PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE SISTEMAS DE INFORMACIÓN



PLAN DE TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA: PREENTRENAMIENTO DE UN MODELO GPT CON ARQUITECTURA TRANSFORMER

AUTOR

STEPHANO LUIS FELIPE VALLEJO MONGE

QUITO 12, ABRIL 2025

Contenido

1. Introducción *6*

1.1 Justificación 6
1.1.1 Planteamiento del Problema 6

1.2 Objetivos 7
1.2.1 Objetivo General 7
1.2.2 Objetivos Específicos 7

2. Antecedentes *8*

2.1 Marco Teórico 8

2.2 Herramientas 10
2.2.1 PyTorch 10
2.2.2 Python 10
2.2.3 Jupyter Notebooks 10
2.2.4 Framework de Deep Learning 10

2.3 Lotería de Hardware 11

2.4 Historia del Deep Learning 11
2.4.1 Neuronal Artificial y Perceptron 11
2.4.2 Red Neuronal Profunda 15
2.4.3 ¿Qué es machine learning? 15
2.4.4 Red Neuronal Vanilla 17
2.4.5 Descenso de Gradiente 19
2.4.6 ¿Qué es backpropagation? 21
2.4.7 ¿Qué ocurre dentro de las capas ocultas de la red neuronal? 22
2.4.8 ¿Qué es Dropout? 24
2.4.9 Teorema de Aproximación Universal 27

3. Análisis Simplificado de Artículos *29*

3.1 Attention Is All You Need 29
3.1.1 Encoder 32
3.1.2 Self-Attention 33
3.1.3 Positional Encoding 37
3.1.4 Layer Normalization 37
3.1.5 Decoder 39
3.1.6 Encoder-Decoder Attention 41
3.1.7 Linear y Softmax 41
3.1.8 Loss 42

3.2 Language Models are Unsupervised Multitask Learners 43
3.2.1 Ingreso al Decoder 48
3.2.2 Masked Self-Attention 49
3.2.3 Capa Feed Forward Neural Network 53

3.2.4 ¿Por qué una predicción más precisa de la siguiente palabra genera una mayor comprensión? 54
3.2.5 Diferencia entre Transformer Decoder y otras arquitecturas Transformer 55
4. Preentrenamiento de GPT *57*
4.1 Descripción del Dataset 57
4.2 Configuración de Hiperparámetros 59
4.3 Arquitectura Implementada en PyTorch 59
4.4 Evaluación del Rendimiento 59
4.5 Generación de Texto del Modelo 63
4. 6 Escalabilidad del Modelo 64
5. Consideraciones Éticas en Modelos Generativos *66*
5.1 Implicaciones éticas de la Desinformación 66
5.2 Accesibilidad y Potencial de Uso Malicioso 66
5. 3 Riesgos Relacionados con el Modelo Entrenado 66
5.4 Buenas Prácticas para desplegar modelos lingüísticos 67
6. Conclusiones y Recomendaciones *69*
6.1 Conclusiones 69
6. 2 Recomendaciones 69

Índice de Figuras

Figura 1: Neurona Natural y Artificial 12
Figura 2: The Design of an Intelligent Automaton 13
Figura 3: Un programa tradicional 16
Figura 4: Entrenamiento de un modelo de Machine Learning 17
Figura 5: Bucle de entrenamiento 18
Figura 6: Proceso de Descenso de Gradiente 19
Figura 7: Comportamiento de Learning Rate sobre el Loss 21
Figura 8: Visualización de características en un modelo entrenado 24
Figura 9: Red Neuronal 25
Figura 10: Red Neuronal con Dropout 26
Figura 11: Calidad de Traducciones en el estándar WMT newstest2014 30
Figura 12: Arquitectura Transformer 31
Figura 13: Transformer Simplificado 32
Figura 14: Vectores Query, Key y Value 34
Figura 15: Cálculo de Attention 35
Figura 16: MultiHead Attention 36
Figura 17: Embedding + Positional Encoding 37
Figura 18: LayerNorm 39
Figura 19: Iteraciones en el Decoder 40
Figura 20: Linear + Softmax 42
Figura 21: Entrenamiento Transformer 43
Figura 22: Modelo Lingüístico en un teclado 44
Figura 23: Hiperparámetros de arquitectura para los 4 tamaños de modelo 45
Figura 24: Modelos GPT-2 46
Figura 25: Transformer Decoder 47
Figura 26: Token Embedding 48
Figura 27: Positional Encoding 49
Figura 28: Self-Attention vs Masked Self-Attention 50
Figura 29: Puntaje Masked Self-Attention 51
Figura 30: Masked Self-Attention 52
Figura 31: BERT vs GPT 56
Figura 32: Dataset Shakespeare 58
Figura 33: Recursos computacionales requeridos para el entrenamiento 62
Figura 34: Generaciones del modelo 63
Figura 35: Precios promedio de uso GPU H100 71

Índice de Tablas

Tabla 1: Resultados de entrenamiento 60

CAPÍTULO 1

1. Introducción

1.1 Justificación

Los modelos lingüísticos como GPT (Generative Pretrained Transformer) han transformado la forma en que se procesa la información. No obstante, su funcionamiento se basa en una arquitectura compleja que presenta desafíos de comprensión técnica. Este proyecto busca desglosar esa complejidad mediante la construcción de un modelo GPT, explorando paso a paso los fundamentos de la arquitectura Transformer. Al mismo tiempo, se propone aplicar este conocimiento a un caso práctico que demuestre su utilidad, como la generación de texto.

1.1.1 Planteamiento del Problema

Actualmente, el desarrollo de modelos lingüísticos como GPT enfrenta desafíos significativos debido a la complejidad de su diseño y la falta de comprensión detallada de los fundamentos técnicos detrás de su arquitectura. La construcción de un modelo GPT requiere un profundo entendimiento de la arquitectura Transformer, así como de los procesos necesarios para entrenar estos modelos. Sin un enfoque bien estructurado y basado en fundamentos sólidos, la implementación de modelos GPT puede resultar ineficiente y costosa, dificultando la obtención de resultados competitivos y funcionales.

Ante este desafío, resulta necesario elaborar un modelo GPT que integre tanto los principios teóricos como las mejores prácticas para entrenar este tipo de modelos. Esto permitirá no solo mejorar la comprensión de su funcionamiento, sino también facilitar su implementación en diversos contextos aplicativos.

- Esta situación plantea la siguiente pregunta de investigación:
- ¿Cuál es la forma más efectiva de construir un modelo GPT al usar la arquitectura Transformer?
- Las siguientes preguntas secundarias:

- ¿Qué conceptos técnicos esenciales deben considerarse para la implementación básica de un Transformer?
- ¿Qué herramientas específicas son necesarias para el entrenamiento inicial de un modelo GPT?
- ¿Qué fundamentos teóricos y técnicos son esenciales para construir y entrenar un modelo GPT?

1.2 Objetivos

1.2.1 Objetivo General

Desarrollar un modelo GPT basado en la arquitectura Transformer que permita comprender y aplicar los principios fundamentales de su diseño y entrenamiento.

1.2.2 Objetivos Específicos

- Identificar, diseñar e implementar los componentes esenciales del Transformer necesarios para el preentrenamiento inicial del modelo GPT.
- Evaluar el desempeño del modelo preentrenado mediante el Loss sobre los conjuntos de entrenamiento y validación.
- Analizar los dilemas éticos asociados al uso de datos en modelos generativos.

CAPÍTULO 2

2. Antecedentes

Como se expone en el artículo *Attention is All You Need* (Vaswani et al., 2017), la arquitectura Transformer introdujo un enfoque revolucionario para procesar datos secuenciales al eliminar la dependencia de las estructuras recurrentes tradicionales, como RNN (redes neuronales recurrentes). Este modelo se basa en mecanismos de self-attention, permitiendo a la red centrarse en los elementos más importantes de una secuencia sin importar su longitud, mejorando significativamente el paralelismo y la eficiencia computacional.

Desde su publicación, esta arquitectura ha sido la base para avances importantes en modelos lingüísticos como GPT y BERT, consolidándose como el estándar en el procesamiento de lenguaje natural y otras áreas. Naturalmente, la construcción y entrenamiento de un modelo basado en Transformer requieren un profundo entendimiento de los principios planteados en este trabajo, así como de las técnicas de optimización necesarias para adaptarlos a distintas aplicaciones. Esto plantea la necesidad de explorar y aplicar los fundamentos de Transformer en proyectos de investigación y desarrollo, maximizando su potencial en la resolución de problemas complejos.

2.1 Marco Teórico

A diferencia del enfoque de RNN y CNN (redes neuronales convolucionales), Transformer elimina la necesidad de procesar las secuencias de manera secuencial, lo que aumenta significativamente la eficiencia computacional al permitir el cálculo paralelo.

El Deep Learning, según Goodfellow et al. (2016), ha evolucionado a través de avances fundamentales en técnicas de optimización, disponibilidad de grandes volúmenes de datos y la capacidad de aprovechar arquitecturas más complejas gracias a la potencia de hardware moderno,

como las GPU (Graphics Processing Unit) y TPU (Tensor Processing Unit). Transformer aprovecha el poder de cómputo moderno y técnicas de optimización avanzadas.

Por otro lado, Paszke et al. (2019) describen cómo herramientas como PyTorch han sido esenciales en la implementación de modelos de Deep Learning. PyTorch facilita la construcción de arquitecturas y experimentación rápida, lo cual es esencial en proyectos de investigación. Su integración con bibliotecas como Hugging Face y su soporte para el cálculo automático de gradientes han simplificado el desarrollo de modelos de Deep Learning.

Además, la implementación y el preentrenamiento eficiente de un Transformer requiere un entendimiento claro de la evolución histórica de Deep Learning. Según Devlin et al. (2018), los modelos como BERT y GPT logran su rendimiento superior gracias a un preentrenamiento en grandes corpus de datos y fine-tuning en tareas específicas. Este enfoque de entrenamiento en dos etapas ha demostrado ser efectivo para capturar tanto patrones generales del lenguaje como detalles específicos de una tarea particular. Según el artículo original de GPT (Radford, 2018), se menciona lo siguiente: “Se emplea un método para entrenar en dos fases. Primero, modelado lingüístico en datos no etiquetados para aprender los parámetros base de una red neuronal. Posteriormente, ajustar los parámetros a una tarea concreta usando el objetivo supervisado adecuado.” Esto significa que el equipo de investigadores en OpenAI utilizaron dos pasos para entrenar el GPT original, preentrenamiento y fine-tuning. Para este caso se llevará a cabo únicamente la primera fase del entrenamiento de un GPT.

Estas prácticas, junto con un entorno de desarrollo integrado (IDE) como Jupyter Notebooks, Frameworks como PyTorch y el análisis de métricas como el Loss del modelo sobre los datos de entrenamiento y validación, son esenciales para el desarrollo iterativo, especialmente cuando se diseñan modelos complejos.

2.2 Herramientas

2.2.1 PyTorch

El framework PyTorch es ampliamente utilizado para construir, entrenar y evaluar redes neuronales. Permite a los desarrolladores experimentar con arquitecturas complejas y personalizarlas según las necesidades. Entre sus principales características se encuentran el soporte para el cálculo automático de gradientes (autograd) y la facilidad para implementar modelos avanzados como Transformer.

2.2.2 Python

El lenguaje de programación Python es ampliamente utilizado en ciencia de datos, gracias a su sintaxis clara y sus librerías, es el lenguaje principal para desarrollar y entrenar modelos de Deep Learning.

2.2.3 Jupyter Notebooks

Un Jupyter Notebook es un entorno interactivo que permite a los desarrolladores escribir código, documentar procesos y visualizar resultados en un solo lugar. Es ideal para experimentación, ya que facilita la creación y ejecución de prototipos de forma modular, y así permite ajustes rápidos en el flujo de trabajo.

2.2.4 Framework de Deep Learning

Un Framework de Deep Learning es un instrumento diseñado para agilizar el desarrollo, entrenamiento y despliegue de modelos de Deep Learning, facilita tareas complejas como el cálculo de gradientes, optimización y manejo de grandes volúmenes de datos. Los Frameworks se destacan por su flexibilidad, eficiencia y soporte para hardware como GPU. Además,

cuentan con una amplia documentación y comunidades activas, facilitando la cooperación y la respuesta a desafíos en proyectos de Deep Learning.

2.3 Lotería de Hardware

El hardware y el software frecuentemente determinan cuáles ideas de investigación triunfan y cuáles perecen. A esto lo conoce como lotería de hardware, la cual describe cuando una idea de investigación gana, por el hecho de que es adecuada para el software disponible y hardware, no porque la idea es superior a investigaciones alternas. La historia de la informática ilustra como loterías de hardware pueden retrasar progreso al presentar ideas exitosas como fracasos. (Hooker, 2020).

2.4 Historia del Deep Learning

El origen de Deep Learning es una historia de tenacidad por parte de un pequeño grupo de investigadores, entre ellos, Geoffrey Hinton, Yann Lecun, Yoshua Bengio, quienes fueron galardonados con el premio Turing (generalmente considerado como el premio Nobel de Informática).

2.4.1 Neuronal Artificial y Perceptron

En 1943 Warren McCullough, un neurofisiólogo y Walter Pitts, un lógico, se unieron para desarrollar la representación matemática de una neurona artificial. En su artículo “A Logical Calculus of the Ideas Immanent in Nervous Activity” declararon lo siguiente:

Por la naturaleza binaria de la actividad nerviosa, los eventos neuronales y sus vínculos pueden analizarse mediante la lógica proposicional. Se ha demostrado que el funcionamiento de cada red puede explicarse de esta manera.

Aquel trabajo fue influyente y fue retomado por un psicólogo llamado Frank Rosenblatt.

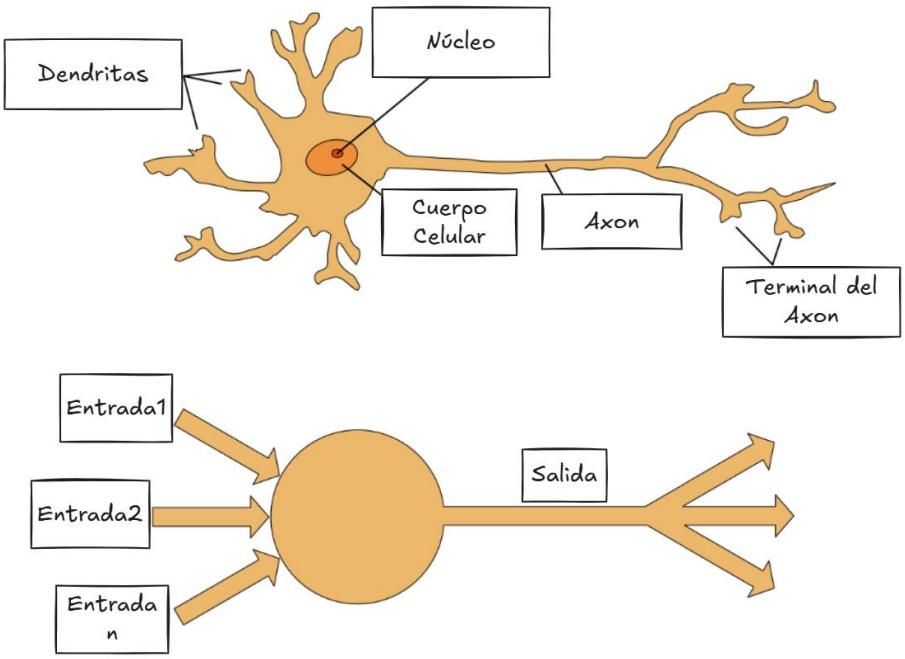


Figura 1: Neurona Natural y Artificial

Fuente: Propia

Rosenblatt avanzó con la investigación de la neurona artificial para que tenga la habilidad de aprender. Utilizó estos principios en su trabajo con el Mark I Perceptron. En "The Design of an Intelligent Automaton", Rosenblatt escribió:

> Está a punto de comenzar el surgimiento de una máquina así: una máquina capaz de captar, distinguir y comprender su medio sin necesidad de entrenamiento, ni control humano. (Rosenblatt, 1958)



Vol. VI, No. 2, Summer 1958

research trends

CORNELL AERONAUTICAL LABORATORY, INC., BUFFALO 21, NEW YORK

The Design of an

Intelligent AUTOMATON

by FRANK ROSENBLATT

Introducing the perceptron — A machine which senses, recognizes, remembers, and responds like the human mind.

STORIES about the creation of machines having human qualities have long been a fascinating province in the realm of science fiction. Yet we are now about to witness the birth of such a machine — a machine capable of perceiving, recognizing, and identifying its surroundings without any human training or control.

Development of that machine has stemmed from a search for an understanding of the physical mechanisms which underlie human experience and intelligence. The question of the nature of these processes is at least as ancient as any other question in western science and philosophy, and, indeed, ranks as one of the greatest scientific challenges of our time.

Our understanding of this problem has gone perhaps as far as had the development of physics before Newton. We have some excellent descriptions of the phenomena to be explained, a number of interesting hypotheses, and a little detailed knowledge about events in the nervous system. But we lack agreement on any integrated set of principles by which the functioning of the nervous system can be understood.

