Transformers

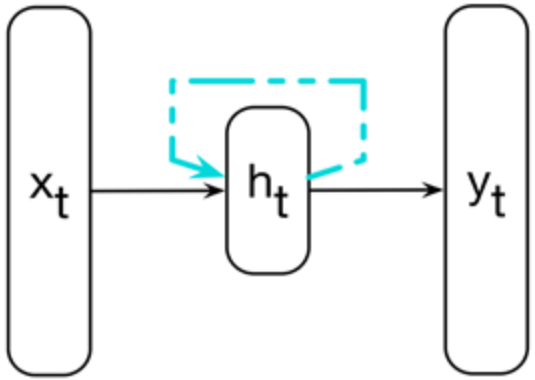
Andrés Abeliuk

Representaciones de texto

Para poder representar un texto podemos utilizar **métodos simples de vectorización que no toman en cuenta el orden de las palabras** dentro del documento (bolsa de palabras o TF*IDF)

o podemos utilizar representaciones más complejas que pueden tomar en cuenta el orden de las palabras.

BoW



RNN

Word embeddings: Representaciones de Texto Modernas

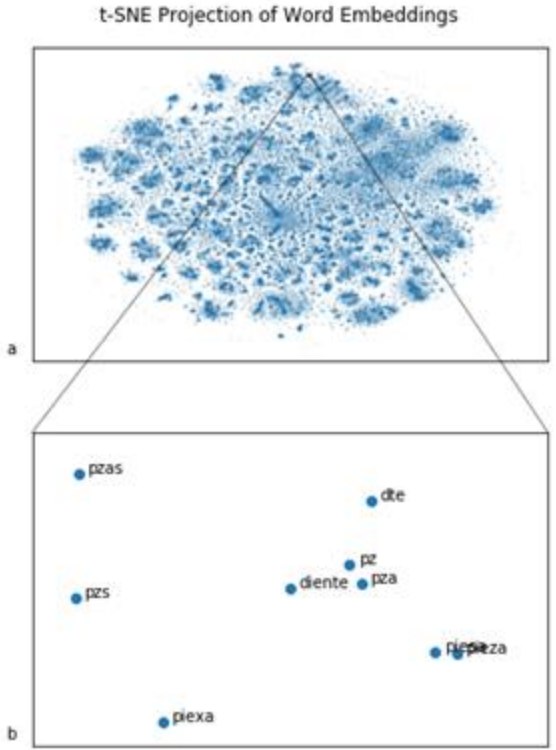
- Asignan un vector numérico único a cada palabra en el conjunto de entrenamiento.

Ventajas

- Capturan relaciones semánticas y similitudes entre palabras.
- Permiten **reducir la dimensionalidad** del texto, facilitando su procesamiento.

Limitaciones

- Modelos como **Word2Vec** y **GloVe** generan vectores estáticos (una representación fija por palabra).



Redes Neuronales Recurrentes

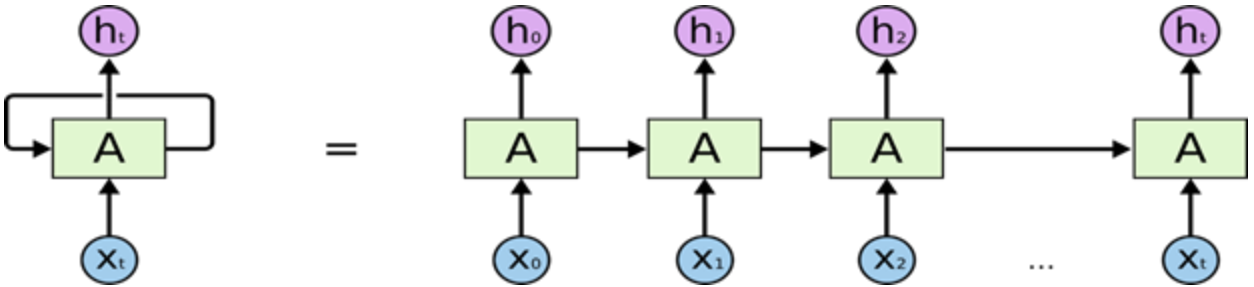
Técnica para representar texto que **considera el orden de las palabras**, modelando relaciones en secuencias como oraciones o documentos completos.

Ventajas:

- Capturan el contexto secuencial, lo que permite procesar texto en su **orden natural**.

Limitaciones:

- **Cálculo ineficiente:** Las RNN procesan las palabras de manera **serial**, lo que limita su velocidad y escalabilidad.
- **Pérdida de información distante:** Tienden a olvidar dependencias lejanas en la secuencia, dificultando la captura de relaciones a largo plazo.



