



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Aprendizaje Automático

Clase[-1]:
Intro a Modelos de Lenguaje



INTRO A MODELOS DE LENGUAJE

MODELOS DE LENGUAJE

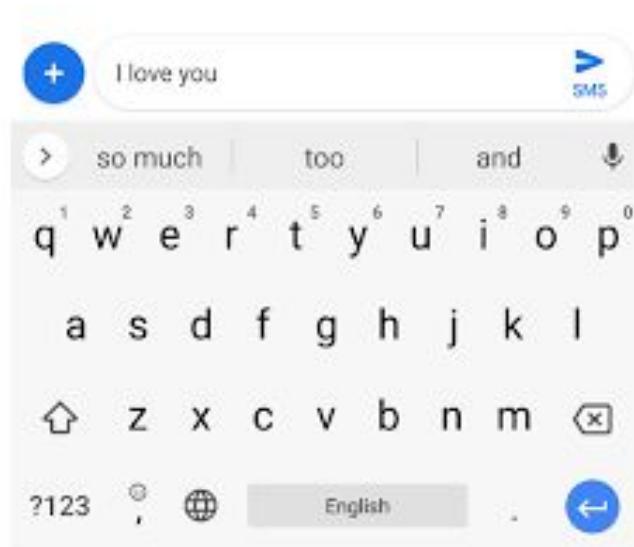


Un modelo de lenguaje **predice** la probabilidad de una secuencia de palabras.

Cuando llegue a casa me tengo que acordar de darle de comer al

- gato
- perro
- silla

MODELOS DE LENGUAJE



MODELOS DE LENGUAJE

Google



MODELOS DE LENGUAJE

❖ Model: GPT-4

LU

Quien gano el mundial de futbol en 1974?



La Copa Mundial de la FIFA de 1974 fue ganada por Alemania Occidental. Derrotaron a los Países Bajos 2-1 en la final.



MODELOS DE LENGUAJE UNIDIRECCIONALES



¿Cómo calcular la probabilidad de una secuencia de palabras?

$$p(w_1 w_2 w_3 \dots w_n)$$

MODELOS DE LENGUAJE UNIDIRECCIONALES



¿Cómo calcular la probabilidad de una secuencia de palabras?

$$p(w_1 w_2 w_3 \dots w_n) = p(w_1) * p(w_2 | w_1) * p(w_3 | w_1 w_2) * p(w_4 | w_1 w_2 w_3) * \dots * p(w_n | w_1 w_2 w_3 \dots w_{n-1})$$

MODELOS DE LENGUAJE UNIDIRECCIONALES



¿Cómo calcular la probabilidad de una secuencia de palabras?

$$p(w_1 w_2 w_3 \dots w_n) = p(w_1) * p(w_2 | w_1) * p(w_3 | w_1 w_2) * p(w_4 | w_1 w_2 w_3) * \dots * p(w_n | w_1 w_2 w_3 \dots w_{n-1})$$

$$p(x_1^n) = \prod_{k=1}^N p(w|w_1^{n-1})$$

MODELOS DE LENGUAJE UNIDIRECCIONALES



¿Cómo calcular la probabilidad de una secuencia de palabras?

$$p(w_1 w_2 w_3 \dots w_n) = p(w_1) * p(w_2 | w_1) * p(w_3 | w_1 w_2) * p(w_4 | w_1 w_2 w_3) * \dots * p(w_n | w_1 w_2 w_3 \dots w_{n-1})$$

$$p(x_1^n) = \prod_{k=1}^N p(w|w_1^{n-1})$$

$$\begin{aligned} p(\text{"Cuando llegue a casa me tengo que acordar de darle de comer al gato"}) &= \\ p(\text{cuando}) * p(\text{llegue} | \text{Cuando}) * p(\text{a} | \text{Cuando llegue}) * \dots * p(\text{gato} | \text{Cuando llegue ... al}) \end{aligned}$$

MODELOS DE LENGUAJE

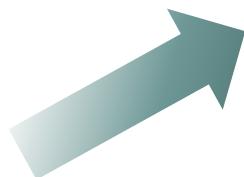


Voy a guardar el queso en el

- refrigerador [0.5]
- freezer [0.3]
- heladera [0.1]
- horno [0.05]
- termotanque [0.0001]
- ...
- Callao [0.000000001]

MODELOS DE LENGUAJE

$$f(x) = y$$



Voy a guardar el queso en el



refrigerador

MODELOS DE LENGUAJE GENERATIVOS

Voy a guardar el queso en el



Un modelo de Lenguaje generativo produce una secuencia de palabras basado en patrones aprendidos en grandes cantidades de datos.

MODELOS DE LENGUAJE GENERATIVOS



Voy a guardar el queso en el refrigerador

MODELOS DE LENGUAJE GENERATIVOS



Voy a guardar el queso en el refrigerador para

MODELOS DE LENGUAJE GENERATIVOS



Voy a guardar el queso en el refrigerador para conservarlo

MODELOS DE LENGUAJE GENERATIVOS



Voy a guardar el queso en el refrigerador para conservarlo fresco.

MODELOS DE LENGUAJE GENERATIVOS



Voy a guardar el queso en el refrigerador para conservarlo fresco.

MODELOS DE LENGUAJE GENERATIVOS



Voy a guardar el queso en el refrigerador para conservarlo fresco. <end>

MODELOS DE LENGUAJE GENERATIVOS



Voy a guardar el queso en el refrigerador para conservarlo fresco. <end>

MODELOS DE LENGUAJE BIDIRECCIONALES



No hay una formulación única pero ...

$$p(w_t \mid w_1, \dots, w_{t-1}, w_{t+1}, \dots, w_n)$$

$$p_{\rightarrow}(w_1, w_2, \dots, w_n) = \prod_{t=1} p(w_t \mid w_1, w_2, \dots, w_{t-1})$$

$$p_{\leftarrow}(w_1, w_2, \dots, w_n) = \prod_{t=1}^n p(w_t \mid w_{t+1}, w_{t+2}, \dots, w_n)$$

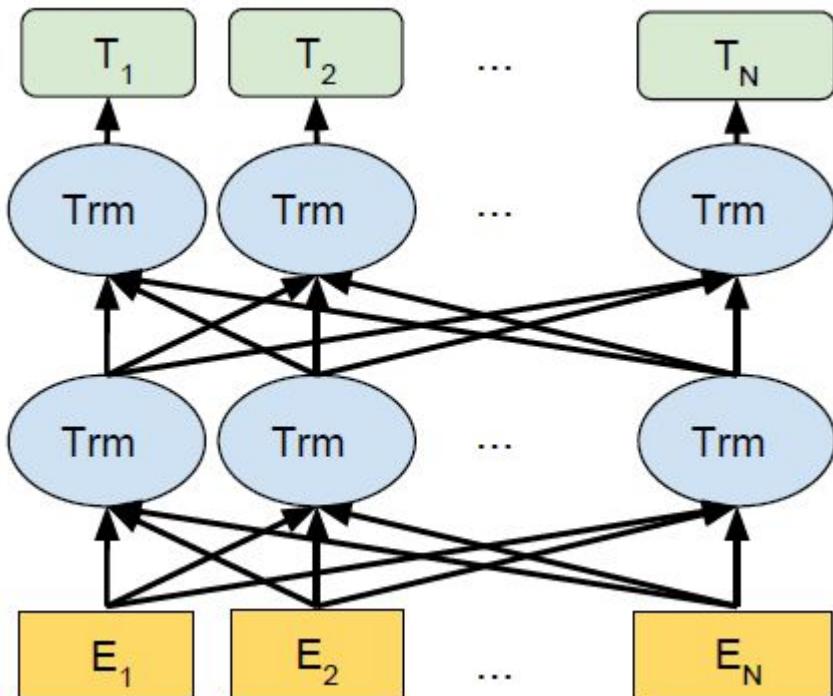
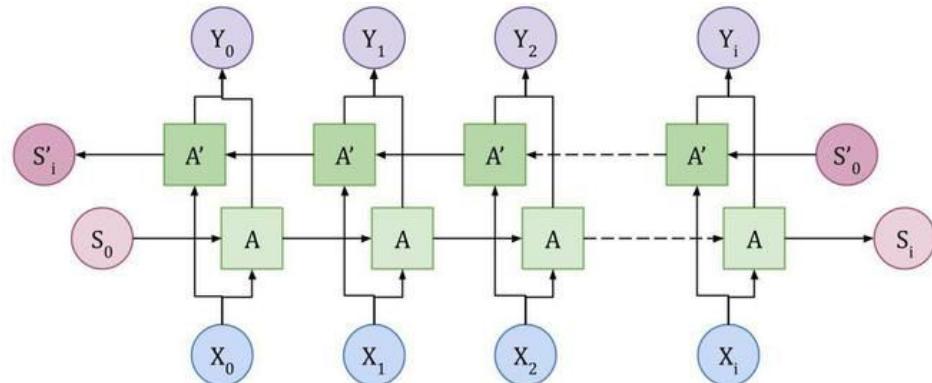
$$\log p_{\text{bi}}(w_1, w_2, \dots, w_n) = \sum_{t=1}^n [\log p_{\rightarrow}(w_t \mid w_{<t}) + \log p_{\leftarrow}(w_t \mid w_{>t})]$$

estilo **ELMO (LSTM)**

$$\log p(w_t \mid w_1, \dots, w_{t-1}, [\text{MASK}], w_{t+1}, \dots, w_n)$$

estilo **BERT (Transformer)**

MODELOS DE LENGUAJE BIDIRECCIONALES





REPASO: TRANSFER LEARNING + RNN + ATTENTION

UNSUPERVISED LEARNING (también llamado self-supervised)

Es no supervisado en tanto los datos no son etiquetados por humanos, se utilizan los datos as-is.

Se sitúa dentro del **aprendizaje no supervisado**, pero difiere al definir una **tarea pretexto** (una tarea general) que permite generar ejemplos etiquetados sin intervención humana.

Sean $X \in \mathcal{X}$ los datos sin etiquetar.

Definimos una transformación $T(\cdot)$ (como enmascaramiento o aumento) para producir un par $(X_{\text{input}}, X_{\text{target}})$.

- Ej.: X_{input} = texto con máscaras, X_{target} = texto original.

Definimos una función de pérdida $\mathcal{L}(f_\theta(X_{\text{input}}), X_{\text{target}})$, donde f_θ es el modelo.

Objetivo de optimización:

$$\min_{\theta} \mathbb{E}_{x \sim p(X)} [\mathcal{L}(f_\theta(T(X)), \text{target}(T(X)))]$$

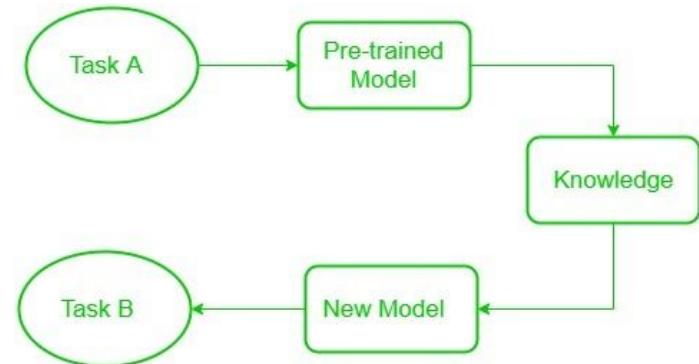
```
1 import torch
2 import torch.nn as nn
3 from transformers import BertTokenizer, BertForMaskedLM, AdamW
4
5 # 1. Inicialización del modelo y tokenizer
6 tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
7 model = BertForMaskedLM.from_pretrained("bert-base-uncased")
8
9 # 2. Inicialización del optimizador
10 optimizer = AdamW(model.parameters(), lr=5e-5)
11
12 # 3. Texto de entrada sin etiquetar
13 text = "Self-supervised learning creates its own labels from the data."
14
15 # 4. Tokenización con enmascaramiento aleatorio
16 inputs = tokenizer(text, return_tensors="pt")
17 input_ids = inputs["input_ids"].clone()
18 labels = input_ids.clone()
19
20 # 5. Aplicar enmascaramiento al 15% de los tokens (excepto tokens especiales)
21 mask_token_id = tokenizer.mask_token_id
22 probability_matrix = torch.full(labels.shape, 0.15)
23
24 special_tokens_mask = tokenizer.get_special_tokens_mask(
25     | labels[0], already_has_special_token=True
26 )
27 probability_matrix.masked_fill_(
28     | torch.tensor(special_tokens_mask, dtype=torch.bool), value=0.0
29 )
30
31 masked_indices = torch.bernoulli(probability_matrix).bool()
32 input_ids[masked_indices] = mask_token_id
33
34 # 6. Ignorar pérdida para los tokens no enmascarados
35 labels[~masked_indices] = -100 # Se ignoran en la función de pérdida
36
37 # 7. Adelante y cálculo de pérdida
38 model.train()
39 outputs = model(input_ids=input_ids, labels=labels)
40 loss = outputs.loss
41 loss.backward()
42
43 # 8. Paso del optimizador
44 optimizer.step()
45 optimizer.zero_grad()
```

TRANSFER LEARNING: MAXIMIZANDO EL CONOCIMIENTO PREEXISTENTE



Transfer Learning: Técnica donde un modelo desarrollado para una tarea se reutiliza como punto de partida para un modelo en una segunda tarea.

- Aprovechar el conocimiento aprendido de tareas anteriores para acelerar o mejorar el aprendizaje en tareas nuevas o relacionadas.
- Reducción en el tiempo y recursos de entrenamiento, especialmente cuando se tienen datos limitados para la nueva tarea.



TRANSFER LEARNING EN REDES NEURONALES



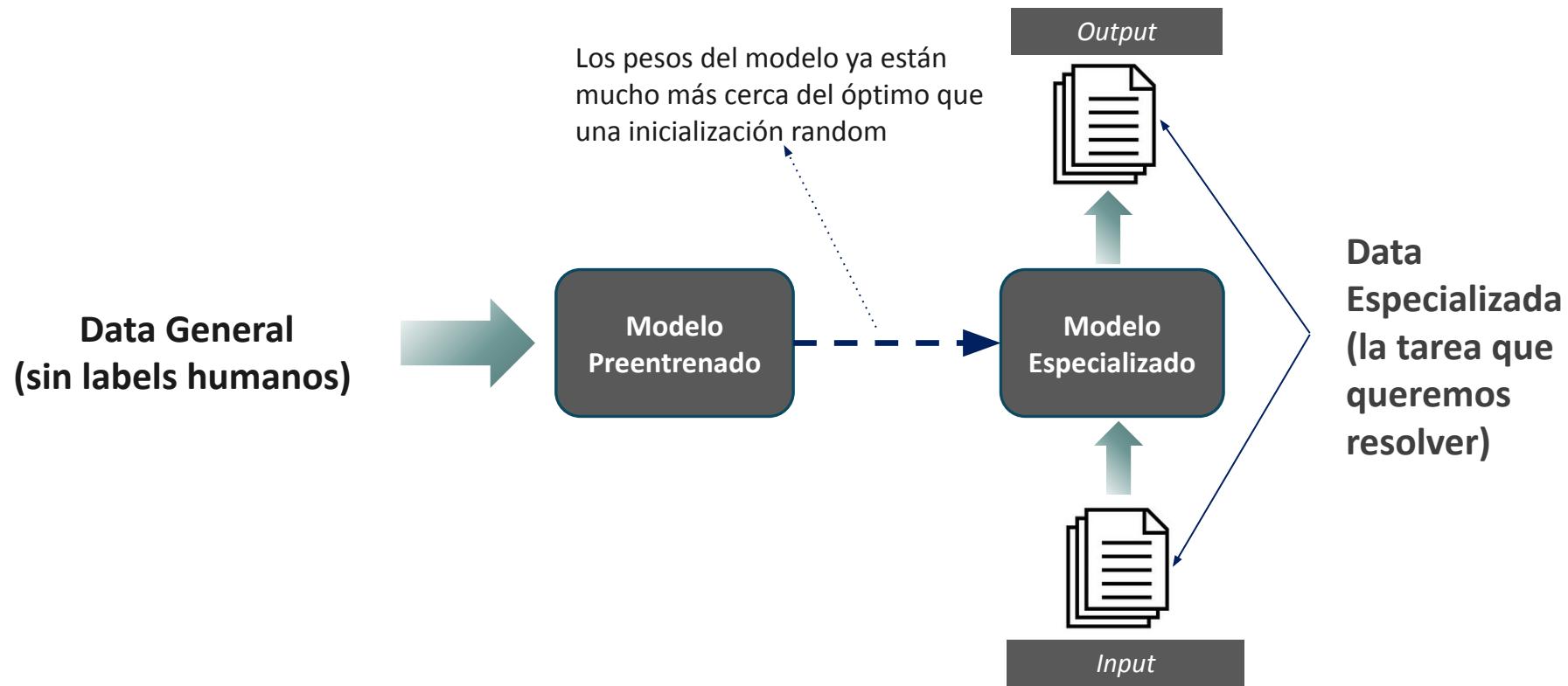
Transfer Learning se refiere al uso de representaciones aprendidas en tareas generales an grandes corpus de texto, en otras tareas más específicas.

Las primeras capas de una red neuronal suelen aprender características generales (POS tags, relaciones sintácticas, etc), mientras que las capas más profundas aprenden características más específicas para la tarea original (semántica).

Ejemplos:

- Uso de modelos preentrenados en ImageNet para tareas de clasificación de imágenes específicas.
- Uso de embeddings de palabras preentrenados (como Word2Vec o BERT) para tareas específicas de procesamiento de lenguaje natural.
- Uso de redes pre entrenadas para tareas especializadas (fine tuning)

PRETRAINING Y FINE TUNING



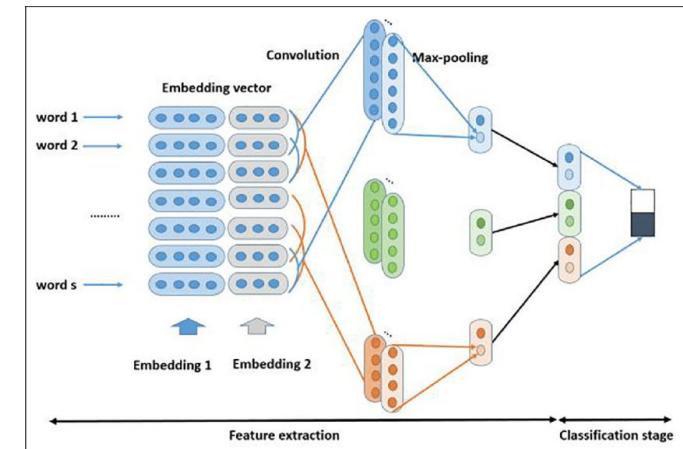
WORD VECTORS EN DEEP LEARNING

Representaciones numéricas de palabras basadas en su significado y contexto

- **Compactos:** Capturan la semántica de una palabra en un vector compacto.
- Palabras con significados similares tienen vectores cercanos en el espacio vectorial.

Integración con Redes Neuronales

- Las redes neuronales requieren inputs numéricos.
- Los vectores de palabras se adaptan naturalmente a este requisito, input ideal para redes neuronales en tareas de NLP.



EMBEDDINGS AS FEATURES



Embedding



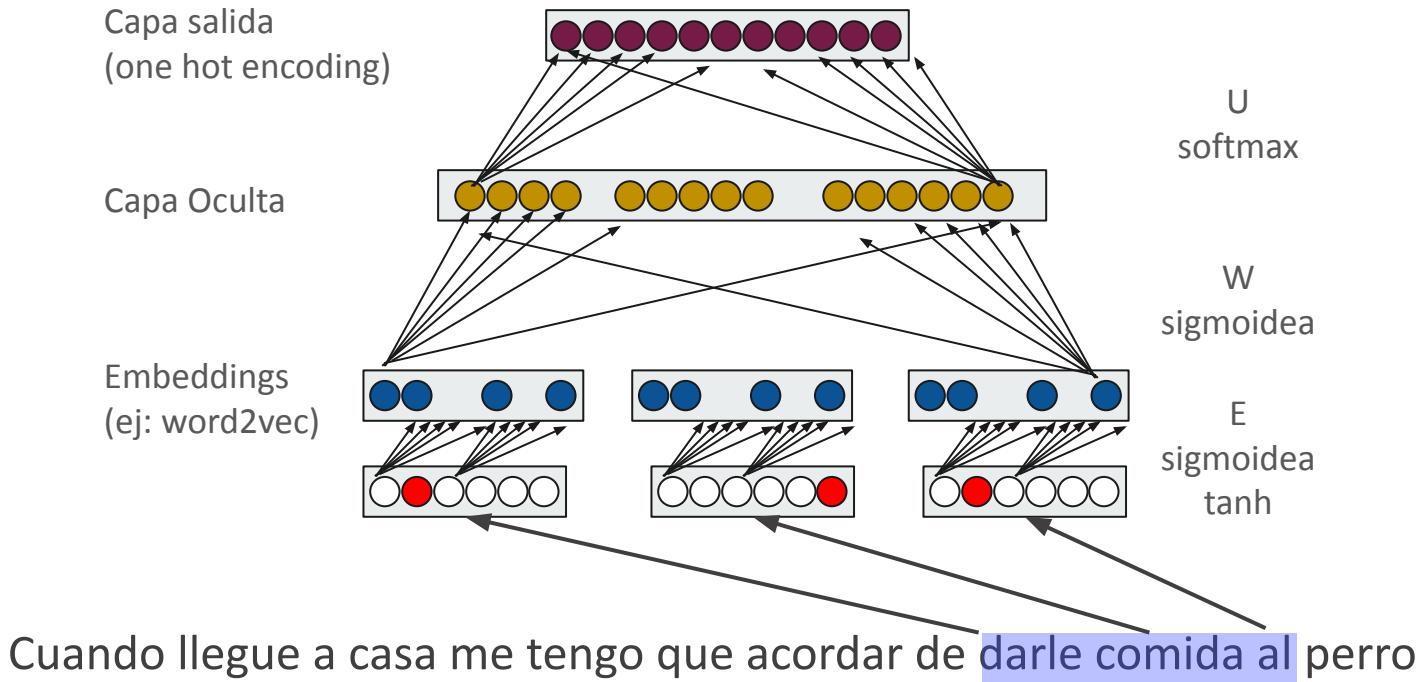
Que → (0.4,0.9,...0.3)
bien → (0.5,0.8,...0.5)
... → (...)
infalible → (0.1,0.9,...0.1)