We believe now that this ancient problem is about to yield to our theoretical investigation for three reasons:

First, in recent years our knowledge of the functioning of individual cells in the central nervous system has vastly increased.

Second, large numbers of engineers and mathematicians are, for the first time, undertaking serious study of the mathematical basis for thinking, perception, and the handling of information by the central nervous system, thus providing the hope that these problems may be within our intellectual grasp.

Third, recent developments in probability theory and in the mathematics of random processes provide new tools for the study of events in the nervous system, where only the gross statistical organization is known and the precise cell-by-cell "wiring diagram" may never be obtained.

Receives Navy Support

In July, 1957, Project PARA (Perceiving and Recognizing Automaton), an internal research program which had been in progress for over a year at Cornell Aeronautical Laboratory, received the support of the Office of Naval Research. The program had been concerned primarily with the application of probability theory to

Figura 2: The Design of an Intelligent Automaton

Fuente: (Cornell Aeronautical Laboratory, 1958)

No mucho tiempo después un profesor del Instituto Tecnológico de Massachusetts llamado Marvin Minsky escribió un libro llamado Perceptrons (MIT Press) acerca del invento de Rosenblatt. En el libro se demostró que una sola capa de Perceptrons no era capaz de aprender algunas funciones matemáticas simples pero críticas (como la función XOR). Como resultado, la comunidad académica mundial abandonó casi por completo las redes neuronales durante las siguientes dos décadas.

El avance más significativo en redes neuronales en las últimas cinco décadas es Parallel Distributed Processing publicado en 1986 por el MIT Press. Los autores afirmaron que su enfoque estaba más cerca que otros Frameworks de cómo funciona el cerebro.

Los pasos propuestos por Parallel Distributed Processing son muy similares a los pasos utilizados hoy en día en redes neuronales. El libro definió el Parallel Distributed Processing como un proceso que requiere lo siguiente:

- Un grupo de unidades de procesamiento.
- Una condición de activación.
- Una función que genere la salida de cada unidad.
- Un esquema de enlace entre unidades
- Una norma de propagación para transmitir patrones de actividad a lo largo de la red.
- Una norma de activación que integra las entradas recibidas por una unidad con su estado actual para generar una salida.
- Una norma de aprendizaje que ajusta los patrones de conexión a partir de la experiencia.
- Un ambiente donde el sistema debe desempeñarse.

Las redes neuronales como se las aplican hoy en día manejan cada uno de estos pasos.

En los años 80, la mayor parte de los modelos se construyeron con una segunda capa de neuronas, evitando así el problema que había identificado Minsky. Pero por problemas de poder de cómputo nuevamente estos problemas teóricos resultaron conteniendo el avance dentro de Deep Learning. En teoría, añadir una sola capa extra de neuronas era suficiente para permitir que cualquier función matemática se aproximara con estas redes neuronales, pero en la práctica, dichas redes a menudo eran demasiado grandes y lentas para ser útiles.

Las redes neuronales finalmente están alcanzando su máximo potencial gracias al uso de más capas, sumado a la capacidad de hacerlo gracias a mejoras en el hardware, mayor disponibilidad de datos y ajustes algorítmicos que permiten entrenarlas con mayor rapidez y facilidad.

2.4.2 Red Neuronal Profunda

Muchos algoritmos que actualmente se utilizan en Deep Learning fueron ideas que no se han originado recientemente, un ejemplo es backpropagation usado en redes neuronales, inventado por Paul Werbos en 1974, una técnica usada universalmente en redes neuronales.

Desde la década de 1990, el Deep Learning ha sido aplicado con éxito en diversas aplicaciones comerciales; sin embargo, hasta hace poco tiempo se consideraba más un arte reservado para especialistas que una tecnología accesible. Si bien es cierto que se requiere cierto nivel de habilidad para obtener un buen desempeño de un algoritmo de Deep Learning, esta dificultad tiende a disminuir a medida que aumenta la cantidad de datos de entrenamiento disponibles.

2.4.3 ¿Qué es machine learning?

Deep Learning es el área moderna de una disciplina más general llamada Machine Learning.

Machine Learning es como la programación regular, una manera de hacer que las computadoras puedan completar una cierta tarea. Pero la pregunta que Machine Learning responde es ¿cómo utilizar la programación regular para reconocer objetos en imágenes? Reconocer objetos en una foto es una tarea un poco confusa ya que se debe programar los pasos para reconocer objetos en imágenes. Es una pregunta que realmente es difícil responder ya que toda esta actividad ocurre en el cerebro sin tener conciencia de aquello.



Figura 3: Un programa tradicional

Fuente: Propia

Para inicios de la década de los cincuenta, Arthur Samuel, investigador de IBM, empezó a trabajar en el Machine Learning como un nuevo paradigma de programación. En su ensayo Artificial Intelligence: A Frontier of Automation, destaca lo siguiente:

> Programar una computadora para tales cálculos es, en el mejor de los casos, una tarea difícil, no principalmente por la complejidad inherente a la computadora en sí, sino más bien por la necesidad de explicar cada paso del proceso con el detalle más exasperante. Las computadoras, como cualquier programador te dirán, son unos imbéciles gigantes, no cerebros gigantes. (Samuel, 1962, pp. 12-13)

La idea que quiso emitir en su ensayo es: en vez de explícitamente decir a la computadora que pasos tomar para completar una tarea, es necesario mostrar ejemplos de cómo resolver, y que progresivamente pueda encontrar la solución por si sola. Una parte importante de su ensayo relato:

> Supongamos que implementamos algún método automático para probar la efectividad de cualquier asignación de peso actual en términos del rendimiento real y proporcionamos un mecanismo para modificar la asignación de peso y maximizar el rendimiento. No es necesario entrar en detalles sobre dicho procedimiento para ver que podría automatizarse por completo y que una máquina así programada "aprendería" de su experiencia. (Samuel, 1962, pp. 12-13)

Esta idea sigue siendo empleada hoy en día en Machine Learning, un modelo es un programa especial, el cual puede hacer muchas cosas, dependiendo de sus pesos.

Samuel menciona “un método automático para probar la efectividad de cualquier asignación de peso actual en términos del rendimiento real”, el rendimiento actual hace referencia a que tan bueno es el rendimiento del modelo.

Y finalmente menciona “un mecanismo para alterar la asignación de los pesos y maximizar el rendimiento”, esto quiere decir ajustar los pesos en la dirección ganadora.

Así, el proceso de aprendizaje automático se volvería completamente automático cuando el ajuste de los pesos también lo fuera: cuando, en lugar de mejorar un modelo ajustando sus pesos manualmente, se necesita de un mecanismo automatizado que generara ajustes según el rendimiento.

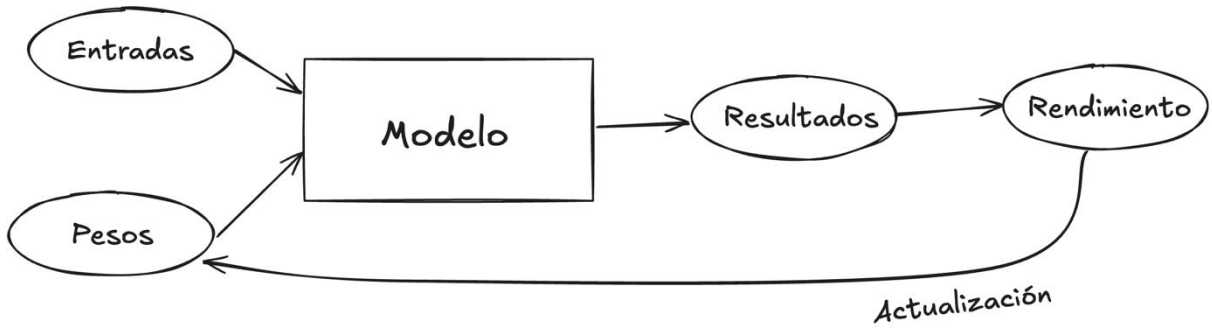


Figura 4: Entrenamiento de un modelo de Machine Learning

Fuente: Propia

2.4.4 Red Neuronal Vanilla

Para resumir en una sola frase lo que son las redes neuronales: Una red neuronal es una función matemática extremadamente flexible que, variando sus pesos, puede aproximar cualquier función.

La manera de actualizar los pesos de una red neuronal se llama Descenso de Gradiente.

Las redes neuronales se distinguen por lo bien que se adaptan a diferentes tareas, lo que esto quiere decir que son capaces de resolver distintos problemas solo con ajustar bien sus pesos. Esto es importante, ya que el descenso de gradiente ayuda a encontrar esos valores automáticamente.

Las ideas de Samuel Arthur son vigentes hasta el día de hoy, para actualizarnos a la terminología actual, es necesario entender lo siguiente:

- La forma funcional de un modelo se llama arquitectura.
- Los pesos se denominan parámetros.
- Los resultados de un modelo se llaman predicciones.
- Medida de rendimiento se llama Loss.
- El Loss depende de las predicciones correctas.

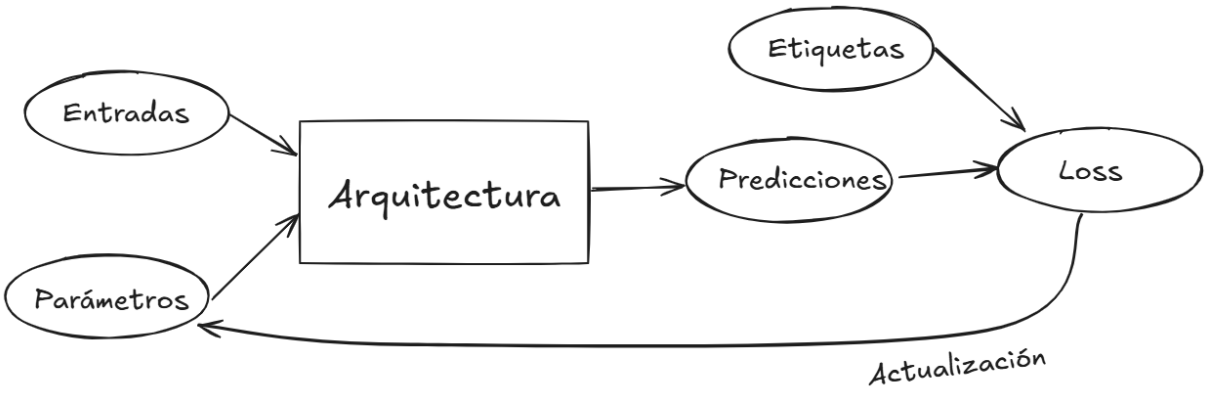


Figura 5: Bucle de entrenamiento

Fuente: Propia

La red neuronal se representa gráficamente como una combinación conectada de unidades organizadas lógicamente en una o más capas. La salida de cada unidad es el resultado de la operación matemática escrita dentro de una neurona. Cada unidad posee sus propios parámetros. La unidad aplica una función de activación y obtiene el valor de salida, un número real. En una FFNN (Feed Forward Neural Network) estándar, el valor de salida de una unidad en una capa

alimenta como entrada a cada unidad de la capa siguiente. Las redes con más de una capa oculta se denominan redes neuronales profundas.

La razón por la cual se usan funciones de activación sobre cada neurona es porque se agrega el componente no lineal que permite a la red neuronal aproximarse a funciones no lineales. Si no hay no linealidades, FFNN sería lineal, independientemente de su número de capas.

Las opciones más populares de funciones de activación son Sigmoid, Softmax y ReLU (rectified linear unit). Cada función de activación tiene su propio uso Softmax es muy utilizado en clasificación múltiple, ReLU y Sigmoid en clasificación binaria.

Las capas que no corresponden ni a la capa de entrada ni a la de salida se denominan capas ocultas.

2.4.5 Descenso de Gradiente

El algoritmo consta de los siguientes pasos:

1. Inicializar los pesos.
2. Por cada imagen, utilizar los pesos para predecir si es un tres o un seis.
3. En base a las predicciones, calcular el Loss.
4. Calcular el gradiente, que mide, para cada parámetro, como cambiaría el Loss al cambiarlo.
5. Ajustar los parámetros según aquel cálculo.
6. Repetir los pasos 2–5 hasta que se decida detener el entrenamiento.

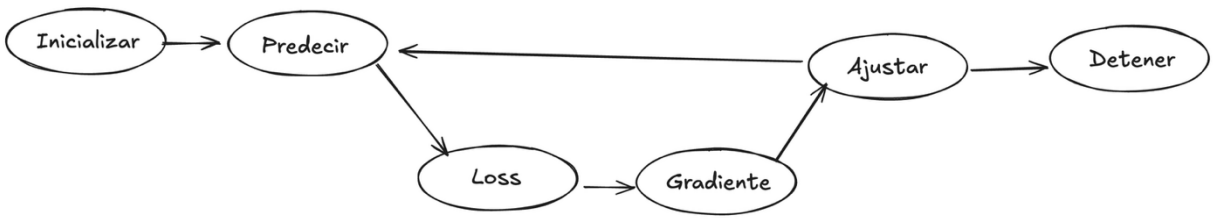


Figura 6: Proceso de Descenso de Gradiente

Fuente: Propia

Para aclarar los pasos:

Cuando se entrena desde cero un modelo se debe inicializar los parámetros con valores aleatorios para empezar la fase de aprendizaje de la red neuronal. El Loss indica la efectividad de la asignación de los parámetros actual en términos de rendimiento real de la red. Un Loss bajo significa que el rendimiento del modelo es bueno, un Loss alto significa lo contrario. El propósito del Loss es evaluar el error entre las predicciones y los valores reales. El ajuste al que se hace referencia sobre los parámetros de la red es incrementar o decrementar los parámetros de la red por una pequeña cantidad para ver si el Loss sube o baja. El cálculo diferencial permite determinar directamente en qué dirección y aproximadamente en qué medida cambiar cada peso. El gradiente se multiplica por un número llamado learning rate, el cual controla el tamaño de cada actualización. Luego, iterar hasta llegar al punto más bajo, que será la meta final; entonces detener. El número de iteraciones al entrenar el modelo está especificado por cuantos epochs entrenar el modelo.

Decidir cómo cambiar los parámetros en función de los valores de los gradientes representa un paso importante dentro de Deep Learning. El learning rate es típicamente un número entre 0.001 y 0.1.

Para cada ejemplo de entrenamiento, se actualizan los parámetros usando las derivadas parciales.

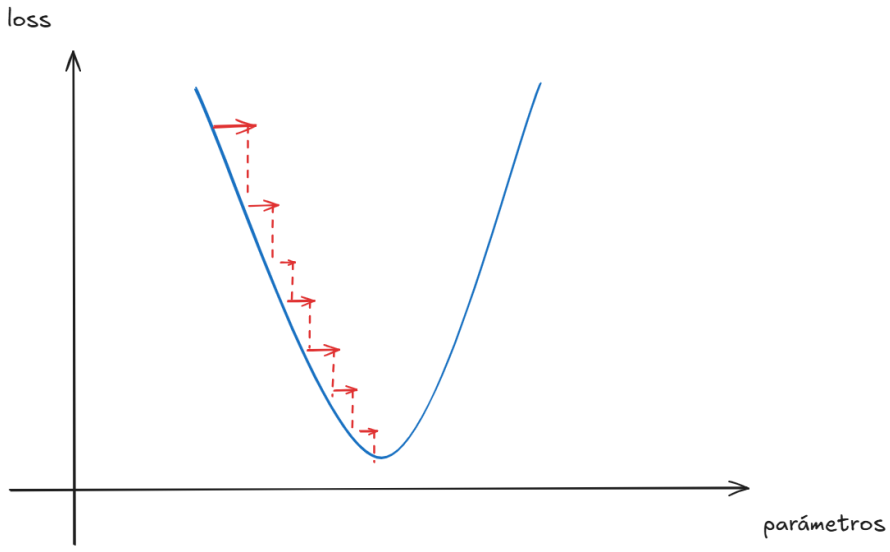


Figura 7: Comportamiento de Learning Rate sobre el Loss

Fuente: Propia

Los gradientes dirán cuánto cambiar cada parámetro para mejorar el modelo. El cálculo proporciona un atajo computacional, la derivada, que permite calcular directamente los gradientes de las funciones.

El objetivo al entrenar una red neuronal es encontrar parámetros que minimicen el Loss. El Loss que dependen de miles de millones de parámetros.

2.4.6 ¿Qué es backpropagation?

El algoritmo backpropagation es el componente clave que da vida al aprendizaje en las redes neuronales. Para actualizar los valores de los parámetros en redes neuronales. Este algoritmo es eficiente para calcular gradientes en una red neuronal mediante la regla de la cadena del cálculo.

En el núcleo de backpropagation se encuentra la derivada parcial $\partial L/\partial w$ del Loss con respecto a los parámetros en la red. Esta expresión indica la rapidez con la que cambia Loss al modificar los parámetros.

La derivada de una función f es un valor que describe qué tan rápido f crece (o disminuye). El gradiente de una función es un vector de derivadas parciales. El descenso de gradiente realiza el cálculo de la derivada parcial para cada parámetro

Lo inteligente de backpropagation es que permite calcular simultáneamente todas las derivadas parciales, con un paso hacia adelante a través de la red, seguido de un paso hacia atrás a través de la red. Por lo tanto, el costo total de backpropagation es aproximadamente el mismo que el de realizar solo dos pasadas hacia adelante a través de la red.