Arquitectura de los Transformers

Componentes clave:

Los Transformers están formados por una **pila de bloques de transformer**, que combinan:

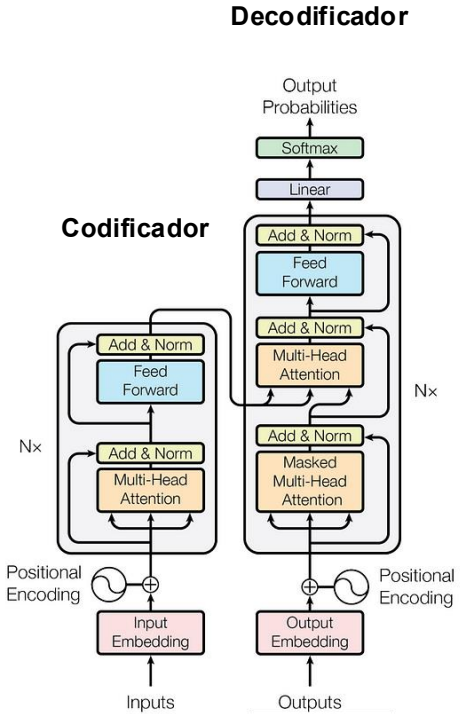
- **Capas lineales**: Transformaciones simples aplicadas a los vectores.
- **Redes totalmente conectadas**: Mejoran la capacidad de aprendizaje no lineal.
- **Capas de auto-atención**: El núcleo innovador que permite al modelo capturar relaciones entre elementos en cualquier posición de la secuencia, sin necesidad de procesarla de manera secuencial.

Ventajas:

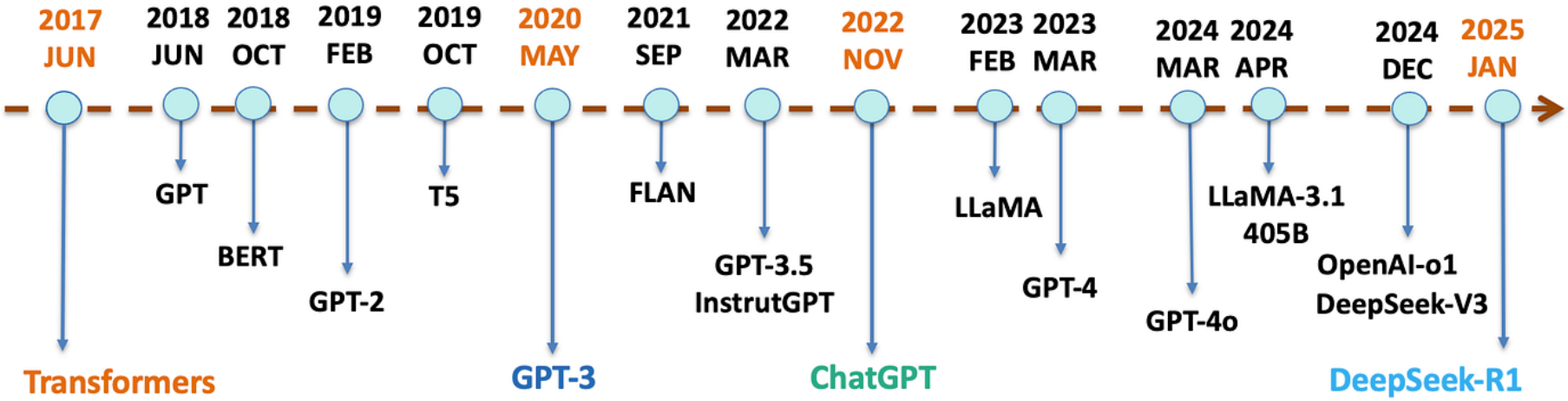
- **Paralelización:** Permiten procesar secuencias completas simultáneamente, aumentando la eficiencia.
- **Captura de relaciones globales:** Identifican dependencias entre elementos distantes en la secuencia.

Limitaciones:

- Requieren **gran capacidad computacional**

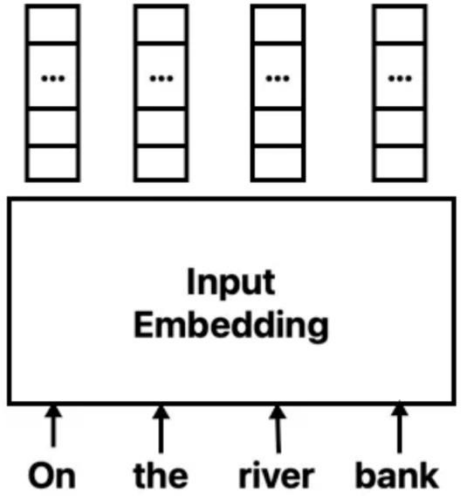


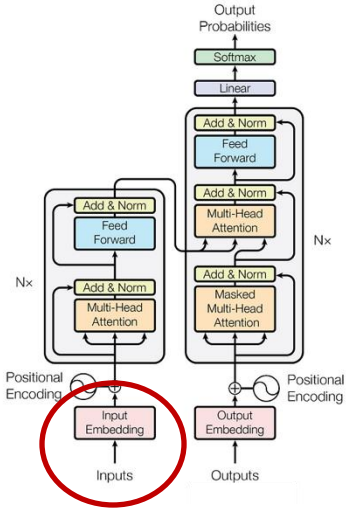
A Brief History of LLMs



Embeddedings de entrada

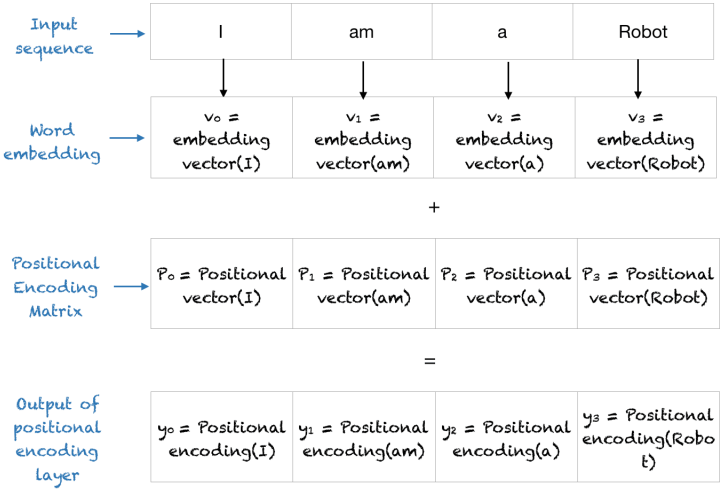
Los transformers no aceptan texto en bruto como entrada. Por lo tanto, al igual que hacemos con las RNNs, generamos las embeddings de palabras para la secuencia de entrada.

