IA Generativa: Particularidades de los Modelos de Lenguaje Grandes

Miguel Murga Guevara
miguelmurgaguevara@hotmail.com

October 17, 2024

Abstract

Este documento presenta una introducción accesible a los fundamentos de la inteligencia artificial desde la perspectiva de la ciencia de datos, haciendo hincapié en las redes neuronales y el mecanismo de atención en Transformers. Se analizan sus principales características y aplicaciones, sirviendo como base para entender las particularidades de los Modelos de Lenguaje Grandes (LLM) en el contexto de la IA Generativa. Además, se propone una práctica para explorar el uso de un modelo a través de LM Studio.

Contents

1 Introducción

La inteligencia artificial (IA) ha experimentado un crecimiento exponencial en las últimas décadas, influenciando diversos campos y revolucionando la forma en que interactuamos con la tecnología. Para comprender plenamente el alcance y las capacidades de la IA moderna, es fundamental comenzar por la ciencia de datos, disciplina que permite extraer conocimiento y patrones significativos a partir de grandes volúmenes de datos.

Este documento explora los cimientos de la IA, comenzando con los conceptos básicos de la ciencia de datos, avanzando hacia las redes neuronales y culminando en el mecanismo de atención en Transformers. Esta progresión conceptual es esencial para entender las particularidades y el funcionamiento de los Modelos de Lenguaje Grandes (LLM), que serán abordados en detalle en secciones posteriores.

2 Objetivos

El objetivo principal de este trabajo es proporcionar una comprensión clara y estructurada de los fundamentos que sustentan la inteligencia artificial y cómo estos conducen al desarrollo de los Modelos de Lenguaje Grandes. Específicamente, se busca:

- Introducir los conceptos básicos de la ciencia de datos y su relevancia en la IA.
- Explicar el funcionamiento de las redes neuronales y su papel en el reconocimiento de patrones complejos.
- Describir el mecanismo de atención en Transformers y su importancia en el procesamiento de secuencias.
- Preparar el terreno para una discusión detallada sobre los Modelos de Lenguaje Grandes en la sección 4.
- Proponer una práctica para experimentar con un modelo de lenguaje en LM Studio.

3 Fundamentos de la Ciencia de Datos y la Inteligencia Artificial

La ciencia de datos es una disciplina que combina técnicas de estadística, matemáticas y computación para analizar e interpretar datos. En el contexto de la inteligencia artificial, la ciencia de datos proporciona las herramientas necesarias para procesar y extraer información útil de grandes conjuntos de datos, lo que es fundamental para entrenar modelos inteligentes.

3.1 Redes Neuronales y Compuertas Lógicas

Las redes neuronales artificiales son modelos computacionales inspirados en el funcionamiento del cerebro humano. Están compuestas por unidades interconectadas llamadas neuronas artificiales, que procesan información y aprenden a reconocer patrones complejos.

3.1.1 Modelo de la Neurona Artificial

Una neurona artificial realiza una combinación lineal de sus entradas y aplica una función de activación para generar una salida. Matemáticamente, se representa como:

$$\text{Salida} = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Donde:

- x_i son las entradas.
- w_i son los pesos asociados a cada entrada.
- b es el sesgo (*bias*).
- ϕ es la función de activación.

Este modelo básico permite emular operaciones lógicas fundamentales y sirve como bloque de construcción para redes más complejas.

3.1.2 Emulación de Compuertas Lógicas

Ajustando los pesos y el sesgo, una neurona artificial puede emular compuertas lógicas como AND, OR y NOT. Por ejemplo, para emular una compuerta AND:

- Pesos: $w_1 = 1$, $w_2 = 1$
- Sesgo: $b = -1.5$
- Función de activación: Función escalón

El comportamiento de la neurona replicará la tabla de verdad de la compuerta AND, demostrando cómo las redes neuronales pueden realizar operaciones lógicas básicas.

3.1.3 Limitaciones y Redes Neuronales Complejas

Mientras que neuronas individuales pueden emular funciones linealmente separables, para funciones más complejas como XOR es necesario utilizar redes neuronales multicapa. Estas redes pueden aprender representaciones más abstractas y resolver problemas no lineales.