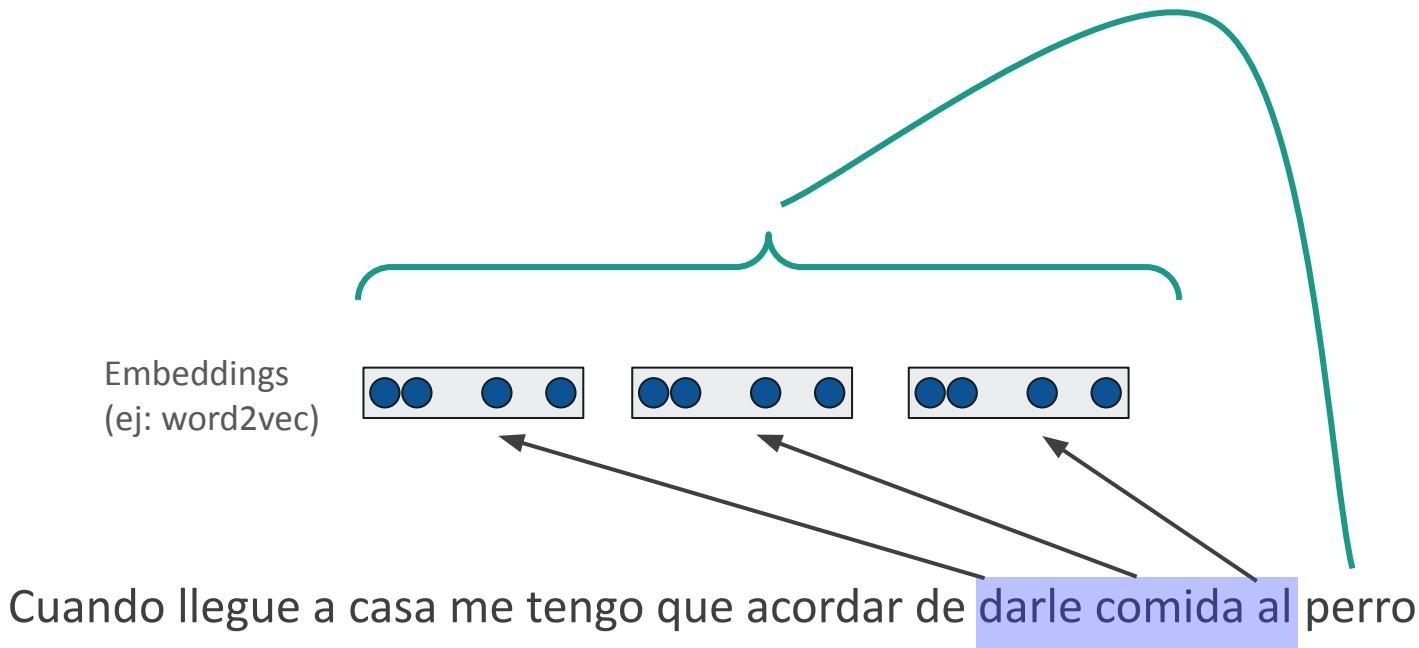
Prediction

Deep Neural Network

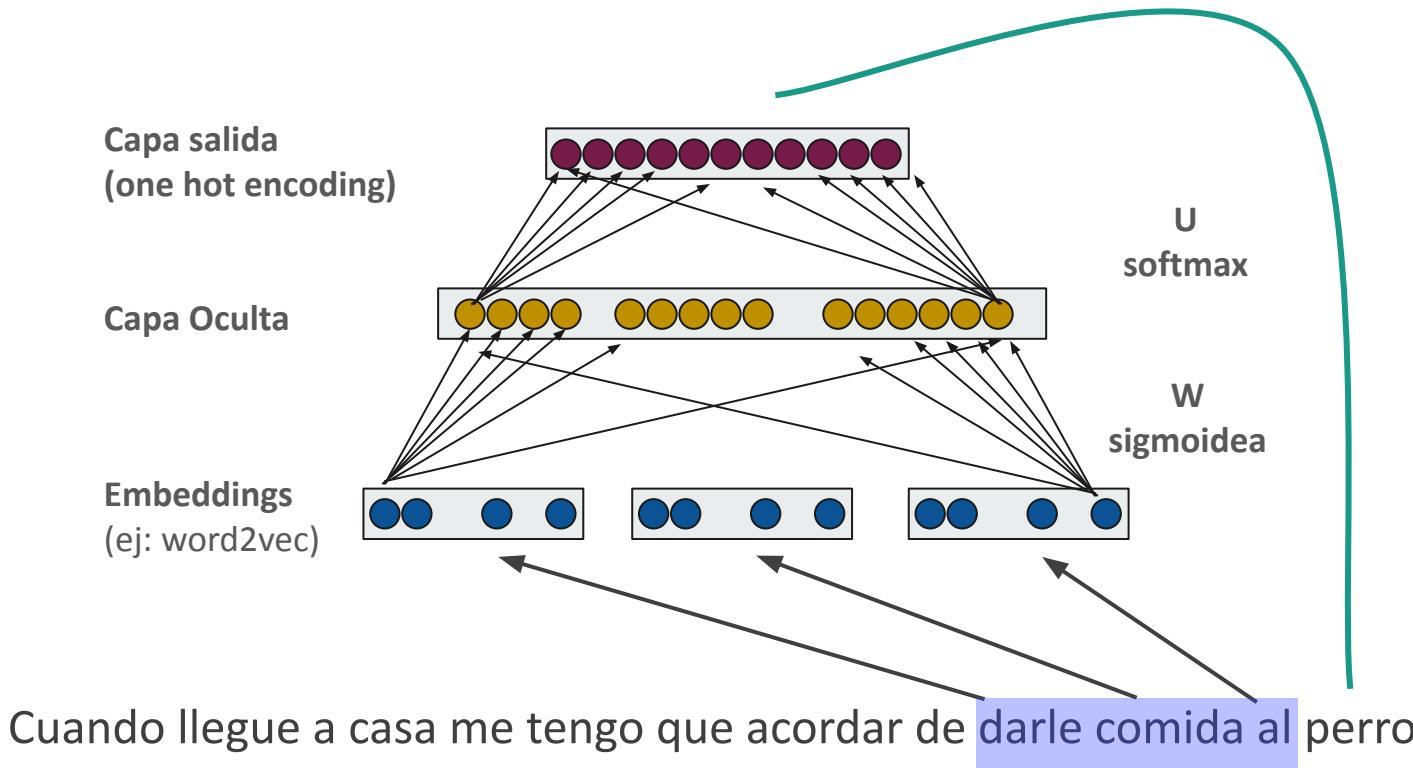
NEURAL LM



NEURAL LM



NEURAL LM





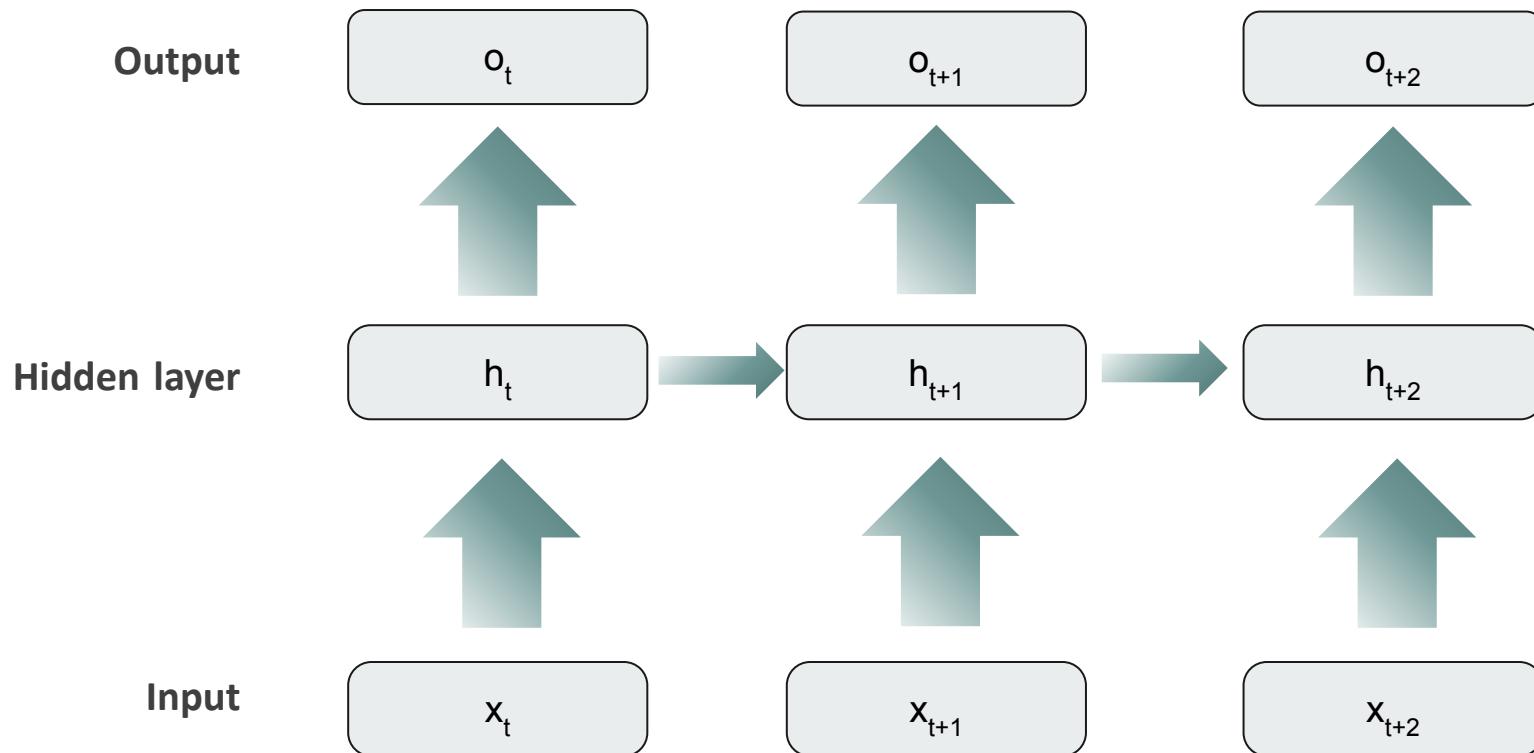
REDES NEURONALES RECURRENTES

REDES NEURONALES RECURRENTES

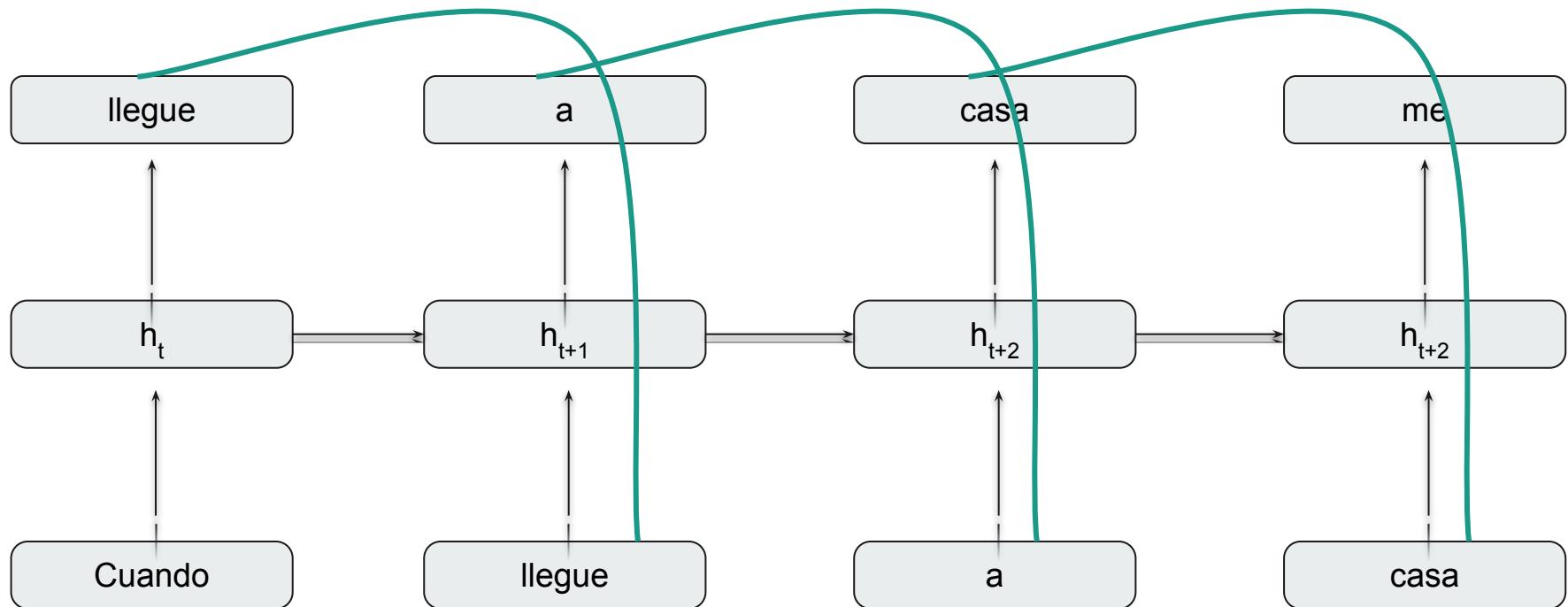
Diseñadas específicamente para trabajar con datos secuenciales. Las RNR tienen la capacidad de "recordar" información de pasos anteriores.

- **Conexiones que se Retroalimentan:** Tienen conexiones que se retroalimentan hacia sí mismas o hacia unidades anteriores. Mantiene un "estado" o memoria de los datos previos.
- **Memoria:** Capacidad de recordar pasos anteriores.
- **Procesamiento Secuencial:** Procesan los datos de entrada uno a uno, manteniendo y actualizando su estado interno a medida que avanzan.
- **Flexibilidad en Longitudes de Secuencia:** Las RNR pueden procesar secuencias de diferentes longitudes, ideales para tareas como el procesamiento de lenguaje natural donde las oraciones y los documentos varían en tamaño.

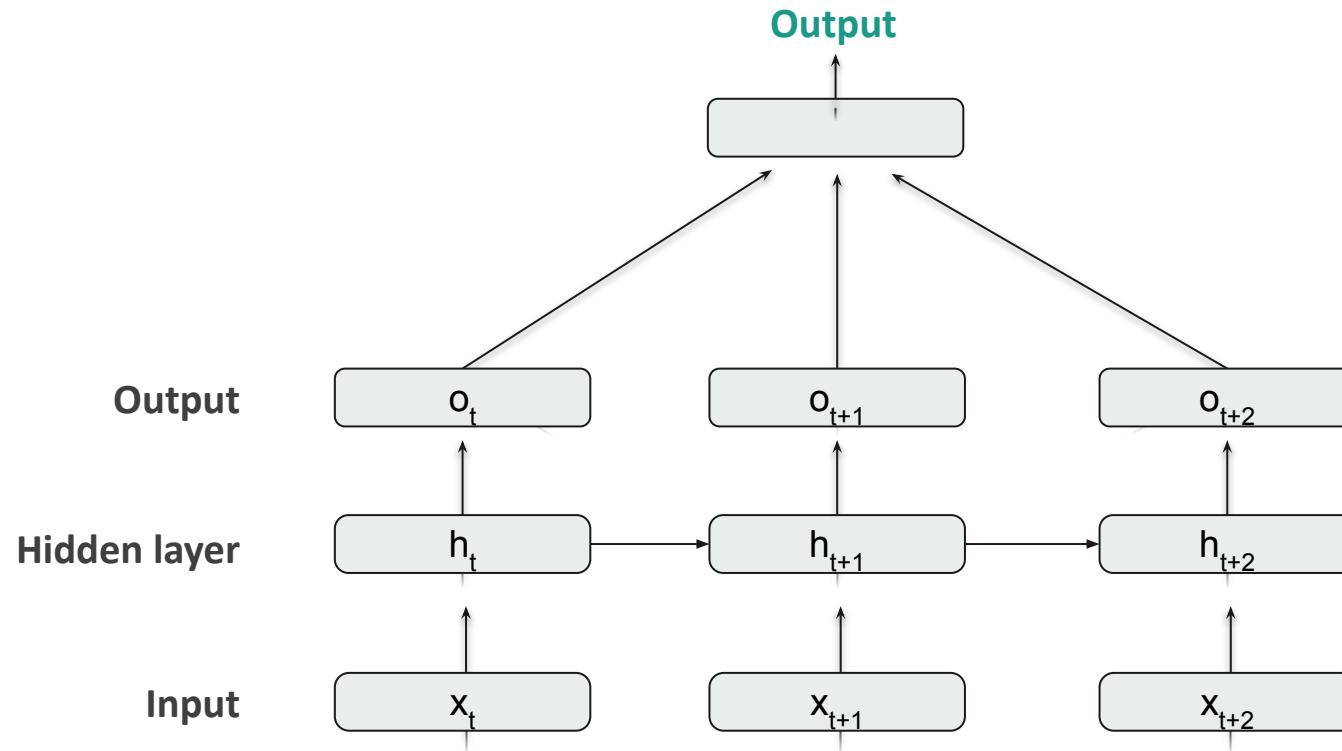
RECURRENT NEURAL NETWORK



GENERACIÓN DE TEXTO

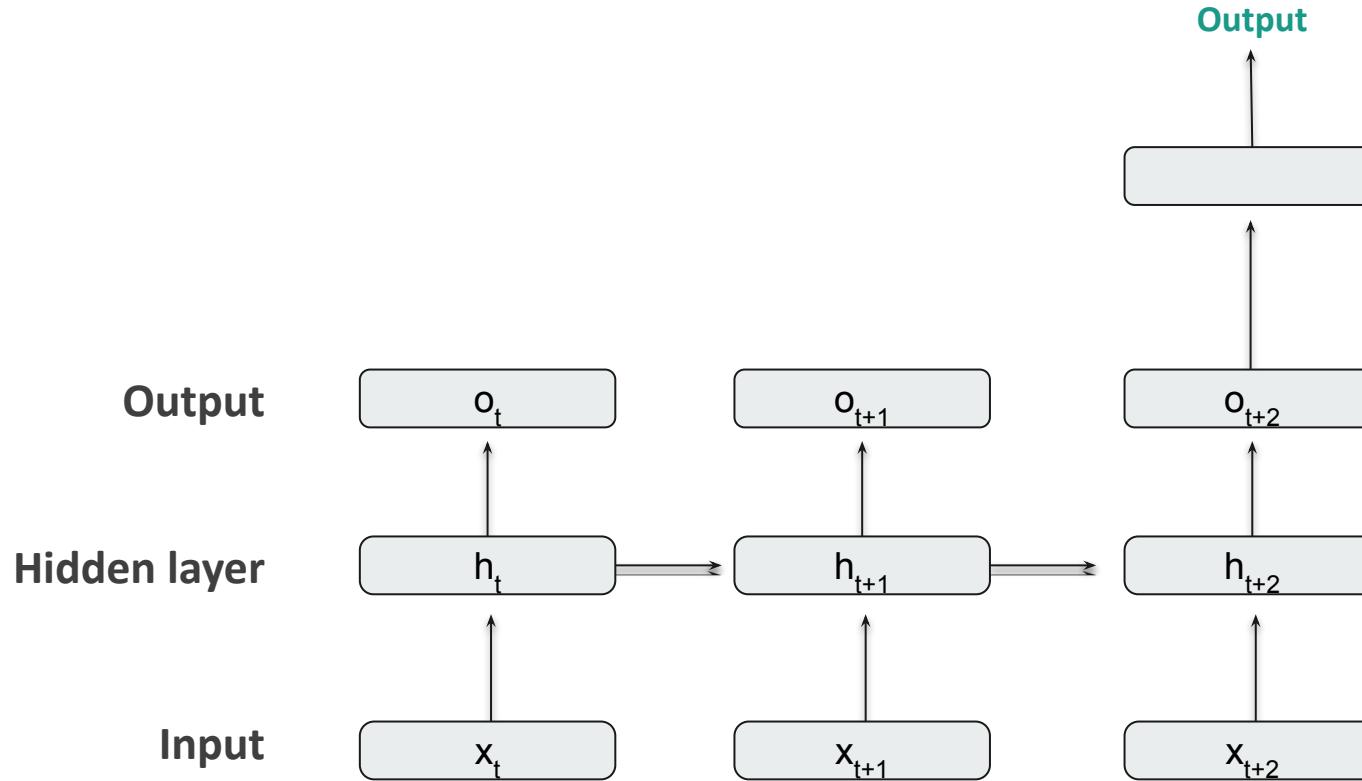


CLASIFICACIÓN

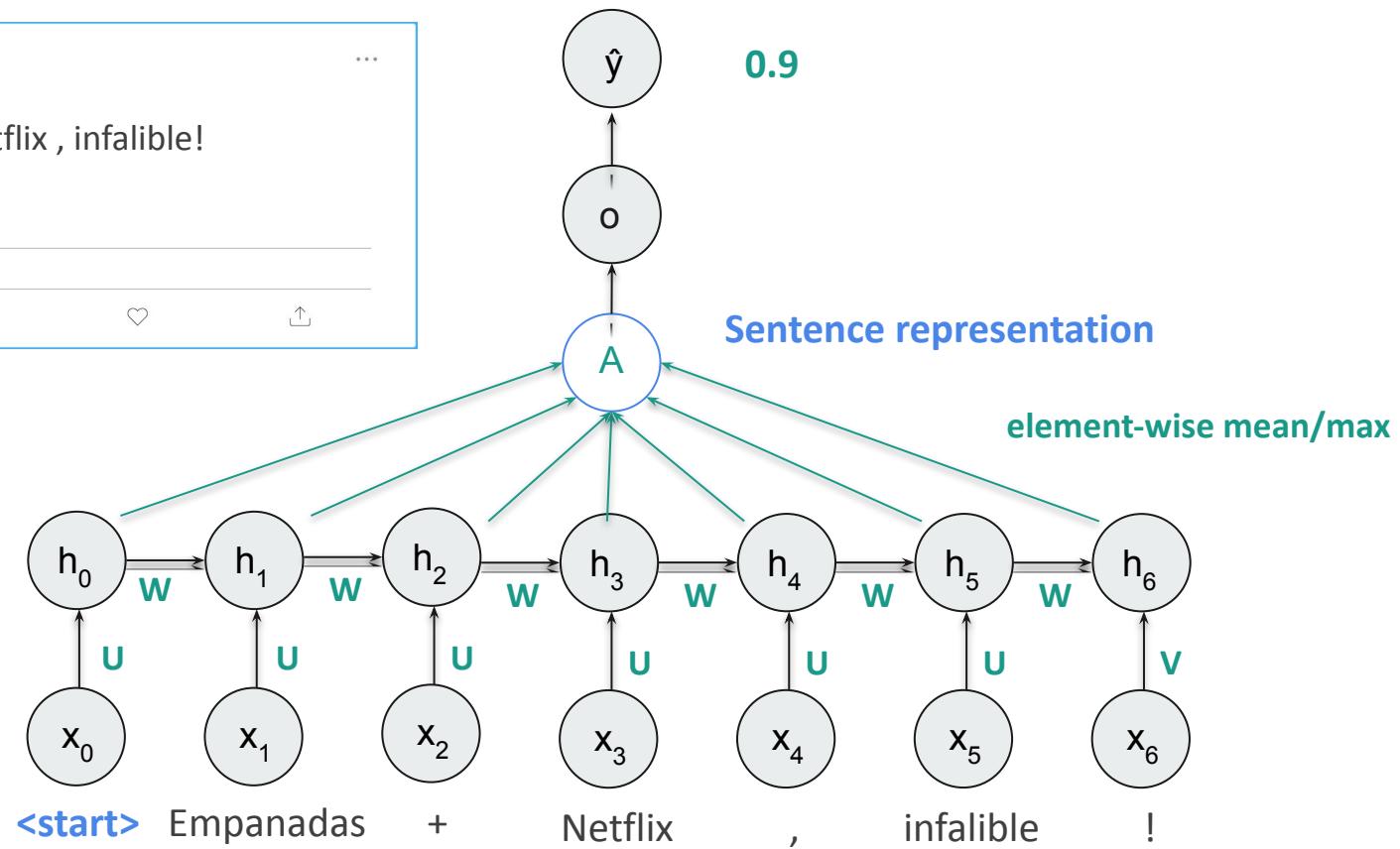


¿Cómo se genera esa representación?

CLASIFICACIÓN



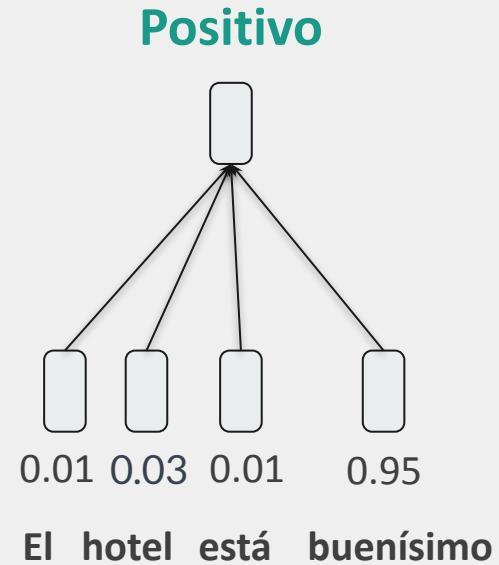
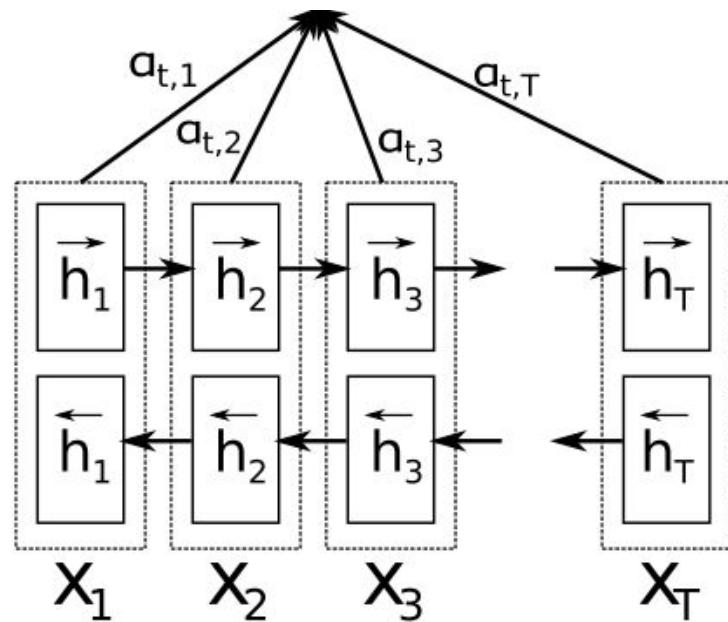
CLASIFICACIÓN





ATENCIÓN

MECANISMO DE ATENCIÓN (ATTENTION)



MECANISMO DE ATENCIÓN (ATTENTION)

- **Selector de Inputs:** No todas las palabras en una oración contribuyen igualmente a la decisión.
- Permite a los modelos centrarse en diferentes partes de la entrada, emulando cómo los humanos prestan "atención" a partes específicas del contexto.



Atención:

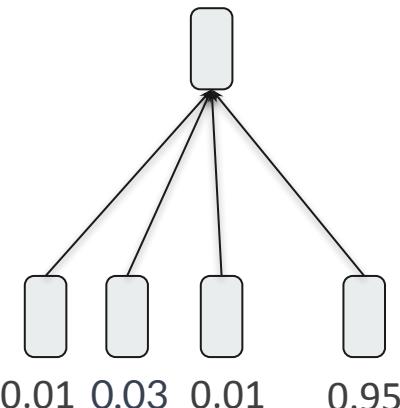
1. **Ponderación de Entradas:** Se asignan pesos a diferentes partes de la entrada según su relevancia.
2. **Combinación Contextual:** Se forma un vector de contexto con una combinación ponderada las entradas según sus pesos.
3. **Generación de Salida:** El vector de contexto se utiliza para generar la salida (normalmente un FF).



Ventajas:

- Captura de Relaciones a Largo Plazo
- Mejora del Rendimiento
- Interpretabilidad: Los pesos de atención ofrecen una visión sobre qué partes de la entrada el modelo considera importantes

Positivo



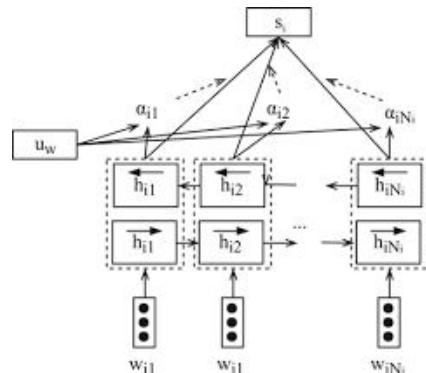
El hotel está buenísimo

MECANISMO DE ATENCIÓN (ATTENTION)



Atención:

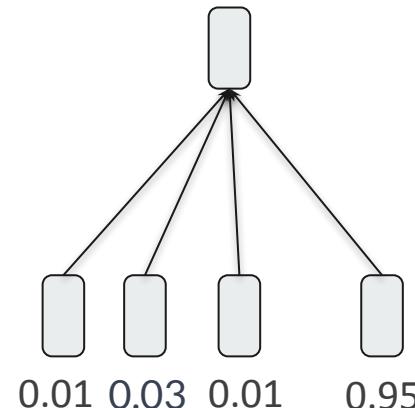
1. **Ponderación de Entradas:** Se asignan pesos a diferentes partes de la entrada según su relevancia.
2. **Combinación Contextual:** Se forma un vector de contexto con una combinación ponderada las entradas según sus pesos.
3. **Generación de Salida:** El vector de contexto se utiliza para generar la salida (normalmente un FF).



$$u_{it} = \tanh(W_w h_{it} + b_w)$$
$$\alpha_{it} = \frac{\exp(u_{it}^\top u_w)}{\sum_t \exp(u_{it}^\top u_w)}$$
$$s_i = \sum_t \alpha_{it} h_{it}.$$

Hierarchical Attention Networks for Document Classification, Yang et al, 2016

Positivo



El hotel está buenísmo

MECANISMO DE ATENCIÓN (ATTENTION)

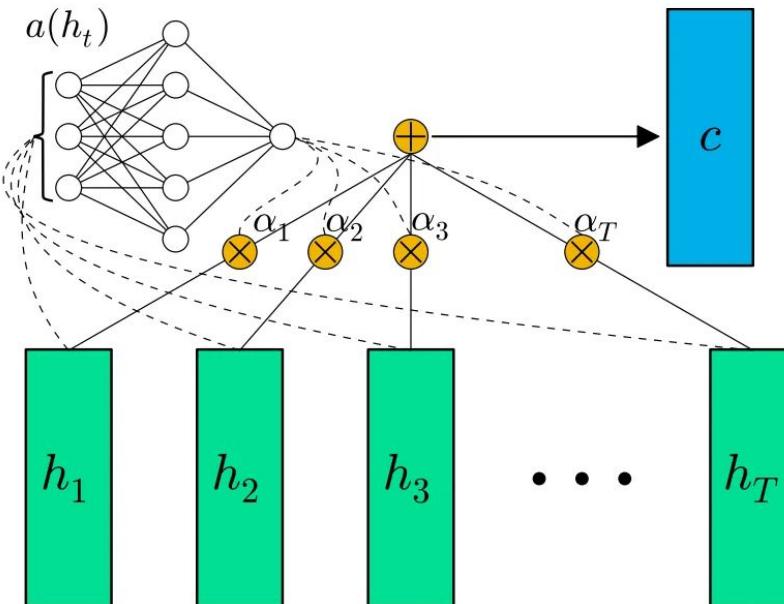
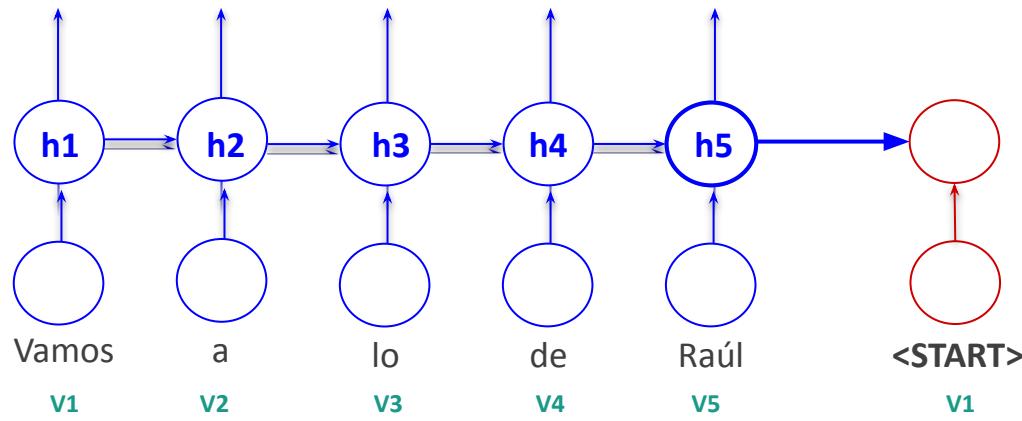


Figure 1: Schematic of our proposed “feed-forward” attention mechanism (cf. (Cho, 2015) Figure 1). Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α . The vector c is computed as a weighted average of h_t , with weighting given by α .

