# Redes Transformers

Clase VI - Introducción al Aprendizaje Profundo

# ¿Qué vimos hasta ahora?

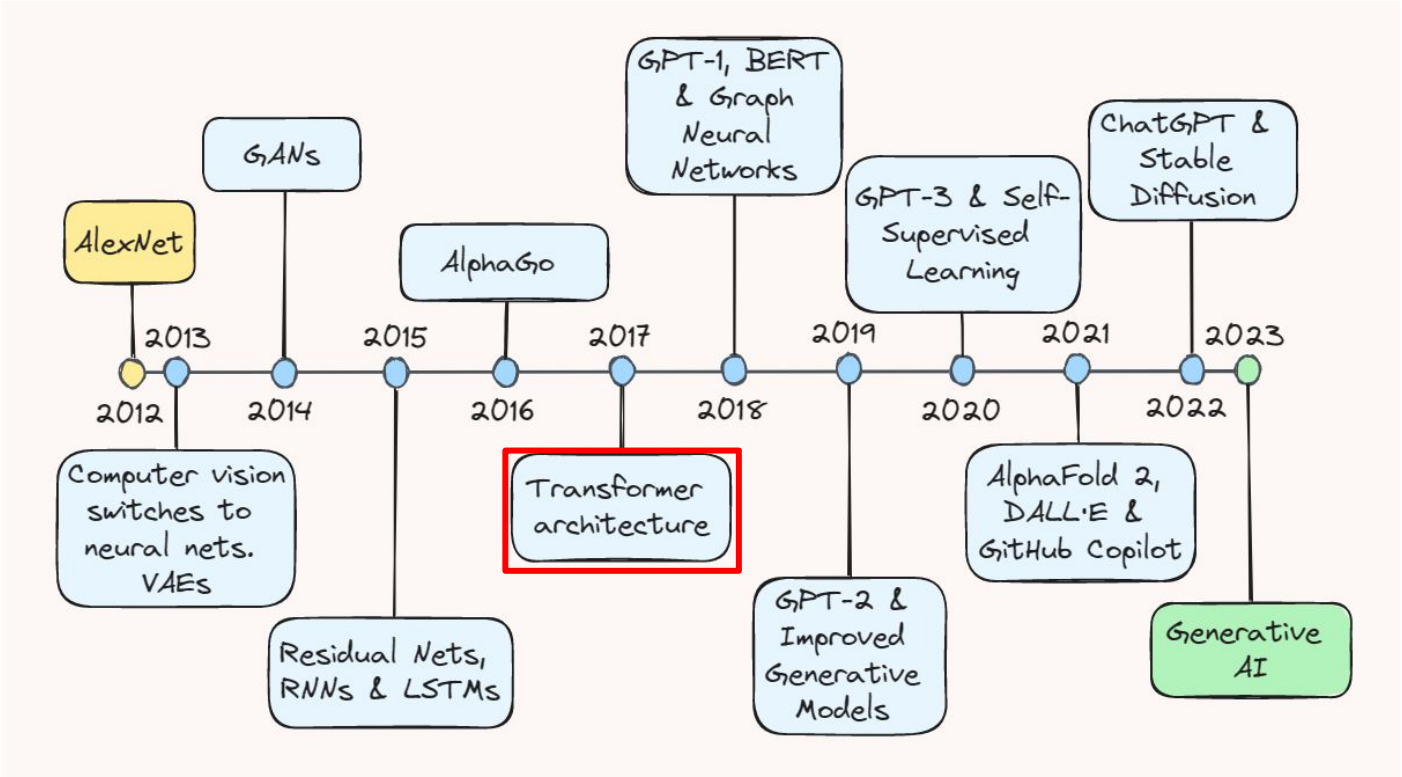
- Redes neuronales MLP.
- Redes convolucionales.
- Redes recurrentes.

