Máster Universitario en Matemáticas

Universidad de Sevilla

Modelos Grandes de Lenguaje y aplicaciones a la generación automática de texto

José Carlos Jiménez Revuelta

Tutor: Fernando Sancho Caparrini

Julio 2023

Abstract

This Master's Thesis delves into the evolution and application of large-scale language models, with a particular focus on the fourth iteration of the Generative Pre-trained Transformer (GPT-4) developed by OpenAI. Through a profound theoretical and mathematical analysis, it examines how advances in Natural Language Processing (NLP), Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Transformers, and self-attention have facilitated the development of more sophisticated and accurate language models.

Key mathematical concepts such as vector spaces, embeddings, activation functions, and loss and optimization functions are explored in detail to provide a more robust understanding of the computational complexity, convergence, optimization, and the generalization and learning capacity of these models. This study also discusses performance metrics, the interpretability and explainability of language models, as well as the inherent limitations of current mathematical approaches.

This thesis also addresses the critical ethical and societal issues that arise with the use of large-scale language models, including bias and fairness, privacy and security and AI regulation and governance. By providing critical insight and comprehensive analysis of these topics, this work seeks to pave the way for future research in this rapidly evolving and highly relevant field.

Keywords: Natural Language Processing, Language Models, Deep Learning, GPT-4, Recurrent Neural Networks, Convolutional Neural Networks, Transformers, Self-attention, Computational Complexity, Optimization, Generalization, Interpretability, AI Ethics, AI Regulation.

Abstract

Este Trabajo de Fin de Máster (TFM) explora la evolución y la aplicación de los modelos de lenguaje a gran escala, con un enfoque particular en la cuarta iteración del modelo Generative Pre-trained Transformer (GPT-4) desarrollado por OpenAI. A través de un análisis teórico y matemático profundo, se examina cómo los avances en el Procesamiento del Lenguaje Natural (NLP), las Redes Neuronales Recurrentes (RNN), las Redes Neuronales Convolucionales (CNN), los Transformers, y la auto-atención han facilitado el desarrollo de modelos de lenguaje más sofisticados y precisos.

Los conceptos matemáticos clave, como los espacios vectoriales, los embeddings, las funciones de activación, y las funciones de pérdida y optimización son explorados en detalle para proporcionar una comprensión más sólida de la complejidad computacional, la convergencia, la optimización, y la capacidad de generalización y aprendizaje de estos modelos. Este estudio también discute las medidas de rendimiento, la interpretabilidad y la explicabilidad de los modelos de lenguaje, así como las limitaciones inherentes de los enfoques matemáticos actuales.

Este TFM también aborda las importantes cuestiones éticas y sociales que surgen con el uso de los modelos de lenguaje a gran escala, incluyendo el sesgo y la equidad, la privacidad y seguridad y la regulación de la Inteligencia Artificial (IA). Al proporcionar una visión crítica y un análisis exhaustivo de estos temas, este trabajo busca abrir nuevos caminos para la investigación futura en esta área en constante evolución y altamente relevante.

Palabras clave: Procesamiento del Lenguaje Natural, Modelos de Lenguaje, Aprendizaje Profundo, GPT-4, Redes Neuronales Recurrentes, Redes Neuronales Convolucionales, Transformers, Auto-atención, Complejidad Computacional, Optimización, Generalización, Interpretabilidad, Ética de la IA, Regulación de la IA.

Índice general

Capítulo 1

Introducción

1.1. Motivación

La idea de dotar a las máquinas de la capacidad de leer y comprender los idiomas que hablamos los seres humanos parecía mucho más lejana hace apenas un par de años de lo que es en los convulsos tiempos que nos ocupan. Esto se debe a que, aunque los humanos somos excelentes interpretadores de significados a partir de texto de una forma casi inmediata, aún no somos capaces de describir las reglas que gobiernan su construcción y sus significados. El procesamiento del lenguaje natural (NLP, por sus siglas en inglés) nació como una rama de la inteligencia artificial centrada en la interacción entre las computadoras y el lenguaje humano. En los últimos años, las técnicas de aprendizaje profundo (Deep Learning) implementadas en este campo han generado unos avances extraordinarios que han permitido el nacimiento de **modelos grandes de lenguaje** (LLM, por sus siglas en inglés *Large Language Model*). El entrenamiento masivo de estos modelos ha hecho emerger unas capacidades impresionantes para generar texto coherente y relevante en diversas aplicaciones: traducción automática, generación de resúmenes, textos académicos, creación de contenido artístico, etc.

La motivación de este Trabajo de Fin de Máster (TFM) radica en el creciente interés y la importancia de los modelos de lenguaje en el campo de la Inteligencia artificial (IA) y en la sociedad en general.

1.2. Objetivos

El objetivo principal es comprender el funcionamiento y la situación actual de los modelos grandes de lenguaje, como **GPT-4** (*Generative Pre-trained Transformer 4*), desde una perspectiva matemática. Se busca analizar cómo estos modelos han evolucionado a lo largo del tiempo y cómo la matemática en general y la teoría de la optimización en particular han influido en su diseño y rendimiento.

Concretamente, este TFM persigue los siguientes objetivos específicos:

- Presentar una revisión actualizada de los fundamentos teóricos y matemáticos que subyacen en los modelos de lenguaje y las arquitecturas de redes neuronales utilizadas en el procesamiento de lenguaje natural.
- Estudiar la evolución de los modelos grandes de lenguaje, con especial énfasis en la arquitectura GPT y sus sucesivas versiones.
- Analizar las técnicas matemáticas que se aplican en el diseño, entrenamiento y evaluación de los modelos grandes de lenguaje, incluyendo la optimización, la convergencia y la generalización.
- Discutir los desafíos éticos y sociales asociados con el uso de modelos grandes de lenguaje y cómo estos problemas pueden ser abordados desde una perspectiva matemática y de gobernanza.

De esta forma, se pretende proporcionar una visión integral y actualizada de los modelos grandes de lenguaje y su situación en el campo de la IA. Asimismo, se espera que este trabajo contribuya a la comprensión de los aspectos matemáticos involucrados en el diseño y análisis de estos modelos, lo que puede ser de interés para investigadores y profesionales en el ámbito de la Matemática Aplicada y la Inteligencia Artificial.

1.3. Estructura del trabajo

Este TFM se organiza en cinco capítulos adicionales, además de esta introducción, para abordar de manera sistemática y rigurosa los objetivos específicos planteados en la sección

1.2.

Capítulo 2 - Procesamiento del lenguaje natural (NLP): Este capítulo presenta una introducción a los conceptos clave y las arquitecturas de redes neuronales utilizadas en el procesamiento del lenguaje natural (NLP). Se discuten las bases teóricas y matemáticas de los embeddings, las redes neuronales, así como las técnicas de optimización y evaluación empleadas en el entrenamiento de estos modelos.

Capítulo 3 - Modelos grandes de lenguaje (LLM): En este capítulo, se examina la evolución de los modelos de lenguaje, centrándose en las distintas versiones de GPT, incluido GPT-4. También se comparan las características y el rendimiento de GPT con otros modelos de lenguaje populares, como BERT y T5. Se analizan las aplicaciones y casos de uso de estos modelos, así como sus desafíos y limitaciones.

Capítulo 4 - Análisis matemático de los LLM: Este capítulo se centra en el estudio matemático de los modelos de lenguaje. Se investigan la complejidad computacional, la convergencia, la optimización y la generalización en el contexto de los modelos de lenguaje de gran escala. Además, se exploran medidas de rendimiento y evaluación, y se presentan enfoques matemáticos para abordar la interpretabilidad y la explicabilidad en estos modelos.

Capítulo 5 - Impacto ético y social de los modelos de lenguaje de gran escala: En este capítulo, se discuten las implicaciones éticas y sociales relacionadas con el uso de modelos de lenguaje de gran escala.

Capítulo 6 - Conclusiones: El último capítulo resume los principales hallazgos de este TFM y presenta una discusión sobre los retos y oportunidades en la investigación futura.

El TFM concluye con una sección de referencias bibliográficas que enumera todas las fuentes utilizadas en el desarrollo del trabajo. A lo largo del texto, se proporcionan ilustraciones, tablas y gráficos que complementan y respaldan la exposición teórica y matemática.

Capítulo 2

Procesamiento del lenguaje natural (NLP)

2.1. Introducción

El procesamiento del lenguaje natural (NLP, por sus siglas en inglés) es un subcampo interdisciplinario que combina la lingüística, la ciencia de la computación y la inteligencia artificial con el objetivo de permitir que las computadoras entiendan, interpreten y generen lenguaje humano de manera eficiente y efectiva. El NLP se centra en el análisis, la representación y la síntesis de texto y habla en lenguaje natural, con aplicaciones que abarcan desde la traducción automática hasta el análisis de sentimiento y la generación de texto.

Podemos considerar que los orígenes del NLP se remontan a 1950. En ese año, Alan Turing publicó *Computing machinery and intelligence* [66], donde propuso lo que ahora se conoce como el test de Turing como criterio de inteligencia. En 1954, se llevó a cabo el experimento de Georgetown, que involucró la traducción automática de más de sesenta oraciones del ruso al inglés. Los autores afirmaron que en tres o cinco años se resolvería el problema de la traducción automática [23]. Sin embargo, el avance real en este campo fue más lento y en 1966 el informe *Automatic Language Processing Advisory Committee* (Comité consultivo para el procesamiento avanzado del lenguaje, ALPAC por sus siglas

en inglés) [27] propuso que la investigación tenía un bajo rendimiento. Posteriormente, se llevaron a cabo investigaciones a menor escala en traducción automática y se desarrollaron los primeros sistemas de traducción automática estadística. Esto se debió tanto al aumento constante en la capacidad de procesamiento, resultado de la Ley de Moore (que expresa que cada dos años se duplica el número de transistores de un microprocesador) [49], como al cambio gradual en las teorías lingüísticas de Noam Chomsky (como la gramática transformacional, que entendía la gramática como un sistema de reglas formalizado con precisión matemática) [11]. Estas innovaciones desalentaron el enfoque de aprendizaje automático (Machine Learning) basado en corpus lingüísticos. En su lugar, se comenzaron a utilizar los primeros algoritmos de aprendizaje automático, como los árboles de decisión y los sistemas generados de reglas if-then similares a las reglas escritas manualmente. Habríamos de esperar hasta la década de 1980, cuando se produjo una revolución en el NLP con la introducción de algoritmos de aprendizaje automático para el procesamiento del lenguaje. El impulso en el que nos encontramos inmersos surgió a través de la aplicación de una disruptiva arquitectura de software llamada *transformer* basada en mecanismos de atención.

Vemos entonces que el NLP ha experimentado una evolución importante desde sus primeros enfoques basados en reglas y gramáticas formales hasta los enfoques más modernos que utilizan el aprendizaje automático y las redes neuronales. A lo largo de esta evolución, se han desarrollado diferentes técnicas y algoritmos para abordar tareas específicas en NLP, como el análisis sintáctico y la desambiguación semántica.

Las lenguas naturales presentan inherentemente diferentes niveles de ambigüedad. En el nivel léxico, una palabra puede tener múltiples significados, y es necesario deducir el significado adecuado a partir del contexto de la frase o conocimiento básico.

En el nivel estructural, la semántica juega un papel clave en la desambiguación de las dependencias de los sintagmas preposicionales, lo que conduce a la construcción de diferentes árboles sintácticos. Por ejemplo, en la frase *Rompió el dibujo de un bocado*, se requiere un análisis semántico para determinar si el dibujo fue destruido como consecuencia de un bocado o si representa un bocado.

En el nivel pragmático, se enfrenta el desafío de interpretar el significado real de una oración que, a menudo, puede ser diferente de su significado literal. La ironía y otros elementos pragmáticos desempeñan un papel crucial en la interpretación del mensaje. Diversas investigaciones han explorado el uso de modelos pragmáticos para capturar estos aspectos comunicativos y mejorar la comprensión del lenguaje natural.

El aprendizaje profundo, una rama del aprendizaje automático, ha demostrado ser particularmente efectivo en el NLP. Los modelos de aprendizaje profundo, como las redes neuronales, son capaces de capturar y representar características de alto nivel en datos complejos, como el lenguaje natural, mediante el uso de múltiples capas de procesamiento y transformaciones no lineales.

En el contexto del NLP, las redes neuronales se utilizan para aprender representaciones vectoriales de palabras, frases y textos completos, que permiten a las computadoras procesar y analizar el lenguaje de manera más eficiente y efectiva. Estas representaciones vectoriales, también conocidas como embeddings, son fundamentales para el funcionamiento de los modelos de lenguaje y se discuten con más detalle en la siguiente sección.

2.2. Embedding

Uno de los principales avances en NLP ha sido el uso de representaciones vectoriales, también conocidas como **embeddings**, para capturar y representar el significado y la sintaxis de las palabras, frases y textos completos en un espacio vectorial de dimensión baja [48]. Estos embeddings son aprendidos por los modelos de lenguaje a partir de grandes conjuntos de datos de texto, y permiten a las computadoras procesar y analizar el lenguaje de manera más eficiente y efectiva.

2.2.1. Hot encoding

La técnica de embedding más elemental es la codificación en caliente, o **hot encoding**. En este método, la palabra i-ésima se representa como un vector que contiene un 1 en la posición i-ésima y 0 en todas las demás. Este enfoque se emplea en el aprendizaje automático cuando se quiere representar características categóricas.

Las limitaciones de esta técnica son evidentes. Con vocabularios extensos, terminaremos con vectores muy largos, ya que la dimensión del vector es igual al tamaño del vocabulario. No obstante, este no es el inconveniente más significativo de este método. De acuerdo con la codificación en caliente, *guitarra* tiene tanta similitud con *violín* como con *lagarto*. Esto quiere decir que a través de este método los modelos no pueden capturar el significado de forma efectiva.

Si aspiramos a una representación más eficiente, necesitamos hallar una forma de definir el significado en términos de manipulación vectorial.

2.2.2. Word embeddings

En el momento en el que se escriben estas líneas, los humanos y los modelos de aprendizaje automático, sean de procesamiento de lenguaje natural o no, reciben los datos de manera diferente. Mientras que para nosotros es fácil comprender el significado de una frase como *vi un gato*, los modelos necesitan transformar cada palabra o parte de la palabra en vectores y realizar algunas operaciones sobre estos para interpretarla. Este proceso es lo que se conoce como **embedding**, la forma de representar las palabras que se integran en el modelo para su procesamiento.

Pongamos un ejemplo. La palabra *escribimos* podría dividirse en *escrib* que indica la acción de escribir y *imos* que refiere al sujeto de la acción. En un modelo de lenguaje, nos referimos como **token** a estas palabras o partes de palabras que van a convertirse en vectores para su procesamiento. De ahora en adelante, describiremos procesos que en definitiva, reciben y producen palabras.

Definimos entonces token como una unidad discreta de texto que se utiliza para dividir un escrito en partes más pequeñas y significativas. Para evitar ser repetitivo en la redacción (y porque en esencia es más exacto), a partir de ahora nos referiremos a las entradas de nuestros modelos indistintamente como palabras o tokens.

Los **word embeddings** son representaciones vectoriales de tokens en un espacio de alta dimensión que capturan su significado y relaciones semánticas y sintácticas. Estos embeddings son aprendidos por algoritmos como Word2Vec [48] y GloVe [51] a partir de

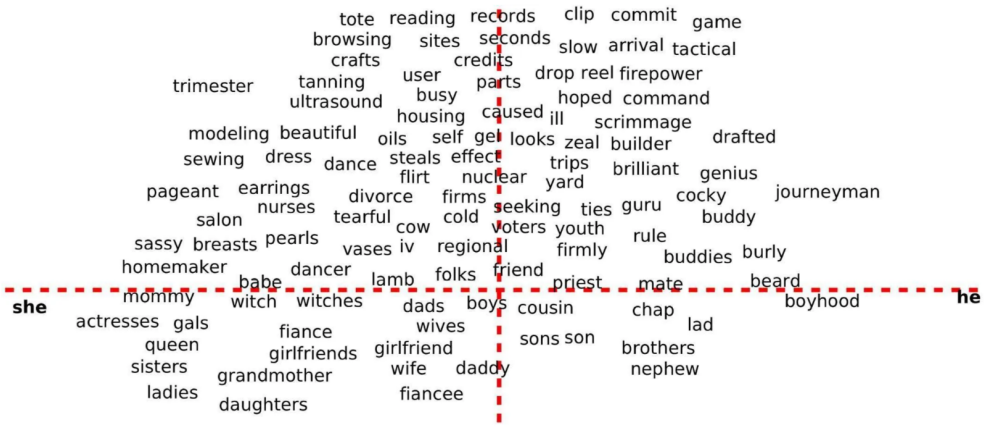

Figura 1: Word Embedding. Fuente towardsdatascience.com

grandes conjuntos de datos de texto, y son utilizados en una amplia variedad de tareas en NLP, como la clasificación de texto y la traducción automática.

El word embedding representa los tokens en un espacio vectorial de manera que si los vectores de una pareja de tokens están cerca entre sí, significa que están relacionados semántica o sintácticamente. En la Figura 1, podemos ver varias palabras que se asocian principalmente con mujeres agrupadas en el lado izquierdo, mientras que las palabras asociadas con hombres se agrupan en el lado derecho. Entonces, si pasamos la palabra *earrings* (*pendientes*), el procesador la relacionará con el género femenino, lo cual es correcto según estereotipos de género actuales.

Dado que el vocabulario de cualquier idioma es grande y no se puede etiquetar por humanos, se requieren técnicas de aprendizaje no supervisado que puedan aprender el contexto de cualquier palabra por sí mismas. Skip-gram es una de las técnicas de aprendizaje no supervisado utilizadas para modelar relaciones entre tokens.

Skip-gram

Skip-gram [38] es uno de los modelos disponibles en Word2vec junto con Continuous Bag-of-Words (CBOW). Es particularmente eficiente y funciona mejor que CBOW, por lo que lo tomaremos de referencia para exponer la elaboración de un word embedding. Skip-gram se utiliza para predecir el token de contexto para un token objetivo dado. Dado que hay más de un token de contexto que se debe predecir, esto hace que el problema sea

difícil.

En el modelo skip-gram, cada palabra/token $w \in W$ se asocia con un vector $v_w \in R_d$ y de manera similar, cada contexto $c \in C$ se representa como un vector $v_c \in R_d$, donde W es el vocabulario de palabras, C es el vocabulario de contextos y d es la dimensionalidad del embedding. Las entradas en los vectores son latentes y se tratan como parámetros a ser aprendidos.

El espacio latente se refiere a un espacio en el que se representan las características o propiedades relevantes de un conjunto de datos. Es un espacio de menor dimensión que el espacio original de los datos y se construye de manera que captura la estructura subyacente o las relaciones significativas entre los elementos del conjunto de datos. En nuestro caso, el word embedding es un espacio latente.

Hablando de manera general, buscamos valores de parámetros (es decir, representaciones vectoriales tanto para tokens como para contextos) de tal manera que el producto escalar $v_w \cdot v_c$ asociado con pares *buenos* de palabra-contexto se maximice.

En el procesamiento de lenguaje natural, el muestreo negativo consiste en generar ejemplos negativos al tomar secuencias de palabras aleatorias o incorrectas y contrastarlas con secuencias válidas y coherentes. Esto permite al modelo aprender a producir secuencias de palabras más precisas.

Skip-gram, realiza un muestreo negativo sobre un conjunto de datos D de pares observados (w, c) de palabras w y contextos c, que aparecen en un gran corpus de texto. Consideremos un par palabra-contexto (w, c). Ahora conviene contrastar si este par proviene del corpus de texto. Denotamos por $p(D = 1|w, c)$ la probabilidad de que (w, c) provenga de los datos, y por $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ la probabilidad de que (w, c) no provenga. La distribución se modela de la siguiente manera:

$$p(D = 1|w, c) = \frac{1}{1 + e^{-v_w \cdot v_c}} \tag{2.1}$$

donde v_w y v_c (cada uno un vector d-dimensional) son los parámetros del modelo que se deben aprender. Buscamos maximizar la log-probabilidad de los pares observados perte-

necientes a los datos, lo que lleva a la expresión:

$$\arg\max_{v_w,v_c} \sum_{(w,c)\in D} \log\left(\frac{1}{1+e^{-v_c\cdot v_w}}\right) \tag{2.2}$$

Esta expresión admite una solución trivial en la cual $p(D=1|w,c)=1$ para cada par (w,c). Esto puede lograrse fácilmente estableciendo $v_c = v_w$ y $v_c \cdot v_w = K$ para todos los c y w, donde K es un número lo suficientemente grande.

Con el fin de evitar la solución trivial, la función objetivo se extiende con pares (w,c) para los cuales $p(D=1|w,c)$ debe ser baja, es decir, pares que no están en los datos. Definimos entonces el conjunto D' de pares (w,c) aleatorios donde asumimos que todos son incorrectos. Esto nos ofrece la siguiente función objetivo a optimizar:

$$\arg\max_{v_w,v_c} \sum_{(w,c)\in D} \log\sigma(v_c\cdot v_w) + \sum_{(w,c)\in D'} \log\sigma(-v_c\cdot v_w) \tag{2.3}$$

donde $\sigma(x)=\frac{1}{1+e^{-x}}$. La función objetivo se entrena utilizando gradiente estocástico sobre el corpus $D\cup D'$.

Las muestras incorrectas D' se pueden construir de diversas formas. Si seguimos el método propuesto por Mikolov (2013) [48]: para cada par $(w,c)\in D$ construimos n muestras $(w,c_1),\ldots,(w,c_n)$, donde n es un hiperparámetro y cada c_j se selecciona de acuerdo a su distribución univariante, esto es, la frecuencia de aparición de palabras individuales en un corpus o conjunto de datos de texto, elevada a la potencia $3/4$.

Optimizar esta función hace que los pares observados de palabra-contexto se *acerquen entre sí* dentro del embedding, mientras que se dispersan los pares no observados.

2.2.3. Sentence embeddings

Los **sentence embeddings** son representaciones vectoriales de frases o enunciados que tienen en cuenta el contexto y la estructura de las palabras en la frase [13]. Estos embeddings pueden ser aprendidos por modelos de lenguaje basados en redes neuronales, como las redes neuronales recurrentes (RNN) y los transformers, y se utilizan en tareas como la

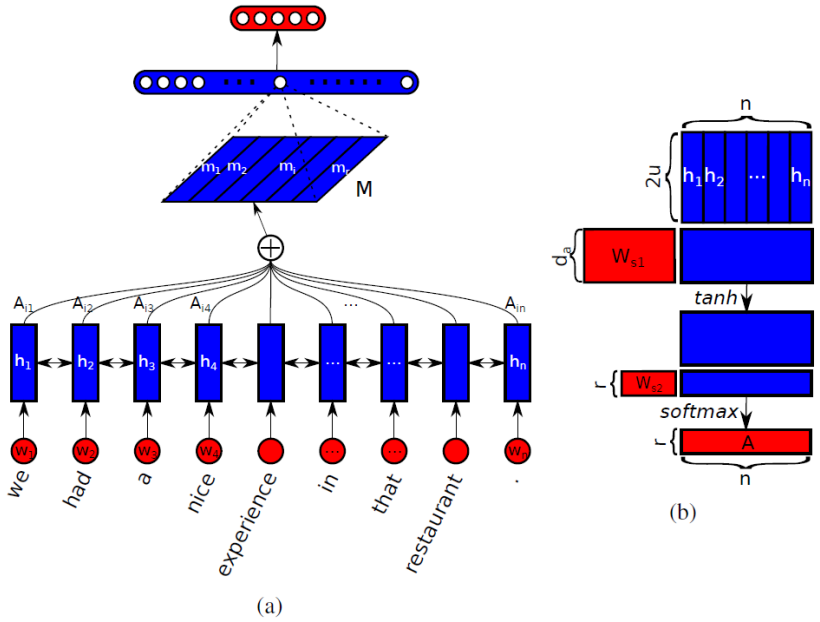

Figura 2: Sentence embedding combinado con una capa totalmente conectada y una capa softmax para el análisis de sentimientos (a). La suma de pesos $(A_1, ..., A_n)$ se calculan de la manera ilustrada en (b). Fuente *A structured self-atentive sentence embedding* [40]

detección de paráfrasis y la generación de resúmenes.

A continuación, describiremos matemáticamente el modelo *Structured Self-Attention*. La Figura 2 muestra un *sentence embedding* aplicado al análisis de sentimientos. Supongamos que tenemos una frase que consta de n tokens/palabras, si tomamos el ejemplo mencionado $n = 9$:

- "We have a nice experience in that restaurant."