ATENCIÓN

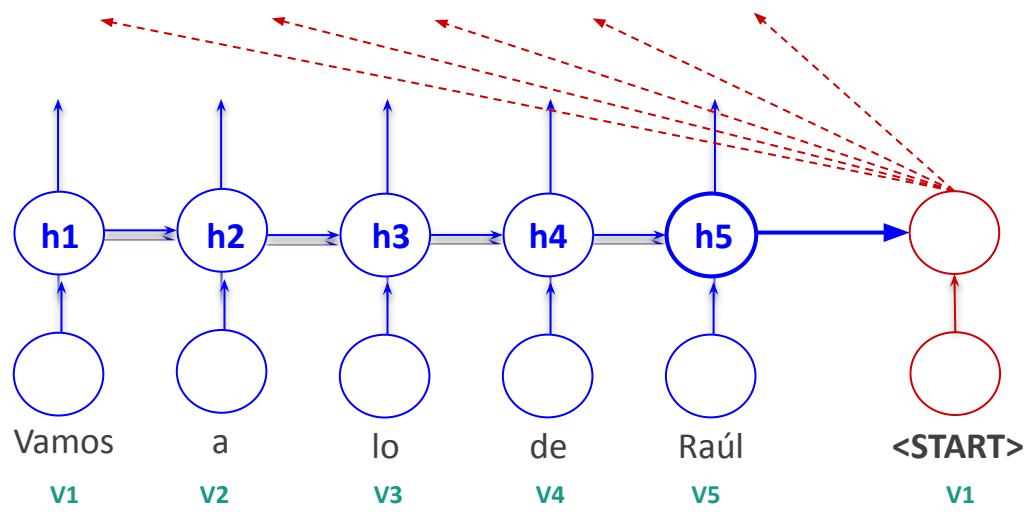
Encoder RNN



Decoder RNN

ATENCIÓN

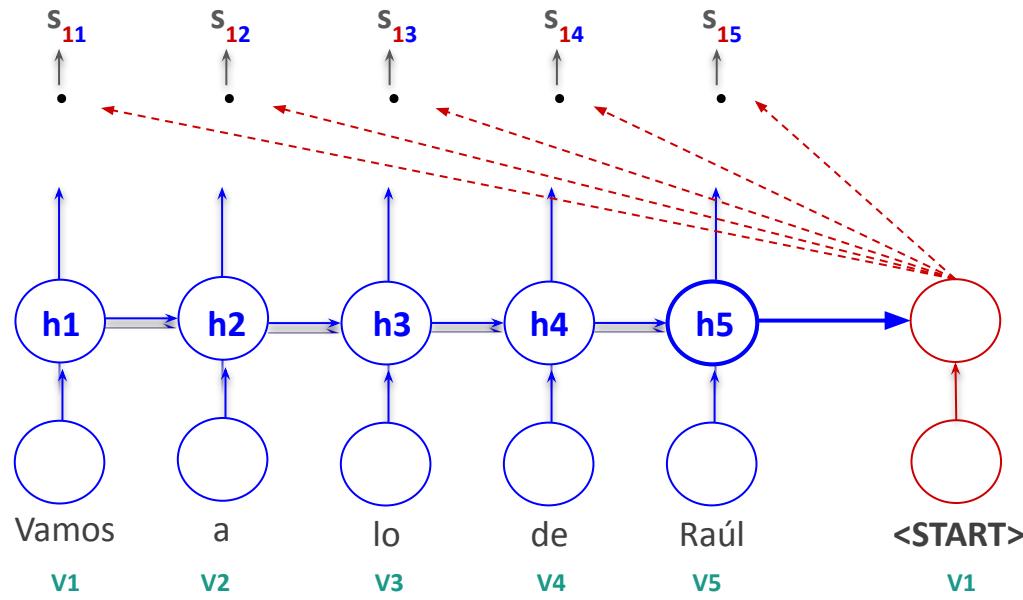
Encoder RNN



Decoder RNN

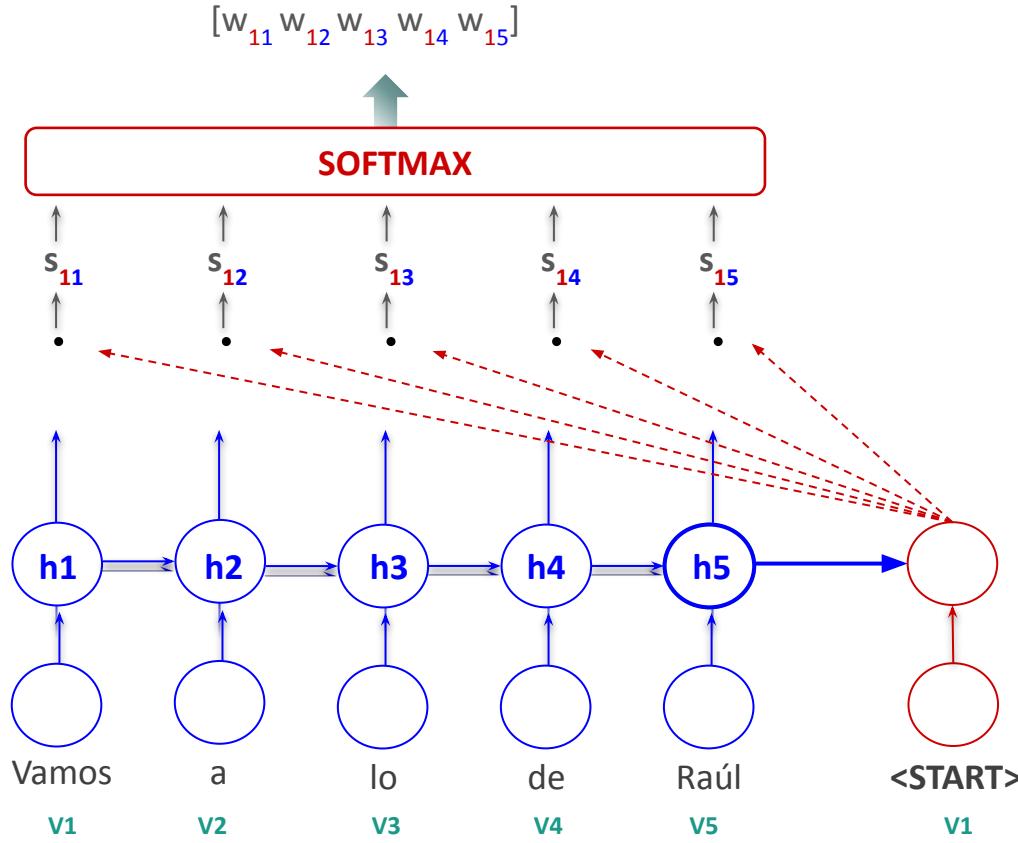
ATENCIÓN

Encoder RNN



ATENCIÓN

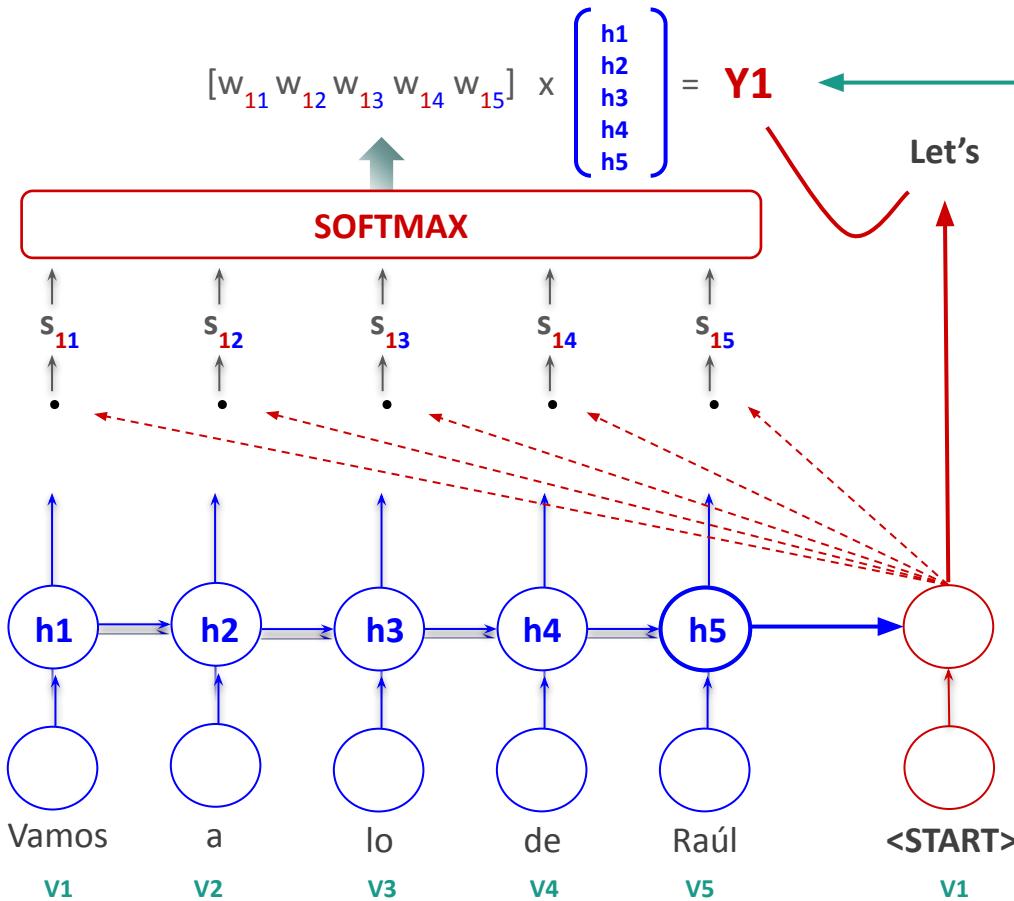
Encoder RNN



Decoder RNN

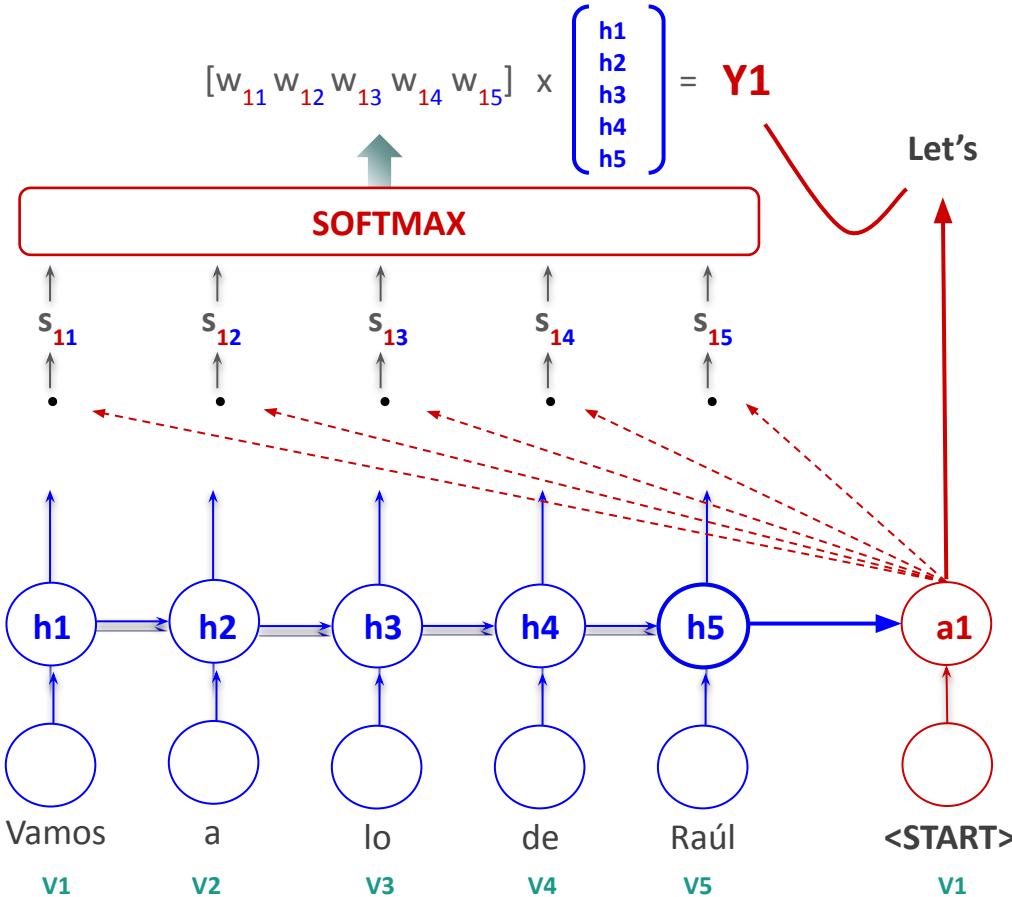
ATENCIÓN

Encoder RNN



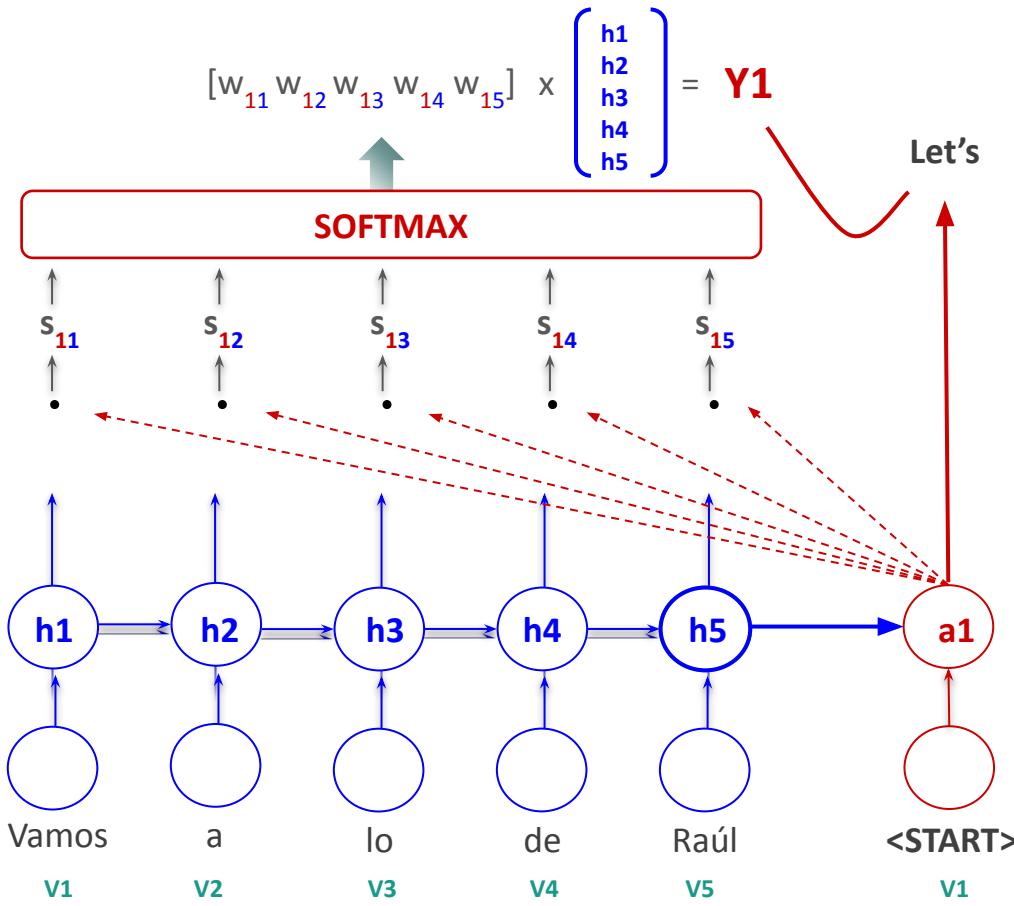
ATENCIÓN

Encoder RNN



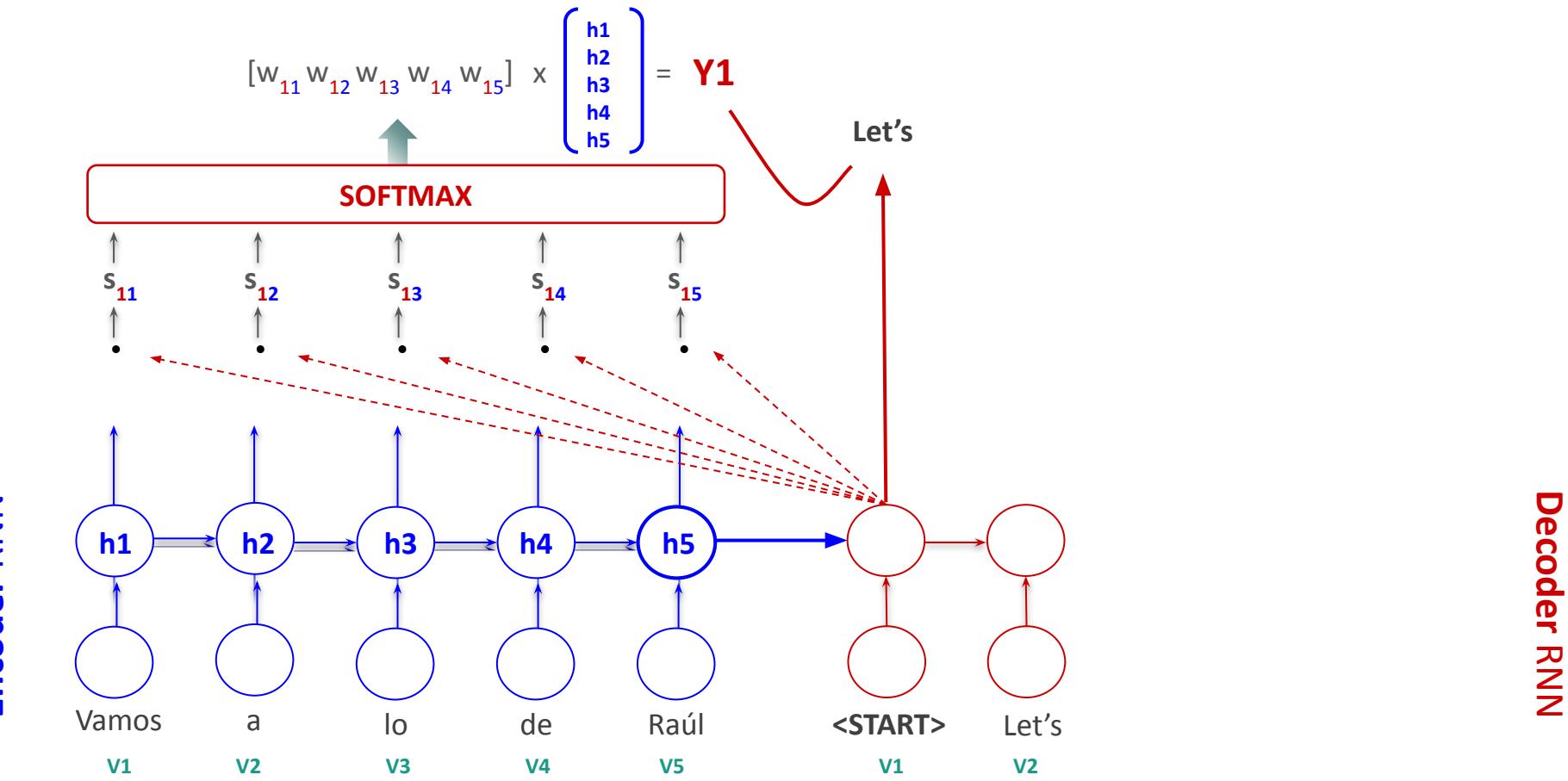
ATENCIÓN

Encoder RNN

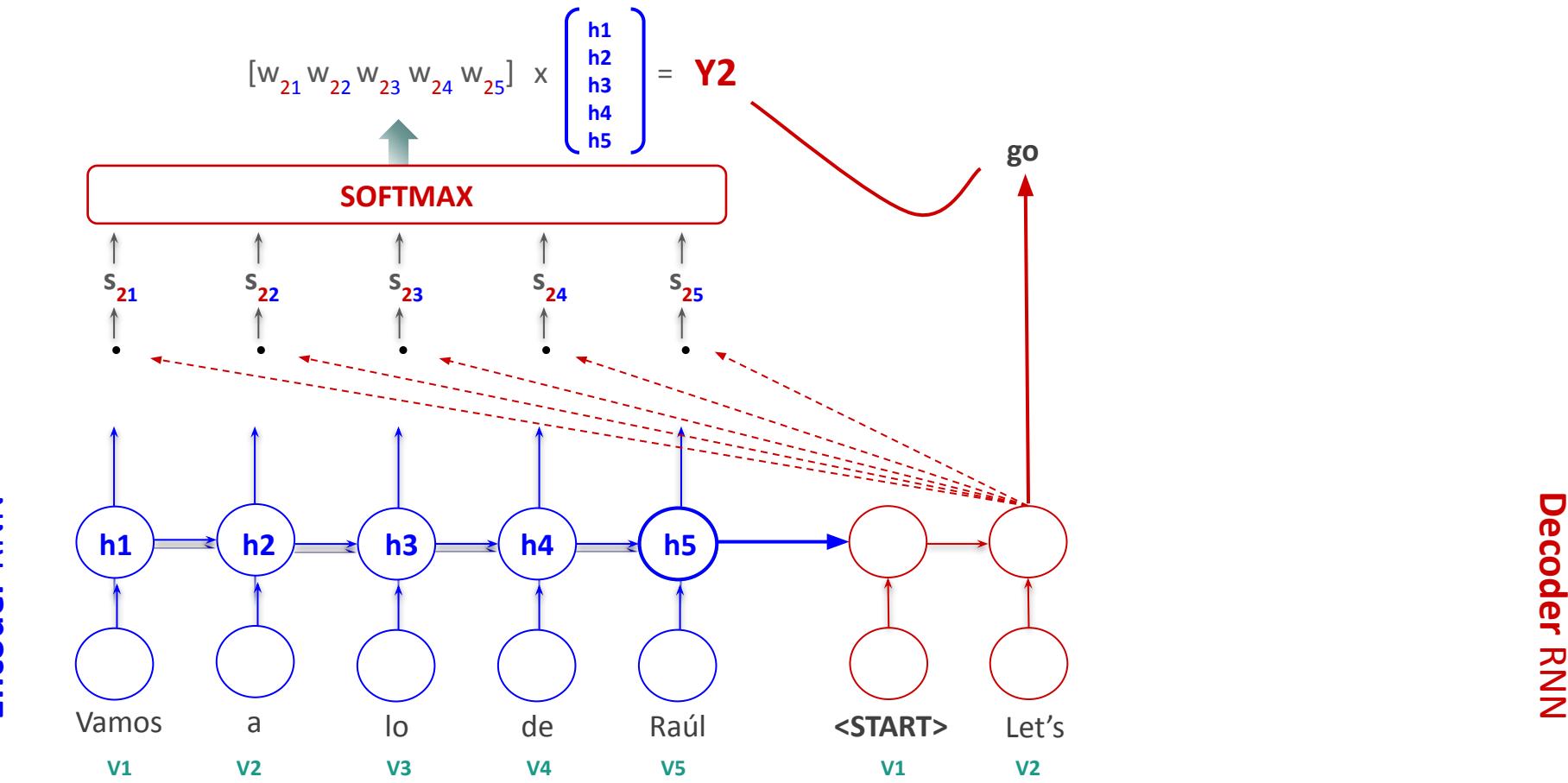


Decoder RNN

ATENCIÓN



ATENCIÓN

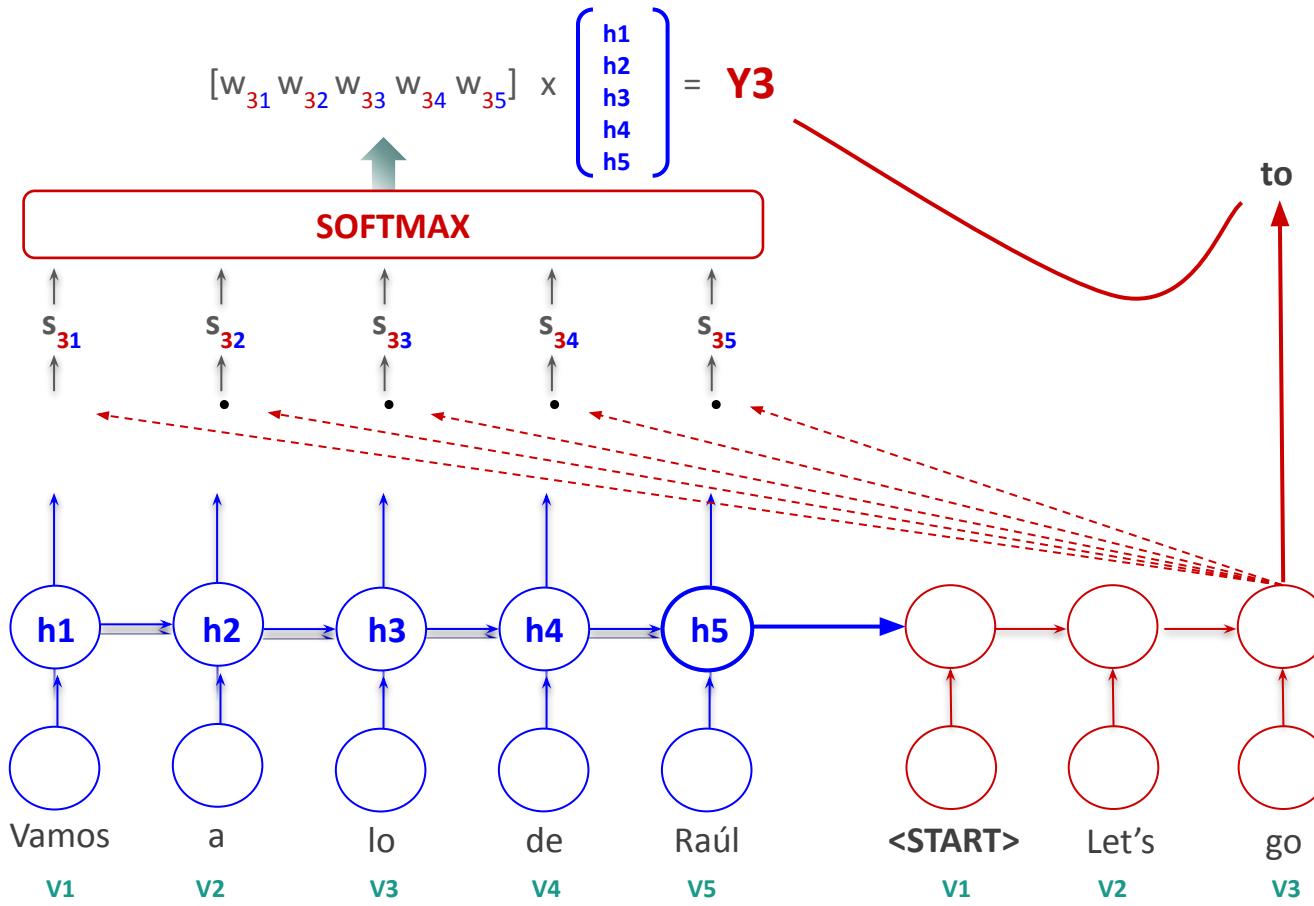


Encoder RNN

Decoder RNN

ATENCIÓN

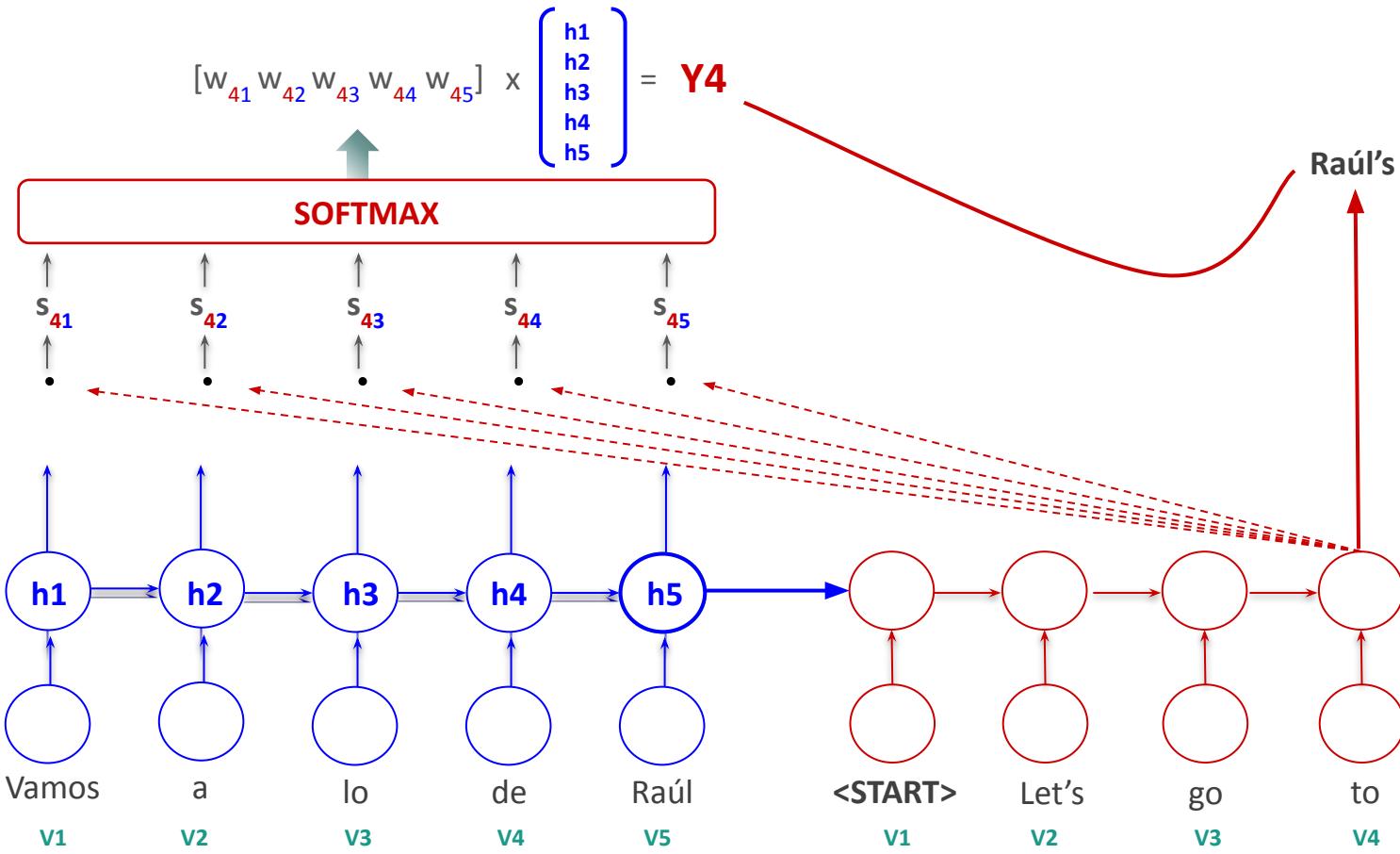
Encoder RNN



Decoder RNN

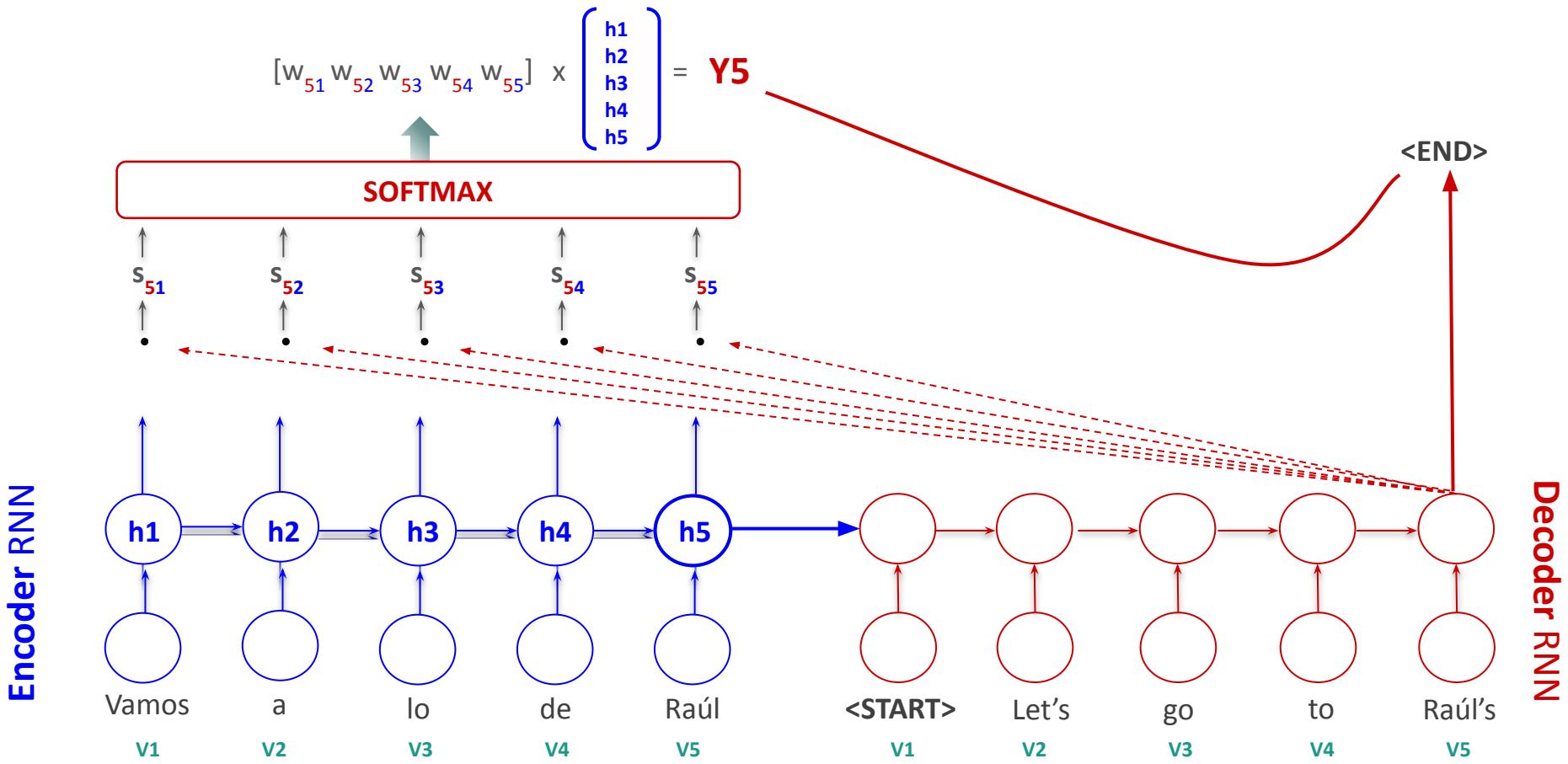
ATENCIÓN

Encoder RNN



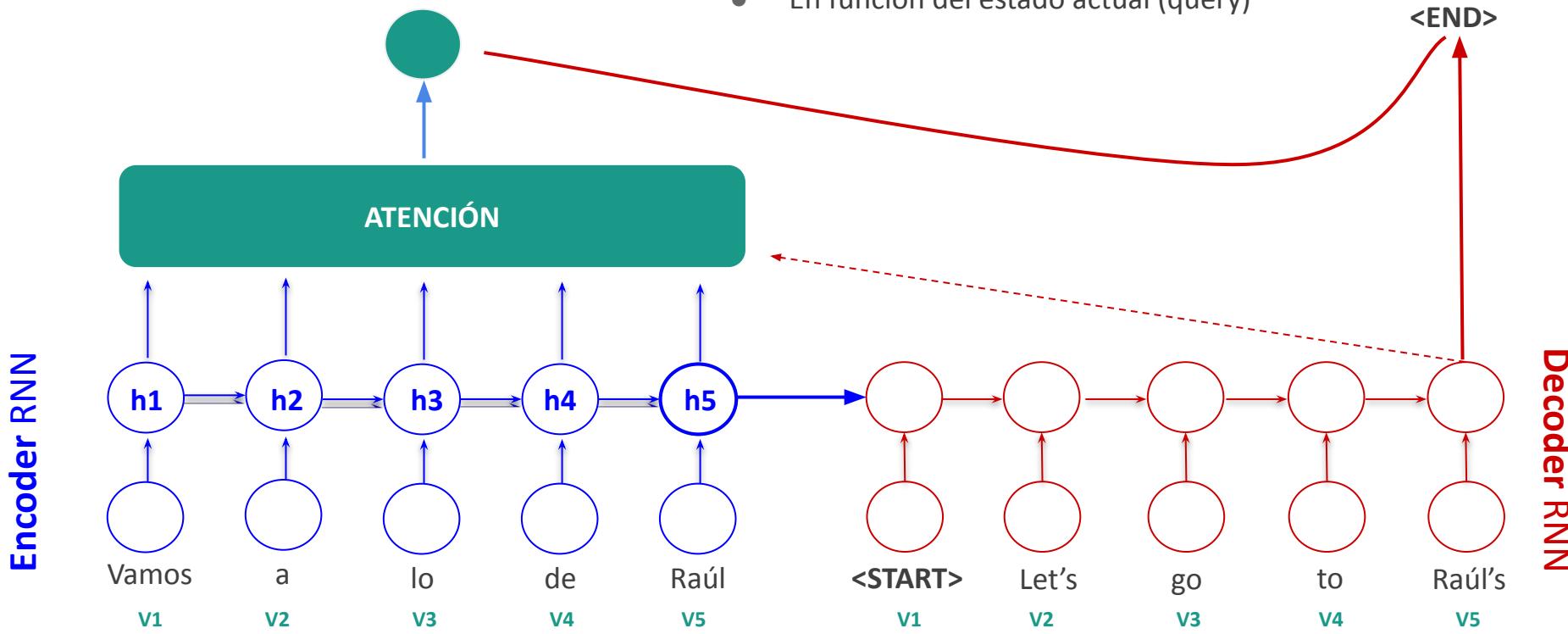
Decoder RNN

ATENCIÓN



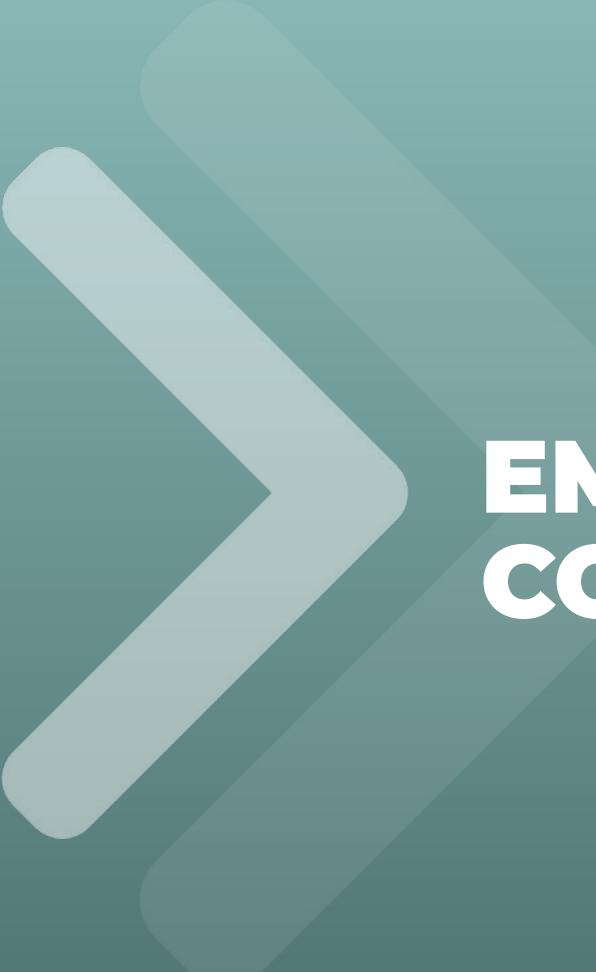
ATENCIÓN

- Tamaño fijo
- Información de toda la secuencia (valores)
- En función del estado actual (query)



ATENCIÓN

- Mejora las tareas de seq2seq como la traducción
