar paquetes necesarios

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

library(maps)

library(mapdata)

library(ggplot2)

# Crear un dataframe de ejemplo con los datos del índice de concentración de mercado

datos <- data.frame(

  Pais = c("Argentina", "Brasil", "Chile", "China", "Estados Unidos", "India", "Japón",
"Rusia"),

  Indice = c(0.35, 0.42, 0.28, 0.65, 0.5, 0.33, 0.6, 0.55)

)

# Cargar los datos de los mapas mundiales
```

```

mapa_mundial <- map_data("world")

# Unir los datos del mapa con los datos del índice de concentración

mapa_datos <- merge(mapa_mundial, datos, by.x = "region", by.y = "Pais", all.x =
TRUE)

# Crear el gráfico utilizando ggplot2

ggplot() +

  geom_polygon(data = mapa_datos, aes(x = long, y = lat, group = group, fill = Indice),

    color = "black", size = 0.2) +

  scale_fill_gradient(low = "green", high = "red", name = "Índice de concentración") +

  theme_void()

# Mostrar el gráfico

'''

```

Este código utiliza los paquetes `maps`, `mapdata` y `ggplot2`. Asegúrate de tenerlos instalados ejecutando los comandos `install.packages()` antes de cargarlos con `library()`.

El dataframe `datos` contiene los nombres de los países y sus respectivos índices de concentración de mercado. Puedes reemplazar estos datos de ejemplo con tus propios datos.

El código carga los datos de los mapas mundiales utilizando la función `map_data()`. Luego, une estos datos con los datos del índice de concentración utilizando la función `merge()`. El resultado es un dataframe que contiene las coordenadas de los países y el índice de concentración correspondiente.

Finalmente, se utiliza `ggplot()` para crear el gráfico y `geom_polygon()` para dibujar los polígonos de los países en el mapa. La estética `fill = Indice` indica que se debe utilizar el índice de concentración como variable para el relleno de los polígonos. La función `scale_fill_gradient()` establece los colores degradados que van desde el verde (bajo índice) hasta el rojo (alto índice). El `name` dentro de `scale_fill_gradient()` establece la etiqueta de la leyenda.

Puedes ejecutar el código en R y verás el mapa mundial con los colores que representan el índice de concentración de mercado de cada país según los datos proporcionados. Asegúrate de tener una conexión a Internet activa para cargar los datos del mapa.

Entrada 2

Tu tarea es generar un código de R que pinte un mapa mundial a partir de unos datos importados de un archivo excel en formato xlsx llamado "HHIndex". A continuación te proporciono más información:

Este archivo contiene una columna con los datos de los países (llamada "País") y otra columna con la puntuación en el índice de concentración de mercado (llamada "Puntuación").

El mapa generado tiene que representar los países en una escala de color degradado en función de su puntuación. Las puntuaciones se mueven en un rango entre el 0 y el 1. Un valor de 0 corresponde con el color blanco. Un valor de 1 se corresponde con el color rojo. Si no hay dato para algún país, aparecerá representado en color gris.

En color gris también aparecerán las líneas que delimitan los países.

Salida 2

Claro, aquí tienes un código de R que importará los datos de un archivo Excel en formato xlsx llamado "HHIndex" y generará un mapa mundial con una escala de color degradado basada en la puntuación del índice de concentración de mercado:

```

``R

# Instalar y cargar paquetes necesarios

install.packages("readxl")

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

library(readxl)

library(maps)

library(mapdata)

library(ggplot2)

# Ruta al archivo Excel

ruta_archivo <- "ruta/al/archivo_excel.xlsx"

# Leer datos del archivo Excel

datos <- read_excel(ruta_archivo, sheet = "HHIndex")

# Cargar los datos de los mapas mundiales

mapa_mundial <- map_data("world")

# Unir los datos del mapa con los datos del índice de concentración

mapa_datos <- merge(mapa_mundial, datos, by.x = "region", by.y = "País", all.x = TRUE)

# Crear el gráfico utilizando ggplot2

```

```

ggplot() +
  geom_polygon(data = mapa_datos, aes(x = long, y = lat, group = group, fill = Puntuación),
              color = "gray60", size = 0.2) +
  scale_fill_gradient(low = "white", high = "red", name = "Puntuación del Índice de
Concentración",
                    na.value = "gray70", limits = c(0, 1)) +
  theme_void()

