Redes Neuronales para Lenguaje Natural

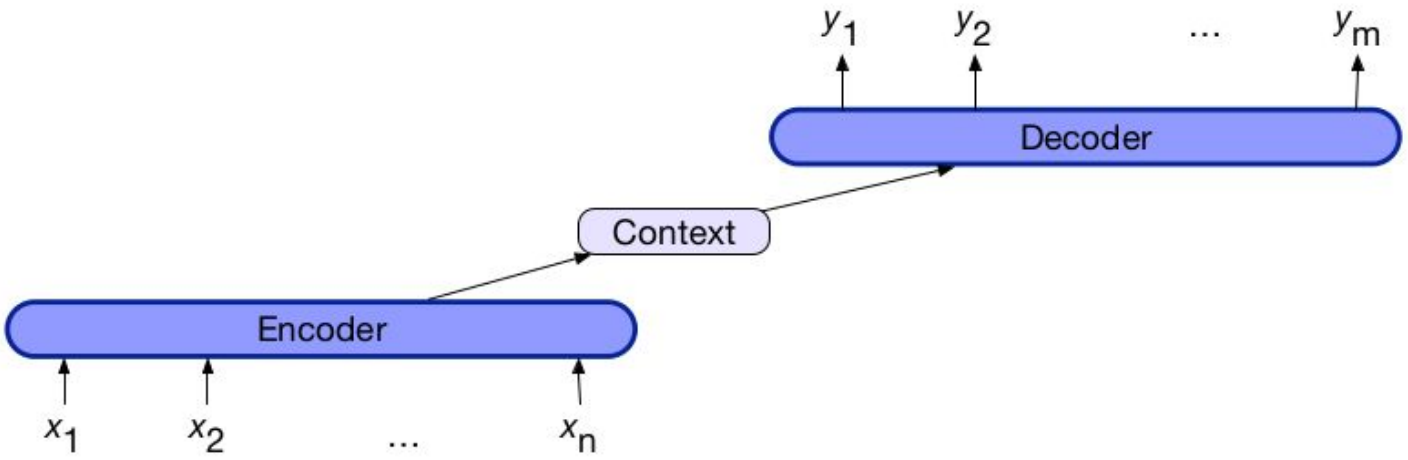
2024

Grupo de Procesamiento de Lenguaje Natural
Instituto de Computación

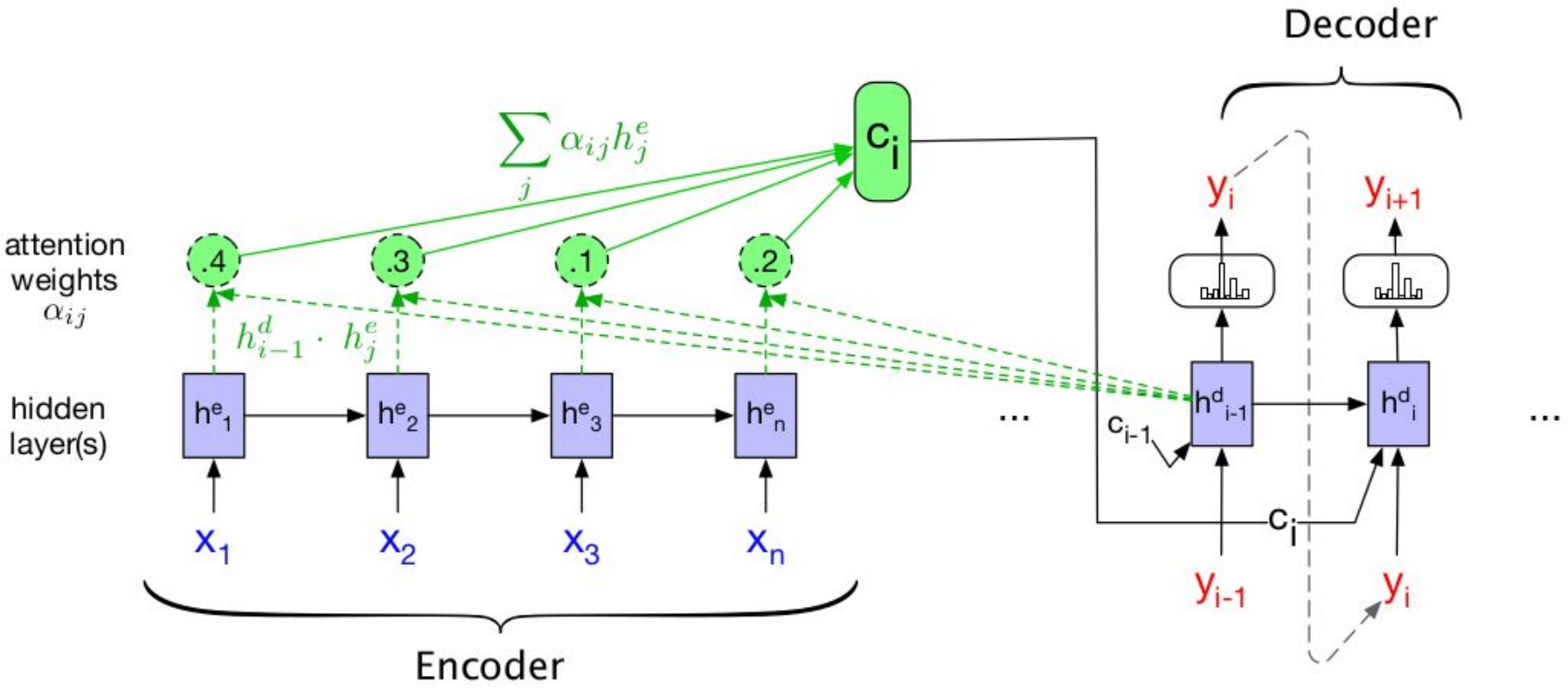
Arquitectura Encoder-Decoder

Una red compuesta por dos subredes:

- **Encoder**: red que codifica la entrada
- **Decoder**: red que decodifica (y construye) la salida
- **Context vector**: “embedding” de toda la secuencia de entrada



Mecanismo Atencional




Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

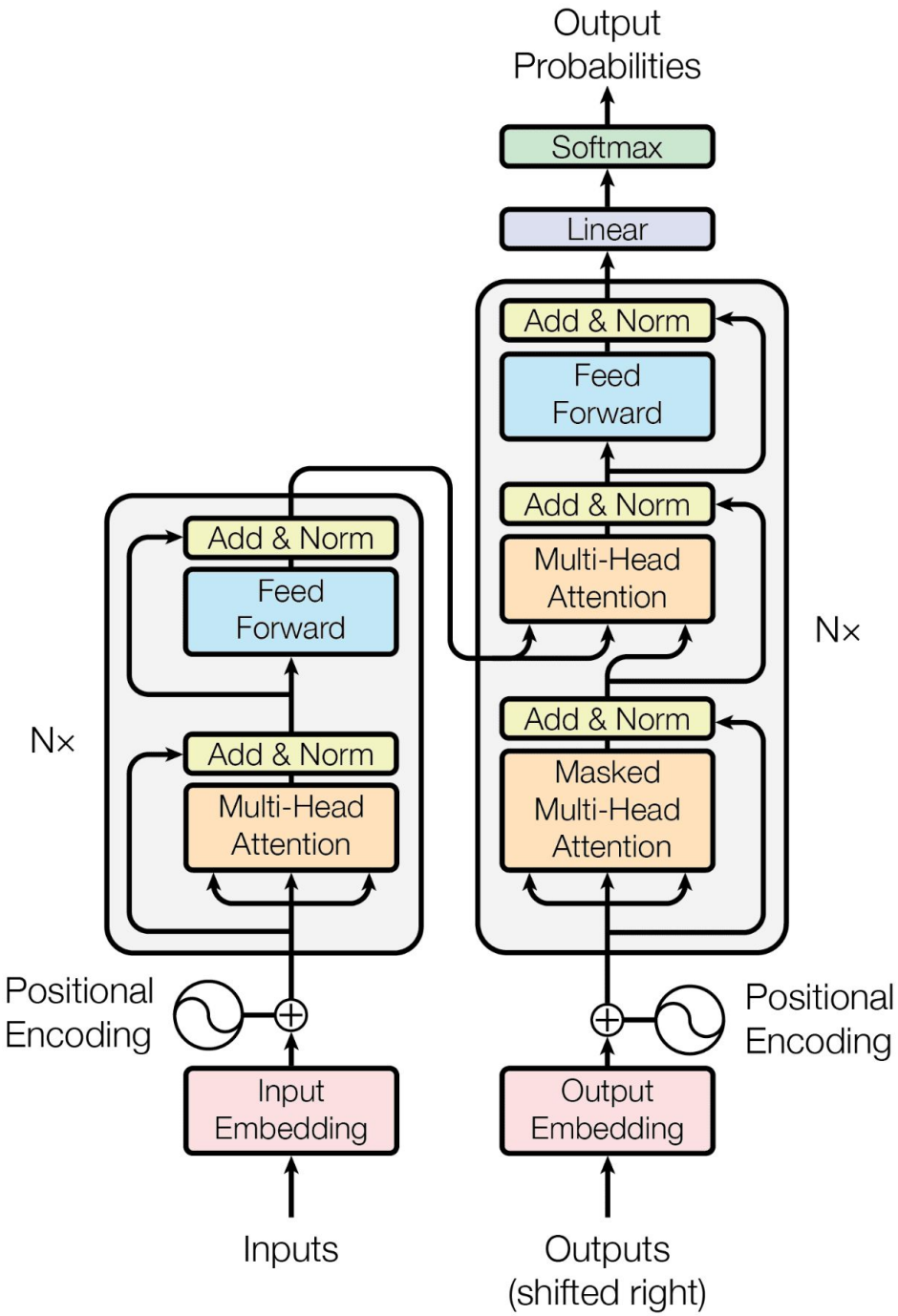
1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Transformer

Las RNNs obligan a procesar una palabra después de otra en orden

El transformer liberará esta restricción procesando todo en paralelo

Introduce varios conceptos a analizar:

- Self-attention / Autoatención
- Conexiones residuales
- Embeddings posicionales
- Masked attention y Cross-attention

…

Motivación

Los transformers son capaces de crear embeddings contextuales (no son los únicos)

Ejemplo:

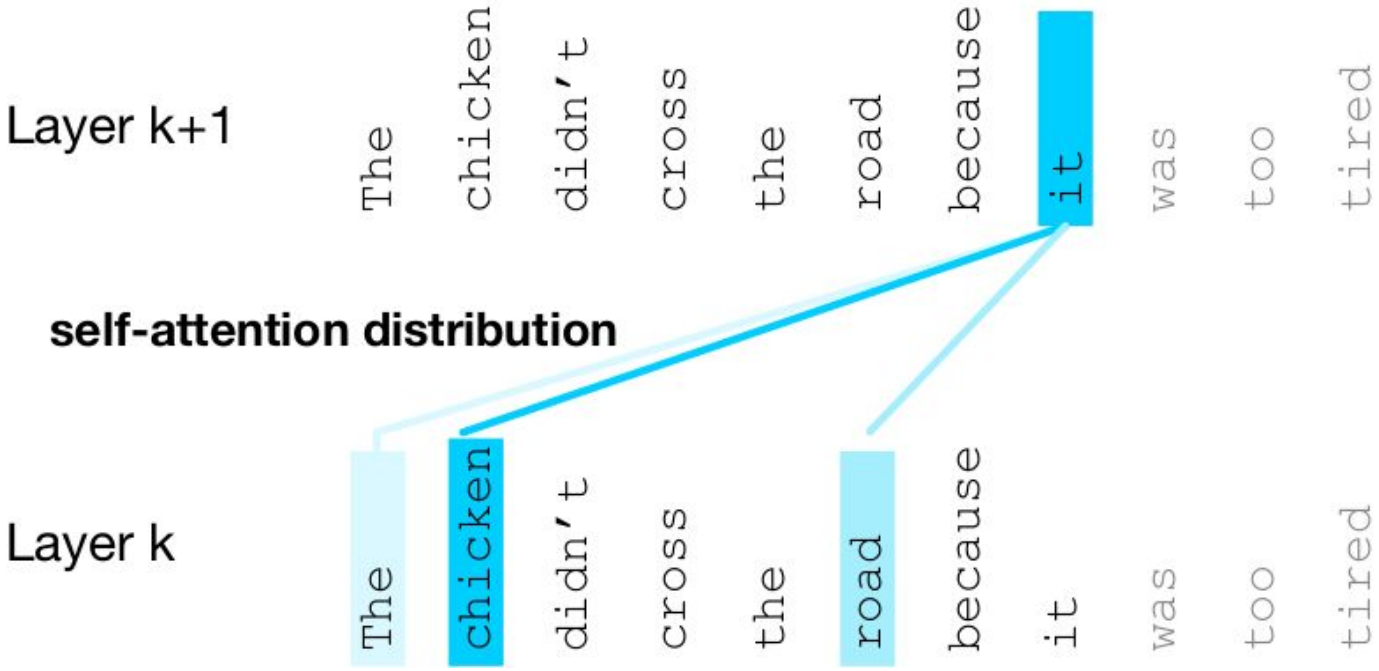
> *The chicken didn't cross the road because **it** was too tired.*
>
> *The chicken didn't cross the road because **it** was too wide.*

