Tema 14. Transformers

Razonamiento y Representación del Conocimiento

Introducción

- Redes Neuronales → Combinación lineal del vector de entrada por los pesos de la red
- Evolución → deep learning
- Deep learning → Redes neuronales con muchas capas, principalmente **convolucionales**

Introducción

- Redes neuronales (convolucionales):
  - Capturan muy bien las relaciones entre datos
  - Pueden aprender estas relaciones
    - Cuando los datos están localmente ‘cerca’
- Problema: secuencias de datos largas
  - Lenguaje natural
  - Secuencias de video

Introducción

- Solución:

Transformers

- Sirven para encontrar relaciones entre secuencias de datos
  - Gran rango
  - Dependientes de la entrada

El gato está sobre la mesa
El gato, que es de color negro, está sobre la mesa

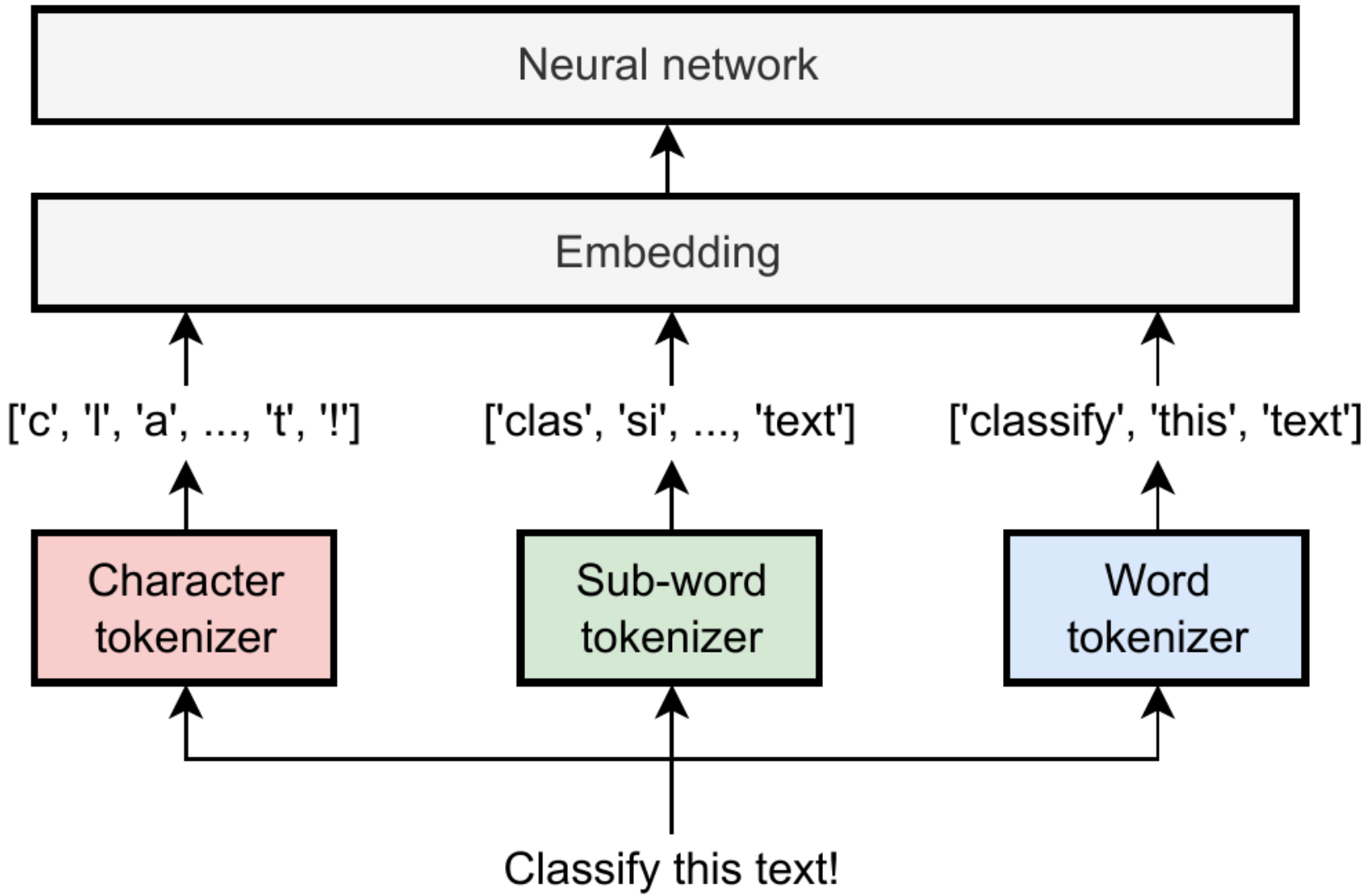
Transformers

- Secuencia de datos:
  1) Tokenizar
  2) Embeber
  3) Aprender

Tokenizer

- Separar la secuencia de datos de entrada en unidades discretas: tokens
  - Character Tokenizer: separa la entrada en caracteres
  - Word Tokenizer: separa la entrada en palabras
  - Sub-word Tokenizer: cualquier separación entre los dos anteriores

Tokenizer


Neural network
Embedding
['c', 'l', 'a', ..., 't', '!']
['clas', 'si', ..., 'text']
['classify', 'this', 'text']
Character tokenizer
Sub-word tokenizer
Word tokenizer
Classify this text!