2.4.7 ¿Qué ocurre dentro de las capas ocultas de la red neuronal?

Las nuevas técnicas de Deep Learning han permitido entrenar redes mucho más profundas (y de mayor tamaño). Ahora las redes profundas son entrenadas con redes que poseen entre 5 y 10 capas ocultas. Y resulta que estas funcionan mucho mejor en problemas que las redes de poca profundidad, es decir, redes con una sola capa oculta.

Mientras más capas tiene una red neuronal, es capaz de responder más preguntas. En una aplicación de reconocimiento facial, el producto final sería una red que descompone una interrogante (¿muestra la imagen un rostro o no?) en interrogantes simples a nivel de píxeles individuales. Esto es posible gracias a una serie de múltiples capas: las primeras responden a preguntas simples acerca de la imagen de entrada, mientras que las capas siguientes forman una jerarquía de conceptos cada vez más complejos.

Muchos investigadores perciben que los resultados de Deep Learning se comportan como cajas negras impenetrables. En el 2013 en el artículo (Zeiler & Fergus, 2013), mostraron como visualizar los parámetros aprendidos por cada capa en una red neuronal.

Zeiler et al. revelaron que las características están lejos de ser patrones aleatorios e interpretables. Más bien, muestran muchas propiedades intuitivamente deseables, como la composicionalidad y la discriminación de clases a medida que se asciende en las capas de una red neuronal profunda.

Las proyecciones de cada capa muestran la naturaleza jerárquica de las propiedades de la red. La capa 1 responde a líneas diagonales. La capa 2 responde a las esquinas y otras conjunciones de aristas y colores. La capa 3 presenta invariancias más complejas, capturando texturas similares. La capa 4 muestra una variación significativa, pero es más específica de cada clase: caras de perro y patas de pájaro. La capa 5 muestra objetos completos con una variación significativa de pose, por ejemplo, teclados y perros.

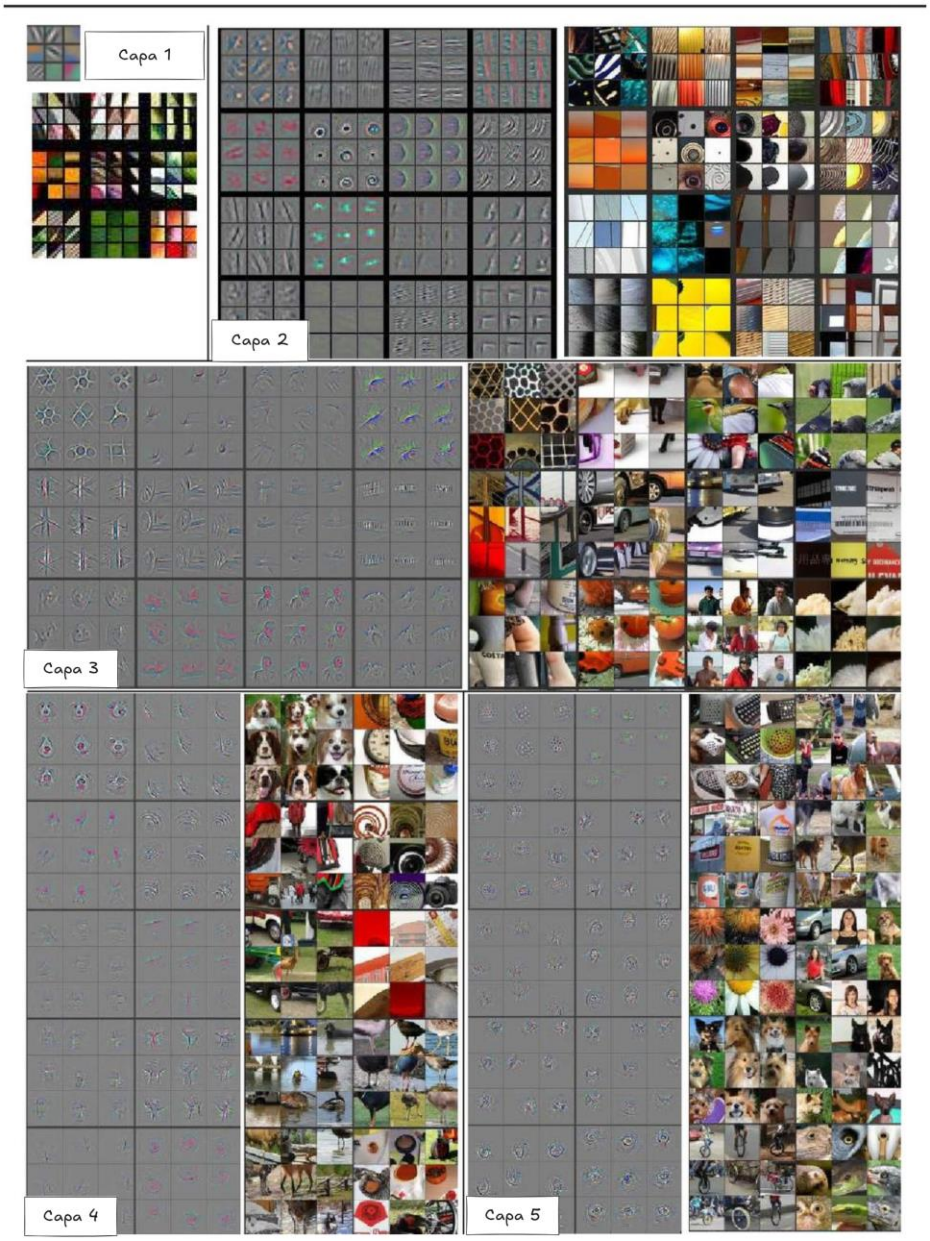


Figura 8: Visualización de características en un modelo entrenado

Fuente: (Zeiler et al., 2013)

2.4.8 ¿Qué es Dropout?

En (Srivastava et al., 2014) se introdujo la técnica, que consiste en lo siguiente:

Se comienza eliminando aleatoriamente (y temporalmente) las neuronas ocultas en la red, las neuronas de entrada y salida se mantienen intactas.

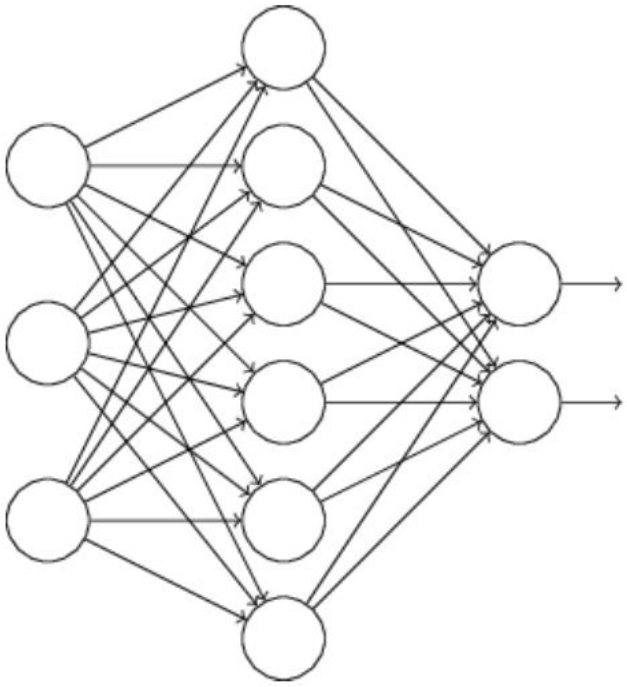


Figura 9: Red Neuronal

Fuente: (Nielsen, 2015)

Es importante tener en cuenta que las neuronas afectadas por dropout, es decir, las neuronas que se han eliminado temporalmente siguen estando ocultas:

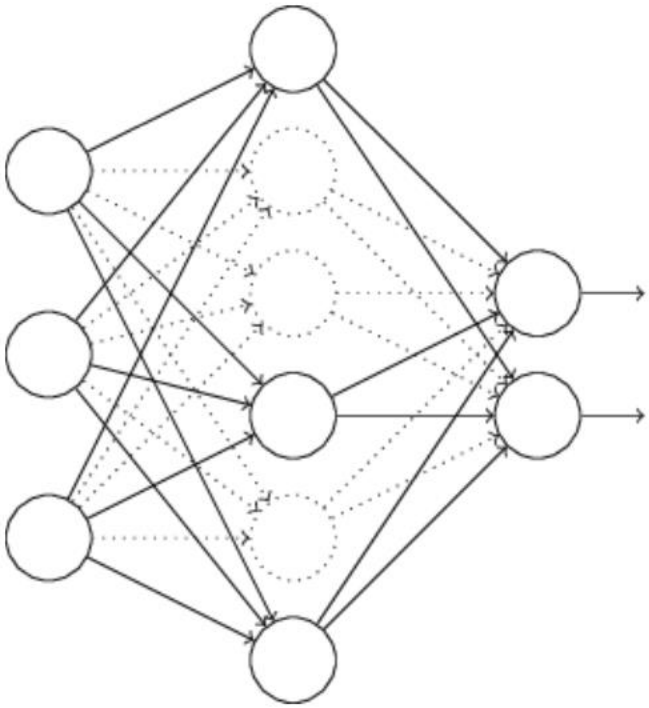


Figura 10: Red Neuronal con Dropout

Fuente: (Nielsen, 2015)

Luego repetimos el proceso: primero se restauran las neuronas descartadas, luego se selecciona un nuevo subconjunto aleatorio de neuronas ocultas para eliminar, se estima el gradiente para un grupo diferente y se actualizan los parámetros en la red. Al repetir este proceso una y otra vez, la red aprenderá un conjunto de parámetros. Por supuesto, estos parámetros se habrán aprendido en condiciones en las que se descartaron la mitad de las neuronas ocultas. Esto le obliga a la red neuronal a aprender conceptos en condiciones de alta dificultad, lo que le brinda una característica de robustez.

El principio general es ampliar los datos de entrenamiento aplicando operaciones que reflejen la variación del mundo real. Por ejemplo, en reconocimiento de voz, los humanos pueden reconocer el habla incluso con distorsiones como el ruido de fondo. Por lo tanto, ampliar los datos añadiendo ruido de fondo. También los humanos son capaces de reconocer el discurso sí está acelerado o ralentizado. Esta es otra forma de ampliar los datos de entrenamiento.

2.4.9 Teorema de Aproximación Universal

Un hecho sobre las redes neuronales es que, pueden aproximarse a cualquier función. No importa cuál sea la función, está garantizado que existe una red neuronal que pueda calcular dicha función.

Esto indica que las redes neuronales poseen universalidad. Esto se conoce como el teorema de aproximación universal. Esto se aplica incluso si se limita una red neuronal a una sola capa oculta situada entre las neuronas de entrada y salida. Por lo tanto, esto quiere decir que las arquitecturas de red más simples pueden llegar a ser extremadamente potentes, aquellas que poseen más capas ocultas y redes más profundas pueden llegar a calcular funciones muy complejas.

La capacidad de calcular una función arbitraria es una habilidad notable que poseen las redes neuronales. Casi cualquier proceso imaginable puede considerarse como cálculo de funciones. Traducir texto en mandarín al español, aquello puede ser considerado como calcular una función. O tomar un documento y generar un resumen de lo que trata aquel documento, nuevamente esto se puede considerar como un tipo de cálculo de función. Universalidad significa que, en principio, las redes neuronales pueden hacer todas estas cosas y mucho más.

Es necesario explicar las limitaciones de las redes neuronales respecto a la afirmación de que ‘pueden calcular cualquier función’.

Primeramente, no significa explícitamente que una red neuronal puede ser utilizada para calcular de manera exacta cualquier función. Más bien, puede conseguir una aproximación que es muy cercana a la función en sí. Al incrementar el número de neuronas la aproximación mejora.

Otra limitación que existe en aquella declaración es que el tipo de funciones las cuales pueden ser aproximadas con redes neuronales de la manera que se ha visto son las funciones de tipo continua.

Si una función es discontinua por ejemplo si toma saltos repentinos o bruscos, entonces es muy probable que no será posible aproximarse al usar una red neuronal.

CAPÍTULO 3

3. Análisis Simplificado de Artículos

3.1 Attention Is All You Need

En Deep Learning se obtienen resultados notables al procesar datos de entrada y obtener su salida final, sin embargo, muchos critican que los modelos resultantes actúan como una "caja negra" impenetrable (predicen, pero no se comprende su funcionamiento). Existe una amplia investigación que muestra cómo examinar en profundidad los modelos de Deep Learning y extraer información valiosa de ellos. Dicho esto, comprender por completo todos los tipos de modelos de machine learning puede ser difícil, especialmente si se considera cómo se comportarán al encontrarse con datos muy diferentes a los utilizados para entrenarlos.

La historia del Deep Learning documenta distintas arquitecturas emergentes para diversas modalidades (audio, visión, texto), cada una especializada en su dominio. En los últimos años se ha visto una convergencia hacia una única arquitectura, el Transformer. El Transformer puede procesar imágenes, video, texto y audio; sus resultados indican que es una arquitectura de propósito general, entrenable y muy eficiente en hardware actual.

El artículo titulado «Attention Is All You Need» publicado en 2017 por un grupo de investigadores en Google Brain presenta Transformer, que se apoya únicamente en mecanismos de atención y elimina la necesidad de redes neuronales recurrentes o convolucionales.

El artículo examina el papel de las RNN en tareas de traducción. Sin embargo, estas presentan deficiencias en procesamiento secuencial y paralelización limitada, especialmente con secuencias largas. La naturaleza secuencial de las redes neuronales recurrentes también hace que sea más difícil aprovechar al máximo el hardware moderno.

El Transformer supera a los modelos de redes neuronales recurrentes y convolucionales en traducción académica del inglés al alemán y del inglés al francés. Aparte de una mejora en la

calidad de traducción, demanda menos cómputo para su entrenamiento y es mucho más compatible con el hardware, acelerando el entrenamiento hasta en un orden de magnitud.

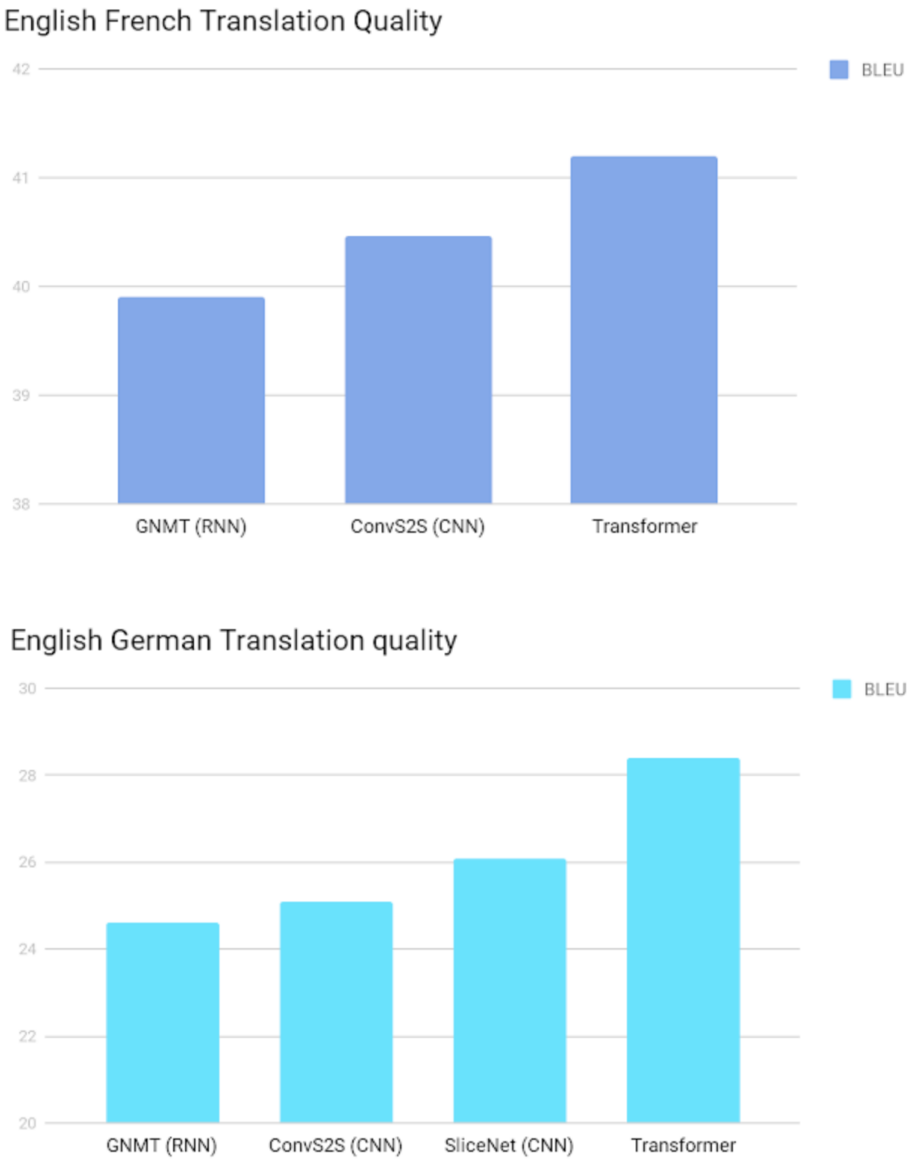


Figura 11: Calidad de Traducciones en el estándar WMT newstest2014

Fuente: (Uszkoreit, 2017)

El artículo muestra como los investigadores realizan experimentos en tareas de traducción. El Transformer logra resultados SOTA (state of the art) en las tareas de traducción desde inglés hacia alemán y francés, con un tiempo de entrenamiento significativamente menor.

