Índice

(Versión simple)

Cualquier red de neuronas, y por lo tanto los transformers, no entienden las palabras como los humanos. Para procesarlas, necesitan convertirlas en **vectores numéricos**.

Por ejemplo, tomemos la frase: **"El gato juega"**

¿Cómo procesa esa frase un transformer?

1. Representación de Palabras con Embeddings

Cada palabra (supongamos que una palabra coincide con un token) se convierte en una lista de números (embedding), donde los números representan características como "animal", "acción", "objeto", etc. Supongamos que los embeddings para las 3 palabras de la frase son:

Palabra	Embedding (Ejemplo en 4 dimensiones)
El	[0.1, 0.2, 0.3, 0.4]
Gato	[0.7, 0.9, 0.2, 0.5]
Juega	[0.4, 0.8, 0.9, 0.3]

Estos números **no tienen un significado fijo**, sino que son aprendidos por el modelo.

Ejemplo Visual en 3D:

Imagina que **"Gato"** está representado en un espacio de 3 dimensiones así:

- **[0.7, 0.9, 0.2]**

Si agregamos "**Perro**", su embedding será similar porque también es un animal:

- **[0.6, 0.85, 0.25]**

Pero "**Juega**" tendrá un embedding diferente porque es un verbo:

- **[0.4, 0.8, 0.9]**

Así, palabras con significados similares debido a sus características, tendrán embeddings cercanos entre sí.

2. Encoding posicional: Recordando el Orden

El Transformer no solo necesita saber **qué palabras hay**, sino también **en qué orden aparecen**. Para eso, se usa un **vector especial de posición** que se suma a cada embedding.

Vamos a añadir un **Positional Encoding** y obtener, con la suma, el vector embedding final:

Palabra	Embedding	Positional Encoding	Embedding Final
El	[0.1, 0.2, 0.3, 0.4]	[0.05, 0.1, 0.15, 0.2]	[0.15, 0.3, 0.45, 0.6]
Gato	[0.7, 0.9, 0.2, 0.5]	[0.1, 0.15, 0.2, 0.25]	[0.8, 1.05, 0.4, 0.75]
Juega	[0.4, 0.8, 0.9, 0.3]	[0.15, 0.2, 0.25, 0.3]	[0.55, 1.0, 1.15, 0.6]

Ahora, el modelo no solo sabe el **significado** de cada palabra, sino también **su posición en la frase**. Conviene agregar que el **Positional Encoding** introduce una forma de "codificar" el orden de las palabras, ayudando al modelo a distinguir entre palabras que tienen el mismo significado pero aparecen en diferentes lugares de la secuencia.

3. Atención: Decidiendo qué es importante

El Transformer usa la **atención (self-attention)** para determinar **qué palabras se relacionan entre sí**.

Pensemos!!!! En nuestro ejemplo **"El gato juega"**, la palabra "gato" es importante para "juega", pero el artículo "el" no es tan importante. Lo presuponemos, pero también lo comprobaremos.

Para calcular esto, **cada palabra** incluye tres nuevos conceptos:

- **Query (Q)** → Lo que busca ("¿A qué palabras debo prestar atención?")
- **Key (K)** → Su identidad ("¿Qué importante soy para las demás palabras?")
- **Value (V)** → La información que aporta ("¿Qué información tengo para compartir?")

Supongamos que los valores Q, K y V (consulta, clave y valor, simplificados en 3 dimensiones) para cada palabra son:

Palabra	Q (Query)	K (Key)	V (Value)
El	[0.1, 0.2, 0.3]	[0.2, 0.3, 0.1]	[0.05, 0.1, 0.15]
Gato	[0.5, 0.6, 0.7]	[0.6, 0.5, 0.4]	[0.7, 0.9, 0.2]
Juega	[0.8, 0.7, 0.6]	[0.9, 0.8, 0.5]	[0.4, 0.8, 0.9]

Cálculo de Atención

El Transformer compara cada **Query** con cada **Key** y calcula qué tan fuerte es la conexión.

Por ejemplo, "Gato" y "Juega" tienen valores cercanos, lo que indica que están relacionados.

El cálculo que realiza el modelo es largo; multiplica **Q × K^T** (hace una normalización con softmax y lo multiplica por V) y usa esos valores para asignar **importancias (pesos)** a cada palabra. El resultado de **Q × K^T** es una **matriz de similitudes**, y en ella, los valores altos indican palabras que están relacionadas entre sí.

De forma simple, nuestros **resultados de la atención** podríamos predecirlos usando el producto escalar. El Transformer calcula la similitud entre las palabras usando el **producto escalar** entre **Q** y **K**. Si el resultado es alto, las palabras están muy relacionadas; si es bajo, no tanto.