3.2 Procesamiento de Secuencias y Limitaciones de las RNN

Las Redes Neuronales Recurrentes (RNN, *Recurrent Neural Networks*) están diseñadas para procesar datos secuenciales, manteniendo información de estados anteriores. Sin embargo, tienen dificultades para capturar dependencias a largo plazo debido al problema del desvanecimiento del gradiente.

3.2.1 Funcionamiento de las RNN

Las RNN procesan secuencias de forma iterativa, actualizando su estado interno en cada paso. A pesar de su capacidad para manejar secuencias, su rendimiento disminuye cuando la dependencia entre elementos de la secuencia es muy extensa.

3.3 El Mecanismo de Atención en Transformers

Los Transformers introducen el mecanismo de atención, permitiendo capturar relaciones globales en la secuencia sin procesarla de forma secuencial.

3.3.1 Vectores Query (Q), Key (K) y Value (V)

Cada elemento de la secuencia se representa mediante tres vectores:

- **Query (Q)**: La consulta que busca información relevante.
- **Key (K)**: La clave que identifica la información en otros elementos.
- **Value (V)**: El contenido que se utiliza para generar la salida.

Estos vectores permiten calcular la atención que un elemento debe prestar a otros en la secuencia.

3.3.2 Mecanismo de Atención Escalada (*Scaled Dot-Product Attention*)

El cálculo de atención se realiza mediante:

$$\text{Atención} = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Este mecanismo permite al modelo enfocarse en las partes relevantes de la secuencia para cada paso de procesamiento.

3.3.3 Atención Multicabeza (*Multi-Head Attention*)

La atención multicabeza mejora la capacidad del modelo para capturar diferentes tipos de relaciones en la secuencia, procesando múltiples atenciones en paralelo.

4 Modelos de Lenguaje Grandes

Los Modelos de Lenguaje Grandes (LLMs, *Large Language Models*) han revolucionado el campo del procesamiento del lenguaje natural (NLP, *Natural Language Processing*), permitiendo que las máquinas comprendan y generen

texto con un nivel de coherencia y fluidez sin precedentes. Para entender cómo funcionan estos modelos, es esencial conocer conceptos clave como tokens, embeddings, codificación posicional y técnicas avanzadas como RAG y PEFT.

4.1 Introducción a los Tokens

Los **tokens** son las unidades básicas en las que se divide el texto para su procesamiento por parte de los modelos de lenguaje. No son necesariamente palabras completas; pueden ser caracteres, subpalabras o símbolos especiales. La tokenización es el proceso de convertir una secuencia de texto en una lista de tokens.

4.1.1 Ejemplo de Tokenización y Análisis Semántico

Consideremos un ejemplo utilizando un modelo de lenguaje en inglés. Supongamos que tenemos una lista de preguntas:

- "What is the capital of France?"
- "What is the smallest state in India?"
- "What is the smallest state in the US?"
- "What is the longest river in the world?"
- "What is the highest mountain in Africa?"

Cada pregunta se tokeniza y se convierte en una representación vectorial mediante embeddings. En una visualización denominada *Output Vector Projection*, se muestran las relaciones entre las preguntas en un espacio bidimensional.

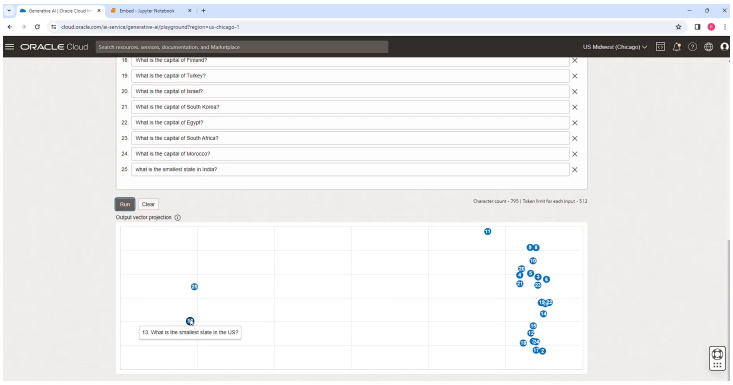

Figure 1: Proyección de vectores de salida mostrando la agrupación semántica de las preguntas.