# ¿Qué vamos a ver ahora?

- Introducción a transformers
- Introducción a BERT y GPT

Fuentes: del Optimus Prime: fanpop.com.
Del ícono: flaticon

# Usted está aquí

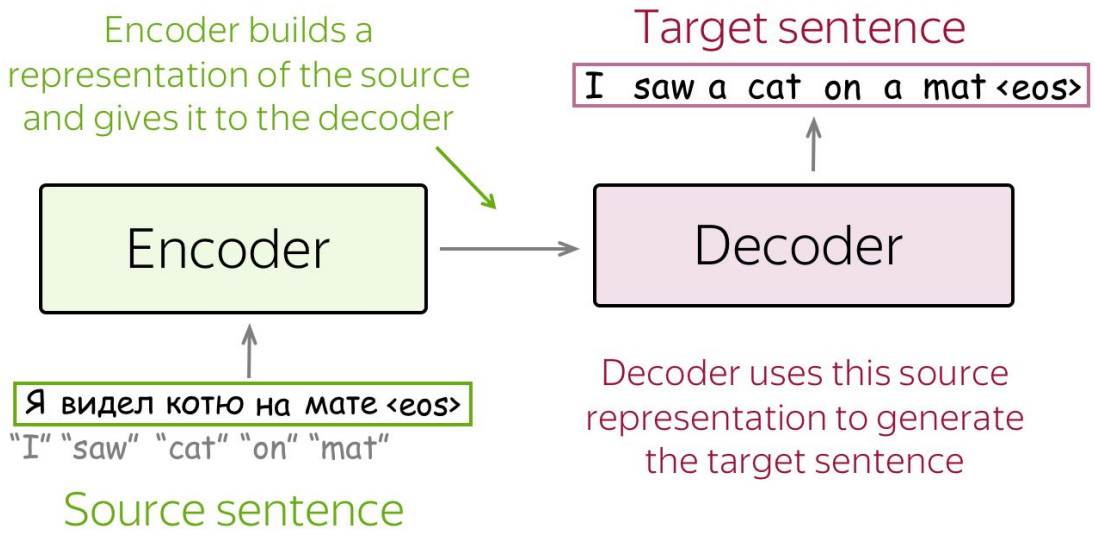


# Contexto

- El entrenamiento de las RNN es secuencial
- Esto hace que el entrenamiento no sea paralelizable como en una CNN
- Además, todo el contexto se resumía a un sólo vector
- Era un problema considerable en traducción de textos: por velocidad de entrenamiento y para mantener secuencias largas

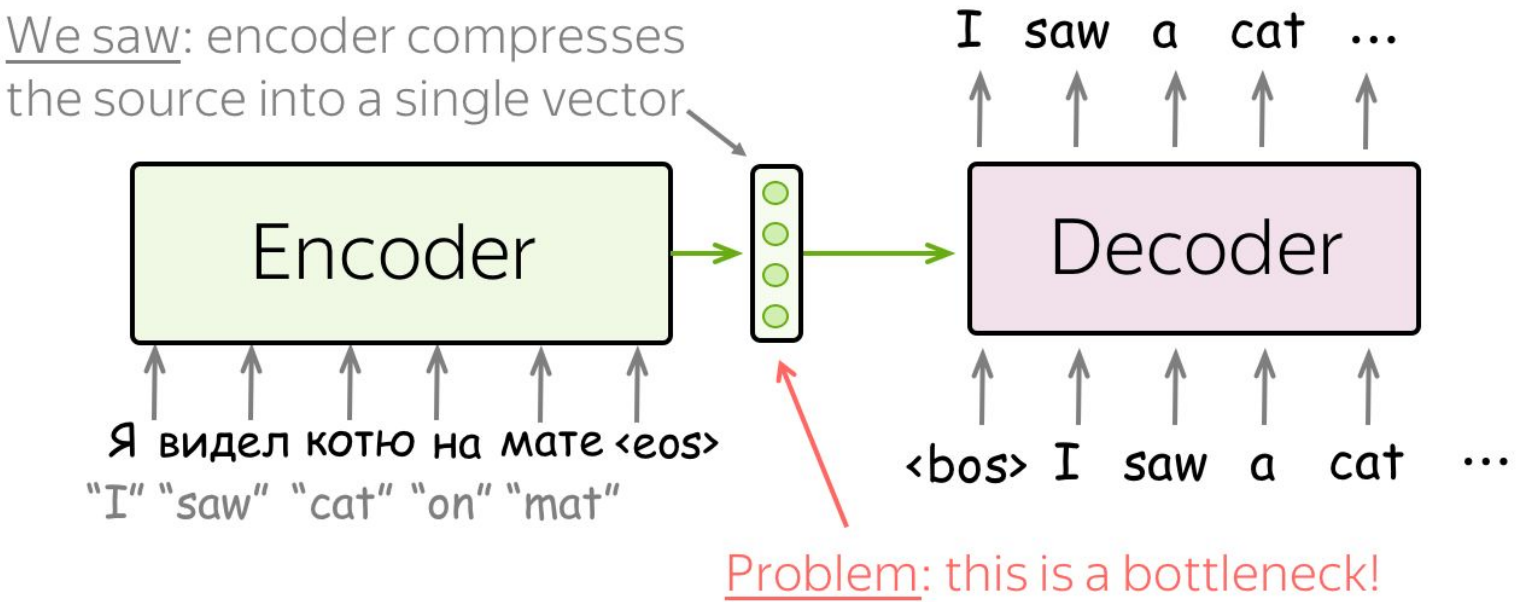
# Contexto

- Transformers surgen en 2017, originalmente para problemas de traducción
- En los mismos, se usaba una arquitectura de tipo encoder–decoder con LSTM




Fuente: Lena Voita

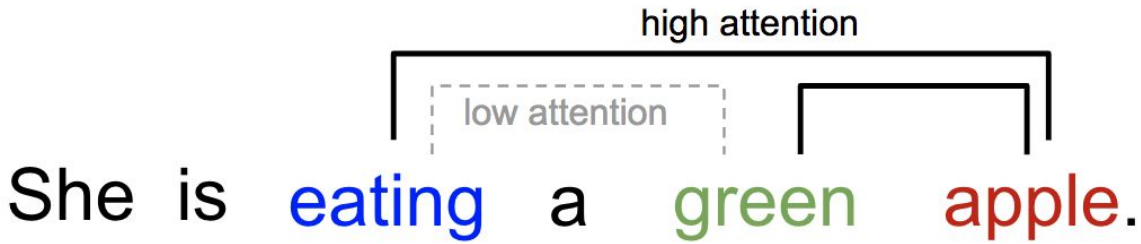
# Problema




Fuente: Lena Voita

# Mecanismo de Atención

- Método (del 2014) por el cual el modelo, en distintos momentos, se enfoca en distintas partes de la entrada.
- Para esto se pesan los tokens para que se “preste” atención a los más relevantes en un cierto momento.
- Este pesado lo aprende el modelo en el entrenamiento.