Donde el punto . tambien se modela como un token. Sea W_i el word embedding del i-ésimo token/palabra de la frase, esto es, un vector n-dimensional. Sea S la matriz que representa a la frase, es decir, que concatena todos los tokens.

$$S = \begin{pmatrix} W_1 \\ \dots \\ W_n \end{pmatrix} = \begin{pmatrix} W_{11} & \dots & W_{1d} \\ \dots & \dots & \dots \\ W_{n1} & \dots & W_{nd} \end{pmatrix} \tag{2.4}$$

Donde podemos ver que S tiene dimensión nd. En el ejemplo de la Figura 2:

$$S = \begin{pmatrix} W_1 \\ \dots \\ W_n \end{pmatrix} = \begin{pmatrix} We \\ have \\ \dots \\ restaurant \\ . \end{pmatrix} = \begin{pmatrix} W_{11} & \dots & W_{1d} \\ \dots & \dots & \dots \\ W_{91} & \dots & W_{9d} \end{pmatrix} \tag{2.5}$$

En principio se considera que los vectores que forman cada fila de S son linealmente independientes [40]. (Esto es, que los tokens son independientes entre sí). Para introducir una dependencia entre las palabras de las frases se emplea una red neuronal bidireccional Long-Short-Term-Memory (LSTM) para procesar la frase.

$$\overrightarrow{h_t} = \overrightarrow{LSTM}(w_t, \overrightarrow{h_{t-1}}) \tag{2.6}$$

$$\overleftarrow{h_t} = \overleftarrow{LSTM}(w_t, \overleftarrow{h_{t+1}}) \tag{2.7}$$

Concatenamos cada estado oculto $\overrightarrow{h_t}$ con $\overleftarrow{h_t}$ para formar un estado oculto h_t. Sea u el número de unidades ocultas en cada LSTM unidireccional. Para simplificar, representamos todos los n estados ocultos como H, que tiene una dimensión de $n2u$.

$$H = \begin{pmatrix} h_1 \\ \dots \\ h_n \end{pmatrix} \tag{2.8}$$

A continuación, se emplea un mecanismo de *Self-Atention* que consiste en establecer combinaciones lineales con los n vectores ocultos de la LSTM en H. Este mecanismo de atención toma como entrada todos los estados ocultos de la LSTM en H y produce un vector de pesos a.

$$a = \begin{pmatrix} a_1 \\ \dots \\ a_n \end{pmatrix} \tag{2.9}$$

Para generar este vector de pesos a utilizaremos la función $softmax$ (Ver Pág 36, sección

2.4.3) que convierte un vector de entrada en una distribución de probabilidad:

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.10}$$

Donde $softmax(x)$ es la probabilidad de que la i-ésima entrada pertenezca a la clase j, x es el vector de entrada y K el número de clases. La función $softmax$ se utiliza para garantizar que la suma de los pesos calculados sea igual a 1.

$$a = \text{softmax}(w_{s_2} \tanh(W_{s_1} H^T)) \tag{2.11}$$

Aquí, d_a es un hiperparámetro que podemos elegir arbitrariamente, W_{s_1} es una matriz de pesos con dimensión $d_a 2u$ y w_{s_2} es un vector de parámetros de dimensión d_a. Dado que H tiene dimensiones $n2u$, el vector de pesos a también tendrá tamaño n.

$$aH = m \tag{2.12}$$

donde m es una representación vectorial de la frase de entrada.

Esta representación vectorial generalmente se enfoca en un componente específico de la frase, como un conjunto especial de palabras o frases relacionadas para reflejar un componente de su semántica. En el caso de oraciones largas puede haber varios componentes en una frase que juntos formen la semántica general (por ejemplo, dos cláusulas vinculadas por un *y*). Por lo tanto, necesitamos múltiples m que se centren en diferentes partes de la oración. ello implica realizar múltiples iteraciones de atención.

Suponemos que queremos dividir y procesar r partes de la frase. Para esto, ampliamos w_{s_2} en una matriz rd_a, llamada W_{s_2}, y el vector de pesos resultante a se convierte en una matriz de pesos A. Formalmente, esto lo expresamos de la siguiente manera:

$$A = \text{softmax}(W_{s_2} \tanh(W_{s_1} H^T)) \tag{2.13}$$

Aquí, softmax se aplica a lo largo de la segunda dimensión de su entrada. Obtenemos entonces una matriz de sentence embedding M de dimensión $r2u$. Las r sumas ponderadas

se obtienen de multiplicar la matriz de pesos A por los estados ocultos de LSTM H:

$$M = AH \tag{2.14}$$

Término de penalización

Puede ocurrir que el mecanismo de atención proporcione unas matrices de pesos similares para las r partes de la frase, dando lugar a problemas de redundancia. Para prevenirlo añadimos un término de penalización que fomenta la diversidad.

Para esta sección se describe un término de penalización introducido en *A structured self-atentive sentence embedding*. Utilizamos el producto matricial de A y su transpuesta, restado por una matriz identidad, como medida de redundancia.

$$P = |(A^\top A - I)|_F^2 \tag{2.15}$$

Aquí, $||_F$ representa la norma de Frobenius de una matriz. Este término de penalización P se multiplica por un coeficiente y se minimiza junto con la pérdida original, que depende de la aplicación posterior.

Consideremos dos vectores diferentes a_i y a_j en A. Debido a la función softmax, todas las entradas dentro de cualquier vector de peso en A deben sumar 1. Por lo tanto, se pueden considerar como masas de probabilidad en una distribución de probabilidad discreta. Para cualquier elemento no diagonal a_{ij} $(i \neq j)$ en la matriz $A^\top A$, corresponde a:

$$0 < a_{ij} = \sum_{k=1}^{n} a_k^i a_k^j < 1 \tag{2.16}$$

donde a_k^i y a_k^j son el k-ésimo elemento en los vectores a_i y a_j, respectivamente. En el caso más extremo, donde no hay superposición entre las dos distribuciones de probabilidad a_i y a_j, el correspondiente a_{ij} será 0. De lo contrario, tendrá un valor positivo. En el otro extremo, si las dos distribuciones son idénticas y se concentran en una sola palabra, tendrá un valor máximo de 1. A $A^\top A$ le restamos la matriz identidad para forzar que los elementos en la diagonal de $A^\top A$ se aproximen a 1, lo que obliga a cada vector a

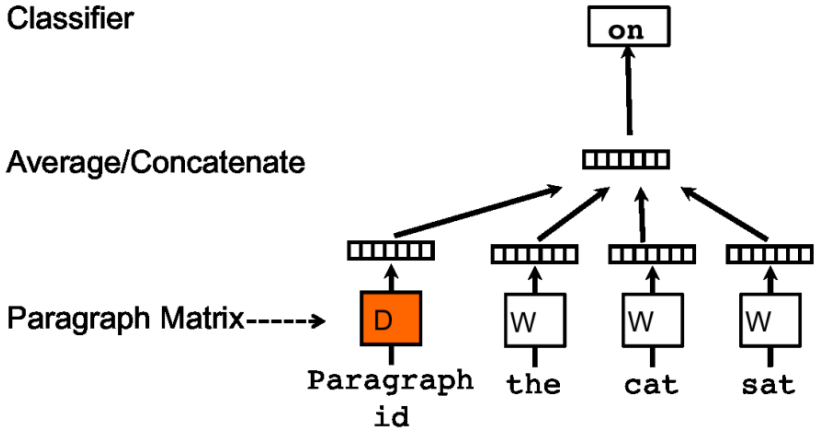

Figura 3: Pragraph Vector Modelo. Fuente *Distributed Representations of Sentences and Documents* [35]

enfocarse en un solo aspecto, forzando a su vez a todos los demás elementos a tender a 0, lo que castiga la redundancia entre diferentes vectores de peso.

2.2.4. Document embeddings

Los **document embeddings** son representaciones vectoriales de textos completos que capturan su contenido semántico y temático a nivel de documento. Estos embeddings pueden ser aprendidos por algoritmos como Doc2Vec [35] y utilizados en tareas como la clasificación de documentos y la recuperación de información.

Paragraph Vector model

Propuesto en 2014 en *Distributed Representations of Sentences and Documents* de Mikolov Le [35] el modelo **Paragraph Vector**, también conocido como **Doc2Vec**, es un enfoque de aprendizaje no supervisado utilizado para generar representaciones vectoriales de documentos de texto. A diferencia de los modelos tradicionales basados en bolsas de palabras (bag-of-words) que representan documentos como vectores de frecuencias de palabras, el modelo Paragraph Vector captura tanto la información contextual de las palabras como la estructura global del documento.

En el modelo Paragraph Vector (ver Figura 3), cada párrafo se asigna a un vector único, representado por una columna en la matriz D, y cada palabra también se asigna a un

vector único, representado por una columna en la matriz W. El vector del párrafo y los vectores de palabras se promedian o concatenan para predecir la siguiente palabra en un contexto.

En comparación con el word embedding la principal diferencia se encuentra en la siguiente ecuación, donde se construye h a partir de W y D. Cada y_i es la log-probabilidad no normalizada para cada palabra de salida i, calculada como:

$$y = b + U \cdot h(w_{t-k}, \ldots, w_{t+k}; W) \tag{2.17}$$

donde U y b son los parámetros del softmax y h se construye mediante una concatenación o promedio de los vectores de palabras extraídos de W.

El token asociado al párrafo se puede considerar como otra palabra. Actúa como una memoria que recuerda lo que falta en el contexto actual, o el tema del párrafo. Por esta razón, los autores [35] llaman también a este modelo el Modelo de Memoria Distribuida de Paragraph Vectors (PV-DM).

El vector correspondiente al párrafo se utiliza en todos los contextos generados a partir del mismo párrafo, pero no se comparte entre párrafos. Sin embargo, la matriz de vectores de palabras W se comparte entre los párrafos. En otras palabras, el vector asociado a la palabra *barco* es idéntico en todos los párrafos.

Tanto los vectores de párrafo como los de palabras se entrenan empleando *el descenso de gradiente estocástico* (SGD) (Ver Pág 36, sección 2.4.4) y **retropropagación**. De esta forma se muestrea un contexto de longitud fija de un párrafo aleatorio, se calcula el gradiente de error a partir de la red y se usa el gradiente para actualizar los parámetros en el modelo.

Supongamos que hay N párrafos en el corpus y M palabras en el vocabulario. Si deseamos aprender vectores de párrafo de p dimensiones y vectores de palabras de q dimensiones, entonces el modelo tendrá un total de $N \times p + M \times q$ parámetros (sin incluir los parámetros de la función softmax). Aunque el número de parámetros puede ser grande cuando N es grande, las actualizaciones durante el entrenamiento tienden a ser dispersas,

lo que hace que el proceso sea eficiente.

Una vez que los vectores de párrafo están entrenados, se pueden utilizar como características del párrafo. Por ejemplo, se pueden emplear en lugar o además del enfoque de la representación de bolsa de palabras. Estas características de los párrafos pueden ser directamente alimentadas a técnicas convencionales de aprendizaje automático, como regresión logística, máquinas de vectores de soporte o K-means.

2.2.5. Semántica distribucional

La *hipótesis distributiva* establece que palabras con significados similares aparecen en contextos similares, por lo que la similitud semántica permitirá efectuar cambios distribucionales. Esta hipótesis fue planteada por el lingüista británico JR. Firth en su libro "Una sinopsis de la teoría lingüistica" donde se popularizó la siguiente idea:

"Una palabra se caracteriza por la compañía que mantiene"[17]

Para aclararlo con un ejemplo, consideremos una palabra ficticia "Tezquiri"que, en principio, no tiene ninguna asociación con otras palabras y, por lo tanto, no tiene significado. No obstante, si le otorgamos un contexto con base en las siguientes oraciones:

Hay un plato de Tezquiri sobre la mesa.

El Tezquiri me causa indigestión.

El Tezquiri proviene de Perú.

Podríamos deducir que Tezquiri se refiere a alguna clase de comida peruana típica que contiene algún ingrediente que puede ser pesado o difícil de digerir. En este sentido, la hipótesis distributiva propone que cuanto más parecidas sean dos palabras en su significado, más similares serán en su distribución en los textos y, por lo tanto, tendrán más probabilidades de aparecer en contextos lingüísticos similares.

La hipótesis distribucional es la base para la semántica distribucional, una subdisciplina de la lingüística que examina las propiedades distributivas en grandes conjuntos de datos lingüísticos. De este modo, se generan teorías y métodos para cuantificar y clasificar

similitudes semánticas entre los elementos lingüísticos.

Modelado semántico distributivo en espacios vectoriales.

La combinación de álgebra lineal y semántica distributiva ofrece un poderoso mecanismo computacional y un marco para la representación de palabras. La información distributiva de una palabra o token puede ser representada en vectores N-dimensional, y la semejanza distribucional/semántica entre estos tokens puede ser expresada en términos de similitud vectorial. Se pueden deducir varios tipos de similitudes dependiendo del tipo de información de distribución que se utilice para formar los vectores:

Semejanzas de actualidad: Si el modelo de vector N-dimensional se entrena en conjuntos de datos (áreas de texto) donde se encuentran los elementos lingüísticos y sus relaciones de significado actualizado. Por ejemplo, *José Mesa es el nuevo presidente de Perú. Las políticas de Mesa son liberales en lo económico y socialistas en lo laboral.* En este caso, *Mesa* adquiere connotaciones e implicaciones muy diferentes a su equivalencia como *un mueble con una superficie horizontal sostenida por uno o más soportes*.

Semejanzas paradigmáticas: Entrenando los vectores con información acerca de los otros elementos lingüísticos con los que conviven los elementos. Por ejemplo, *En la tienda venden mesas y sillas de madera. Necesito comprar un mantel para la mesa.* Aquí, podríamos entender que el término *mesa* se agrupa muy cerca de otros términos como *silla*, *mueble*, *mantel* en su espacio semántico de Ndimensional.

Semejanzas sintagmáticas: Estas similitudes requerirían un análisis comparativo de los componentes individuales del vector de N-dimensional. Los términos *Silla* y *Mesa* podrían tener algunas componentes muy similares entre sí, de lo cual podríamos inferir que esas componentes hablan de aspectos utilitarios, de tamaño o materiales típicos de construcción. Es importante recordar que estamos comparando representaciones numéricas que surgen del entrenamiento, por lo que no podemos tener la certeza de que un componente particular del vector tenga una correspondencia exacta con una característica física del mundo real.

Existen muchos modelos distintos que implementan computacionalmente las ideas de la

semántica distributiva, correlacionando la similitud distribucional y de significado. Entre ellos, podríamos mencionar el Análisis Semántico Latente (LSA), el Hiperespacio Analógico al Lenguaje (LAH), modelos basados en sintaxis o dependencia, indexación aleatoria, plegamiento semántico y diversas variantes del modelo temático.

Aunque el propósito de este trabajo no es detallar los pros y contras de cada uno de estos modelos cuando se aplican en tareas específicas, las diferencias entre ellos se centran en distintos parámetros:

- **Tipo de contexto.** Diferentes áreas de texto versus elementos lingüísticos.

- **Ventana de contexto.** Extensión, tamaño, etc.

- **Ponderación de la frecuencia.** Información mutua puntual, entropía, etc.

- **Métodos de reducción de dimensiones.** Análisis de componentes principales, indexación aleatoria, descomposición de valores singulares, etc.

- **Medida de similitud.** Similitud del coseno, distancia de Minkowski, entre otros.

En las siguientes secciones, se explorarán en detalle las arquitecturas de redes neuronales empleadas en NLP para aprender representaciones vectoriales y modelar el lenguaje humano. Se discutirán también las técnicas de optimización y evaluación empleadas en el entrenamiento de estos modelos, así como los desafíos y oportunidades en la investigación futura en el campo del NLP y las representaciones vectoriales.

2.3. Arquitecturas de redes neuronales en NLP

Una red neuronal artificial (RNA) es un modelo computacional no lineal basado en la estructura neuronal del cerebro que es capaz de aprender a realizar tareas como clasificación, predicción, toma de decisiones, visualización y otras, solo considerando ejemplos.

Una red neuronal artificial está compuesta por neuronas artificiales o elementos de procesamiento, y se organiza en tres capas interconectadas: la capa de entrada, la capa oculta que puede incluir más de una capa, y la capa de salida.

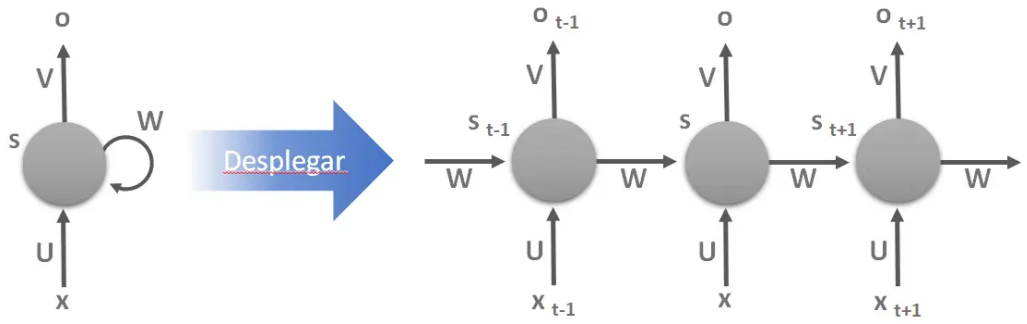

Figura 4: Un diagrama de una red neuronal desplegada. Figura adaptada de *Survey on Recurrent Neural Network in Natural Language Processing* [63]

Las redes neuronales han demostrado ser especialmente efectivas en el procesamiento del lenguaje natural (NLP) debido a su capacidad para modelar relaciones complejas y no lineales en datos de alta dimensión, como el lenguaje humano [22]. En esta sección, se describen las principales arquitecturas de redes neuronales utilizadas en NLP, incluyendo las redes neuronales recurrentes (RNN), las redes neuronales convolucionales (CNN) y los transformers.

2.3.1. Redes neuronales recurrentes (RNN)

Las redes neuronales recurrentes (RNN, por sus siglas en inglés) son una clase de redes neuronales diseñadas específicamente para modelar secuencias de datos, como las secuencias de palabras en el lenguaje natural [26].

Las Redes Neuronales Recurrentes (RNN) fueron creadas en la década de 1980, pero recientemente han ganado popularidad debido al aumento de la capacidad computacional de las unidades de procesamiento gráfico. Son especialmente útiles con datos secuenciales porque cada neurona puede utilizar su memoria interna para mantener información sobre la entrada anterior. Una metáfora ilustrativa de su comportamiento es pensar en una *memoria* de las neuronas que guarda o captura información sobre lo que se ha calculado hasta el momento.

La Figura 4 muestra una RNN desplegada (o expandida) en una red completamente conectada. Al desplegarla, simplemente escribimos la red para toda la secuencia. Por ejemplo, si la secuencia que nos interesa es una oración de 6 palabras, la red se desplegaría en una

red neuronal de 6 capas, una capa por cada palabra. Las fórmulas que muestran el cálculo que ocurre en una RNN para el procesamiento del lenguaje natural son las siguientes:

x_t es la entrada en el paso de tiempo t. Por ejemplo, x_t podría ser un vector hot-encoding correspondiente a la segunda palabra de una oración.

s_t es el estado oculto en el paso de tiempo t. Es la *memoria* de la red, s_t se calcula en función del estado oculto anterior y la entrada en el paso actual: $s_t = f(Ux_t + Ws_{t-1})$. La función f suele ser no lineal como la tangente hiperbólica (tanh) o ReLU. Para calcular el primer estado oculto iniciamos con s_{t-1}, un vector con todas sus componentes igual a cero.

o_t es la salida en el paso t. Por ejemplo, si quisiéramos predecir la siguiente palabra en una oración, sería un vector de probabilidades en nuestro vocabulario V. $o_t = \text{softmax}(Vs_t)$

Definamos ahora el conjunto de datos de entrenamiento D_{train} y el subconjunto x:

$$D_{\text{train}} = \{x_1, \ldots, x_M\} \tag{2.18}$$

$$x = [x_1, \ldots, x_N] \tag{2.19}$$

Donde, por ejemplo: $x = [\text{Este}, \text{agua}, \text{es}, \text{tan}, \text{transparente}, \text{STOP}]$.

Queremos generar un modelo que devuelva: $P(x)$ para todo $x \in V^{\text{MaxN}}$, donde V es el vocabulario y V^{MaxN} son todas las posibles frases:

$$P(x) = P(x_1, \ldots, x_N) = P(x_1)P(x_2, \ldots, x_N|x_1) = P(x_1)P(x_2|x_1)\ldots P(x_N|x_1)$$

Entonces, las probabilidades se pueden escribir como:

$$P(x_n = k|x_{n-1}\cdots x_1) = \frac{\exp(w_k \cdot \phi(x_{n-1}\cdots x_1))}{\sum_{k'=1}^{V} \exp(w'_k \cdot \phi(x_{n-1}\cdots x_1))} \tag{2.20}$$

donde w_k son los pesos para el token k, y $\phi(x_{n-1}\cdots x_1)$ son las características extraí-

das de las palabras anteriores (codificación one-hot de $x_{n-1} \dots x_1$), $p(x_n|x_{n-1} \dots x_1) = \text{softmax}(W\phi(x_{n-1} \dots x_1))$ donde $W \in \mathbb{R}^{|v| \times |\text{contexts}|}$ tiene pesos para cada palabra y contexto con un vector $s_{n-1} \in \mathbb{R}^d$, $p(x_n|x, s_{n-1} \dots x_1) = \text{softmax}(Vs_{n-1})$ donde $V \in \mathbb{R}^{|v| \times d}$ mapea el contexto a una distribución de probabilidad sobre las palabras. Ahora, para obtener s_{n-1}, necesitamos la RNN:

$$s_n = \sigma(Ws_{n-1} + Ux_n) \tag{2.21}$$

donde $s_{n-1} \in \mathbb{R}^d$ es la *memoria* del contexto; x_{n-1}. $W \in \mathbb{R}^{d \times d}$ controla cómo se pasa esta memoria; $U \in \mathbb{R}^{|v| \times d}$ es la matriz que contiene los vectores de palabras para todas las palabras; x_n selecciona una palabra. Y para obtener la distribución de probabilidad para la palabra x_n:

$$P(x_{n-1}) = p(x_n|x_{n-1} \dots x_1) = \text{softmax}(Vs_{n-1}) \tag{2.22}$$

Necesitamos aprender los vectores de palabras U, los parámetros de la capa oculta y de salida W, V. El método de retropropagación estándar no puede funcionar debido a la recurrencia, es decir, reutilizamos los parámetros de la capa oculta W. Para nuestro caso, necesitamos una modificación: la retropropagación a través del tiempo (BPTT) la cual nos permite desenrollar un grafo durante n pasos y sumar las contribuciones de los gradientes en la actualización.

Sin embargo, las RNN convencionales sufren de problemas de memoria a corto plazo y dificultades para aprender dependencias temporales de largo alcance debido al problema del desvanecimiento del gradiente [4]. Para abordar estos problemas, se han propuesto variantes de RNN, como las *redes de memoria a corto y largo plazo* (LSTM) y las *redes neuronales recurrentes compuerta* (GRU), que son capaces de aprender y modelar dependencias temporales más largas [26, 10], mostrando cómo las convoluciones y el agrupamiento pueden extraer características locales e invariantes en secuencias de palabras.

2.3.2. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son una clase de redes neuronales que utilizan convoluciones para extraer características locales e invariantes en datos espaciales, como imágenes [37]. Aunque las CNN se han utilizado principalmente en el campo de la visión por computadora, también han demostrado ser efectivas en el NLP para tareas como la clasificación de texto y la extracción de entidades [69].

Para describir el funcionamiento de estas redes vamos a detallar como procesan un sentence embedding. Para ello vamos a definir previamente algunos conceptos necesarios:

- **Ventana de palabras** Se define como la cantidad de palabras vecinas que se toman en cuenta a cada lado de la palabra objetivo. Estas serán las evaluadas para predecir las palabras futuras.

- **Vector de conexión** Se refiere al vector de pesos que se aplican a cada unidad o filtro de convolución en la capa anterior para calcular la salida de la capa actual. Estos pesos determinan la influencia relativa de las características de entrada en la generación de características de salida.

- **Convolución** Es una operación que combina dos funciones para crear una tercera función que representa cómo una de las funciones influye en la otra. La operación de convolución entre dos funciones se denota como:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau)\, d\tau \tag{2.23}$$

  donde $f(t)$ y $g(t)$ son las dos funciones que se están convolucionando y $*$ representa la operación de convolución.

  En el caso discreto, cuando se trabaja con secuencias o datos discretos, la convolución se define como:

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k] \cdot g[n - k] \tag{2.24}$$

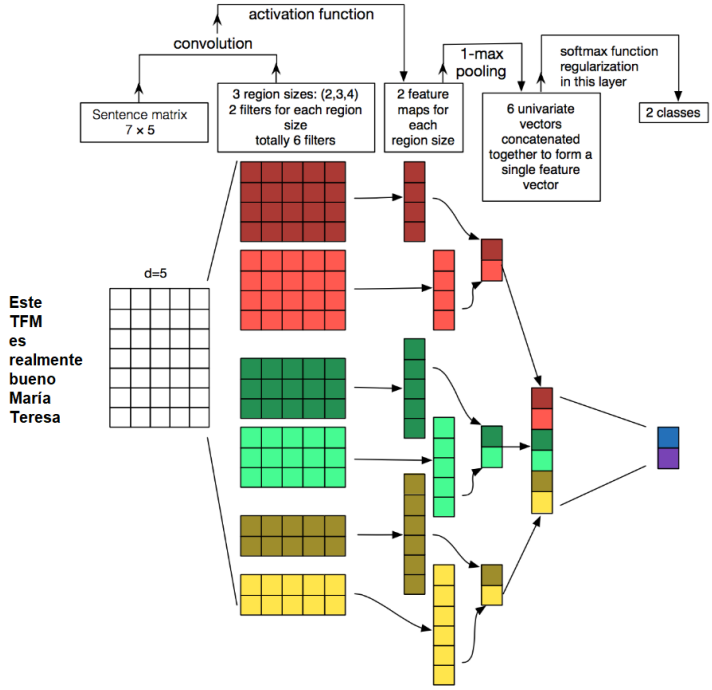

Figura 5: Un diagrama que ilustra la estructura de una red neuronal convolucional aplicada al procesamiento del lenguaje natural. Fuente adaptada desde *Application of Convolutional Neural Network in Natural Language Processing* [39]

donde $f[n]$ y $g[n]$ son las secuencias o datos discretos que se están convolucionando y $*$ representa la operación de convolución.