En la visualización, los números dentro de círculos representan las distintas preguntas, donde cada número corresponde a una pregunta específica de la lista. Por ejemplo, el número "13" está vinculado a la pregunta "What is the smallest state in the US?". La proyección agrupa visualmente preguntas similares basándose en relaciones semánticas, una aproximación común en el procesamiento del lenguaje natural para agrupar entradas según su significado subyacente.

La proximidad de los números en la proyección indica la similitud semántica entre las preguntas. Por ejemplo, las preguntas sobre el estado más pequeño en diferentes países se agrupan juntas, reflejando que el modelo ha capturado su similitud en contenido.

4.2 Introducción a los Embeddings

Los **embeddings** son representaciones numéricas de los tokens en espacios vectoriales de alta dimensión. Transforman los tokens, que son entidades discretas, en vectores continuos que capturan información semántica y sintáctica.

4.2.1 Importancia de los Embeddings

Los embeddings permiten que los modelos:

- **Capturen similitudes entre palabras y frases**: Palabras o frases con significados similares tendrán vectores cercanos en el espacio de

embeddings.

- **Manejen relaciones complejas**: Capturan relaciones semánticas y sintácticas, permitiendo al modelo entender contextos y significados más profundos.
- **Visualicen y agrupen datos**: Facilitan la representación gráfica de datos para análisis y comprensión, como en la Figura 1.

4.2.2 Proceso de Generación de Embeddings

Cada token obtenido tras la tokenización se transforma en un vector de dimensión fija mediante una capa de embeddings. Estos vectores son entrenables y se ajustan durante el entrenamiento del modelo para optimizar su rendimiento en tareas específicas.

4.3 Conversión Inicial de la Secuencia

Con la comprensión de tokens y embeddings, podemos describir el proceso inicial que sigue un LLM al recibir texto de entrada:

1. **Tokenización**: El texto se divide en tokens utilizando un método adecuado (por ejemplo, Byte Pair Encoding o WordPiece).
2. **Generación de Embeddings**: Cada token se transforma en un vector numérico mediante la capa de embeddings.
3. **Codificación Posicional**: Se suman los embeddings con vectores posicionales para incorporar información sobre la posición de cada token en la secuencia.

4.4 Codificación Posicional

Los Transformers procesan las secuencias de entrada de manera no secuencial, es decir, consideran todos los tokens simultáneamente. Esto plantea el desafío de cómo mantener la información sobre el orden de los tokens, que es crucial para entender el contexto y el significado en el lenguaje natural.

4.4.1 Necesidad de la Codificación Posicional

A diferencia de las redes neuronales recurrentes (RNNs), que procesan secuencias de manera secuencial y, por tanto, tienen incorporada la información de orden, los Transformers requieren un mecanismo para representar la posición de cada token. Sin información posicional, el modelo no podría distinguir entre diferentes permutaciones de los mismos tokens, lo que afectaría su capacidad para entender y generar lenguaje coherente.

4.4.2 Primeros Intentos de Codificación

Una idea inicial podría ser asignar un número absoluto a cada posición, por ejemplo, 1 para el primer token, 2 para el segundo, y así sucesivamente. Cada posición tendría asociado un vector donde todas sus componentes son iguales al número de posición. Sin embargo, esto presenta problemas:

- **Valores Elevados**: En secuencias largas, los valores numéricos pueden volverse muy grandes, lo que puede afectar negativamente al aprendizaje del modelo, ya que los embeddings posicionales podrían dominar a los embeddings de los tokens.
- **Generalización Limitada**: Modelos entrenados en secuencias de longitud fija podrían no generalizar bien a secuencias más largas o más cortas.