- Resuelve el problema del cuello de botella
- Ayuda a evitar el vanishing gradient
- Ayuda a la interpretabilidad de los modelos
- Selección de entradas, importancia



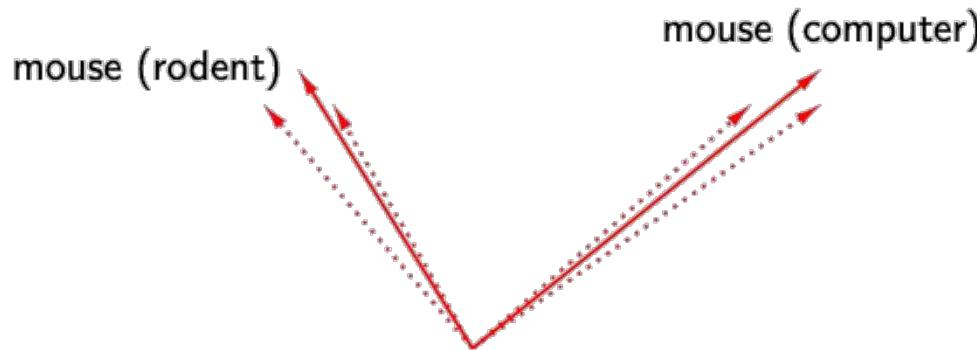
EMBEDDINGS CONTEXTUALES

EMBEDDINGS CONTEXTUALES

➤ Embeddings no específicos a una palabra aislada sino en contexto.

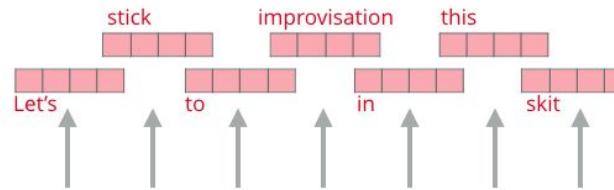
- Saqué dinero del banco
- Me siento en el banco

➤ Generan representaciones dinámicas basadas en el contexto específico de la palabra.



EMBEDDINGS CONTEXTUALES

Embeddings

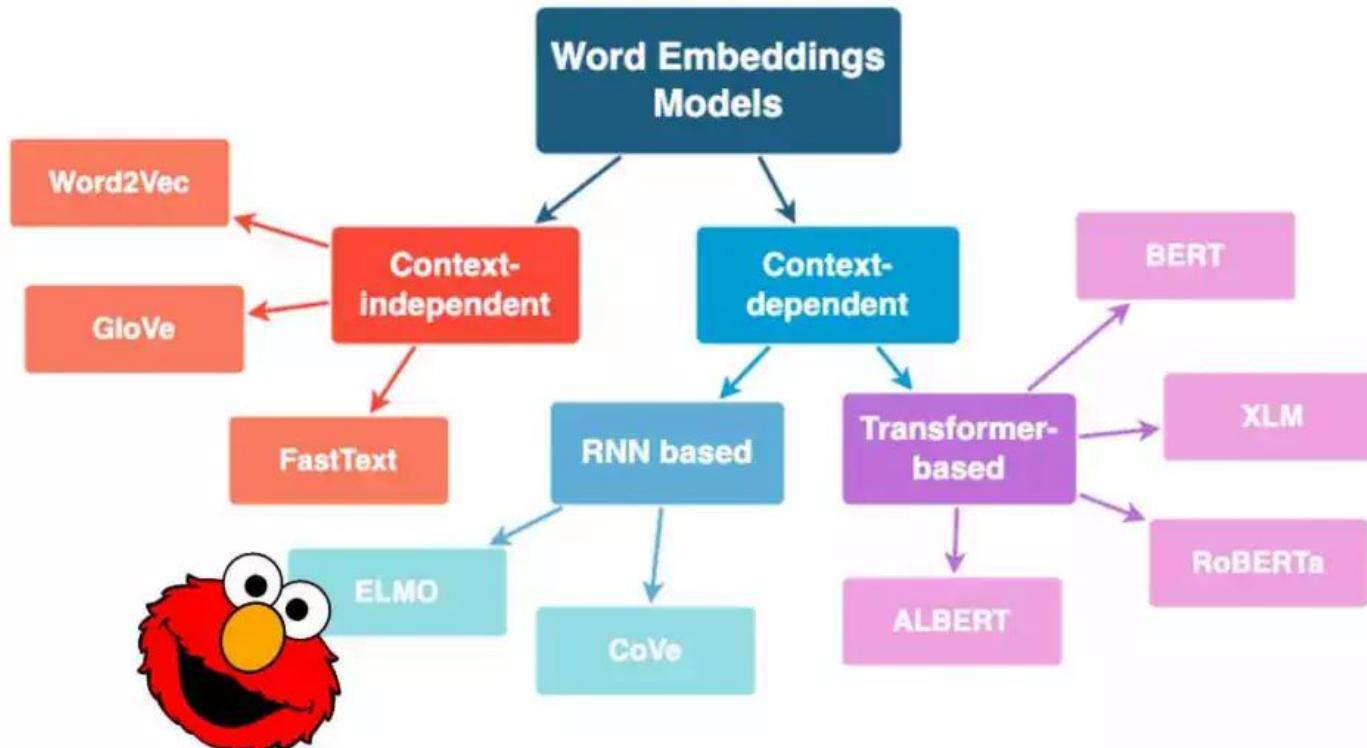


RED NEURONAL

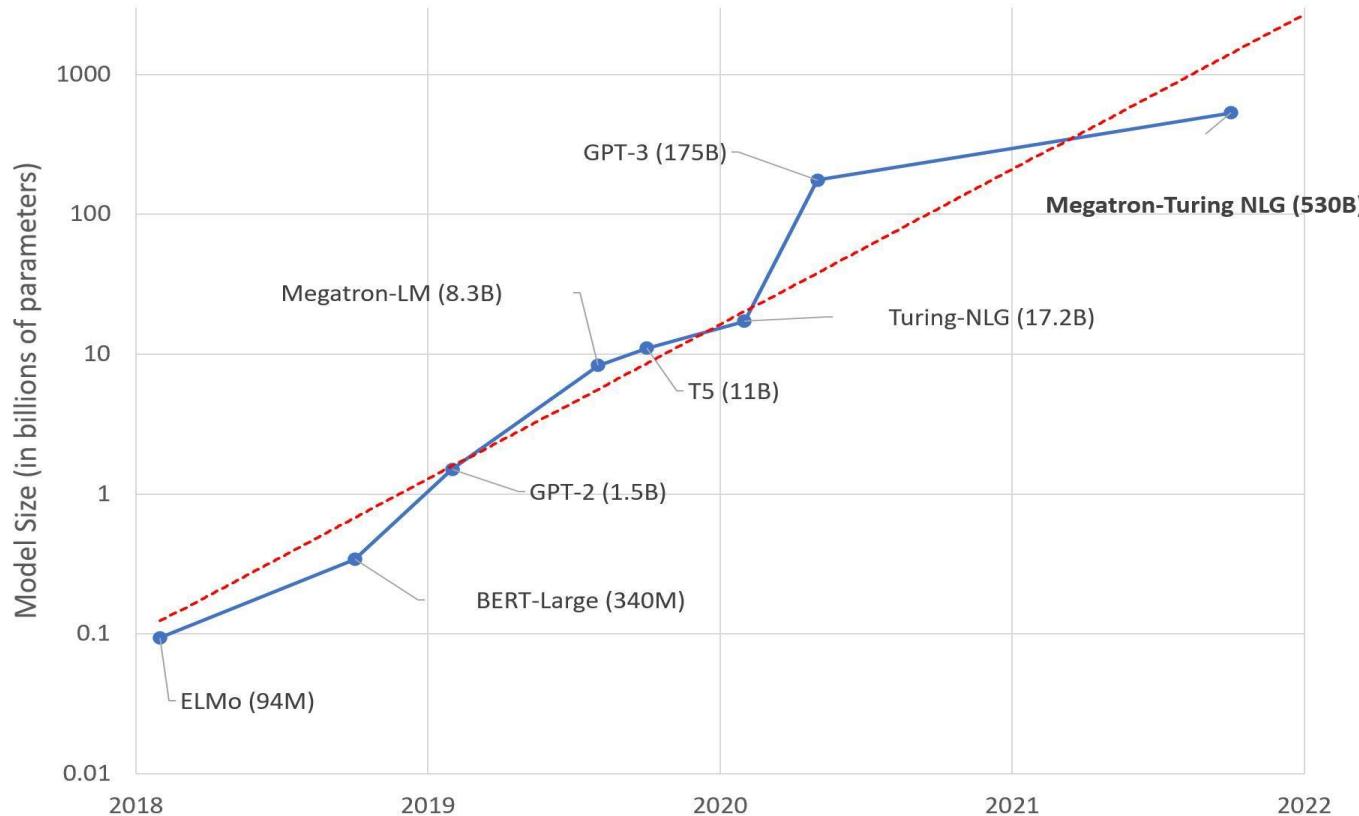
Words to embed



EMBEDDINGS CONTEXTUALES



EMBEDDINGS CONTEXTUALES



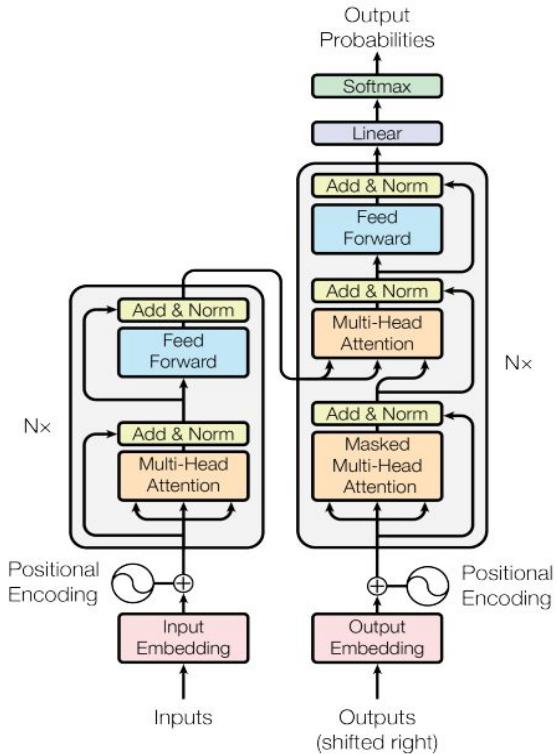


TRANSFORMERS

Vaswani, A. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

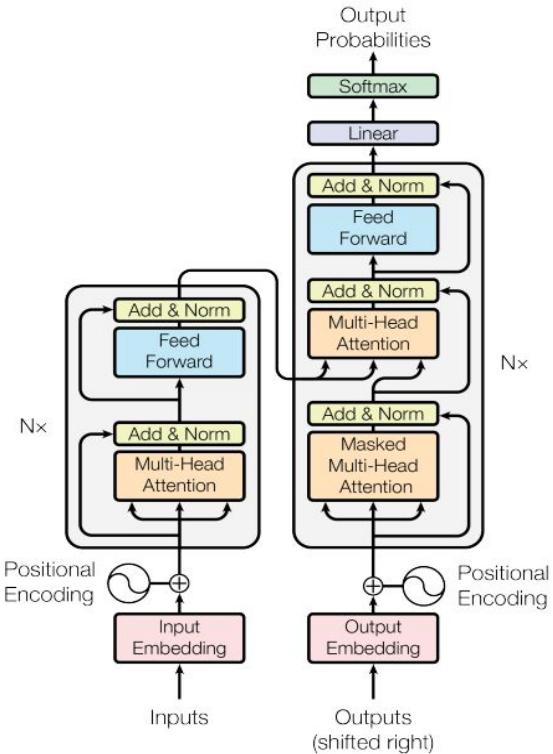
TRANSFORMERS

Voy a guardar el queso en el [?]

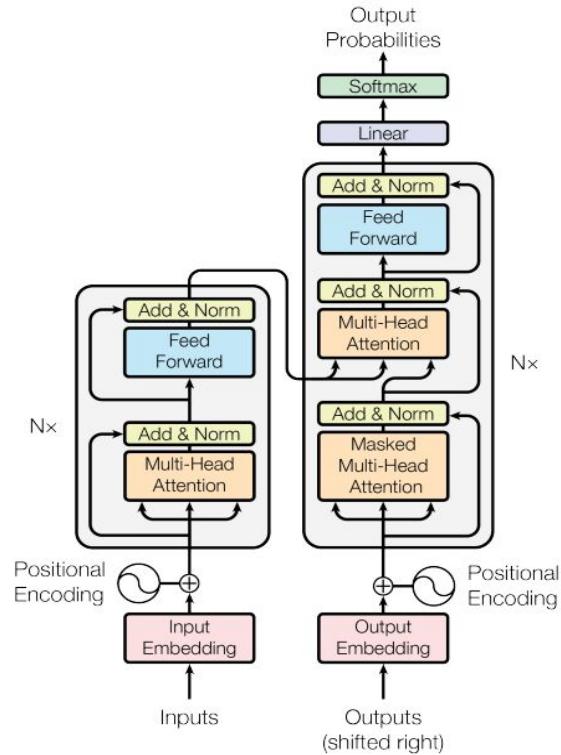


TRANSFORMERS

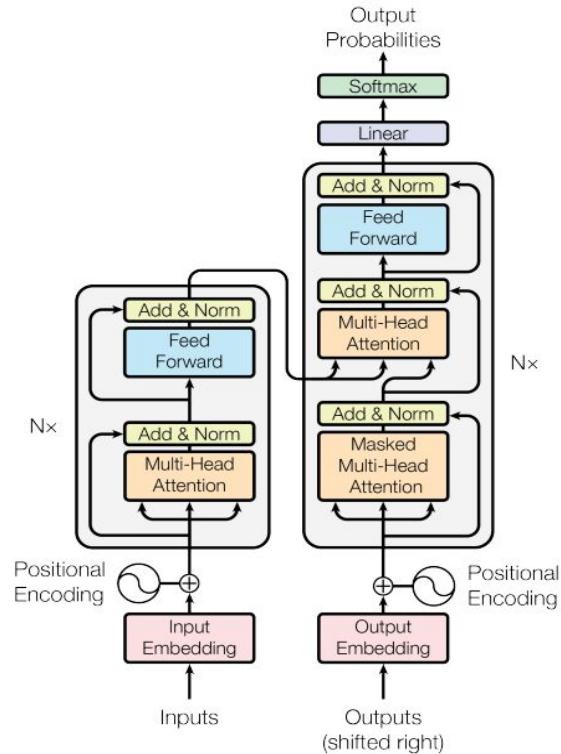
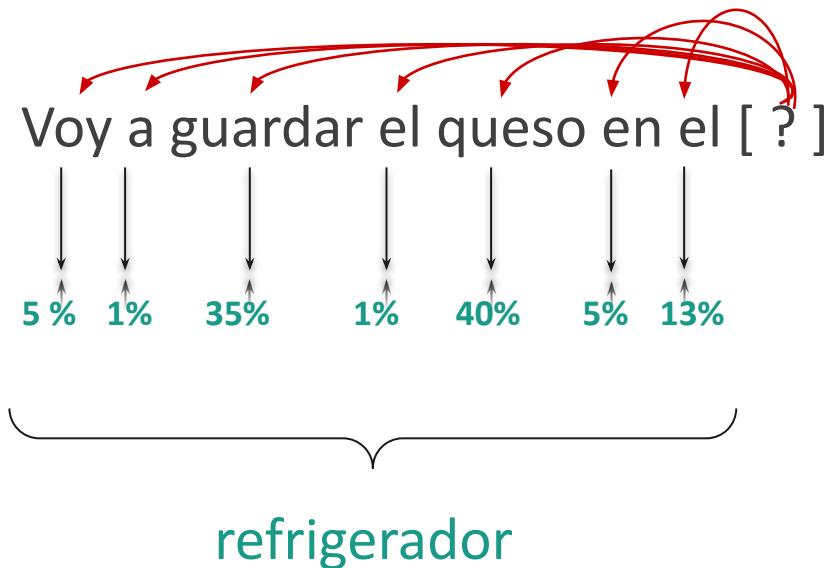
Voy a guardar el queso en el [?]