# Mostrar el gráfico

'''

```

Asegúrate de reemplazar `"ruta/al/archivo_excel.xlsx"` con la ruta real al archivo Excel que contiene tus datos. El archivo Excel debe tener una hoja llamada "HHIndex" que contiene dos columnas: una columna llamada "País" que contiene los nombres de los países, y otra columna llamada "Puntuación" que contiene la puntuación en el índice de concentración de mercado.

El código utiliza los paquetes ``readxl``, ``maps``, ``mapdata`` y ``ggplot2``. Asegúrate de tenerlos instalados ejecutando los comandos ``install.packages()`` antes de cargarlos con ``library()``.

El código lee los datos del archivo Excel utilizando la función ``read_excel()`` del paquete ``readxl``. Se especifica ``sheet = "HHIndex"` para leer los datos de la hoja llamada "HHIndex".

Luego, carga los datos de los mapas mundiales utilizando la función ``map_data()``. A continuación, se realiza una unión de los datos del mapa con los datos del índice de concentración utilizando la función ``merge()``. El resultado es un dataframe que contiene las coordenadas de los países y la puntuación correspondiente.

Finalmente, se utiliza `ggplot()` para crear el gráfico y `geom_polygon()` para dibujar los polígonos de los países en el mapa. La estética `fill = Puntuación` indica que se debe utilizar la puntuación del índice de concentración como variable para el relleno de los polígonos. La función `scale_fill_gradient()` establece los colores degradados que van desde el blanco (puntuación 0) hasta el rojo (puntuación 1). Se establece `na.value = "gray70"` para asignar el color gris a los países sin datos. El `name` dentro de `scale_fill_gradient()` establece la etiqueta de

la leyenda. Se utiliza `limits = c(0, 1)` para asegurarse de que la escala de color esté en el rango adecuado.

Ejecuta el código en R con los datos de tu archivo Excel y se generará un mapa mundial con los colores que representan la puntuación del índice de concentración de mercado de cada país. Asegúrate de tener una conexión a Internet activa para cargar los datos del mapa.

Entrada 3

El código proporcionado no representa correctamente todos los países porque la nomenclatura de los países de la columna "País" del conjunto "datos" no corresponden siempre con la nomenclatura utilizada en la columna "region" del conjunto "mapa_mundial".

Para solucionar esto modifica el código para que, en primer lugar, añada una columna con los códigos ISO de los países tanto al conjunto "datos" como al conjunto "mapa_mundial", y después una estos dos conjuntos utilizando los códigos ISO.

Salida 3

Mis disculpas por la confusión anterior. Aquí tienes una versión modificada del código que agrega una columna con los códigos ISO de los países y luego une los conjuntos de datos utilizando esos códigos:

```
```R
```

```
Instalar y cargar paquetes necesarios
```

```

install.packages("readxl")

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

library(readxl)

library(maps)

library(mapdata)

library(ggplot2)

Ruta al archivo Excel

ruta_archivo <- "ruta/al/archivo_excel.xlsx"

Leer datos del archivo Excel

datos <- read_excel(ruta_archivo, sheet = "HHIndex")

Cargar los datos de los mapas mundiales

mapa_mundial <- map_data("world")

Obtener los códigos ISO de los países en el conjunto "datos"

library(countrycode)

datos$ISO <- countrycode(sourcevar = datos$País, origin = "country.name", destination =
"iso2c")

Obtener los códigos ISO de los países en el conjunto "mapa_mundial"

mapa_mundial$ISO <- countrycode(sourcevar = mapa_mundial$region, origin =

```

```
"country.name", destination = "iso2c")
```

```
Unir los datos del mapa con los datos del índice de concentración utilizando los códigos ISO
```

```
mapa_datos <- merge(mapa_mundial, datos, by = "ISO", all.x = TRUE)
```

```
Crear el gráfico utilizando ggplot2
```

```
ggplot() +
```

```
 geom_polygon(data = mapa_datos, aes(x = long, y = lat, group = group, fill = Puntuación),
```

```
 color = "gray60", size = 0.2) +
```

```
 scale_fill_gradient(low = "white", high = "red", name = "Puntuación del Índice de
Concentración",
```

```
 na.value = "gray70", limits = c(0, 1)) +
```

```
 theme_void()
```

```
Mostrar el gráfico
```

```
...
```

Asegúrate de reemplazar `"ruta/al/archivo_excel.xlsx"` con la ruta real al archivo Excel que contiene tus datos. El archivo Excel debe tener una hoja llamada "HHIndex" que contiene dos columnas: una columna llamada "País" que contiene los nombres de los países y otra columna llamada "Puntuación" que contiene la puntuación en el índice de concentración de mercado.