En español podría ser:

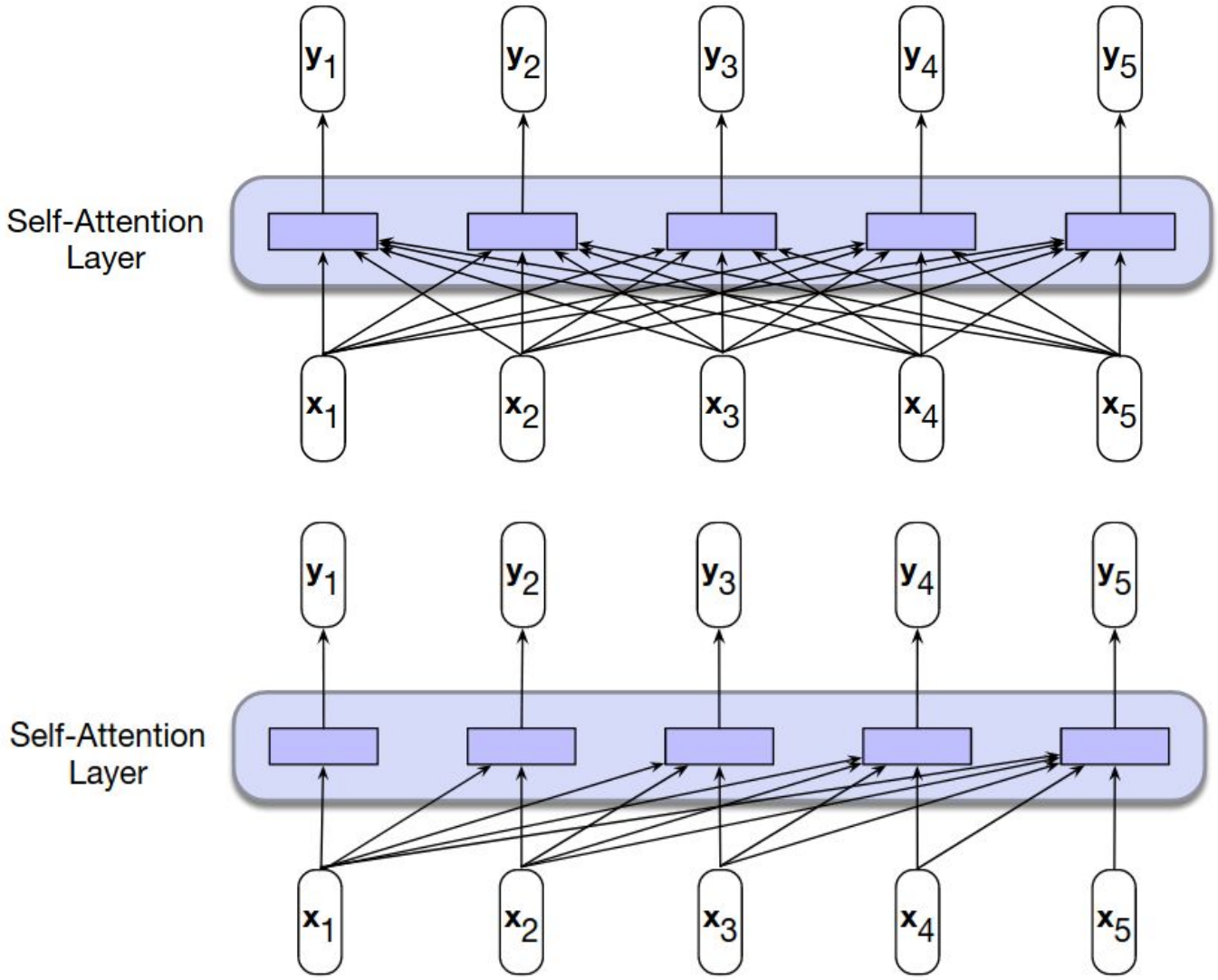
> *El pollo no cruzó la calle porque **estaba** muy cansado*
>
> *El pollo no cruzó la calle porque **estaba** cortada*

Motivación

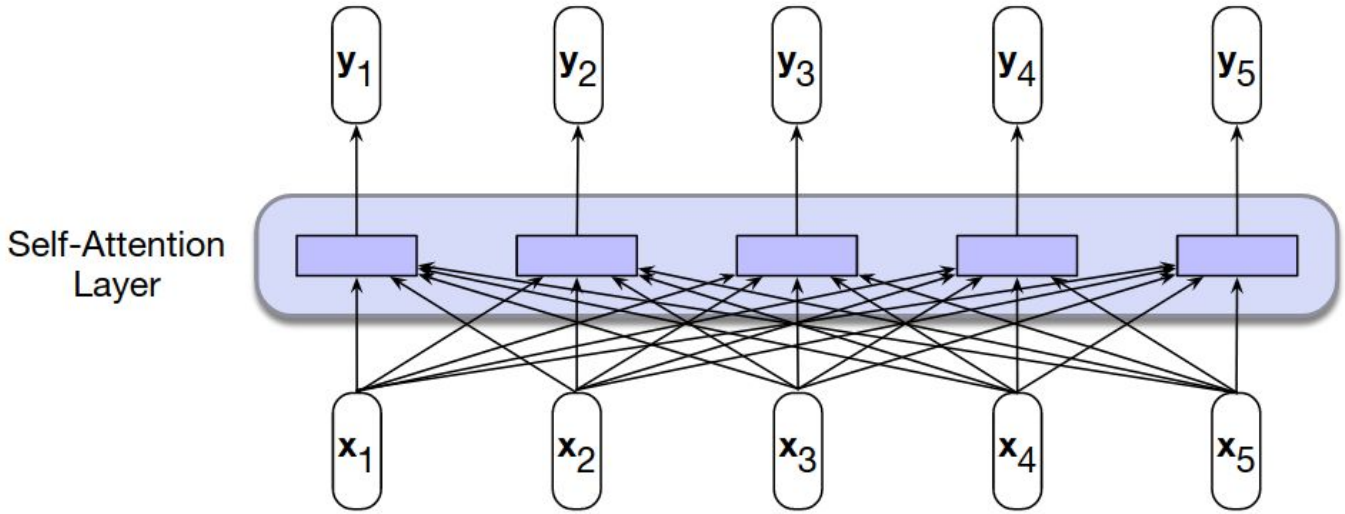
¿Qué esperamos que pase con el mecanismo de atención en estos casos?



Self-attention



Self-attention

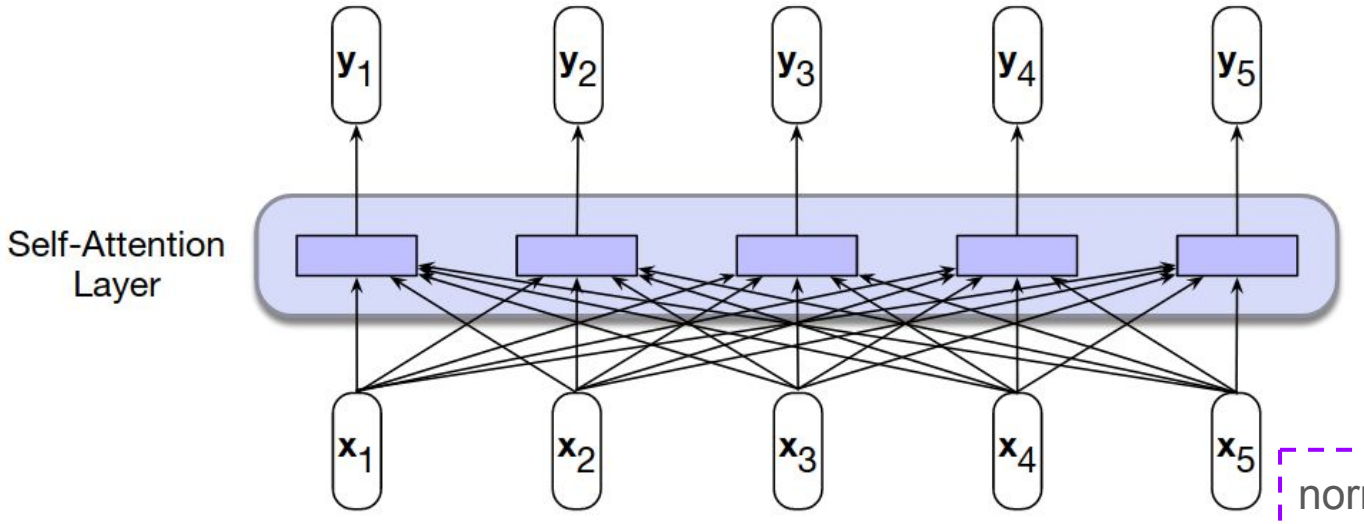


$$\mathbf{y}_i = \sum \alpha_{ij}\mathbf{x}_j$$

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i}\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))}\end{aligned}$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = ?$$

Self-attention



normalización para disminuir la varianza

Tres proyecciones de cada $\mathbf{x}_i$ (**query**, **key** y **value**), con tres matrices

Los vectores $\mathbf{q}_i$ y $\mathbf{k}_j$ computan el **score** de cómo influye la entrada j para la entrada i

Luego softmax (en j) que ponderan los $\mathbf{v}_i$

$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{V}}$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j))$$

$$\mathbf{y}_i = \sum \alpha_{ij}\mathbf{v}_j$$

Self-attention

Ejemplo calculando y_3 a partir de tres tokens x_1, x_2, x_3

d → dimensión de la entrada (x)

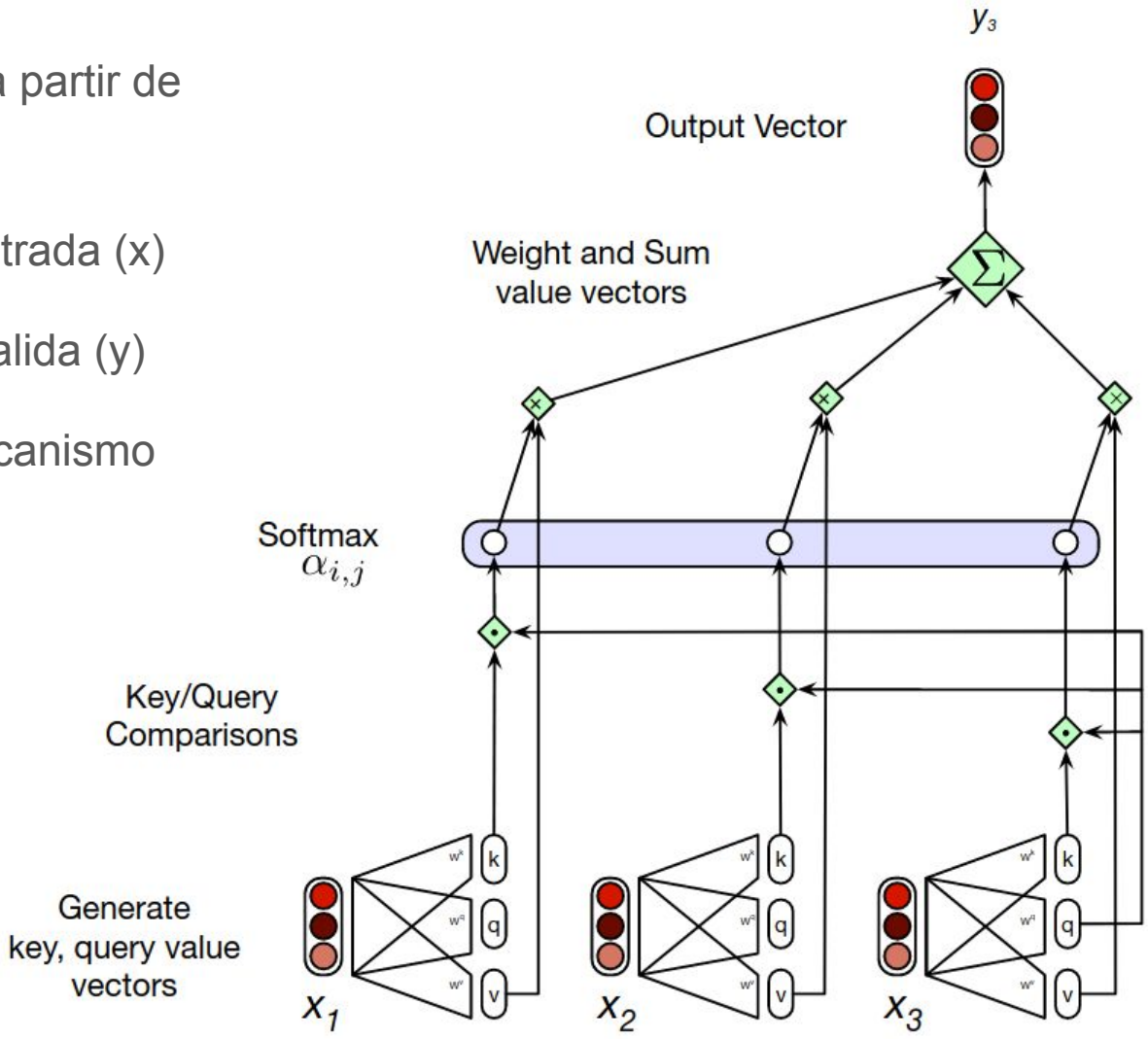
d_v → dimensión de la salida (y)

d_k → dimensión del mecanismo de self-attention

$W^Q \rightarrow d \times d_k$

$W^K \rightarrow d \times d_k$

$W^V \rightarrow d \times d_v$



Todo en una matriz

Supongamos que todos los vectores de la entrada [x_1, x_2, … x_N] (todo el texto) están apilados en una matriz X de tamaño N x d