TRANSFORMERS



TRANSFORMERS



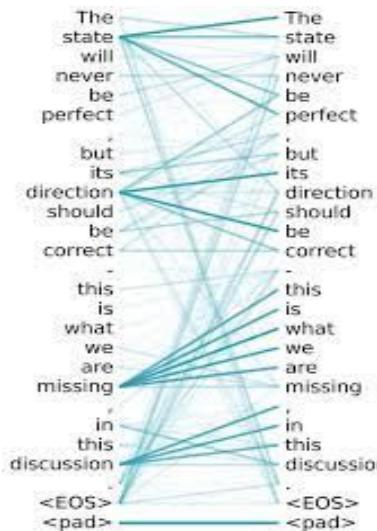
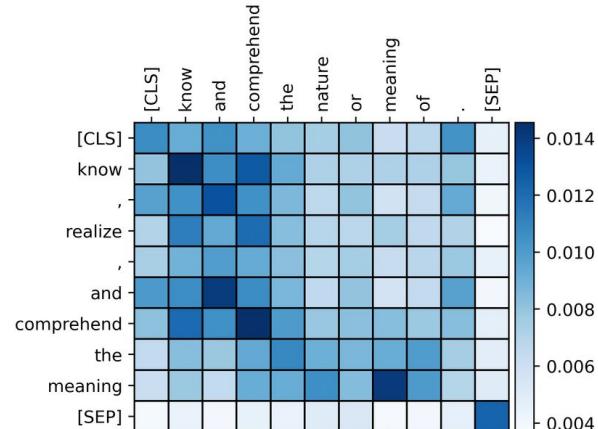
TRANSFORMERS



En lugar de mirar las palabras de una secuencia de forma aislada o en un contexto limitado, el **mecanismo de atención** permite que cada palabra "vea" y se relacione con todas las demás palabras en la secuencia, proporcionando una comprensión contextual más rica.



Cada token calcula su representación ponderando al resto de los tokens en la secuencia.



VENTAJAS SOBRE RNNs

- **RNNs no permiten interacción directa:**
 - Los token no tienen interacción directa, a medida que se alejan la relación se desvanece.
 - Fuerza interacción lineal.
- **Procesamiento en Paralelo:** A diferencia de las RNNs que procesan secuencias paso a paso, los Transformers pueden procesar todos los elementos de la secuencia simultáneamente, lo que los hace significativamente más rápidos y escalables.
- **Dependencias a Largo Plazo:** Las RNNs tienen problemas con dependencias a largo plazo debido al desvanecimiento del gradiente. Los Transformers, con su mecanismo de atención, pueden capturar mejor estas dependencias.
- **Arquitectura más flexible:** Los transformers pueden usar diferentes tipos de atención, diferentes tipos de capas, y son más adaptables a diferentes tareas o dominios.



POSITIONAL EMBEDDINGS

CODIFICACION POSICIONAL EN TRANSFORMERS

- **Ausencia de Orden:** A diferencia de las RNNs los Transformers procesan la entrada en paralelo. Esto significa que, por defecto, no tienen una noción del orden de los datos.
- **Importancia del Orden:** En muchas tareas de NLP, el orden de las palabras es esencial para entender el significado. Por ejemplo, "Juan golpeó a Pedro" tiene un significado diferente de "Pedro golpeó a Juan".



Positional Embeddings:

- **Idea:** Proporcionar a los Transformers información sobre la posición de cada palabra en una secuencia.
- **Suma a los Embeddings:** Estas codificaciones se suman directamente a los embeddings de palabras antes de pasar por el modelo.

CODIFICACION POSICIONAL EN TRANSFORMERS



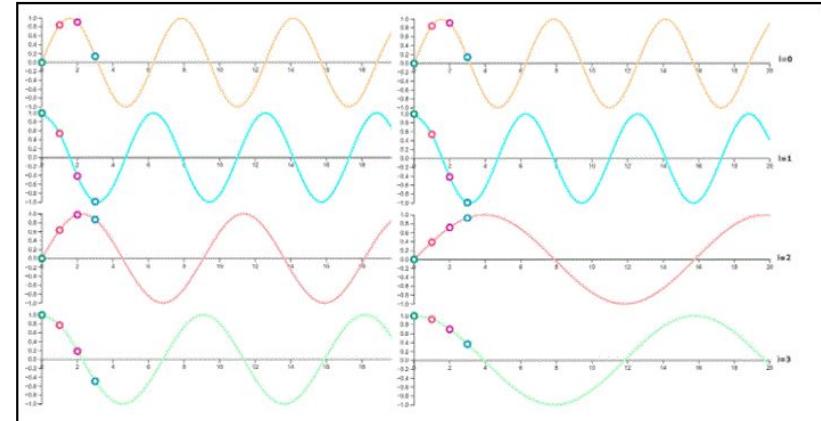
Solución: Para la posición p y la dimensión i del embedding, las codificaciones se calculan como:

$$\text{PE}_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d}}\right)$$

$$\text{PE}_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d}}\right)$$

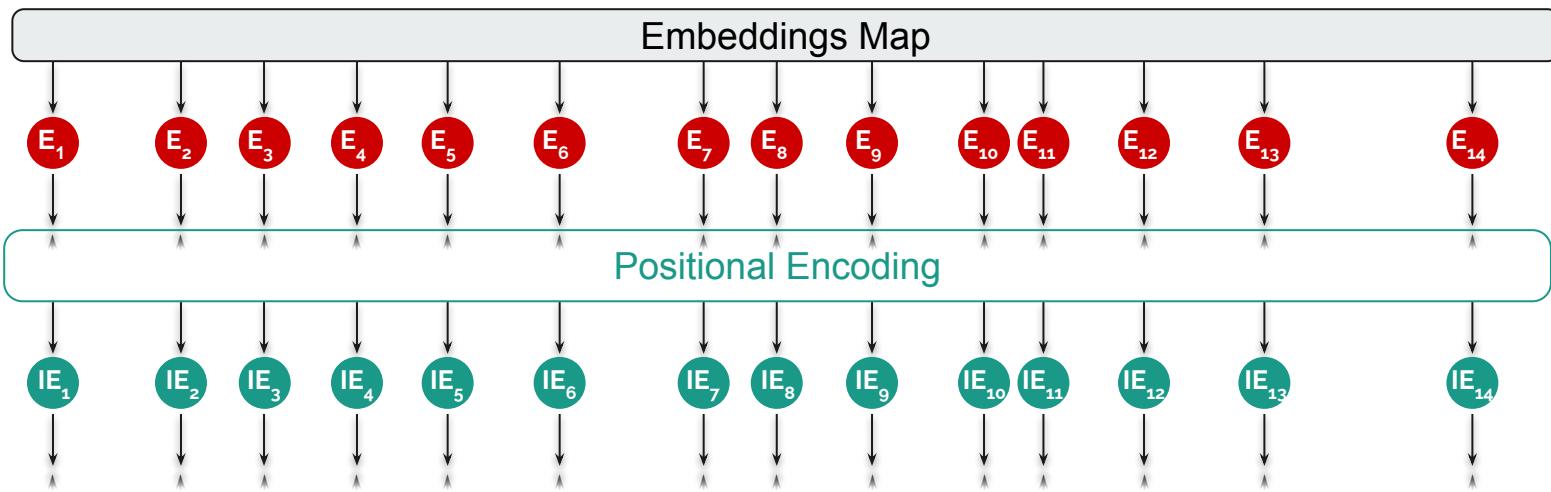
Ventajas

- Desambiguación:** Proporciona claridad donde el orden de las palabras es crucial para el significado.
- Escala a Secuencias de Diferentes Longitudes:** A diferencia de los embeddings aprendidos que requieren una longitud fija, las codificaciones sinusoidales pueden calcularse para secuencias de cualquier longitud.



POSITIONAL ENCODING

el ganó y sumó 3 puntos que lo dejan en la punta del campeonato

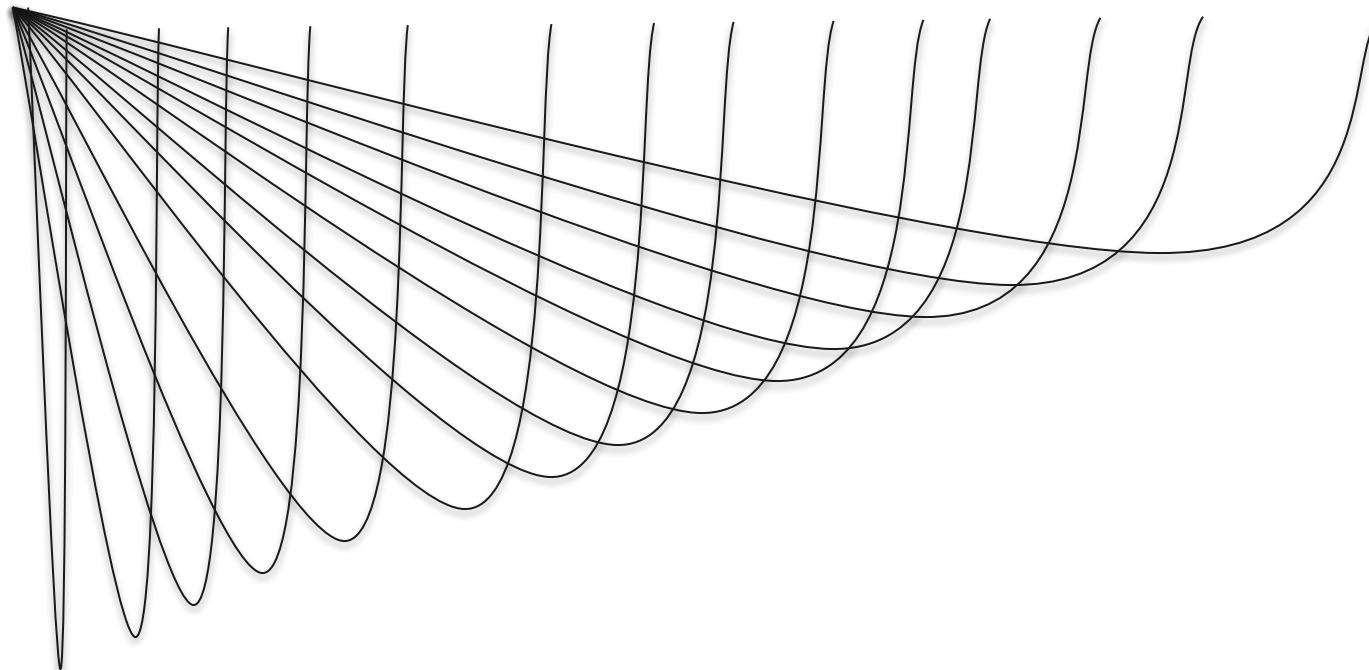




SELF-ATTENTION

SELF-ATTENTION

River ganó y sumó 3 puntos que lo dejan en la punta del campeonato

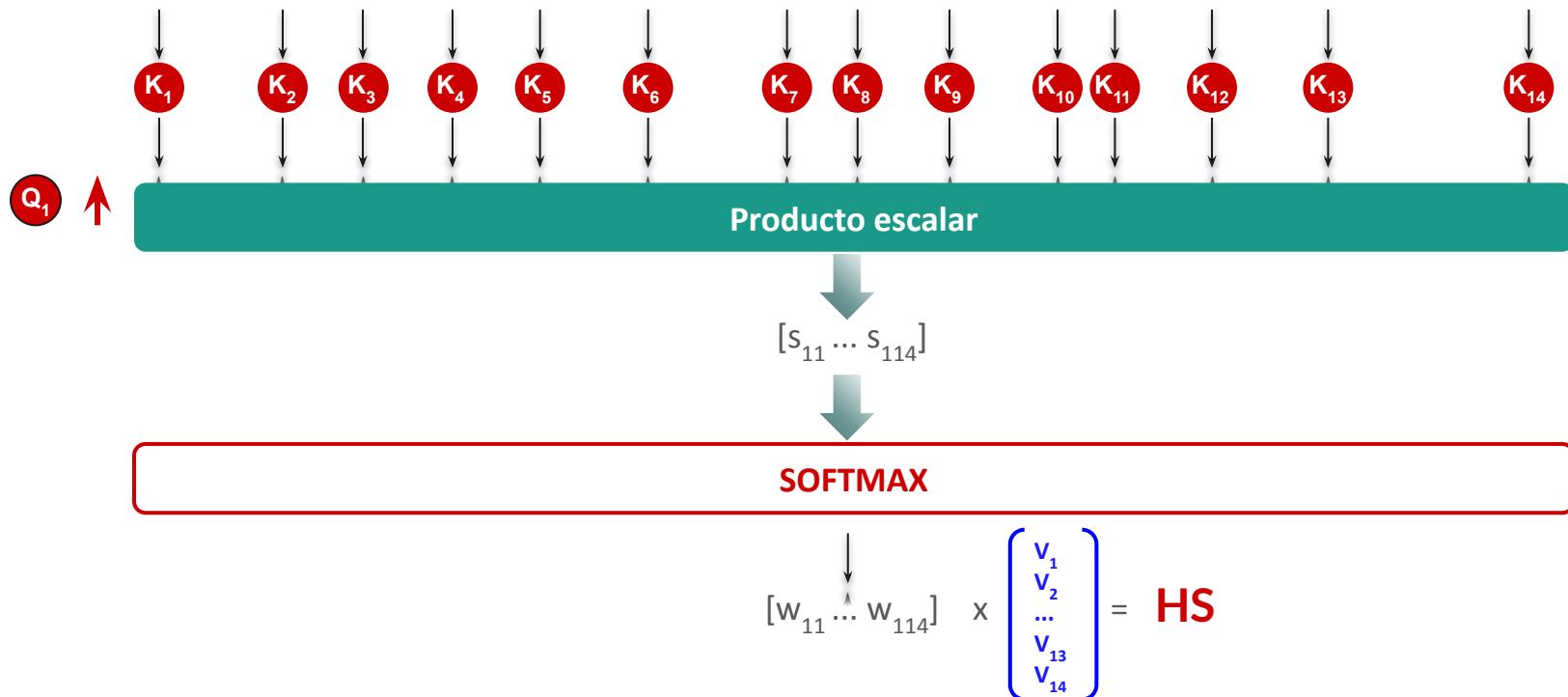


SELF-ATTENTION

- Permite al modelo centrarse en diferentes partes de la entrada simultáneamente.
- Q, K, V:
 - Queries (Q), Keys (K) y Values (V) son derivados de los embeddings de entrada.
 - En el caso de la auto-atención (self-attention), Q, K, y V provienen de la misma fuente, es decir, los embeddings de entrada.
- Puntuación de Atención:
$$\text{score} = \frac{QK^T}{\sqrt{d_k}}$$
- Softmax y Ponderación:
 - Los scores se pasan por una función softmax para obtener los pesos de atención.
 - Estos pesos determinan cuánta atención se debe prestar a cada palabra en la secuencia.
 - La salida de la atención se obtiene multiplicando los pesos por los Values (V).
- Representación del Token:
 - La representación de cada token es una suma ponderada de los Values (V) de cada palabra en la secuencia, donde los pesos provienen de la etapa de softmax.

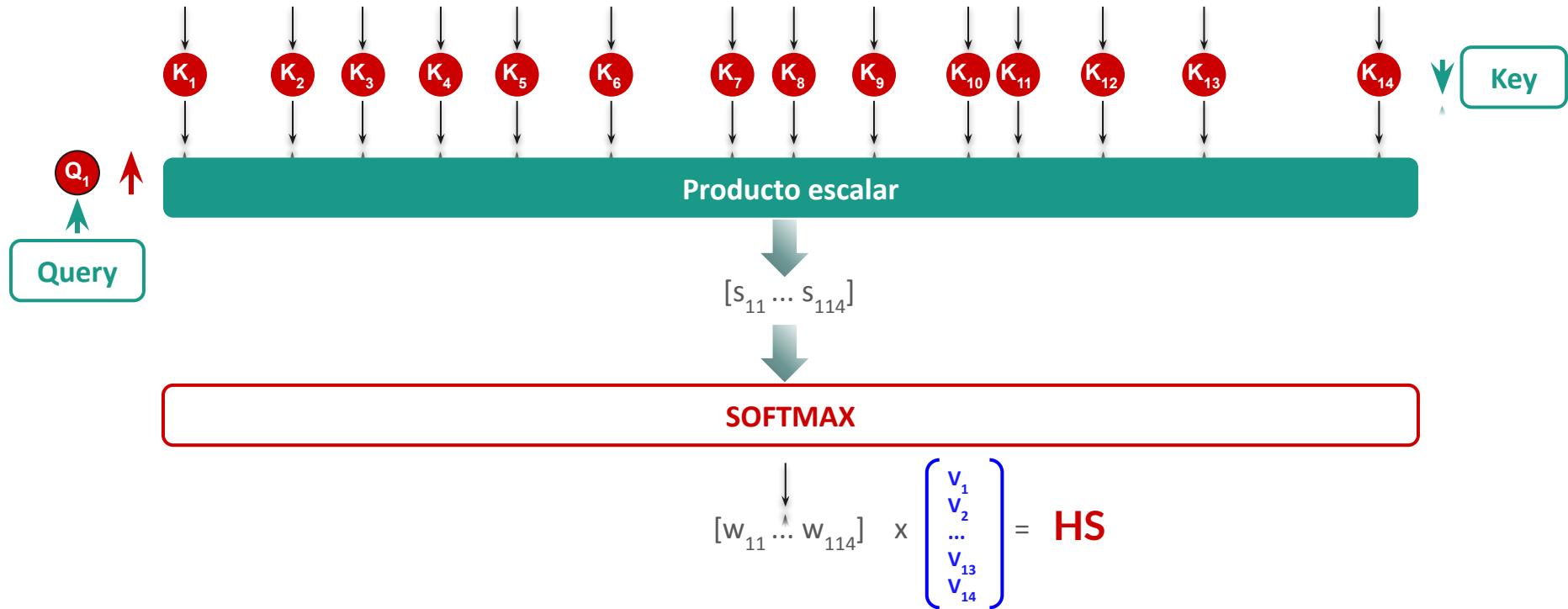
SELF-ATTENTION: QUERIES, KEYS Y VALUES

River ganó y sumó 3 puntos que lo dejan en la punta del campeonato

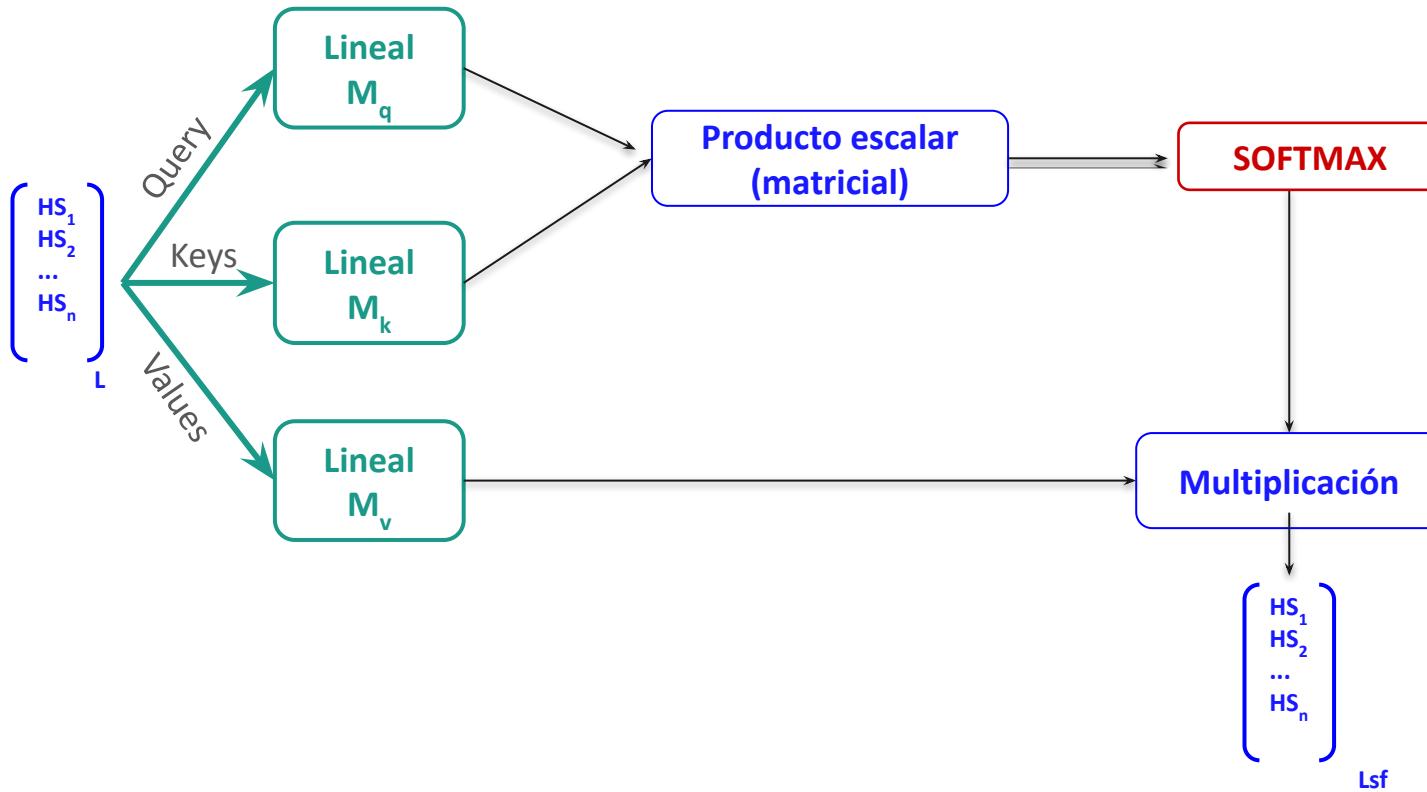


SELF-ATTENTION: QUERIES, KEYS Y VALUES

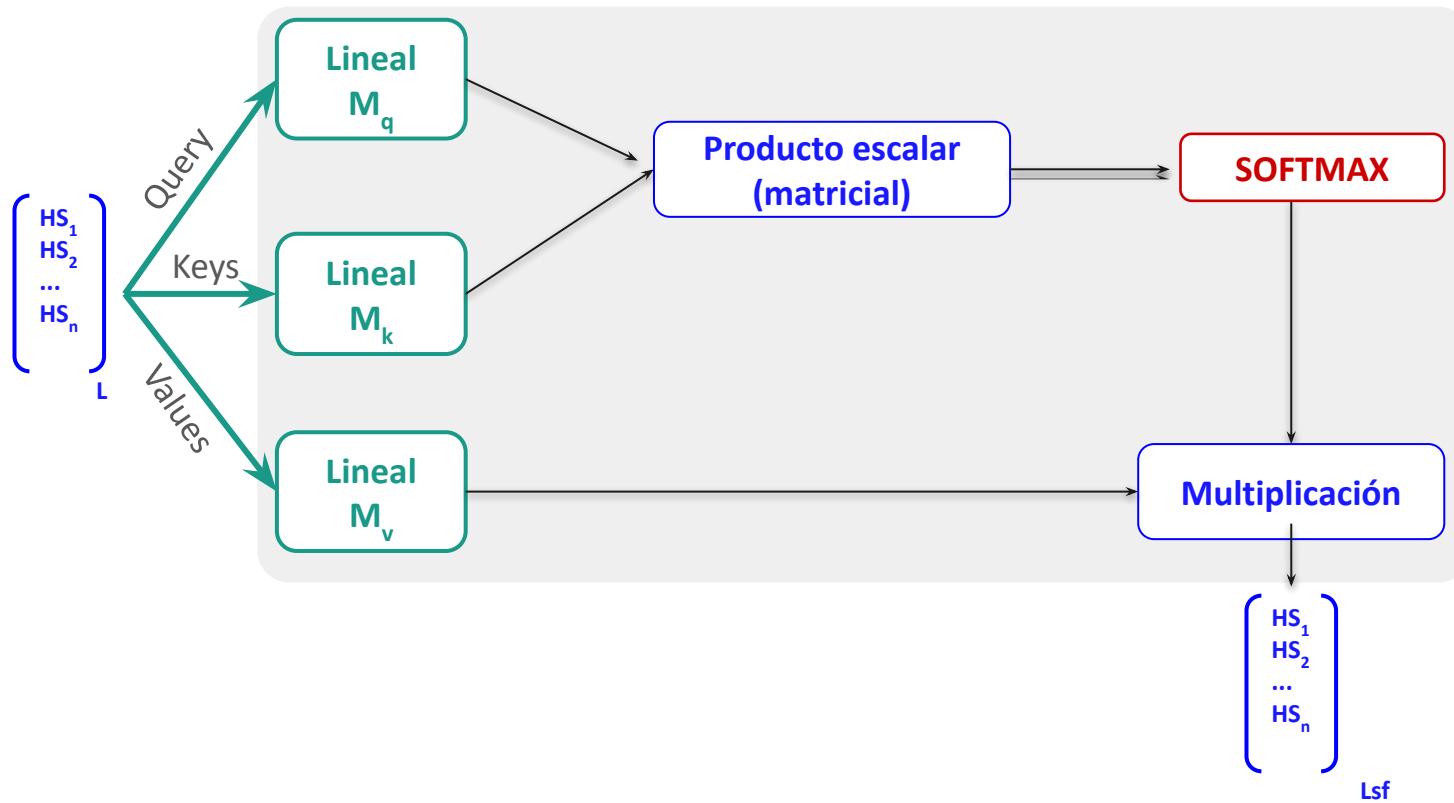
River ganó y sumó 3 puntos que lo dejan en la punta del campeonato