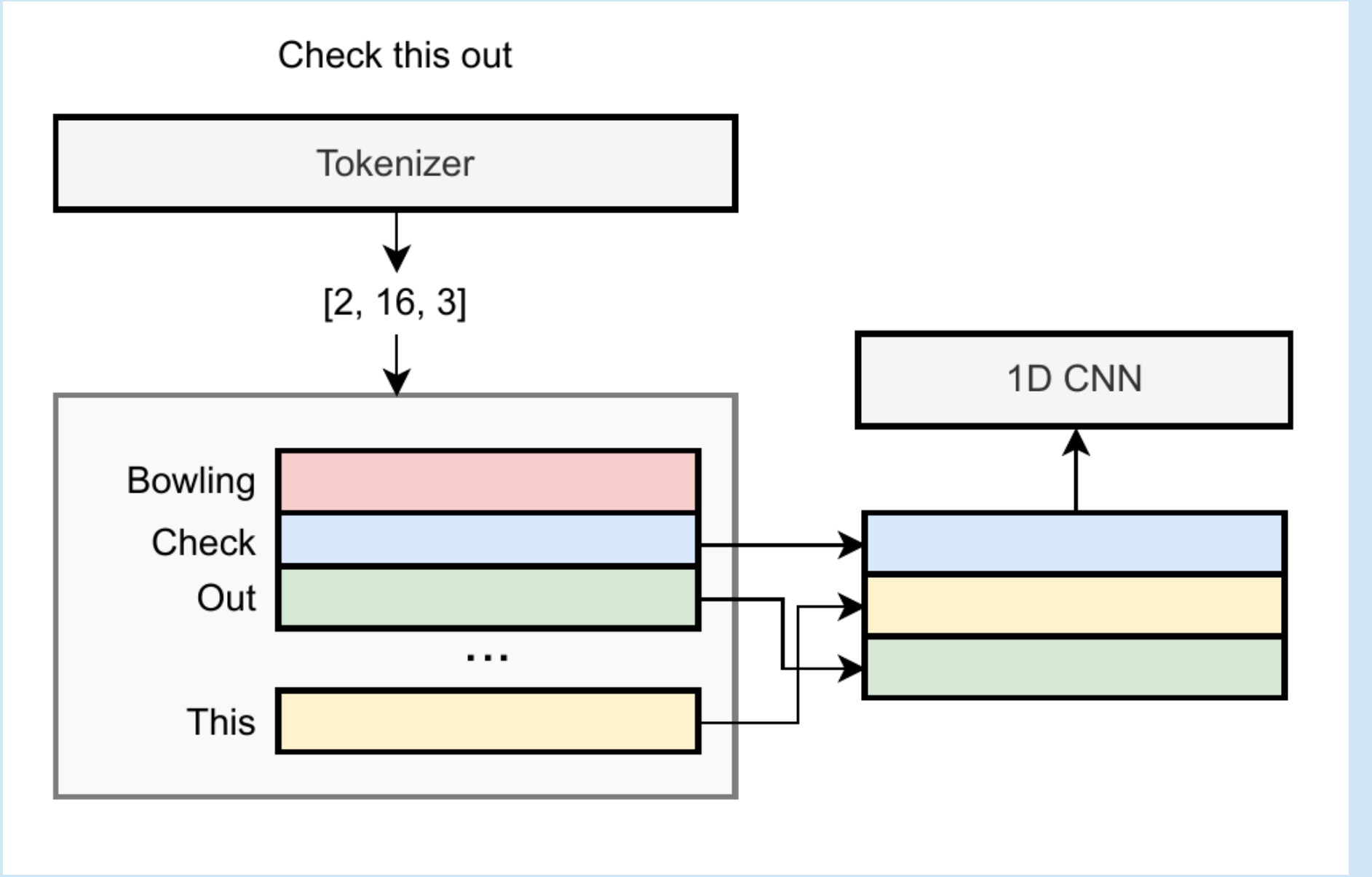
Tokenizer

- GPT Tokenizer → Byte-pair encoding
  - Combina tokens de 1 caracter* con tokens de palabras completas
    - Comienza con un conjunto inicial con los caracteres de la entrada: conjunto de 1-caracter n-grams
    - Después, las parejas de caracteres adyacentes más frecuentes son consideradas: 2-caracter n-grams
    - El proceso sigue hasta llegar a un vocabulario del tamaño deseado (GPT-4 100256 n-grams)

Embeddings

- Tokenizer: Sequence → tokens
- Embedding: Tokens → vectors
- El proceso de embedding consiste en asociar un vector de características a cada token
- El proceso se ajusta utilizando una CNN

Embeddings



Self-Attention

- Convolución larga
  - Necesitamos que las convoluciones tengan en cuenta toda la vecindad
  - g(·): attention scoring function