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

$$\mathit{SelfAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

QK^T → N

q1•k1	q1•k2	q1•k3	q1•k4	q1•k5
q2•k1	q2•k2	q2•k3	q2•k4	q2•k5
q3•k1	q3•k2	q3•k3	q3•k4	q3•k5
q4•k1	q4•k2	q4•k3	q4•k4	q4•k5
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

Multi-head Attention

Por ahora de cada vector de entrada x_i obtenemos un solo vector y_i como salida del esquema atencional (combinado con todos los otros x_j)

Es un solo cabezal de atención

Supongamos que tenemos varias matrices de proyección W^{Kc}, W^{Qc}, W^{Vc}

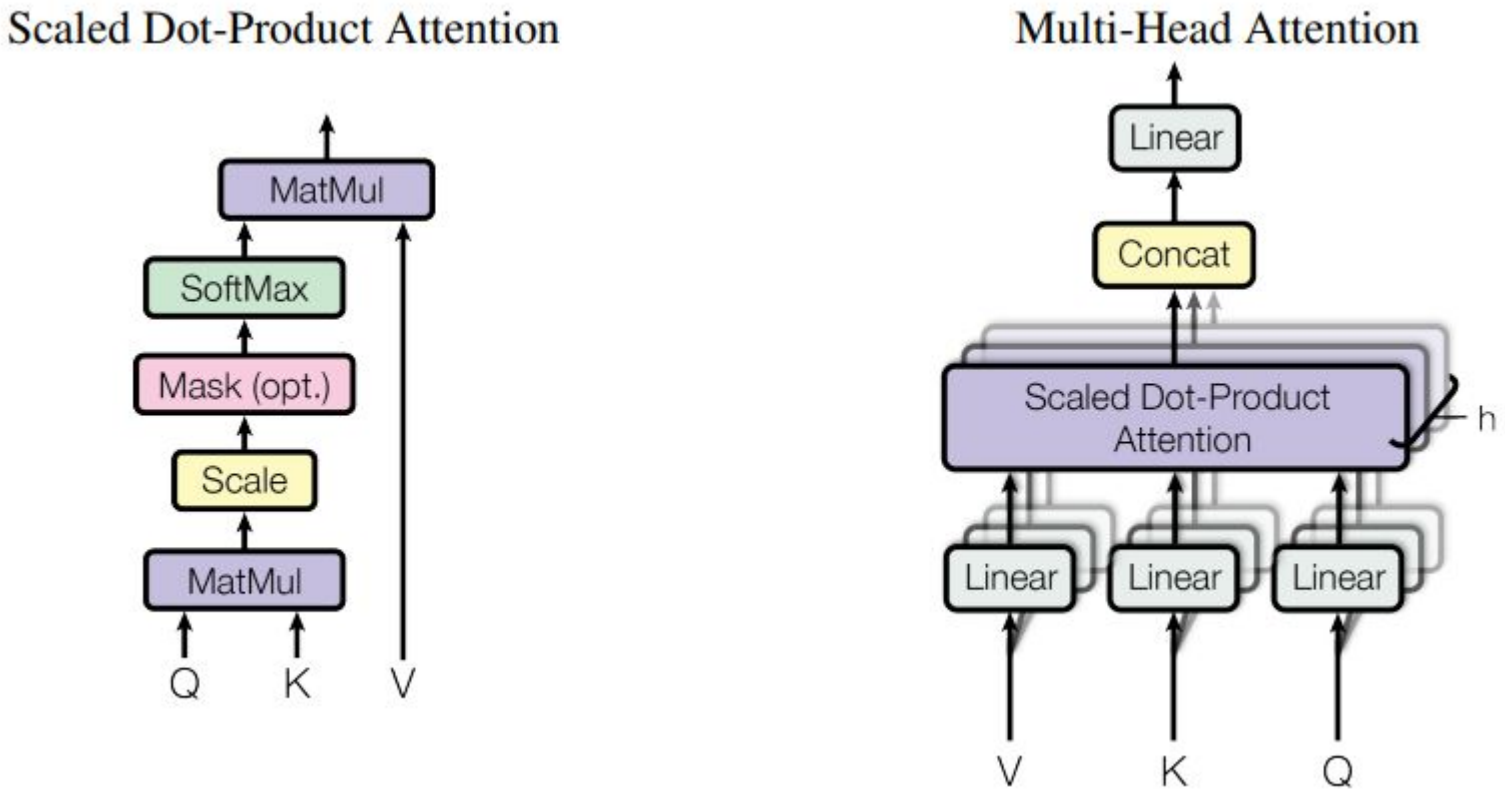
(sean h tripletes de matrices en total)

Por cada conjunto de matrices, obtendremos un y_i^c diferente

¿Para qué se hace esto?

La intuición es que diferentes cabezales de atención pueden aprender cosas distintas: relaciones entre elementos, buscar patrones particulares

Multi-head Attention



Multi-head Attention

Cada matriz W^{Kc} y W^{Qc} tiene tamaño d x d_k

Cada matriz W^{Vc} tiene tamaño d x d_v

Por cada conjunto de matrices, obtendremos un y_i^c diferente, obteniendo en total de h vectores de tamaño d_v

Queremos que y_i sea un solo vector de tamaño d (igual a la entrada)

Nos inventamos una nueva matriz W^O de tamaño h*d_v x d, que toma la concatenación de los y_i^c devolviendo un solo y_i

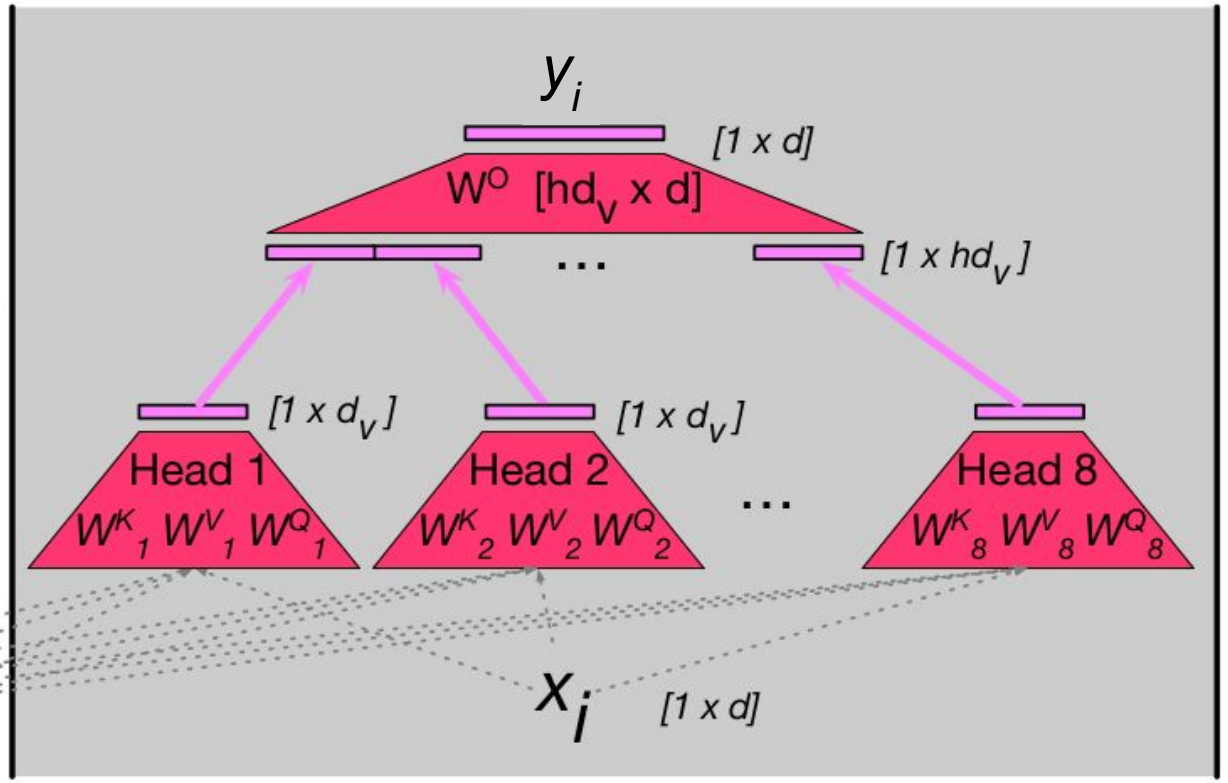
Multi-head Attention

Se proyecta de nuevo a dimensión d

Se concatenan las salidas

Cada cabezal atiende distinto al contexto

…otros x_j…



$$\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_k}, \mathbf{W}_i^K \in \mathbb{R}^{d \times d_k}, \mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$$

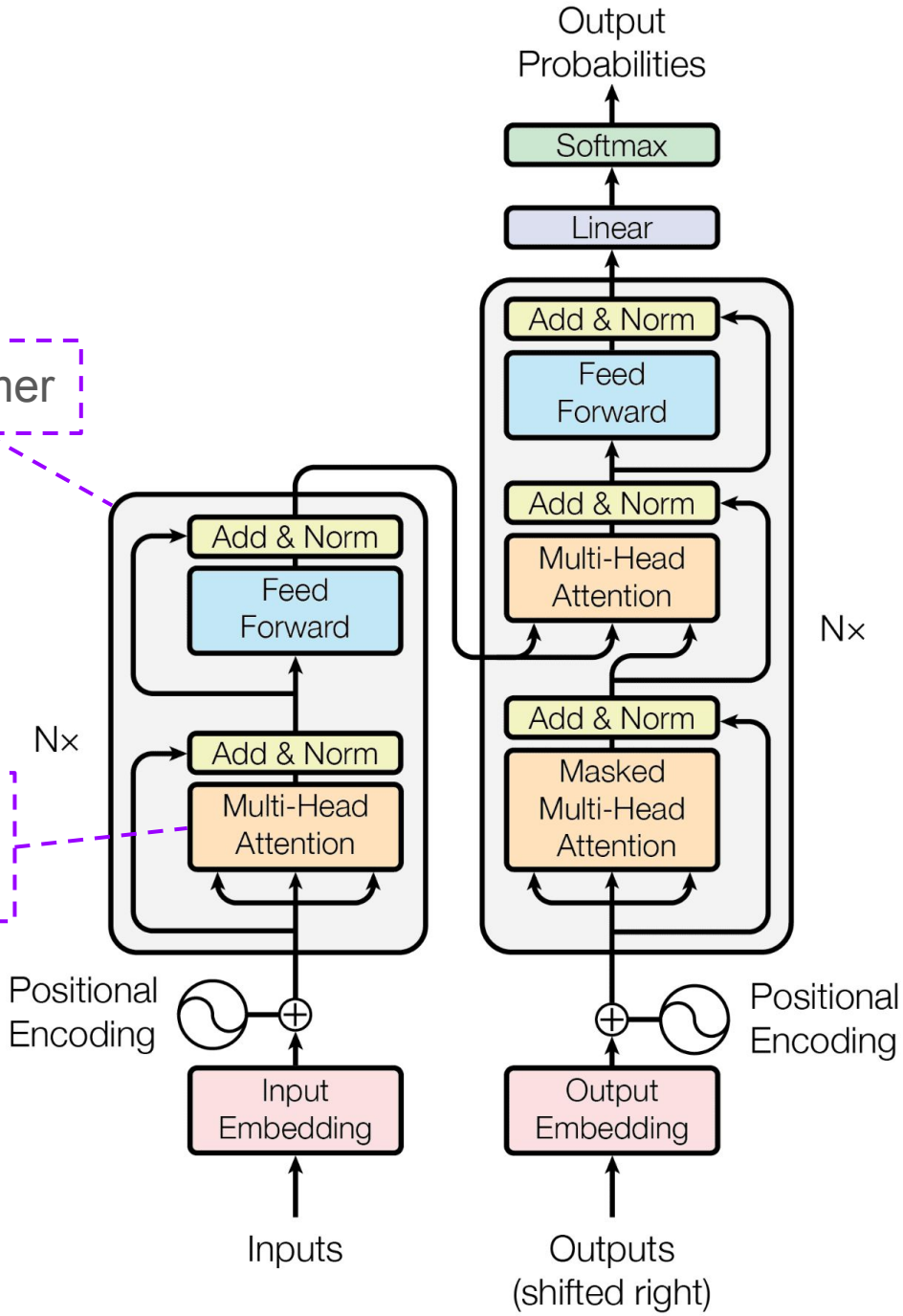
$$MultiHeadAttn(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

$$\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q \; ; \; \mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K \; ; \; \mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V$$

$$\mathbf{head}_i = SelfAttention(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)$$