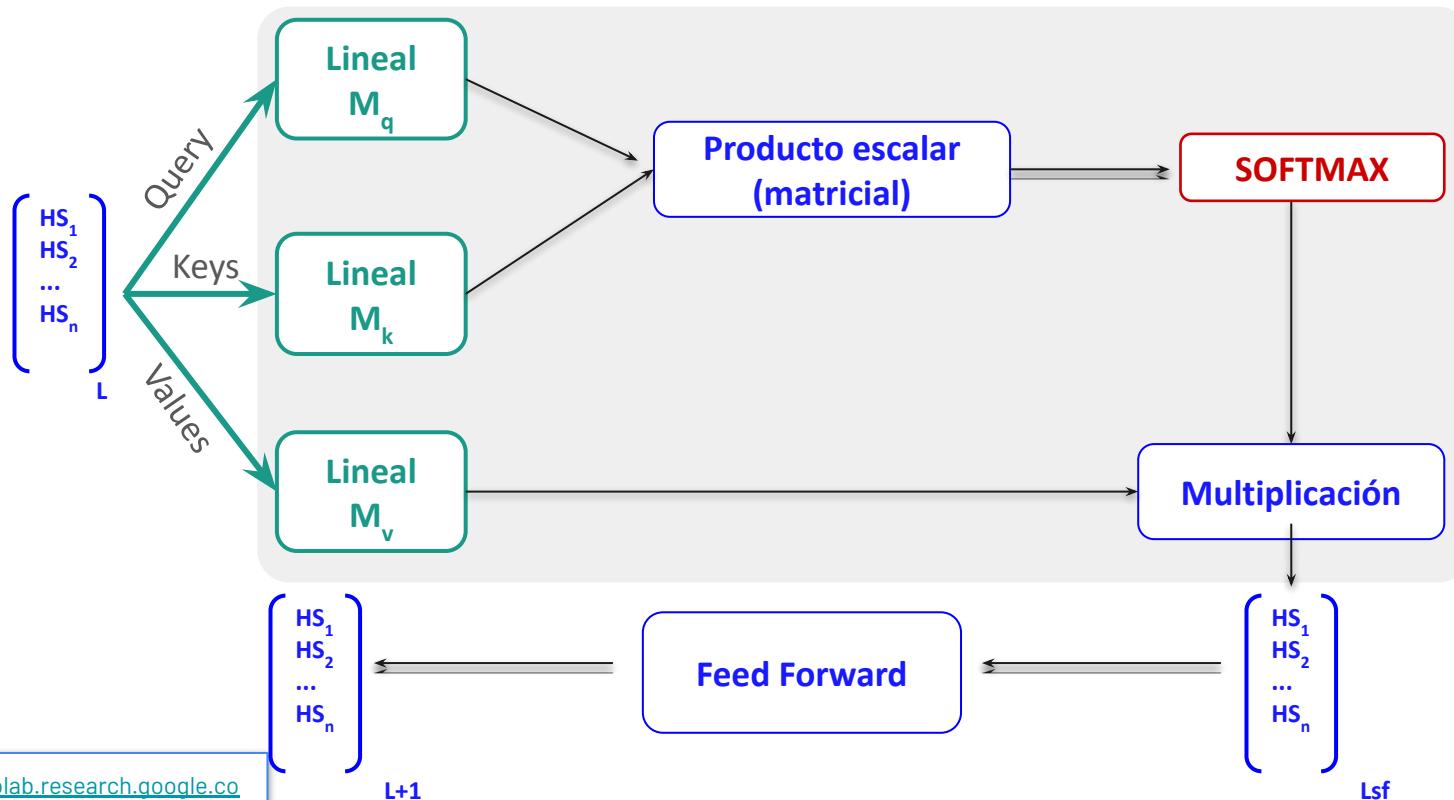
SELF-ATTENTION



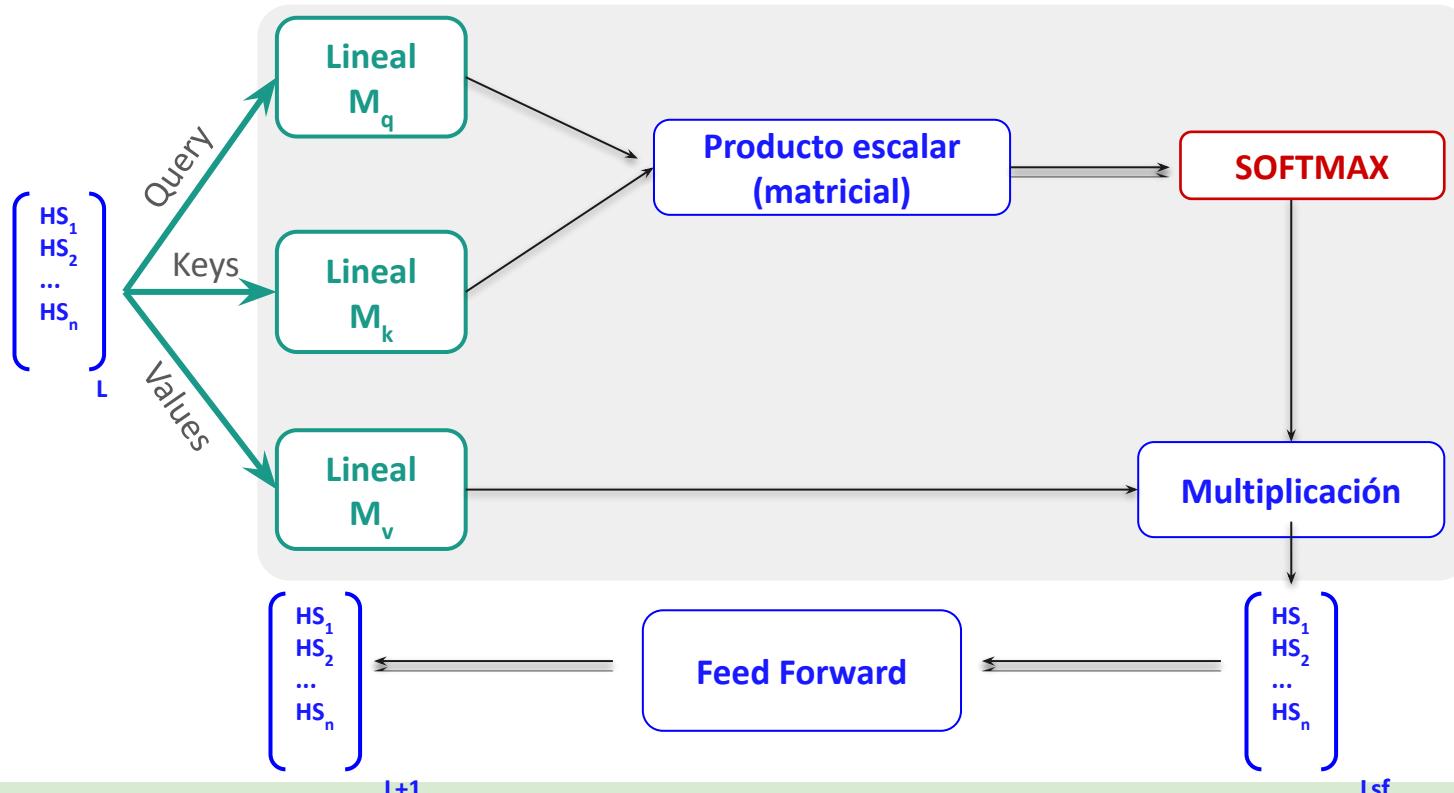
SELF-ATTENTION



SELF-ATTENTION

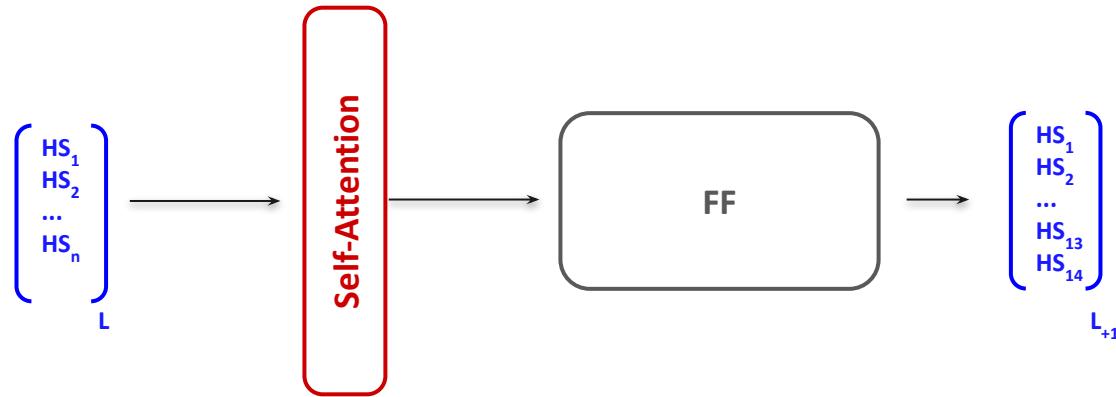


SELF-ATTENTION

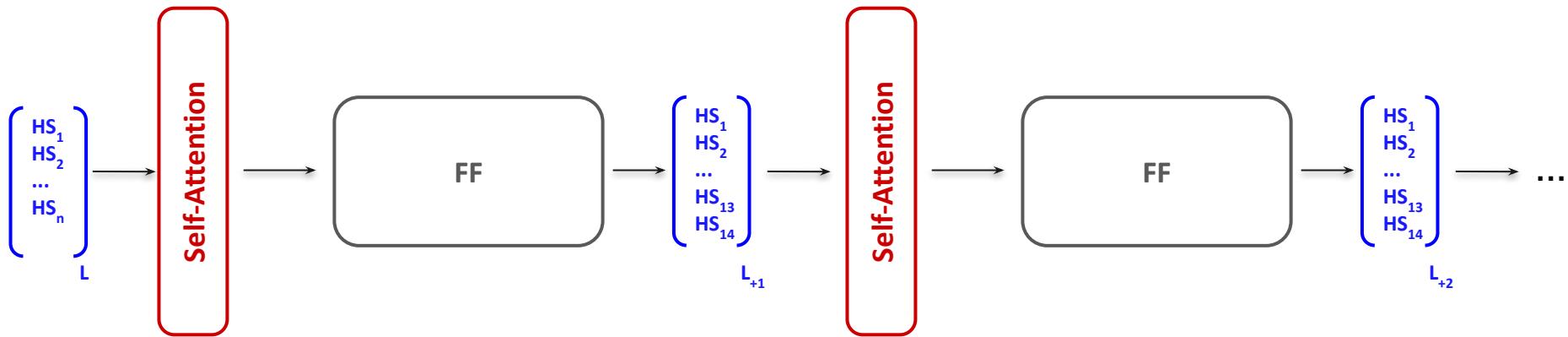


! ¿Por que necesitamos la Feed Forward?

SELF-ATTENTION



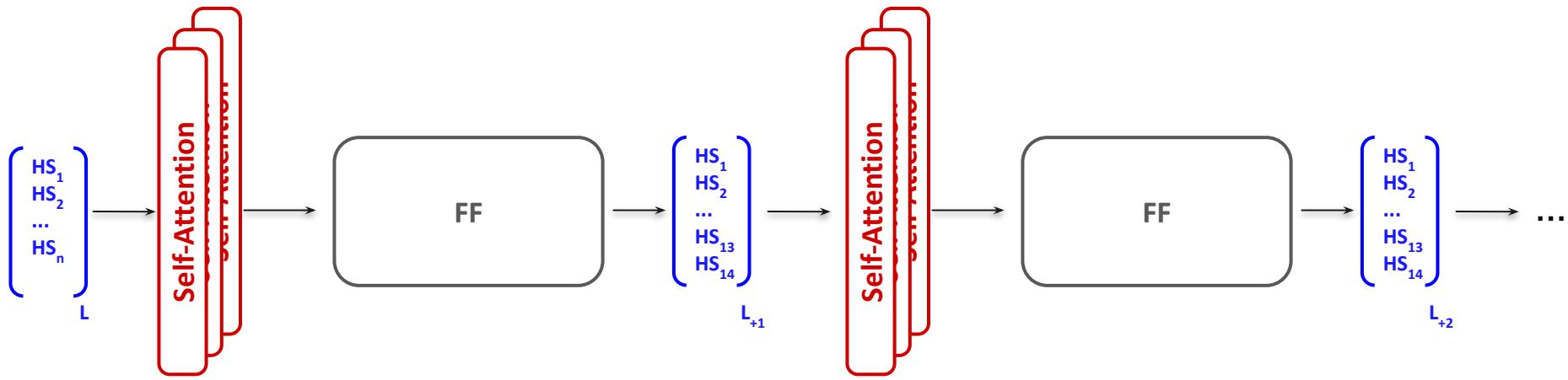
SELF-ATTENTION



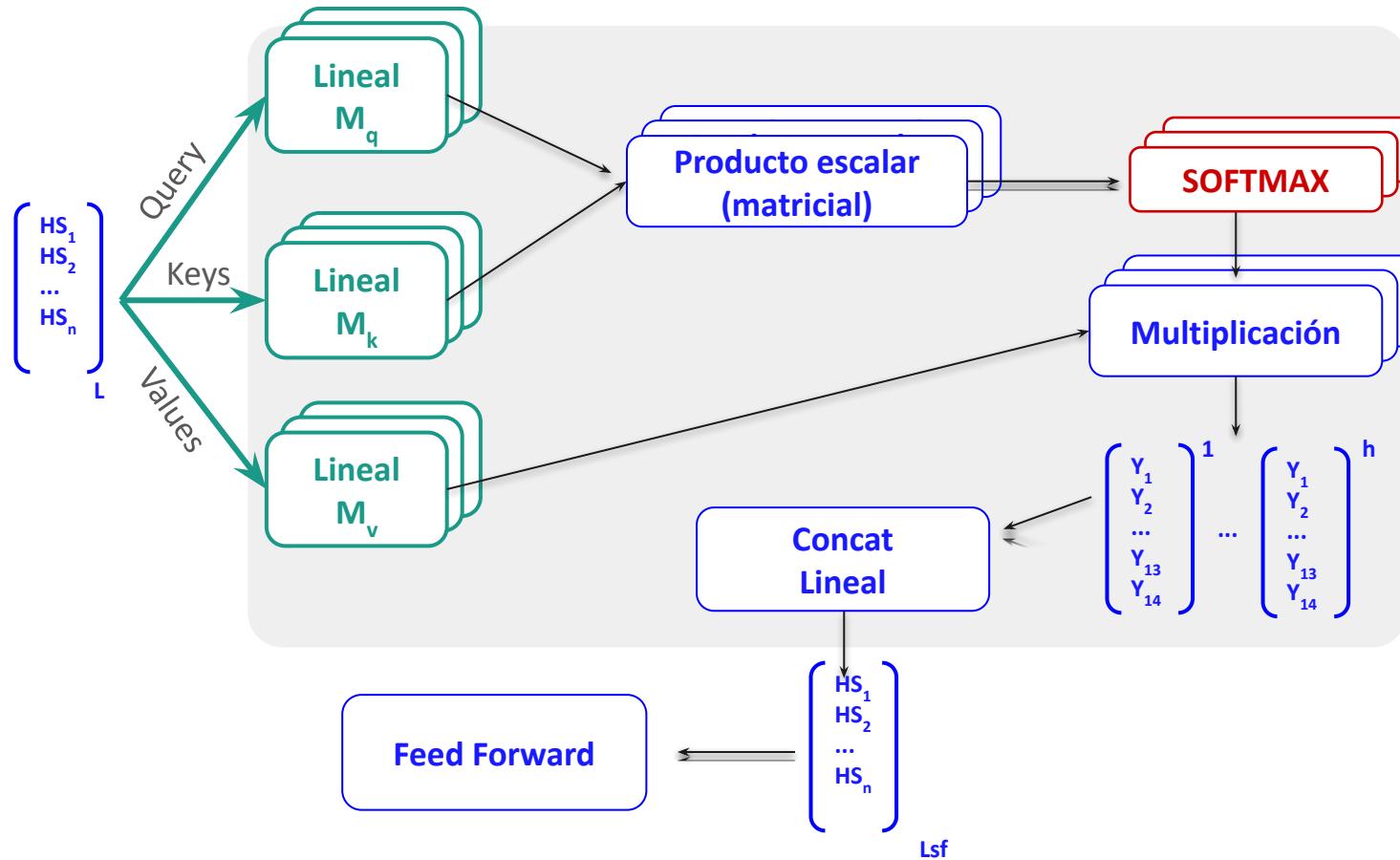
Apilando bloques de self-attention podemos lograr extraer más información sobre las relaciones entre palabras

Pero así y todo nos puede seguir faltando atención

MULTIHEADED SELF-ATTENTION

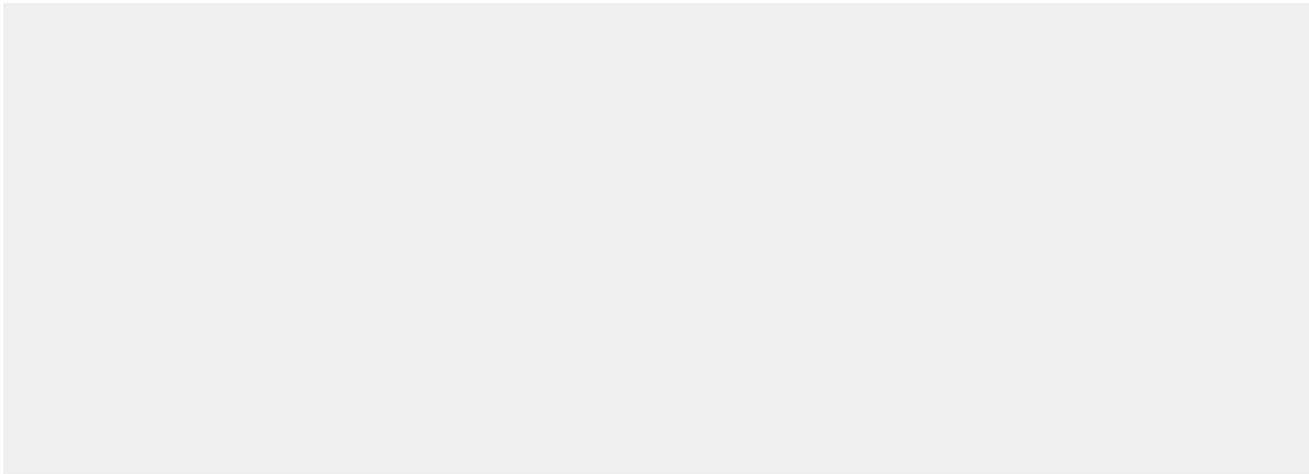


MULTIHEADED SELF-ATTENTION



SELF-ATTENTION STEP BY STEP

Self-attention



input #1

1	0	1	0
---	---	---	---

input #2

0	2	0	2
---	---	---	---

input #3

1	1	1	1
---	---	---	---

<https://colab.research.google.com/drive/1rPk3ohrmVclqhh7u07qys4oznDdAhpzF>

SELF-ATTENTION CODE

```
def forward(
    self,
    hidden_states: torch.Tensor,
    attention_mask: Optional[torch.FloatTensor] = None,
    head_mask: Optional[torch.FloatTensor] = None,
    encoder_hidden_states: Optional[torch.FloatTensor] = None,
    encoder_attention_mask: Optional[torch.FloatTensor] = None,
    past_key_value: Optional[Tuple[Tuple[torch.FloatTensor]]] = None,
    output_attentions: Optional[bool] = False,
) -> Tuple[torch.Tensor]:
    mixed_query_layer = self.query(hidden_states)

    # If this is instantiated as a cross-attention module, the keys
    # and values come from an encoder; the attention mask needs to be
    # such that the encoder's padding tokens are not attended to.
    is_cross_attention = encoder_hidden_states is not None

    if is_cross_attention and past_key_value is not None:
        # reuse k,v, cross_attentions
        key_layer = past_key_value[0]
        value_layer = past_key_value[1]
        attention_mask = encoder_attention_mask
    elif is_cross_attention:
        key_layer = self.transpose_for_scores(self.key(encoder_hidden_states))
        value_layer = self.transpose_for_scores(self.value(encoder_hidden_states))
        attention_mask = encoder_attention_mask
    elif past_key_value is not None:
        key_layer = self.transpose_for_scores(self.key(hidden_states))
        value_layer = self.transpose_for_scores(self.value(hidden_states))
        key_layer = torch.cat([past_key_value[0], key_layer], dim=2)
        value_layer = torch.cat([past_key_value[1], value_layer], dim=2)
    else:
        key_layer = self.transpose_for_scores(self.key(hidden_states))
        value_layer = self.transpose_for_scores(self.value(hidden_states))

    query_layer = self.transpose_for_scores(mixed_query_layer)
```

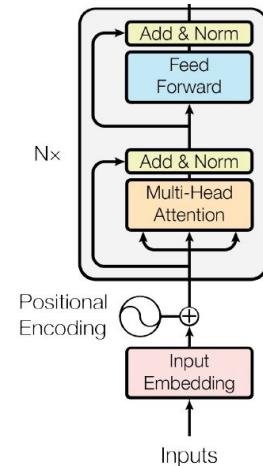
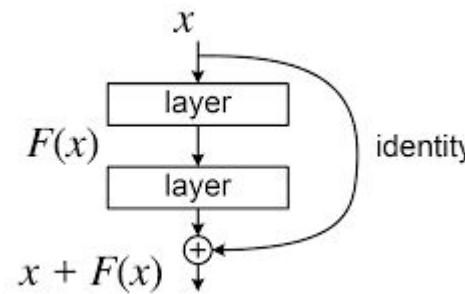
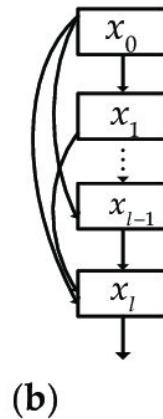
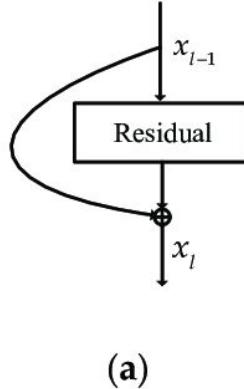
https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py



RESIDUAL CONNECTIONS

CONEXIONES RESIDUALES

- Conexiones que "saltan" una o más capas, sumando la entrada directamente a la salida.
- Permiten que la información fluya directamente desde una capa anterior a una posterior, sin pasar por las capas intermedias.
- Permiten que el gradiente de la función de pérdida se propague directamente hacia la entrada a través de las conexiones de salto, evitando así el problema del desvanecimiento del gradiente.



IMPLICANCIAS PRÁCTICAS

- **Entrenamiento Estable:** Reducen el desvanecimiento del gradiente, aceleran la convergencia.
- **Preservación de Información:** Aseguran que la información vital no se pierda entre capas.
- **Flexibilidad:** Permiten que el modelo se ajuste finamente a la información contextual sin redefinir completamente un token.
- **Actualización Incremental:** Permiten ajustes aditivos a los estados ocultos, conservando información original y agregando nueva.
- **Refinamiento a través de Capas:** Cada capa refina el significado de un token basado en el contexto.
 - "hombre" en "hombre joven" vs. "hombre araña". Las conexiones residuales ayudan a mantener la esencia original mientras se ajusta el contexto.



LAYER NORMALIZATION

LAYER NORMALIZATION (NORMALIZACIÓN DE CAPA)

- Técnica de regularización diseñada para acelerar el entrenamiento de modelos neuronales profundos.
- Reducir la variación no informativa en los valores del vector oculto normalizando dentro de cada capa.

1. Promedio de los valores del hidden vector.

$$\mu = \frac{1}{d} \sum_{j=1}^d x_j$$

2. Desviación Estándar.

$$\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (x_j - \mu)^2}$$

3. Parámetros de escala y sesgo: γ y β son parámetros entrenables.

$$x_{\text{norm}} = \frac{x - \mu}{\sigma + \epsilon} \times \gamma + \beta$$

LAYER NORMALIZATION (NORMALIZACIÓN DE CAPA)

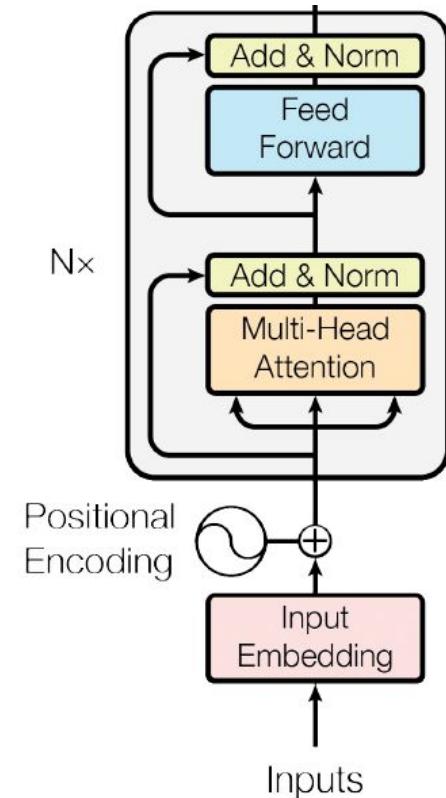
- **Aceleración del Entrenamiento:** Conduce a una convergencia más rápida al hacer que cada capa opere en un rango similar.
- **Estabilidad:** Puede reducir problemas como el desvanecimiento o explosión del gradiente.
- **Normalización de Gradientes:** El éxito de LayerNorm podría deberse a la normalización de los gradientes.



ENCODER

ESTRUCTURA DEL ENCODER

- **Componentes:** Cada bloque del encoder consta de dos subcapas principales:
 - Atención Multi-Head
 - Red Feed-Forward (FFN)
- **Estado Oculto (hidden state):**
 - Es la representación del embedding en la capa i , la información acumulada de las entradas procesadas hasta el momento.
 - **Evolución:** En cada capa del encoder, el estado oculto se actualiza y refina a través de las operaciones de atención y la red FFN.
- **Normalización y Conexiones Residuales:**
 - Despues de cada subcapa (atención y FFN), se aplica una conexión residual seguida de una normalización de capa.