Otra opción es normalizar las posiciones dentro de un rango, como [0, 1], dividiendo cada posición por la longitud total de la secuencia. Sin embargo, esto introduce ambigüedad:

- **Ambigüedad en Posiciones**: La misma posición normalizada puede corresponder a diferentes posiciones absolutas en secuencias de distinta longitud. Por ejemplo, en una secuencia de 5 tokens, la posición 2 correspondería a $2/5 = 0.4$, mientras que en una secuencia de 10 tokens, la posición 4 también correspondería a $4/10 = 0.4$.

4.4.3 Codificación Posicional mediante Codificación Binaria

Para evitar estos problemas, se podría pensar en representar las posiciones utilizando codificación binaria. Cada posición se representa mediante un vector de bits que codifica el número de posición en binario. Por ejemplo:

- Posición 1: (0001)
- Posición 2: (0010)
- Posición 3: (0011)
- Posición 4: (0100)
- Posición 5: (0101)
- Posición 6: (0110)

Esta representación tiene ventajas:

- **Valores Acotados**: Los vectores contienen solo 0s y 1s, evitando valores elevados que podrían interferir con los embeddings de los tokens.
- **Posiciones Únicas**: Cada posición tiene una representación única.

Sin embargo, la codificación binaria introduce discontinuidades y es de naturaleza discreta, mientras que las redes neuronales funcionan mejor con representaciones continuas y diferenciables.

4.4.4 De la Codificación Binaria a las Funciones Sinusoidales

Observando los patrones en la codificación binaria, podemos notar que cada bit en la representación binaria alterna entre 0 y 1 a diferentes frecuencias:

- **Bit menos significativo (LSB)**: Cambia cada posición (0, 1, 0, 1, ...).
- **Siguiente bit**: Cambia cada dos posiciones (0, 0, 1, 1, 0, 0, 1, 1, ...).
- **Bits superiores**: Cambian cada cuatro, ocho posiciones, etc.

Estos patrones de alternancia discreta pueden ser vistos como señales que oscilan a diferentes frecuencias. Para adaptar esto al dominio continuo y aprovechar la naturaleza diferenciable de las redes neuronales, se utilizan funciones sinusoidales.

4.4.5 Codificación Posicional Sinusoidal

Los autores de "Attention is All You Need" [5] propusieron utilizar funciones seno y coseno para generar las codificaciones posicionales. La idea es que las funciones sinusoidales pueden generar patrones ondulatorios continuos a diferentes frecuencias, análogos a los patrones discretos observados en la codificación binaria.

Las fórmulas para calcular los *Positional Encodings* son:

$$\begin{aligned} \mathrm{PE}_{(pos,2i)} &= \sin\left(pos \times \omega_k\right), \\ \mathrm{PE}_{(pos,2i+1)} &= \cos\left(pos \times \omega_k\right), \end{aligned}$$

donde ω_k es la frecuencia angular definida como:

$$\omega_k = \frac{1}{10000^{\frac{2i}{d_{\text{model}}}}}$$

y:

- pos es la posición del token en la secuencia.
- i es el índice de la dimensión del vector.
- d_{model} es la dimensión total del embedding.

Esta definición asegura que las diferentes dimensiones del vector de codificación posicional correspondan a diferentes frecuencias sinusoidales.

4.4.6 Intuición Detrás de las Codificaciones Sinusoidales

Las funciones seno y coseno generan señales ondulatorias que oscilan a diferentes frecuencias para cada dimensión del vector de codificación posicional. Estas señales continuas permiten al modelo captar tanto la posición absoluta como la relativa de los tokens en la secuencia.

La variación de frecuencias en las dimensiones permite distinguir entre diferentes posiciones, ya que cada posición tendrá un patrón único de valores en su vector de codificación posicional.

4.4.7 Ventajas de las Codificaciones Sinusoidales

- **Representación Continua**: Las funciones sinusoidales proporcionan una representación continua y diferenciable, adecuada para redes neuronales.
- **Captura de Relaciones Relativas**: Gracias a las propiedades matemáticas de las funciones seno y coseno, el modelo puede aprender fácilmente a atender a posiciones relativas.
- **Generalización a Secuencias Más Largas**: Las funciones sinusoidales son periódicas y pueden extenderse a posiciones mayores sin necesidad de aprender nuevos parámetros.
- **No Añaden Parámetros Entrenables**: Las codificaciones sinusoidales se calculan mediante una fórmula fija y no requieren parámetros adicionales que deban ser entrenados.