Bloque Transformer

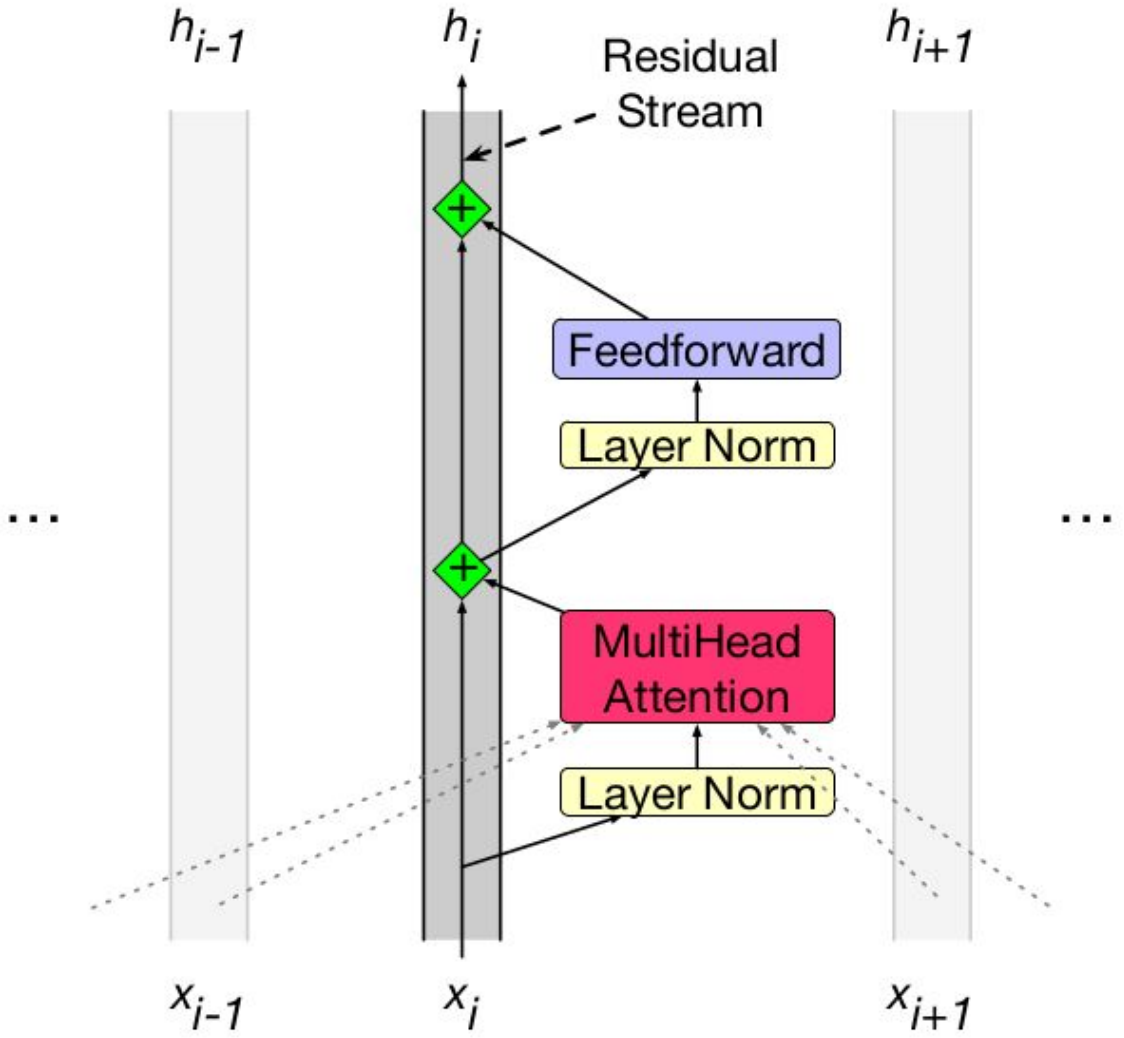
Mecanismo de self-attention



Bloque Transformer

Faltan algunos elementos más del bloque transformer:

- Capa feedforward
- Conexiones residuales
- Capas de normalización



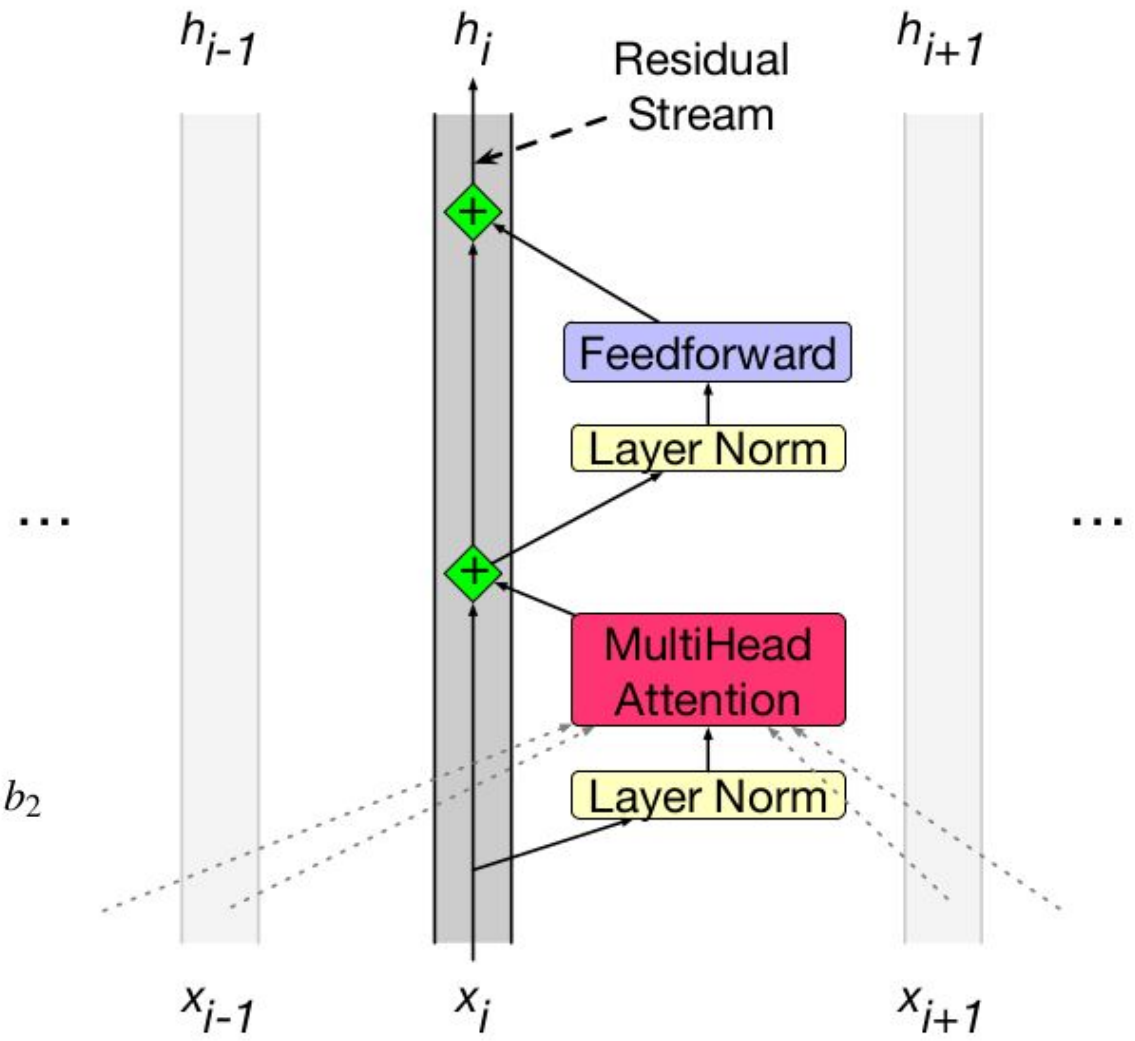
Bloque Transformer

Capa feedforward:

Es como un MLP de dos capas

La capa oculta tiene dimensión d_{ff}, en general mayor que d

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$



Bloque Transformer

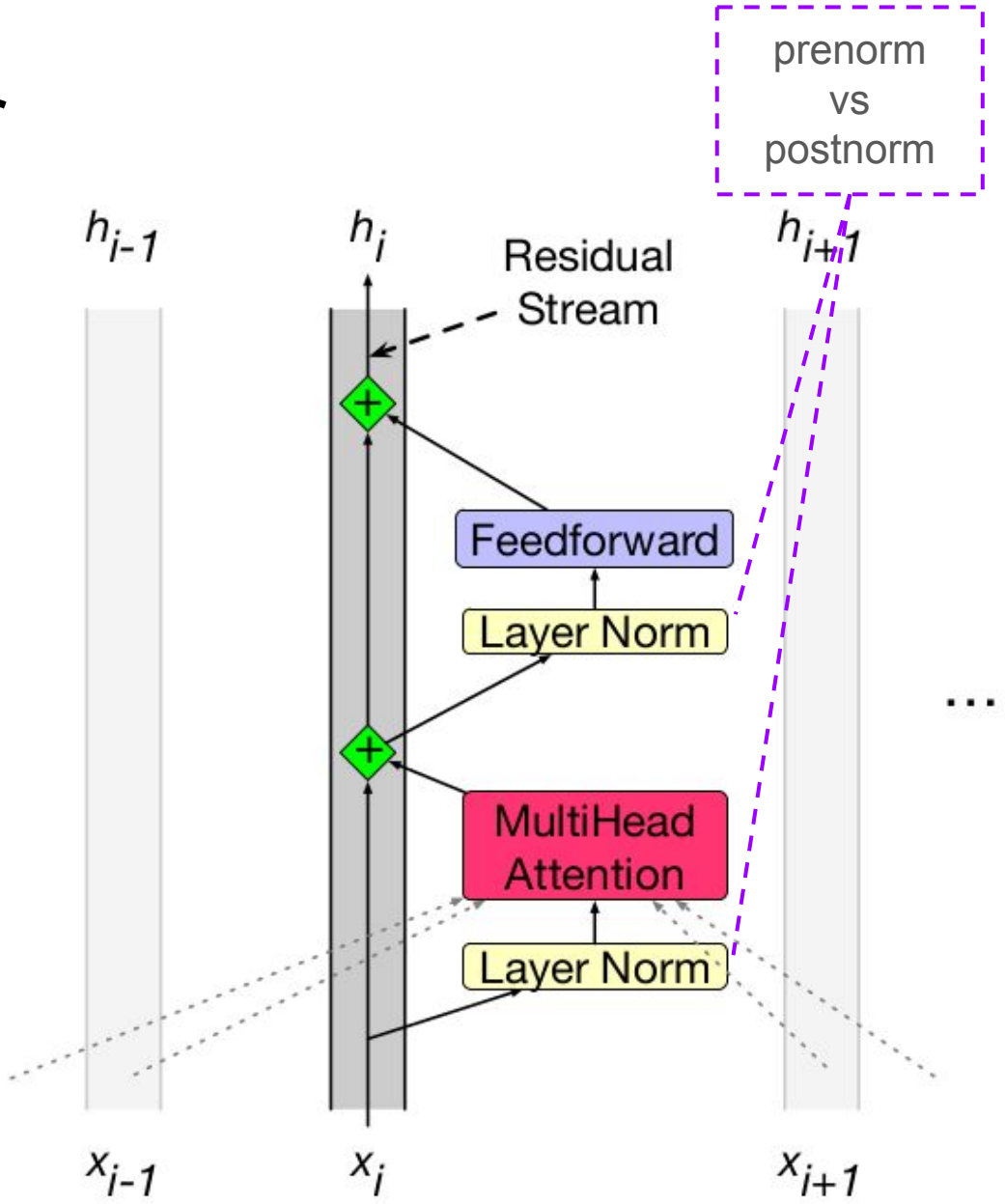
<u>Capas de normalización (Layer Norm):</u>

Ocurre dos veces: antes del mecanismo atencional, y antes de la feedforward (o después!!!)