CONFIGURANDO EL ENCODER

- **Número de Capas (Num. Layers):** Define la profundidad del modelo. BERT base tiene 12, mientras que BERT large tiene 24.
- **Dimensiones Ocultas (Hidden Dimensions):** El tamaño de los embeddings y de las representaciones intermedias. En BERT base es 768 y en BERT large es 1024.
- **Longitud de Secuencia (Sequence Length):** Define la máxima longitud de tokens que el modelo puede procesar en una única entrada. Bert es 512.
- **Otros Parámetros Importantes:**
 - **Num. Attention Heads:** Define cuántas "cabezas" se utilizan en la atención multi-cabeza.
 - **Intermediate Size:** Tamaño de la capa oculta en la red feed-forward.
 - **Activation:** Función de activación utilizada (por ejemplo, gelu).

```
{  
    "architectures": [  
        "BertForMaskedLM"  
    ],  
    "attention_probs_dropout_prob": 0.1,  
    "gradient_checkpointing": false,  
    "hidden_act": "gelu",  
    "hidden_dropout_prob": 0.1,  
    "hidden_size": 768,  
    "initializer_range": 0.02,  
    "intermediate_size": 3072,  
    "layer_norm_eps": 1e-12,  
    "max_position_embeddings": 512,  
    "model_type": "bert",  
    "num_attention_heads": 12,  
    "num_hidden_layers": 12,  
    "pad_token_id": 0,  
    "position_embedding_type": "absolute",  
    "transformers_version": "4.6.0.dev0",  
    "type_vocab_size": 2,  
    "use_cache": true,  
    "vocab_size": 30522  
}
```

REPRESENTANDO EL INPUT

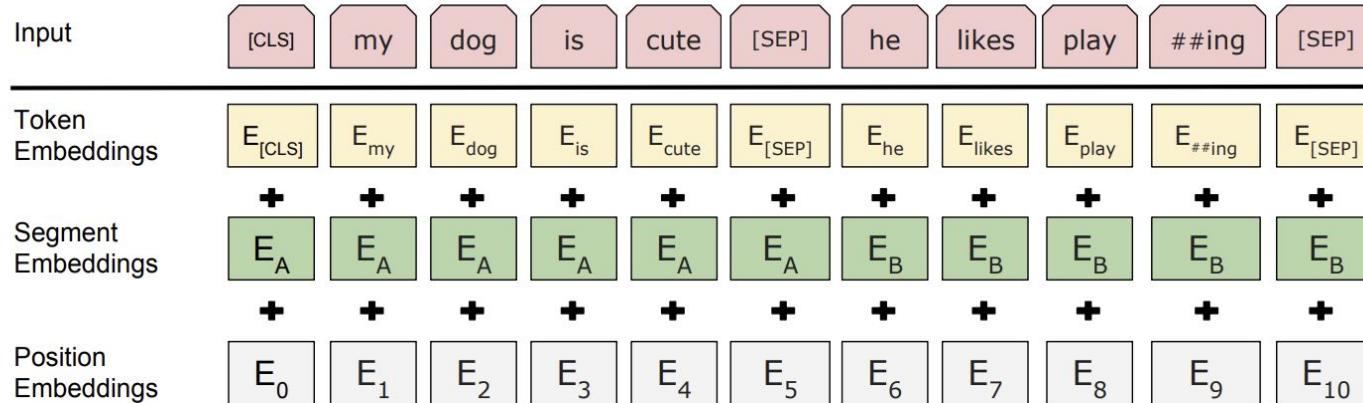


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

PRETRAINING + FINE TUNING

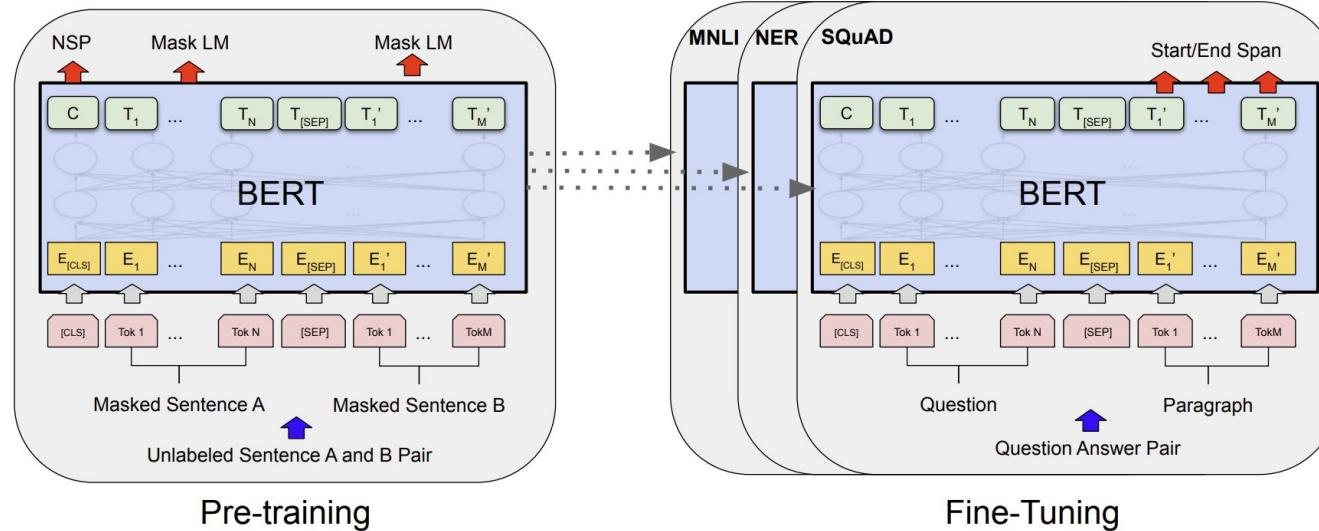
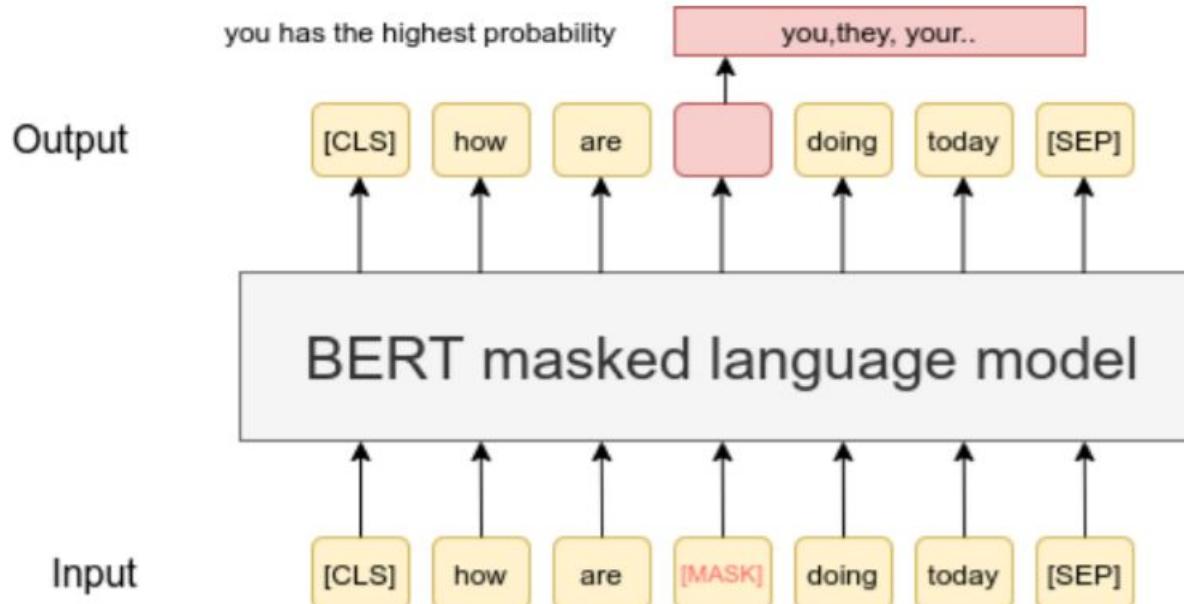
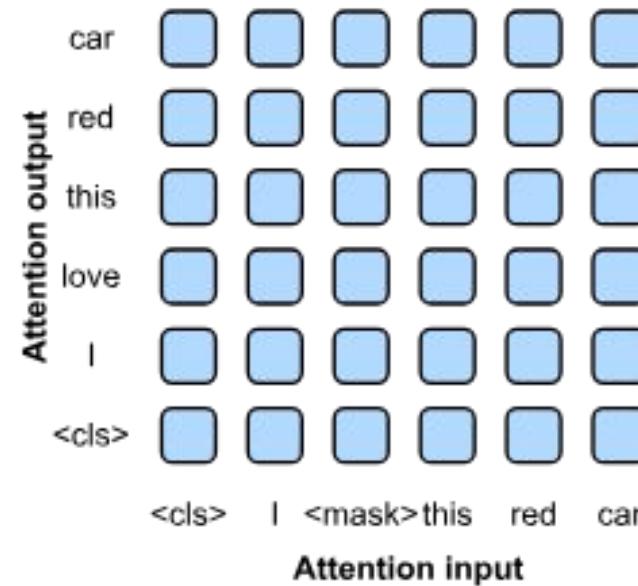
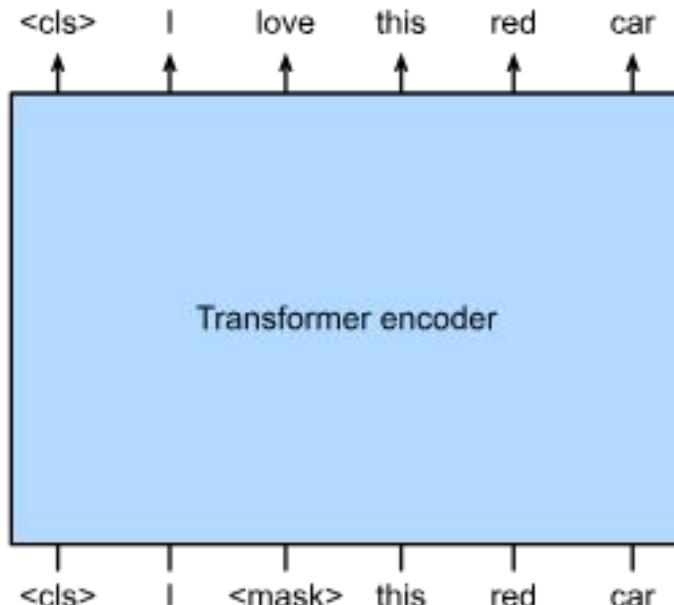


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. $[CLS]$ is a special symbol added in front of every input example, and $[SEP]$ is a special separator token (e.g. separating questions/answers).

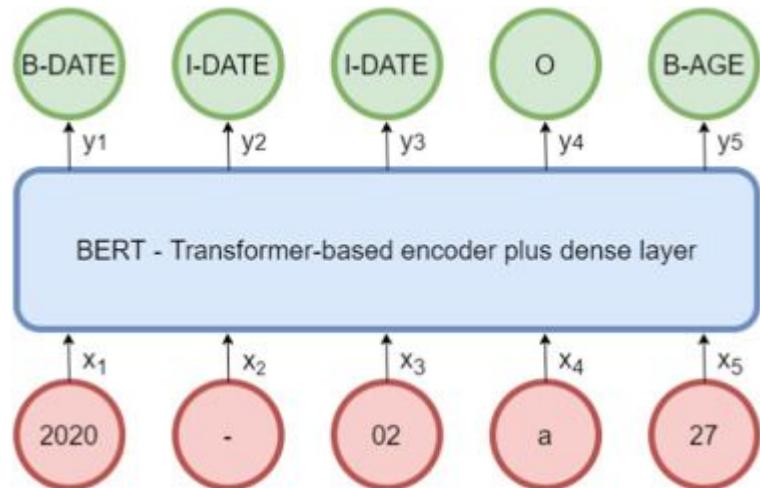
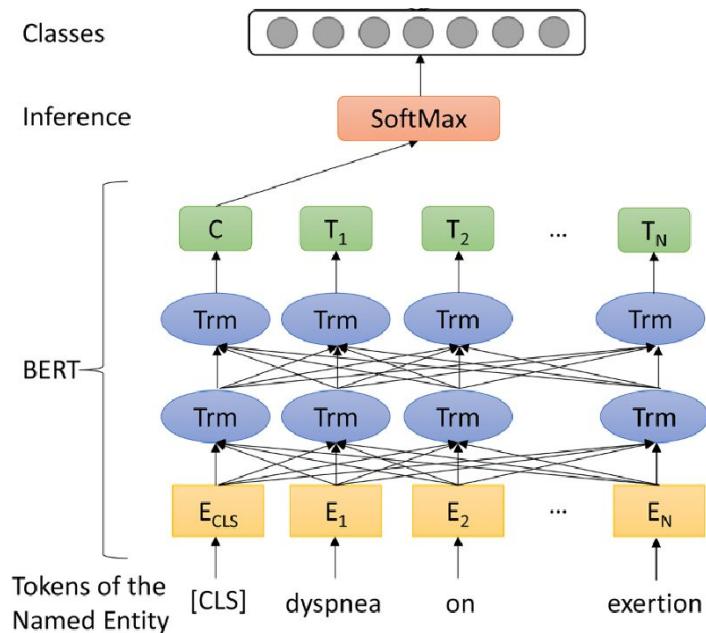
MASKED LANGUAGE PRETRAINING



MASKED LANGUAGE PRETRAINING



IDEALES PARA TAREAS DE CLASIFICACIÓN DE SECUENCIAS O TOKENS



BERT RESULTADOS GLUE

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

Table 1: GLUE Test results, scored by the evaluation server (<https://gluebenchmark.com/leaderboard>). The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.⁸ BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

BERT RESULTADOS SQUAD

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT _{BASE} (Single)	80.8	88.5	-	-
BERT _{LARGE} (Single)	84.1	90.9	-	-
BERT _{LARGE} (Ensemble)	85.8	91.8	-	-
BERT _{LARGE} (Sgl.+TriviaQA)	84.2	91.1	85.1	91.8
BERT _{LARGE} (Ens.+TriviaQA)	86.2	92.2	87.4	93.2

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

BERT SWAG

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
BERT _{BASE}	81.6	-
BERT _{LARGE}	86.6	86.3
Human (expert) [†]	-	85.0
Human (5 annotations) [†]	-	88.0

Table 4: SWAG Dev and Test accuracies. [†]Human performance is measured with 100 samples, as reported in the SWAG paper.

BERT NER

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	93.1
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

BERT HYPERPARÁMETROS

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

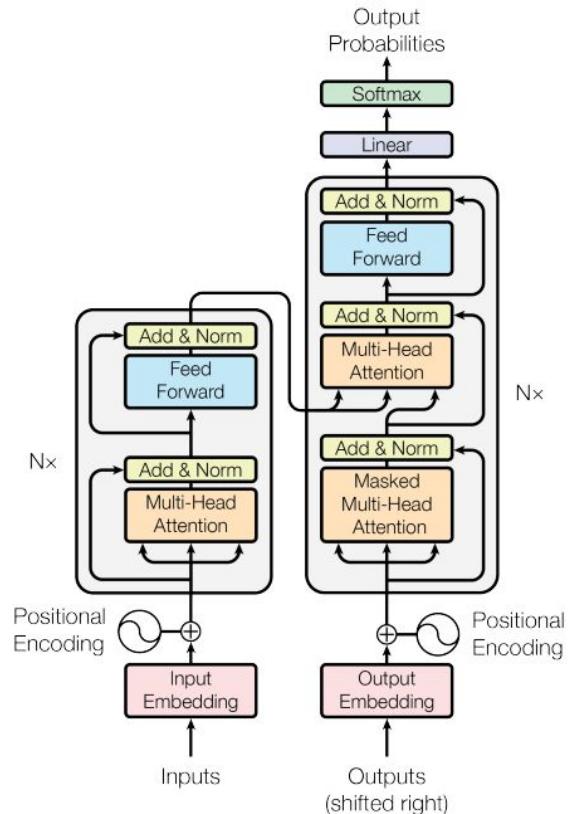
Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. “LM (ppl)” is the masked LM perplexity of held-out training data.



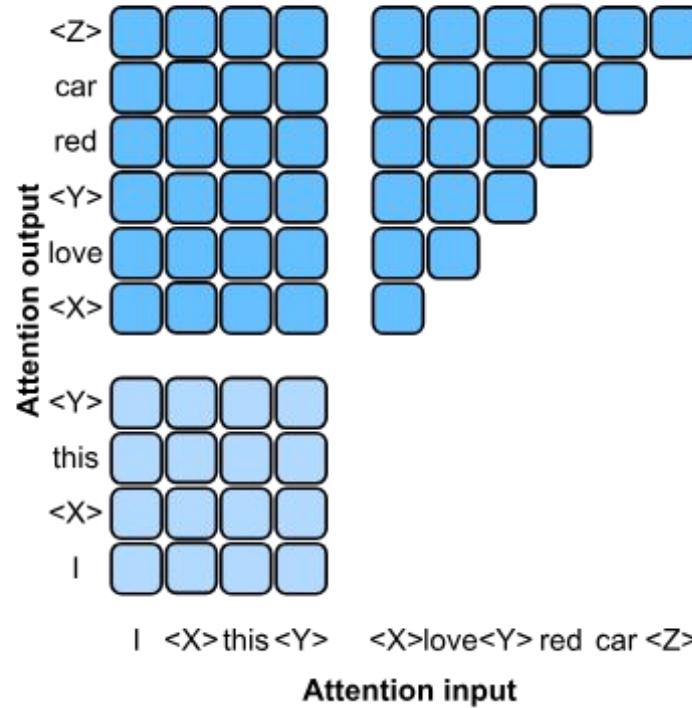
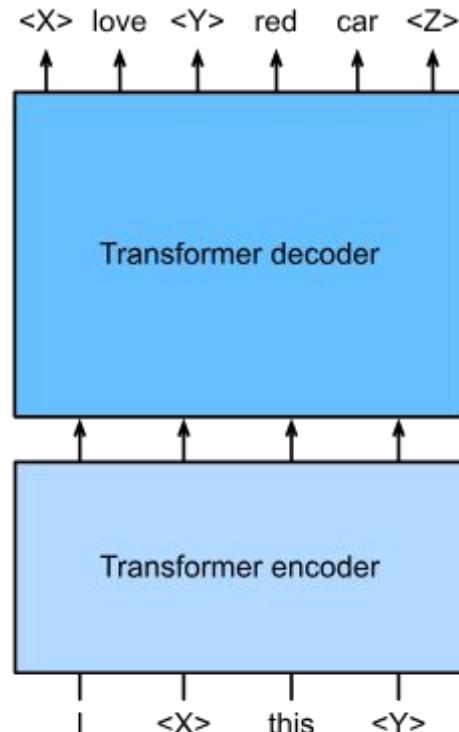
ENCODER-DECODER

TRANSFORMER: ENCODER-DECODER

- Un diseño de dos partes para tareas generativas donde el encoder procesa la entrada y el decoder genera la salida.
 - e.g., traducción automática, summarization, question answering, tareas generativas.
- El encoder crea una representación compacta de la entrada, que el decoder utiliza para producir la salida.
- **Cada capa del decoder tiene dos subcapas de atención:** una que se centra en la salida del decoder anterior y otra que se centra en la salida del encoder (atención encoder-decoder).

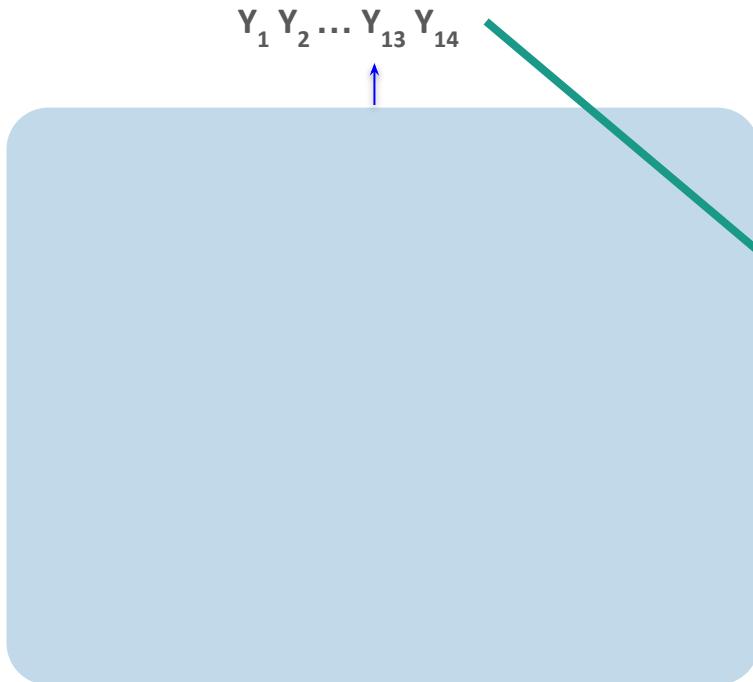


TRANSFORMER: ENCODER-DECODER

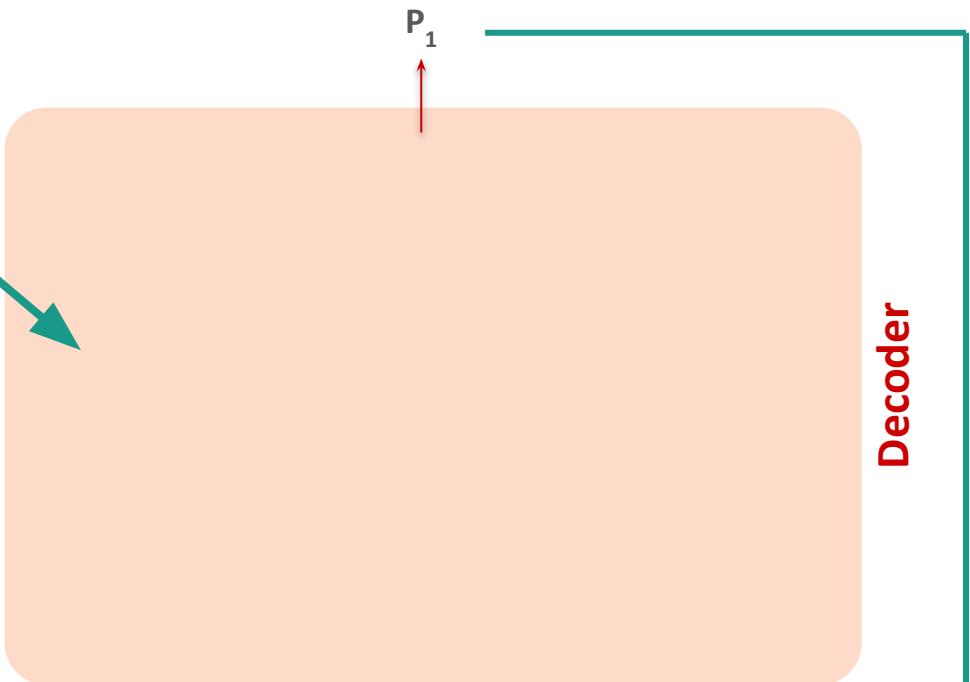


TRANSFORMER: TRADUCCIÓN

Encoder



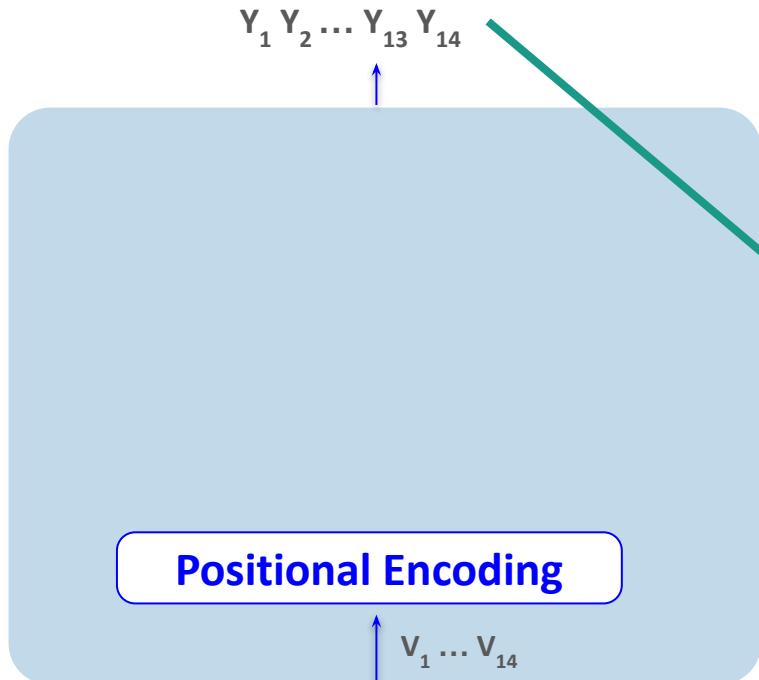
River ganó y sumó 3 puntos que lo
dejan en la punta del campeonato



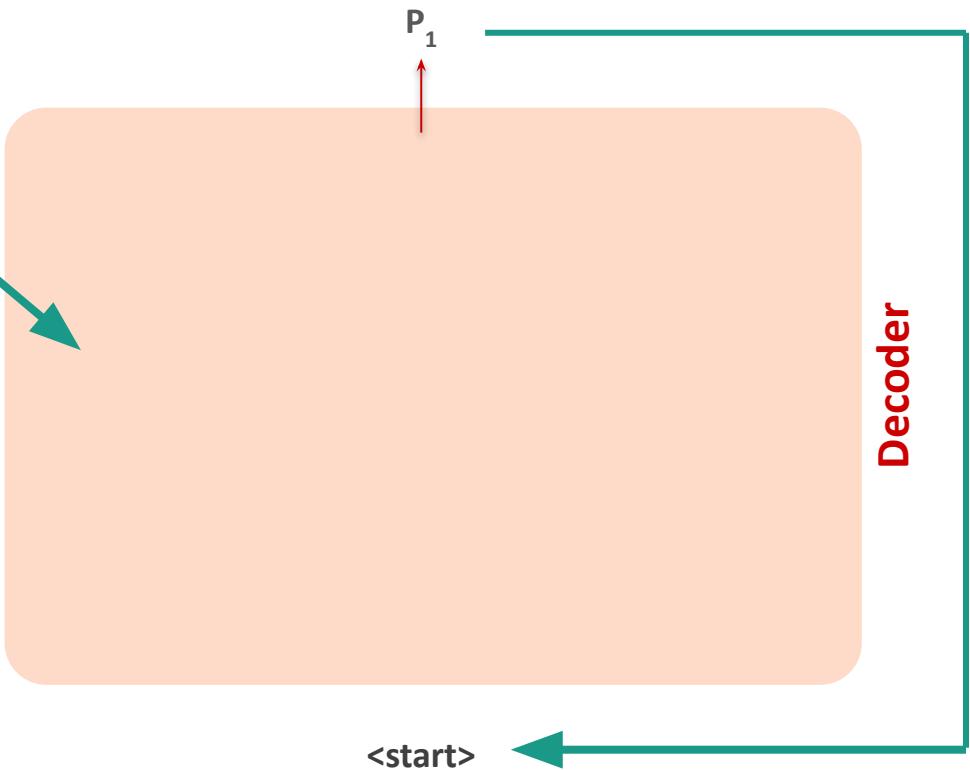
<start>

TRANSFORMER: TRADUCCIÓN

Encoder



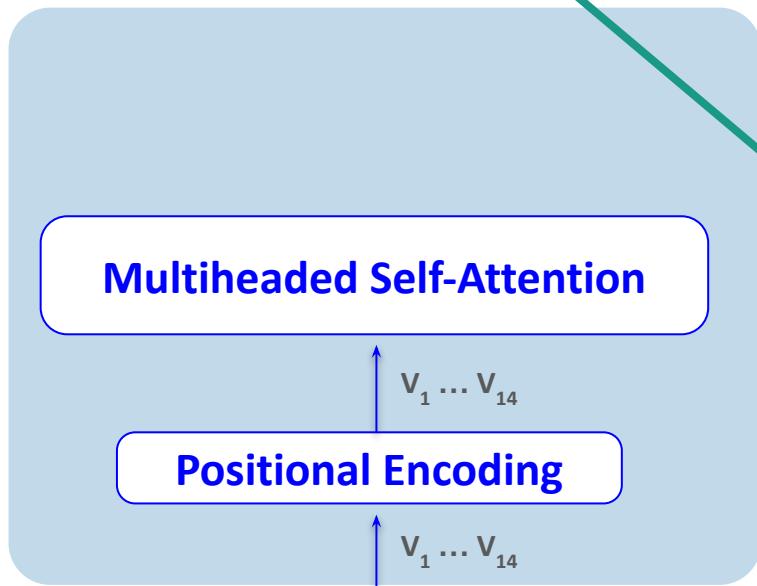
River ganó y sumó 3 puntos que lo
dejan en la punta del campeonato