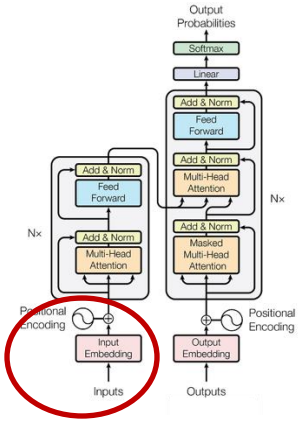




Codificación posicional

- Los embeddings no codifican la posición relativa de los tokens en una frase.
- La codificación posicional codifica la posición de las palabras en la secuencia.
- Los tokens estarán más cerca unos de otros en función de la similitud de su significado y de su posición en la frase.



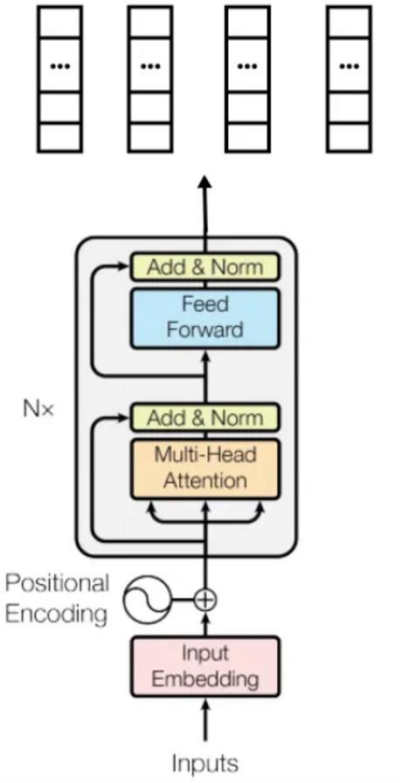


Fuente: https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Codificador del Transformer

El codificador se encarga de mapear una secuencia de entrada a una secuencia de representaciones continuas vectoriales.

La representación vectorial contiene la información de cómo se relacionan las palabras entre sí.



Mecanismo de Atención

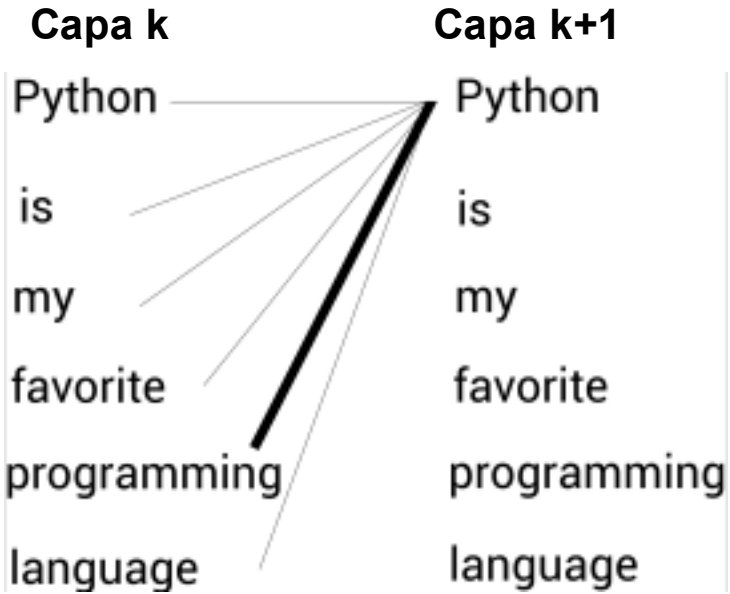
- Las palabras que ayudan a dar significado pueden estar lejos en la oración o párrafo.
- Los Transformers construyen representaciones contextuales (embeddings) integrando estas palabras.

Representaciones por Capas

- En cada capa, el token i se actualiza con información:
  - De su propia representación anterior.
  - De tokens vecinos en el contexto.