Realiza una normalización por esperanza y varianza de las componentes del vector

Agrega dos parámetros (γ, β) ajustables durante el entrenamiento

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$



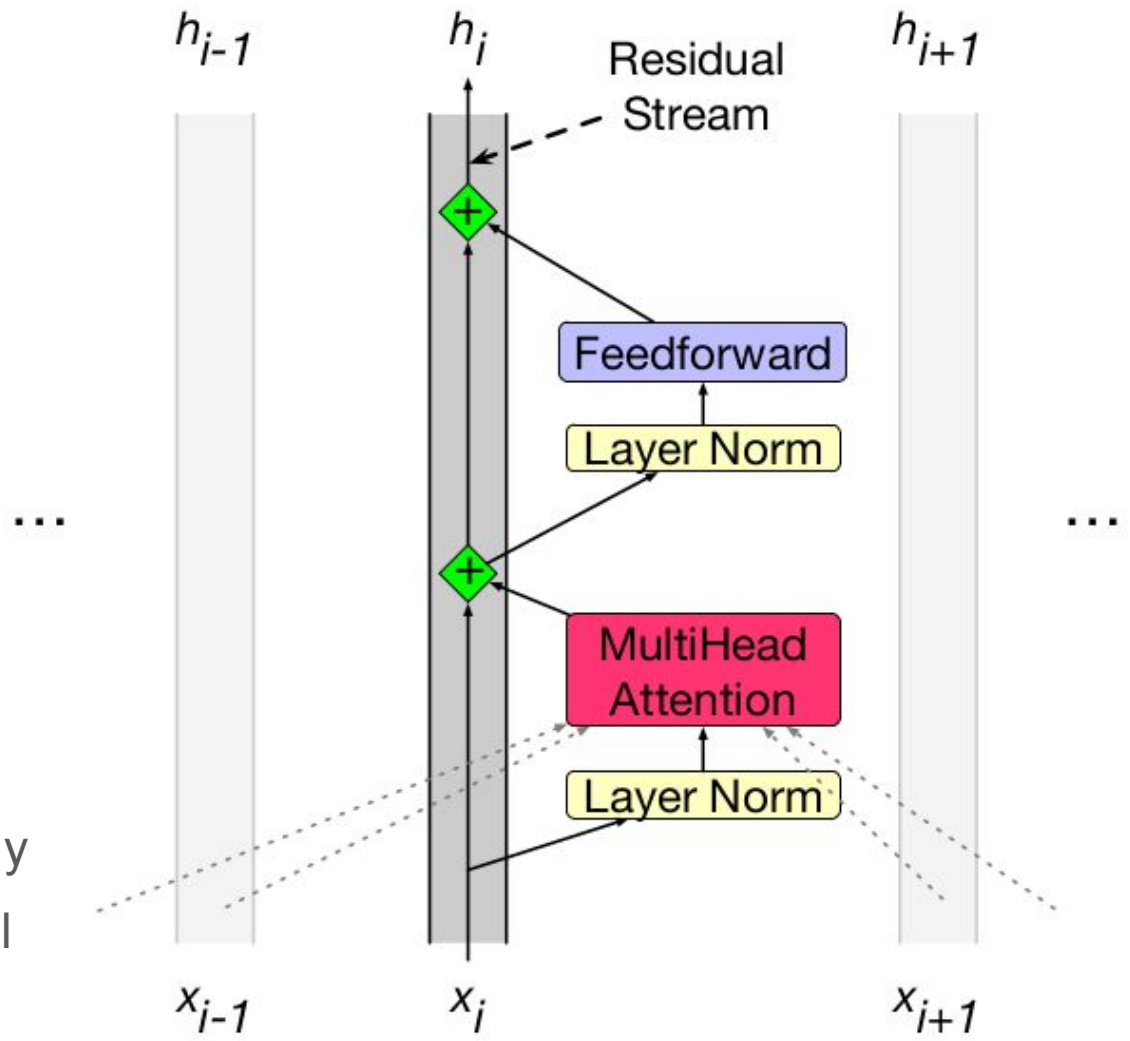
Bloque Transformer

Conexión residual:

Las distintas componentes no se van aplicando cada una al resultado de la otra

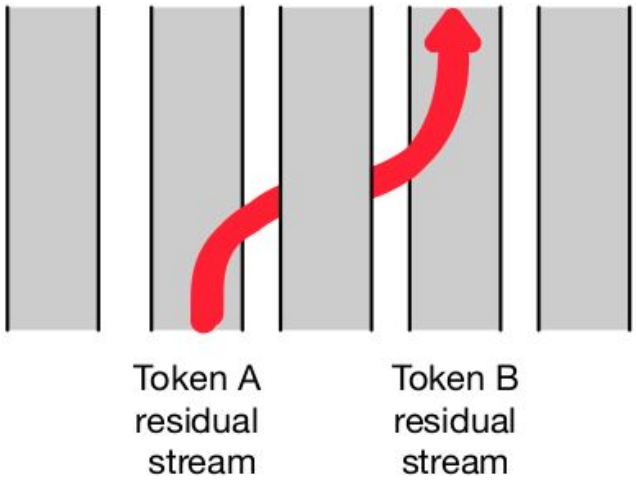
Es como que hay un “canal” (*residual stream*) con el valor del embedding

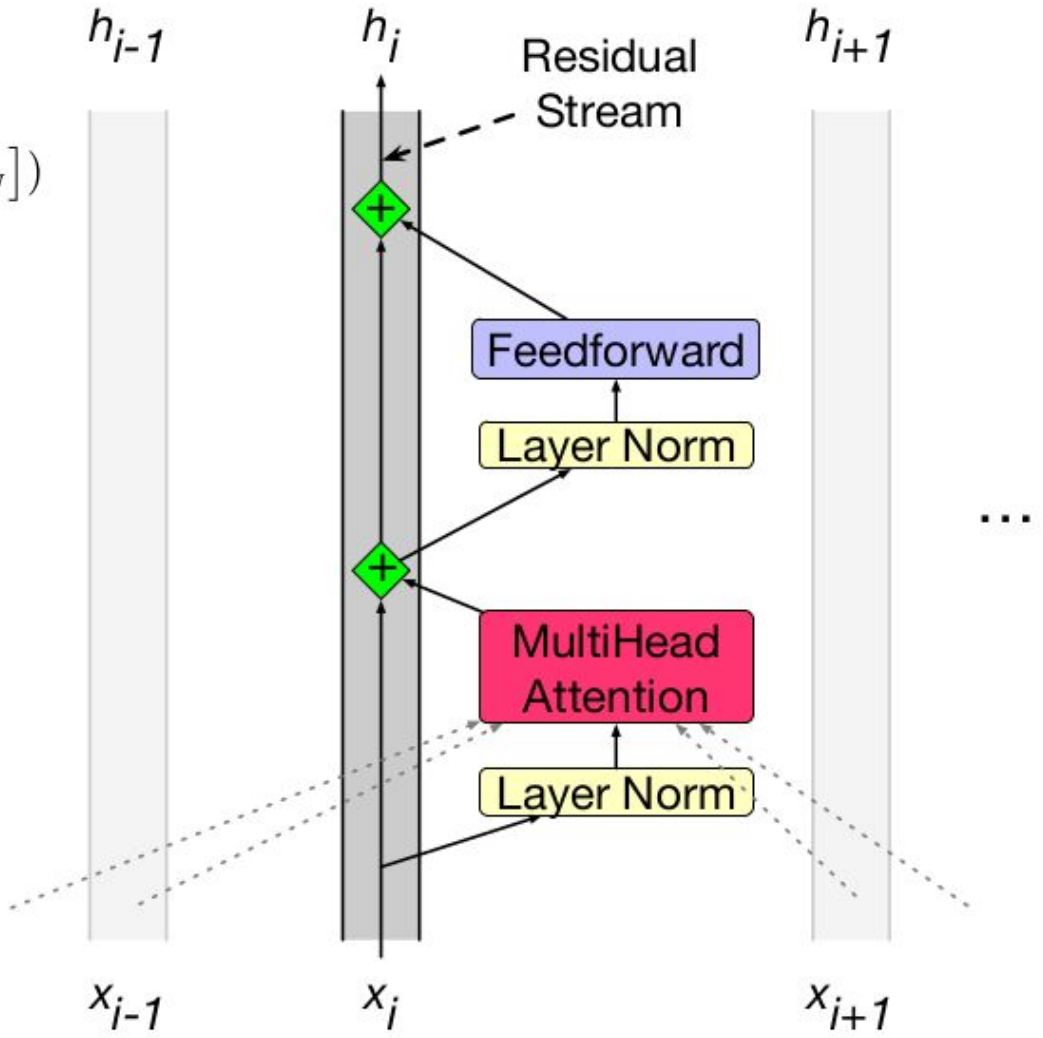
En dos puntos se procesa la información que trae el canal, y se la vuelve a agregar al canal



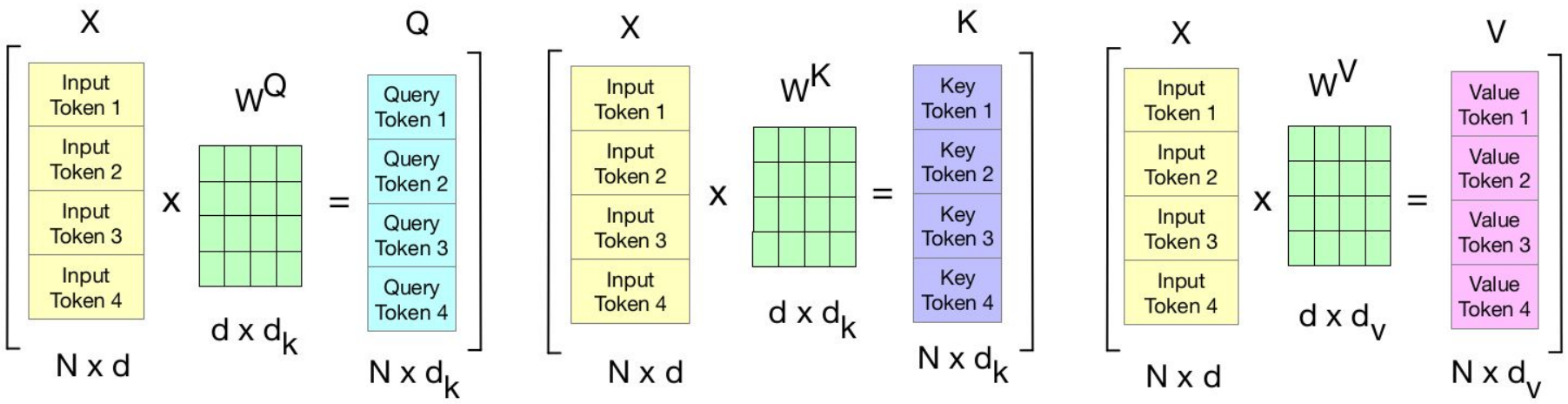
Bloque Transformer

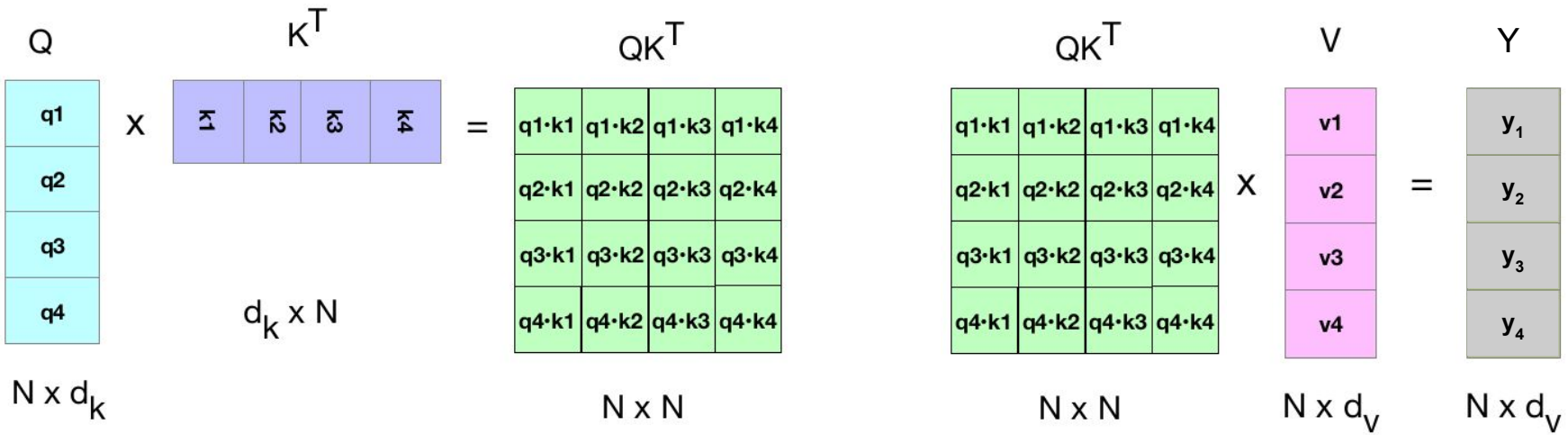
$$
\begin{aligned}
\mathbf{t}_i^1 &= \text{LayerNorm}(\mathbf{x}_i) \\
\mathbf{t}_i^2 &= \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \cdots, \mathbf{x}_N^1]) \\
\mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
\mathbf{t}_i^4 &= \text{LayerNorm}(\mathbf{t}_i^3) \\
\mathbf{t}_i^5 &= \text{FFN}(\mathbf{t}_i^4) \\
\mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3
\end{aligned}
$$