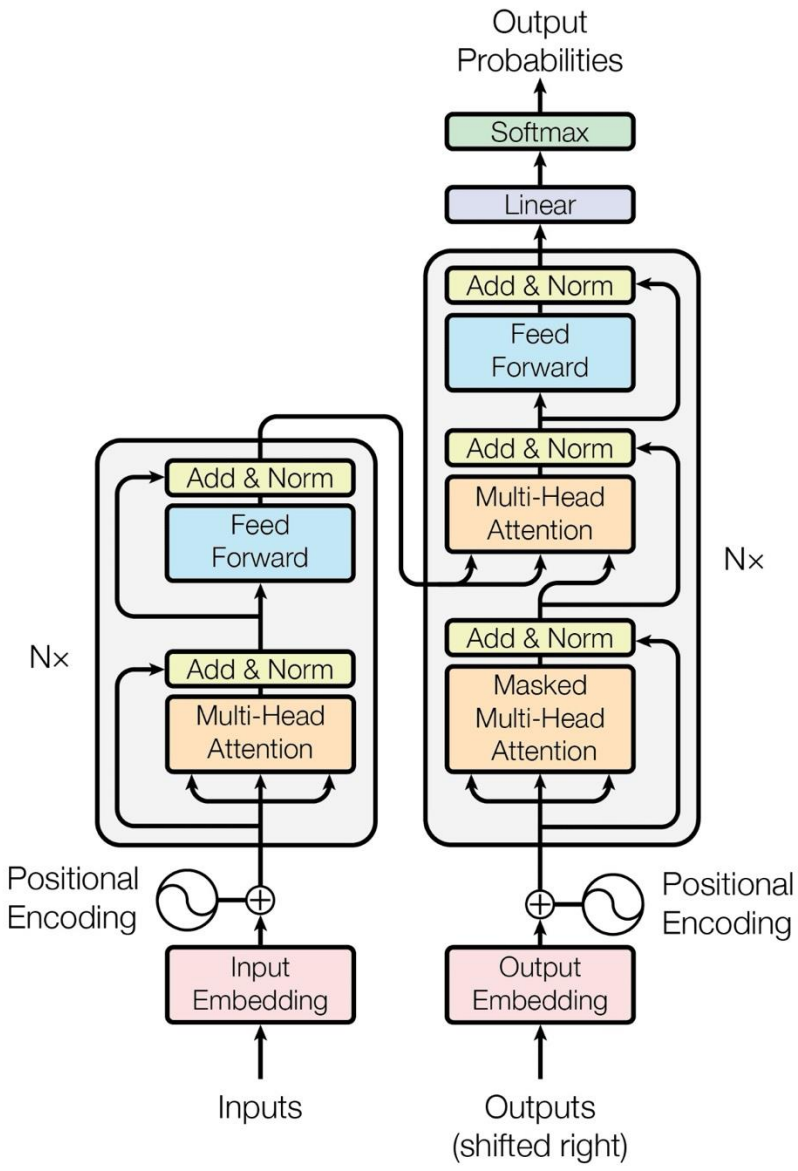


Figura 12: Arquitectura Transformer

Fuente: (Vaswani et al., 2017)

Transformer toma una frase en un idioma y encuentra su correspondiente traducción en otro. Para ilustrar con un ejemplo se muestra como funcionaría el Transformer si se desea traducir del idioma inglés al español. En la imagen existen dos componentes, el encoder y el decoder. En el artículo, el encoder consta de seis capas (encoders) y el decoder también consta de seis capas (decoders).

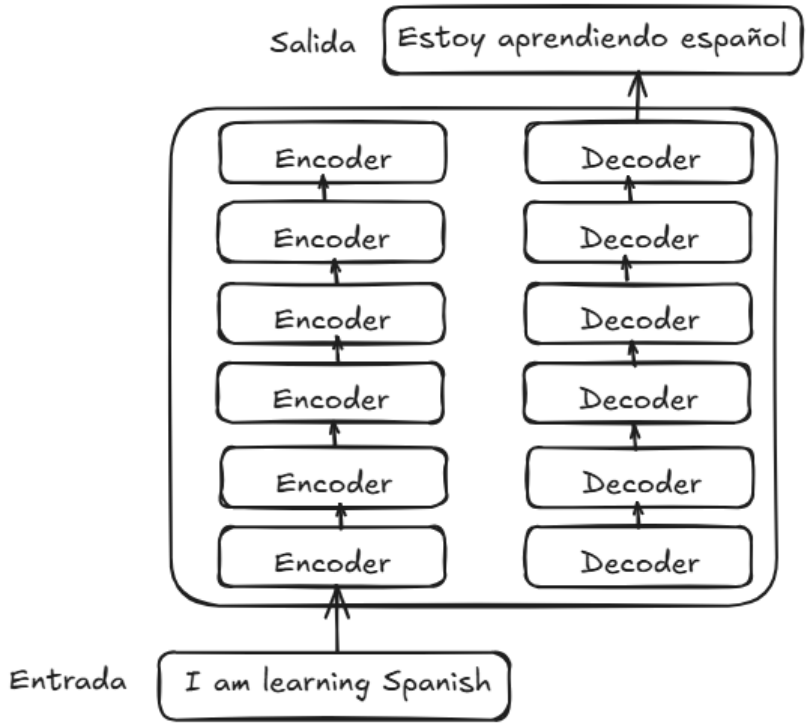


Figura 13: Transformer Simplificado

Fuente: Propia

3.1.1 Encoder

Primero para poder procesar datos en la arquitectura, convertimos las palabras en vectores de tamaño 512 mediante el uso de un algoritmo de embedding, el embedding intenta representar una palabra con un vector. El embedding únicamente ocurre en el primer encoder.

Todos los encoders son idénticos en su estructura, no comparten los mismos pesos, cada uno se compone de una capa de Self-Attention y Feed Forward Neural Network. En esta capa no hay bucles en la red: la información siempre se transmite hacia adelante, nunca hacia atrás.

Cada palabra en el encoder fluye a través de su propio camino. Existen dependencias entre estas rutas en self-attention. Mientras que en la capa FFNN no posee estos vínculos, por lo que los distintos caminos pueden ejecutarse de manera simultánea durante su recorrido

3.1.2 Self-Attention

En este caso la siguiente oración es la que se busca traducir es: “El niño no cruzó la calle porque estaba muy cansado”. ¿Quién estaba muy cansado? ¿Hace referencia al niño o a la calle? Es una interrogante fácil de responder para una persona, pero no lo es en el caso de un algoritmo. Una vez que el modelo procesa “estaba”, self-attention logra asociar “estaba” con “niño”.

Como propone Vaswani et al. (2017), la atención es calculada usando la ecuación:

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

En donde:

Q representa el vector de queries.

K representa el vector de keys.

V representa el vector de values.

d_k representa las dimensiones de los keys.

QK^T representa el producto escalar entre queries y keys.

softmax representa la normalización para obtener distribuciones de attention.

El attention es una ecuación que es calculada en seis pasos.

El primer paso es por cada palabra crear tres nuevos vectores: query, key y value. Estos vectores son creados al ser multiplicados por el embedding de cada palabra con tres matrices que son

creadas mientras se entrena el modelo Wq, Wk y Wv. Los vectores resultantes query, key y value son vectores de tamaño 64.

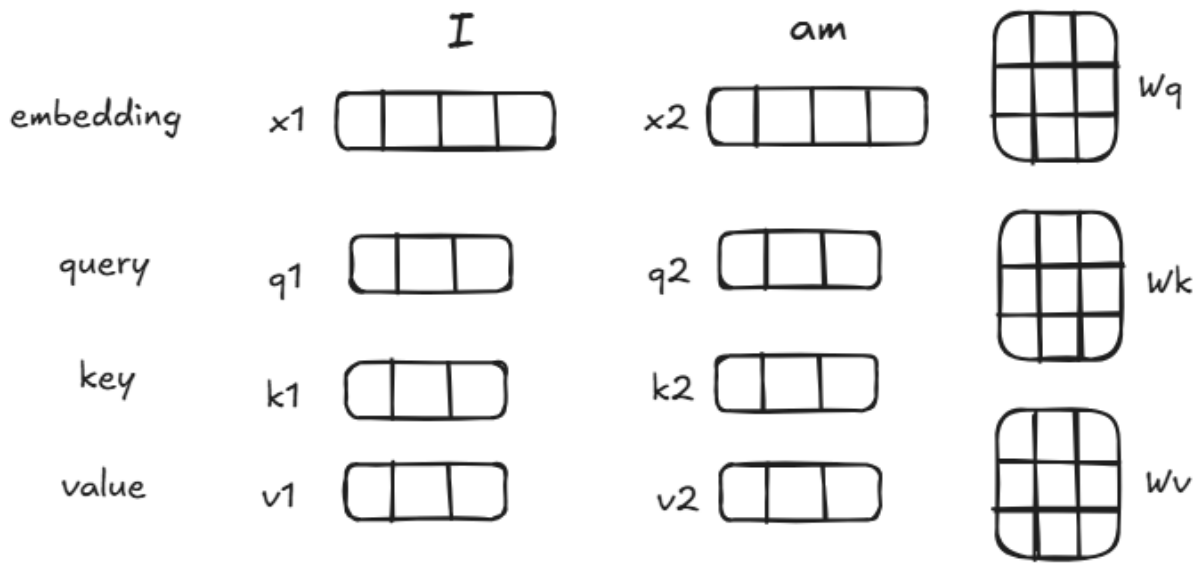


Figura 14: Vectores Query, Key y Value

Fuente: Propia

El segundo paso es conseguir un puntaje por cada palabra, por ejemplo, si se desea conseguir el puntaje de la primera palabra “I” hay que puntuar cada palabra de la entrada por esta palabra. Este puntaje es obtenido mediante la multiplicación escalar del vector query de la primera palabra y el vector key de cada palabra. Entonces el puntaje de la primera palabra corresponde a la multiplicación entre q1 y k1, el segundo puntaje es q1 y k2.

El tercer y cuarto paso es dividir estos puntajes por 8 (la raíz cuadrada del tamaño del vector key) y pasar el resultado por Softmax. Softmax ajusta los puntajes para que todo sea positivo y sume a 1.

El quinto paso es multiplicar el vector value por el resultado de haber aplicado Softmax.

El último paso es sumar todos los vectores en un único vector. Esto produce un vector Z como resultado de self-attention en esta primera palabra.

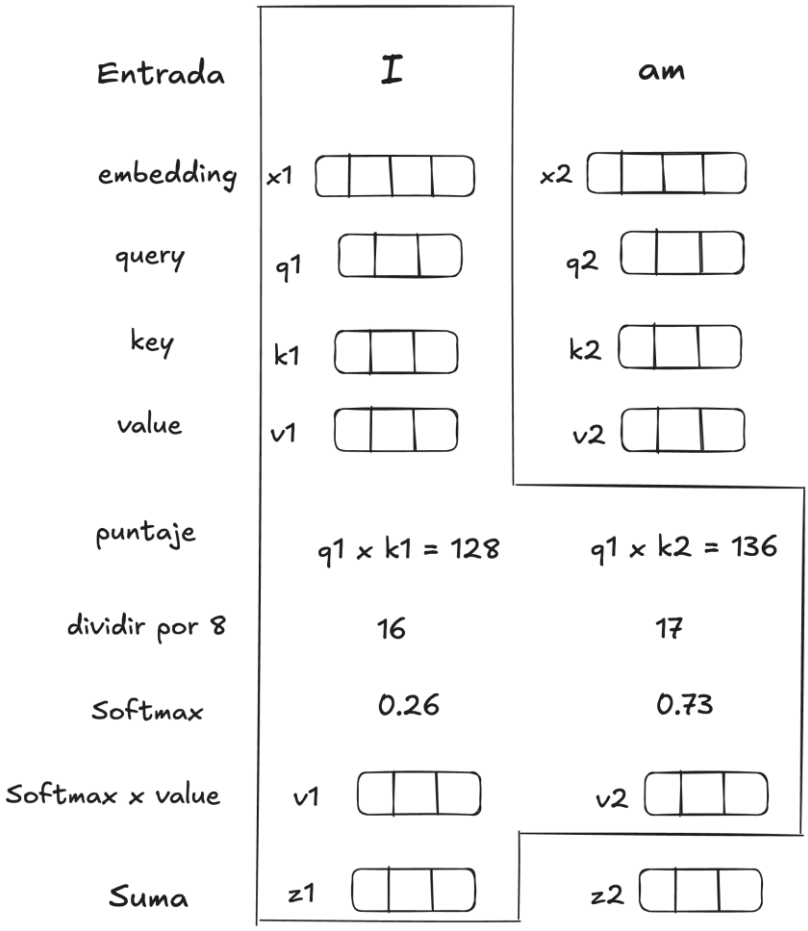


Figura 15: Cálculo de Attention

Fuente: Propia

El vector resultante se envía a la capa FFNN.

El artículo perfeccionó aún más la capa de self-attention añadiendo un mecanismo denominado MultiHead attention. Esto mejora el desempeño de attention en dos maneras:

- Ayuda al modelo a centrarse en diferentes palabras de una oración, lo cual resulta útil para comprender el contexto, como a qué hace referencia "estaba".

- Permite que la capa de attention aprenda desde múltiples perspectivas (el artículo original utiliza 8 diferentes cabezas de attention, 8 por cada encoder/decoder) mediante el uso de varios conjuntos de pesos (multicabezas), cada uno de los cuales captura diferentes características de la entrada.

Básicamente es la misma intuición a lo que se ha hecho, pero se mantiene vectores query, key, value separados para cada cabezal, lo que genera distintos vectores query, key y value.

Si se hace el mismo cálculo, se produce 8 vectores Z, pero la FFNN requiere uno solo. Para condensarlos, se multiplican por una matriz WO, generando un único vector que resume la información de los ocho anteriores. Este único vector ahora se utiliza para poder alimentarla a la capa FFNN.

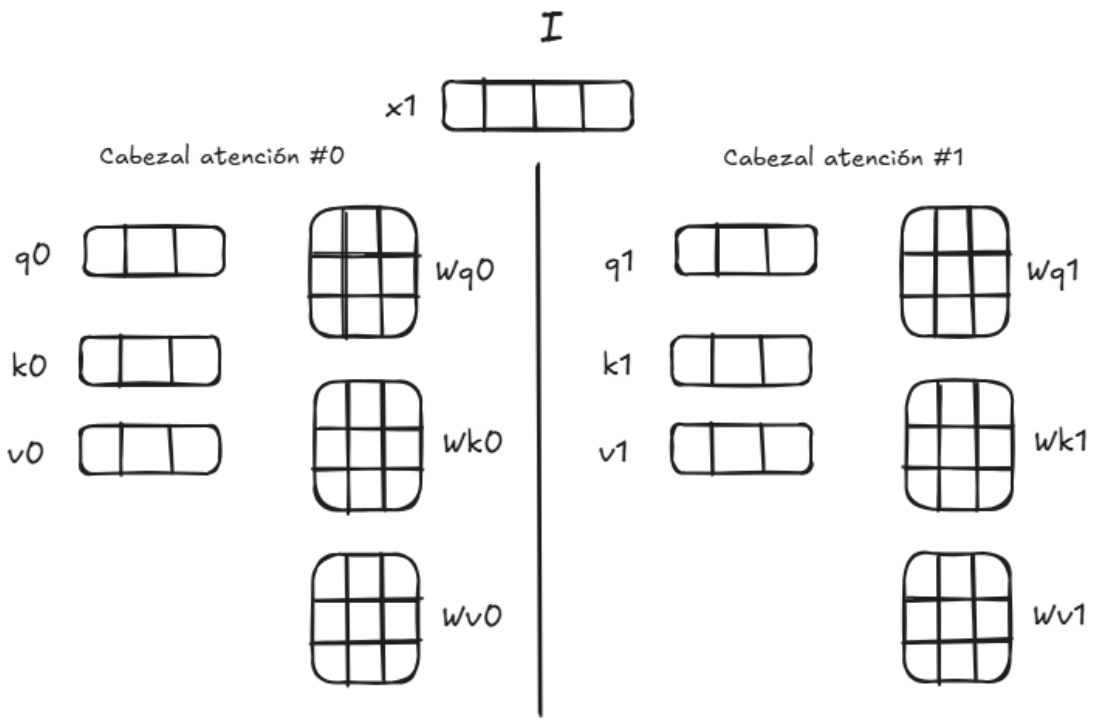


Figura 16: MultiHead Attention

Fuente: Propia

3.1.3 Positional Encoding

El Positional Encoding trata de proporcionar al modelo una idea del orden de las palabras.