Fuente: Lilian Weng

# Transformers

- Surgen en 2017, con el paper seminal: Attention is all you need.
- Proponen una nueva arquitectura para resolver el anterior cuello de botella. Ahora pueden capturar contextos más largos y en paralelo.
- Se deja de usar recurrencia, y en su lugar va el mecanismo de atención, junto con varios ingredientes más.

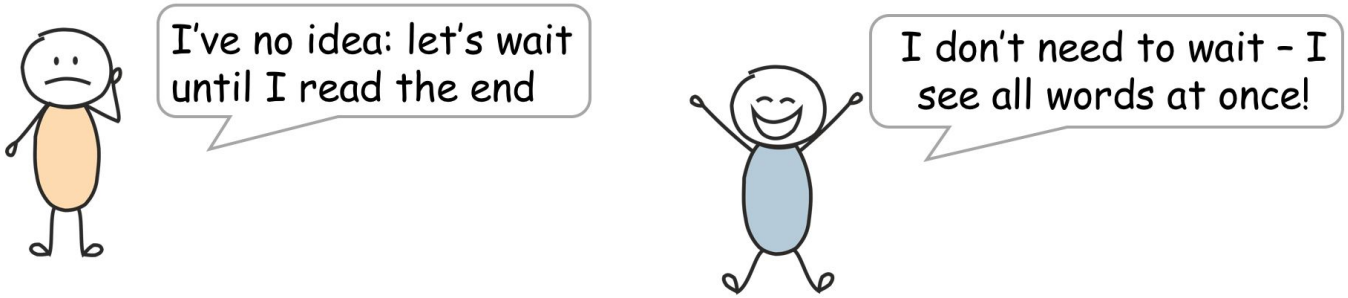
# Transformers

- En lugar de procesar token por token de una oración, procesa todos a la vez.
- Añade también otros ingredientes para poder usar tales mecanismos (referirse al material extra de la clase para un deep dive en los mismos).
- Corona el “momento computer vision” de NLP

# Transformers

I arrived at the bank after crossing the ... ...street? ...river?

What does **bank** mean in this sentence?




RNNs

O(N) steps to process a sentence with length N

Transformer

Constant number of steps to process any sentence

Fuente: Lena Voita

# Desventajas vs RNN

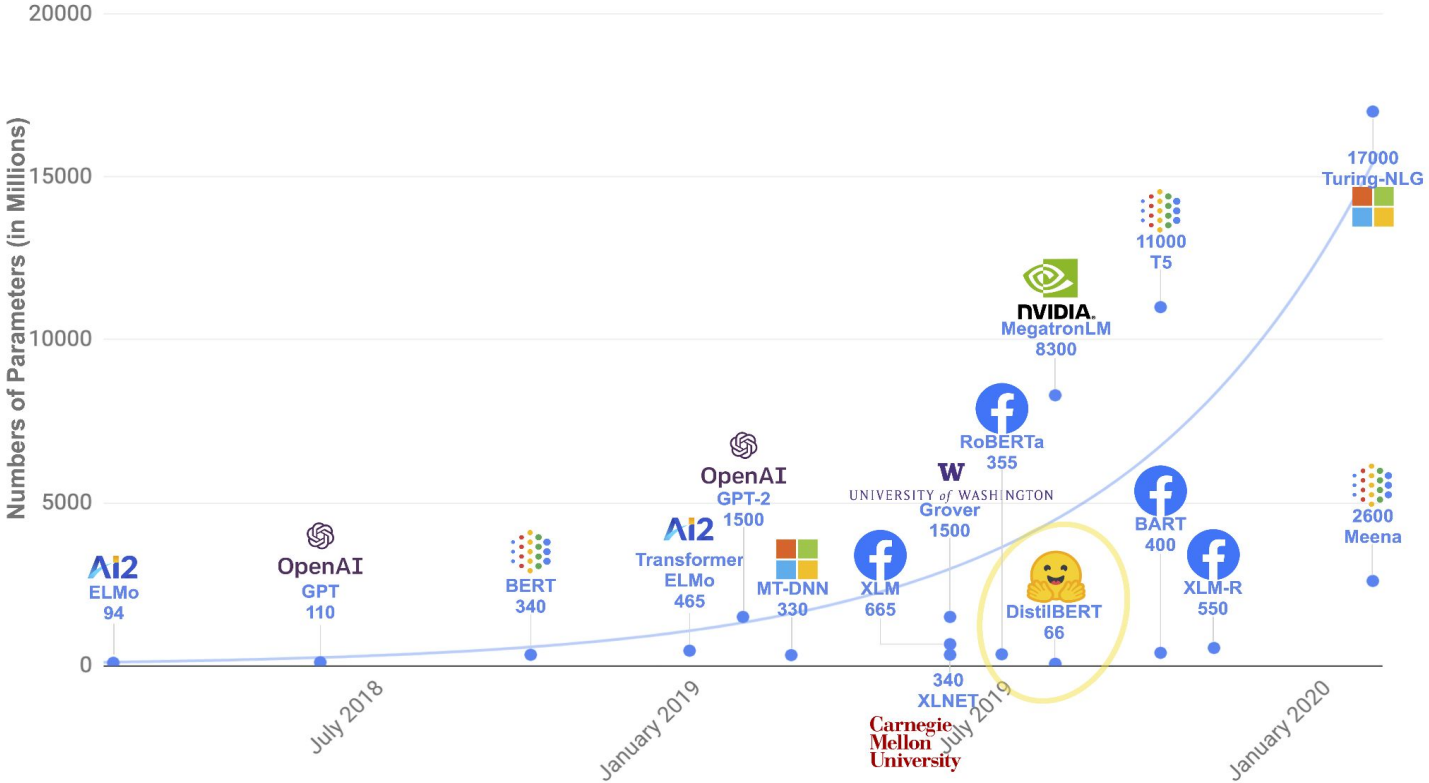
Los transformers requieren:

- Muchos (**muchos**!) más datos de entrenamiento
- Mucho más poder computacional

Además, la inferencia es más costosa y demora más

# Donde transformers se hacen LLMs...

Los transformers ya suponen modelos **MUY** grandes...

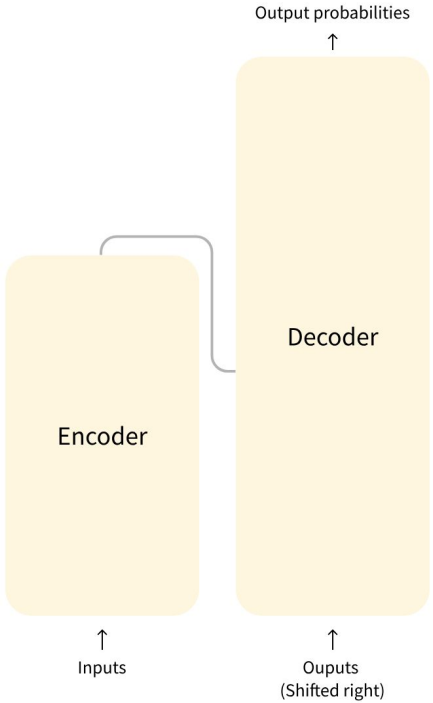



[Fuente: Huggingface]

# Vista general

Tres “sabores” básicos de transformers

- Encoder-decoder
  (el de la imágen, el original)
- Encoder solamente
- Decoder solamente