Cálculo en paralelo





Cálculo en paralelo

$$
\begin{aligned}
\mathbf{T}^1 &= \text{MultiHeadAttention}(\mathbf{X}) \\
\mathbf{T}^2 &= \mathbf{X} + \mathbf{T}^1 \\
\mathbf{T}^3 &= \text{LayerNorm}(\mathbf{T}^2) \\
\mathbf{T}^4 &= \text{FFN}(\mathbf{T}^3) \\
\mathbf{T}^5 &= \mathbf{T}^4 + \mathbf{T}^3 \\
\mathbf{H} &= \text{LayerNorm}(\mathbf{T}^5)
\end{aligned}
$$

$$
\mathbf{Q}^i = \mathbf{X}\mathbf{W}^{Qi}; \quad \mathbf{K}^i = \mathbf{X}\mathbf{W}^{Ki}; \quad \mathbf{V}^i = \mathbf{X}\mathbf{W}^{Vi}
$$

$$
\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}^i, \mathbf{K}^i, \mathbf{V}^i) = \text{softmax}\left(\frac{\mathbf{Q}^i\mathbf{K}^{iT}}{\sqrt{d_k}}\right)\mathbf{V}^i
$$

$$
\text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O
$$

esta es la versión postnorm

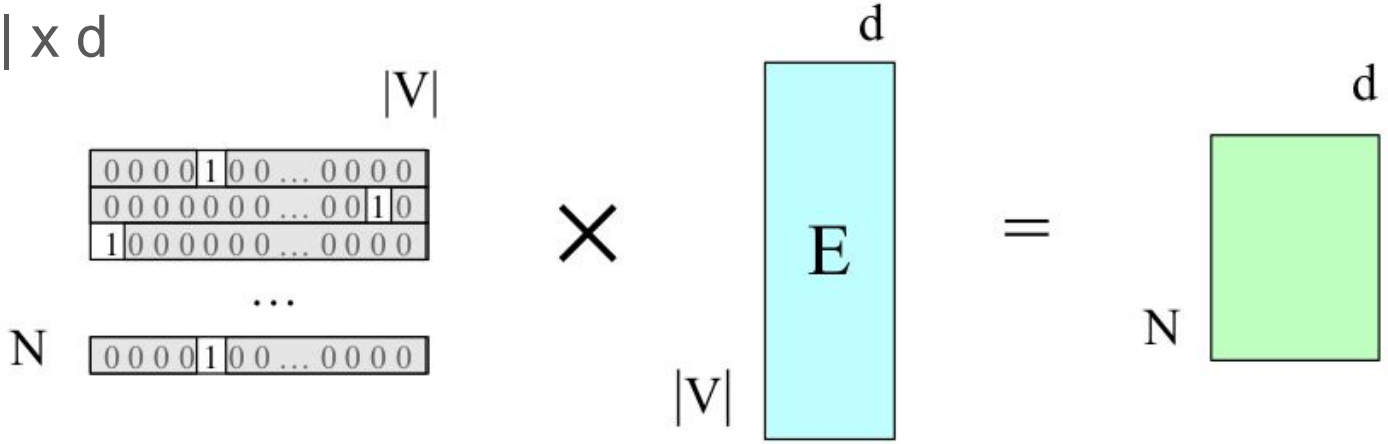
Codificación de la entrada

La representación del texto de entrada es mediante tokens

- BPE, word-piece, otros…

Al entrar a la red, esos tokens pasan por una capa de embeddings

E → |V| x d

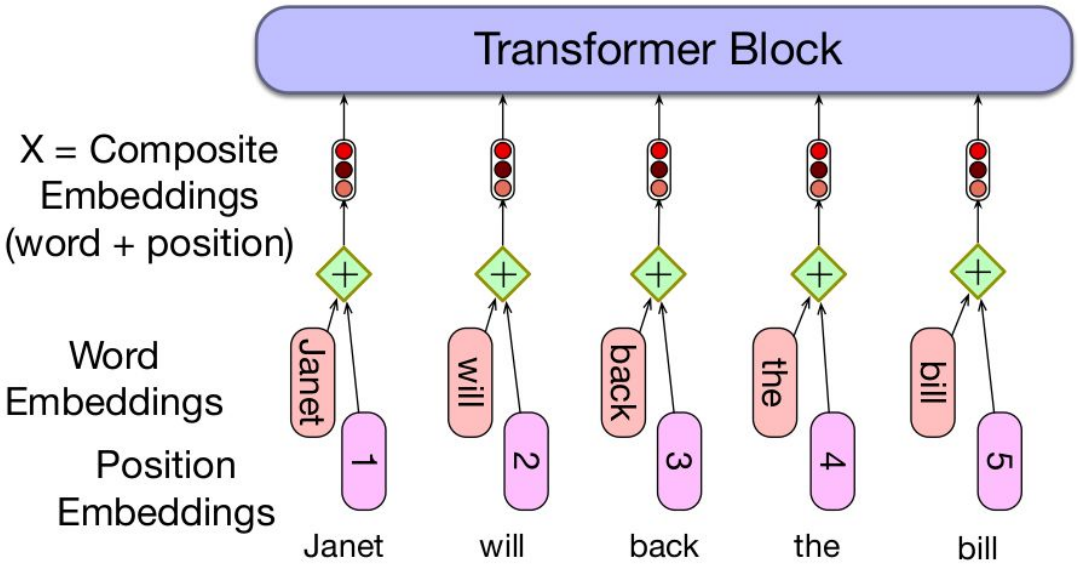


Pero tenemos un problema extra… ¿qué pasa con el orden?

Codificación de la entrada

Hasta este punto no hay noción de orden en la entrada

Ni hay una recurrencia que “imponga” orden procesando de a uno



Solución para los Transformers:

“Aumentar” cada x_i para que incluya una representación de su posición

Embeddings posicionales (position embeddings)

Codificación de la entrada

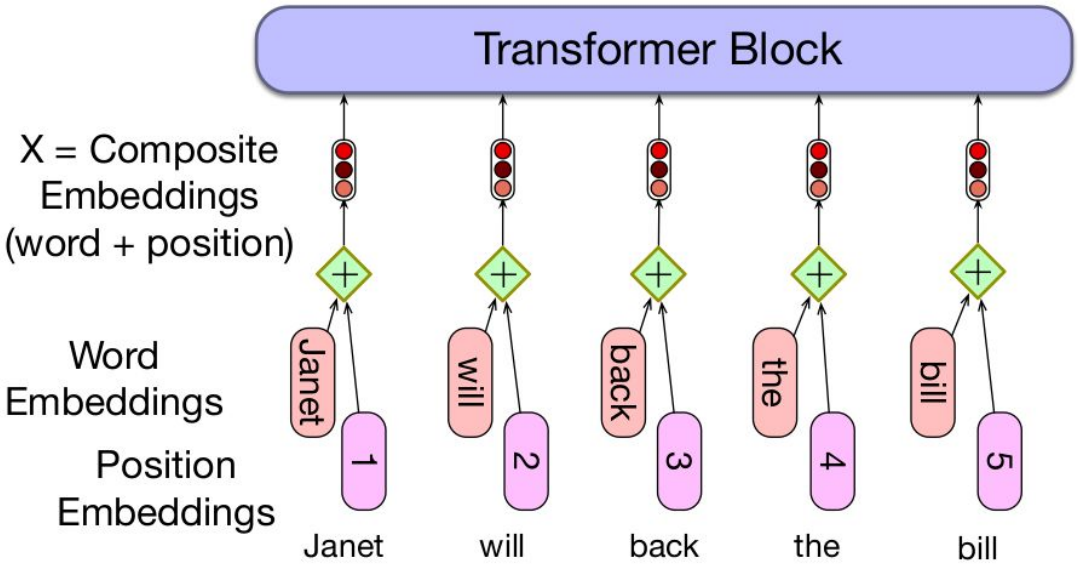
Embeddings posicionales:

Pueden ser entrenados, nos definimos un conjunto de embeddings hasta cierto largo máximo

Las primeras posiciones van a ser más comunes y las últimas quedarán “menos” entrenadas

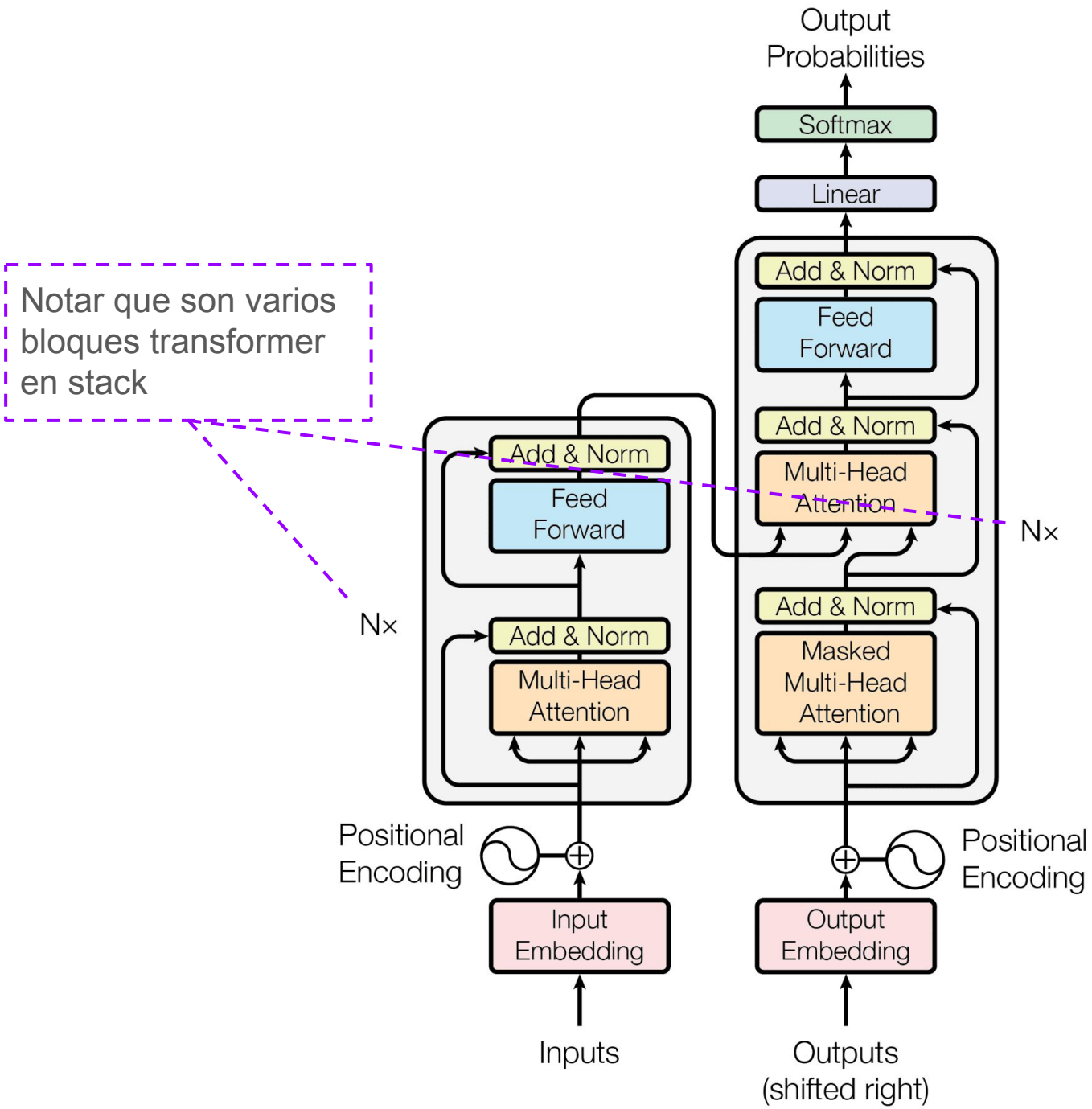
Alternativa: embeddings estáticos

El transformer original usa una combinación de senos y cosenos



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$


Notar que son varios bloques transformer en stack
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Formas de uso