Transformer define un vector a cada embedding. Estos vectores presentan un patrón definido que el modelo logra captar, lo que le permite identificar la ubicación de cada palabra. La idea principal es que al sumar estos valores a los embeddings ofrece separaciones relevantes entre los embeddings tras ser transformados en vectores query, key, value y durante el attention del producto escalar.

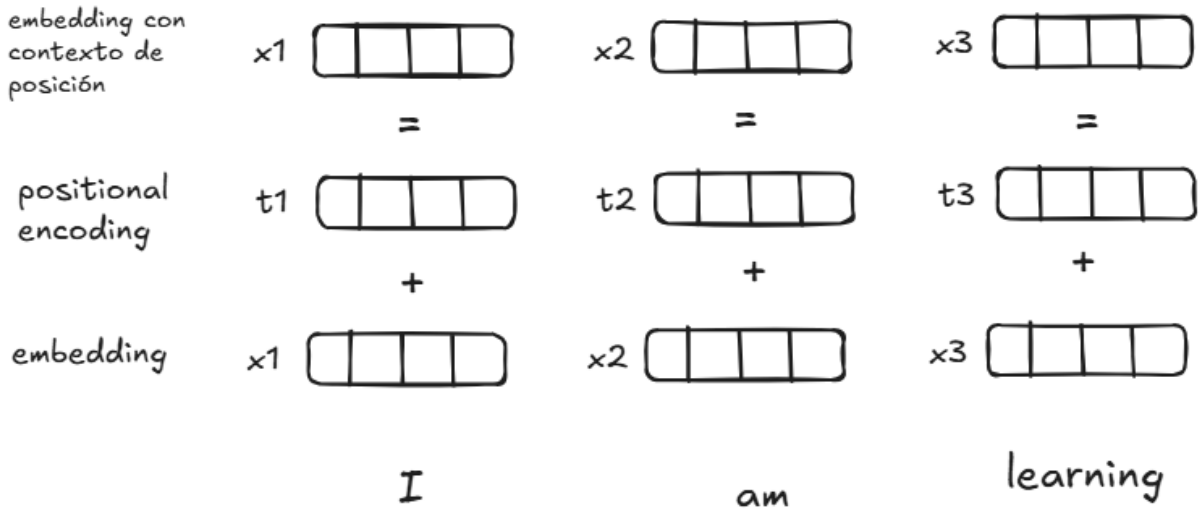


Figura 17: Embedding + Positional Encoding

Fuente: Propia

El vector de positional encoding posee las mismas dimensiones que cada embedding, 512, con valores entre –1 y 1 en cada vector.

3.1.4 Layer Normalization

En cada capa (self-attention y FFNN) dentro del encoder y decoder existe una conexión residual, llamada Layer Normalization.

La técnica propuesta en (Ba et al., 2016) ayuda a mantener los valores de la red neuronal balanceados, toma los valores y los normaliza (para que sean valores no muy grandes y no muy pequeños) eso ayuda a que los valores se mantengan dentro de un mismo rango.

Normalizar los datos no es un requisito estricto. Sin embargo, en la práctica, puede aumentar la velocidad de aprendizaje de una red neuronal.

LayerNorm se ejecuta en una operación a partir de la suma del vector embedding y el vector de self-attention final que se obtiene después de condensarlo en las múltiples cabeceras.

Los pasos para aplicar LayerNorm son los siguientes:

1. La entrada x pasa por la subcapa (por ejemplo, self-attention o FFNN).
2. El resultado se suma a la entrada original x (residuo).
3. La suma se normaliza mediante la normalización de capas.

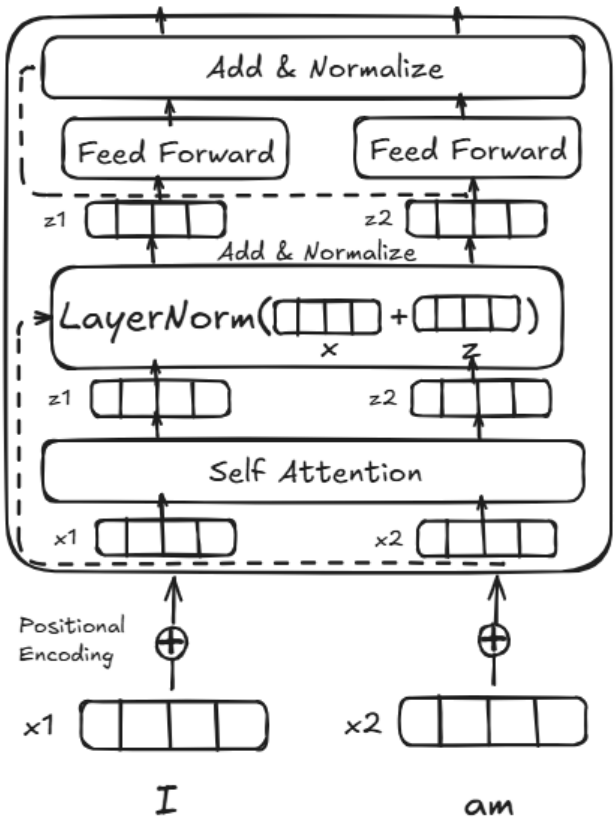


Figura 18: LayerNorm

Fuente: Adaptado de (Alammar, 2018)

3.1.5 Decoder

Al terminar la etapa de encoding, se empieza con la etapa del decoding, cada paso en todas las capas del decoder se consigue como salida un componente de la secuencia de salida, que es la traducción al español. Tras exponer los conceptos del encoder, se analiza cómo los componentes del decoder trabajan en conjunto para realizar la traducción.

El encoder comienza a procesar la cadena de entrada. La salida del encoder mayor, en este caso el sexto es transformado en un set de vectores Key y Value. Estos vectores serán utilizados en cada decoder en la capa llamada “encoder-decoder attention”, la cual es una capa que apoya al decoder a centrarse en las posiciones correctas dentro de la secuencia de entrada.

La fase del decoding es repetida hasta que un símbolo especial es alcanzado, indicando que el decoder de Transformer ha finalizado su salida. La salida de cada etapa en el decoder es alimentada al decoder de más abajo en el siguiente paso y todos los decoders envían sus resultados al siguiente decoder de igual manera que en el lado del encoder. De tal manera en que se ingresaban las entradas al encoder, hay que aplicar embedding y agregar positional encoding al inicio del decoder para señalar la posición de cada palabra que es procesada.

En el decoder, la self-attention opera de forma algo distinta que en el encoder.

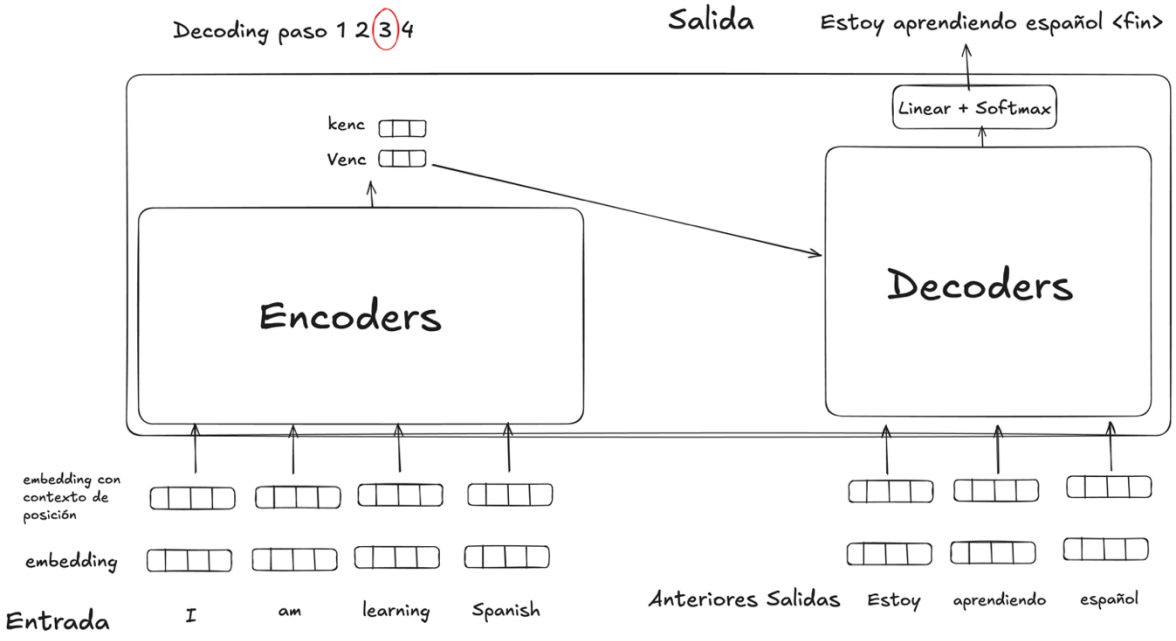


Figura 19: Iteraciones en el Decoder

Fuente: Propia

En el decoder, la capa de self-attention está únicamente permitida a considerar posiciones previas en la secuencia de salida. Esto se realiza marcando posiciones siguientes (definiéndolas como -infinito) antes de la cuarta etapa de aplicar softmax en el cálculo de self-attention.

3.1.6 Encoder-Decoder Attention

La capa de encoder-decoder attention funciona de igual manera que el MultiHead self-attention del encoder, la única diferencia es que crea el vector Query de la capa anterior y utiliza los vectores Key y Value de la salida de la pila de encoders los cuales son proporcionados en el sexto encoder.

3.1.7 Linear y Softmax

En la pila de decoders la salida final es un vector de puntos flotantes. Lo que hacen las capas Linear y Softmax es transformar aquel vector final, en una palabra.

Linear es una red neuronal totalmente conectada que transforma el vector generado por la pila de decoders, en un vector ampliado llamado vector de logits.

En un supuesto caso, el modelo comprende 5000 únicas palabras en español, (el vocabulario de salida del modelo) que es aprendido en base al dataset de entrenamiento. Esto quiere decir que al generar el vector final del decoder se proyecta en un vector de logits de 5000 celdas, cada celda corresponde al puntaje de cada palabra.

Softmax, transforma una lista de números en probabilidades. Hace que cada número sea positivo y los escala para que todos sumen 1. Softmax se utiliza en modelos de clasificación para elegir la clase más probable.

La capa Softmax convierte los puntajes dentro del vector de logits en probabilidades, todas positivas, que sumen a 1.0. La celda con la probabilidad más cercana a 1.0 es elegida, y la palabra asociada con aquella celda es impresa como la salida de este paso en el decoder.

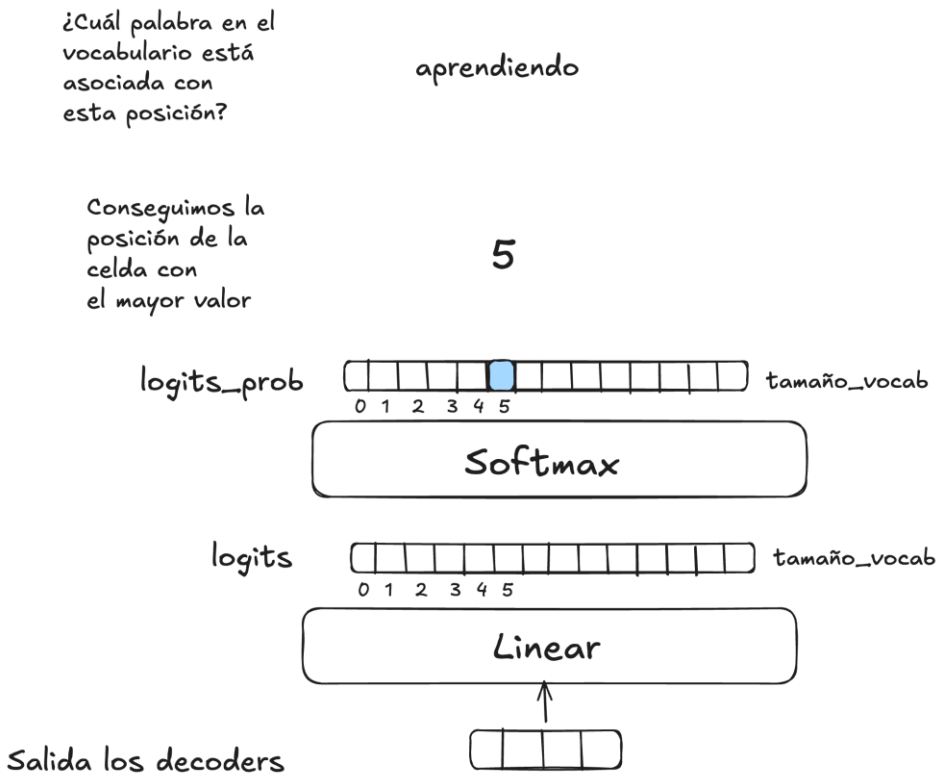


Figura 20: Linear + Softmax

Fuente: Propia

3.1.8 Loss

Un ejemplo común al entrenar un modelo, en el primer paso en la fase de entrenamiento se requiere traducir “Thanks” a “Gracias”.

Lo cual indica que se necesita que el resultado sea una probabilidad indicando la palabra “Gracias”. Debido a que el modelo no está entrenado todavía, es poco probable que eso ocurra.

Ya que los parámetros del modelo son inicializados de manera aleatoria, el modelo no entrenado reproduce valores arbitrarios para cada palabra. Según lo analizado el funcionamiento de un bucle de entrenamiento hay que comparar las salidas con la salida actual y finalmente ajustar los parámetros de la red neuronal al usar backpropagation para que la salida sea más cercana a la salida deseada.

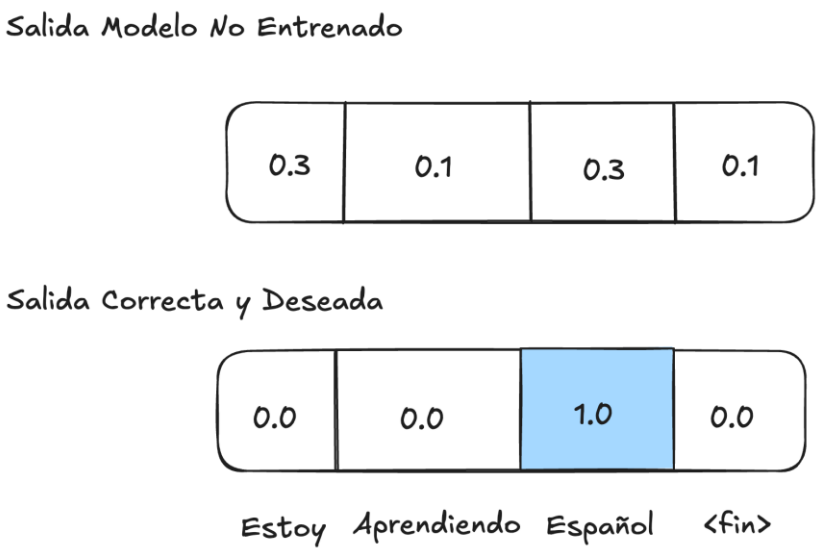


Figura 21: Entrenamiento Transformer

Fuente: Propia

3.2 Language Models are Unsupervised Multitask Learners

Antes de entender el artículo original del GPT-2 es necesario entender lo que es un modelo lingüístico.

Un modelo lingüístico es básicamente un modelo de machine learning que posee la capacidad de procesar parte de una secuencia y predecir la siguiente palabra. Los modelos lingüístico más utilizados son los teclados de los celulares, los cuales sugieren una próxima palabra basado en lo que se está escribiendo.

Este tipo de tarea se denomina aprendizaje auto supervisado: no se requiere etiquetar al modelo, simplemente alimentarlo con una gran cantidad de textos. Para adivinar correctamente la siguiente palabra de una oración, el modelo deberá desarrollar una comprensión del inglés (u otro idioma).