$$\mathbf{h}_i = \sum_{j=1}^{n} g(i-j)\mathbf{x}_j$$

$$\mathbf{h}_i = \sum_{j=1}^{n} g(\mathbf{x}_i, \mathbf{x}_j)\mathbf{x}_j$$

$$g(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

Self-Attention layer

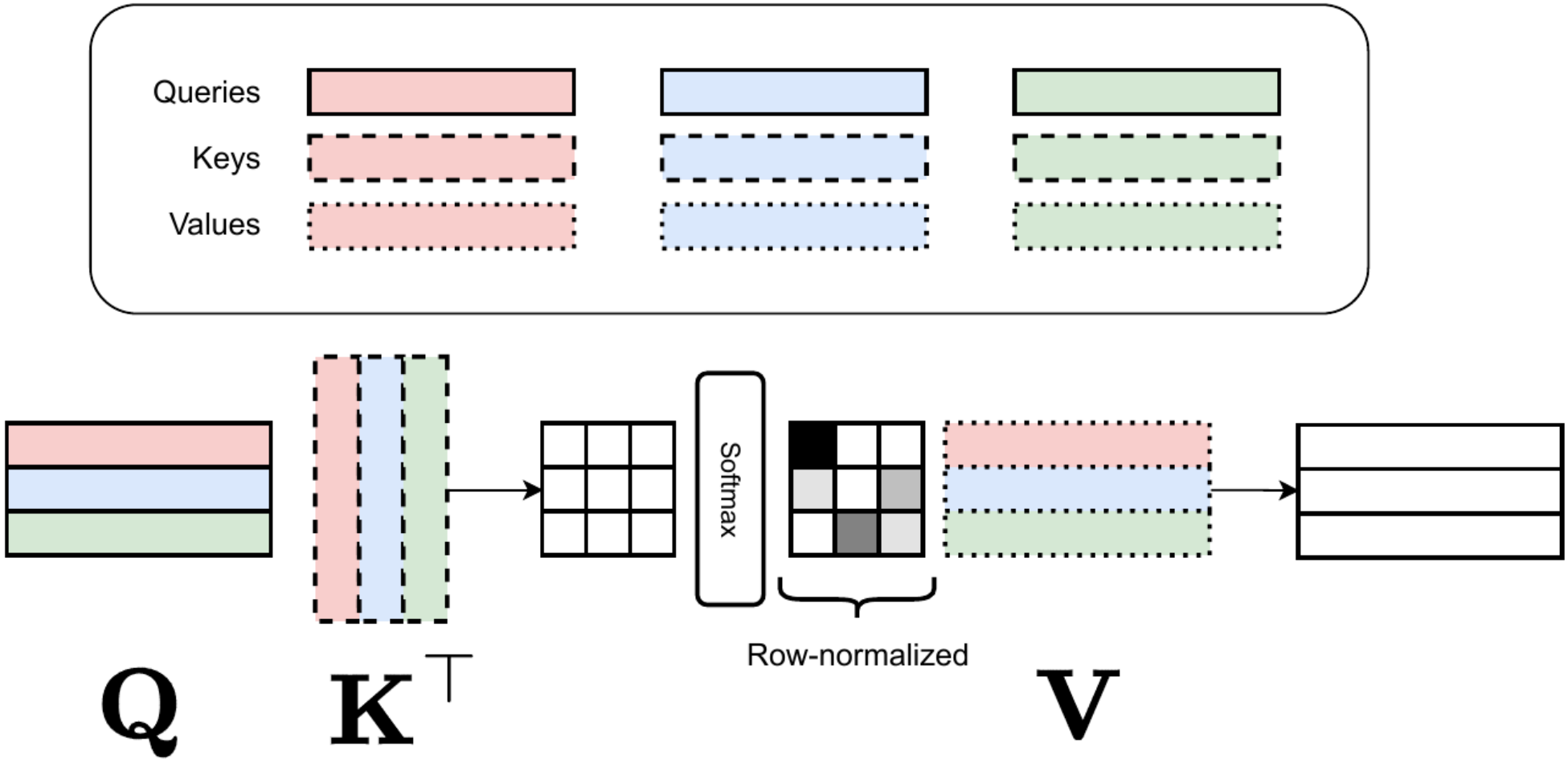
- Definimos 3 matrices ‘entrenables’

$$\begin{aligned} \text{Key tokens:} \quad & \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \\ \text{Value tokens:} \quad & \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i \\ \text{Query tokens:} \quad & \mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \end{aligned}$$

- Las combinamos para obtener una capa self-attention

$$\mathbf{h}_i = \sum_{j=1}^{n} \text{softmax}_j(g(\mathbf{q}_i, \mathbf{k}_j))\mathbf{v}_j$$