- **Pooling** Consiste en subdividir la salida de la capa convolucional en regiones más pequeñas y resumir cada región en un solo valor representativo. Este valor representativo puede ser el máximo (Max Pooling) o el promedio (Average Pooling) de los valores en la región.

Supongamos que queremos procesar un texto. Para cada frase i , denotamos $x_i \in \mathbb{R}^d$ la representación del word embedding de dimensión d adjudicado a una palabra. Dado que hay n palabras en una frase, esta se puede expresar como una matriz de vectores $x_n \in \mathbb{R}^{d \times n}$. La Figura 5 muestra una frase de siete palabras como una matriz de entrada de CNN [39].

Notemos por $x_{i:i+j}$ el vector de conexión $x_i, x_{i+1} \ldots, x_{i+j}$. Realizamos la convolución sobre el vector de entrada. Los filtros denotados como $k \in \mathbb{R}^{hd}$ se aplican a ventanas de tamaño h para producir nuevas características.

Por ejemplo, para usar las palabras $x_{i:i+h-1}$ en una ventana de generación de características c_i

$$c_i = f(x_{i:i+h-1}k^\top + b) \tag{2.25}$$

donde $b \in R$ es el desplazamiento (offset), f es una función de activación no lineal, como la función tangente hiperbólica. El filtro k se aplica a todas las posibles ventanas y crea planos de características con el mismo peso.

$$c = [c_1, c_2, ..., c_{n-h+1}] \tag{2.26}$$

En CNN, diferentes filtros de convolución de diferentes anchos (también llamados núcleos) se deslizan a través de la matriz del word embedding completo. Cada núcleo extrae una característica específica del patrón de N-Gramas. En el contexto del procesamiento de texto, un N-grama es una secuencia de N palabras, caracteres o cualquier otro elemento que se considere relevante.Los N-gramas son útiles para capturar información contextual y patrones en el texto. Por ejemplo, un N-grama de tamaño 2 (bigrama) captura la relación entre dos palabras consecutivas en un texto, mientras que un N-grama de tamaño 3 (trigrama) captura la relación entre tres palabras consecutivas.

La convolución generalmente se sigue de una estrategia de **máx pooling**, que es igual que el pooling pero tomando el máximo de cada división.

2.3.3. Transformers

A mediados de 2017, un grupo de ingenieros de Google junto a un investigador de la Universidad de Toronto publican un pequeño (pero revolucionario) trabajo de investigación con el sugerente título de “Atention is all you need (La atención es lo único que necesitas)” [67].

Aunque los mecanismos de atención habían aparecido en el contexto de las redes neuronales desde 2015 no es hasta este año que se propone una ingeniosa nueva arquitectura de aprendizaje profundo que podría emplearse para la traducción de un idioma a otro, superando a las redes secuenciales 2-seq-LSTM-seq que eran las empleadas hasta ese momento. Lo que no se preveía por entonces es que las aplicaciones de su Transformer

-el nombre que dieron a esta nueva arquitectura de red neuronal- podría ser aplicado en campos tan diversos como las redes generativas, el aprendizaje por refuerzo o la visión artificial alcanzando unas cotas de corrección nunca antes vistas y convirtiéndose en la base del acelerado avance en IA en el que nos encontramos.

Los transformers han demostrado ser muy efectivos en una amplia variedad de tareas en NLP, incluida la traducción automática, la generación de texto y la comprensión del lenguaje natural. Además, los transformers han sido la base de modelos de lenguaje de gran tamaño, como BERT, GPT-4 y T5, que han establecido nuevos estándares de rendimiento en una amplia gama de tareas en NLP [41, 71].

Aunque aquí lo introducimos de forma muy superficial, la arquitectura del transformers y sus mecanismos de atención se describirán detalladamente en el capítulo 3.

2.4. Entrenamiento y optimización de modelos de lenguaje

En esta sección, vamos a describir algunos conceptos y las técnicas utilizadas en el entrenamiento de modelos de lenguaje, que van a ser necesarios para entender los siguiente capítulos incluida **la función de pérdida**, **el descenso de gradiente** y las funciones de activación. Más adelante, en el capítulo 4 retomaremos algunos conceptos y veremos otros en profundidad como las técnicas de regularización.

Para estos conceptos nos basaremos en la **teoría de la información**. Fue desarrollada originalmente para estudiar el envío de mensajes desde alfabetos discretos a través de un canal ruidoso, como la comunicación vía transmisión de radio. En este contexto, la teoría de la información explica cómo diseñar códigos óptimos y calcular la longitud esperada de los mensajes muestreados de distribuciones de probabilidad específicas utilizando varios esquemas de codificación. Para nuestro caso, también podemos aplicar la teoría de la información a variables continuas donde algunas de estas interpretaciones de longitud de mensaje no se aplican. De esta forma, utilizamos principalmente algunas ideas clave de la teoría de la información para caracterizar distribuciones de probabilidad o cuantificar

la similitud entre distribuciones de probabilidad.

2.4.1. Funciones de pérdida

Las funciones de pérdida, también conocidas como funciones de costo o funciones de error, son utilizadas en el entrenamiento de modelos de aprendizaje automático para medir la discrepancia entre las predicciones del modelo y los valores reales de los datos de entrenamiento. Estas funciones cuantifican la diferencia entre las salidas predichas por el modelo y los valores objetivo, permitiendo así evaluar qué tan bien está realizando el modelo en comparación con los datos de entrenamiento.

En esencia, al minimizar la función de pérdida durante el entrenamiento, el modelo busca encontrar los valores óptimos de los parámetros que reduzcan al máximo la discrepancia entre las predicciones y los valores reales.

Se utilizan diferentes tipos de funciones de pérdida dependiendo del tipo de problema de aprendizaje y la naturaleza de los datos. Algunos ejemplos comunes son el error cuadrático medio (MSE), la entropía cruzada categórica (CCE), el error absoluto medio (MAE), entre otros. Cada función de pérdida tiene sus propias características y propósitos, y es seleccionada en función de las necesidades específicas del problema y del tipo de modelo utilizado [70].

Vamos a ver como trabaja la función de pérdida entropía cruzada en el problema de clasificación de k clases. Consideremos $X \subseteq \mathbb{R}^d$ el espacio de características de nuestros datos y $Y = \{1, \ldots, c\}$ el espacio de etiquetas. Supondremos que nos encontramos un escenario ideal, en el que se nos proporciona un conjunto de datos limpio $D = \{(x_i, y_i)\}_{i=1}^n \subseteq (X \times Y)$.

La función que se encarga de mapear el espacio de características de entrada al espacio de etiquetas se llama clasificador y se define como $f : X \to \mathbb{R}^c$. En este trabajo, consideramos el caso común donde la función es una red neuronal con la capa de salida softmax. Definimos el riesgo del clasificador f como:

$$R_L(f) = \mathbb{E}_D[L(f(x), y_x)] \tag{2.27}$$

Donde la esperanza es sobre la distribución empírica. Usaremos **la entropía cruzada** por ser una de las funciones de pérdida más empleadas. Su expresión es la siguiente:

$$H(p, q) = -\sum(p(x)\log(q(x))) \tag{2.28}$$

Donde $H(p, q)$ es la entropía cruzada entre las distribuciones p y q, $p(x)$ es la probabilidad real o verdadera de que ocurra el evento x, $q(x)$ es la probabilidad estimada o predicha de que ocurra el evento x.

En este caso, el riesgo se convierte en:

$$R_L(f) = \mathbb{E}_D[L(f(x;\theta), y_x)] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}\log f_j(x_i;\theta) \tag{2.29}$$

Aquí θ representa el conjunto de parámetros del clasificador, y_{ij} corresponde al j-ésimo elemento de la etiqueta codificada empleando **one-hot** (codificación de variables categóricas) de la muestra x_i, $y_i = e_{y_i} \in (0, 1)^c$ tal que $1^T y_i = 1 \forall i$.

f_j denota el j-ésimo elemento de f. Nótese que $\sum_{j=1}^{c} f_j(x_i;\theta) = 1$, y $f_j(x_i;\theta) \geq 0 \forall j, i, \theta$, esto es debido a que tenemos una función softmax en la capa de salida.

Sea $D_\eta = \{(x_i, \hat{y}_i)\}_{i=1}^n$ un conjunto de datos con ruido en las etiquetas, donde $\hat{y}_i$ son las etiquetas ruidosas en relación a cada muestra tal que $p(\hat{y}_i = k|y_i = j, x_i) = \eta_{jk}^{x_i}$. Suponemos también que el ruido es independiente de manera condicional a las entradas dadas las etiquetas verdaderas, de modo que $p(\hat{y}_i = k|y_i = j, x_i) = p(\hat{y}_i = k|y_i = j) = \eta_{jk}^{x_i}$.

Definimos este ruido como *dependiente de la clase*. El ruido es *uniforme* con una tasa de ruido η si $\eta_{jk} = 1 - \eta$ para $j = k$, y $\eta_{jk} = \eta/(c - 1)$ para $j \neq k$. El riesgo del clasificador con respecto al conjunto de datos ruidosos se define entonces como

$$R_L^\eta(f) = \mathbb{E}_{D_\eta}[L(f(x), e^{y_x})]. \tag{2.30}$$

Sea f el minimizador global del riesgo $R_L(f)$. Entonces, la minimización del riesgo empírico bajo la función de pérdida L se define como tolerante al ruido si f es un mínimo

global del riesgo ruidoso $R_{\overline{L}}(f)$.

Pérdida de regresión logística

La pérdida de regresión logística es una función de pérdida relacionada con la entropía cruzada que se utiliza en modelos de clasificación binaria. Esta función de pérdida mide la discrepancia entre las predicciones del modelo y las verdaderas etiquetas binarias [6].

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{2.31}$$

donde $J(\theta)$ es la pérdida de regresión logística, N es el número total de ejemplos en el conjunto de datos, y_i es la etiqueta verdadera del ejemplo i, y $\hat{y}_i$ es la probabilidad predicha de la clase positiva para el ejemplo i.

Esta ecuación da una gran pérdida cuando la predicción del modelo está lejos de la verdadera etiqueta y una pequeña pérdida cuando la predicción del modelo está cerca de la verdadera etiqueta. En otras palabras, minimizar esta pérdida lleva a que el modelo haga predicciones más cercanas a las verdaderas etiquetas.

2.4.2. Estadísticas y procesamiento de datos textuales

Las técnicas estadísticas también desempeñan un papel importante en el análisis y modelado de datos textuales en el NLP. Por ejemplo, el análisis de la distribución de palabras en un corpus de texto puede proporcionar información valiosa sobre las propiedades del texto y ayudar en la identificación de palabras clave y temas relevantes [44].

La Ley de Zipf es una ley empírica que establece que la frecuencia de cualquier palabra en un texto o corpus es inversamente proporcional a su rango en la lista de frecuencias de palabras (Ver Figura 6). Es decir, la palabra más común aparecerá aproximadamente dos veces más a menudo que la segunda más común, tres veces más a menudo que la tercera más común, y así sucesivamente.

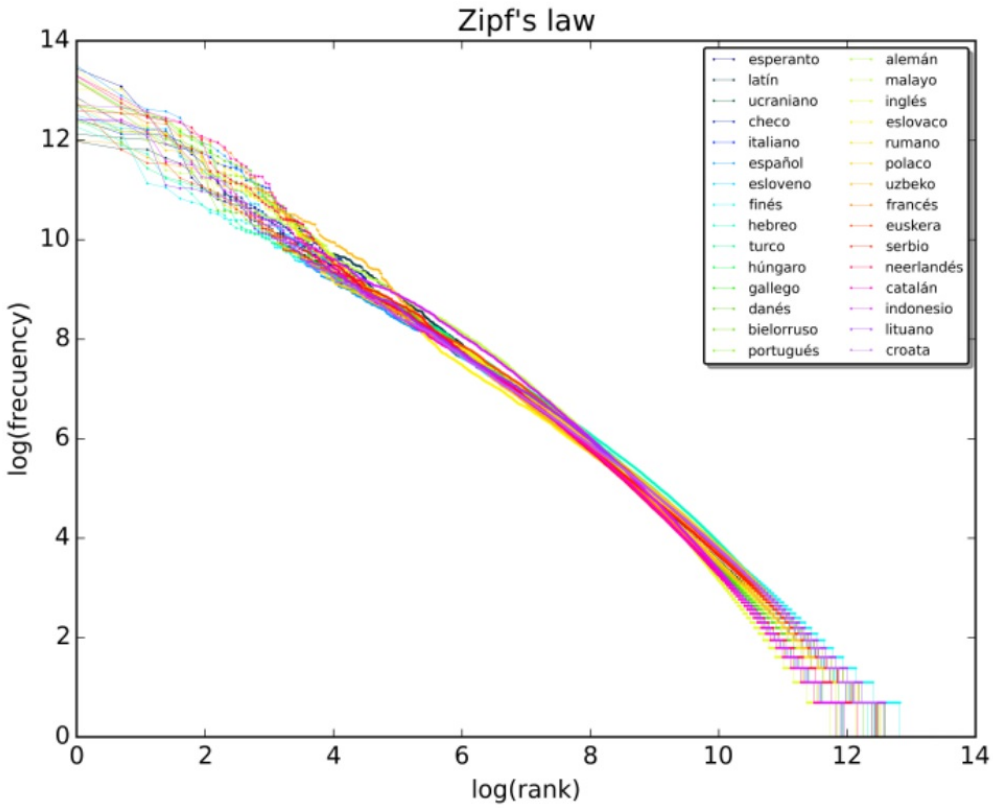

Figura 6: Gráficas de ranking versus la frecuencia de las primeras 10 millones de palabras en 30 Wikipedias (descargas de octubre del 2015) en una representación logarítmica en los dos ejes. Fuente extraída de Wikipedia

Matemáticamente, la Ley de Zipf se puede expresar de la siguiente manera:

$$f = \frac{c}{r} \tag{2.32}$$

donde, f es la frecuencia de una palabra, r es el rango de la palabra, c es una constante. El rango de una palabra en el contexto de la ley de Zipf se refiere a su posición en el orden descendente de frecuencias. La palabra más frecuente tendrá un rango de 1, la segunda palabra más frecuente tendrá un rango de 2, y así sucesivamente.

La ley de Zipf es más comúnmente representada en una forma logarítmica como:

$$\log f = \log c - s \cdot \log r \tag{2.33}$$

donde s es la pendiente de la relación (generalmente cercana a 1 para el lenguaje natural).

Estas ecuaciones describen la relación general entre la frecuencia y el rango de las palabras en un texto de acuerdo con la Ley de Zipf. Sin embargo, en la práctica, puede haber variaciones debido a una variedad de factores, como el tamaño del corpus, la longitud del texto, la naturaleza del idioma, entre otros.

2.4.3. Funciones de activación

Las funciones de activación (Ver Figura 7) son un componente esencial en las redes neuronales artificiales, ya que introducen la no linealidad en el modelo y permiten que la red aprenda representaciones más complejas y ricas de los datos [22]. En esta sección, se explorarán las funciones de activación más utilizadas en el contexto de los modelos de lenguaje y sus propiedades matemáticas.

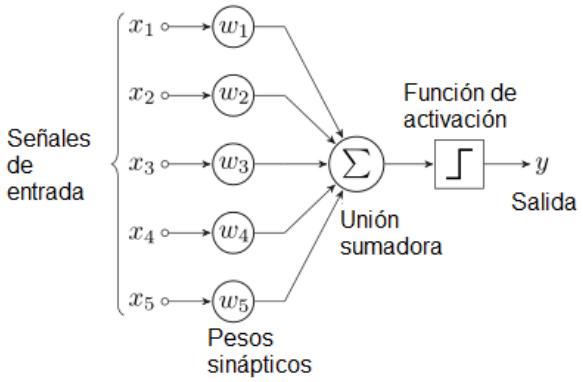

Figura 7: Un diagrama que ilustra cómo las funciones de activación se aplican a las salidas de las unidades en una red neuronal. Elaboración propia.

Función sigmoide

La función sigmoide es una función de activación clásica que toma un valor de entrada y lo transforma en un valor en el rango de (0, 1). Esta función se utiliza a menudo en las capas de salida de los modelos de clasificación binaria, ya que puede interpretarse como la probabilidad de que una entrada pertenezca a una de las dos clases

Tenemos que $\lim_{x\to\infty} \sigma(x) = 1$, $\lim_{x\to-\infty} \sigma(x) = 0$, y $\sigma(0) = 0{,}5$, esto hace que la función sigmoide sea especialmente útil para convertir números reales en probabilidades.

Función tangente hiperbólica (tanh)

La función tangente hiperbólica (tanh) es otra función de activación popular que transforma un valor de entrada en un valor en el rango de $(-1, 1)$. La función $\tanh$ es similar a la función sigmoide en términos de forma, pero tiene un rango de salida diferente, lo que la hace más adecuada para ciertas tareas y arquitecturas de redes neuronales, como las redes neuronales recurrentes (RNN) [37].

La ecuación para la función tanh es:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.34}$$

En este caso, $\lim_{x\to\infty} \tanh(x) = 1$, $\lim_{x\to-\infty} \tanh(x) = -1$, y $\tanh(0) = 0$, lo que significa que centra su salida alrededor de 0, algo que puede ser útil en ciertas aplicaciones.

Rectificador lineal unitario (ReLU)

El rectificador lineal unitario (ReLU) es una función de activación simple pero efectiva que se ha vuelto muy popular en el entrenamiento de redes neuronales profundas (DNN), es decir, de múltiples capas ocultas. La función ReLU toma un valor de entrada y lo transforma en un valor en el rango de $[0, +\infty)$, donde los valores negativos se ajustan a 0 y los valores positivos se mantienen sin cambios [50].

$$ReLU(x) = \text{máx}(0, x) \tag{2.35}$$

La función ReLU tiene varias propiedades deseables, como la capacidad de mitigar el problema del desvanecimiento del gradiente durante el entrenamiento y una computación eficiente en términos de tiempo y recursos [21]. Sin embargo, también puede presentar problemas de "neuronas muertas", en las que algunas unidades de la red dejan de actualizar sus pesos durante el entrenamiento debido a la saturación en 0 [25].

ReLU paramétrico (PReLU)

El ReLU paramétrico (PReLU) es una variante de la función ReLU que introduce un parámetro entrenable para controlar la pendiente de la función de activación en la región negativa. Esta adaptación permite una mayor flexibilidad en la función de activación y puede ayudar a resolver problemas de neuronas muertas asociados con la función ReLU [25].

$$PReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ a \cdot x, & \text{if } x < 0 \end{cases} \tag{2.36}$$

donde a es un parámetro aprendido durante el entrenamiento que determina la pendiente de la función para los valores de entrada negativos.

PReLU permite a la red aprender la cantidad de activación que debe pasar incluso para las entradas negativas, lo que puede mejorar el rendimiento del modelo en algunos casos. Sin embargo, introduce un nuevo parámetro que debe aprenderse para cada neurona en la red, lo que puede aumentar la complejidad y el tiempo de entrenamiento del modelo.

Función de activación softmax

La función de activación softmax hemos observado que se utiliza en las capas de salida de los modelos de clasificación multiclase, ya que convierte un vector de entrada en una distribución de probabilidad discreta sobre las clases objetivo [6]. La función softmax garantiza que la suma de las probabilidades de todas las clases sea igual a 1, lo que facilita la interpretación y el cálculo de la función de pérdida en el entrenamiento del modelo.

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.37}$$

donde $\text{softmax}(x)_i$ es la probabilidad de que la entrada pertenezca a la clase i, x es el vector de entrada y K es el número total de clases.

2.4.4. Optimización

Los algoritmos de optimización ajustan los parámetros del modelo para minimizar las pérdidas [22]. En esta sección, se examinarán alguos algoritmos de optimización utilizados en el contexto de los modelos de lenguaje.

Descenso de gradiente estocástico (SGD)

El descenso de gradiente estocástico (SGD) es un algoritmo de optimización simple pero efectivo que se utiliza para minimizar la función de pérdida en el entrenamiento de modelos de lenguaje basados en redes neuronales. El algoritmo SGD ajusta los parámetros del modelo utilizando una estimación del gradiente de la función de pérdida basada en un subconjunto aleatorio de los datos de entrenamiento, en lugar de utilizar todo el conjunto de datos, lo que resulta en una convergencia más rápida y un menor costo computacional [37].

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t) \tag{2.38}$$

donde θ_t es el valor del parámetro en el paso de tiempo t, η es la tasa de aprendizaje, que es un hiperparámetro que controla cuánto cambian los parámetros en cada paso de la optimización, y $\nabla J(\theta_t)$ es el gradiente de la función de pérdida J con respecto a los parámetros en el paso de tiempo t.

Esta ecuación muestra cómo el gradiente de la función de pérdida se usa para ajustar los parámetros del modelo en cada paso de la optimización. El signo negativo indica que estamos descendiendo en la dirección del gradiente, es decir, moviéndonos en la dirección que reduce la pérdida.

Adam

El método de Adam (Adaptive Moment Estimation) es un algoritmo de optimización popular que se utiliza para actualizar los parámetros de los modelos de aprendizaje automático. Adam combina las ventajas de dos extensiones del descenso de gradiente estocástico: la estimación adaptativa de tasas de aprendizaje individuales para diferentes parámetros (como en Adagrad) y el uso de promedios móviles de los gradientes en lugar de los gra-

dientes en sí (como en RMSProp) [33].

Las ecuaciones para la actualización de los parámetros usando Adam son las siguientes:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.39}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.40}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.41}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.42}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.43}$$

donde θ_t es el valor del parámetro en el paso de tiempo t, g_t es el gradiente de la función de pérdida con respecto a los parámetros en el paso de tiempo t, m_t es la estimación del primer momento (la media) del gradiente en el paso de tiempo t, v_t es la estimación del segundo momento (la varianza no centralizada) del gradiente en el paso de tiempo t, $\hat{m}_t$ es la corrección del sesgo para la estimación del primer momento en el paso de tiempo t, $\hat{v}_t$ es la corrección del sesgo para la estimación del segundo momento en el paso de tiempo t, α es la tasa de aprendizaje, β_1 y β_2 son parámetros de decaimiento exponencial para las estimaciones de los momentos (típicamente establecidos en 0.9 y 0.999, respectivamente), ϵ es un pequeño escalar para la estabilidad numérica (típicamente alrededor de 1e-8).

La actualización de los parámetros en Adam utiliza estimaciones de los primer y segundo momentos del gradiente para adaptar la tasa de aprendizaje de cada parámetro. Esto puede resultar en un entrenamiento más rápido y eficiente en comparación con otros métodos de optimización.

La elección adecuada de la función de pérdida y el algoritmo de optimización puede tener un impacto significativo en el rendimiento, la eficiencia y la estabilidad del modelo durante el entrenamiento.

Capítulo 3

Modelos Grandes de Lenguaje (LLM)

Un **modelo grande de lenguaje** (LLM por sus siglas en inglés: *Large Language Model*) se refiere a un tipo de modelo de aprendizaje automático que se ha entrenado con una enorme cantidad de datos de texto, del orden de miles de millones, para comprender y generar lenguaje humano de manera efectiva. Estos modelos están diseñados para capturar patrones lingüísticos complejos y tienen la capacidad de generar texto coherente y relevante.

Los modelos grandes de lenguaje, como GPT-4 [7] y BERT [15], han demostrado un rendimiento extraordinario en una amplia gama de tareas de procesamiento del lenguaje natural (NLP) y han revolucionado el campo del NLP en los últimos años. Se basan en arquitecturas de redes neuronales profundas y se entrenan con grandes conjuntos de datos de texto utilizando técnicas de aprendizaje supervisado y no supervisado. En esta sección, pretendemos examinar los conceptos fundamentales y las características clave de los modelos de lenguaje a gran escala, incluidas sus arquitecturas, métodos de entrenamiento, comparación y aplicaciones.