Mecanismo de Atención

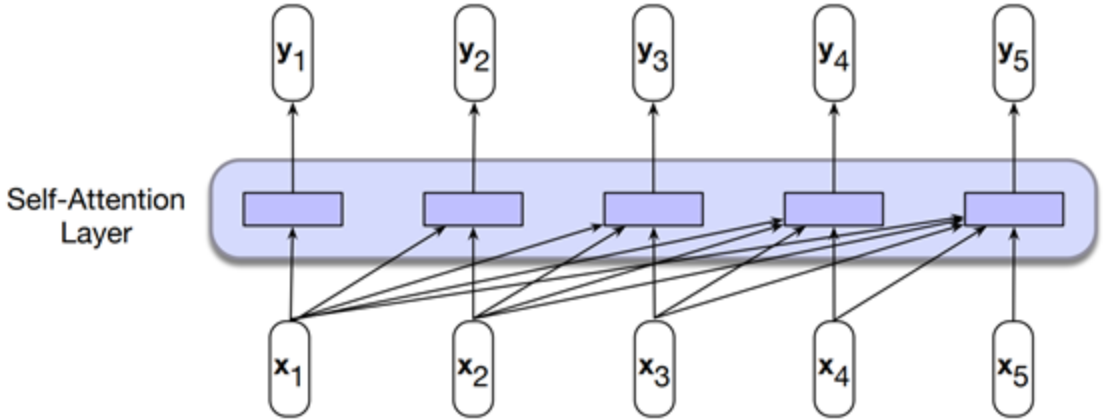
- Atención combina y pondera las representaciones de otros tokens relevantes.
- Con esto se construye la nueva representación en la capa siguiente (k+1).



Autoatención

La **autoatención permite a la red directamente extraer y usar información desde contextos arbitrariamente largos sin la necesidad de pasar a través de conexiones recurrentes intermedias**.

Las capas de autoatención mapean secuencias de entrada hacia secuencias de salida del mismo tamaño.

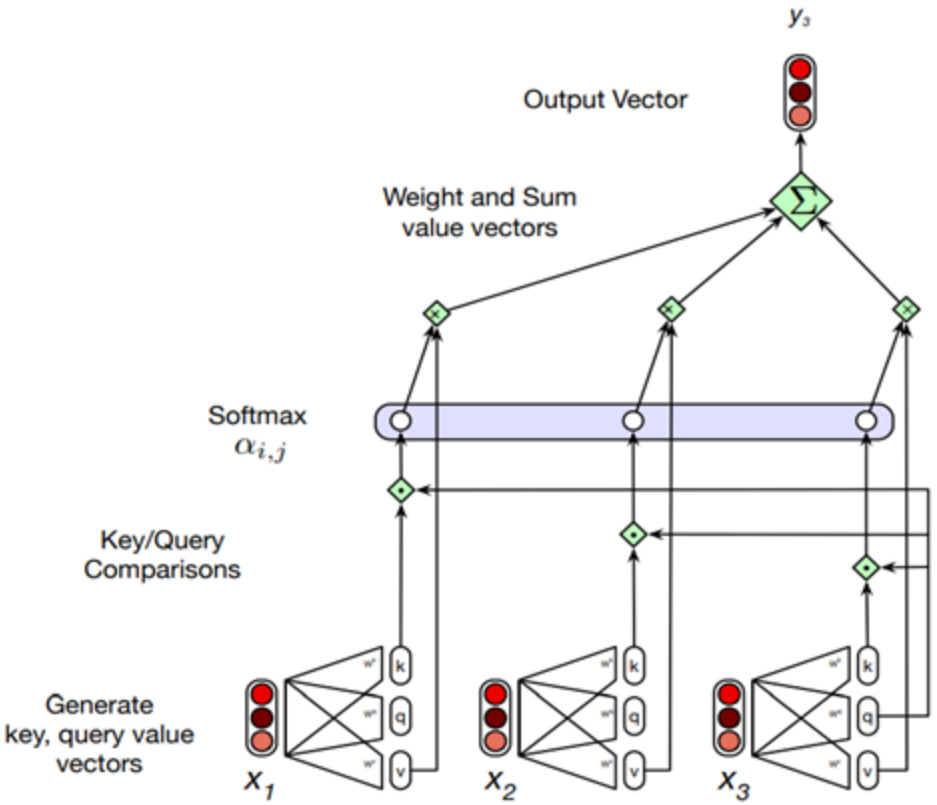


Algoritmo de autoatención

El vector de salida es una combinación de cada uno de los vectores de entrada, en donde **se pondera de mayor manera a los vectores que son más relevantes como contexto**

Consideremos los tres roles que desempeña cada token de entrada en el proceso de atención.

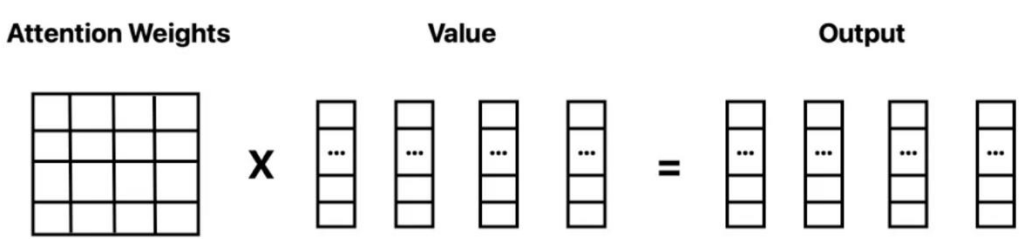
1. **Query**: Este vector representa la información específica que un token está buscando dentro de la secuencia, actuando como una consulta de búsqueda.
2. **Key:** Cada token de la secuencia también tiene un vector «clave», que se compara con el vector de consulta para determinar su relevancia en el contexto actual.
3. **Value:** El vector «valor» contiene la información real asociada a un token, que sólo se considera relevante si su «clave» correspondiente coincide con la «consulta».



Autoatención

Los valores de la matriz resultante determinan cuánta atención debe prestarse a las demás palabras de la secuencia dada la palabra actual.