Ejemplo para "Gato":

Queremos saber cómo se relaciona **"Gato"** con **"El"**, **"Gato"** y **"Juega"**.

- **"Gato" vs "El"**:

$$Q_{Gato} \cdot K_{El} = (0.5 \times 0.2) + (0.6 \times 0.3) + (0.7 \times 0.1) = 0.1 + 0.18 + 0.07 = 0.35$$

- **"Gato" vs "Gato"**:

$$Q_{Gato} \cdot K_{Gato} = (0.5 \times 0.6) + (0.6 \times 0.5) + (0.7 \times 0.4) = 0.3 + 0.3 + 0.28 = 0.88$$

- **"Gato" vs "Juega"**:

$$Q_{Gato} \cdot K_{Juega} = (0.5 \times 0.9) + (0.6 \times 0.8) + (0.7 \times 0.5) = 0.45 + 0.48 + 0.35 = 1.28$$

Para evitar valores demasiado grandes en el producto se suele hacer un escalado, que consiste en dividir el producto escalar por **la raíz cuadrada de la dimensión de los vectores**, que en nuestro ejemplo es 3:

$$\text{Escalado} = \frac{\text{Producto Escalar}}{\sqrt{3}} \approx \frac{\text{Producto Escalar}}{1.732}$$

Resultados para "Gato":

Producto escalar	Escalado
Similitud con "El": **0.35**	$\frac{0.35}{1.732} = 0.20$
Similitud consigo mismo: **0.88**	$\frac{0.88}{1.732} = 0.51$
Similitud con "Juega": **1.28**	$\frac{1.28}{1.732} = 0.74$

- "Gato" → Alta relación con "Juega" ✅

Si hacemos el producto escalar con “El” y las otras palabras, veremos que:

- "El" → Relación baja con las demás ❌

En el caso de **Self-Attention** → Se calcula qué palabras son más importantes entre sí (para cada una).

💡 **Problema**: Si usamos solo **una** atención, el modelo puede enfocarse en **una sola relación** entre palabras. Pero en el lenguaje, hay muchas relaciones diferentes:

- Algunas palabras se relacionan por significado (**quién hace qué**).
- Otras se relacionan por gramática (**tiempo, género, etc.**).

La solución es usar **Multi-Head Attention** → Se analizan diferentes aspectos de la frase en paralelo. Permite que el Transformer **mire las palabras de distintas formas al mismo tiempo**.

Si quieres saber más con la atención multi cabeza, vete al apéndice.

5. Feed-Forward: Refinando la Información

Después de decidir qué palabras son importantes, el modelo **transforma** los embeddings con una red neuronal adicional para hacerlos más útiles. Los modelos de Transformers utilizan redes neuronales adicionales para **refinar** los embeddings y hacerlos más útiles para la tarea en cuestión.

Por ejemplo, si antes "Gato" tenía este embedding:

📍 **[0.8, 1.05, 0.4, 0.75]**

Después del procesamiento, podría cambiar a:

📍 **[0.9, 1.1, 0.5, 0.85]**

Este paso ayuda al modelo a entender mejor los significados.

6. Conexión residual y Normalización: Manteniendo el Equilibrio

Antes de dar la salida final, el modelo revisa que la información esté bien escalada en número y evita errores al sumar los valores previos con los nuevos. El embedding final **representa la información procesada del input**, capturando relaciones y contexto del texto original. A veces, cuando se hacen muchos cambios, la información se puede **desordenar** o perder. Para evitar

esto, el Transformer **mantiene una copia de la información original** y la **mezcla** con la nueva información. Esto se llama **conexión residual**.

Luego, el modelo **normaliza** la información para que todos los números estén bien equilibrados y el modelo no se "pierda". Es como si al hacer muchos cambios, te aseguras de que todo siga siendo claro y organizado.

Si quieres saber más la conexión residual y la normalización, vete al apéndice

7. Salida: Texto Entendido o Generado

Finalmente, el modelo usa los embeddings finales procesados de las entradas para:

✅ **Para generación de texto (GPT, BERT, T5, etc.)** → Se pasa a una capa **Softmax** para predecir la siguiente palabra.

✅ **Para clasificación (ej. análisis de sentimiento)** → Se conecta a una **capa densa** y se predice la clase.

✅ **Para traducción automática (ej. Google Translate)** → Se pasa a un **decodificador** para generar la oración en otro idioma.

Por ejemplo, si el modelo es **GPT**, podría continuar la oración:

📌 Entrada: **"El gato juega"**

📌 Salida generada: **"con una pelota en el jardín."**

Esto es posible porque ha aprendido relaciones entre palabras basadas en sus embeddings.