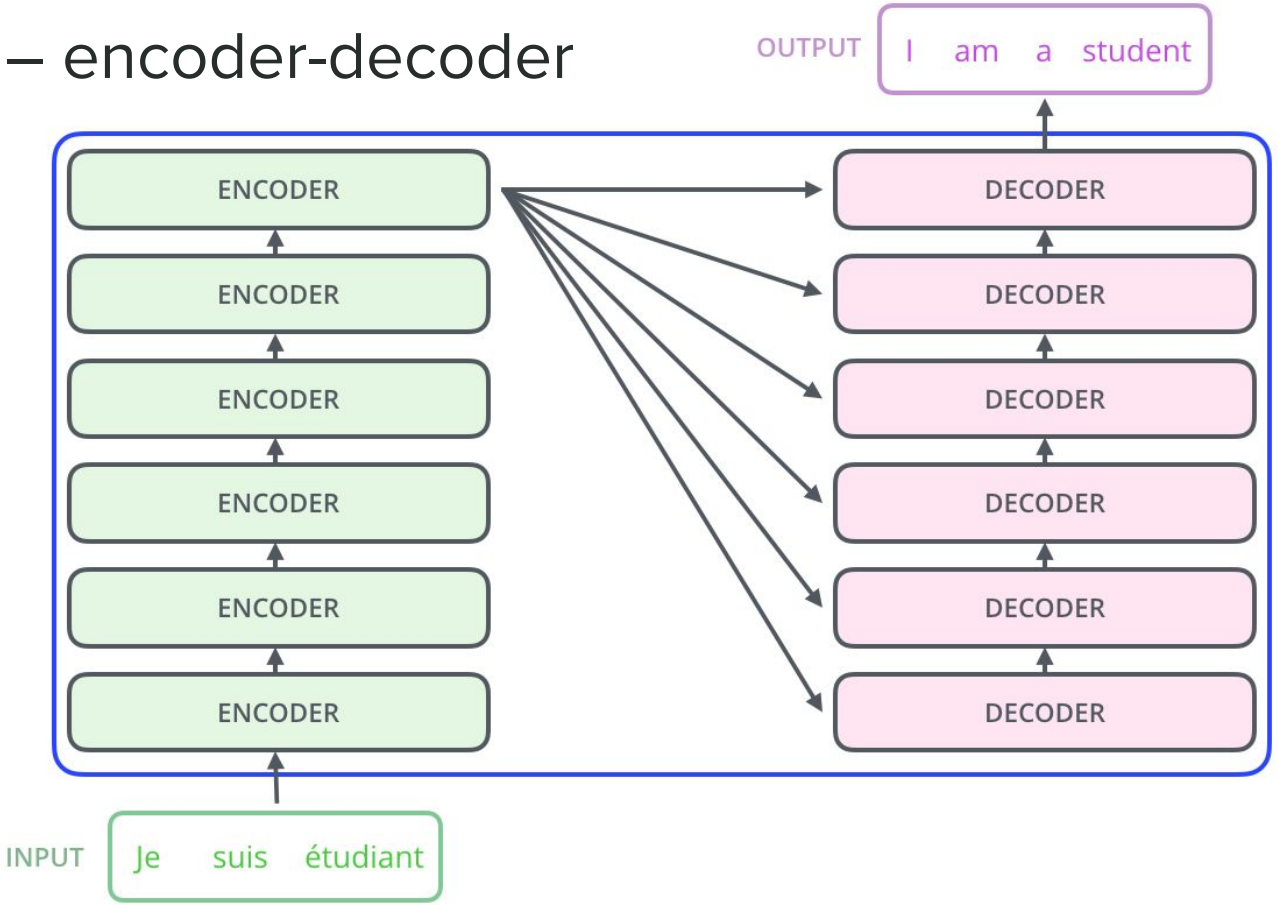



Fuente: Huggingface

# Vista general – encoder-decoder

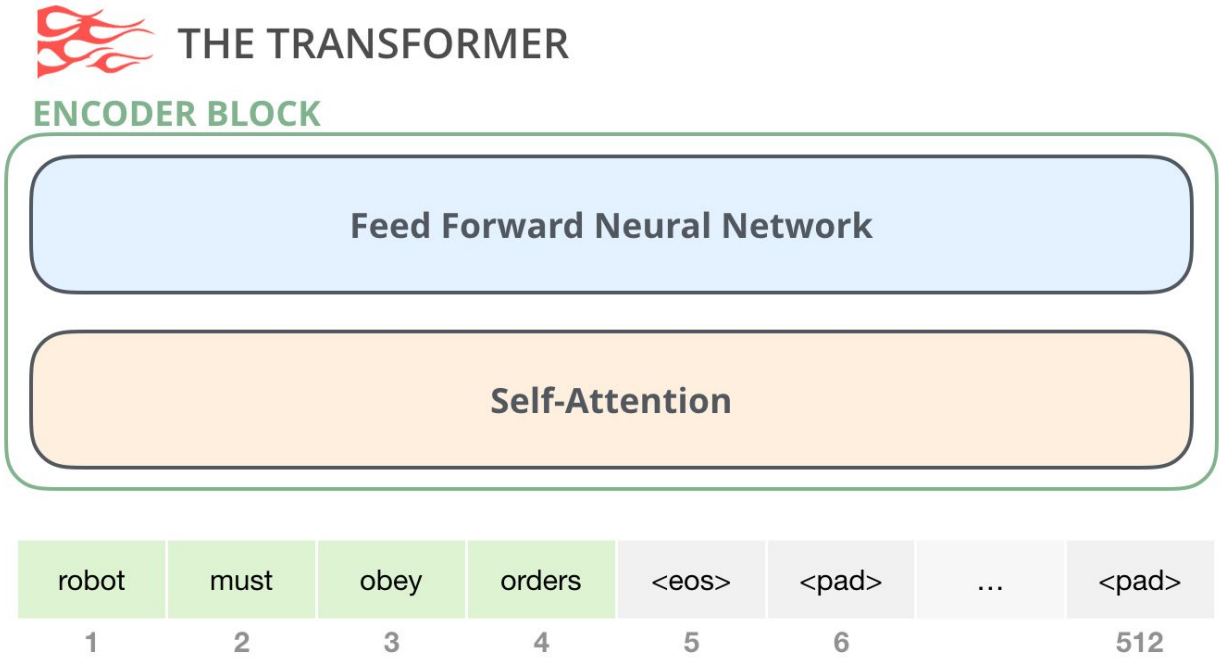
- Entrada: el texto se tokeniza, cada token se transforma en su embedding, y se le suma un encoding posicional para preservar info de la posición en la oración.
- Probabilidades de salida: se aplica softmax a los logits de salida, con lo que el modelo puede tener las probabilidades de los mismos
- Inició la arquitectura, suele usarse para tareas que involucren transformaciones como la traducción de texto.