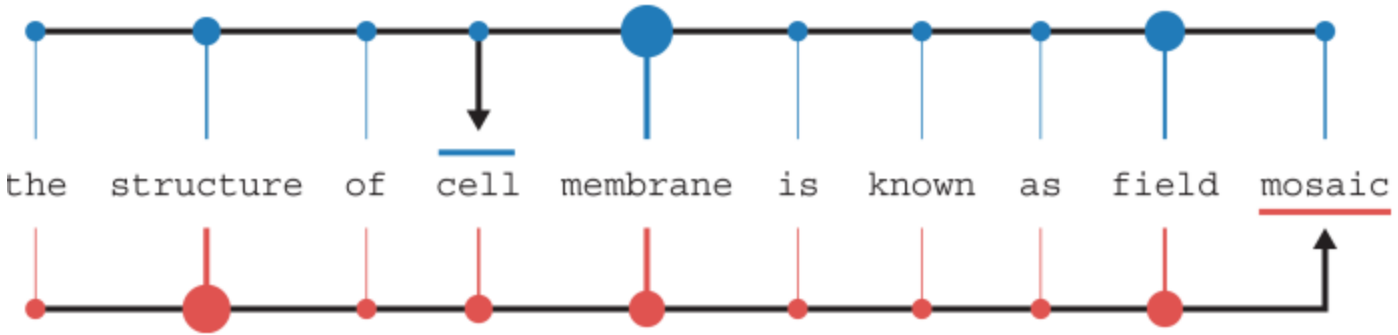
	On	the	river	bank
On	0.7	0.1	0.1	0.1
the	0.2	0.9	0.5	0.4
river	0.4	0.1	0.8	0.3
bank	0.3	0.4	0.3	0.8



La autoatención contextualiza las palabras

Con la autoatención podemos **mezclar los vectores de entrada individuales para contextualizar la información de cada palabra aislada**.

La salida de esta capa genera la misma cantidad de vectores de entrada, pero estos son mezclas de los vectores de entrada en función de su relevancia para el vector de entrada.

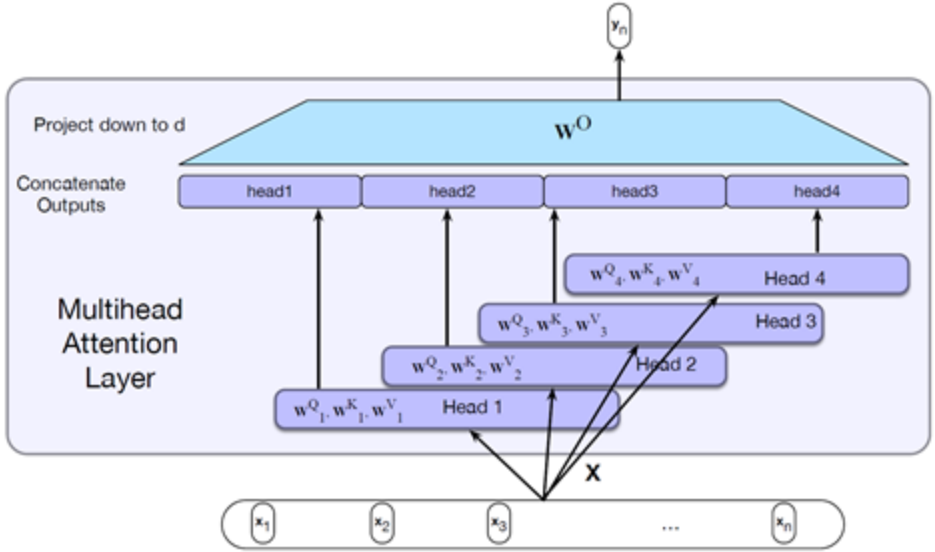


Atención multi-cabezal

Las distintas palabras en una oración se pueden relacionar con otras palabras de distintas maneras.

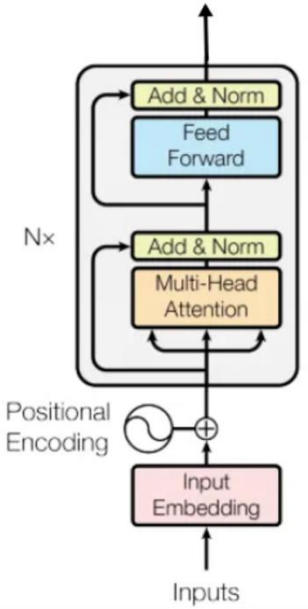
Una sola matriz de autoatención no sería suficiente para extraer todos los tipos de relaciones que existen entre palabras.

La atención multicabezal son múltiples capas paralelas de autoatención, llamadas *heads*, que contienen sus propios conjuntos de parámetros.



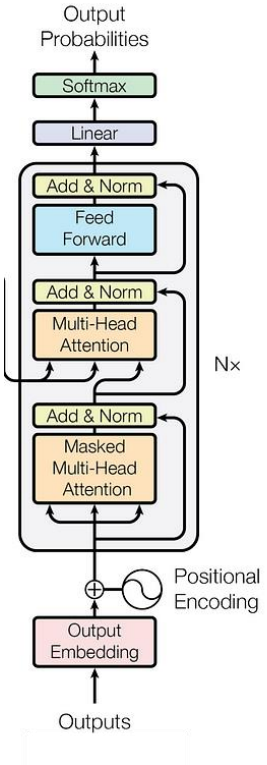
Transformer block

La autoatención es el centro de los bloques de transformers, estos bloques pueden contener más elementos, como redes densas y normalizaciones, junto con conexiones residuales que mejoran el rendimiento de las redes.