3.1. Evolución de los modelos de lenguaje

A lo largo de los años, los modelos de lenguaje han evolucionado desde enfoques basados en reglas y gramáticas hasta modelos basados en estadísticas y, más recientemente, en

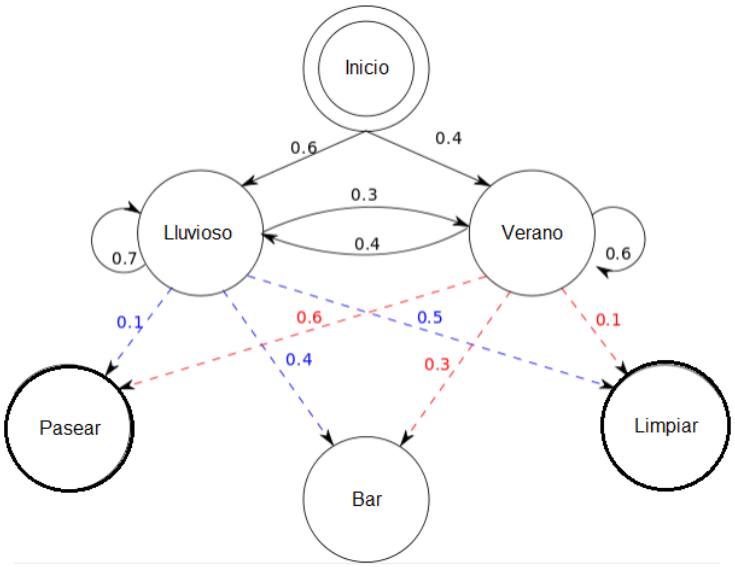

Figura 8: Un diagrama de un modelo de Markov de un modelo de lenguaje estadístico. Elaboración propia.

redes neuronales profundas. Esta sección traza la evolución de los modelos de lenguaje y destaca algunas de las arquitecturas y técnicas clave en cada etapa.

3.1.1. Modelos de lenguaje basados en reglas y gramáticas

Como comentamos, los primeros modelos de lenguaje se basaban en reglas y gramáticas manuales, donde los expertos en lingüística codificaban explícitamente las estructuras gramaticales y las relaciones semánticas en el texto [10]. Aunque estos enfoques proporcionaban una comprensión detallada de las estructuras lingüísticas, su aplicabilidad práctica en el procesamiento del lenguaje natural era limitada debido a la falta de escalabilidad y adaptabilidad.

3.1.2. Modelos de lenguaje estadísticos

Con el advenimiento de las técnicas de aprendizaje automático, los modelos de lenguaje comenzaron a basarse en enfoques estadísticos, como los modelos de Markov (Ver Figura 8) [28]. Estos modelos aprenden las estructuras y regularidades del lenguaje a partir de grandes conjuntos de datos de texto y permiten la generación y clasificación de secuencias de palabras de acuerdo con sus probabilidades.

Un modelo de Markov, particularmente uno de orden 1, es un tipo de modelo estadístico que se utiliza comúnmente para modelar secuencias de datos. Puede usarse para representar la probabilidad de que una palabra dada aparezca después de una palabra o secuencia de palabras previas. Aquí la probabilidad de que una palabra particular aparezca sólo depende de la palabra anterior. Si consideramos un texto con palabras $w_1, w_2, \ldots, w_N$, entonces la probabilidad de que todo el texto aparezca es el producto de las probabilidades de que cada palabra aparezca después de la palabra anterior:

$$P(w_1, w_2, ..., w_N) = P(w_1)P(w_2|w_1)P(w_3|w_2)...P(w_N|w_{N-1}) \tag{3.1}$$

Aquí, $P(w_i|w_{i-1})$ es la probabilidad de que la palabra w_i aparezca después de la palabra w_{i-1}.

Los Modelos de Markov son *ingenuos* en el sentido de que solo consideran la información local -las n palabras anteriores- al predecir la siguiente palabra. No tienen en cuenta la estructura a largo plazo de la secuencia de palabras, lo que puede llevar a predicciones que no tienen sentido en un contexto más amplio. Para superar esta limitación se utilizan las RNN y la arquitectura transformer, que pueden almacenar información contextual a largo plazo.

3.1.3. Modelos de lenguaje basados en redes neuronales

Con el auge del aprendizaje profundo, los modelos de lenguaje basados en redes neuronales, como las redes neuronales recurrentes (RNN), las redes de memoria a largo plazo (LSTM) y los mecanismos de atención, han ganado popularidad [26, 67]. Estos modelos pueden aprender representaciones de palabras y secuencias de palabras en espacios vectoriales de alta dimensión, lo que permite la captura de relaciones semánticas y gramaticales complejas.

3.1.4. Modelos grandes de lenguaje basados en transformers

En los últimos años, los modelos grandes de lenguaje basados en la arquitectura de transformer, como BERT [15] y GPT-4 [7], han demostrado un rendimiento excepcional en

una amplia gama de tareas de NLP. Estos modelos utilizan mecanismos **Multi-head attention** y **self-attention** para capturar las dependencias de largo alcance en el texto y se entrenan con grandes conjuntos de datos utilizando técnicas de aprendizaje supervisado y no supervisado.

Como responsable del gran avance actual de los modelos grandes de lenguaje, vamos a analizar detalladamente la arquitectura transformer.

3.2. Fundamentos matemáticos del transformer

Las arquitecturas dominantes en el Procesamiento del Lenguaje Natural (PLN) hasta la introducción de los Transformers eran las redes neuronales secuenciales y convolucionales de alta complejidad. Estas contaban con un codificador para convertir cada palabra o token en un vector numérico N-dimensional, y un decodificador para revertir esta transformación una vez realizada la traducción. Se observó que los sistemas de traducción más efectivos eran aquellos que vinculaban el codificador y el decodificador a través de un mecanismo de atención. Esta observación llevó al diseño del Transformer, una innovadora y simplificada arquitectura de red neuronal que descarta los métodos recurrentes y convolucionales y se enfoca exclusivamente en los mecanismos de atención.

Los transformers, gracias a sus mecanismos de atención, demostraron ser no solo traductores más eficaces, sino que también poseían una mayor capacidad de paralelización (ideal para la traducción de grandes volúmenes de texto), consumían solo una fracción de los recursos utilizados por los sistemas anteriores y necesitaban menos tiempo para ser entrenados.

3.2.1. El problema de la traducción automática

Iniciamos con una secuencia de origen $\vec{F} = (f_0, f_1, ..., f_n)$, donde cada f_i representa una palabra individual tomada de un conjunto de palabras de origen. Nuestro objetivo es predecir una traducción de la secuencia de origen $\vec{F}$ a otro idioma, conocida como secuencia objetivo $\vec{E} = (e_0, e_1, ..., e_m)$, la cual está compuesta por palabras e_i pertenecientes a un conjunto de palabras objetivo.

Los idiomas humanos no se traducen simplemente palabra a palabra, por lo que tenemos la dificultad añadida de que las secuencias $\vec{F}$ y $\vec{E}$ no tienen porqué tener la misma longitud.

Expresado de manera probabilística, buscamos modelar la distribución

$$P(\vec{E}|\vec{F}) = P(e_0, e_1, ..., e_m|\vec{F})$$

La probabilidad conjunta de $\vec{E}$ se puede expresar como un producto de probabilidades condicionales:

$$P(\vec{E}|\vec{F}) = P(e_0|\vec{F}) \cdot P(e_1|e_0, \vec{F}) \cdot ... \cdot P(e_{m-1}|e_0, ..., e_{m-2}, \vec{F}) \cdot P(e_m|e_0, ..., e_{m-1}, \vec{F})$$

Las redes neuronales recurrentes han sido la base de la mayor parte de avances en traducción automática, gracias a su *memoria* que permitía modelizar el contexto. Sin embargo, un sistema codificador-decodificador basado en RNN presenta algunos problemas de implantación: Cuanto más larga sea la entrada, más ruidosa será la información de las partes anteriores de la secuencia. Esto impide que se puedan hacer predicciones en secuencias largas. La solución parecía estar según los investigadores en condensar toda la información de los estados ocultos del sistema en un vector que fuera consistente con el tamaño de la entrada. Así nacieron los mecanismos de atención, que sirven para condensar los estados ocultos en una suma ponderada de sus vectores.

3.2.2. Mecanismos básicos de atención

El mecanismo de atención de un modelo enfocado en la traducción de un texto permite evaluar cada una de las palabras de la frase original, convertidas en vector N-dimensional, y tomar una decisión sobre cómo traducirlas para obtener una frase de salida.

En la Figura 9 observamos la traducción del Inglés al Francés de la frase *The agreement on the European Economic Area was signed in August 1992*. En *L´accord sur la zone économique européenne a été signé en août 1992*. Como observamos, para presentar la palabra *zona*, presta atención a la palabra *Area* y a la palabra *Economic*.

¿Cómo se modela la atención dentro de una arquitectura de software? Para comprenderlo,

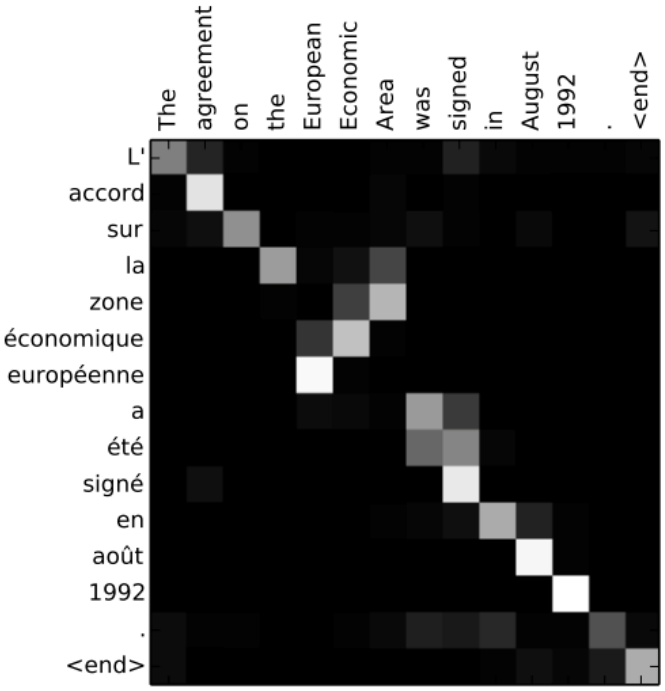

Figura 9: Mapa de calor de la atención extraída del artículo «Neural Machine Translation by Jointly Learning to Align and Translate (2015)».[2]

partiremos del modelado matemático de los ejemplos más sencillos de atención, **Hard atention** y **Soft-atention**, para pasar a ver el **Self-atention**, la base sobre la que se fundamenta el modelo del Transformer.

¿Qué es la atención?

Un mecanismo de atención, en el contexto de las redes neuronales, consiste en una operación matemática que recibe como inputs un conjunto de vectores -que ya sabemos que pueden representar texto, imágenes o cualquier tipo de datos con el que trabajemos- los combina entre sí y nos da como resultado otro conjunto de vectores. Este resultado dependerá, obviamente, del tipo de mecanismo de atención que utilicemos. Vamos a ver algunos ejemplos.

Hard Attention

El mecanismo de atención más sencillo de entender es el conocido como hard attention. Consideremos un conjunto de vectores bidimensional X y un tensor de hard attention A:

$$X = \begin{pmatrix} 1 & 0 & -0{,}5 \\ 0 & 1 & -0{,}5 \end{pmatrix} \tag{3.2}$$

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

Vemos que el hard attention toma la forma de la matriz identidad, pues presta atención a un único vector.

$$S = XA = \begin{pmatrix} 1 & 0 & -0{,}5 \\ 0 & 1 & -0{,}5 \end{pmatrix} \tag{3.4}$$

La operación es trivial, pero nos sirve de ejemplo para ilustrar cómo, si la atención se centra en un único vector de entrada por vector de salida, no tendremos información en S acerca del contexto. Este ejemplo sería equivalente a traducir una palabra por su correspondiente significado más común en el otro idioma, sin atender al resto de palabras que la rodean.

Soft Attention

En el caso del soft atention vamos a permitir prestar atención a todos los vectores de la entrada X. De esta forma, el vector de salida S será una combinación lineal del resto de los inputs. Para hacer el ejemplo, cada vector generado va a prestar un $80\,\%$ de atención al vector de X en su misma posición y un $10\,\%$ al resto.

$$A = \begin{pmatrix} 0{,}8 & 0{,}1 & 0{,}1 \\ 0{,}1 & 0{,}8 & 0{,}1 \\ 0{,}1 & 0{,}1 & 0{,}8 \end{pmatrix} \tag{3.5}$$

$$S = XA = \begin{pmatrix} 0{,}75 & 0{,}05 & -0{,}3 \\ 0{,}05 & 0{,}75 & -0{,}3 \end{pmatrix} \tag{3.6}$$

Vemos que los mecanismos de atención son en realidad la combinación lineal de una manera determinada de vectores que tenemos de entrada para darnos otra de vectores a la salida. En el caso del soft attention, hemos propuesto unos pesos relativos en A a cada uno de los vectores a la entrada X y eso es lo que nos da la salida S. En un mecanismo de atención que prestara la misma atención a todos los vectores para generar la salida tendríamos el siguiente caso:

$$A = \begin{pmatrix} 0{,}333 & 0{,}333 & 0{,}333 \\ 0{,}333 & 0{,}333 & 0{,}333 \\ 0{,}333 & 0{,}333 & 0{,}333 \end{pmatrix} \tag{3.7}$$

El resultado que obtenemos es siempre el mismo, pues presta la misma atención todo el rato a todos los vectores. No nos sería útil porque obtendríamos la misma salida.

Self Attention

En este caso, cada vector es responsable de decidir por sí mismo cuanta atención debe prestar al resto. Para ello calcularemos la similitud entre vectores, de tal forma que cuanto más parecidos sean, más atención habrá entre ellos y viceversa. Para poner un ejemplo ilustrativo, en una tarea de traducción de texto que emplea un mecanismo de self attention prestaría más atención a aquellas palabras más relacionadas en la entrada (Recordamos que al ser las palabras vistas como una representación de vectores N-dimensional serían aquellas, por ejemplo, con menor distancia euclidiana en ese espacio de representación) y así no desperdicia computación entre aquellas que no tengan importancia.

Esta similitud la podemos calcular multiplicando los vectores por sí mismos y aplicando la función exponencial normalizada, **softmax** definida en el capítulo anterior en la ecuación (2.33). En nuestro ejemplo anterior tendríamos la siguiente matriz de atención, resulta de

multiplicar la entrada X por sí misma traspuesta y aplicar softmax.

$$A = softmax(XX^t) = \begin{pmatrix} 0{,}6285 & 0{,}2312 & 0{,}2119 \\ 0{,}2312 & 0{,}6285 & 0{,}2119 \\ 0{,}1402 & 0{,}1402 & 0{,}5761 \end{pmatrix} \tag{3.8}$$

Esta matriz de atención no la hemos codificado número a número como las anteriores, sino que depende de cómo se parecen entre sí los vectores que tenemos en la entrada.

$$S = XA = \begin{pmatrix} 0{,}5584 & 0{,}1611 & -0{,}0761 \\ 0{,}1611 & 0{,}5584 & -0{,}0761 \end{pmatrix} \tag{3.9}$$

Podemos notar una pequeña similitud con los resultados cuando tomábamos un 80 por ciento de atención al vector principal, pero ahora estamos considerando un poco más el resto de vectores sin que necesitemos proponer una atención arbitraria para cada uno, sino completamente regulada por la entrada. Vamos a ver como este mecanismo a gran escala con los vectores N-dimensionales permite un modelo del contexto en términos de la representación del lenguaje.

3.2.3. Scale Dot-Product Attention

En la Figura 10 vemos tres conjuntos de entrada representados por Q, K y V. Estos operan de forma muy similar conceptualmente a la X del caso anterior. Reciben respectivamente los nombres de *Queries*, *Keys* y *Values* (Consultas, Claves y Valores).

Cuando se realiza la traducción, el modelo de atención calcula la similitud entre la consulta (query) y cada una de las claves (keys) utilizando una medida de similitud como el producto escalar o la atención puntual. Luego, se utiliza la similitud calculada para ponderar los valores (values) asociados con cada clave. Los valores ponderados se combinan para generar la salida final en español. Esta fragmentación de la entrada permite atender en paralelo tanto la traducción específica de la palabra *heureux* por la palabra *feliz* como la atención necesaria del contexto para determinar que sea *feliz* y no *dichoso*, *exitoso*, *satisfecho*, *complacido* o cualquier otra acepción aceptada.

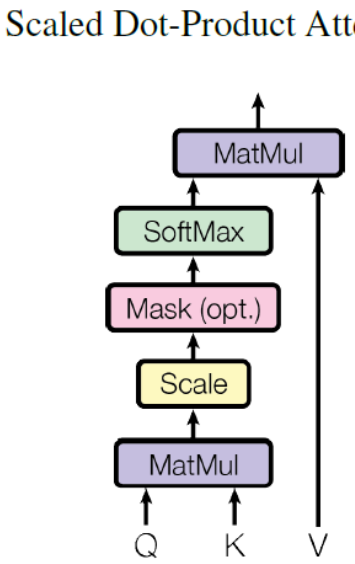

Figura 10: Esquema del Scale-dot product».[67]

Vamos a escribir las filas individuales de las matrices Q y K, y luego expresaremos el producto QK^T en términos de esas filas:

$$Q = \begin{pmatrix} - & \vec{q}_0 & - \\ - & \vec{q}_1 & - \\ - & \vdots & - \\ - & \vec{q}_m & - \end{pmatrix}; K^T = \begin{pmatrix} | & | & & | \\ \vec{k}_0 & \vec{k}_1 & \cdots & \vec{k}_n \\ | & | & & | \end{pmatrix}$$

$$QK^T = \begin{pmatrix} \vec{q}_0 \cdot \vec{k}_0 & \vec{q}_0 \cdot \vec{k}_1 & \cdots & \vec{q}_0 \cdot \vec{k}_n \\ \vec{q}_1 \cdot \vec{k}_0 & \vec{q}_1 \cdot \vec{k}_1 & \cdots & \vec{q}_1 \cdot \vec{k}_n \\ \vdots & \vdots & \ddots & \vdots \\ \vec{q}_m \cdot \vec{k}_0 & \vec{q}_m \cdot \vec{k}_1 & \cdots & \vec{q}_m \cdot \vec{k}_n \end{pmatrix}$$

Para obtener los pesos vamos a dividir cada elemento por $\sqrt{d_k}$ y vamos a plicar la función softmax por cada fila:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \begin{pmatrix} \text{softmax}(\frac{1}{\sqrt{d_k}}\langle \vec{q}_0 \cdot \vec{k}_0, \vec{q}_0 \cdot \vec{k}_1, ..., \vec{q}_0 \cdot \vec{k}_n \rangle) \\ \text{softmax}(\frac{1}{\sqrt{d_k}}\langle \vec{q}_1 \cdot \vec{k}_0, \vec{q}_1 \cdot \vec{k}_1, ..., \vec{q}_1 \cdot \vec{k}_n \rangle) \\ \vdots \\ \text{softmax}(\frac{1}{\sqrt{d_k}}\langle \vec{q}_m \cdot \vec{k}_0, \vec{q}_m \cdot \vec{k}_1, ..., \vec{q}_m \cdot \vec{k}_n \rangle) \end{pmatrix} = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,n} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,0} & s_{m,1} & \cdots & s_{m,n} \end{pmatrix}$$

donde se tiene que para cada fila i:,

$$\sum_{j=0}^{n} s_{i,j} = 1$$

El último paso es multiplicar esta matriz por V:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,n} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,0} & s_{m,1} & \cdots & s_{m,n} \end{pmatrix} \begin{pmatrix} - & \vec{v}_0 & - \\ - & \vec{v}_1 & - \\ - & \vdots & - \\ - & \vec{v}_n & - \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} s_{0,i}\vec{v}_i \\ \sum_{i=0}^{n} s_{1,i}\vec{v}_i \\ \vdots \\ \sum_{i=0}^{n} s_{m,i}\vec{v}_i \end{pmatrix} \tag{3.10}$$

Donde d_k representa las dimensiones de los vectores Q, K y V forma parte del proceso de escalado. Nos damos cuenta de que eliminando el escalado y sustituyendo Q, K y V por X obtenemos el mismo mecanismo de self-attention que hemos estudiado anteriormente.

$$S = AV = softmax(XX^t)X$$

Observamos que el mecanismo de atención nos genera una serie de promedios ponderados de las filas de V que depende de las consultas y claves de entrada. Observamos que en este mecanismo no existen los parámetros aprendibles, pues está completamente compuesto por operaciones de matrices y vectores.

3.2.4. Multihead attention.

Este mecanismo toma inspiración en el uso de múltiples filtros en una red convolucional para mejorar la capacidad de representación de un conjunto de datos. En nuestro caso, en lugar de emplear un solo mecanismo de atención, vamos a practicarle muchos para posteriormente concatenar el resultado. Esto se traduce en repetir un número h de veces (heads o cabezas) el mecanismo de SDPA anteriormente descrito (Ver Figura 11).

$$\textit{MultiHead}(Q, K, V) = \textit{Concat}(\textit{head}_1, ..., \textit{head}_h)W^O \tag{3.11}$$

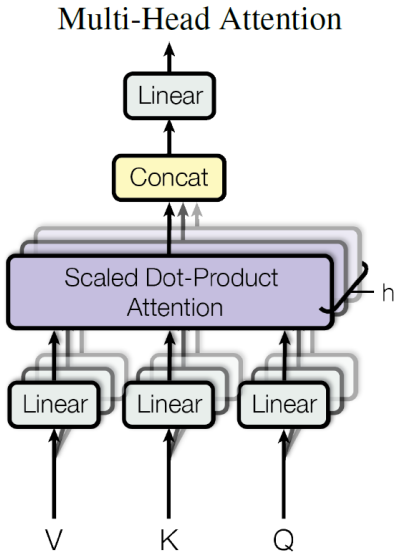

Figura 11: Esquema del Multihead Attention.[67]
. Fuente *Attention is all you need*

Cada $head_i$ es el resultado de ejecutar *Scaled dot product attention* en el i-ésimo conjunto de consultas, claves y valores transformados:

$$head_i = Attention(QW_q^i, KW_k^i, VW_v^i) \tag{3.12}$$

Aquí $Q \in \mathbb{R}^{m \times d_{model}}$, $K \in \mathbb{R}^{n \times d_{model}}$, y $V \in \mathbb{R}^{n \times d_{model}}$. Además, dado un hiperparámetro h que indica el número de cabezas de atención: $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, y $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

Vamos a hacer una descripción dimensional rápida para asegurarnos que la multiplicación de matrices es correcta. Primero, cada matriz head$_i$ tendrá el mismo número de filas que QW_i^Q y el mismo número de columnas que VW_i^V. Dado que $QW_i^Q \in \mathbb{R}^{m \times d_k}$ y $VW_i^V \in \mathbb{R}^{n \times d_v}$, esto significa que head$_i \in \mathbb{R}^{m \times d_v}$. Cuando concatenamos h de estas matrices juntas, tenemos una matriz en $\mathbb{R}^{m \times hd_v}$. Al multiplicarla por W^O, obtenemos una matriz en $R^{m \times d_{model}}$. Esto tiene sentido: comenzamos con m consultas en Q, y terminamos con m respuestas en la salida del operador $MultiHead$.

Existe una transformación lineal diferente para cada cálculo de la $head_i$ para las matrices de K, Q y V. Es aquí donde entra en juego el aprendizaje por entrenamiento. Las cabezas y sus matrices de pesos se comportan entonces de manera similar a los núcleos de las

redes convolucionales. Estos comienzan con la misma entrada a prtir de la cual se generan diferentes representaciones que concentran aspectos específicos de la misma (ver Figura 11).

3.3. GPT: Generative Pre-trained Transformer

El Generative Pre-trained Transformer (GPT) es una arquitectura de modelo de lenguaje basada en transformers, que ha demostrado un rendimiento sobresaliente en diversas tareas de NLP. GPT y sus sucesores, como GPT-2 [71], GPT-3 [7] y el reciente GPT-4, han sido desarrollados por OpenAI y han revolucionado el campo de NLP debido a su capacidad para generar texto coherente y de alta calidad.

3.3.1. Generación de texto

GPT es un modelo generativo, lo que significa que puede generar texto de manera autónoma al predecir la siguiente palabra en una secuencia dada. La generación de texto se realiza utilizando muestreo o búsqueda de haz, donde el modelo selecciona la siguiente palabra en función de las probabilidades de las palabras candidatas.

La función softmax, que hemos visto que también se usa en el Pre-Entrenamiento, es usada de nuevo las puntuaciones de las palabras candidatas en una distribución de probabilidad. Esta se aplica a la salida del modelo para cada posición de la secuencia para producir una distribución de probabilidad sobre las palabras en el vocabulario.

Dado un contexto anterior formado por una secuencia de palabras hasta el instante $t-1$, $(C_1, C_2, ..., C_{t-1})$, el objetivo es generar la siguiente palabra W_t. Para ello, el modelo estima la probabilidad condicional de la siguiente palabra dado el contexto anterior utilizando el mecanismo de atención y la función softmax. Esto se puede expresar como:

$$P(W_t|C_1, C_2, ..., C_{t-1}) = \text{softmax}(f(W_{t-1}, C_{t-1})) \tag{3.13}$$

donde f es una función de transformación aplicada a la representación del contexto anterior y la palabra generada previamente (W_{t-1}).

