

Procesamiento del Lenguaje Natural (NLP). Transformers: Attention is All you Need

Inteligencia Artificial e Ingeniería del Conocimiento

Constantino Antonio García Martínez

Universidad San Pablo Ceu

- Vaswani et al. Attention is all you need. Advances in neural information processing systems. 2017.

Motivación

Recordatorio:

- Word2Vec, GloVe: 1 palabra = 1 vector (embedding *estático*)
- Modelos actuales (BERT, GPT): embeddings se aprenden junto con la red
- **Pero los embeddings iniciales siguen siendo estáticos**

Recordatorio:

- Word2Vec, GloVe: 1 palabra = 1 vector (embedding *estático*)
- Modelos actuales (BERT, GPT): embeddings se aprenden junto con la red
- **Pero los embeddings iniciales siguen siendo estáticos**

Objetivo: Convertir embeddings estáticos en *contextuales*

Example: Embeddings de “Banco”

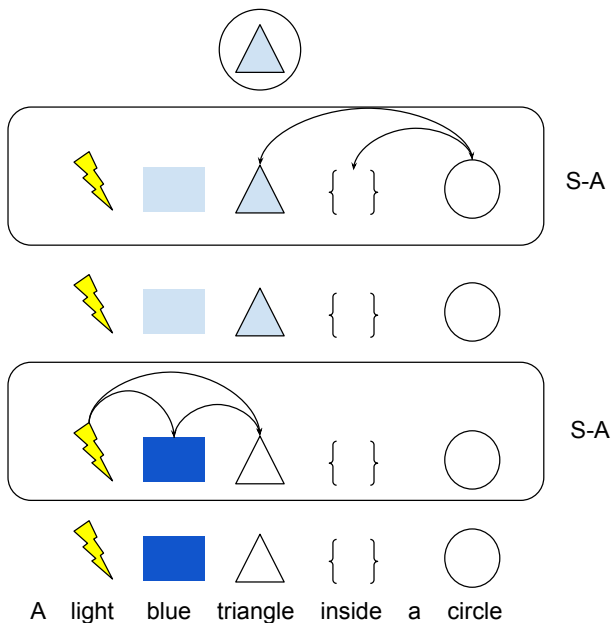
El embedding de “banco” debería depender del contexto:

- He ido al parque a sentarme en un banco.
- El banco me ha denegado el préstamo.

Idea clave: Atención

Nos gustaría que el embedding de “banco” prestase **atención** a su contexto.

Intuición: Embeddings que Evolucionan con el Contexto



Arquitectura Transformer

Arquitectura Transformer

Mecanismo de Atención

El Problema: Ponderar Relevancia

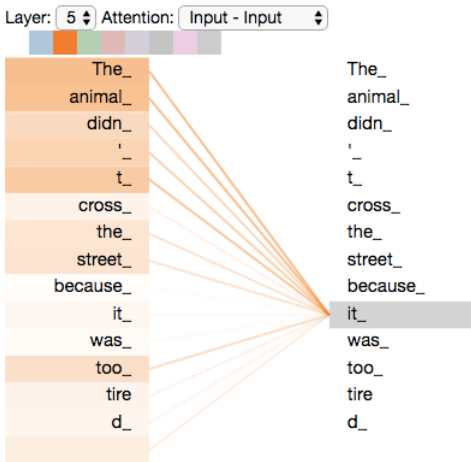
Objetivo: Dado un token, determinar qué otros tokens son relevantes

Ejemplo: “The animal didn’t cross the street because it was too tired”

El Problema: Ponderar Relevancia

Objetivo: Dado un token, determinar qué otros tokens son relevantes

Ejemplo: “The animal didn’t cross the street because it was too tired”



Idea: Usar pesos de “atención” para saber a qué debe atender “it”

Atención: Queries, Keys, Values

Analogía: Base de datos de características semánticas

Base de datos \mathcal{D} :

Key	Value (royal, male, power)
"king"	[0.9, 1.0, 0.8]
"queen"	[0.9, 0.0, 0.8]
"man"	[0.1, 1.0, 0.3]