El código utiliza el paquete `countrycode` para obtener los códigos ISO de los países en ambos conjuntos de datos. Se utiliza la función `countrycode()` para realizar la conversión de nombres de países a códigos ISO. Los códigos ISO se agregan como una nueva columna llamada "ISO" tanto en el conjunto "datos" como en el conjunto "mapa\_mundial".

Luego, se utiliza la función `merge()` para unir los conjuntos de datos utilizando la columna "ISO" como clave de unión. El resultado es un dataframe que contiene las coordenadas de los países, la puntuación del índice de concentración y otros atributos.

Finalmente, se utiliza `ggplot()` para crear el gráfico y `geom_polygon()` para dibujar los polígonos de los países en el mapa. La estética `fill = Puntuación` indica que se debe utilizar la puntuación del índice de concentración como variable para el relleno de los polígonos. La función `scale_fill_gradient()` establece los colores degradados que van desde

el blanco (puntuación 0) hasta el rojo (puntuación 1). Se establece `na.value = "gray70"` para asignar el color gris a los países sin datos. El `name` dentro de `scale_fill_gradient()` establece la etiqueta de la leyenda.

Ejecuta el código en R con los datos de tu archivo Excel y se generará un mapa mundial con los colores que representan la puntuación del índice de concentración de mercado de cada país. Asegúrate de tener una conexión a Internet activa para cargar los datos del mapa.



#### **Entrada 4**

¿Podrías modificar el código para que utilice la librería "ggthemes" para representar el mapa?

#### **Salida 4**

Por supuesto, aquí tienes el código modificado que utiliza la librería "ggthemes" para representar el mapa mundial:

```
``R

Instalar y cargar paquetes necesarios

install.packages("readxl")

install.packages("maps")

install.packages("mapdata")

install.packages("ggplot2")

install.packages("ggthemes")

library(readxl)

library(maps)

library(mapdata)

library(ggplot2)

library(ggthemes)

Ruta al archivo Excel

ruta_archivo <- "ruta/al/archivo_excel.xlsx"

Leer datos del archivo Excel

datos <- read_excel(ruta_archivo, sheet = "HHIndex")
```

```

Cargar los datos de los mapas mundiales

mapa_mundial <- map_data("world")

Obtener los códigos ISO de los países en el conjunto "datos"

library(countrycode)

datos$ISO <- countrycode(sourcevar = datos$País, origin = "country.name", destination =
"iso2c")

Obtener los códigos ISO de los países en el conjunto "mapa_mundial"

mapa_mundial$ISO <- countrycode(sourcevar = mapa_mundial$region, origin =
"country.name", destination = "iso2c")

Unir los datos del mapa con los datos del índice de concentración utilizando los códigos ISO

mapa_datos <- merge(mapa_mundial, datos, by = "ISO", all.x = TRUE)

Crear el gráfico utilizando ggplot2 y ggthemes

ggplot() +

 geom_map(data = mapa_datos, map = mapa_datos, aes(x = long, y = lat, map_id = region,
fill = Puntuación),

 color = "gray60", size = 0.2) +

 scale_fill_gradient(low = "white", high = "red", name = "Puntuación del Índice de
Concentración",

 na.value = "gray70", limits = c(0, 1)) +

 theme_map() +

 theme(legend.position = "right")

```

*# Mostrar el gráfico*

...

*Asegúrate de reemplazar `"ruta/al/archivo_excel.xlsx"` con la ruta real al archivo Excel que contiene tus datos. El archivo Excel debe tener una hoja llamada "HHIndex" que contiene dos columnas: una columna llamada "País" que contiene los nombres de los países y otra columna llamada "Puntuación" que contiene la puntuación en el índice de concentración de mercado.*

*El código utiliza la función `geom_map()` en lugar de `geom_polygon()` para dibujar el mapa. Se establece `map_id = region` en la estética para asociar los polígonos con los códigos ISO de los países.*

*Se utiliza `theme_map()` de la librería "ggthemes" para establecer el estilo del mapa. También se agrega `theme(legend.position = "right")` para ajustar la posición de la leyenda a la derecha del gráfico.*

*Ejecuta el código en R con los datos de tu archivo Excel y se generará un mapa mundial utilizando la librería "ggthemes" para estilizar el gráfico. Asegúrate de tener una conexión a Internet activa para cargar los datos del mapa.*