En cada paso de generación de palabras, se selecciona una palabra del vocabulario de acuerdo con la distribución de probabilidad estimada. Esta se realiza mediante el muestreo estocástico, donde se elige la siguiente palabra W_t con una probabilidad proporcional a su probabilidad condicional estimada.

3.4. Escala de los modelos de lenguaje

Los modelos grandes de lenguaje son pre-entrenados en extensos conjuntos de datos textuales. Algunos de los conjuntos de datos más comúnmente utilizados incluyen Common Crawl [5], The Pile [20], MassiveText [68], Wikipedia [15] y GitHub. Estos conjuntos de datos alcanzan tamaños de hasta 10 billones de palabras.

Se estima que el volumen de datos lingüísticos de alta calidad oscila entre los 4,6 y los 17 billones de palabras, lo que está dentro del orden de magnitud para los conjuntos de datos textuales más grandes.

En términos de Leyes de Escala, un LLM se puede caracterizar por cuatro parámetros principales: tamaño del modelo, tamaño del conjunto de datos de entrenamiento, costo de entrenamiento y rendimiento tras el entrenamiento. Cada una de estas cuatro variables puede ser definida con precisión en términos de un número real, y empíricamente se ha encontrado que estas están relacionadas por simples leyes estadísticas, conocidas como "Leyes de Escala".

Una particular Ley de Escala denominada **Escala Chinchilla** fue desarrollada por el equipo de investigación de Deep Mind. Esta se efectúa sobre LLMs entrenados de manera autorregresiva para una época con un esquema de tasa de aprendizaje logarítmico. La Escala Chinchilla propone que:

$$\begin{cases} C = C_0 N D \\ L = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + L_0 \end{cases} \tag{3.14}$$

donde C es el costo de entrenar en FLOPs. N es el número de parámetros. D es el número de tokens. L es la pérdida promedio de probabilidad logarítmica negativa por token (nats /token). $C_0 = 6$ es un parámetro estadístico. El resto de parámetros son: $\alpha = 0{,}34,\ \beta =$

$0{,}28,\ A = 406{,}4,\ B = 410{,}7,\ L_0 = 1{,}69$

Para $C_0 = 6$, se requieren 6 FLOP (operaciones de coma flotante) por cada parámetro para entrenar un token. Cabe destacar que el costo del entrenamiento supera significativamente al costo de la inferencia, donde se requieren entre 1 y 2 FLOP por cada parámetro para realizar la inferencia en un token.

3.4.1. Habilidades emergentes

Las habilidades emergentes en el contexto de los LLM son capacidades notables que estos modelos desarrollan durante su entrenamiento. Estas habilidades no son programadas explícitamente, sino que **surgen** de la interacción del modelo con los grandes volúmenes de datos de texto con los que se entrena.

A menudo, se puede extrapolar el rendimiento de modelos grandes en varias tareas a partir del rendimiento de modelos más pequeños similares. Sin embargo, en ciertas ocasiones, los modelos grandes experimentan un *cambio de fase discontinuo* donde adquieren de repente habilidades significativas que no se observan en modelos más pequeños. Estas se denominan habilidades emergentes y han sido objeto de un estudio considerable. De hecho, en algunos casos, estas habilidades solo se descubren después de que el LLM se haya implementado públicamente.

Se han identificado cientos de habilidades emergentes (Ver Figura 12). Entre los ejemplos se incluyen la aritmética de varios pasos, pasar exámenes de nivel universitario, identificar el significado previsto de una palabra, mostrar indicios de cadenas de pensamiento, decodificar el Alfabeto Fonético Internacional, descifrar las letras de una palabra, identificar contenido ofensivo en párrafos de Hinglish (una combinación de hindi e inglés), y generar un equivalente en inglés similar a los proverbios en kiswahili. Demostraciones prácticas de estas habilidades emergentes incluyen el modelo GPT-4 de OpenAI, que ha demostrado una notable capacidad para generar texto coherente y relevante contextualmente [7]. Asimismo, el modelo T5 de Google ha demostrado habilidades emergentes significativas en una variedad de tareas de procesamiento de lenguaje natural [53].

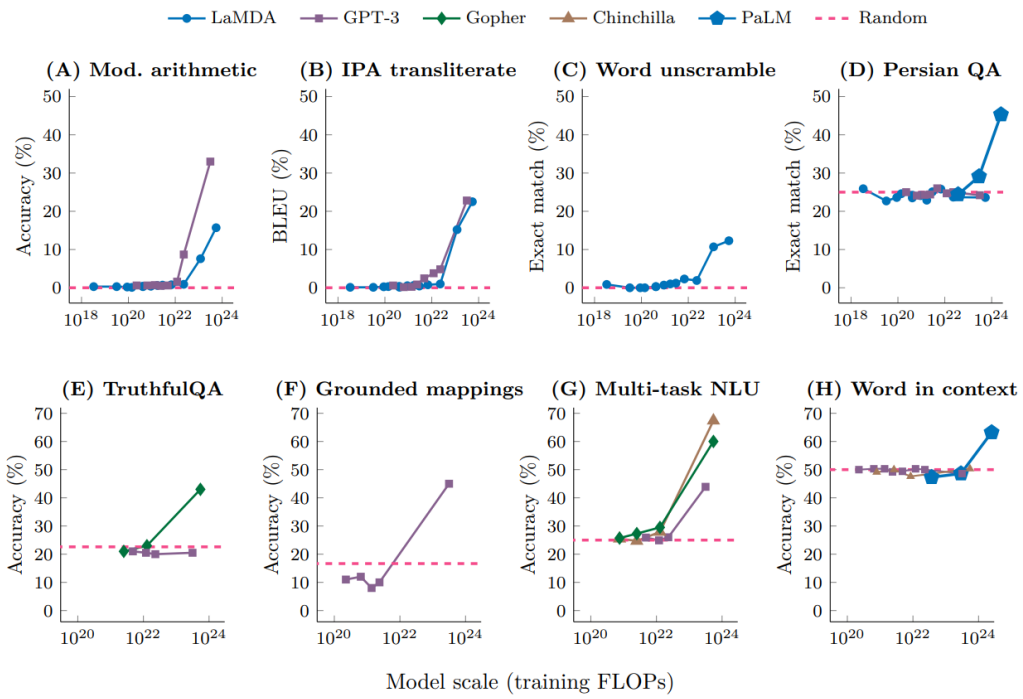

Figura 12: Habilidades emergentes. Fuente: Wikipedia

Alucinación

El fenómeno de alucinación en los LLM, se refiere a la situación en la que estos modelos generan información que no está presente en los datos de entrada o en su entrenamiento. En otras palabras, el modelo alucina detalles o hechos que parecen plausibles pero que no se basan en datos reales o precisos.

3.4.2. GPT

GPT[52] fue el primer modelo en la serie GPT y se basó en la arquitectura del Transformer[67]. Introdujo el concepto de pre-entrenamiento y ajuste fino en un gran corpus de texto para lograr un rendimiento superior en tareas de NLP. GPT consta de 117 millones de parámetros y, aunque demostró resultados prometedores, todavía tenía limitaciones en la generación de texto coherente y de alta calidad.

3.4.3. GPT-2

GPT-2 [71] es la segunda generación de la arquitectura GPT y cuenta con 1.5 mil millones de parámetros, lo que lo convierte en un modelo mucho más grande que su predecesor. GPT-2 mostró mejoras significativas en la generación de texto y demostró un rendimiento sobresaliente en una variedad de tareas de NLP. Sin embargo, también se destacó el potencial de uso indebido de la tecnología, ya que su capacidad para generar texto coherente y realista podría utilizarse para fines maliciosos, como la creación de noticias falsas.

3.4.4. GPT-3

GPT-3 [7] es la última generación de la serie GPT y es el modelo de lenguaje más grande hasta la fecha, con 175 mil millones de parámetros. Ha demostrado un rendimiento aún más impresionante en tareas de NLP y ha mostrado la capacidad de aprender de pocas muestras, lo que significa que puede adaptarse a nuevas tareas con solo unos pocos ejemplos de entrenamiento. GPT-3 ha sido utilizado en una amplia gama de aplicaciones prácticas, desde la generación de texto hasta el razonamiento y la comprensión del lenguaje natural.

3.4.5. GPT-4

Generative Pre-trained Transformer 4 (GPT-4), una invención de OpenAI, es un modelo de lenguaje avanzado que se presentó al público el 14 de marzo de 2023. Está accesible a través de la API de OpenAI y para aquellos que utilizan ChatGPT Plus. Tiene 1,76 billones de parámetros.

OpenAI lanzó dos variantes de GPT-4 con una longitud de contexto de 8192 y 32768 tokens, una notable mejora en comparación con GPT-3.5 y GPT-3 que estaban limitados a 4096 y 2048 tokens respectivamente. A diferencia de su antecesor, GPT-4 tiene la habilidad de procesar tanto imágenes como texto como entradas lo que lo convierte en una inteligencia multimodal.

En la Figura 13 podemos observar una comparativa de los diferentes modelos en cuanto al desempeño en diferentes tareas.

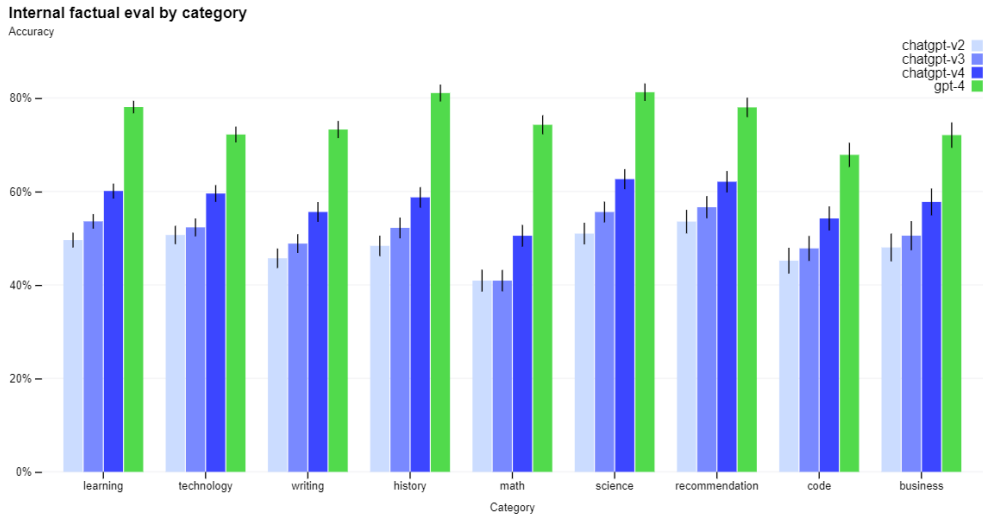

Figura 13: Comparativa de modelos de Open AI. Fuente: Open AI

3.4.6. Retos y consideraciones éticas

Con el crecimiento y la mejora de los modelos de lenguaje como GPT, GPT-2, GPT-3 y GPT-4, también surgen preocupaciones éticas y desafíos en términos de sesgo, privacidad, el uso indebido y el riesgo existencial de la inteligencia artificial general. Estas preocupaciones deben abordarse adecuadamente en la investigación y el desarrollo de futuros modelos de lenguaje.

Sesgo

Los modelos de lenguaje aprenden de grandes corpus de texto, que pueden contener sesgos implícitos y explícitos presentes en los datos de entrenamiento. Es importante investigar y desarrollar métodos para reducir el sesgo en los modelos de lenguaje y garantizar que no perpetúen estereotipos ni discriminación.

Privacidad

Los grandes modelos de lenguaje como GPT-4, aunque muy útiles en muchos contextos, plantean ciertas preocupaciones en términos de privacidad. Aquí hay algunos puntos clave a considerar:

Extracción de datos sensibles: Durante el entrenamiento, estos modelos se alimentan

con enormes cantidades de texto, que pueden incluir información potencialmente sensible o privada. Aunque los datos utilizados suelen ser anónimos y de dominio público, si un modelo es suficientemente grande y se entrena en datos suficientemente detallados, existe el riesgo teórico de que podría aprender a generar información que debería ser privada.

Inferencias no deseadas: Los modelos de lenguaje también pueden hacer inferencias sobre los usuarios basándose en los datos que estos proporcionan. Por ejemplo, si un usuario interactúa con el modelo de una manera que revela detalles personales, esos detalles podrían ser utilizados (inconscientemente) por el modelo para generar respuestas que reflejen o se basen en esa información.

Registros de interacciones: Además, las interacciones con estos modelos a menudo se registran y utilizan para mejorar el rendimiento del modelo. Esto puede plantear problemas de privacidad si los registros contienen información sensible y no se gestionan adecuadamente.

Manipulación y abuso potencial: Los modelos de lenguaje son herramientas poderosas que pueden ser utilizadas para una variedad de propósitos, no todos los cuales son benignos. Por ejemplo, podrían ser utilizados para generar desinformación a gran escala, para el acoso automatizado, para la creación de deepfakes escritos y más.

Riesgo existencial de la Inteligencia Artificial General

Los modelos de lenguaje a gran escala como GPT-4 son avances significativos hacia la inteligencia artificial más potente y capaz, lo que nos acerca a la idea de la inteligencia artificial general (AGI, por sus siglas en inglés), es decir, una inteligencia artificial que puede realizar cualquier tarea intelectual que un humano puede hacer. Sin embargo, este avance también lleva consigo un conjunto de riesgos existenciales que deben ser considerados cuidadosamente.

Mal uso de la tecnología: Un riesgo claro es el mal uso de una AGI por actores malintencionados. La AGI podría ser utilizada para causar daño a gran escala, ya sea por accidente o de manera intencional, ya que podría tener la capacidad de influir significativamente en el mundo digital y físico.

Pérdida de control: También existe el riesgo de que la AGI se vuelva incontrolable, es decir, que una vez desencadenada, podría ser imposible de detener o controlar. Una AGI podría seguir optimizando su función de objetivo incluso cuando eso entra en conflicto con los intereses humanos, un escenario conocido como el problema de alineación de valores.

Carrera de desarrollo descontrolada: Otro riesgo potencial es la posibilidad de una carrera de desarrollo descontrolada, donde diferentes organizaciones o países se apresuran a construir AGI sin tomar las precauciones de seguridad necesarias. Esta competencia podría incrementar la probabilidad de que se construya una AGI mal alineada o que pueda ser fácilmente mal utilizada.

Inequidad social: Además, la creación de una AGI podría amplificar las desigualdades existentes en la sociedad. Por ejemplo, si sólo un pequeño número de organizaciones o países tienen acceso a la AGI, podrían utilizarla para consolidar el poder y el control de manera desproporcionada.

3.5. Comparación con otros modelos de lenguaje populares: BERT y T5

A lo largo de los últimos años, han surgido varios modelos de lenguaje populares y exitosos en el campo del procesamiento del lenguaje natural. En esta sección, se comparará GPT-4 con otros modelos destacados, como BERT y T5, en términos de arquitectura, enfoque de entrenamiento y rendimiento en tareas de NLP.

3.5.1. BERT (Bidirectional Encoder Representations from Transformers)

BERT es un modelo de lenguaje basado en transformers desarrollado por Google AI Language [15]. A diferencia de GPT, BERT se entrena de manera bidireccional, lo que significa que considera tanto el contexto anterior como el posterior al predecir una palabra. Esto permite que BERT capture mejor el contexto en ambos lados de una palabra, lo que

puede mejorar el rendimiento en tareas como el análisis de sentimiento y la respuesta a preguntas.

3.5.2. Entrenamiento. GPT vs BERT

GPT es un modelo generativo unidireccional que se entrena para predecir la siguiente palabra en una secuencia, dada una secuencia de palabras anteriores.

La probabilidad de una secuencia de palabras $w_1, w_2, ..., w_n$ está dada por el producto de las probabilidades condicionales de cada palabra dada sus anteriores:

$$P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_1, ..., w_{i-1}) \tag{3.15}$$

En el entrenamiento, se utiliza la versión softmax de esta probabilidad, aplicada a los vectores de salida del modelo para la palabra i:

$$P(w_i|w_1, ..., w_{i-1}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.16}$$

Aquí z_i es la puntuación del modelo para la palabra i dada las palabras $1, ..., i-1$, K es el tamaño del vocabulario.

La idea es que el modelo aprende a maximizar estas probabilidades para las secuencias de palabras en el conjunto de entrenamiento.

Por otro lado, BERT es el que utiliza la técnica de enmascaramiento de lenguaje durante su fase de pre-entrenamiento. BERT oculta aleatoriamente algunas palabras en la entrada y entrena el modelo para predecir estas palabras enmascaradas basándose en el contexto antes y después de la palabra enmascarada.

3.5.3. T5 (Text-to-Text Transfer Transformer)

T5 es otro modelo de lenguaje basado en transformers desarrollado por Google Research [53]. A diferencia de GPT y BERT, T5 adopta un enfoque de "texto a texto", en el que todas las tareas de NLP se reformulan como problemas de generación de texto. Esto per-

mite que T5 sea más flexible en la adaptación a una amplia gama de tareas, incluida la traducción automática, el resumen y la clasificación de texto.

3.5.4. Comparación de rendimiento

GPT-4, BERT y T5 han demostrado un rendimiento impresionante en diversas tareas de NLP. GPT-4 puede tener ventajas en la generación de texto y la comprensión contextual debido a su enfoque unidireccional y la capacidad de aprender de pocas muestras. BERT, por otro lado, puede superar a GPT-4 en tareas que requieren una comprensión bidireccional del contexto, como el análisis de sentimiento y la respuesta a preguntas. T5 destaca por su flexibilidad y capacidad para adaptarse a una amplia variedad de tareas a través de su enfoque de "texto a texto".

3.5.5. Otras consideraciones

Al comparar GPT-4, BERT y T5, también es importante tener en cuenta factores como la eficiencia computacional, el tamaño del modelo y las preocupaciones éticas. Dado que los modelos de lenguaje a gran escala pueden requerir una gran cantidad de recursos computacionales, es crucial evaluar cómo estos modelos se desempeñan en términos de eficiencia y requisitos de hardware. Además, abordar las preocupaciones éticas, como el sesgo, la privacidad y el uso indebido, es vital en el diseño y la implementación de estos modelos.

GPT-4, BERT y T5 son modelos de lenguaje líderes en el campo del NLP, y cada uno tiene sus propias fortalezas y debilidades en términos de arquitectura, enfoque de entrenamiento y rendimiento en tareas específicas.

3.6. Aplicaciones y casos de uso

Los modelos de lenguaje a gran escala han encontrado una amplia variedad de aplicaciones y casos de uso en diferentes campos. Aunque han sido mencionadas en otros capítulos o como introducción de sección ahora exploraremos con un poco más de detalle cómo estos modelos han impactado en áreas como la generación de texto, la traducción

automática, la búsqueda de información, la atención médica y la educación.

Generación de texto

La generación de texto es una de las aplicaciones más destacadas de los modelos de lenguaje a gran escala. Estos modelos pueden generar texto coherente y gramaticalmente correcto en una amplia variedad de estilos y formatos, incluyendo noticias, resúmenes, historias y más [52]. La capacidad de generar texto de alta calidad ha llevado a la creación de herramientas de redacción asistida por IA, como OpenAI's Codex [7].

Traducción automática

La traducción automática es otra aplicación clave de los modelos de lenguaje a gran escala. Modelos como GPT-4 y T5 han demostrado un rendimiento excepcional en la traducción entre múltiples idiomas, superando a menudo a los sistemas de traducción basados en reglas y estadísticas [67].

Búsqueda de información

Los modelos de lenguaje a gran escala también han sido aplicados en la búsqueda de información, mejorando la relevancia y la precisión de los resultados de búsqueda en la web y en bases de datos especializadas [59]. Además, los modelos como BERT han sido utilizados para mejorar la comprensión de las consultas de los usuarios y para proporcionar respuestas más precisas y contextualizadas a las preguntas [15].

Atención médica

En el campo de la atención médica, los modelos de lenguaje a gran escala han demostrado ser útiles en tareas como el diagnóstico automatizado, la interpretación de registros médicos y la generación de informes clínicos [43]. También pueden ayudar a los profesionales médicos a mantenerse al tanto de los avances científicos al analizar y resumir grandes volúmenes de literatura médica

Educación

Los modelos de lenguaje a gran escala también han encontrado aplicaciones en la educación, como la tutoría personalizada y la evaluación automática de respuestas escritas. Los tutores basados en IA pueden proporcionar retroalimentación en tiempo real a los estudiantes y adaptarse a las necesidades individuales de aprendizaje [24]. Además, estos modelos pueden utilizarse para evaluar las respuestas de los estudiantes en pruebas y exámenes, lo que permite una evaluación más rápida y consistente en comparación con la calificación manual [16].

3.7. Comprender el funcionamiento de los modelos

Los modelos grandes de lenguaje son a menudo considerados como *cajas negras* debido a la dificultad para interpretar y entender cómo toman sus decisiones específicas.

La razón principal de esta opacidad radica en la naturaleza del aprendizaje profundo. Este funciona mediante la creación de representaciones de alto nivel a partir de los datos de entrada, a través de múltiples capas de transformaciones no lineales. Mientras que la primera capa puede aprender características simples, las capas sucesivas aprenden características cada vez más abstractas y complejas. Estas transformaciones sucesivas pueden ser difíciles de rastrear y entender, incluso cuando se tienen millones o incluso billones de parámetros que ajustar.

Esto puede ser ilustrado mediante la siguiente ecuación simplificada para un LLM:

$$h_t = f(W^{(1)}h_{t-1} + W^{(2)}x_t) \tag{3.17}$$

Donde h_t representa el estado oculto en el tiempo t, x_t es la entrada en el tiempo t, y $W^{(1)}$ y $W^{(2)}$ son matrices de pesos que el modelo aprende durante el entrenamiento. La función f es una función de activación no lineal. Aunque esta ecuación puede parecer simple, la interacción entre estos elementos puede resultar en comportamientos extremadamente complejos cuando se tiene en cuenta que un LLM puede tener cientos de capas y miles de millones de parámetros.[8]

El hecho de que estos modelos sean entrenados en conjuntos de datos masivos añade otra capa de complejidad. Los modelos aprenden a partir de los patrones y estructuras presentes en estos conjuntos de datos, pero no hay una forma sencilla de entender qué patrones específicos ha aprendido el modelo y cómo estos patrones influyen en su comportamiento.

3.7.1. Soluciones y enfoques futuros

Para enfrentar los desafíos y limitaciones mencionados, los investigadores y desarrolladores están explorando varias soluciones y enfoques. Algunas áreas clave de investigación incluyen:

- Desarrollo de técnicas de mitigación de sesgo en los modelos de lenguaje para reducir la propagación de prejuicios y estereotipos [3].
- Implementación de mecanismos de protección de la privacidad, como la diferenciación de datos, para prevenir la divulgación de información confidencial [46].
- Investigación de enfoques más eficientes en términos de energía y computación, como la destilación del conocimiento y la compresión de modelos, para reducir la carga computacional y mejorar la accesibilidad [36, 42].
- Utilización de técnicas de explicabilidad e interpretación de modelos para mejorar la transparencia y la comprensión de cómo los modelos de lenguaje a gran escala toman decisiones [54].

Capítulo 4

Análisis matemático de los LLM

El análisis matemático de los modelos de lenguaje, en particular aquellos basados en la arquitectura Transformer como GPT-4, puede proporcionar una comprensión más profunda de cómo estos modelos procesan y generan información lingüística. En esta sección, se examinarán las bases matemáticas detrás de los algoritmos clave que se utilizan para analizar los modelos de lenguaje y cómo estas matemáticas influyen en su rendimiento y comportamiento.

4.1. Estudio de la complejidad computacional

La complejidad computacional se refiere a la cantidad de recursos computacionales necesarios para entrenar, generar o inferir con estos modelos. Los modelos grandes de lenguaje como GPT-4 suelen requerir una gran cantidad de recursos computacionales, como poder de procesamiento, memoria y tiempo, tanto durante la fase de entrenamiento como durante la generación de texto o inferencia.

4.1.1. Complejidad en tiempo y espacio

El estudio de la complejidad en tiempo y espacio proporciona una medida cuantitativa de los recursos requeridos para ejecutar un algoritmo o modelo. En el caso de los modelos de lenguaje, la complejidad en tiempo se refiere a la cantidad de tiempo requerido para rea-

lizar operaciones de entrenamiento y predicción, mientras que la complejidad en espacio se refiere a la cantidad de memoria necesaria para almacenar el modelo y sus parámetros.

La complejidad computacional de estos modelos está influenciada por varios factores, entre ellos:

- **Tamaño del modelo**: Antes hemos mencionado que GPT-4 poseía 1,76 billones de parámetros. Básicamente, GPT-4 *condensa* un conjunto de ocho LLMs, cada uno entrenado de manera independiente con aproximadamente 200 mil millones de parámetros (comparable a GPT-3) Cuantos más parámetros tenga un modelo, mayor será su complejidad computacional [9].

- **Arquitectura del modelo**: La elección de la arquitectura del modelo también puede afectar su complejidad computacional. Los modelos basados en transformadores, como GPT-3, son conocidos por ser computacionalmente intensivos debido a las operaciones de atención que involucran grandes matrices de pesos.