Decodificador

- La función del descodificador es generar texto.
- El descodificador tiene capas ocultas similares a las del codificador.
- Sin embargo, a diferencia del codificador, la salida del decodificador se envía a una capa softmax para calcular la probabilidad de la siguiente palabra de la secuencia.
- El descodificador es autorregresivo, lo que significa que predice valores futuros basándose en valores anteriores.

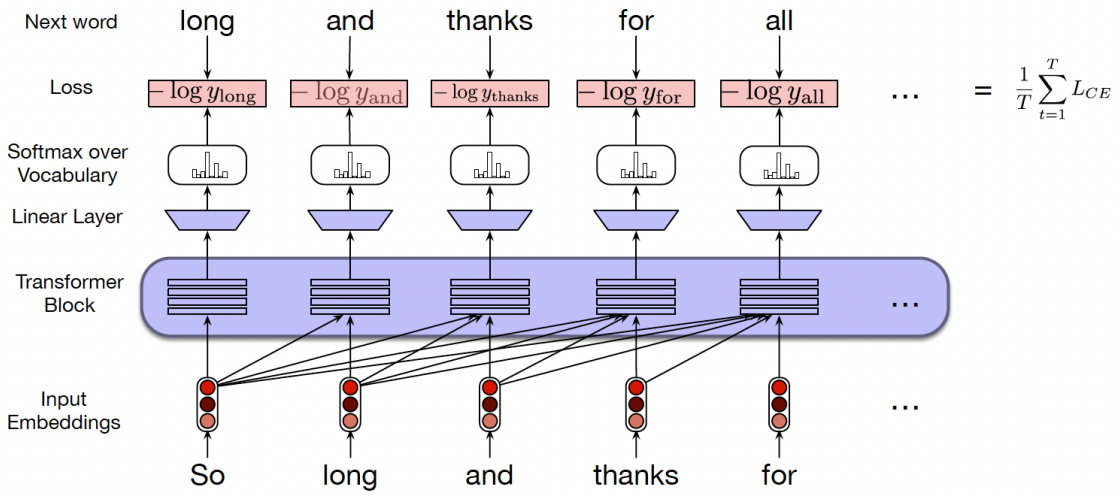


Enmascaramiento

Dado que el decodificador intenta generar la secuencia palabra por palabra, se utiliza una máscara de anticipación para indicar qué entradas no deben utilizarse.
Por ejemplo, al predecir el tercer token de la frase, sólo deben utilizarse los tokens anteriores, es decir, el primero y el segundo.

	On	the	river	bank
On	0.7	0	0	0
the	0.2	0.9	0	0
river	0.4	0.1	0.8	0
bank	0.3	0.4	0.3	0.8