4.4.8 Implementación en el Modelo

La codificación posicional se suma directamente a los embeddings de los tokens:

$$\text{Embedding de Entrada} = \text{Embedding de Token} + \text{Codificación Posicional}$$

Esta suma combina la información de contenido (del token) y de posición en una única representación que el modelo utiliza en las siguientes capas.

4.4.9 Visualización de las Codificaciones Posicionales

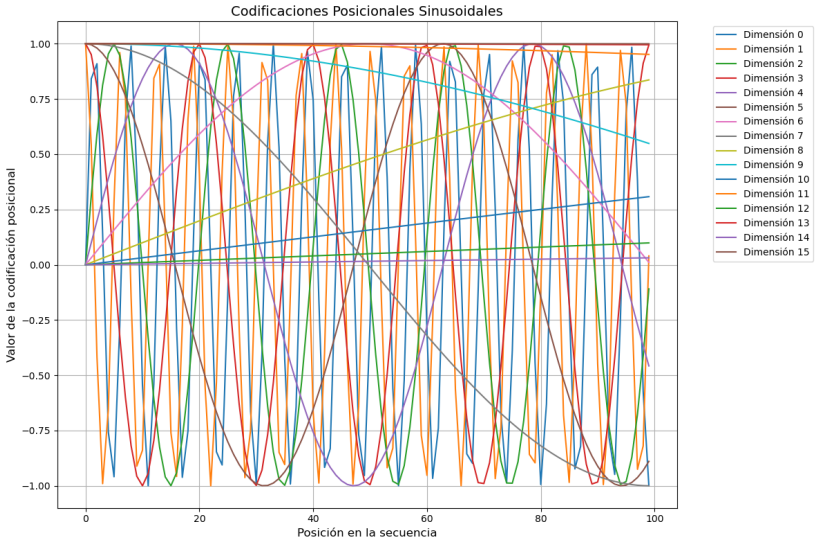

Figure 2: Representación gráfica de las codificaciones posicionales sinusoidales para diferentes posiciones y dimensiones.

En la Figura 2, cada línea representa una dimensión del vector de codificación posicional a lo largo de las posiciones de la secuencia. Las codificaciones posicionales sinusoidales permiten a los Transformers incorporar información sobre el orden de los tokens en una secuencia, algo fundamental para que el modelo pueda interpretar correctamente el significado de las frases.

Los valores de la codificación posicional varían en patrones sinusoidales para cada dimensión, alternando entre seno y coseno, según las fórmulas propuestas por *Vaswani et al.* en su trabajo "Attention Is All You Need" [5]:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \quad \mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right),$$

donde:

- pos es la posición del token en la secuencia,
- i es el índice de la dimensión en el embedding,

- d_{model} es la dimensión del vector de embeddings (en este caso, 16 dimensiones).

Este método asegura que las codificaciones sean únicas para cada posición y permita al modelo aprender relaciones relativas entre las posiciones de los tokens. Las primeras dimensiones (*Dimensión 0, Dimensión 1, etc.*) capturan patrones de alta frecuencia, lo que ayuda a aprender dependencias locales entre palabras cercanas, mientras que las últimas dimensiones (*Dimensión 14, Dimensión 15*) oscilan más lentamente, capturando relaciones a mayor distancia en la secuencia.

Las codificaciones posicionales sinusoidales permiten que el modelo generalice mejor a secuencias de diferentes longitudes, ya que el patrón periódico de las funciones seno y coseno garantiza que el modelo pueda capturar tanto dependencias locales como globales de manera eficiente.