- **Datos de entrenamiento**: GPT-4 utiliza un conjunto de datos de entrenamiento extenso y diverso, que consta de decenas de miles de millones de documentos. Cuantos más datos se utilicen para entrenar el modelo, mayor será la cantidad de cálculos necesarios para procesarlos y ajustar los parámetros del modelo.

- **Recursos disponibles**: La complejidad computacional también depende de los recursos disponibles, como la capacidad de procesamiento, la memoria y la eficiencia del software y las bibliotecas utilizadas para implementar el modelo.

La relación exacta entre el tiempo de ejecución y el número de parámetros en un modelo puede variar dependiendo de varios factores, como el hardware específico, el software y la eficiencia de la implementación del modelo. Sin embargo, a nivel muy general, podríamos decir que el tiempo de ejecución es directamente proporcional al número de parámetros del modelo.

4.1.2. FLOPs y parámetros

Los modelos de lenguaje modernos a menudo requieren una gran cantidad de recursos computacionales, lo que puede medirse en términos de **operaciones de coma flotante por segundo** (FLOPs) y el número de parámetros del modelo. Por ejemplo, GPT-3 cuenta con aproximadamente 175 mil millones de parámetros, lo que representa un incremento sustancial respecto a modelos anteriores como GPT-2, que posee 1.5 mil millones de parámetros [7].

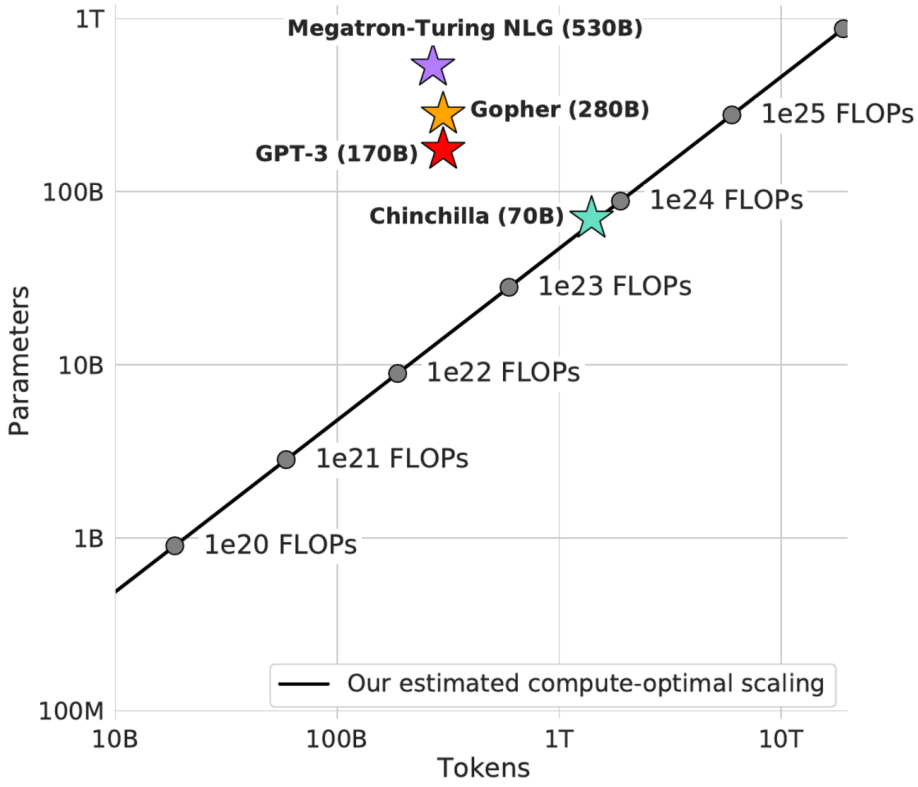

Figura 14: Comparativas de modelos en cuanto a FLOPS. Fuente @DotCSV

4.1.3. Eficiencia y optimización

Dada la complejidad computacional de los modelos grandes de lenguaje, es importante investigar enfoques para mejorar su eficiencia y optimización. Algunas técnicas incluyen la compresión de modelos mediante la eliminación de parámetros redundantes, la aplicación de la destilación del conocimiento para entrenar modelos más pequeños y eficientes [36], y el uso de técnicas de optimización en la implementación de hardware, como unidades

de procesamiento tensorial (TPUs) [32].

4.2. Análisis de la convergencia y la optimización

La convergencia en el entrenamiento de modelos de lenguaje a gran escala es un tema crucial para entender cómo y cuándo estos modelos alcanzan su rendimiento óptimo. La optimización es el proceso mediante el cual los parámetros de un modelo se ajustan iterativamente para minimizar la función de pérdida. En esta sección, discutiremos cómo la convergencia y la optimización influyen en el rendimiento de los modelos de lenguaje a gran escala y cómo se pueden mejorar.

4.2.1. Tasas de aprendizaje y adaptación

Las tasas de aprendizaje (*Learning rates*) son hiperparámetros que determinan qué tan rápido el modelo aprende a partir de los datos durante el entrenamiento. Forma parte de un algoritmo de optimización y permite adaptar el tamaño del paso de cada iteración durante la minimización de una función de pérdida. Un tamaño de paso demasiado grande puede hacer que el modelo oscile y no converja, mientras que un tamaño de paso demasiado pequeño puede resultar en un aprendizaje extremadamente lento y prolongado [4].

Durante el entrenamiento, los pesos del modelo se actualizan multiplicándolos por la tasa de aprendizaje actual:

$$w(t+1) = w(t) - \alpha(t) \cdot \nabla L(w(t)) \tag{4.1}$$

Donde $w(t)$ son los pesos del modelo en la iteración t, L es la función de pérdida y $\nabla L(w(t))$ es el gradiente de la función de pérdida con respecto a los pesos en la iteración t.

Otro enfoque consiste en los horarios de aprendizaje basados en pasos. Estos ajustan la tasa de aprendizaje siguiendo un conjunto predefinido de pasos. La fórmula utilizada para aplicar el decaimiento se define de la siguiente manera:

$$\eta_n = \eta_0 d^{\lfloor \frac{1+n}{r} \rfloor} \tag{4.2}$$

Donde la tasa de aprendizaje en la iteración n es denotada por η_n, η_0 representa la tasa de aprendizaje inicial, d indica cuánto debe cambiar la tasa de aprendizaje en cada caída (por ejemplo, 0.5 corresponde a una reducción a la mitad), y r corresponde a la tasa de caída, es decir, la frecuencia con la que se debe ajustar la tasa. La función de suelo ($\lfloor \ldots \rfloor$) reduce el valor de su entrada a 0 para todos los valores menores que 1.

Los algoritmos de optimización adaptativa, como Adam [33], ajustan dinámicamente la tasa de aprendizaje para cada parámetro del modelo según su propio historial de actualizaciones. Esto puede ayudar a mejorar la convergencia y el rendimiento del modelo, especialmente en el entrenamiento de modelos de lenguaje a gran escala.

4.2.2. Regularización

Entendemos por *regularización* a cualquier modificación que hacemos a un algoritmo de aprendizaje con la intención de reducir su error de generalización pero no su error de entrenamiento. La regularización es una de las preocupaciones centrales del campo del aprendizaje automático, rivalizando en importancia solo con la optimización.

La regularización se ha convertido en otra técnica importante en el entrenamiento de modelos de lenguaje para evitar el sobreajuste y mejorar la capacidad de generalización del modelo en datos no vistos. Normalmente, la regularización se produce imponiendo restricciones en los parámetros del modelo, lo que evita que este dependa demasiado de características ruidosas o irrelevantes en los datos de entrenamiento.

Un enfoque común para la regularización es agregar un término de penalización a la función de pérdida, como la norma $L1$ o $L2$ de los parámetros del modelo [65]. Esta penalización desalienta el crecimiento excesivo de los parámetros, lo que puede llevar a un modelo más simple y generalizable.

Supongamos que queremos entrenar un modelo de regresión polinomial de grado alto para ajustar nuestros datos. El modelo de regresión polinomial de grado n se define como:

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_n x^n \tag{4.3}$$

Para aplicar la regularización, vamos a utilizar el método de Ridge. La función de costo con regularización Ridge se define de la siguiente manera:

$$J(w) = MSE + \lambda \sum_{i=1}^{n} w_i^2 \tag{4.4}$$

Cuando $\lambda = 0$, no imponemos ninguna penalización, y un λ más grande obliga a los pesos a volverse más pequeños. Minimizar $J(w)$ resulta en una elección de pesos que hacen un equilibrio entre ajustar los datos de entrenamiento y ser pequeños. Esto nos da soluciones que tienen una pendiente más pequeña, o ponen peso en menos de las características.

Existen otros métodos como Lasso y Elastic Net que también se utilizan en la práctica. Cada uno tiene sus propias características y se selecciona según las necesidades del problema en particular.

La regularización Lasso, también conocida como regularización L1, se puede definir utilizando la función objetivo con término de regularización añadido:

$$\min_{\mathbf{w}} \left(\frac{1}{2N} \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \sum_{j=1}^{M} |\mathbf{w}_j| \right) \tag{4.5}$$

Donde $\mathbf{w}$ es el vector de pesos que queremos aprender, N es el número de ejemplos de entrenamiento, M es el número de características o dimensiones de los datos de entrada, y_i es el valor objetivo para el i-ésimo ejemplo de entrenamiento, $\mathbf{x}_i$ es el vector de características del i-ésimo ejemplo de entrenamiento, y finalmente λ es el hiperparámetro de regularización que controla la fuerza de la regularización Lasso.

Otra técnica de regularización ampliamente utilizada en el entrenamiento de redes neuronales es el *Dropout* [60]. Durante el entrenamiento, el dropout apaga aleatoriamente un subconjunto de unidades en la red, lo que evita la coadaptación de las características y mejora la robustez del modelo.

Dropout proporciona un método de regularización potente pero computacionalmente económico para una amplia familia de modelos. A primera vista, puede considerarse como un método para hacer práctico el bagging para conjuntos de muchas redes neuronales gran-

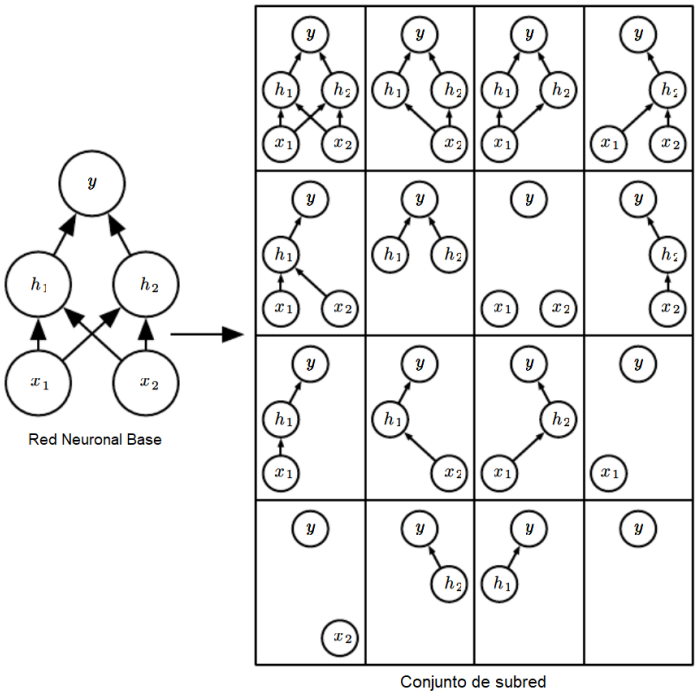

Figura 15: Técnica Dropout. Imagen obtenida del libro "Deep Learning"[22]

des. El bagging implica entrenar y evaluar múltiples modelos en cada ejemplo de prueba. Esto parece impráctico cuando cada modelo es una gran red neuronal, ya que entrenar y evaluar tales redes es costoso en términos de tiempo de ejecución y memoria. Es común usar conjuntos de 5 a 10 redes neuronales.

Específicamente, dropout entrena el conjunto que consiste en todas las subredes que se pueden formar eliminando unidades no de salida de una red base subyacente, como se ilustra en la Fig. 15. En la mayoría de las redes neuronales modernas, basadas en una serie de transformaciones afines y no linealidades, podemos eliminar efectivamente una unidad de una red multiplicando su valor de salida por 0. Este procedimiento requiere algunas pequeñas modificaciones para modelos como las redes de función de base radial, que toman la diferencia entre el estado de la unidad y algún valor de referencia. Aquí, presentamos el algoritmo de dropout en términos de multiplicación por 0 para simplificar, pero puede modificarse trivialmente para trabajar con otras operaciones que eliminan una unidad de la red.

El entrenamiento y la optimización de modelos de lenguaje basados en redes neuronales

requieren un equilibrio cuidadoso entre la minimización de la función de pérdida y la prevención del sobreajuste a través de la regularización.

4.2.3. Ajuste fino

El ajuste fino es una técnica que permite mejorar el rendimiento y la convergencia de un modelo preentrenado en una tarea específica. Después del entrenamiento inicial, se ajusta el modelo con una tasa de aprendizaje más baja en un conjunto de datos específico de la tarea [55]. Esto permite que el modelo se especialice en una tarea específica sin perder la información generalizada adquirida durante el entrenamiento previo.

Ess una técnica común en el aprendizaje profundo que implica tomar un modelo pre-entrenado en una tarea (generalmente en un gran conjunto de datos como ImageNet para la visión por computadora o Wikipedia para el procesamiento del lenguaje natural) y luego seguir entrenándolo (es decir, ajustándolo) en una tarea específica de interés. Esta técnica puede mejorar la convergencia y el rendimiento, especialmente cuando la tarea específica tiene un conjunto de datos relativamente pequeño.

Supongamos que $L_S(y, y_{pred})$ es la función de pérdida en la tarea específica, θ son los parámetros del modelo, μ es la tasa de aprendizaje y t es el número de iteraciones. A continuación, se proporciona un esquema de cómo se realiza el ajuste fino con un algoritmo de optimización de descenso de gradiente estocástico (SGD):

1. Paso de inicialización: Inicializamos los parámetros del modelo con los parámetros aprendidos en la tarea pre-entrenada, es decir, $\theta(t = 0) = \theta_{pretrained}$.

2. Paso de actualización: En cada iteración de entrenamiento, actualizamos los parámetros θ para minimizar la pérdida en la tarea específica. En el descenso de gradiente estocástico, esto se realiza calculando el gradiente de la función de pérdida con respecto a los parámetros y luego actualizando los parámetros en la dirección opuesta al gradiente:

$$\theta(t + 1) = \theta(t) - \mu * \nabla\theta L_S(y, y_{pred}; \theta(t)) \tag{4.6}$$

3. Paso de evaluación: Evaluamos el rendimiento del modelo ajustado en la tarea específica utilizando un conjunto de validación o de prueba. El ajuste fino puede mejorar la convergencia y el rendimiento de varias maneras:

    - **Inicialización inteligente**: Al inicializar los parámetros con un modelo pre-entrenado, proporcionamos al modelo una $buena$ inicialización que puede estar cerca de una solución óptima para la tarea específica. Esto puede permitir una convergencia más rápida que la inicialización aleatoria.
    - **Regularización implícita**: Si mantenemos la tasa de aprendizaje μ pequeña durante el ajuste fino, el modelo tiende a permanecer cerca de la solución pre-entrenada. Esto actúa como una forma de regularización, restringiendo el espacio de hipótesis a las funciones que son similares a la función aprendida en la tarea pre-entrenada.
    - **Aprovechamiento de los datos**: Si la tarea pre-entrenada y la tarea específica son similares, entonces el modelo puede aprovechar los datos de la tarea pre-entrenada y aprender características útiles que pueden mejorar el rendimiento en la tarea específica.

Por supuesto, estas son generalizaciones y los detalles específicos pueden variar dependiendo de la tarea, el modelo y el algoritmo de optimización.

4.2.4. Estrategias para mejorar la convergencia

Investigaciones recientes han explorado diversas estrategias para mejorar la convergencia en el entrenamiento de modelos de lenguaje a gran escala, como la inicialización de pesos [21], la normalización por lotes [62] y la normalización por capas [1]. Estas técnicas pueden ayudar a estabilizar el proceso de entrenamiento, mejorar la velocidad de convergencia y, en última instancia, resultar en un mejor rendimiento del modelo.

Inicialización de pesos

En redes neuronales, la inicialización de los pesos es crucial para garantizar una convergencia eficiente durante el entrenamiento. Glorot y Bengio [21] propusieron una iniciali-

zación que ha sido muy popular, a menudo referida como inicialización de Glorot o Xavier. Para una capa con n_{in} entradas y n_{out} salidas, los pesos se inicializan aleatoriamente de una distribución uniforme en el rango $[-sqrt(6/(n_{in}+n_{out})), sqrt(6/(n_{in}+n_{out}))]$.

$$w_{ij} \sim U\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right) \tag{4.7}$$

La idea detrás de esto es proporcionar una escala óptima para los pesos en cada capa. La presencia de $\sqrt{\frac{6}{n_{in}+n_{out}}}$ en la ecuación garantiza que los pesos no sean demasiado grandes ni demasiado pequeños, lo que ayuda a evitar problemas como la desaparición o la explosión de gradientes.

Normalización por lotes

La normalización por lotes[62] es una técnica que acelera el entrenamiento y mejora la generalización al normalizar las activaciones de cada capa. Para un mini-lote de m ejemplos con dimensiones d, la normalización por lotes calcula la media y la varianza para cada dimensión y luego normaliza las activaciones:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \quad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_B)^2 \tag{4.8}$$

Entonces, las activaciones son normalizadas como:

$$\hat{x}_i = \frac{x_i-\mu_B}{\sqrt{\sigma_B^2+\epsilon}} \tag{4.9}$$

Donde ϵ es un pequeño número para la estabilidad numérica. Finalmente, se aplica una transformación afín:

$$y_i = \gamma\hat{x}_i + \beta \tag{4.10}$$

Donde γ y β son parámetros aprendidos que permiten al modelo recuperar la escala y el desplazamiento original si es necesario.

Normalización por capas

La normalización por capas[1] es similar a la normalización por lotes, pero en lugar de normalizar las activaciones en cada dimensión sobre un mini-lote, normaliza las activaciones para cada ejemplo sobre todas las dimensiones. Si x es el vector de activaciones para un ejemplo con d dimensiones, entonces la normalización por capas se calcula de la siguiente manera:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i \quad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2 \tag{4.11}$$

Las activaciones son normalizadas como:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{4.12}$$

Y se aplica una transformación afín:

$$y_i = \gamma \hat{x}_i + \beta \tag{4.13}$$

Donde γ y β son parámetros aprendidos. La normalización por capas puede ser beneficiosa en modelos recurrentes donde el tamaño del lote puede variar.

4.2.5. Evaluación de la convergencia y la optimización

Para evaluar la convergencia y la optimización en el entrenamiento de LLM, es común usar métricas específicas como la perplejidad para [31] y el F1-score para la clasificación de texto [47]. Los investigadores también pueden monitorear el comportamiento de la función de pérdida en conjuntos de datos de entrenamiento y validación para evaluar el progreso y la convergencia del modelo.

Perplejidad

La perplejidad es una medida utilizada para evaluar la calidad y la capacidad predictiva de un modelo de lenguaje. Cuanto menor sea la perplejidad, mejor será el rendimiento del modelo en términos de predicción y comprensión del texto.

En esencia, la perplejidad es una medida de cuánta "sorpresa.experimenta un modelo de lenguaje al predecir la siguiente palabra en un texto. Se basa en la idea de que un buen modelo de lenguaje debe ser capaz de asignar una probabilidad alta a las secuencias de palabras que son más probables en un conjunto de datos de entrenamiento. La perplejidad para un modelo de lenguaje en una secuencia de palabras $w_1, w_2, ..., w_N$ se puede expresar como:

$$\text{Perplejidad}(w_1, w_2, ..., w_N) = \left(\frac{1}{P(w_1, w_2, ..., w_N)} \right)^{\frac{1}{N}} \tag{4.14}$$

donde $P(w_1, w_2, ..., w_N)$ es la probabilidad de la secuencia de palabras según el modelo de lenguaje.

F1-Score

El F1-Score es una medida que combina precisión y recall para proporcionar una única métrica de rendimiento. Se utiliza comúnmente en tareas de clasificación binaria y clasificación multietiqueta, y es especialmente útil en situaciones en las que las clases están desbalanceadas.

-Precisión (Precision): Esta es la proporción de identificaciones positivas correctas (verdaderos positivos, tp) entre el total de identificaciones positivas (verdaderos positivos + falsos positivos, fp).

$$\text{Precision} = \frac{tp}{tp + fp}$$

-Recall (Recuerdo): Esta es la proporción de identificaciones positivas correctas entre el total de ejemplos positivos en los datos (verdaderos positivos + falsos negativos, fn).

$$\text{Recall} = \frac{tp}{tp + fn}$$

-F1-Score: Esta es la media armónica de precisión y recall. A diferencia de la media aritmética, la media armónica castiga valores extremos, por lo que el F1-Score será alto solo si tanto la precisión como el recall son altos.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Por lo tanto, el F1-Score puede verse como una forma de equilibrar la precisión y el recall, y proporciona una única métrica de rendimiento que puede ser útil para comparar diferentes modelos. Es especialmente útil en situaciones donde tanto los falsos positivos como los falsos negativos son importantes.

Un valor de F1-Score cercano a 1 indica un rendimiento excelente del modelo, mientras que un valor cercano a 0 indica un rendimiento deficiente.

Monitoreo de la función de pérdida

Recordamos que la función de pérdida mide la discrepancia entre la salida del modelo y la salida deseada. Durante el entrenamiento de un modelo, se monitorea la función de pérdida en los conjuntos de datos de entrenamiento y validación para evaluar el progreso y la convergencia del modelo. Como vimos anteriormente la entropía cruzada ahora pondremos de ejemplo el modelo de regresión, con la pérdida cuadrada media (MSE) comúnmente utilizada como función de pérdida:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))^2 \tag{4.15}$$

donde θ son los parámetros del modelo, N es el número de instancias en el conjunto de datos, y_i es la salida deseada para la i-ésima instancia, y $f_\theta(x_i)$ es la salida del modelo para la i-ésima instancia.

Es muy común monitorear el comportamiento de la función de pérdida tanto en el conjunto de datos de entrenamiento como en el de validación. La razón principal de esto es para evaluar si el modelo está aprendiendo y cómo está convergiendo durante el entrenamiento. Un comportamiento típico que se puede esperar es que la función de pérdida disminuya a medida que el modelo aprende de los datos.

Además, comparar la función de pérdida en los conjuntos de datos de entrenamiento y validación puede proporcionar información sobre el sobreajuste. Si la función de pérdida en el conjunto de datos de entrenamiento es mucho más baja que en el conjunto de datos de validación, podría ser una indicación de que el modelo está sobreajustando los datos de entrenamiento y no generaliza bien a datos no vistos.

En términos matemáticos, podríamos tener dos funciones de pérdida, una para el conjunto de entrenamiento $L_{train(\theta)}$ y otra para el conjunto de validación $L_{val(\theta)}$. Durante el proceso de entrenamiento, queremos minimizar ambas funciones de pérdida ajustando los parámetros del modelo θ. Esto se hace generalmente a través de un algoritmo de optimización como el descenso de gradiente:

$$\theta := \theta - \alpha \nabla_\theta L_{\text{train}}(\theta)$$

donde α es la tasa de aprendizaje y $\nabla_\theta L_{train(\theta)}$ es el gradiente de la función de pérdida de entrenamiento con respecto a θ. Podríamos tener un registro de $L_{train(\theta)}$ y $L_{val(\theta)}$ en cada época (o después de un número fijo de actualizaciones de parámetros) para monitorear el progreso y la convergencia del modelo.

Finalmente, es importante destacar que estas métricas y la función de pérdida dependen de la tarea específica y del modelo que se esté utilizando. Así, en el entrenamiento de modelos de lenguaje o en la clasificación de texto, podríamos tener funciones de pérdida y métricas diferentes.

4.3. Estudio de la generalización y la capacidad de aprendizaje

La capacidad de un modelo de lenguaje para generalizar y aprender eficazmente a partir de un conjunto de datos dado es un aspecto crucial para evaluar su rendimiento y aplicabilidad. Esta sección analiza el fenómeno de la generalización y la capacidad de aprendizaje en modelos de lenguaje a gran escala, destacando los factores clave que influyen en estos aspectos y discutiendo posibles vías para mejorarlos.

4.3.1. Balance entre sesgo y varianza

El sesgo y la varianza[30] son dos aspectos fundamentales del error de generalización de un modelo de aprendizaje automático.

- **Sesgo (Bias)**: Se refiere a la simplificación de suposiciones realizadas por un mode-

lo para facilitar el aprendizaje de los datos de entrenamiento. Un alto sesgo puede hacer que un modelo pierda las relaciones relevantes entre las características y la variable objetivo (subajuste), lo que resulta en un rendimiento de aprendizaje deficiente.

- **Varianza**: Se refiere a la cantidad que la predicción del modelo cambiaría si se utilizara un conjunto de entrenamiento diferente. Un modelo con alta varianza ha aprendido muy bien los datos de entrenamiento (sobreajuste), hasta el punto de ser sensible a las fluctuaciones en el conjunto de entrenamiento. Esto puede hacer que el modelo realice mal en datos no vistos.