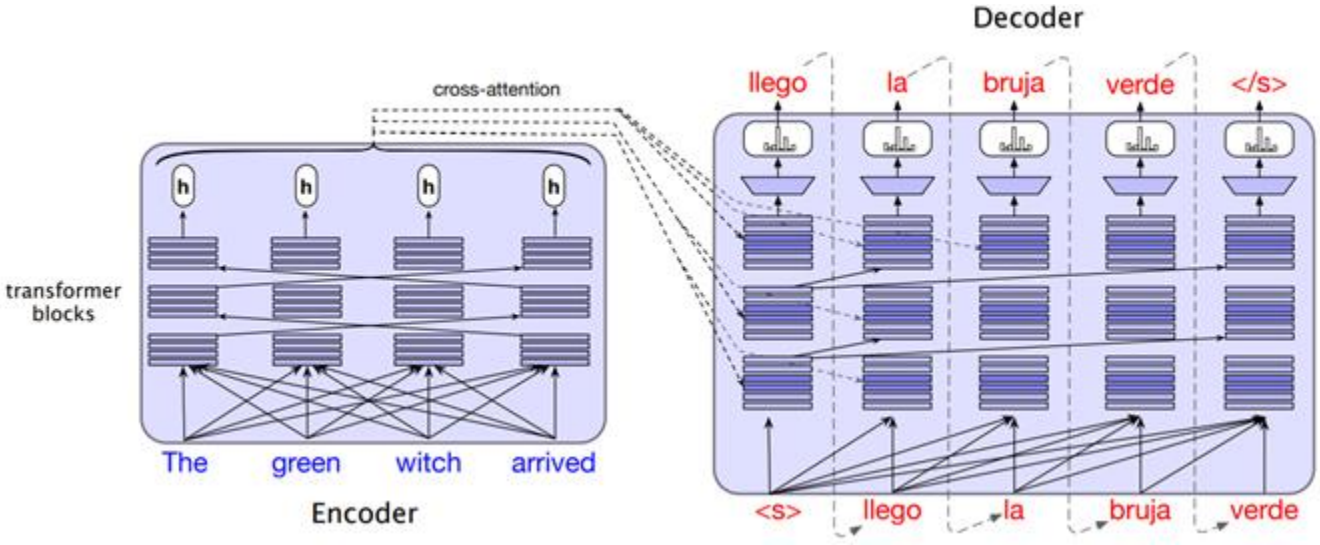
Transformers como modelos de lenguaje



Encoder-Decoder

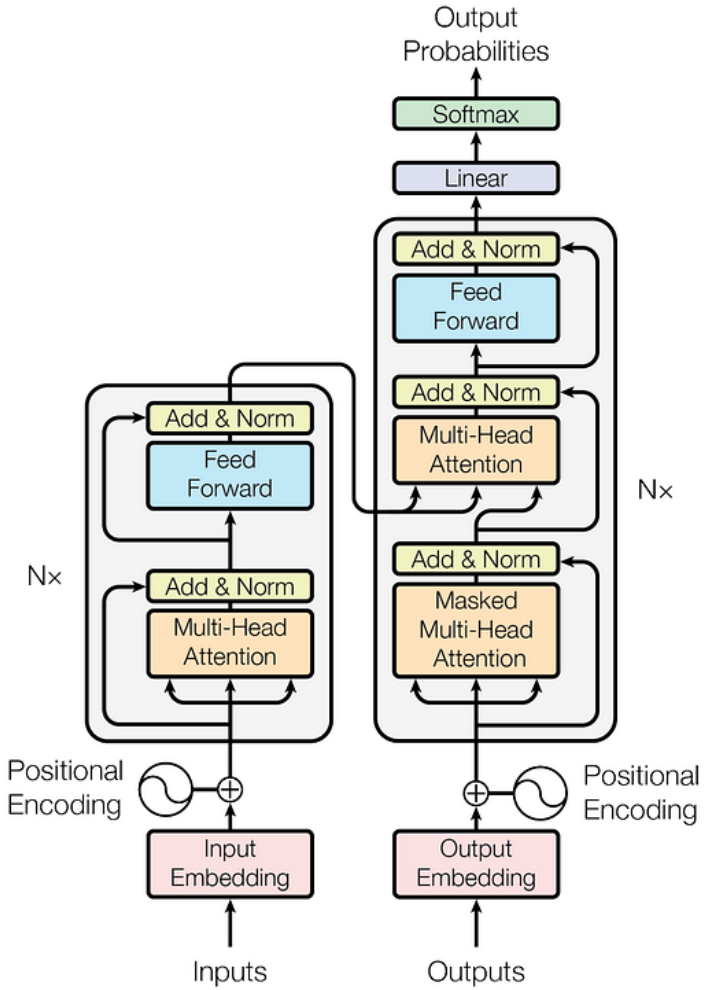
La estructura encoder-decoder también se puede implementar con transformers.

Esto consiste en un encoder que representa la secuencia de entrada y el decoder es un modelo de lenguaje condicional que utiliza la secuencia codificada para generar las palabras de la secuencia de salida.



BERT

Encoder



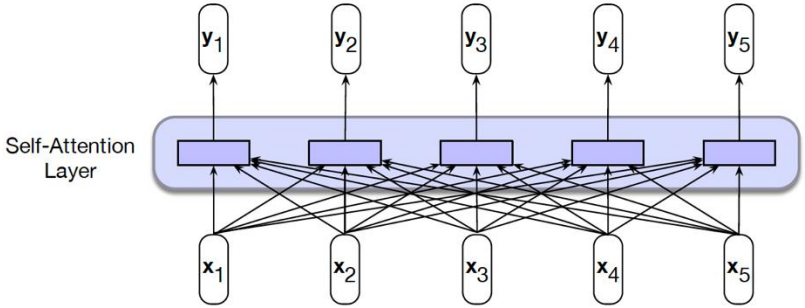
GPT

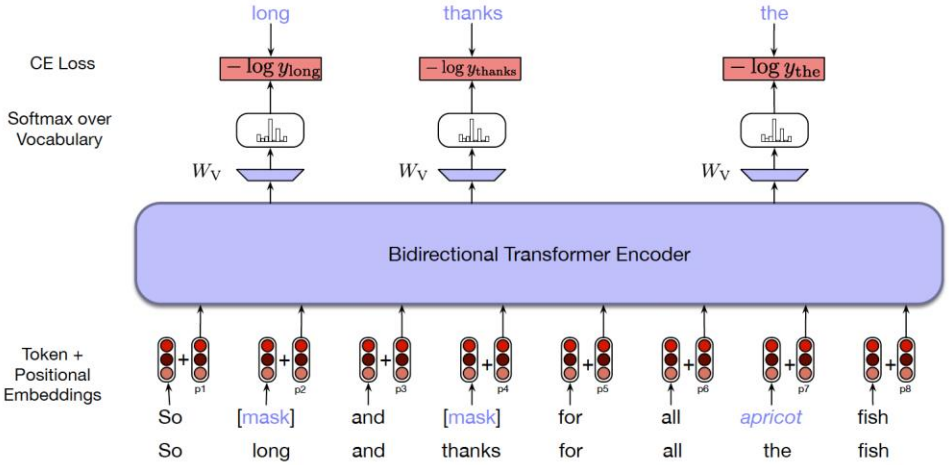
Decoder

BERT: Encoder

Bidirectional Encoder Representations from Transformers utiliza la arquitectura de encoder.

Esta arquitectura permite **obtener información de la secuencia completa a través de cambiar la tarea de predicción de la siguiente palabra por la predicción de una palabra enmascarada** dentro de la oración.





BETO

Este modelo de la arquitectura BERT **fue entrenado con el Big Spanish Corpus** utilizando con la técnica de modelado de lenguaje enmascarado.