El uso de estas codificaciones se suma directamente a los embeddings de tokens, lo que permite que la información posicional y semántica se integren antes de ser procesadas por las capas de atención.

$$\text{Input Embedding} = \text{Token Embedding} + \text{Positional Encoding}$$

Este proceso de sumar codificaciones posicionales y embeddings de tokens facilita que los modelos Transformer manejen secuencias de diferentes longitudes y aprendan relaciones posicionales complejas, lo que es esencial en tareas como traducción automática, resumen de texto y generación de lenguaje natural.

4.4.10 Importancia en el Transformer

La codificación posicional es fundamental para que el Transformer pueda procesar secuencias de manera efectiva, manteniendo la información de orden necesaria para entender el lenguaje. Sin este componente, el modelo no podría distinguir entre secuencias con los mismos tokens en diferente orden, lo que afectaría su rendimiento.

Además, al permitir el procesamiento en paralelo de todos los tokens, los Transformers logran una mayor eficiencia computacional y escalabilidad en comparación con las RNNs, que procesan secuencias de manera secuencial.

4.5 Ventajas de los Transformers sobre las RNNs

La capacidad de procesar secuencias en paralelo es una de las principales ventajas de los Transformers sobre las RNNs. Esto permite:

- **Mayor Eficiencia Computacional**: Al aprovechar el paralelismo, se reducen significativamente los tiempos de entrenamiento, permitiendo entrenar modelos con grandes cantidades de datos.
- **Escalabilidad**: Es posible entrenar modelos con un gran número de parámetros, como GPT-3 o Megatron-Turing, algo inviable con arquitecturas secuenciales.
- **Captura de Dependencias a Largo Plazo**: Los mecanismos de atención permiten que el modelo considere relaciones entre tokens distantes en la secuencia, superando las limitaciones de las RNNs en este aspecto.

4.6 Resumen

En resumen, la codificación posicional es un componente esencial en los Transformers para mantener la información de orden en las secuencias de entrada. A través de funciones sinusoidales, se logra una representación continua y eficiente de las posiciones de los tokens, permitiendo que el modelo procese secuencias en paralelo sin perder el contexto posicional.

La combinación de la codificación posicional con los mecanismos de atención y el procesamiento paralelo ha llevado al desarrollo de modelos de lenguaje grandes y poderosos, capaces de realizar tareas complejas en procesamiento del lenguaje natural y más allá.

5 Práctica: Uso de un Modelo de Lenguaje en LM Studio

Para poner en práctica lo aprendido, proponemos el uso de un modelo en LM Studio, una plataforma diseñada para facilitar la interacción con modelos de lenguaje generativos. Esta práctica permitirá experimentar de primera mano cómo funcionan estos modelos en la generación de texto.

Pasos:

1. Acceder a la plataforma LM Studio.
2. Cargar un conjunto de datos simple, como artículos o preguntas comunes.
3. Probar el modelo para generar respuestas basadas en el texto proporcionado.

6 Conclusiones

Los Modelos de Lenguaje Grandes han transformado el campo del procesamiento del lenguaje natural. La incorporación de técnicas como la codificación posicional y los mecanismos de atención ha permitido superar limitaciones de arquitecturas previas, como las RNNs. Además, avances en estrategias de entrenamiento y adaptación, como RAG y PEFT, continúan expandiendo las capacidades y aplicaciones de estos modelos. Comprender estos conceptos es esencial para aprovechar al máximo sus capacidades y desarrollar aplicaciones innovadoras en diversos campos.

References

[1] Oracle, *Oracle Cloud Infrastructure 2024 Generative AI Certified Professional*, Oracle, 2024.

[2] Oracle, *Oracle Cloud Infrastructure 2024 Certified AI Foundations Associate*, Oracle, 2024.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, y I. Polosukhin, *Attention Is All You Need*, arXiv preprint arXiv:1706.03762v7, 2017.

[4] Author(s), *Deep Learning and Computational Physics (Lecture Notes)*, arXiv preprint arXiv:2301.00942, 2023.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, y I. Polosukhin, *Attention Is All You Need*, arXiv preprint arXiv:1706.03762v7, 2017.

[6] A. Kazemnejad, *Transformer Architecture: The Positional Encoding*, Disponible en: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/, 2019.