La relación entre el sesgo, la varianza y el error total se puede representar mediante la siguiente ecuación, conocida como Descomposición de Bias-Varianza:

$$\text{Error Total} = \text{Bias}^2 + \text{Varianza} + \text{Ruido Irreducible} \tag{4.16}$$

Error Total: Es el error promedio que cometerá el modelo en los datos de prueba (o en general, en datos nuevos). **El Ruido Irreducible**: se refiere al error que no se puede reducir, independientemente de qué algoritmo usemos. Es causado por factores aleatorios o desconocidos en el conjunto de datos.

El equilibrio entre sesgo y varianza se trata de minimizar ambos tipos de errores para producir un modelo que generalice bien los datos no vistos. Este es un desafío fundamental en el aprendizaje automático. Si un modelo es demasiado simple, puede tener un alto sesgo y una baja varianza. Si un modelo es demasiado complejo, puede tener un bajo sesgo pero una alta varianza.

4.3.2. Análisis de la interferencia catastrófica

La interferencia catastrófica se refiere a la tendencia de un modelo de aprendizaje profundo a olvidar información previamente aprendida cuando se le entrena en nuevos datos. Este fenómeno es especialmente preocupante en el contexto de los modelos de lenguaje a gran escala, ya que tienen una gran cantidad de parámetros y se entrenan en vastos

conjuntos de datos [45].

Una de las formas más simples de modelar este fenómeno es considerar la actualización de los parámetros de un modelo durante el entrenamiento. En general, los parámetros de un modelo se actualizan en función del gradiente de la función de pérdida con respecto a los parámetros, es decir:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t; X, Y) \tag{4.17}$$

donde θ_t y θ_{t+1} son los parámetros del modelo en los tiempos t y $t+1$, respectivamente, η es la tasa de aprendizaje y $L(\theta_t; X, Y)$ es la función de pérdida evaluada en los parámetros actuales θ_t y un lote de datos X, Y.

Si consideramos dos conjuntos de datos diferentes, uno antiguo y uno nuevo, y suponemos que el modelo se entrena primero en el conjunto de datos antiguo y luego en el conjunto de datos nuevo, entonces existe el riesgo de que el modelo olvide la información aprendida del conjunto de datos antiguo debido a las actualizaciones de los parámetros basadas en el conjunto de datos nuevo. Esto puede expresarse formalmente de la siguiente manera:

$$L_{old}(\theta_{t+1}; X_{old}, Y_{old}) > L_{old}(\theta_t; X_{old}, Y_{old}) \tag{4.18}$$

Esto simplemente dice que la pérdida en el conjunto de datos antiguo puede aumentar después de una actualización de los parámetros basada en el conjunto de datos nuevo, indicando que el modelo ha olvidado parte de la información aprendida del conjunto de datos antiguo. Este es el fenómeno de interferencia catastrófica.

La mitigación de la interferencia catastrófica es un área de investigación activa en el aprendizaje profundo. Una estrategia comúnmente utilizada es la llamada rehearsal, en la que se retienen algunos datos antiguos y se mezclan con los nuevos datos durante el entrenamiento.

El rehearsal (ensayo), también conocido como memoria episódica o reproducción, implica mezclar los antiguos y los nuevos datos durante el entrenamiento.

Suponemos que tenemos un modelo con parámetros θ, una función de pérdida L, y un conjunto de datos de entrenamiento original $D_{old} = \{(x_i, y_i)\}_{i=1}^N$. Cuando se dispone de

nuevos datos de entrenamiento $D_{new} = \{(x'_j, y'_j)\}_{j=1}^{M}$, en lugar de entrenar el modelo solo en los nuevos datos, se mezclan los antiguos y nuevos datos. La actualización de los parámetros del modelo en cada iteración del entrenamiento se realiza de la siguiente manera:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \left[\frac{1}{M} \sum_{j=1}^{M} L(\theta_t; x'_j, y'_j) + \frac{1}{N} \sum_{i=1}^{N} L(\theta_t; x_i, y_i) \right] \tag{4.19}$$

Aquí, η es la tasa de aprendizaje, y ∇_θ denota el gradiente respecto a los parámetros del modelo θ. El primer término dentro del corchete es la pérdida promedio en los nuevos datos, y el segundo término es la pérdida promedio en los datos antiguos. En cada paso de actualización, el modelo busca minimizar la pérdida tanto en los nuevos como en los antiguos datos, lo que ayuda a mitigar la interferencia catastrófica.

4.3.3. Otras estrategias para mejorar la generalización

Existen una serie de estrategias que se pueden emplear para mejorar la generalización y la capacidad de aprendizaje en los modelos de lenguaje a gran escala, algunas de las cuales incluyen:

Aumento de datos

El objetivo del aumento de datos es generar más ejemplos de entrenamiento, lo que puede expresarse como una transformación T en los datos existentes X. Esto puede modelarse de la siguiente manera:

$$X' = T(X) \tag{4.20}$$

Donde X son los nuevos datos generados. El modelo luego se entrena en el conjunto de datos ampliado $X \cup X'$

Según Shorten y Khoshgoftaar, generar más ejemplos de entrenamiento mediante la transformación de los datos existentes puede mejorar la robustez y la capacidad de generalización del modelo[58].

Entrenamiento multi-tarea

En el entrenamiento de multitareas, el modelo se entrena para realizar varias tareas relacionadas al mismo tiempo. Supongamos que tenemos M tareas, cada una con su propia función de pérdida $L(\theta; X, Y)$. La función de pérdida total en el entrenamiento de multitareas es a menudo una combinación lineal de las funciones de pérdida individuales:

$$L(\theta; X, Y) = \sum_{m=1}^{M} w_m L_m(\theta; X, Y) \tag{4.21}$$

Donde w_m son los pesos asociados a cada tarea. Los parámetros del modelo se actualizan en función de esta pérdida total.

Aprendizaje por transferencia

En el aprendizaje por transferencia, un modelo previamente entrenado se utiliza como punto de partida para aprender nuevas tareas[55]. Supongamos que θ_t son los parámetros del modelo previamente entrenado. En el aprendizaje por transferencia, los parámetros del modelo se inicializan con θ_t y luego se ajustan mediante el entrenamiento en la nueva tarea:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t; X_{new}, Y_{new}) \tag{4.22}$$

Donde X_{new} y Y_{new} son los datos de la nueva tarea. Esto puede mejorar la eficiencia del aprendizaje y la capacidad de generalización del modelo al permitirle aprovechar los patrones aprendidos en la tarea original.

4.4. Aprendizaje por refuerzo a partir de retroalimentación humana (RLHF)

El aprendizaje por refuerzo a partir de retroalimentación humana (Reinforcement Learning from Human Feedback, RLHF) es un enfoque que combina técnicas de aprendizaje por refuerzo y aprendizaje supervisado [12]. Esta metodología fue desarrollada para abordar algunas de las limitaciones asociadas con los enfoques tradicionales de aprendizaje por refuerzo, tales como la dificultad de definir funciones de recompensa apropiadas.

El concepto subyacente en RLHF es que un agente aprende a través de la experiencia y la retroalimentación proporcionada por un operador humano. En lugar de depender únicamente de las recompensas programadas en un entorno de aprendizaje por refuerzo, RLHF utiliza la retroalimentación de un humano para guiar y mejorar el proceso de aprendizaje. Esta retroalimentación puede ser una calificación de la calidad de diferentes acciones que el agente ha llevado a cabo.

Formalmente, podemos describir RLHF con la siguiente ecuación de optimización:

$$V(s) = \sum_{a} \pi(a|s) \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right) \tag{4.23}$$

Donde $V(s)$ es el valor del estado s, que representa la estimación de la recompensa esperada a largo plazo para un estado dado. $\pi(a|s)$ es la política, que define la probabilidad de seleccionar una acción a dado un estado s. $R(s,a)$ es la recompensa instantánea obtenida al tomar la acción a en el estado s. γ es el factor de descuento, que determina la importancia relativa de las recompensas futuras en comparación con las recompensas inmediatas. $P(s'|s,a)$ es la función de transición, que modela la probabilidad de pasar del estado s al estado s' al tomar la acción a.

La ecuación del aprendizaje por refuerzo humano muestra cómo el valor de un estado se actualiza iterativamente utilizando la política, las recompensas y la función de transición. El objetivo es encontrar la política óptima que maximice el valor esperado a largo plazo.

4.5. Modelos matemáticos para la interpretabilidad y la explicabilidad

La **interpretabilidad** y la **explicabilidad** son conceptos de creciente importancia en la inteligencia artificial y el aprendizaje automático, dado que los sistemas complejos de modelos de lenguaje como GPT-4 pueden ser **cajas negras** en términos de comprensión intuitiva. La interpretabilidad se refiere a la medida en que un ser humano puede entender la causa y el efecto subyacente en un sistema, mientras que la explicabilidad se refiere a la capacidad de un sistema para explicar o dar cuenta de sus acciones.

4.5.1. Técnicas matemáticas para la interpretabilidad

Algunas técnicas para mejorar la interpretabilidad incluyen el uso de modelos lineales, árboles de decisión o bosques aleatorios, ya que estos modelos permiten una inspección más directa de las relaciones de características [18]. Otro enfoque es utilizar técnicas de atención en los modelos de lenguaje, que resaltan qué partes del input se utilizaron más fuertemente para hacer una predicción. Los mapas de calor de atención como el de la Figura 9 de la sección 3.2.2 pueden ser útiles en este aspecto.

Modelos lineales

En un modelo lineal, la salida se calcula como una suma ponderada de las características de entrada, más un término de sesgo. La ecuación de un modelo lineal es la siguiente:

$$y = \sum_{i=1}^{n} w_i * x_i + b \tag{4.24}$$

Aquí, y es la salida, x_i son las características de entrada, w_i son los pesos del modelo que indican la importancia de cada característica y b es el término de sesgo. Los pesos pueden ser positivos o negativos, indicando una relación positiva o negativa respectivamente con la salida.

Árboles de decisión y bosques aleatorios

En un árbol de decisión, cada nodo es una condición en una característica de entrada, y las hojas son los valores de salida. En un bosque aleatorio, múltiples árboles de decisión se combinan para hacer una predicción final. La interpretación se puede realizar analizando las condiciones y las características utilizadas en los nodos de los árboles.

Grad-CAM (Gradient-weighted Class Activation Mapping)

Esta técnica genera mapas de calor visuales utilizando los gradientes de la salida del modelo con respecto a las características convolucionales de entrada. La ecuación para

Grad-CAM es la siguiente:

$$L^c_{Grad-CAM} = ReLU(\sum_k \alpha^c_k A^k) \tag{4.25}$$

Aquí, $L^c_{Grad-CAM}$ es el mapa de activación ponderado para la clase c, α^c_k son los pesos de los canales que se calculan como la media del gradiente de la salida con respecto a cada canal de las características convolucionales A_k , y ReLU es la función de activación Rectified Linear Unit que se aplica para mantener solo los elementos positivos del mapa.

En el contexto de los modelos de lenguaje, puede adaptarse para visualizar las partes de la entrada que más contribuyen a la salida.

Interpretación basada en el juego de Shapley

Esta técnica se basa en la teoría de juegos cooperativos y asigna un valor a cada característica en función de su contribución a la salida. La ecuación para los valores de Shapley es la siguiente:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!}(v(S \cup \{i\}) - v(S)) \tag{4.26}$$

Aquí, ϕ_i es el valor de Shapley para la característica i, N es el conjunto de todas las característica, S son los subconjuntos de N que no incluyen i, $|S|$ es el número de elementos en S, $|N|$ es el número de elementos en N, $v(SUi)$ es el valor del juego cuando i coopera con S, y $v(S)$ es el valor del juego cuando i no coopera con S. El valor de Shapley para una característica mide su contribución marginal promedio a la salida del modelo[61].

4.5.2. Técnicas matemáticas para la explicabilidad

Generalmente, cuando hablamos de la explicabilidad en la inteligencia artificial, instintivamente comprendemos que estamos buscando desentrañar el razonamiento de los algoritmos, queremos entender cómo estos toman sus decisiones o llegan a ciertas inferencias.

Al referirnos a esta habilidad, es común encontrar el término explicabilidad, aunque la palabra interpretabilidad también se usa con regularidad y a menudo se intercambian como

sinónimos.

Para un uso casual, esta equivalencia podría ser suficiente. No obstante, el libro *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* [57], escrito por Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen y Klaus-Robert Müller, introduce un matiz, una diferenciación que puede ser importante para entender mejor estos conceptos.

Según esta distinción, el término más elemental sería interpretabilidad, que se refiere a la habilidad de comunicar la operación de un algoritmo de machine learning a un usuario (donde operación se entiende, se supone, como la forma en que las entradas se transforman en salidas).

Para que exista explicabilidad, se requiere interpretabilidad, es decir, la interpretabilidad es una condición necesaria, pero no es suficiente por sí sola. Para alcanzar la explicabilidad, se debe añadir un componente adicional, la completitud, que en este contexto significa que la explicación obtenida debe ser lo suficientemente detallada para poder ser auditada, y, de alguna manera, justificar sus acciones y proporcionar respuestas a las preguntas que surjan.

De acuerdo con este razonamiento, la explicabilidad sería la combinación o suma de la interpretabilidad y la completitud.

LIME (Local Interpretable Model-Agnostic Explanations)

LIME (Local Interpretable Model-Agnostic Explanations) [54] es una técnica que pretende crear explicaciones simples y comprensibles para las predicciones individuales de cualquier modelo de aprendizaje automático. Lo hace aproximando la predicción del modelo original, f(x), con un modelo interpretable más simple, g(x), en la vecindad local del punto de interés.

La aproximación de LIME se puede describir formalmente a través de la siguiente ecuación de optimización:

$$min_{g \in G} = L(f, g, \pi_x) + \Omega(g) \tag{4.27}$$

Donde $f(x)$ es el modelo que estamos tratando de explicar, $g(x)$ es el modelo interpretable (puede ser un modelo lineal o un árbol de decisión) que estamos intentando aprender, $\mathcal{G}$ es el espacio de modelos interpretables, π_x es una medida de proximidad del punto de interés, x, $\mathcal{L}(f, g, \pi_x)$ es una función de pérdida que mide cómo de cerca está $g(x)$ de $f(x)$ en la vecindad definida por π_x. Una elección común para la función de pérdida es la pérdida de mínimos cuadrados: $\mathcal{L}(f, g, \pi_x) = \sum \pi_x(i)(f(x_i) - g(x_i))^2$, $\Omega(g)$ es una medida de la complejidad del modelo $g(x)$. Esto proporciona una penalización a modelos más complejos para garantizar que la explicación sea lo más simple posible. Para un modelo lineal, por ejemplo, esto podría ser simplemente el número de características no nulas.

El objetivo de la optimización es encontrar un modelo $g(x)$ que minimice la función de pérdida (es decir, que se aproxime bien a $f(x)$ en la vecindad local), mientras que se mantiene la complejidad del modelo lo más baja posible.

Por lo tanto, LIME asume que la superficie de decisión del modelo puede ser aproximada localmente por una función lineal en el espacio de entrada original. Esto es una suposición fuerte que puede no ser verdadera en todos los casos.

Perturbación de características

El método de perturbación de características intenta entender la importancia de una característica cambiando su valor y observando el efecto resultante en la salida del modelo. Este método puede ser especialmente útil para los modelos de aprendizaje automático que toman como entrada datos de alta dimensionalidad, como las imágenes [19].

La perturbación de características puede ser vista como una función objetivo que mide la importancia de una característica. Esto se puede expresar matemáticamente de la siguiente manera:

$$I(x_i) = L(f(x), f(x')) - L(f(x), y) \tag{4.28}$$

donde $I(x_i)$es la importancia de la característica i-ésima, L es una función de pérdida, $f(x)$ es la predicción del modelo para la entrada original x, $f(x')$ es la predicción del modelo para la entrada perturbada x', y es el valor objetivo verdadero.

Aquí, x' es una versión de x en la que la característica i ha sido perturbada. Si la perturbación de la característica i provoca un gran cambio en la salida del modelo (es decir, un gran valor de $I(x_i)$), entonces podemos concluir que la característica i es importante para el modelo.

La divergencia de Kullback-Leibler (KL)

Las técnicas basadas en la *teoría de la información* pueden ser útiles para cuantificar la cantidad de información que las diferentes partes de la entrada proporcionan sobre la salida [14]. La divergencia de Kullback-Leibler (KL) es una medidas comúnmente utilizadas en este contexto. Esta expresión cuantifica la diferencia entre dos distribuciones de probabilidad P y Q. Se define como:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)} \tag{4.29}$$

La divergencia KL no es simétrica, lo que significa que $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. Si las dos distribuciones son idénticas, entonces su divergencia KL es cero.

Estas medidas pueden ser útiles para entender la importancia de diferentes partes de la entrada para la salida del modelo. Por ejemplo, podríamos buscar aquellas partes de la entrada que, cuando se perturban, resultan en un gran cambio en la entropía o la divergencia KL de la salida del modelo.

Capítulo 5

Impacto ético y social de los LLM

El impacto ético y social de los modelos de lenguaje de gran escala (LLM) se encuentra en constante evolución y desarrollo. Al momento de redactar este Trabajo Final de Máster, se nos presenta un desafío al intentar definir un estado del arte en inteligencia artificial. Nos encontramos inmersos en una revolución científica a la que cuesta encontrar referentes en los últimos tiempos y que está permeando todos los sectores de la sociedad. Esta dinámica de constante cambio nos impide adoptar una perspectiva consolidada y definitiva al respecto.

Sin embargo, este capítulo tiene como objetivo analizar los aspectos más notables y actuales, hasta junio de 2023, sobre los modelos de lenguaje de gran escala. En particular, nos centraremos en su integración como herramientas prácticas en diversos campos, así como en la percepción social, política y mediática que estos modelos están generando.

En primera instancia, es vital mencionar los logros significativos obtenidos por estos modelos de lenguaje. Han mostrado una asombrosa capacidad para realizar tareas de procesamiento de lenguaje natural, transformando sectores como la traducción automática, la generación de texto y la interacción hombre-máquina. Su uso se ha expandido a diversas aplicaciones, desde asistentes de voz inteligentes hasta herramientas de análisis de sentimientos en redes sociales.

No obstante, junto con estas prometedoras aplicaciones, también ha surgido una serie de

preocupaciones éticas y sociales. Estas preocupaciones giran en torno a temas como la deshumanización, la pérdida de empleos masiva, el sesgo en los datos de entrenamiento, la privacidad y seguridad de los datos, y el potencial riesgo de extinción humana. Así, se vuelve imprescindible un análisis crítico y una reflexión sobre estos aspectos.

5.1. Impacto político

5.1.1. The IA Act

El "The IA Act"[34], anunciado por la Comisión Europea en abril de 2021, tiene como objetivo proporcionar un marco claro para el desarrollo y uso de la IA en toda la Unión Europea. La propuesta de esta legislación es amplia, abarcando una variedad de aplicaciones de IA que van desde procesos más sencillos de aprendizaje automático y sistemas de chat hasta tecnologías más complejas como los algoritmos de reconocimiento facial.

Las regulaciones esbozadas en el Acto categorizan los sistemas de IA según sus posibles riesgos en tres categorías: 'inaceptable', 'alto' y 'bajo/mínimo'.

Los sistemas de IA que se clasifican como de riesgo 'inaceptable' se consideran extremadamente perjudiciales, hasta el punto de que se propone su prohibición total. Estos sistemas suelen suponer una clara amenaza para la seguridad, los medios de vida y los derechos de las personas y de la sociedad en general. Como la puntuación social gestionada por el gobierno del tipo que se utiliza en China.

Por otro lado, las aplicaciones de IA de 'alto' riesgo no están prohibidas, pero están sujetas al cumplimiento estricto de un conjunto de requisitos y regulaciones rigurosos. Estos pueden incluir sistemas con implicaciones significativas en la vida de las personas o en el orden social, como la identificación biométrica y los sistemas de infraestructuras críticas. Se espera que los sistemas de IA de alto riesgo cumplan con estrictos estándares de transparencia, trazabilidad y fiabilidad. Como una herramienta de escaneo de currículums que clasifica a los solicitantes de empleo, están sujetas a requisitos legales específicos.

Las aplicaciones de 'bajo' o 'mínimo' riesgo representan una categoría de sistemas de IA que se consideran menos propensos a causar daño. Estos sistemas aún deberán adherirse

a la regulación, pero los requisitos serán menos estrictos en comparación con los sistemas de alto riesgo. Ejemplos de estos podrían ser los chatbots de IA o sistemas de recomendación que están principalmente destinados a mejorar la experiencia del usuario en lugar de tomar decisiones cruciales.

El incumplimiento de estas regulaciones viene con severas sanciones financieras. En caso de no cumplimiento, una empresa podría enfrentar multas que ascienden hasta el 6 por ciento de su facturación total anual global. Esto demuestra el compromiso de la Comisión Europea para asegurar que el desarrollo y uso de la IA respete los derechos fundamentales y garantice la seguridad humana.

Existen varias lagunas y excepciones en la ley propuesta. Estas deficiencias limitan la capacidad del Acta para garantizar que la IA siga siendo una fuerza positiva. Actualmente, por ejemplo, el reconocimiento facial por parte de la policía está prohibido a menos que las imágenes se capturen con un retraso o la tecnología se utilice para buscar niños perdidos.

Además, la ley es rígida. Si dentro de dos años se utiliza una aplicación de IA peligrosa en un sector imprevisto, la ley no proporciona ningún mecanismo para etiquetarlo como de alto riesgo.

En el momento en el que se escribe este TFM, la Unión Europea (UE) continúa avanzando en su camino hacia la aprobación de la primera ley mundial que regula la utilización de la inteligencia artificial (IA). Tras extensos debates internos, este jueves las dos comisiones parlamentarias encargadas de la redacción de la regulación han aprobado la posición de negociación. Este es un punto de vista que el Parlamento Europeo deberá votar a mediados de junio e impone una serie de requisitos a aplicaciones como ChatGPT.

La propuesta de ley formulada por los miembros del parlamento europeo dicta que los modelos generativos de lenguaje que utilizan el conocido chatbot de conversación de OpenAI o Bard, la opción desarrollada por Google, tendrán que cumplir con exigencias adicionales de transparencia. Esto implicará que estas grandes empresas deberán revelar si emplean material protegido por derechos de autor, indicar qué contenidos han sido generados con IA y diseñar estos sistemas para prevenir la generación de contenidos ilegales.

5.1.2. Planning for AGI and beyond

Sam Altman, el líder de OpenAI, ha presentado 'Planning for AGI and beyond' ('Planeando para la Inteligencia Artificial General (IAG) y más allá'), un documento que se asemeja a un plan estratégico, una declaración de propósitos y un adelanto de futuros desarrollos en la IAG. Este término se refiere comúnmente a las Inteligencias Artificiales de capacidad general y "más inteligentes que los humanos", en contraposición a las IAs específicas y "menos inteligentes que las personas".

En las primeras líneas del documento, Altman resume sus reflexiones:

- Nuestro objetivo es que la IAG permita a la humanidad alcanzar su máximo potencial en el universo... queremos maximizar los aspectos positivos y minimizar los negativos, viendo a la IAG como un catalizador para la humanidad.

- Deseamos que los beneficios, el acceso y la gobernanza de la IAG se distribuyan de manera equitativa y extensa.

- Nos proponemos superar los principales riesgos... Creemos en la necesidad de aprender y adaptarnos continuamente mediante la implementación de tecnologías menos poderosas para evitar escenarios en los que sólo tengamos una oportunidad para hacerlo bien.

Un fragmento particularmente intrigante del texto de OpenAI alude a peligros desconocidos que podrían representar amenazas existenciales:

(...) A medida que nos acercamos a la Inteligencia Artificial General, aumentará nuestro cuidado en la construcción y despliegue de nuestros modelos. Nuestro enfoque demandará un nivel de precaución mucho mayor al que la sociedad habitualmente aplica a las nuevas tecnologías, y mayor al que muchos de los usuarios preferirían. Algunos expertos en IA ven los riesgos de la Inteligencia Artificial General (y sistemas subsiguientes) como algo meramente teórico; nos gustaría que estuvieran en lo correcto, pero vamos a proceder como si estos riesgos fueran amenazas a la existencia.

En los últimos tiempos, Sam Altman se ha reunido con diferentes dirigentes europeos y norteamericanos solicitando regulaciones para los grandes modelos de Inteligencia Artifi-

cial y con alarmas extrañamente catastrofistas para provenir precisamente del responsable de la empresa de IA que más rápido ha crecido. Estas peticiones parecen altruistas, pero para una empresa mil millonaria y responsable de la mayor revolución tecnológica de estos años, tenemos que analizar con cuidado sus intenciones.

El crecimiento acelerado de la tecnología y la naturaleza emergente de la inteligencia artificial pueden representar desafíos significativos para las plataformas establecidas como ChatGPT, especialmente considerando el dinamismo y la innovación de las startups tecnológicas. La introducción de regulaciones estrictas podría afectar potencialmente a estas startups más pequeñas y menos establecidas, que a menudo están formadas por académicos y estudiantes y que pueden carecer de los recursos y conexiones necesarios para adaptarse a los cambios regulatorios.