Apéndice “Saber más”

1. Atención multi-cabeza

Ejemplo Numérico: Multi-Head Attention en acción

Vamos a dividir el cálculo de atención en **dos cabezas**. Esto significa que en lugar de calcular una sola matriz de **Q, K y V**, ahora tenemos **dos conjuntos diferentes** de estos valores.

Seguimos con la frase:

📌 **"El gato juega"**

Los embeddings para cada palabra ya los tenemos:

- **Gato** → [0.7, 0.9, 0.2]
- **Perro** → [0.6, 0.85, 0.25]
- **Juega** → [0.4, 0.8, 0.9]

Ahora, **para cada cabeza de atención**, creamos versiones diferentes de **Q, K y V**.

Ejemplo de los valores **para la primera cabeza**:

- **Q** (Consultas): [0.5, 0.6, 0.7]
- **K** (Claves): [0.2, 0.3, 0.1]
- **V** (Valores): [0.4, 0.5, 0.6]

Ejemplo de los valores **para la segunda cabeza**:

- **Q** (Consultas): [0.1, 0.9, 0.3]
- **K** (Claves): [0.7, 0.5, 0.2]
- **V** (Valores): [0.8, 0.3, 0.9]

Cada cabeza **calcula su propia atención** por separado (usando el mismo método que se explicó, con el producto escalar).

Luego, obtenemos dos matrices de atención **diferentes**, porque cada cabeza mira las palabras de una forma distinta.

Unir las cabezas: Concatenación

Después de calcular la atención en cada cabeza, el Transformer **toma las salidas y las une**.

🔹 **¿Cómo?** Se concatenan los vectores de cada cabeza en un solo vector más grande.

Ejemplo:

- **Primera cabeza** genera el resultado [0.6, 0.3, 0.8]
- **Segunda cabeza** genera el resultado [0.7, 0.9, 0.4]
- **Salida después de concatenar**: [0.6, 0.3, 0.8, 0.7, 0.9, 0.4]

Después, este vector se pasa por una **capa lineal** que mezcla toda la información y la deja con la dimensión original.

2. Conexión residual y normalización

Paso 1: Conexión Residual (Skip Connection)

Después de cada bloque de **Atención Multi-Cabeza** o de **Feed Forward**, se usa una **conexión residual** para ayudar al modelo a aprender mejor y estabilizar el flujo de gradientes.

🔹 **Fórmula matemática de la conexión residual**

Si tenemos una entrada **X** y una transformación del bloque (como la auto-atención o el feed forward) llamada **F(X)**, la conexión residual se define como:

Salida Residual=X+F(X)

📌 **¿Por qué se usa?**

1. **Evita que la información original se pierda** → El modelo aún conserva la señal original X.
2. **Facilita el aprendizaje** → Ayuda a que los gradientes fluyan mejor en redes profundas (como en ResNets).
3. **Permite actualizaciones pequeñas** → Si F(X) es un ajuste pequeño, la red solo cambia ligeramente el estado anterior, en lugar de sobrescribirlo por completo.

Ejemplo Numérico:

Si $X = [0.5, 0.2, -0.3]$ y $F(X) = [0.1, -0.05, 0.2]$, la salida residual es:

$$[0.5, 0.2, -0.3] + [0.1, -0.05, 0.2] = [0.6, 0.15, -0.1]$$

Paso 2: Normalización de Capa (Layer Normalization)

Antes de pasar al siguiente bloque del Transformer, la salida residual se normaliza para mantener valores bien escalados y estabilizar el entrenamiento.

📌 **¿Por qué se usa?**

1. **Evita que los valores exploten** → Mantiene activaciones estables en cada capa.
2. **Hace que el entrenamiento sea más rápido** → Reduce la dependencia de una inicialización cuidadosa de los pesos.
3. **Mejora la generalización** → Evita que el modelo dependa de valores extremos.

Ejemplo Numérico: Supongamos que la salida residual es: [0.6,0.15,−0.1]

1. Calculamos la **media**:

$$\mu = \frac{0.6 + 0.15 - 0.1}{3} = 0.2167$$

2. Calculamos la **desviación estándar**:

$$\sigma = \sqrt{\frac{(0.6 - 0.2167)^2 + (0.15 - 0.2167)^2 + (-0.1 - 0.2167)^2}{3}}$$

$$\approx 0.304$$

3. Normalizamos

$$\hat{z}_1 = \frac{0.6 - 0.2167}{0.304} \approx 1.26, \quad \hat{z}_2 = \frac{0.15 - 0.2167}{0.304} \approx -0.22, \quad \hat{z}_3 = \frac{-0.1 - 0.2167}{0.304} \approx -1.06$$

4. Aplicamos los parámetros aprendibles (γ=1.5 y β=0.3):

$$\text{Salida Normalizada} = [1.26, -0.22, -1.06] \times 1.5 + 0.3$$

$$= [1.89, -0.03, -1.29]$$