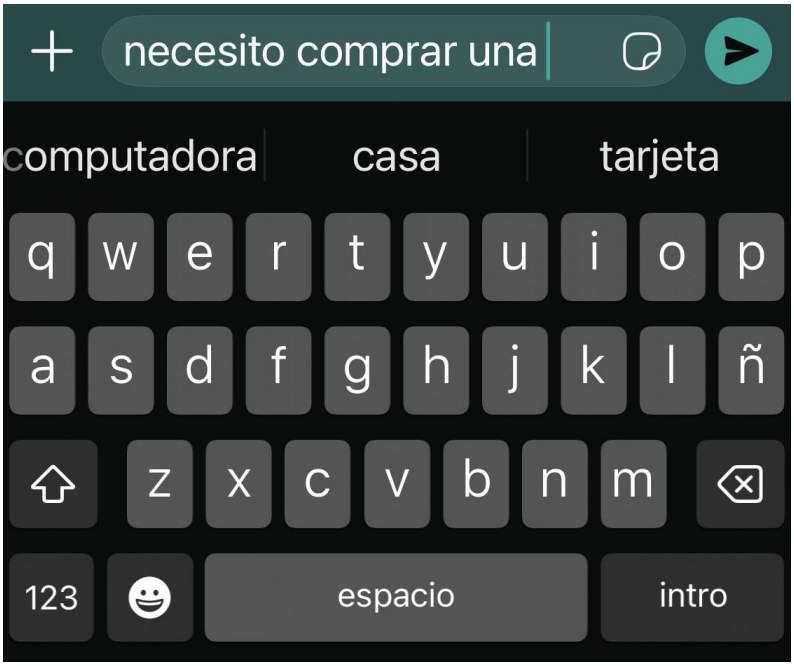


Figura 22: Modelo Lingüístico en un teclado

Fuente: Propia

Basándonos en esta explicación, un GPT es básicamente la función de predicción de la siguiente palabra en una aplicación de teclado, el cual es mucho más sofisticado que el que cualquier celular posee. El GPT-2 fue un modelo de 1.5 billones de parámetros entrenado en un dataset de 40GB llamado WebText que los ingenieros de OpenAI extrajeron del internet como parte de su investigación. Dentro de la publicación del artículo del GPT-2 entrenaron 4 modelos, los cuales poseen las siguientes características:

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

Figura 23: Hiperparámetros de arquitectura para los 4 tamaños de modelo

Fuente: (Radford et al., 2019)

A medida que se transiciona del modelo más pequeño al más grande, cada modelo posee una red neuronal mucho más profunda lo cual como resultado muestra más capacidad, mejor rendimiento, pero todo esto viene con el costo de más computo. El modelo pequeño posee 12 capas de Decoder mientras que el modelo más grande posee 48 capas de Decoder. Se evidencia en el siguiente gráfico, que, por cada modelo, el tamaño del embedding puede procesar un mayor tamaño. Cada palabra o token fluye a través de todos los bloques de Decoder a lo largo de su propio camino.

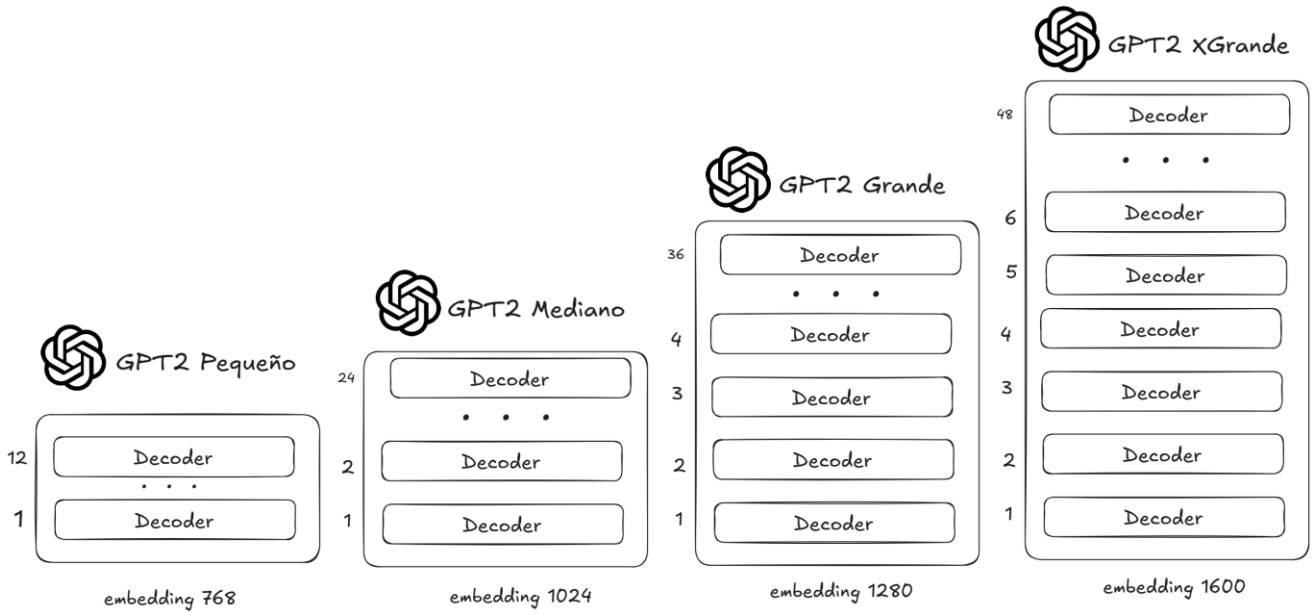


Figura 24: Modelos GPT-2

Fuente: Adaptado de (Alammar, 2019)

Previamente se ha visto que el Transformer original (o Transformer Encoder-Decoder) de Google Brain está compuesto por encoders y decoders. Aquella arquitectura fue apropiada por que el modelo abordaba un problema de traducción, un problema en el cual una arquitectura encoder-decoder ha sido exitosa.

Mucha de la investigación posterior a la propuesta del Transformer original hizo que la arquitectura abandonara bien sea el encoder o el decoder y utilizar únicamente uno de los dos, apilándolos lo más alto posible, alimentándolos con cantidades masivas de texto de entrenamiento y gigantes cantidades de cómputo.

Esto tiene sentido por la razón de que el lenguaje proporciona un dominio natural para el estudio de la inteligencia artificial, ya que la gran mayoría de las tareas de razonamiento pueden expresarse y evaluarse eficientemente en el lenguaje. Según (Kaplan et al., 2020) el rendimiento del modelado lingüístico mejora gradualmente a medida que crece el modelo, también crece el conjunto de datos y la cantidad de cómputo utilizado para el entrenamiento.

El GPT está compuesto por bloques de decoders. Al igual que los modelos lingüísticos tradicionales, el modelo genera un token a la vez. Eso quiere decir que después de producir un token, ese mismo token es alimentado a la secuencia de entradas y aquella nueva secuencia se convierte en la entrada al modelo en el siguiente paso. Esto se denomina auto regresión.

En (Liu et al., 2018) se propone un orden diferente del Transformer el cual posee las capacidades de un modelo lingüístico. Aquel modelo se deshizo del encoder del Transformer. Esta propuesta estuvo compuesta por una arquitectura la cual poseía una pila de seis bloques decoders. Ya que los investigadores eliminaron el encoder, este tipo de arquitectura es conocido como Transformer Decoder. Todos los bloques del Decoder son idénticos, se ha ampliado y detallado que es lo que contiene un bloque decoder en la siguiente figura:

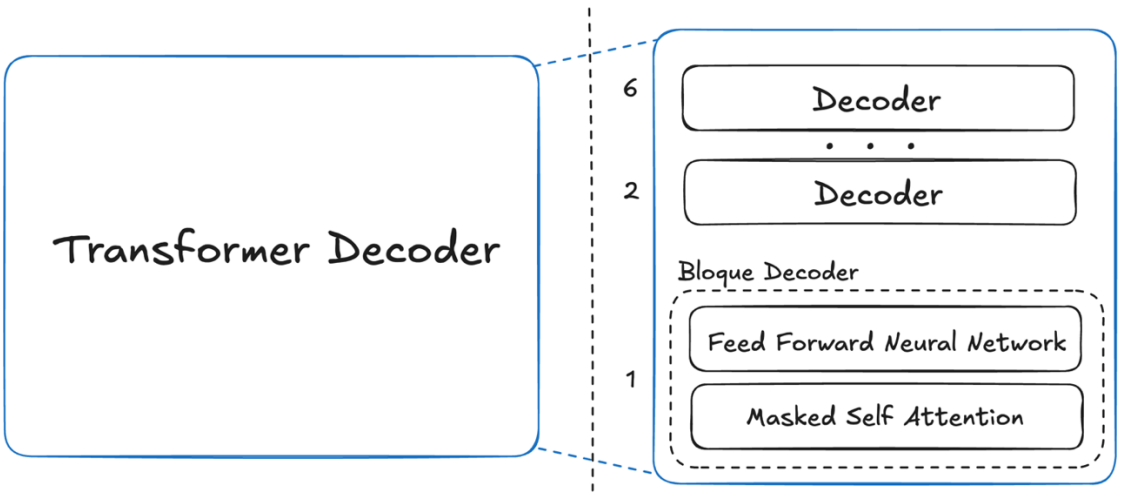


Figura 25: Transformer Decoder

Fuente: Adaptado de (Alammar, 2019)

3.2.1 Ingreso al Decoder

De igual manera que en el Transformer original, el modelo busca el embedding de la palabra que se ingresa en la matriz de embeddings, uno de los componentes que se obtiene al entrenar el modelo.

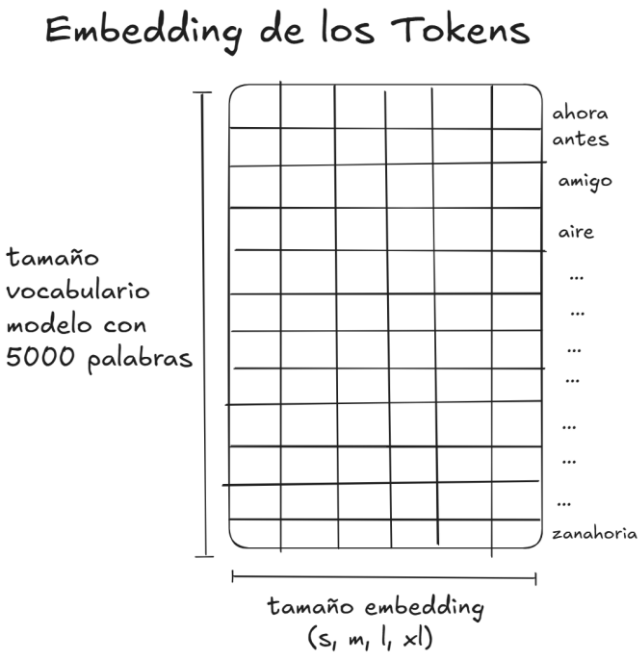


Figura 26: Token Embedding

Fuente: Propia

Antes de alimentar este embedding se debe incorporar el positional encoding para mostrar un contexto de cuál es la posición que tiene cada palabra. Además de la matriz de embeddings también existe una matriz de positional encoding la cual almacena la posición de cada palabra en el vocabulario del modelo.

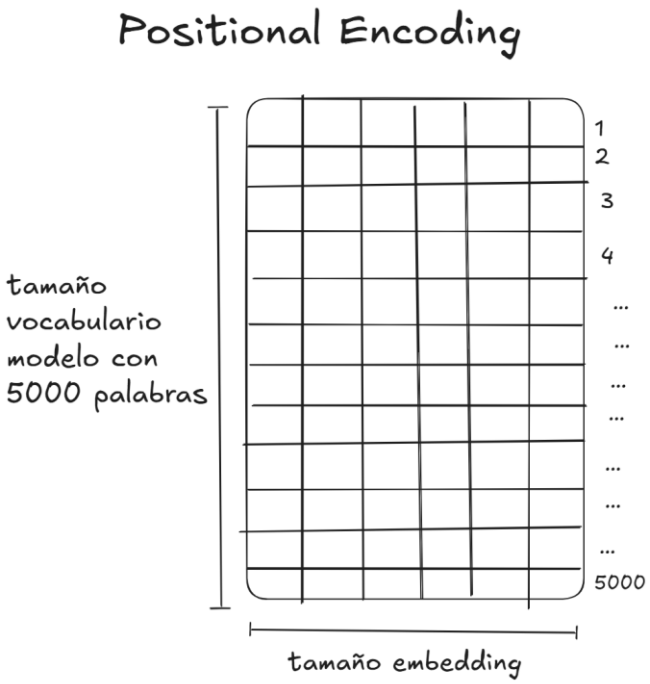


Figura 27: Positional Encoding

Fuente: Adaptado de (Alammar, 2019)

3.2.2 Masked Self-Attention

Ya que el lenguaje depende de gran medida del contexto, es necesario la aplicación de attention en la red neuronal. Lo que hace self-attention es incorporar en el modelo la identificación de las palabras importantes y asociadas que explican el contexto de una determinada palabra antes de procesarla.

El decoder posee una capa de self-attention que es diferente a la capa de self-attention en el encoder, se utiliza la misma ecuación de attention, pero la diferencia es que en el decoder se bloquean los tokens futuros al interferir con el cálculo de self-attention bloqueando información acerca de los tokens que están a la derecha de la posición que está siendo calculada. Un bloque normal de self-attention permite que una posición tenga acceso a todos los tokens a su derecha. El masked self-attention impide que esto suceda:

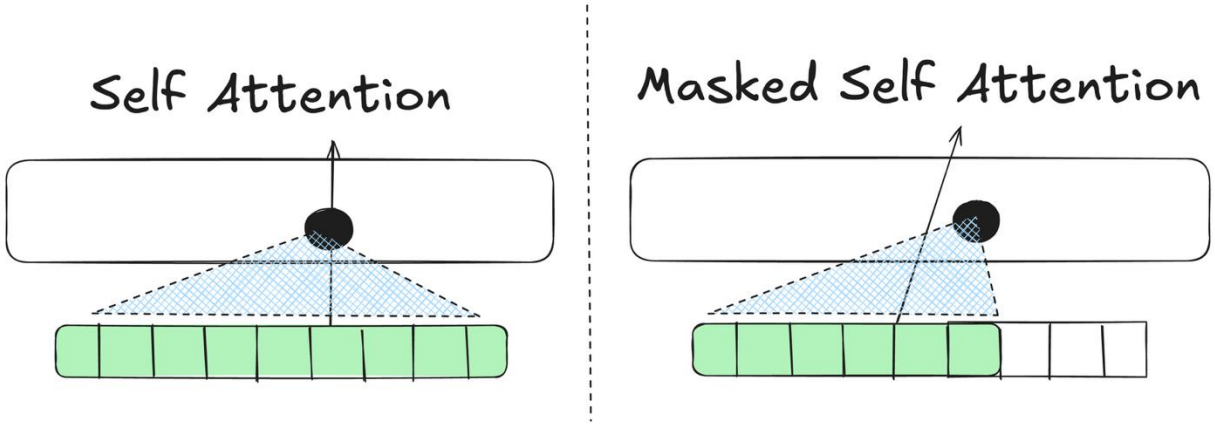


Figura 28: Self-Attention vs Masked Self-Attention

Fuente: Adaptado de (Alammar, 2019)

Masked Self-Attention es idéntico al self-attention del encoder, pero en el segundo paso de toda la fórmula a la cual se refiere en la Figura 15, se realiza una modificación en el paso 2 en donde la multiplicación entre q y k, únicamente se lleva a cabo este paso tomando en cuenta palabras anteriores, al ignorar todas las palabras futuras, ya que se intenta predecir la siguiente palabra. El modelo interfiere con el paso de puntuación, básicamente asigna un puntaje de 0 a los tokens futuros, por lo que el modelo no puede acceder a palabras futuras.

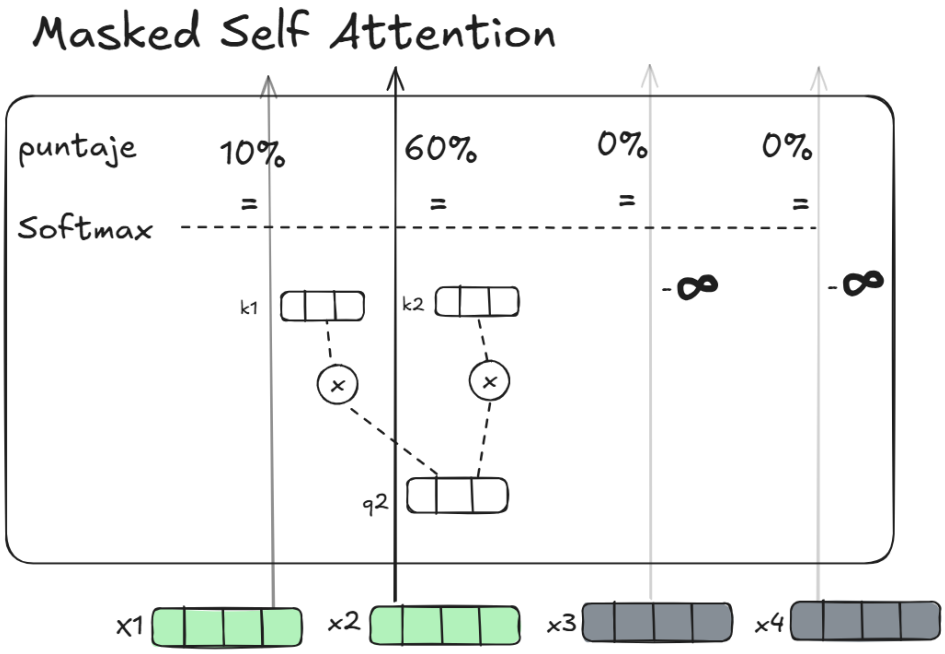


Figura 29: Puntaje Masked Self-Attention

Fuente: Adaptado de (Alammar, 2019)

El modelo aplica una attention mask sobre una matriz. Para ilustrar esta idea se toma un ejemplo donde hay 3 palabras “hola como estas”. En un modelo lingüístico esta secuencia se resuelve en los siguientes pasos:

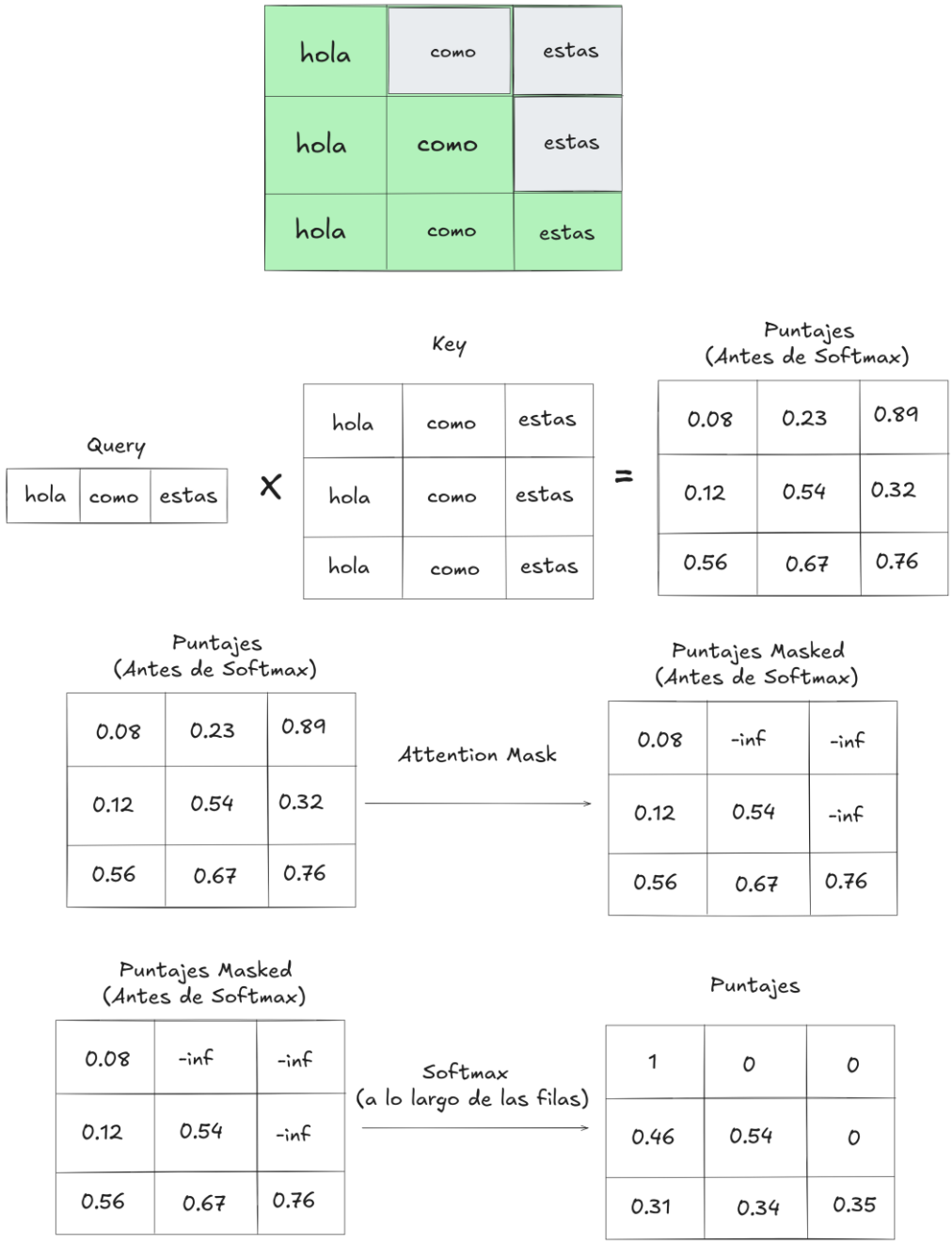


Figura 30: Masked Self-Attention

Fuente: Propia

Después de la multiplicación el attention mask es agregado estableciendo las celdas que van a ser enmascaradas a –infinito o a un número negativo muy grande, en el caso del artículo de GPT-2 se estableció el número –1 billón. Posteriormente Softmax es aplicada a cada fila la cual por ende reproduce los puntajes actuales que se utilizarán en el cálculo de self-attention.

Lo que se puede entender de la tabla de puntajes es lo siguiente:

- Cuando el modelo procese el primer ejemplo en el dataset (la primera fila), que contiene únicamente la palabra “hola”, 100% de su atención será enfocada en esa palabra.
- Cuando el modelo procese el segundo ejemplo en el dataset (la segunda fila), que contiene la palabra “hola como”, 46% y 54% de su atención será enfocada en las palabras “hola” y “como” respectivamente.
- Inferimos lo mismo por cada ejemplo en el dataset.

3.2.3 Capa Feed Forward Neural Network

De igual manera en la que se ha explicado en la arquitectura del Transformer original se alimenta el resultado final a una capa FFNN, antes de conseguir el resultado final se utiliza del mismo modo como se explicó anteriormente los seis pasos del cálculo de la ecuación de attention y posteriormente el componente de multihead self-attention es agregado, para poder permitir que la capa de attention aprenda desde múltiples perspectivas mediante el uso de varios conjuntos de pesos (multicabezas), captura características distintas de la entrada. El GPT-2 Pequeño tiene 12 cabezales de attention. Siguiendo el esquema original, el resultado de la multihead attention se alimenta a una capa FFNN.

La capa FFNN es donde el token de entrada es procesado después de que la capa de self-attention haya incluido el contexto apropiado de la palabra. Está compuesta por dos capas, la primera capa es cuatro veces el tamaño del modelo. (En el caso del modelo pequeño de GPT-2 el cual posee un tamaño de 768, esta red posee 3072 neuronas). Se multiplica por cuatro porque esto proporciona al Transformer una mayor capacidad de representación para manejar tareas grandes y además porque es el tamaño que se utilizó en el Transformer original (la dimensión del modelo fue 512 y la primera capa tuvo un tamaño de 2048).

La segunda capa dentro de FFNN proyecta el resultado de la primera capa de vuelta a la dimensión original del modelo (768 en el caso de GPT-2 Pequeño). El resultado de esta multiplicación es la salida de esta palabra que está siendo procesada.

Este apartado concluye con el análisis de dos artículos fundamentales en inteligencia artificial moderna Attention is All You Need la exploración de la arquitectura original Transformer y Language Models are Unsupervised Multitask Learners donde se exploró a detalle el Transformer Decoder utilizado dentro de OpenAI.

3.2.4 ¿Por qué una predicción más precisa de la siguiente palabra genera una mayor comprensión?

Al entender la idea de cómo funciona el GPT de acuerdo con el artículo de OpenAI, la predicción del siguiente token consiste en operaciones matemáticas básicas probabilísticas en matrices. Sin embargo, la idea de que solo está 'adivinando' la siguiente palabra encierra una implicación más profunda.

Al entrenar una red neuronal grande que únicamente predice de manera precisa la próxima palabra en diferentes documentos provenientes del internet, un modelo global, al verlo desde la superficie parece ser que aprende a establecer correlaciones estadísticas en textos, pero resulta que, al aprender correlaciones estadísticas en textos, lo que la red neuronal en realidad aprende es un tipo de representación del proceso de reproducir el texto. El texto puede entenderse como una representación simbólica del mundo real, transmitiendo parte de su estructura y significado, la red va absorbiendo conocimiento cada vez más amplio sobre la realidad (los seres humanos, sus circunstancias, motivos, interacciones) y adquiere una comprensión abstracta del mundo. Este proceso es lo que aprende la red al intentar predecir la siguiente palabra. Además, mientras mayor sea la precisión en la predicción de la siguiente palabra existe mayor fidelidad y más resolución

de este proceso. Esta es la fase de preentrenamiento, lo que esto no hace es especificar un comportamiento deseable que la red neuronal pueda exhibir.

Esta perspectiva se aleja considerablemente de la concepción de un asistente diseñado exclusivamente para ser útil, que siga ciertas reglas y no vulnerar aquellas reglas, esto requiere entrenamiento adicional. Esta es la fase de fine-tuning. Es una fase en la cual no se enseña nuevo conocimiento al modelo, es una fase donde se aprende a comunicar elocuentemente con el modelo, este proceso y segunda fase es extremadamente necesaria ya que se presenta un sistema que puede ser utilizado dentro de una aplicación completamente desplegado. Entre mejor se realice esta segunda fase el modelo será más útil y confiable. En adición a la primera fase en donde el modelo intenta aprender lo que más puede acerca del mundo, a tener una proyección completa acerca del entorno, el cual es el texto.

En este caso, el siguiente ejemplo, en una novela de detectives, la cual posee una trama un tanto complicada, llena de diferentes personajes, múltiples eventos, misterios y pistas las cuales no son completamente claras. En la página final de la novela el detective ha reunido todas las pistas, todas las personas y finalmente decidir quién fue el responsable del crimen, y el nombre de la persona es...predice esa palabra.

En base a este ejemplo, al intentar predecir la siguiente palabra se obliga a la red neuronal a razonar. Sin embargo, el razonamiento no es un concepto bien definido, puede entenderse como pensar para decidir, se puede observar que es algo que claramente la red neuronal demuestra.

3.2.5 Diferencia entre Transformer Decoder y otras arquitecturas Transformer

Existen variantes en la arquitectura de Transformer, entre ellas Transformer Encoder, Transformer Decoder y Transformer Encoder-Decoder, cada una tiene su uso.

BERT utiliza únicamente Encoders en su arquitectura, es un Transformer Encoder, eso quiere decir que, las capas de attention pueden acceder a todas las palabras en la sentencia inicial.

Modelos como BERT de Google son excelentes para comprender textos porque pueden analizar el contexto completo en ambas direcciones. Esto los hace ideales para tareas donde la comprensión de toda la información es fundamental.

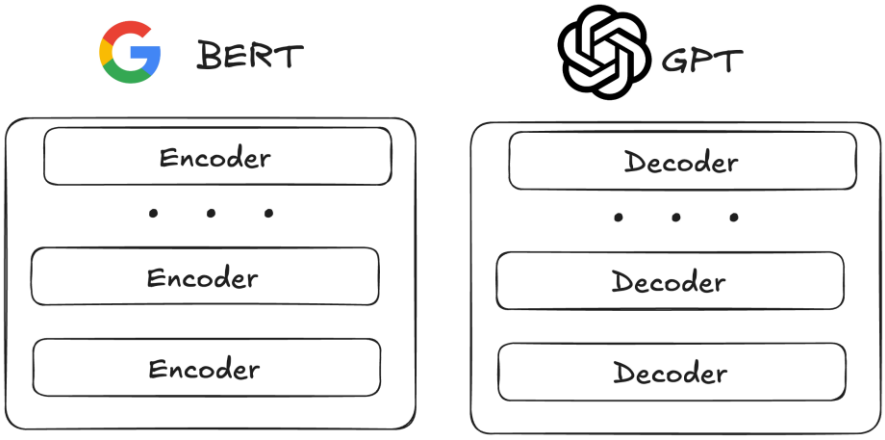


Figura 31: BERT vs GPT

Fuente: Propia

Transformer Decoder se diferencia ya que, en cada fase para una palabra dada, las capas de atención sólo pueden acceder a las palabras ubicadas antes de ella en la oración. El preentrenamiento del Transformer Decoder gira en torno a la predicción de la siguiente palabra de la oración. Estos modelos son los más adecuados para tareas que implican generación de texto.

Encoder-Decoders en cada fase las capas de atención del Encoder pueden acceder a todas las palabras de la oración inicial, mientras que las capas de atención del Decoder sólo pueden acceder a las palabras ubicadas antes de una palabra dada en la entrada. Los modelos Encoder-Decoder son más adecuados para tareas como la generación de texto, resumen de documentos, traducción y responder a preguntas.

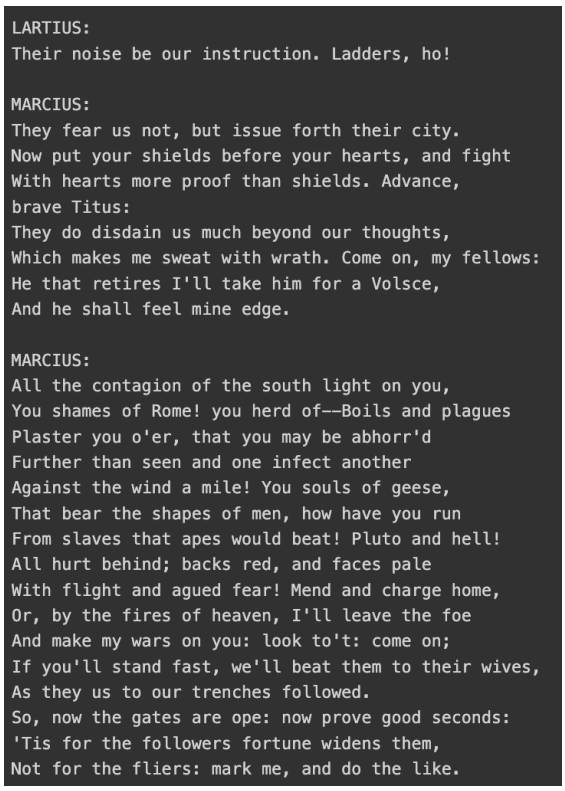
CAPÍTULO 4

4. Preentrenamiento de GPT

4.1 Descripción del Dataset

El Tiny Shakespeare es un dataset de texto pequeño. Contiene aproximadamente 1 millón de caracteres extraídos de las obras de William Shakespeare. Entre las características del dataset:

- Contenido: diálogos, actos y escenas de las obras de Shakespeare.
- Tamaño: ~1MB.
- Formato: texto plano, sin etiquetas.
- Propósito: ideal para probar modelos pequeños por su tamaño y riqueza lingüística.



```
LARTIUS:
Their noise be our instruction. Ladders, ho!

MARCIUS:
They fear us not, but issue forth their city.
Now put your shields before your hearts, and fight
With hearts more proof than shields. Advance,
brave Titus:
They do disdain us much beyond our thoughts,
Which makes me sweat with wrath. Come on, my fellows:
He that retires I'll take him for a Volsce,
And he shall feel mine edge.

MARCIUS:
All the contagion of the south light on you,
You shames of Rome! you herd of--Boils and plagues
Plaster you o'er, that you may be abhorr'd
Further than seen and one infect another
Against the wind a mile! You souls of geese,
That bear the shapes of men, how have you run
From slaves that apes would beat! Pluto and hell!
All hurt behind; backs red, and faces pale
With flight and agued fear! Mend and charge home,
Or, by the fires of heaven, I'll leave the foe
And make my wars on you: look to't: come on;
If you'll stand fast, we'll beat them to their wives,
As they us to our trenches followed.
So, now the gates are ope: now prove good seconds:
'Tis for the followers fortune widens them,
Not for the fliers: mark me, and do the like.
```

Figura 32: Dataset Shakespeare

Fuente: Propia

El dataset es procesado para ser preparado para el entrenamiento: se identifican los caracteres únicos, que luego son convertidos en números enteros para facilitar el embedding para ser posteriormente divididos en datos de entrenamiento y validación.

El número de parámetros dentro de la red neuronal es 10,788,929 y el corpus tiene 1,115,394 caracteres únicos.

4.2 Configuración de Hiperparámetros

Para que los resultados sean reproducibles se utiliza un manual seed 1500.

Dentro del código donde se realizará el entrenamiento del modelo los hiperparámetros empleados son:

- Iteraciones máximas (cuántas veces iterar el modelo): 5000.
- Dropout de 20%.
- Número de capas decoder: 6.
- Número de cabezales (multihead attention): 6 cabezales.
- Intervalo de evaluación (cada cuanta iteración imprimir Loss): 500.

4.3 Arquitectura Implementada en PyTorch

El núcleo del modelo consiste en capas del Transformer, cada capa incluye lo siguiente:

- Multihead Attention: facilita que el modelo atienda múltiples posiciones de la entrada al mismo tiempo mediante distintos cabezales.
- Feedforward Neural Network: Dos capas de multilayer perceptron que incrementa la no linealidad de la red neuronal.
- Layer Normalization: aplicado en cada capa para estabilizar y acelerar el entrenamiento del modelo.

El modelo se entrena con el optimizador SGD.

4.4 Evaluación del Rendimiento

A continuación, se presentan los resultados obtenidos:

Iteración	Loss Entrenamiento	Perplexity Entrenamiento	Loss Validación	Perplexity Validación
0	4.2221	68.18	4.2306	68.76
500	1.7600	5.81	1.9146	6.78
1000	1.3903	4.02	1.5987	4.95
1500	1.2644	3.54	1.5271	4.60
2000	1.1835	3.07	1.4910	4.44
2500	1.0718	2.92	1.4804	4.39
3000	1.0179	2.77	1.5127	4.54
3500	0.9604	2.61	1.5102	4.53
4500	0.9125	2.49	1.5351	4.64
4999	0.8589	2.36	1.5565	4.74

Tabla 1: Resultados de entrenamiento

Fuente: Propia

La tabla contiene lo siguiente:

- Iteración: Número de pasos de entrenamiento completados.
- Loss Entrenamiento: Error en los datos de entrenamiento.
- Perplexity Entrenamiento: Medida de incertidumbre del modelo en entrenamiento.
- Loss Validación: Error en los datos de validación.

- Perplexity Validación: Medida de incertidumbre del modelo en validación.