Self-Attention Layer



Multi-head attention

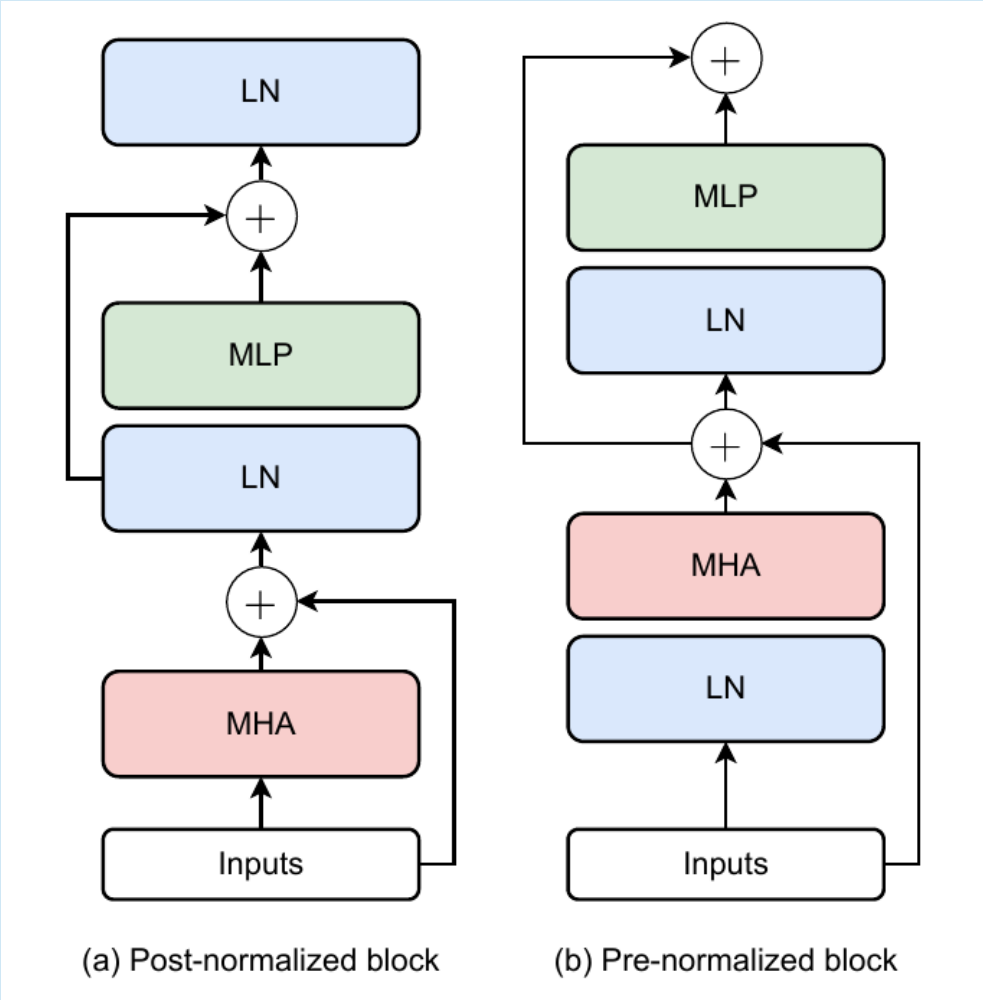
- Queremos encontrar relaciones entre los distintos tokens de una secuencia
- Disponemos de varias instancias de self-attentional layer → diferentes heads con sus 3 conjuntos de parámetros entrenables (W_q, W_k, W_v)

Transformer

- Podríamos construir el modelo a partir de múltiples bloques MHA
- Pero los mejores resultados se obtienen alternando un MHA con un MLP regularizadas con etapas de normalización de los datos