# Vista general – encoder-decoder



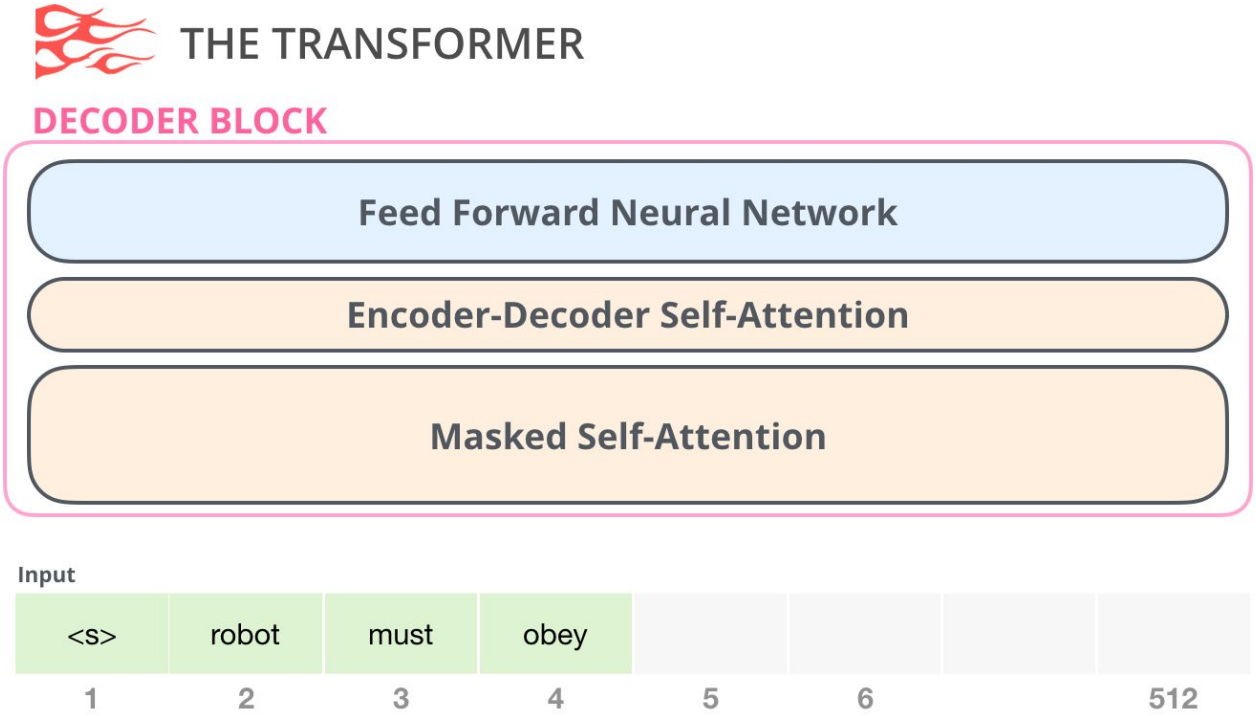

Fuente: Jay Alammar

# Vista general – encoder




Fuente: Jay Alammar

# Vista general – decoder



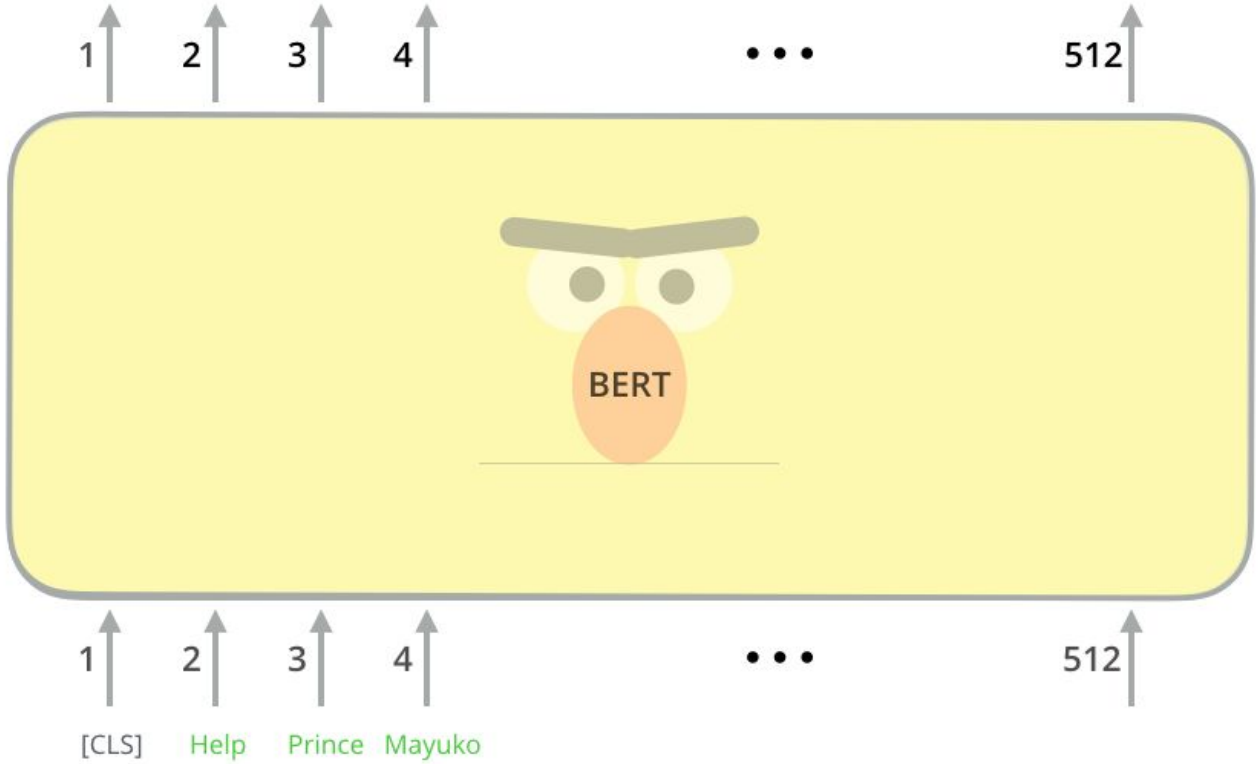

Fuente: Jay Alammar