El transformer original fue planteado como arquitectura encoder-decoder

Primer ejemplo de uso: traducción automática

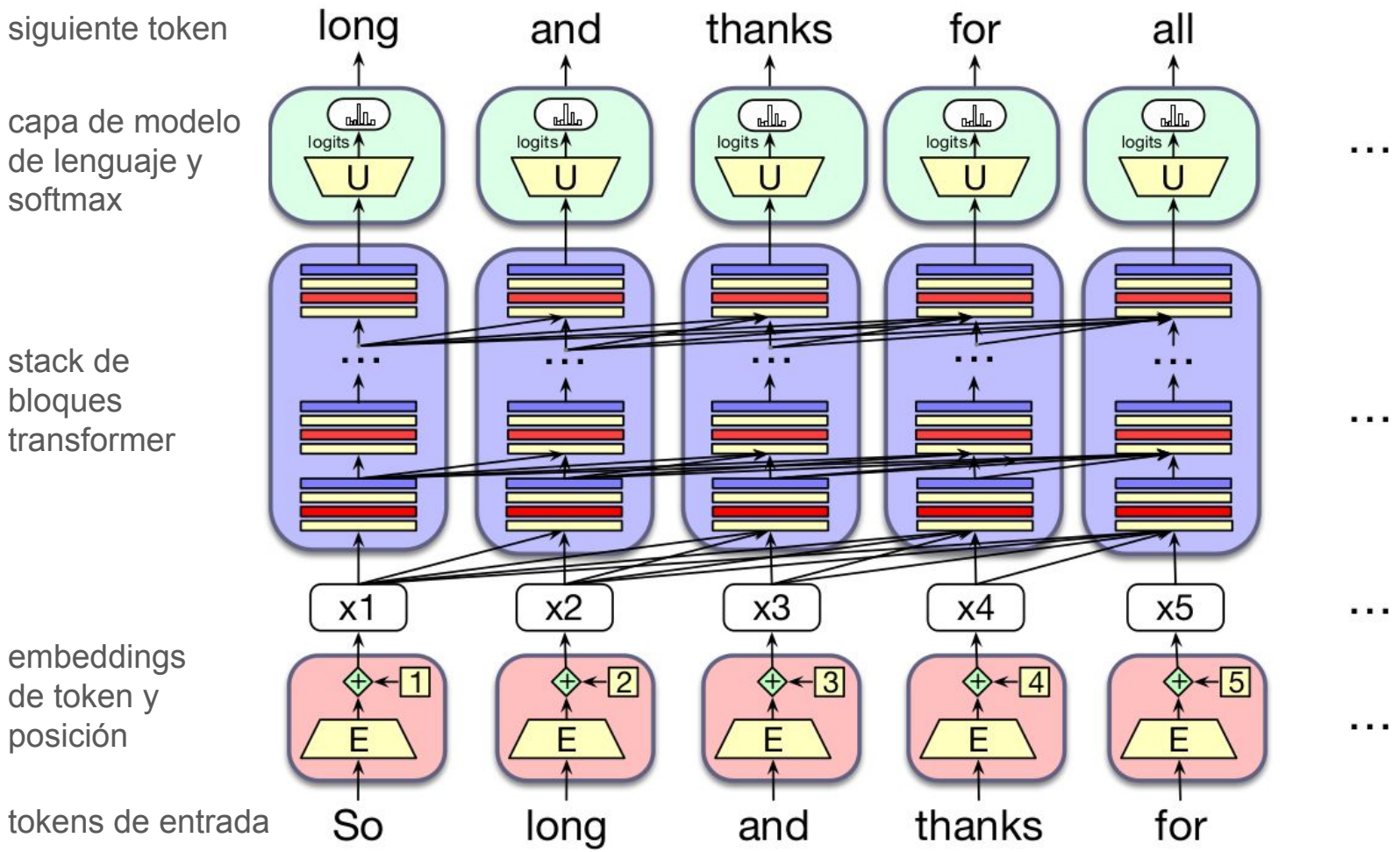
Pero luego aparecen otras variantes

Encoder-only (por ejemplo BERT)

Decoder-only (por ejemplo GPT)

Transformer: Modelo de Lenguaje

Se usa el transformer en modo “causal”

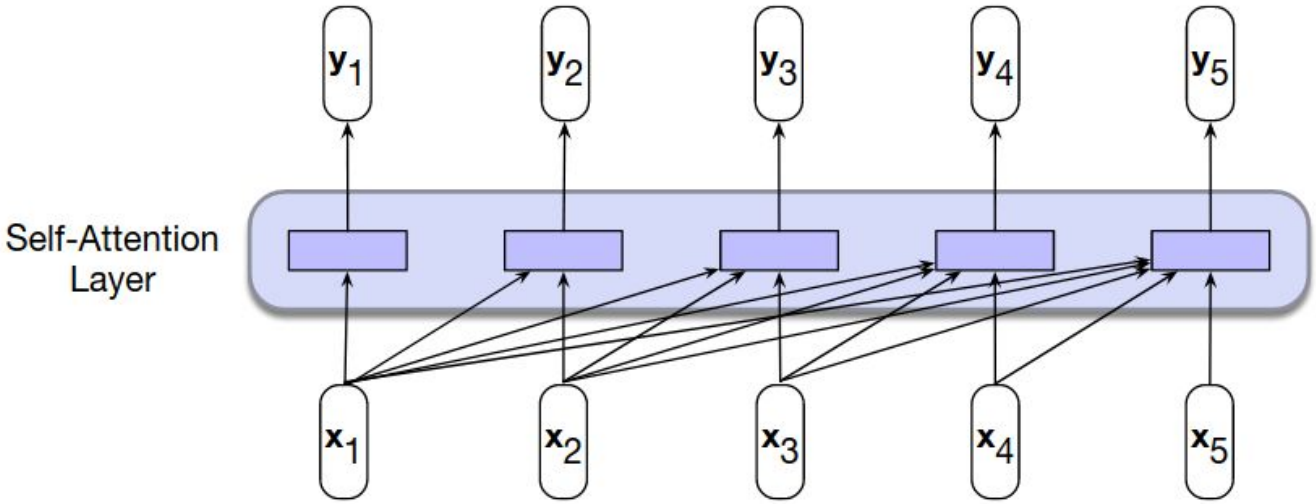


Transformer: Modelo de Lenguaje

En estos casos estamos usando el transformer en modo decoder

Va generando las palabras de a una

Por lo tanto, al procesar una palabra, no podemos tener acceso a las palabras siguientes, solo a las anteriores



Máscara causal de Self-attention

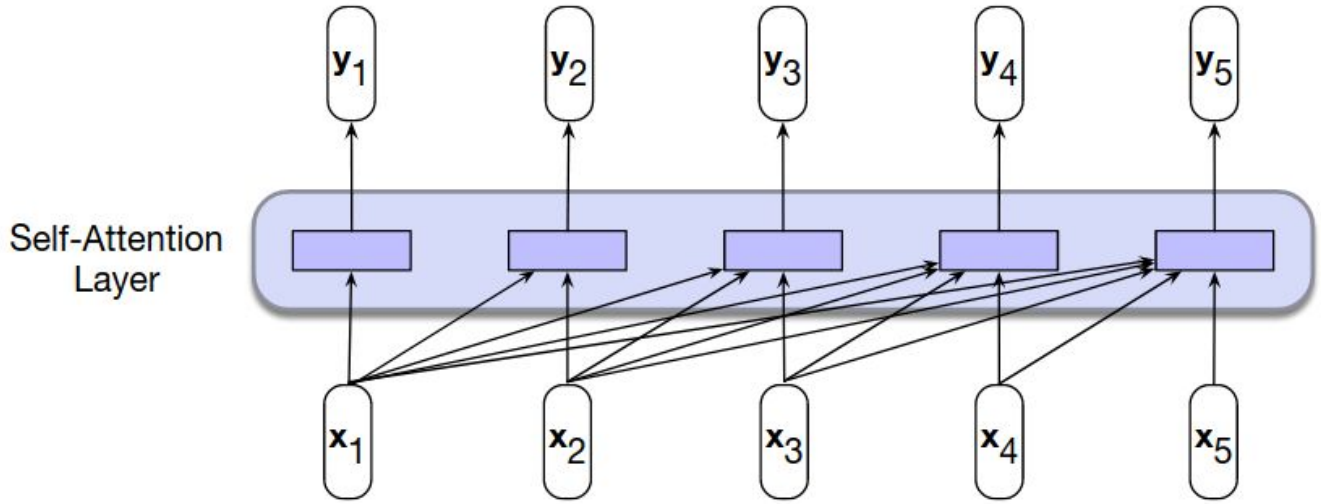
El mecanismo es igual, pero agrega un enmascaramiento para no ver palabras futuras

Similar a sumar una matriz con -∞ en el triángulo superior derecho y 0 en el inferior izquierdo

De esta forma, al hacer el softmax todos los -∞ se traducen en palabras con probabilidad 0

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

$$\mathit{SelfAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\mathrm{mask}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