Transformer

- Bloque básico
  - Pre-normalizado
  - Post-normalizado

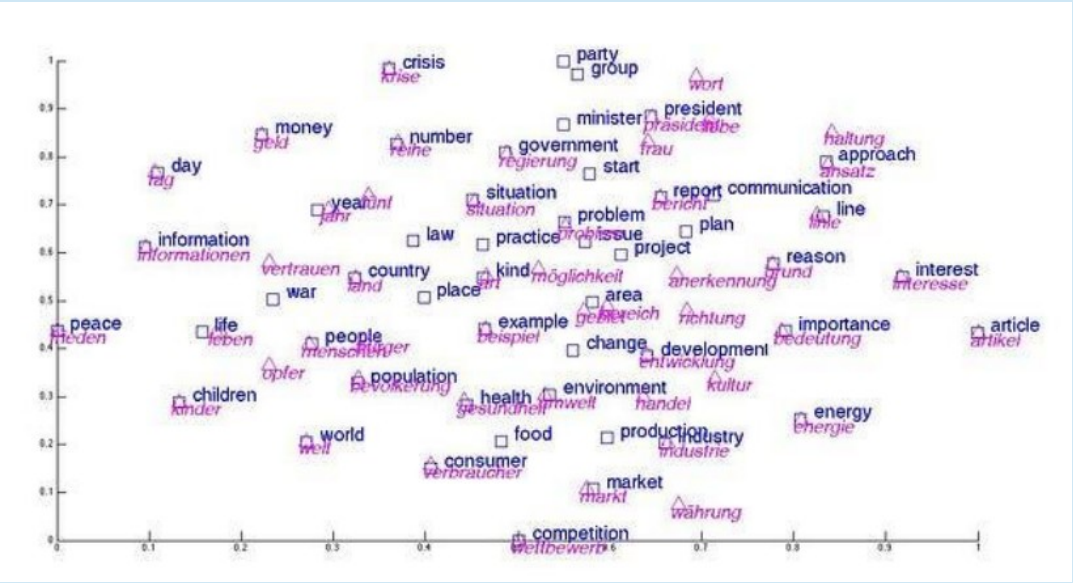


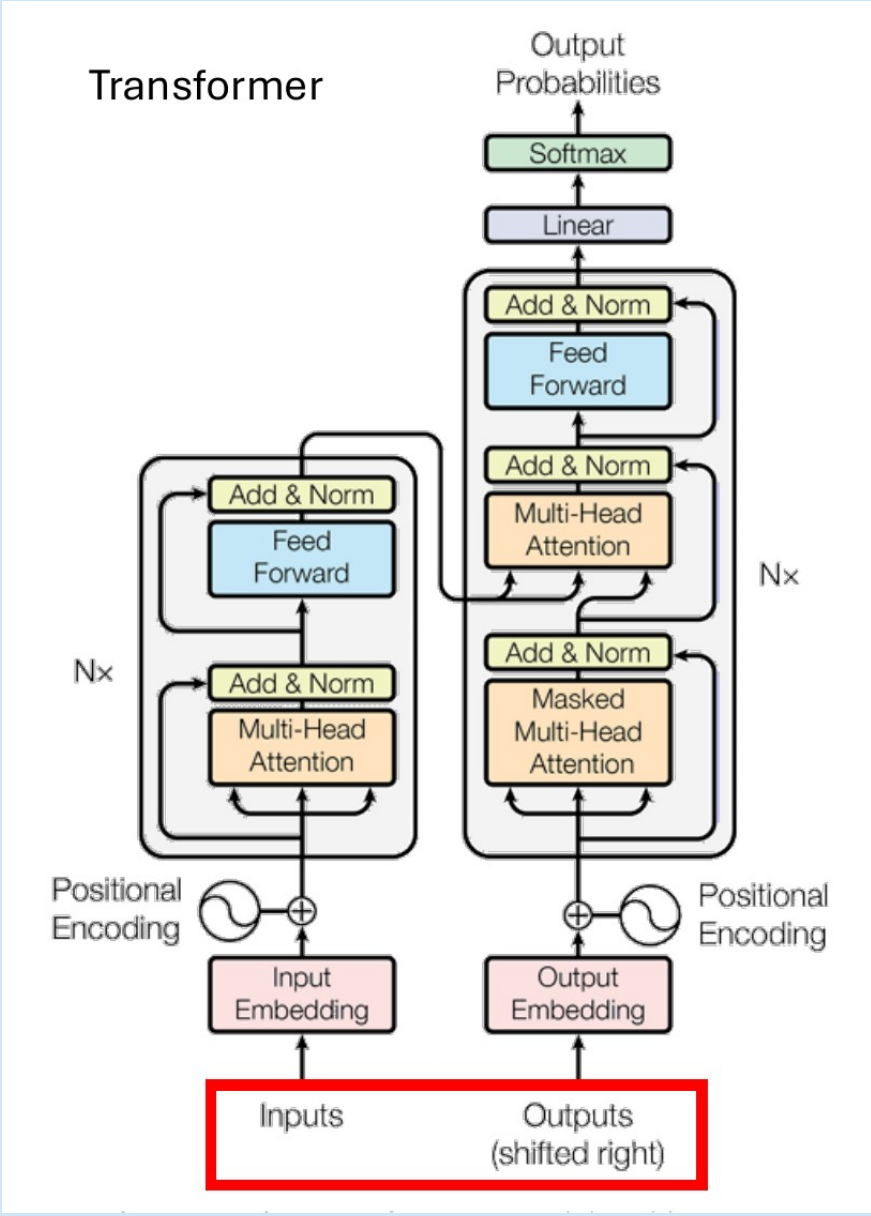
Transformer

- Modelo básico de transformer
  1) Tokenizar y embeber la entrada
  2) Añadir *absoute positional embeddings*
  3) Aplicar 1 o más bloques Transformer
  4) Aplicar una etapa de salida dependiendo de la aplicación

Transformer - ejemplo

- Ejemplo: traductor entre idiomas





Bibliiografía

- Alice’s Adventures in a differentiable wonderland de Simone Scardapane (arXiv)