# Flavors: BERT y GPT

# Encoders: BERT

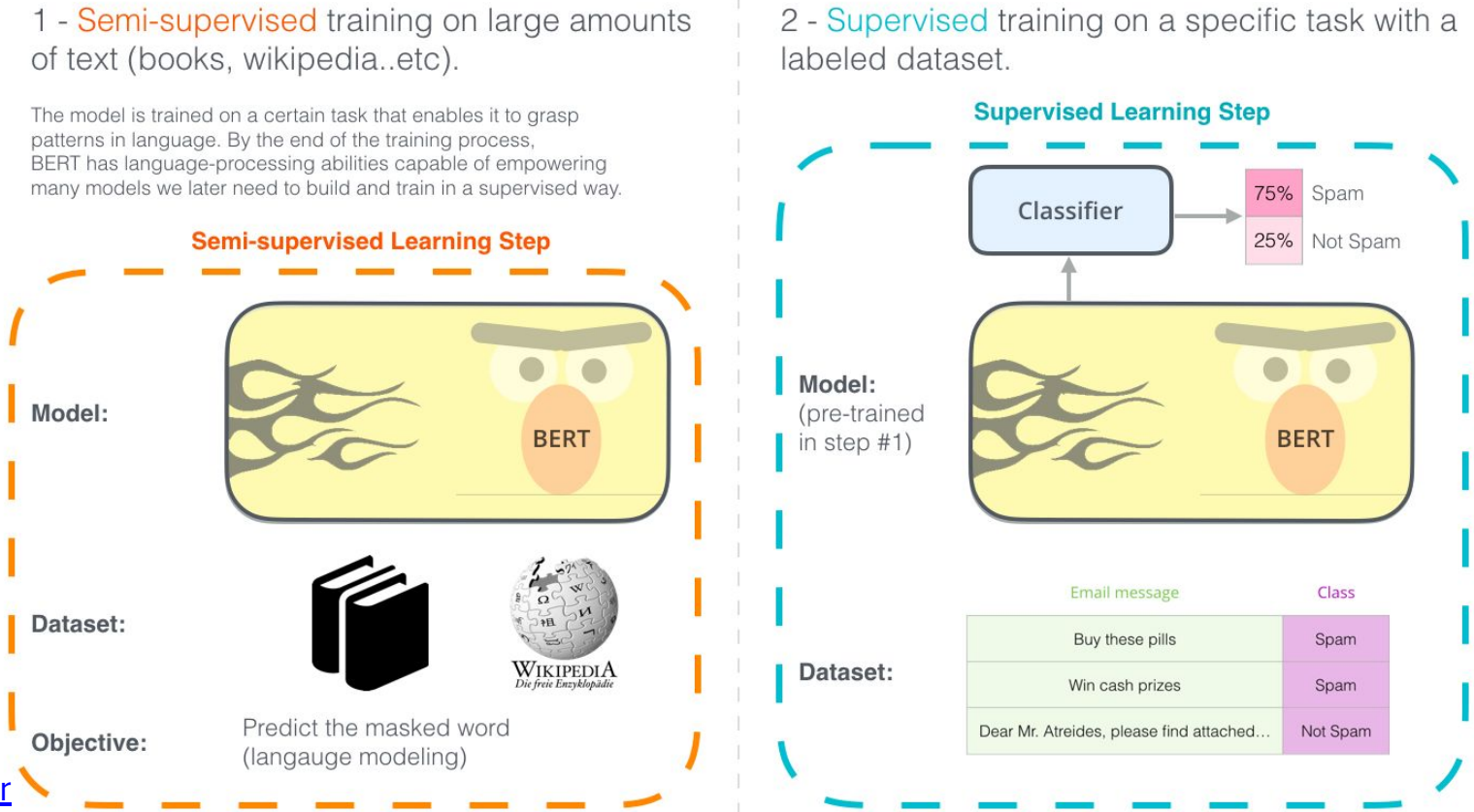
- Encoder-only transformer (*autoencoder*).
- Muy útiles para tareas de clasificación o generación de embeddings.
- No son modelos buenos para generar, no es su foco.
- Al no tener esta restricción, pueden ver todo el contexto de palabras.