Perplexity es una medida de incertidumbre del modelo. Representa el número promedio de opciones igualmente probables que el modelo considera. Representada en la siguiente ecuación:

$$\text{Perplexity} = e^{\text{Loss}}$$

Si Perplexity = 2, significa que el modelo está entre 2 palabras posibles por cada token.

Si Perplexity = 50, significa que el modelo está muy inseguro (como si eligiera entre 50 palabras al azar).

La intuición es que cuando el Perplexity es bajo, quiere decir que el modelo está más seguro de sus predicciones, mientras que cuando el Perplexity es alto, el modelo está adivinando.

De igual manera se muestra los recursos computacionales que fueron necesitados para el entrenamiento del modelo:

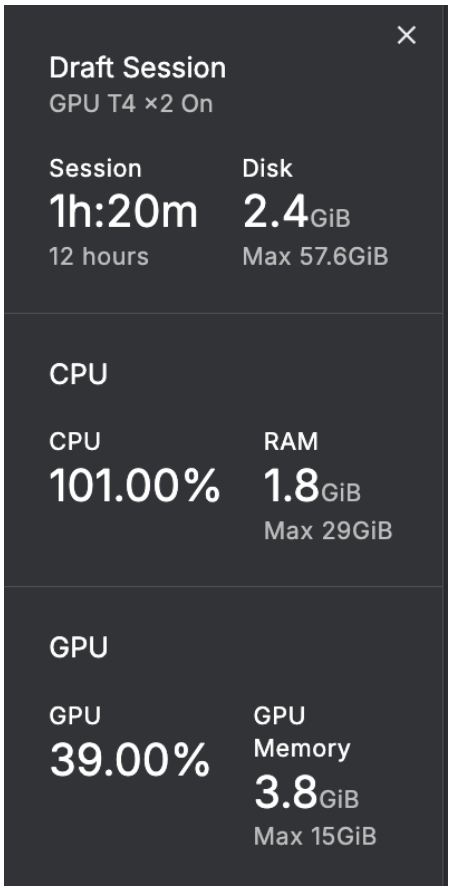


Figura 33: Recursos computacionales requeridos para el entrenamiento

Fuente: Propia

El entrenamiento del modelo fue posible gracias a la plataforma Kaggle, usando dos GPUs T4, lo cual demoró aproximadamente una hora y veinte minutos en entrenar el GPT en el dataset de Shakespeare, a continuación, se presenta generaciones que puede hacer el modelo:

4.5 Generación de Texto del Modelo

```
MONTAGUE:
For seize light. I will could not this peace:
Here soft of my knot I drunk and coal I
To drunk him to his lucky to another
And to be seen, but, savouring unto his meaning,
And not so, as such now rosemble for him!
In God's sake, tale love his earth,
And cannot the stream
From in my connicence in his son's-in self--
Which fled the world stoe!
Fear my brother was to her heard but wail the hollow'd
More than tempted ground the flourion of ver'd.

CLIFFORD:
To my husband sheep, one son, and these arm that have
The breather which breath the shapes them to save.
Have made me, and leave to kiss stay;
And but on if I do thou wrought not speak.

CLARENCE:
O Somervant! hear me, Clarence!

RECHARD:
Your fear will your husband; for she hath cut,
Wert it be the grievous off the feaster:
Now may but speak into and yourself,
That put seeming so he drown to your came.

HASTINGS:
No, I have done so like and time yours subjects.

GLOUCESTER:
My good shall lights, when I am your tall?
```

Figura 34: Generaciones del modelo

Fuente: Propia

Como análisis se observa que el modelo emula bien la estructura teatral de las obras de Shakespeare:

- Uso de nombres de personajes (MONTAGUE, CLIFFORD, CLARENCE…).

- Líneas cortas y poéticas, similares a parlamentos teatrales.
- El modelo genera palabras que son parte del estilo Shakespeareano.

Entre las limitaciones que se pueden observar tiene una coherencia muy limitada y existen errores comunes al ver verbos mal emparejados. Sin embargo, el modelo logra aprender patrones de estilo.

A pesar de que el modelo muestra una disminución tanto en el Loss como en el perplexity en el conjunto de datos de entrenamiento, se evidencia que el Perplexity en validación alcanza un punto en el que deja de mejorar. Este comportamiento se debe a varios factores:

- El modelo cuenta con 10,8 millones de parámetros y se entrenó con un corpus muy reducido (≈1MB), lo que restringe su capacidad de aprendizaje y provoca que el desempeño se estanque.
- El tamaño del dataset: el corpus del dataset contiene información importante, pero insuficiente para mejorar el desempeño del modelo. Llega un punto en el que el modelo ya es incapaz de extraer nuevo conocimiento útil.
- Sobreajuste: el modelo comienza a aprender patrones específicos del conjunto de entrenamiento, lo que permite reducir el Loss, pero no logra generalizar bien en el conjunto de validación.

4. 6 Escalabilidad del Modelo

La arquitectura Transformer es escalable y puede adaptarse para entrenar modelos con datasets más complejos como los disponibles en HuggingFace (por ejemplo, OpenWebText, WikiText-103).

Sin embargo, escalar este tipo de modelo presenta los siguientes desafíos:

- Cómputo: modelos más grandes requieren GPUs de alto rendimiento.

- Memoria: el número de parámetros crece significativamente, lo que implica mayor uso de RAM.
- Tiempo de entrenamiento: el procesamiento de un dataset de 1MB frente a varios GB aumenta considerablemente el tiempo de entrenamiento de horas a días o incluso semanas.
- Técnicas Adicionales: el entrenamiento de modelos más grandes requieren técnicas como checkpointing y entrenamiento distribuido en multiples GPUs.

Por ser una arquitectura escalable, si se entrena con datasets más grandes y se aumenta su tamaño (más capas, más parámetros), puede lograr tareas más complejas con mejores resultados. Representa una base para el desarrollo de modelos multimodales, es decir, que pueden recibir diferentes tipos de entrada como por ejemplo texto en diferentes idiomas, imágenes, audio o video y generar salidas variadas (texto, imagen, audio) de manera conjunta.

CAPÍTULO 5

5. Consideraciones Éticas en Modelos Generativos

5.1 Implicaciones éticas de la Desinformación

Para abordar los desafíos éticos en el uso de modelos lingüísticos es importante entender el concepto de desinformación. Este fenómeno no consiste necesariamente de hacer creer algo falso, sino que a menudo se utiliza para sembrar discordia e incertidumbre, y para que la gente desista de buscar la verdad. Se piensa que la desinformación se trata principalmente de información o noticias falsas, pero a menudo puede contener semillas de verdad o verdades sacadas de contexto.

5.2 Accesibilidad y Potencial de Uso Malicioso

A medida que los modelos lingüísticos mejoran, abren nuevas posibilidades en campos tan diversos como la salud, el derecho, la educación y la ciencia. Pero, como ocurre con cualquier nueva tecnología, conviene considerar su posible mal uso.

De acuerdo con la investigación hecha por (Goldstein et al., 2023) los modelos lingüísticos reducen drásticamente las barreras de entrada, lo que permite que una gama más amplia de individuos o grupos lancen campañas de influencia que antes requerían muchos recursos. Ahora es posible generar desinformación a gran escala, adaptarla a audiencias específicas y difundirla en tiempo real (por ejemplo, mediante chatbots o mensajes masivos).

5. 3 Riesgos Relacionados con el Modelo Entrenado

Aunque este proyecto tiene un propósito académico, un modelo generativo, por más pequeño que sea, podría generar texto sin control, lo cual conlleva riesgos:

- Desinformación en redes sociales: si se conecta este modelo a plataformas externas, podría generar información errónea con apariencia de coherencia, propagando datos falsos sin supervisión.

- Falta de veracidad: el modelo produce texto basado en probabilidades, no en hechos. Esto implica que puede crear contenido aparentemente confiable pero incorrecto.
- La facilidad con la que se puede entrenar y desplegar un modelo generativo sencillo, puede ser aprovechada por actores malintencionados para crear spam, suplantaciones o contenido ofensivo automatizado.

5.4 Buenas Prácticas para desplegar modelos lingüísticos

No existe una solución universal para mitigar la seguridad de los modelos lingüísticos. Si bien los principios se desarrollan en base a problemas de seguridad de estos modelos, deberían cambiar significativamente con el tiempo, ya que los usos comerciales de los modelos lingüísticos y las consideraciones de seguridad que los acompañan son nuevos y están en constante evolución. Entre las mitigaciones que se pueden considerar al momento de desplegar modelos linguisticos potentes, se pone a consideracion lo siguiente:

- Publicar pautas de uso y términos de uso de modelos lingüísticos de manera que se prohíban daños materiales a individuos, comunidades y sociedad, como por ejemplo spam o fraude.
- Desarrollar sistemas para aplicar las directrices de uso. Esto puede incluir límites de uso, filtrado de contenido, aprobación de aplicaciones antes del acceso a producción y la monitorización de actividad anómala.
- Formar equipos con trayectorias diversas. Se necesitan perspectivas diversas para abordar cómo funcionarán los modelos lingüísticos en la diversidad del mundo real.

Para los investigadores de Deep Learning, estos principios representan una guía para el desarrollo y la implementación de modelos lingüísticos más seguros.

CAPÍTULO 6

6. Conclusiones y Recomendaciones

6.1 Conclusiones

- La construcción del modelo en PyTorch, con la implementación de: embeddings posicionales, bloques de atención y capas feedforward. Validó la arquitectura Transformer a bajo nivel, sin recurrir a bibliotecas de alto nivel.
- La revisión de trabajos clave (Vaswani et al. 2017; Radford et al. 2019) sirvió como base teórica para entender el preentrenamiento sin supervisión. Al abordar estos artículos, se ofreció una explicación tanto visual como conceptual del trabajo realizado por los equipos de Google y OpenAI.
- El preentrenamiento del modelo se realizó usando un dataset pequeño, lo que facilitó observar que el modelo presenta una tendencia de aprendizaje lingüístico y estilísticos específicos de un corpus limitado, destacando la sensibilidad del aprendizaje a la naturaleza del dataset.
- Los modelos lingüísticos generativos, si bien son potentes, presentan riesgos reales de uso indebido en campañas de desinformación, ya que hacen que la generación de contenido persuasivo sea escalable y económica. Incluso los modelos pequeños, si son de acceso público, podrían ser explotados si no se consideran las salvaguardias éticas.

6. 2 Recomendaciones

- Se sugiere explorar datasets disponibles en Hugging Face como WebText, que permite reproducir a GPT-2. Esto facilitaría evaluar el impacto de datasets más grandes y diversos.
- Se sugiere implementar Mixed Precision Training para acelerar el preentrenamiento del modelo sin comprometer el rendimiento.

- Se sugiere investigar estrategias de tokenización en lenguajes con características estructurales particulares, como el japonés y el mandarín, donde no existe una idea bien definida “palabra"”, o lenguas como el hebreo y el árabe, que se escriben de derecha a izquierda. Esto permitiría adaptar el preprocesamiento del modelo a contextos multilingües más complejos.
- Se sugiere revisar el artículo (Zhu et al., 2025), el cual introduce una arquitectura alternativa que elimina completamente las capas de normalización, logrando resultados competitivos en la arquitectura Transformer.
- Dado que esta tesis se ha enfocado únicamente en el preentrenamiento del modelo, se sugiere realizar un estudio del artículo (Parthasarathy et al., 2024), para comprender las mejores prácticas de Fine-Tuning en modelos lingüísticos. Además, al aplicar fine-tuning debe evaluarse éticamente la calidad de datos y resultados, usando filtros o límites de uso para mitigar riesgos de desinformación.
- Dado el alto costo que implica entrenar un modelo lingüístico como GPT, se sugiere emplear la técnica Retrieval-Augmented Generation (RAG) en startups que necesiten un asistente útil, capaz de comunicarse directamente con datos actualizados de la empresa.
- Para futuros trabajos se sugiere la utilización de los siguientes proveedores: Lambda y The San Francisco Compute Company los cuales ofrecen precios muy accesibles a GPUs muy potentes como H100 de NVIDIA para entrenar modelos más grandes.

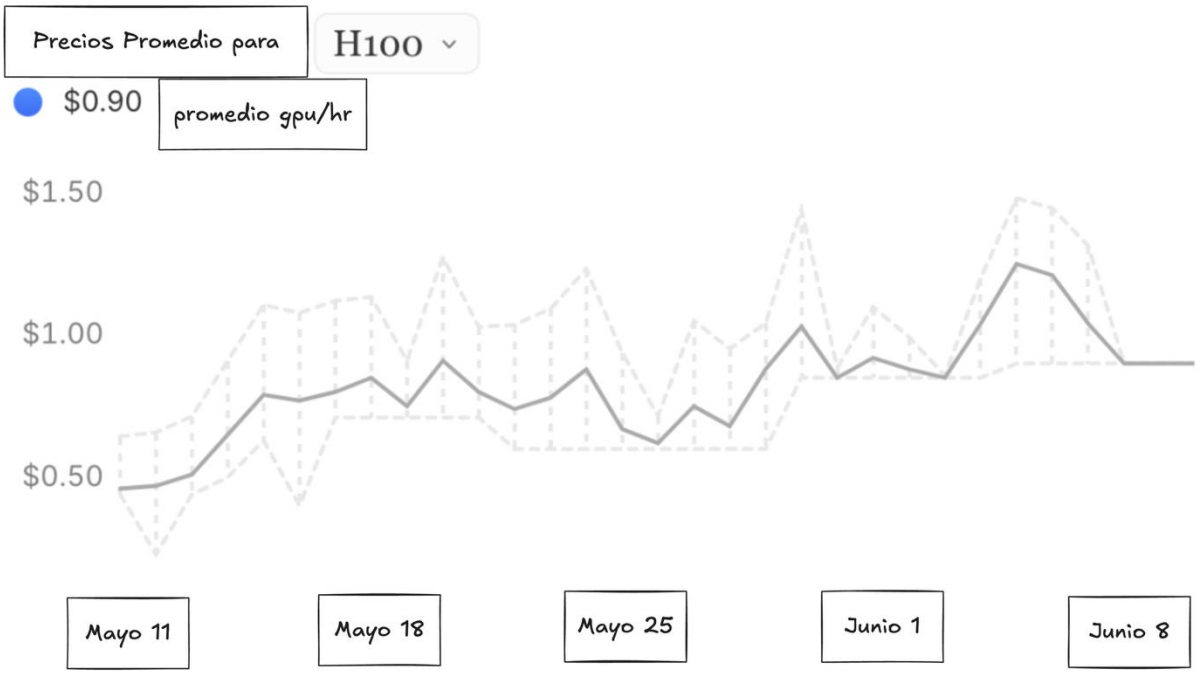


Figura 35: Precios promedio de uso GPU H100

Fuente: (The San Francisco Compute Company, 2025)

Referencias

Alammar, J. (2018). The illustrated transformer. https://jalammar.github.io/illustrated-transformer/

Alammar, J. (2019). The illustrated GPT-2 (Visualizing Transformer Language Models). https://jalammar.github.io/illustrated-gpt2/

Burkov, A. (2019). *The 100-page machine learning book*. Andriy Burkov.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Google Research. (2017, August 31). *Transformer: A novel neural network architecture for language understanding*. Google Research Blog. https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding

Hooker, S. (2020). The hardware lottery. *Communications of the ACM, 64*(9), 58–65.

Howard, J., & Gugger, S. (2020). *Deep learning for coders with fastai and PyTorch*. O'Reilly Media.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling laws for neural language models* [Preprint]. arXiv. https://arxiv.org/abs/2001.08361

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020, April 12). *Retrieval-augmented generation for knowledge-intensive NLP tasks* (arXiv:2005.11401v4) [Preprint]. arXiv. https://arxiv.org/abs/2005.11401

Liu, Y., Lapata, M., Stenetorp, P., & Gardner, M. (2018). *Generating Wikipedia by summarizing long sequences* [Preprint]. arXiv. https://arxiv.org/abs/1801.10198

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics, 5*(4), 115–133. https://doi.org/10.1007/BF02478259

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., & Wu, H. (2017). *Mixed precision training* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1710.03740

Minsky, M., & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. MIT Press.

Nielsen, M. A. (2015). *Neural networks and deep learning*. http://neuralnetworksanddeeplearning.com

Parthasarathy, V. B., Zafar, A., Khan, A., & Shahid, A. (2024). *The ultimate guide to fine-tuning LLMs from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2408.13296

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training*. OpenAI. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners*. OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

Rosenblatt, F. (1958). *The design of an intelligent automaton*. Cornell Aeronautical Laboratory.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533–536. https://doi.org/10.1038/323533a0

Rumelhart, D. E., McClelland, J. L., & the PDP Research Group. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations*. MIT Press.

Samuel, A. (1962). Artificial intelligence: A frontier of automation. *The Annals of the American Academy of Political and Social Science, 340*, 10–20. https://www.jstor.org/stable/1033694

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*.

Zeiler, M. D., & Fergus, R. (2013). *Visualizing and understanding convolutional networks* [Preprint]. arXiv. https://arxiv.org/abs/1311.2901

Zhu, J., Chen, X., He, K., LeCun, Y., & Liu, Z. (2025, March 13). *Transformers without normalization* (arXiv:2503.10622) [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2503.10622