Decoder

TRANSFORMER: TRADUCCIÓN

Encoder

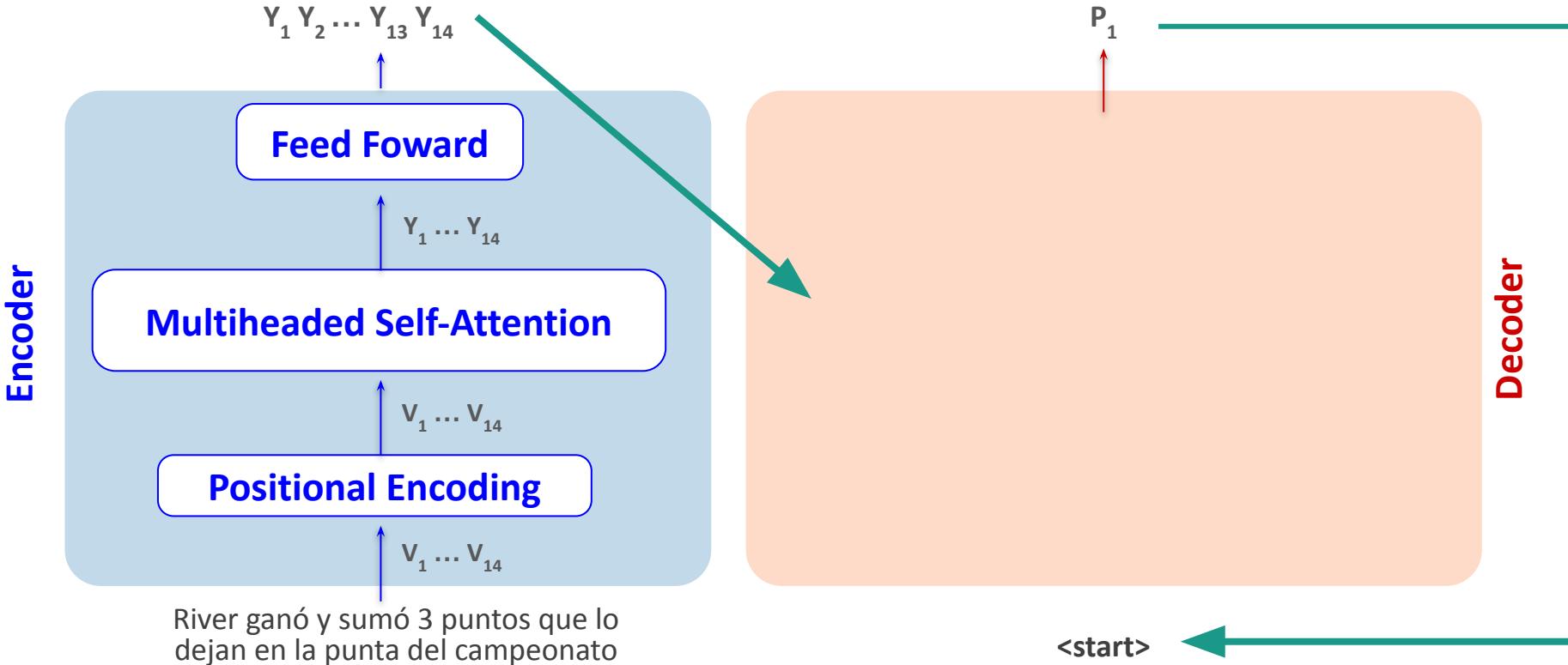


River ganó y sumó 3 puntos que lo
dejan en la punta del campeonato



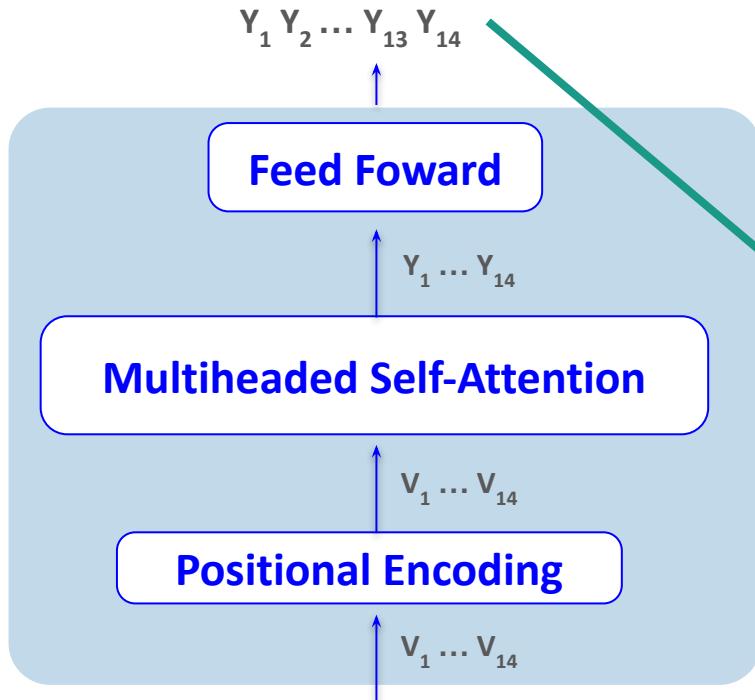
Decoder

TRANSFORMER: TRADUCCIÓN

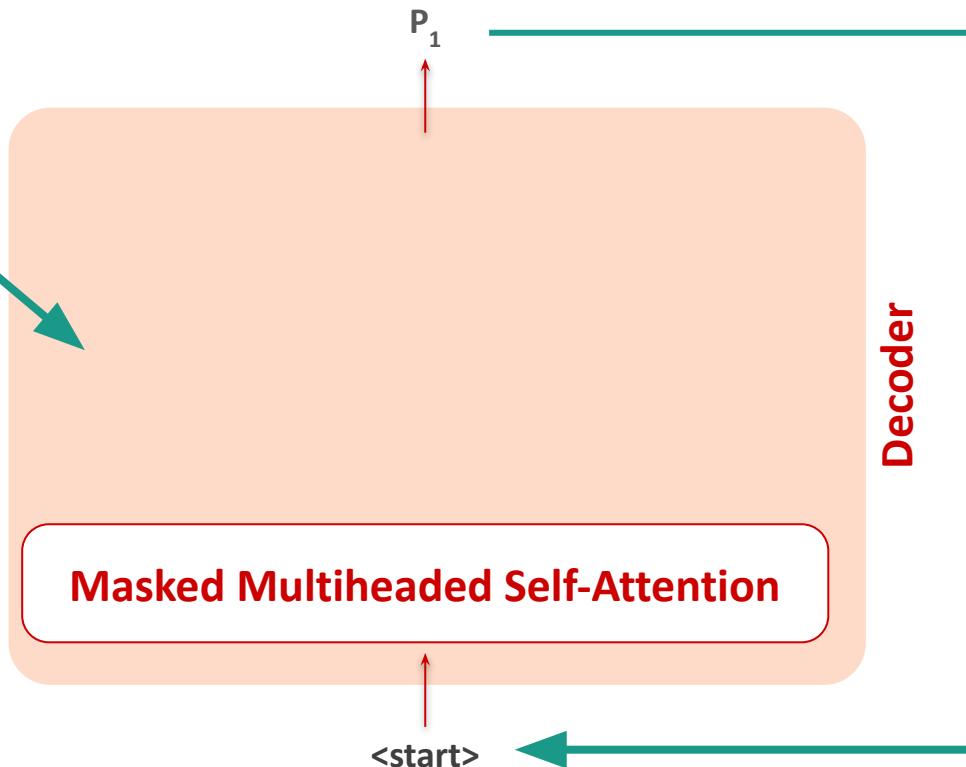


TRANSFORMER: TRADUCCIÓN

Encoder



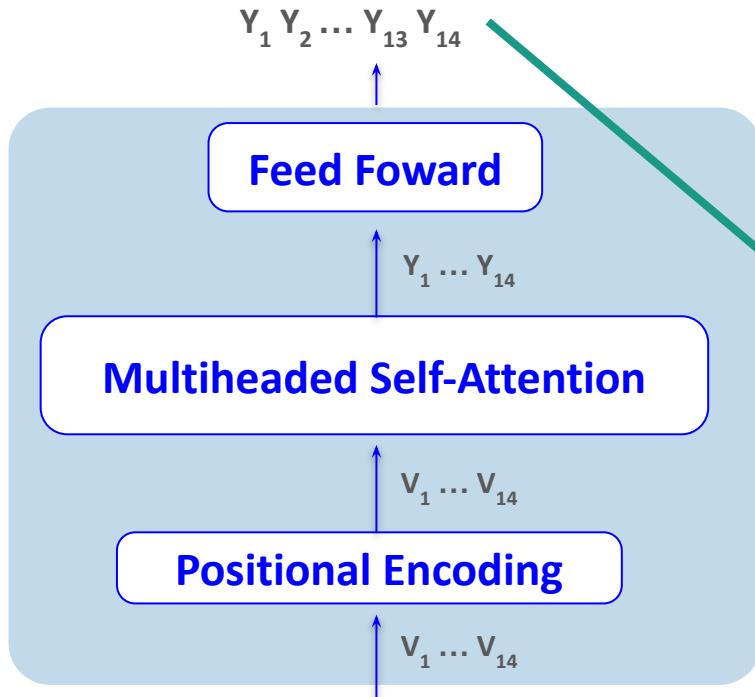
River ganó y sumó 3 puntos que lo
dejan en la punta del campeonato



Decoder

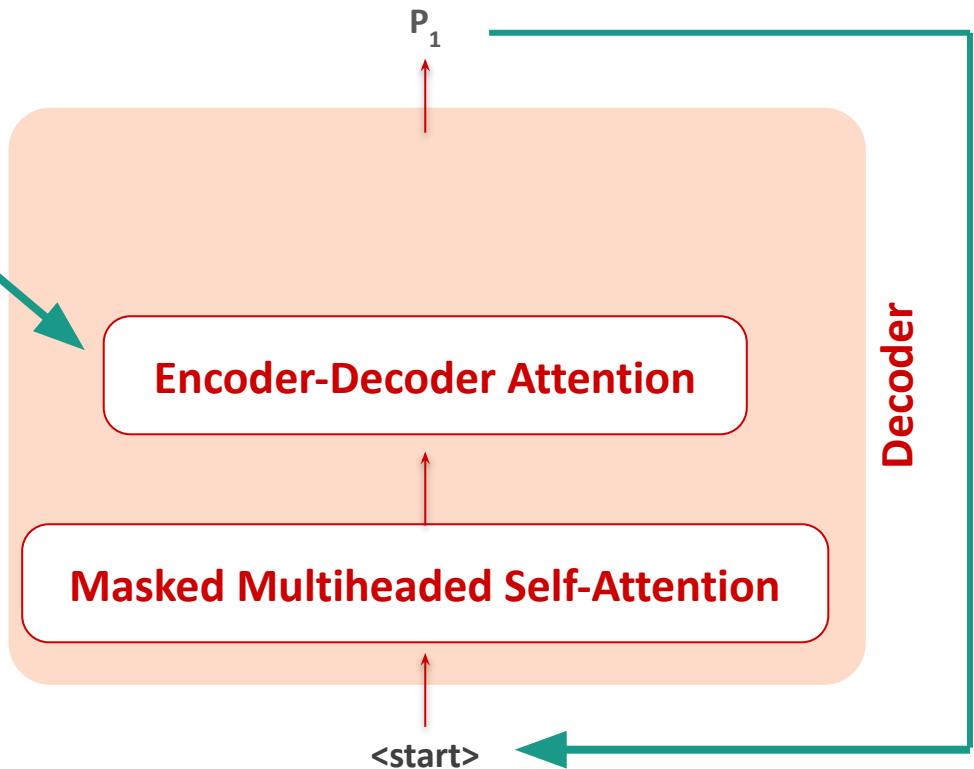
TRANSFORMER: TRADUCCIÓN

Encoder

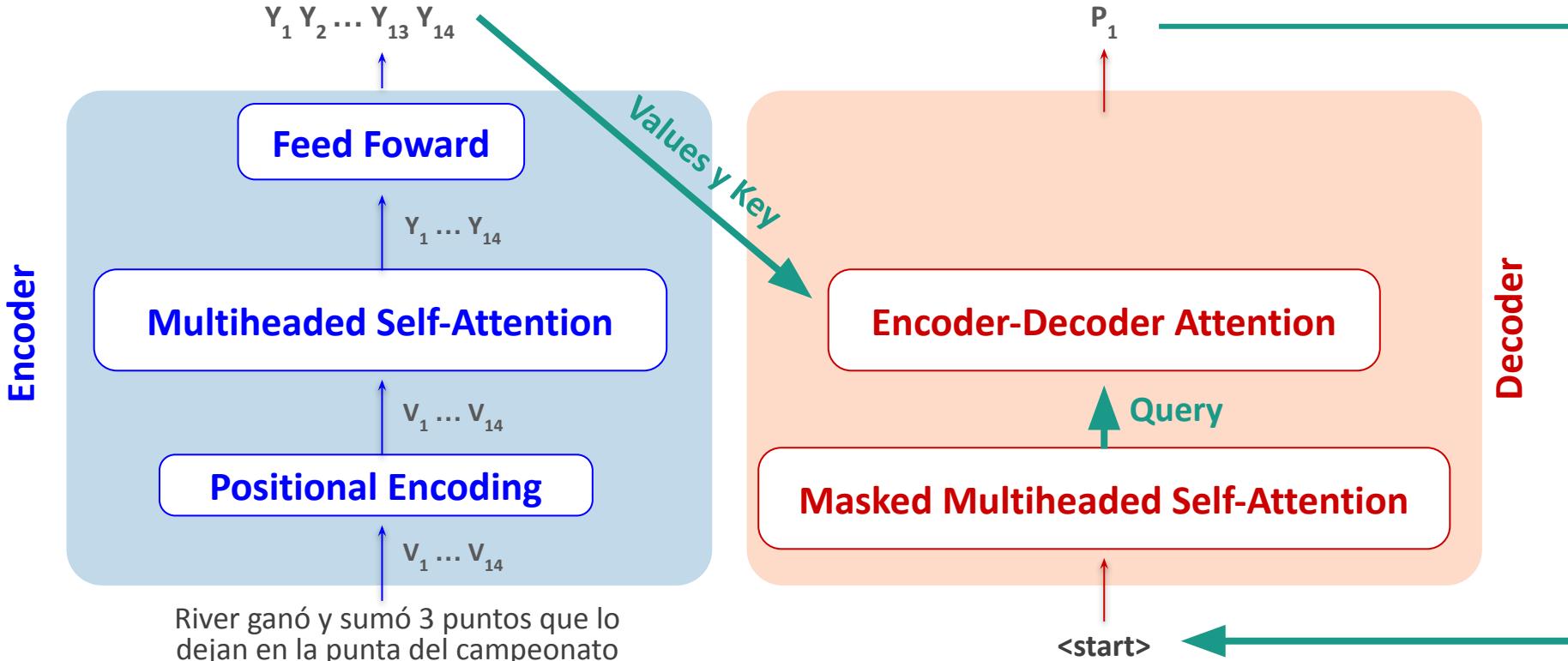


River ganó y sumó 3 puntos que lo
dejan en la punta del campeonato

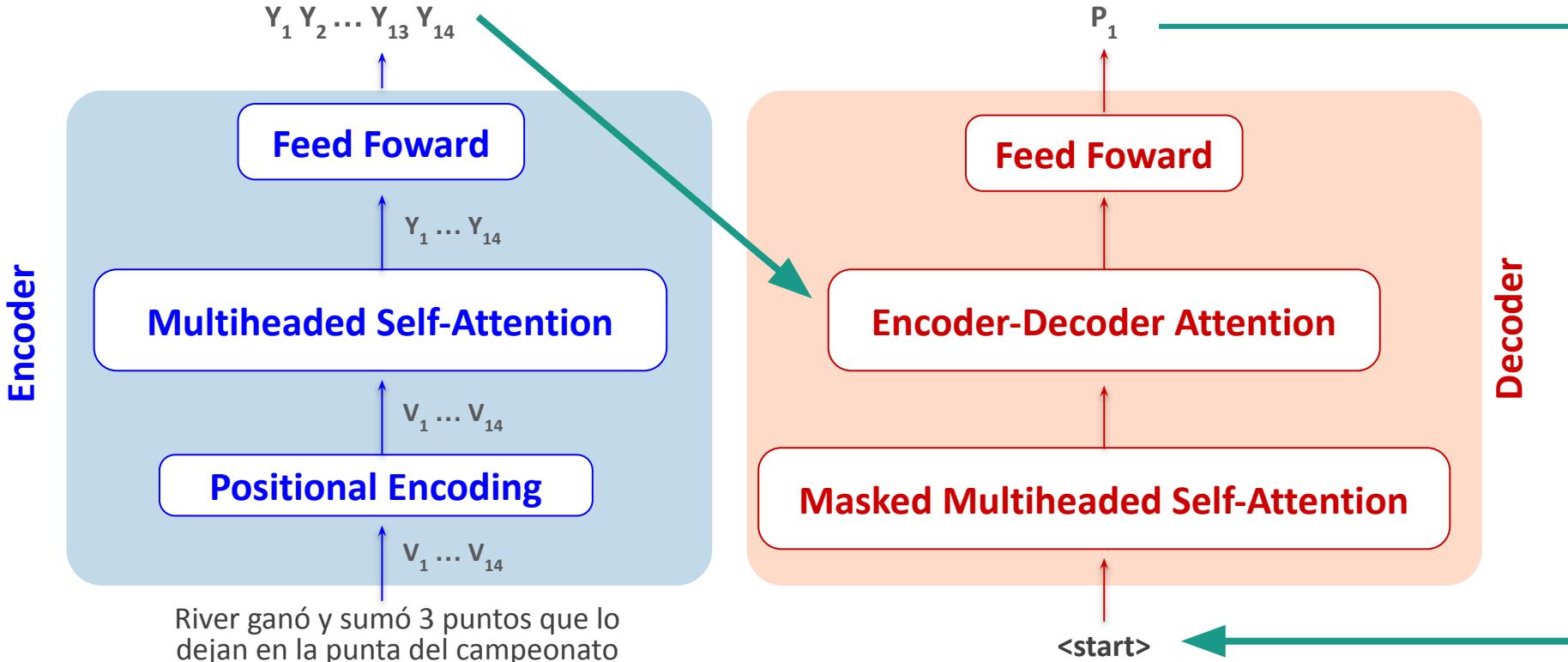
Decoder



TRANSFORMER: TRADUCCIÓN



TRANSFORMER: TRADUCCIÓN

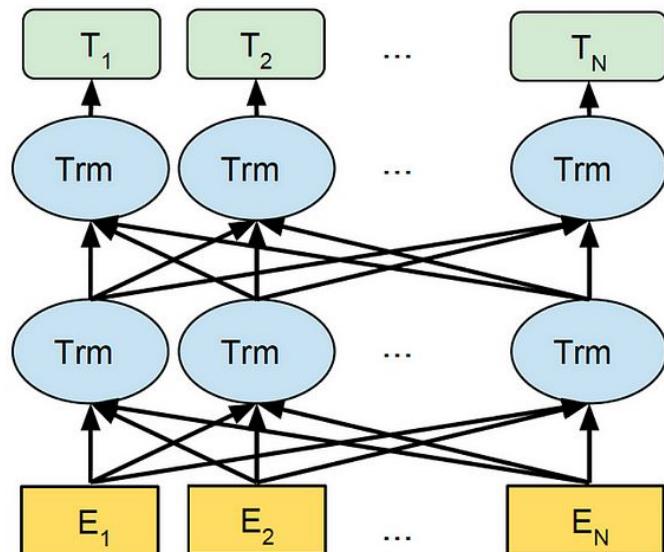




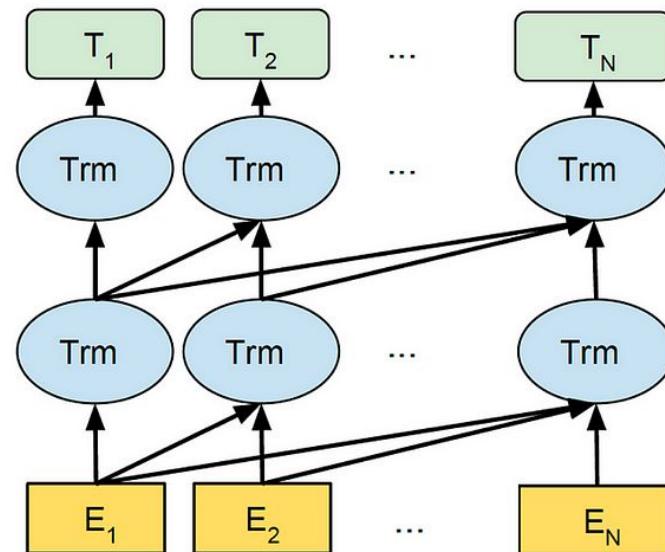
DECODER ONLY

DECODER-ONLY TRANSFORMERS

Google BERT

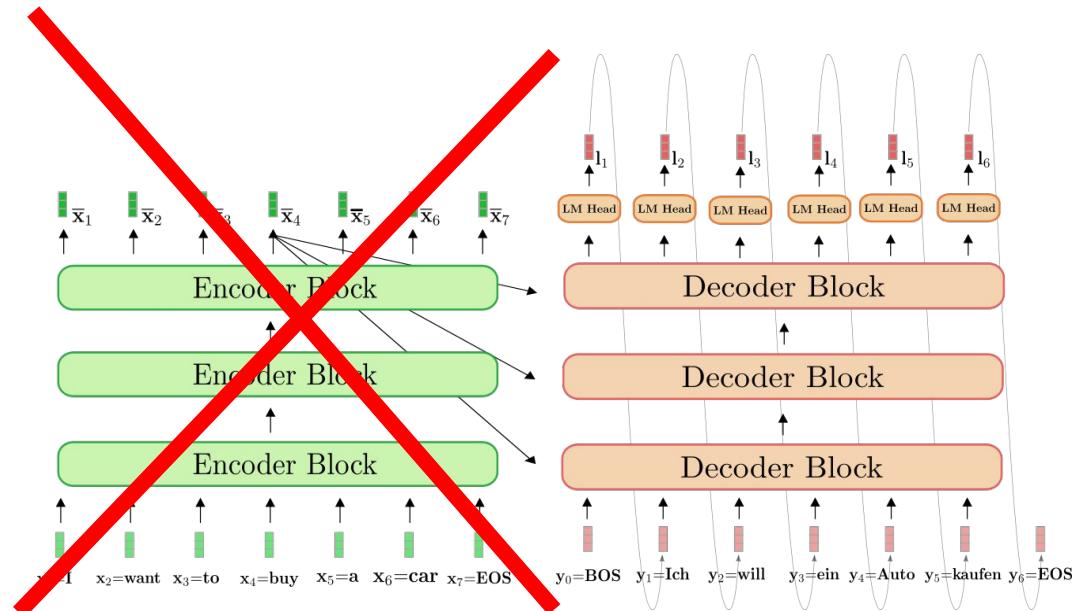


OpenAI GPT



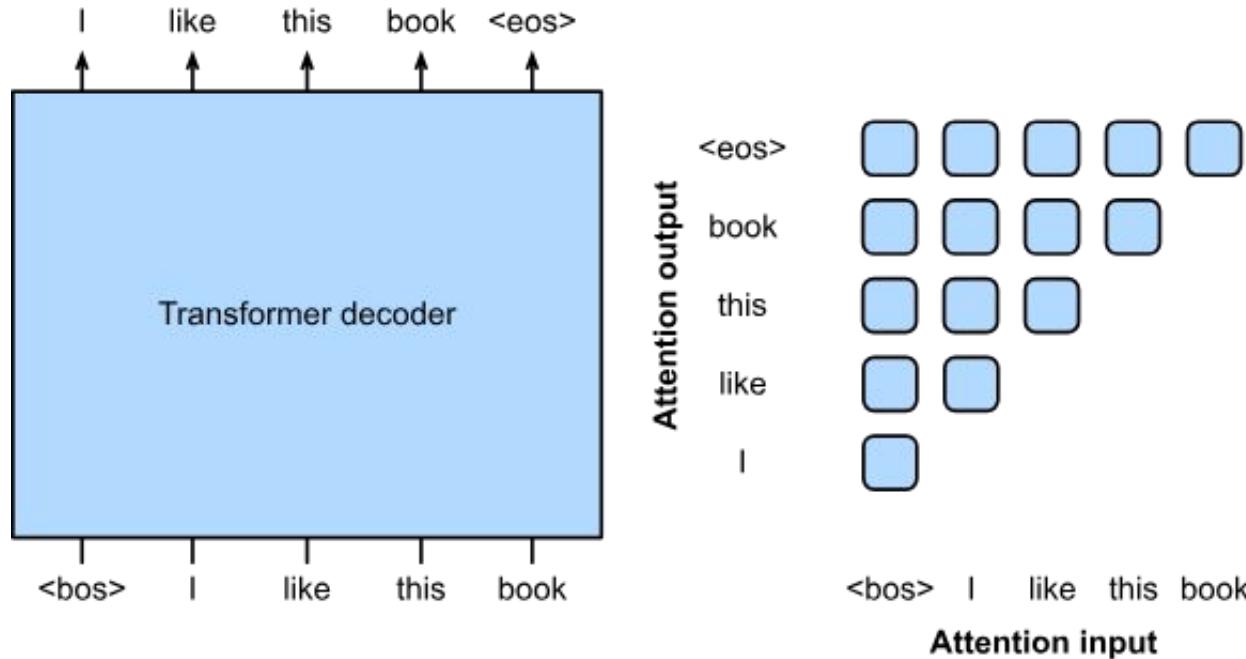
DECODER-ONLY TRANSFORMERS

- A diferencia del modelo encoder-decoder que tiene dos componentes, codificador y decodificador, los modelos solo-decodificadores consisten solo en el componente decoder
- **Entrenamiento Autorregresivo:** Prediciendo la próxima palabra/token basado en tokens anteriores.
- Se alimenta con una secuencia de entrada (a veces precedida por un token especial) y genera directamente la salida sin una representación intermedia explícita.

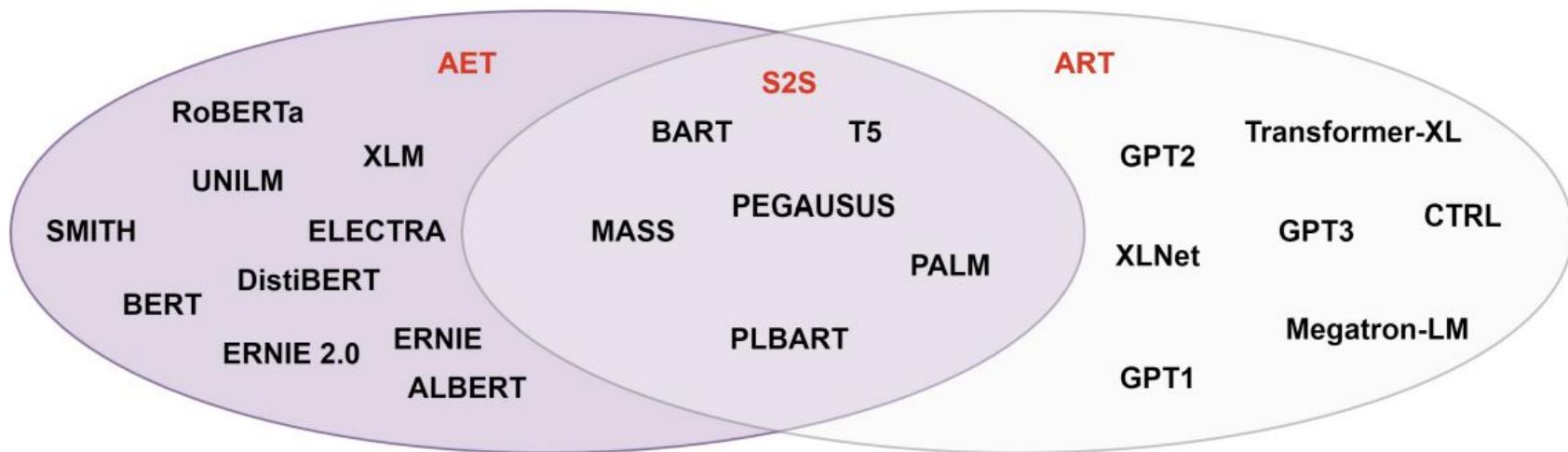


¿Qué ventajas y desventajas tiene?

DECODER-ONLY TRANSFORMERS



LANDSCAPE



TRANSFORMER

ALBERT	CPM	LayoutXLM	ProphetNet	XLM
Auto Classes	CTRL	LED	RAG	XLM-ProphetNet
BART	DeBERTa	Longformer	Reformer	XLM-RoBERTa
BARTnez	DeBERTa-v2	LUKE	RemBERT	XLNet
BEiT	DeiT	LXMERT	RetriBERT	XLSR-Wav2Vec2
BERT	DETR	MarianMT	RoBERTa	
Bertweet	DialoGPT	M2M100	RoFormer	
BertGeneration	DistilBERT	MBart and MBart-50	Speech Encoder Decoder Models	
BertJapanese	DPR	MegatronBERT	Speech2Text	
BigBird	ELECTRA	MegatronGPT2	Speech2Text2	
BigBirdPegasus	Encoder Decoder Models	MobileBERT	Splinter	
Blenderbot	FlauBERT	MPNet	SqueezeBERT	
Blenderbot Small	FNet	mT5	T5	
BORT	FSMT	OpenAI GPT	T5v1.1	
ByT5	Funnel Transformer	OpenAI GPT2	TAPAS	
CamemBERT	herBERT	GPT-J	Transformer XL	
CANINE	I-BERT	GPT Neo	Vision Transformer (ViT)	
CLIP	LayoutLM	Hubert	VisualBERT	
ConvBERT	LayoutLMV2	Pegasus	Wav2Vec2	

¡MUCHAS GRACIAS!

luciano@delcorro.info

READING LIST



- Hierarchical Attention Networks for Document Classification, Yang et al., 2016 (Opcional)
- Attention Is All You Need, Vaswani et. al. 2018 (casi opcional paper fundacional)
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding et. al., 2019 (importante)
- Improving Language Understanding by Generative Pre-Training, Radford et. al., 2019 (opcional, primer GPT, transformer unidireccional)
- Bishop, capítulo transformers (presentación más estructurada de transformers)