Este modelo contiene un vocabulario de 31 mil subpalabras calculadas utilizando SentencePiece.

https://huggingface.co/dccuchile/bert-base-spanish-wwm-uncased

Task	BETO-cased	BETO-uncased	Best Multilingual BERT
POS	**98.97**	98.44	97.10 [2]
NER-C	**88.43**	82.67	87.38 [2]
MLDoc	95.60	**96.12**	95.70 [2]
PAWS-X	89.05	89.55	90.70 [8]
XNLI	**82.01**	80.15	78.50 [2]

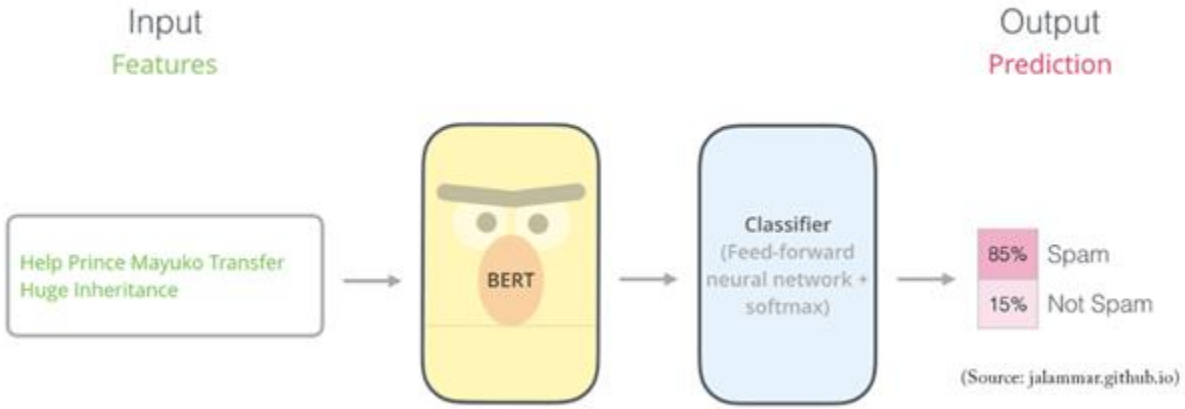
Predicción de la siguiente oración

Además de la predicción de palabras enmascaradas, BERT predice si una oración es posterior a la oración actual o no.

Esto es **motivado porque hay tareas en donde el modelo necesita codificar relaciones entre oraciones** o extraer información fuera de los límites de una oración.

Afinamiento

El afinamiento (*fine tuning*) de un modelo significa **utilizar un modelo de lenguaje pre entrenado para resolver una tarea específica**. Normalmente lo que se realiza es modificar los pesos del modelo y/o añadir capas al final del modelo de lenguaje que puedan resolver la tarea que deseamos.

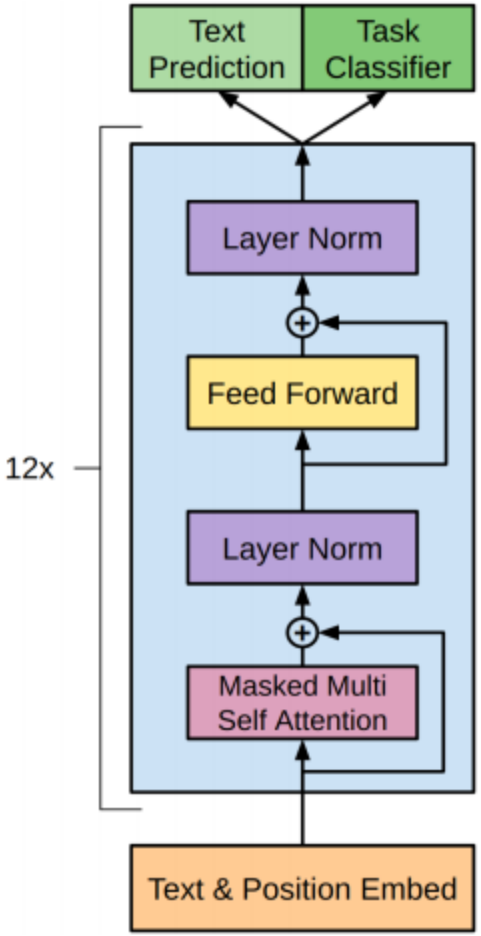


GPT: Decoder

Generative Pretrained Transformer es una arquitectura que **utiliza sólo un decoder para ajustar un modelo de lenguaje para la predicción de la siguiente palabra**.

Esta arquitectura sólo puede atender a las palabras previamente vistas para predecir la siguiente.

Esto puede ser limitante para poder codificar oraciones, porque uno debe observar toda la oración para poder entender una palabra.



Grandes modelos de lenguaje

Los grandes modelos de lenguaje son modelos pre entrenados basados en Transformers que tienen una cantidad de parámetros de órdenes de magnitud más grandes que los modelos pre entrenados anteriores.

Por ejemplo, el modelo pre entrenado BERT tiene $0.3*10^9$ parámetros y GPT-3 tiene $175*10^9$ parámetros y GPT-4 tiene 1000 más que GPT-3

Se sabe que escalar el tamaño de los modelos de lenguaje mejora el rendimiento de los modelos en tareas específicas. ¿La pregunta es hasta cuando?