Atención (hard retrieval):

$$\begin{aligned}\text{Attention}(\mathbf{q}_{\text{king}}, \mathcal{D}) &= \alpha_{\text{king}, \text{king}} \mathbf{v}_{\text{king}} + \alpha_{\text{king}, \text{queen}} \mathbf{v}_{\text{queen}} + \alpha_{\text{king}, \text{man}} \mathbf{v}_{\text{man}} \\ &= 1.0[0.9, 1.0, 0.8] + 0.0[0.9, 0.0, 0.8] + 0.0[0.1, 1.0, 0.3] \\ &= [0.9, 1.0, 0.8]\end{aligned}$$

$\alpha_{i,j}$ mide la atención que i debe prestar a j

Atención: Queries, Keys, Values

Analogía: Base de datos de características semánticas

Base de datos \mathcal{D} :

Key	Value (royal, male, power)
"king"	[0.9, 1.0, 0.8]
"queen"	[0.9, 0.0, 0.8]
"man"	[0.1, 1.0, 0.3]

Atención (soft retrieval): Query: "prince" (no está en \mathcal{D})

$$\begin{aligned}\text{Attention}(\mathbf{q}_{\text{prince}}, \mathcal{D}) &= \alpha_{\text{prince}, \text{king}} \mathbf{v}_{\text{king}} + \alpha_{\text{prince}, \text{queen}} \mathbf{v}_{\text{queen}} + \alpha_{\text{prince}, \text{man}} \mathbf{v}_{\text{man}} \\ &= 0.6[0.9, 1.0, 0.8] + 0.0[0.9, 0.0, 0.8] + 0.4[0.1, 1.0, 0.3] \\ &= [0.58, 1.0, 0.60]\end{aligned}$$

Resultado: Combinación que captura "royal + male + moderate power"

$\alpha_{i,j}$ mide la atención que i debe prestar a j

Implementación: Self-Attention Naive

Implementación Naive: Para derivar los valores de atención, podemos usar directamente los embeddings x_i

$$\alpha_{ij} \propto x_i \cdot x_j$$

Problemas:

- $x_i \cdot x_i$ siempre será alto (norma al cuadrado)
- Cada token tendrá *sesgo* hacia atenderse a sí mismo
- No hay separación entre roles: “¿qué busco?” vs “¿qué ofrezco?”

Implementación: Self-Attention Naive

Implementación Naive: Para derivar los valores de atención, podemos usar directamente los embeddings x_i

$$\alpha_{ij} \propto x_i \cdot x_j$$

Problemas:

- $x_i \cdot x_i$ siempre será alto (norma al cuadrado)
- Cada token tendrá *sesgo* hacia atenderse a sí mismo
- No hay separación entre roles: “¿qué busco?” vs “¿qué ofrezco?”

Solución: Proyectar x_i en tres espacios diferentes

- **Query** q_i : “¿qué estoy buscando?” (attended *from*)
- **Key** k_i : “¿qué información ofrezco?” (attended *to*)
- **Value** v_i : “¿qué contenido proporciono?” (context)

Solución: Proyecciones lineales aprendibles

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i$$

$$\mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

donde $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_k \times d_{model}}$ son matrices de pesos

Solución: Proyecciones lineales aprendibles

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i$$

$$\mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

donde $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_k \times d_{model}}$ son matrices de pesos

Ventajas:

- Separación de roles (query \neq key \neq value)
- $\mathbf{q}_i \cdot \mathbf{k}_i$ ya no es simplemente $\|\mathbf{x}_i\|^2$
- Las matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ se aprenden durante el entrenamiento

Para un token i :

$$\text{Attention}(\mathbf{q}_i, \mathbf{K}, \mathbf{V}) = \sum_j \alpha_{ij} \mathbf{v}_j$$

donde $\alpha_{i,j} \in [0, 1]$ y $\sum_j \alpha_{i,j} = 1$

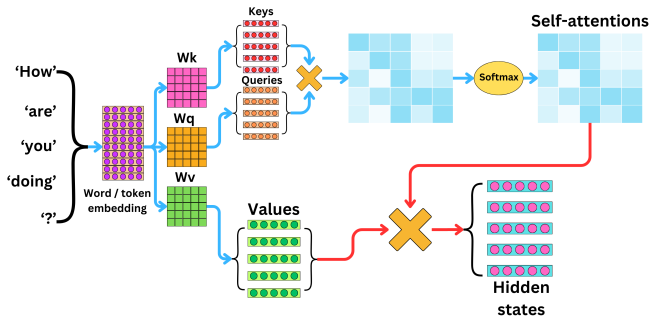
Para un token i :

$$\text{Attention}(\mathbf{q}_i, \mathbf{K}, \mathbf{V}) = \sum_j \alpha_{ij} \mathbf{v}_j$$

donde $\alpha_{i,j} \in [0, 1]$ y $\sum_j \alpha_{i,j} = 1$

$$\alpha_{ij} = \text{softmax} \left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \right)$$