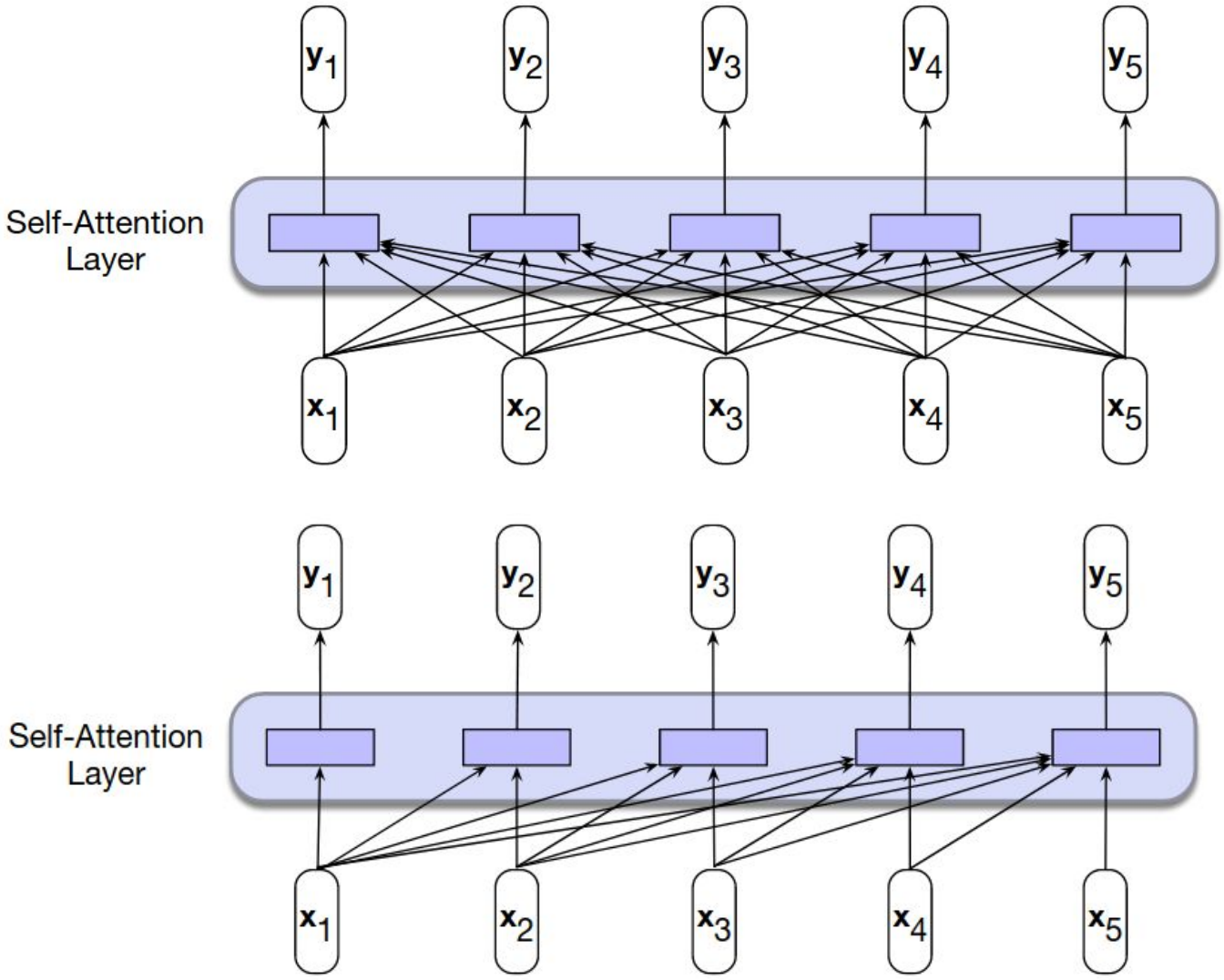


N

q1·k1	−∞	−∞	−∞
q2·k1	q2·k2	−∞	−∞
q3·k1	q3·k2	q3·k3	−∞
q4·k1	q4·k2	q4·k3	q4·k4

N

Bidireccional vs. Causal



N

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4

N

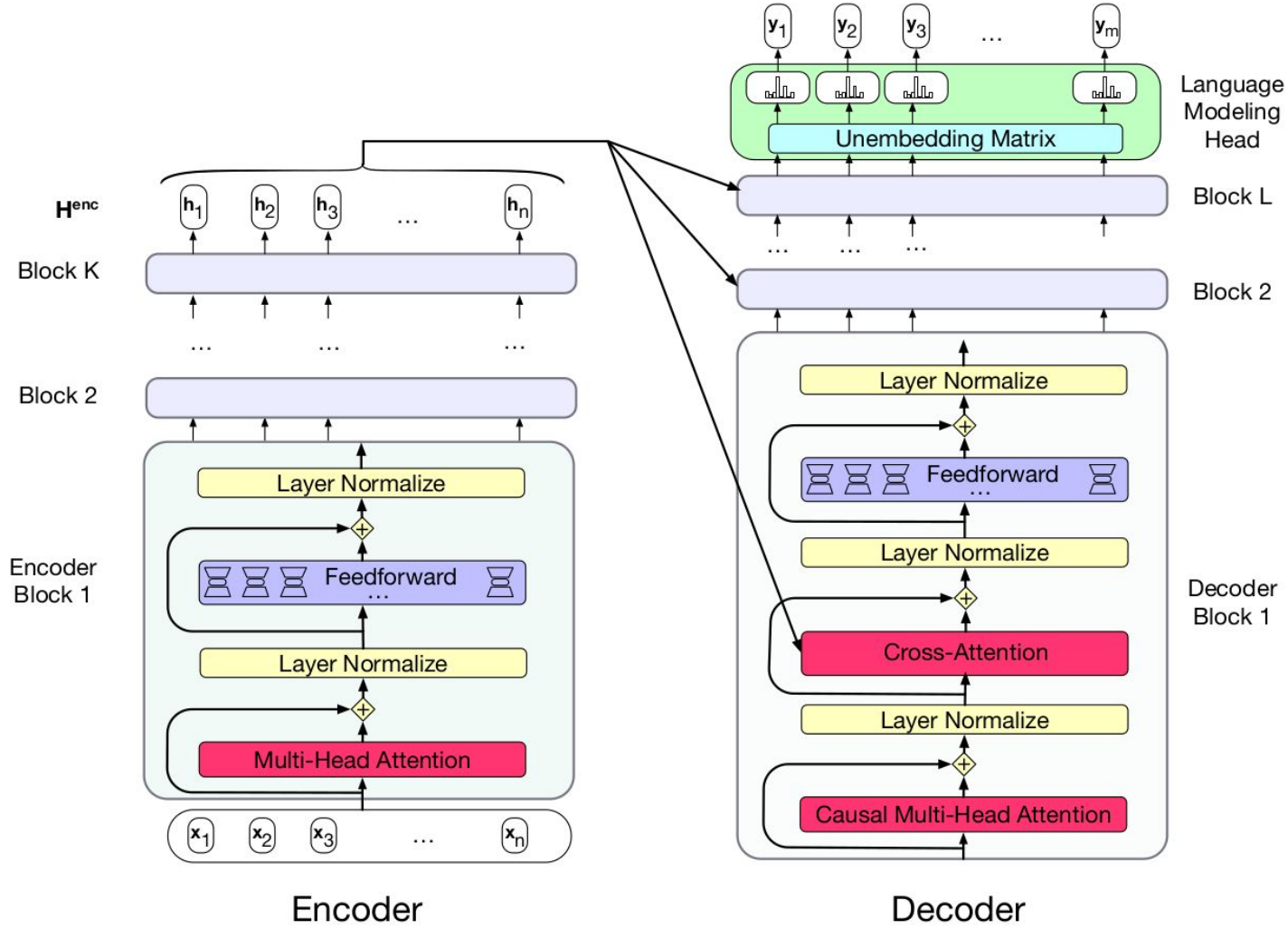
N

q1·k1	−∞	−∞	−∞
q2·k1	q2·k2	−∞	−∞
q3·k1	q3·k2	q3·k3	−∞
q4·k1	q4·k2	q4·k3	q4·k4

N

Transformer: Traducción Automática

Se usa el modelo completo en modo encoder-decoder



Transformer Encoder-Decoder

El encoder crea una representación para cada token

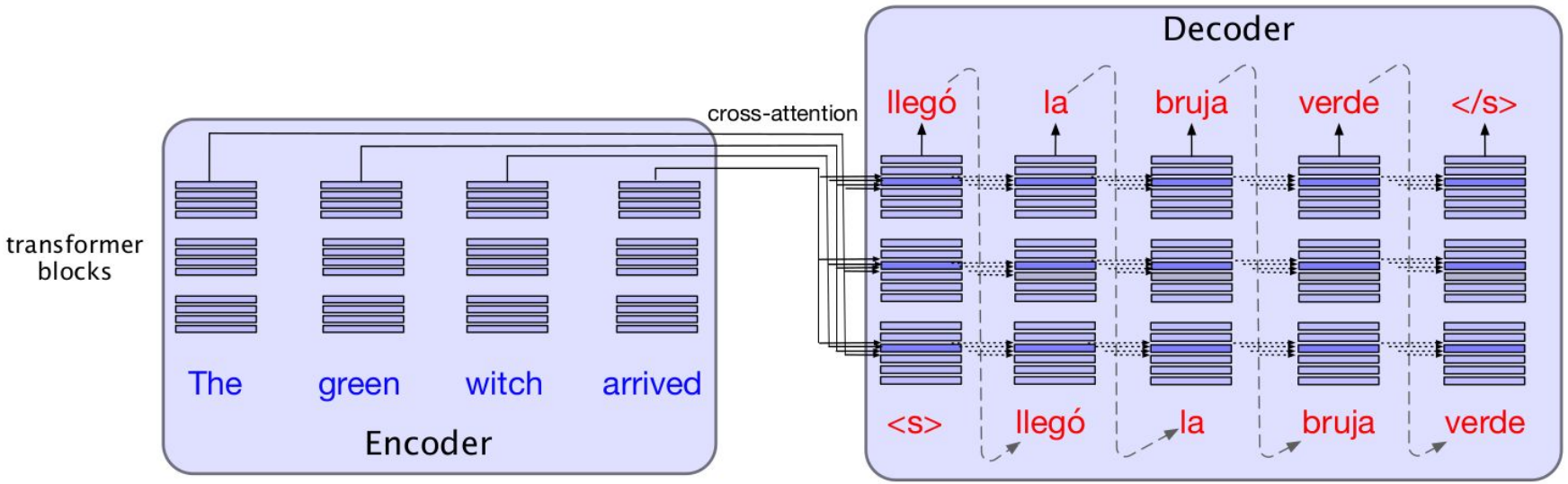
Al decoder lo inicializamos con un token <S>

Usa masked-attention para codificarlo

Y cross-attention para combinar ese token con el resultado del encoder

Lo cual produce como resultado el primer token de salida…

…y repetimos el proceso para decodificar el resto



Transformer Encoder-Decoder

Cross-attention es similar a multi-head attention

Pero las Q se arman con el paso anterior del decoder

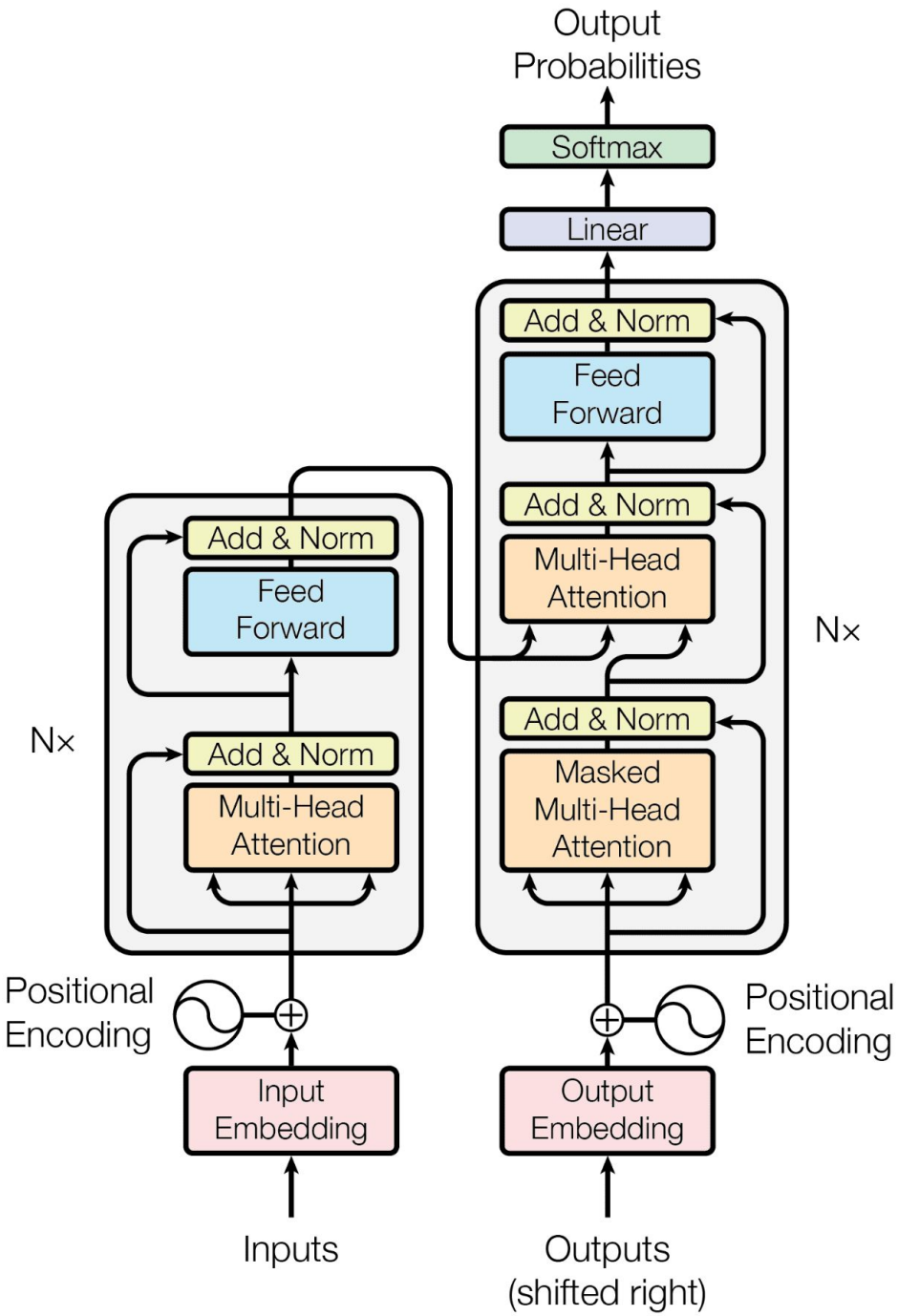
Las K y V se arman con el resultado de la última capa del encoder

$$\mathbf{Q} = \mathbf{H}^{dec[\ell-1]}\mathbf{W}^{\mathbf{Q}};\ \mathbf{K} = \mathbf{H}^{enc}\mathbf{W}^{\mathbf{K}};\ \mathbf{V} = \mathbf{H}^{enc}\mathbf{W}^{\mathbf{V}}$$

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

Traducción Automática (del paper original)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Bibliografía

- Jurafsky and Martin 3rd edition. Capítulos 9 y 13. https://web.stanford.edu/~jurafsky/slp3/

- Machine Learning with PyTorch and Scikit-Learn. Raschka et al. 2022. Cap 16.

- “Attention is all you need” Vaswani et al. 2017 https://arxiv.org/pdf/1706.03762.pdf