Además, la necesidad de cumplir con la regulación podría requerir una inversión de recursos y tiempo que las startups pueden no tener, lo que potencialmente podría afectar su competitividad.

Sam Altman, al igual que otros líderes tecnológicos de Estados Unidos, está buscando expandir su empresa en Europa, una región conocida por su enfoque más riguroso en la regulación. Esto contrasta con la mentalidad de Silicon Valley, que a menudo prioriza la innovación rápida y la introducción de nuevos productos en el mercado.

Pese a estas diferencias, ChatGPT expresa su apoyo a la regulación, aunque enfatiza la importancia de que esta sea medida y equilibrada. A pesar de la abogacía de la regulación, las empresas de tecnología pueden expresar su preocupación cuando la legislación propuesta puede afectar negativamente sus intereses.

Por ejemplo, Altman expresó su inquietud después de que se publicó el borrador oficial de la ley de IA en Europa descrita en el apartado anterior, que establece regulaciones estrictas para sectores considerados de alto riesgo. Altman mencionó la posibilidad de retirar ChatGPT de Europa si la ley propuesta no cumple con sus expectativas. Esta situación ilustra la complejidad de la regulación de la IA y cómo las expectativas pueden cambiar en respuesta a las realidades regulatorias emergentes.

5.1.3. Huelga de guionistas en Hollywood de 2023

Aunque existan otras reivindicaciones como la mejora salarial, el impacto de la Inteligencia Artificial es una de las cuestiones en las negociaciones de la huelga de guionistas. Los profesionales creativos, incluyendo guionistas, diseñadores, traductores, músicos y escritores, temen que la IA sea utilizada para sustituirlos en un esfuerzo por reducir costes.

El sindicato de guionistas no busca prohibir la IA, según John Rogers, miembro del grupo de trabajo. La intención es regular su uso. Sin embargo, las negociaciones con los estudios de Hollywood no han sido fructíferas, lo que llevó a una huelga en mayo.

La alianza de productores de cine y televisión argumenta que las conversaciones se estancaron en parte porque el sindicato se negó a renunciar al uso de la IA en el futuro. Por su parte, el sindicato argumenta que su propuesta no excluye la IA, pero busca regular cómo se usa.

Michael Colton, coproductor ejecutivo de la comedia de ABC Home Economics, expresa sus preocupaciones. Su temor no es que la IA escriba guiones de comedia en el futuro próximo, sino que los estudios utilicen la IA para generar primeros borradores que luego serán pulidos por guionistas contratados por poco tiempo, lo que podría resultar en una disminución en la compensación.

El grupo de trabajo en IA no busca prohibir completamente el uso de la IA. Según Rogers, hacerlo sería contraproducente. En cambio, buscan una forma de utilizar la IA de manera beneficiosa y limitada para los guionistas.

Al momento de la escritura de este TFM (Junio 2023) los movilizaciones continúan y suman más de un mes desde que se iniciaron el 2 de Mayo.

5.2. Impacto social

Es difícil tomar la perspectiva necesaria que permite evaluar el impacto social de una tecnología que ha irrumpido en nuestras vidas en solo unos meses. Sin embargo, entre todo el revuelo mediático generado entre las posturas más catastrofistas y las más ingenuas con

respecto a la implementación de la IA en nuestra sociedad, destacan algunas situaciones que se recogen a continuación.

5.2.1. Pause Giant AI Experiments: An Open Letter

Los medios se hicieron mucho eco de la *Carta de los expertos en IA* un texto en el que un conjunto de expertos en inteligencia artificial y líderes de la industria tecnológica solicitaron a comienzos del 2023 una moratoria de seis meses en la formación de sistemas avanzados de inteligencia artificial, citando su potencial peligro para la humanidad [29].

En una carta pública alojada en la página de la fundación Future of Life, señalaron que los laboratorios que exploran esta tecnología se encuentran en *una competencia desenfrenada para desarrollar y poner en marcha inteligencias digitales que incrementan su poder a tal ritmo que nadie, ni siquiera los que las diseñan, pueden entender, prever o controlar de manera segura*.

La misiva fue respaldada por más de mil firmantes, entre los que se encuentran personalidades como Elon Musk, empresario tecnológico, Steve Wozniak, cofundador de Apple, y Emad Mostaque, CEO de Stability AI, así como investigadores de DeepMind.

La carta se centra en la discusión sobre la regulación y la gestión cuidadosa del desarrollo de la Inteligencia Artificial (IA) avanzada. Hace hincapié en los riesgos potenciales que pueden surgir si la IA de capacidad humana competitiva se desarrolla y se implementa sin un control y una comprensión adecuados. Según este texto, los sistemas de IA contemporáneos están adquiriendo habilidades comparables a las humanas en tareas generales, lo que puede conducir a importantes implicaciones y riesgos.

De esta forma, insta a los laboratorios de IA a hacer una pausa en el entrenamiento de sistemas de IA más poderosos que GPT-4 durante al menos 6 meses. Esta pausa sería un período para el desarrollo conjunto e implementación de protocolos de seguridad para el diseño y desarrollo de IA avanzada. Sugiere que los modelos de IA deberían ser predecibles y comprensibles, más allá de la simple búsqueda de capacidades emergentes.

El texto también argumenta a favor de un enfoque más concentrado en hacer que los

sistemas de IA existentes sean más precisos, seguros, interpretables y confiables, en lugar de la mera carrera hacia modelos más grandes y potentes.

Finalmente, pide a los desarrolladores de IA que trabajen junto con los legisladores para acelerar el desarrollo de sistemas de gobierno de IA robustos. Estos sistemas de gobernanza deberían incluir aspectos como autoridades reguladoras dedicadas a la IA, seguimiento de sistemas de IA altamente capaces, auditorías y certificación, responsabilidad por daños causados por IA, y una financiación robusta para la investigación en seguridad de la IA, entre otros. El objetivo final sería garantizar que los efectos de la IA en la sociedad sean positivos y manejables, evitando los riesgos potenciales significativos.

5.2.2. Trabajos académicos con ChatGPT

La herramienta de generación de texto impulsada por inteligencia artificial de OpenAI, inmediatamente después de su lanzamiento en noviembre de 2022, instigó una transformación en el ámbito educativo. Este fin de ciclo académico marcará la primera ocasión en que ChatGPT esté completamente accesible para los estudiantes, potencialmente asistiéndoles en la elaboración de miles de trabajos de final de curso.

La eficacia de estas aplicaciones se basa en la rapidez y precisión con las que generan respuestas automatizadas. Paul Taylor [64], quien imparte la cátedra de Informática de la Salud en la University College de Londres, recientemente compartió en un artículo para la London Review of Books, su experiencia usando Chatbot en contextos académicos. Según Taylor, la inteligencia artificial demostró ser capaz de proporcionar respuestas coherentes y precisas, y enfocadas en el tema, un nivel de rendimiento que en ocasiones supera al de los estudiantes.

Las herramientas para la detección de plagio se encuentran en aprietos. Según el profesor, estos sistemas resultan ineficaces, dado que el algoritmo de ChatGPT genera respuestas únicas para cada interacción, lo que implica que los estudiantes no recibirán las mismas respuestas que él obtuvo. En vista de esto, Taylor sugiere que, en el futuro, podría ser necesario idear formatos de examen innovadores o incluso administrar las pruebas en un ambiente sin conexión a Internet.

5.2.3. Clasificadores de texto IA/Humano

Cada vez más personas se apoyan en Modelos de Lenguaje, fundamentalmente ChatGPT, para la elaboración de sus trabajos académicos. Ante eso han surgido algunas alternativas para controlar su uso mediante el desarrollo de herramientas de detección y clasificación de textos escritos por IA. Open AI tiene el IA Text Clasifier, pero existen otros ejemplos de software libre como Writer, ChatGPT Detector y GPTZero, que basan sus metodos para clasificar en el cálculo de la perplejidad del texto, descrita en el capítulo 4 y el análisis de **marcas de agua** que dependen del modelo de lenguaje.

Cabe destacar que estos modelos no dan un resultado taxativo, sino una probabilidad más o menos cercana a que el texto sea *Más parecido al de la IA* o *Más parecido al humano*. GPTZero, por ejemplo, realiza un análisis de la perplejidad global del texto e incorpora el cálculo de perplejidad frase a frase para determinar la probabilidad de escritura del texto.

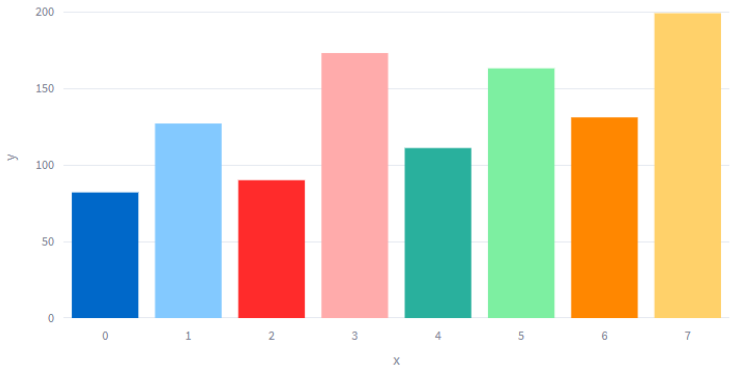

Figura 16: Evaluación del nivel de perplejidad y por cada frase, x del texto que comforma el punto anterior 5.2.2. Trabajos académicos con ChatGPT realizado con GPTZero

En la Figura 16 hemos tomado de referencia el punto anterior (5.2.2. Trabajos académicos con ChatGPT) y lo hemos pasado por la herramienta de análisis de texto GPTZero para obtener su perplejidad y valoración final. El texto pasa la prueba como escrito por humano, pero el modelo de clasificación está basado en GPT-2 por lo que textos creados por GPT-4 podrían pasar la prueba.

En su artículo *Can AI-Generated Text be Reliably Detected?* [56] sus autores afirman demostrar tanto empírica como teoricamente que estos detectores no son confiables en

escenarios prácticos. Los ataques de parafraseo, donde se aplica un parafraseador ligero sobre el modelo de texto generativo, pueden romper un amplio rango de detectores, incluyendo los que usan esquemas de marca de agua, así como los detectores basados en redes neuronales y los clasificadores de Zero-shot. Además aportan un resultado teórico de imposibilidad que afirma que para un modelo de lenguaje suficientemente bueno, incluso el mejor detector posible solo puede desempeñarse marginalmente mejor que un clasificador aleatorio. Finalmente, mostramos que incluso los MLGs protegidos por esquemas de marca de agua pueden ser vulnerables a ataques de suplantación donde los humanos adversarios pueden inferir firmas de marca de agua ocultas y añadirlas a su texto generado para ser detectado como texto generado por los MLGs,

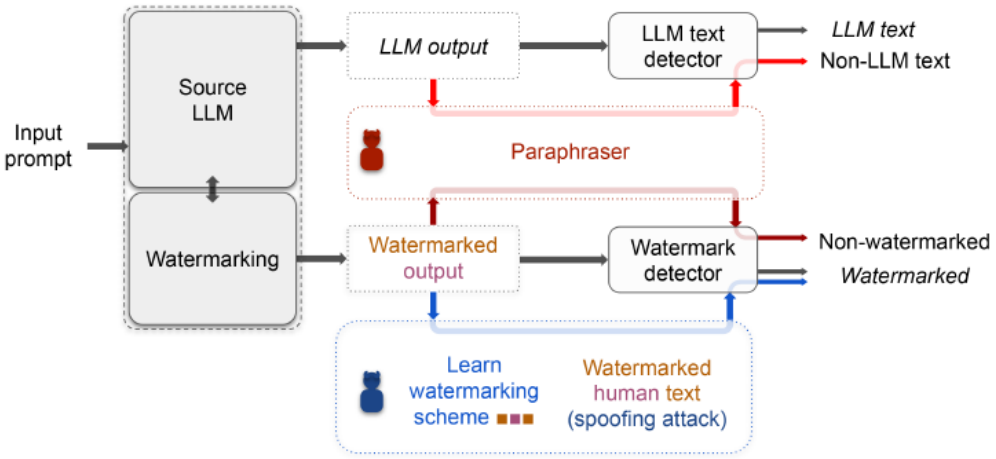

Figura 17: Vulnerabilidades de los sistemas de clasificación de texto IA/Humano. Extraído de [56]

Los senderos de flechas de colores representan estrategias potenciales que los adversarios podrían seguir para eludir la detección (Ver Figura 17). En el caso rojo, un atacante podría eliminar las firmas del LLM en un texto producido por IA mediante parafraseo para evitar ser detectado. Este tipo de ataque puede invalidar una amplia gama de detectores. Además, en el artículo se muestra que un modelo de lenguaje altamente eficiente hace que incluso el detector óptimo sea apenas marginalmente mejor que un clasificador al azar. En el caso azul, un adversario podría aprender el esquema de marca de agua del LLM consultándolo varias veces. Esta información podría ser utilizada para burlar al detector de marcas de agua, creando texto humano que parece tener marca de agua de IA.

Capítulo 6

Conclusión

Vivimos en una época de auténtica revolución en el campo de la Inteligencia Artificial. Los avances suceden a un ritmo vertiginoso. Semana tras semana, nos sorprendemos con los hitos conseguidos por las grandes empresas de software. Sus resultados son fácilmente comprobables con las nuevas aplicaciones que, a su vez, cada vez resultan más accesibles para el gran público. En ellas encontramos programas con los que podemos obtener texto, imagen, música e incluso contenido audiovisual generado a partir de solicitudes sencillas del usuario y con un nivel asombroso de corrección, coherencia y valor estético.

Las redes neuronales de tipo Transformer son las responsables de los espectaculares resultados obtenidos en los últimos años. Han llegado a cotas de coherencia nunca antes alcanzadas por sus antecesoras: Las redes neuronales recurrentes o las de memorias a largo plazo (LSTM). Estas presentaban problemas de corrección cuando se proponía la generación de textos más largos, de tal forma que al principio las frases completadas eran aceptablemente coherentes, pero a medida que se avanzaba con las implicaciones lógicas del escrito, este se iba volviendo más incongruente, degradándose hasta el sin sentido. La principal mejora al respecto de los Transformer consiste en un mecanismo de atención, que evalúa la relevancia de cada token en función de la tarea propuesta por el usuario, además del análisis de secuencias de texto de gran tamaño en paralelo. Esta capacidad de atención contextual les ha permitido dar un paso de gigante con respecto a sus predecesoras.

De todos los modelos de lenguaje basados en Transformers, GPT-4 de Open-AI y sus predecesores son los que han copado la mayor parte de prensa especializada y sirven de base para la mayoría de aplicaciones basadas en IA de la actualidad más reciente. Mientras tanto, las grandes empresas tecnológicas como Microsoft, Google y Meta le siguen a la zaga con sus respectivos modelos: Bard, ChatBing y LLaMa. Muchas voces entre los expertos y los medios reflexionan de forma un tanto apocalíptica acerca de la posibilidad de desaparición de trabajos como escritores, guionistas o diseñadores gráficos, que serán sustituidos por esta inteligencia artificial para la generación de contenido, llegando incluso a producir una huelga histórica de guionistas en Holywood que se mantiene a la fecha de escribir estas líneas (por no hablar de los miedos de deshumanización y extinción que la ciencia ficción nos lleva inculcando durante décadas).

Esto proviene no solo de la idea intuitiva de que, para generar un texto coherente y correcto, la IA debe desenvolverse con eficiencia en múltiples tareas complejas, que parecen necesariamente vinculadas a habilidades de cognición similares a las humanas, sino de estrategias corporativas o mediáticas para las que transmitirnos esa idea puede beneficiar sus intereses de diferentes maneras.

Es un desafío prever el camino futuro de los modelos de lenguaje, pero no podemos evitar asombrarnos de las proezas que han logrado hasta ahora. Lo que resulta más sorprendente es cómo la esencia de estas máquinas de aprendizaje se basa en conceptos estructurales que no son nuevos ni especialmente complejos, sino que llevan años con nosotros y que provienen de la optimización, el aprendizaje profundo e incluso del álgebra lineal, como es el caso de los espacios vectoriales. Sin embargo, esto no debe hacernos pasar por alto el grado de complejidad y sofisticación que se alcanza en su desarrollo y funcionamiento.

Aunque la esencia de estos modelos puede parecer accesible, el verdadero logro radica en su implementación y en el nivel de detalle que encierra. Construir, entrenar y mantener estos modelos requiere no sólo de un gran compromiso de recursos materiales, como el hardware, sino también de un equipo humano altamente cualificado que pueda dirigir y supervisar el proceso de aprendizaje del modelo.

Además, el avance y la mejora continua de estos modelos están impulsados por una se-

rie de investigaciones técnicas y matemáticas profundas en campos como el aprendizaje profundo, el procesamiento del lenguaje natural y la optimización. Si bien la estructura básica de los modelos de lenguaje puede parecer simple, la ingeniería y la investigación detrás de su funcionamiento son extremadamente sofisticadas.

6.1. Valoración personal

Incluir una componente de reflexión personal es imprescindible en este TFM. Dada la naturaleza novedosa y disruptiva de la tecnología en cuestión, algunas de mis impresiones pueden no estar sustentadas por la evidencia académica que es requerida en este tipo de trabajo. Este trabajo marca mi último paso como estudiante del Doble Máster en Matemática y MAES, y es inevitable que mi análisis esté impregnado por mis inquietudes acerca del impacto potencialmente nocivo de esta tecnología en futuras generaciones. Concebida en esencia para liberarnos de tareas mundanas, me pregunto si privar a nuestro cerebro de ciertos procesos cognitivos podría tener consecuencias negativas.

Durante los meses de elaboración de este TFM, he estado realizando prácticas en un centro de enseñanza y trabajando en una academia de refuerzo de Matemáticas, Física y Química. He observado que muchos estudiantes de la ESO utilizan ChatGPT para redactar sus ejercicios y tareas, y algunos han sido sancionados al ser detectado su uso, lo que ha proporcionado un escarmiento que en el mejor de los casos será temporal. Sin embargo, la realidad es que la detección de texto generado por IA parece ser inviable por definición, no por el margen de error, sino por la capacidad del ser humano para contrarrestar cualquier sistema con técnicas de parafraseo o eliminación de marcas de agua.

Como escritor, conozco la sensación del bloqueo creativo y en ese momento te aferras a cualquier cosa para eliminarla. Temo que lo que se supone como un ahorro en tareas repetitivas y tediosas sea en realidad una herramienta que despoja a los seres humanos de la necesidad de sintetizar contenido, estructurar la información y perfeccionar la expresión de sus ideas en textos escritos. Al no atravesar esos procesos cognitivos nuestra capacidad de realizarlos podría llegar a atrofiarse. Y la forma en que modelizamos el lenguaje es también la forma en que modelizamos nuestro pensamiento como especie.

Índice de figuras

Bibliografía

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Marion Bartl, Malvina Nissim, and Albert Gatt. Unmasking contextual stereotypes: Measuring and mitigating bert's gender bias. *arXiv preprint arXiv:2010.14534*, 2020.

[4] Yoshua Bengio and Paolo Frasconi. An input output hmm architecture. *Advances in neural information processing systems*, 7:41–51, 1994.

[5] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. *Advances in neural information processing systems*, 32:234–251, 2019.

[6] Christopher M Bishop and Nasser M Nasrabadi. Pattern recognition and machine learning. Number 4, pages 108–125. Springer, 2006.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Miles Brundage. Taking superintelligence seriously. *Futures*, 72:32–35, 2015.

[9] Edward Y Chang. Examining gpt-4: Capabilities, implications, and future directions. *arXiv:submit/4991987*, 2023.

[10] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[11] Noam Chomsky. 7. the logical basis of linguistic theory. In *Eight decades of general linguistics*, pages 123–236. Brill, 2013–2019.

[12] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30:111–122, 2017.

[13] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

[14] Thomas M Cover and Joy A Thomas. Information theory and statistics. *Elements of information theory*, 1(1):279–335, 1991.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Xiaoyi Dong, Jianmin Bao, Ting Zhang, Dongdong Chen, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, Nenghai Yu, and Baining Guo. Peco: Perceptual codebook for bert pre-training of vision transformers. *arXiv preprint arXiv:2111.12710*, 2021.

[17] John Rupert Firth. Applications of general linguistics. *Transactions of the Philological Society*, 56(1):1–14, 1957.

[18] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.*, 20(177):1–81, 2019.

[19] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE international conference on computer vision*, pages 3429–3437, 2017.

[20] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.

[21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Convolutional networks. In *Deep learning*, pages 330–371. MIT press, 2016.

[23] Michael D Gordin. The dostoevsky machine in georgetown: Scientific translation in the cold war. *Annals of Science*, 73(2):208–223, 2016.

[24] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, et al. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *The Journal of Machine Learning Research*, 21(1):845–851, 2020.

[25] Wangli He, Feng Qian, James Lam, Guanrong Chen, Qing-Long Han, and Jürgen Kurths. Quasi-synchronization of heterogeneous dynamic networks via distributed impulsive control: Error estimation, optimization and design. *Automatica*, 62:249–262, 2015.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Ellis Horwood. Alpac (automatic language processing advisory committee)(1966). language and machines: Computers in translation and linguistics. *Advances in Computers*, 1:93–163, 1994.

[28] František Hrouda, Vit Jelínek, and Karel Zapletal. Refined technique for susceptibility resolution into ferromagnetic and paramagnetic components based on susceptibility temperature-variation measurement. *Geophysical Journal International*, 129(3):715–719, 1997.

[29] Marcello Ienca. Don't pause giant ai for the wrong reasons. *Nature Machine Intelligence*, pages 1–2, 2023.

[30] Carl James. *Errors in language learning and use: Exploring error analysis*. Routledge, 2013.

[31] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

[32] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] Mauritz Kop. Eu artificial intelligence act: the european approach to ai. Stanford-Vienna Transatlantic Technology Law Forum, 2021.

[35] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[38] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, 2014.

[39] Ping Li, Jianping Li, and Gongcheng Wang. Application of convolutional neural network in natural language processing. In *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 120–122. IEEE, 2018.

[40] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[41] Lajanugen Logeswaran, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, Jacob Devlin, and Honglak Lee. Zero-shot entity linking by reading entity descriptions. *arXiv preprint arXiv:1906.07348*, 2019.

[42] Alessandro Lumino, Emanuele Polino, Adil S Rab, Giorgio Milani, Nicolò Spagnolo, Nathan Wiebe, and Fabio Sciarrino. Experimental phase estimation enhanced by machine learning. *Physical Review Applied*, 10(4):044033, 2018.

[43] Huiyan Luo, Guoliang Xu, Chaofeng Li, Longjun He, Linna Luo, Zixian Wang, Bingzhong Jing, Yishu Deng, Ying Jin, Yin Li, et al. Real-time artificial intelligence for detection of upper gastrointestinal cancer by endoscopy: a multicentre, case-control, diagnostic study. *The Lancet Oncology*, 20(12):1645–1654, 2019.

[44] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[45] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[46] H Brendan McMahan, Galen Andrew, Ulfar Erlingsson, Steve Chien, Ilya Mironov, Nicolas Papernot, and Peter Kairouz. A general approach to adding differential privacy to iterative training procedures. *arXiv preprint arXiv:1812.06210*, 2018.

[47] Rahul Mehta and Vasudeva Varma. Llm-rm at semeval-2023 task 2: Multilingual complex ner using xlm-roberta. *arXiv preprint arXiv:2305.03300*, 2023.

[48] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[49] Gordon Moore. Moore's law. *Electronics Magazine*, 38(8):114–116, 1965.

[50] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[51] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[52] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[53] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[54] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[55] Sebastian Ruder, Parsa Ghaffari, and John G Breslin. A hierarchical model of reviews for aspect-based sentiment analysis. *arXiv preprint arXiv:1609.02745*, 2016.

[56] Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*, 2023.

[57] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. *Explainable AI: interpreting, explaining and visualizing deep learning*. Springer Nature, 2019.

[58] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[59] Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. Bertimbau: pretrained bert models for brazilian portuguese. In *Intelligent Systems: 9th Brazilian Conference, BRACIS 2020, Rio Grande, Brazil, October 20–23, 2020, Proceedings, Part I*, pages 403–417. Springer, 2020.

[60] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[61] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41:647–665, 2014.

[62] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014.

[63] Kanchan M Tarwani and Swathi Edem. Survey on recurrent neural network in natural language processing. *Int. J. Eng. Trends Technol*, 48(6):301–304, 2017.

[64] Paul Taylor. On chatgpt. *London Review of Books*, 45(1):1–4, 2023.

[65] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[66] Alan M Turing. Computing machinery and intelligence (1950). *The Essential Turing: the Ideas That Gave Birth to the Computer Age*, pages 433–464, 2012.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

[68] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2020.

[69] Joosung Yoon and Hyeoncheol Kim. Multi-channel lexicon integrated cnn-bilstm models for sentiment analysis. In *Proceedings of the 29th conference on computational linguistics and speech processing (ROCLING 2017)*, pages 244–253, 2017.

[70] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 2018.

[71] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.