Implementación de atención



<https://newsletter.theaiedge.io/p/understanding-the-self-attention>

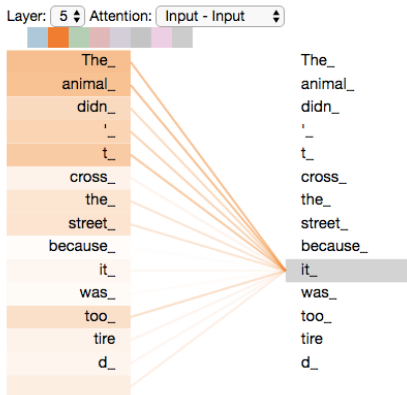
Arquitectura Transformer

Multi-Head Attention

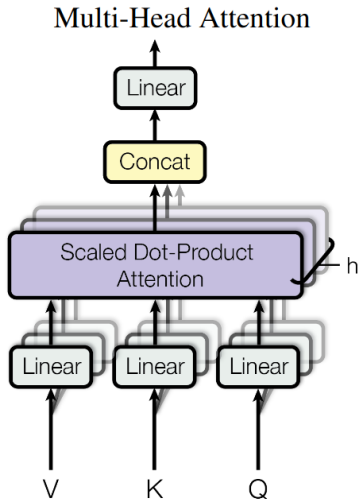
Multi-Head Attention

Problema: Una palabra puede tener que atender a diferentes cosas dependiendo del contexto

Example: “it” no atiende a “tired”



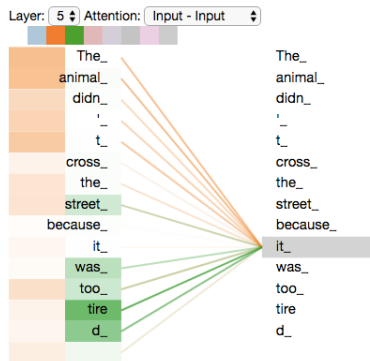
Solución: Repetir el mecanismo de atención h veces (h **Attention Heads**)



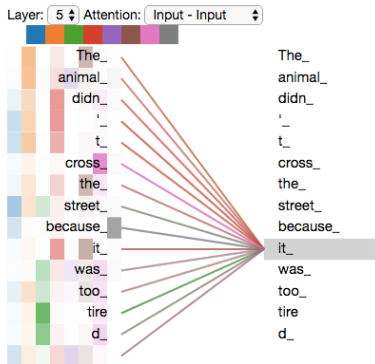
1

¹Fuente: Vaswani et al.

Multi-Head Attention



Multi-Head Attention



Arquitectura Transformer

Positional Encodings

El Problema: Self-Attention Ignora el Orden

Dos frases diferentes:

Frase 1: “The cat chased the mouse”

Frase 2: “The mouse chased the cat”

Dos frases diferentes:

Frase 1: “The cat chased the mouse”

Frase 2: “The mouse chased the cat”

¿Self-attention puede distinguirlas?

Dos frases diferentes:

Frase 1: “The cat chased the mouse”

Frase 2: “The mouse chased the cat”

¿Self-attention puede distinguirlas?

Respuesta: ¡NO!

Conclusión: Self-attention es *permutation-invariant*

- Opera como un *bag-of-words* sofisticado
- No captura información de orden secuencial

La Solución: Positional Encodings

Idea: Inyectar información de posición en los embeddings

$$x'_i = x_i + \mathbf{PE}(i)$$

donde $\mathbf{PE}(i) \in \mathbb{R}^{d_{model}}$ codifica la posición i

Idea: Inyectar información de posición en los embeddings

$$x'_i = x_i + PE(i)$$

donde $PE(i) \in \mathbb{R}^{d_{model}}$ codifica la posición i

Tipos de Positional Encodings:

1. **Sinusoidal** (Vaswani et al., 2017): Funciones sin/cos con diferentes frecuencias
 - Ventaja: Generaliza a secuencias más largas que las del entrenamiento
 - Ventaja: No requiere parámetros adicionales

Idea: Inyectar información de posición en los embeddings

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{PE}(i)$$

donde $\mathbf{PE}(i) \in \mathbb{R}^{d_{model}}$ codifica la posición i

Tipos de Positional Encodings:

1. **Sinusoidal** (Vaswani et al., 2017): Funciones sin/cos con diferentes frecuencias
 - Ventaja: Generaliza a secuencias más largas que las del entrenamiento
 - Ventaja: No requiere parámetros adicionales
2. **Aprendidos** (BERT, GPT): Vector \mathbf{p}_i por posición
 - Ventaja: Mayor flexibilidad (se optimizan durante entrenamiento)
 - Desventaja: Longitud máxima fija

Idea: Inyectar información de posición en los embeddings

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{PE}(i)$$

donde $\mathbf{PE}(i) \in \mathbb{R}^{d_{model}}$ codifica la posición i

Tipos de Positional Encodings:

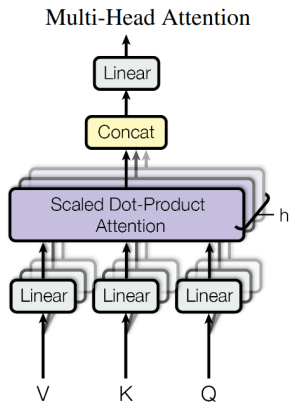
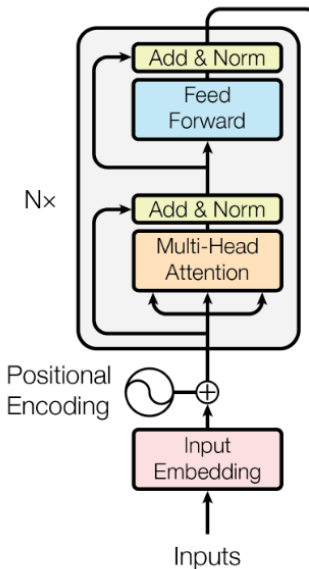
1. **Sinusoidal** (Vaswani et al., 2017): Funciones sin/cos con diferentes frecuencias
 - Ventaja: Generaliza a secuencias más largas que las del entrenamiento
 - Ventaja: No requiere parámetros adicionales
2. **Aprendidos** (BERT, GPT): Vector \mathbf{p}_i por posición
 - Ventaja: Mayor flexibilidad (se optimizan durante entrenamiento)
 - Desventaja: Longitud máxima fija

Resultado: Ahora $\mathbf{x}'_1 \neq \mathbf{x}'_2$ aunque $\mathbf{x}_1 = \mathbf{x}_2$ (mismo token, diferente posición)

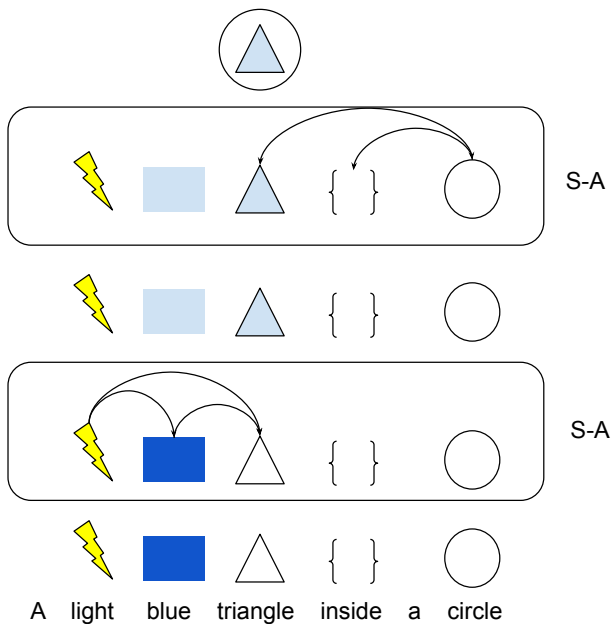
Arquitectura Transformer

El Transformer-Encoder

El Transformer-Encoder



El Transformer-Encoder: resultado



Example: `tokenization_and_embedding.py`