# BERT




Fuente: Jay Alammar

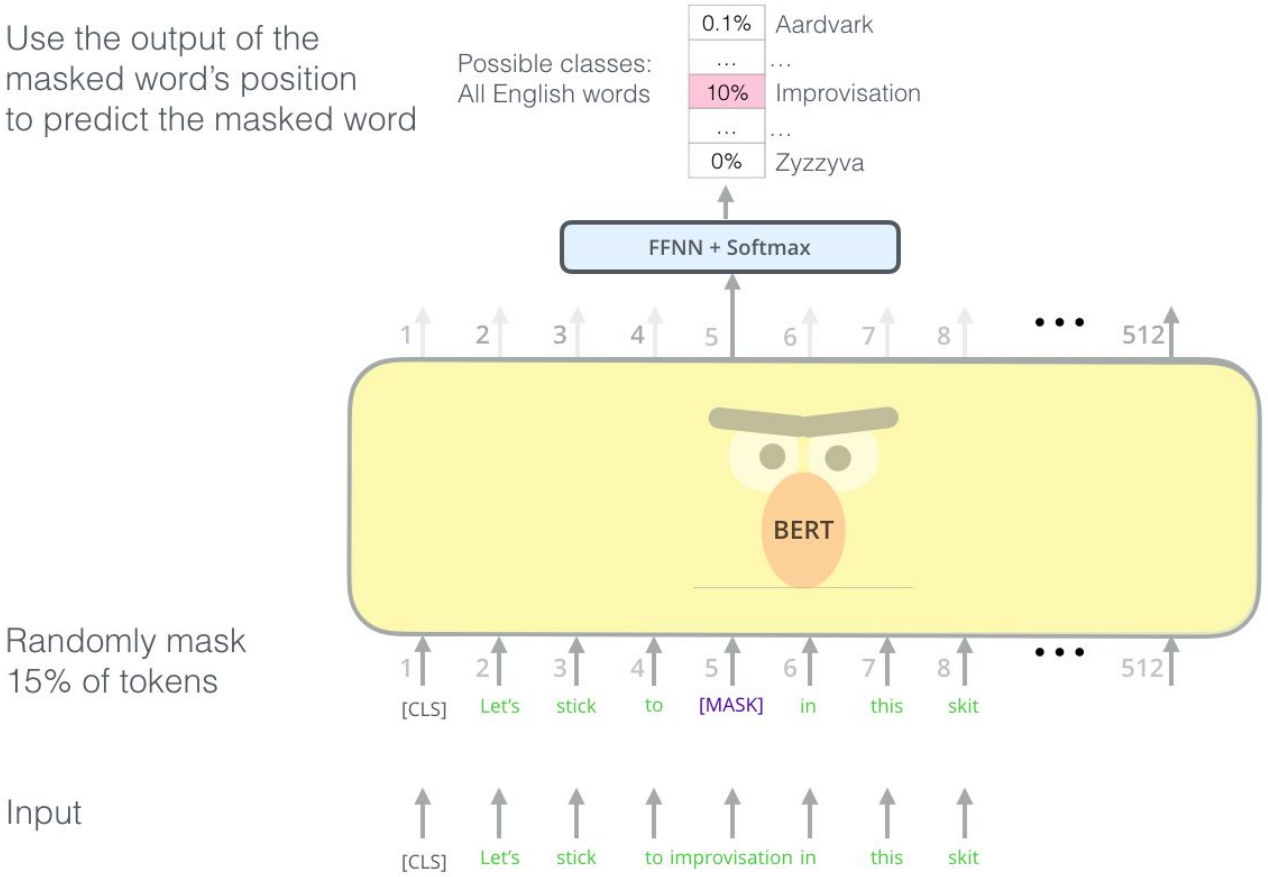
# BERT – Cómo fue entrenado – y cómo usarlo




[Fuente: Jay Alammar]

# BERT

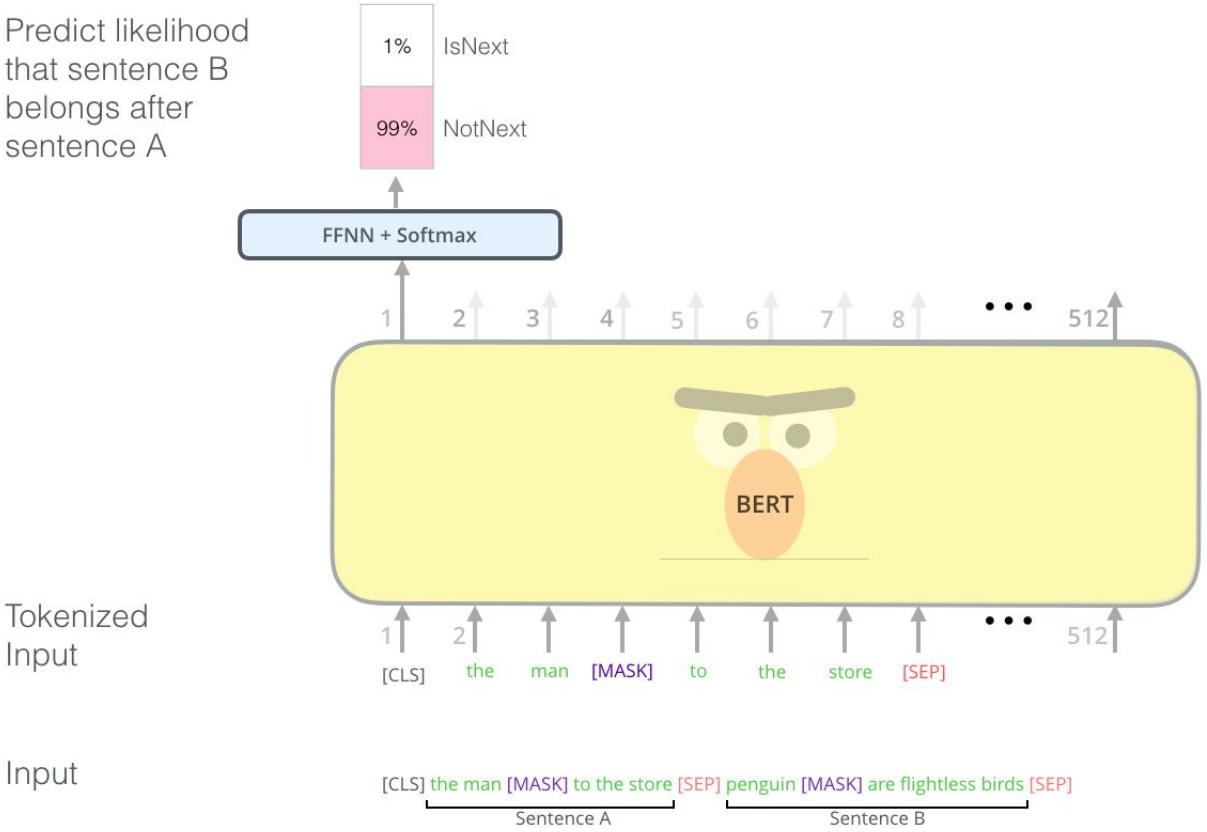
Cómo fue entrenado...




[Fuente: Jay Alammar]

# BERT

Cómo fue entrenado...

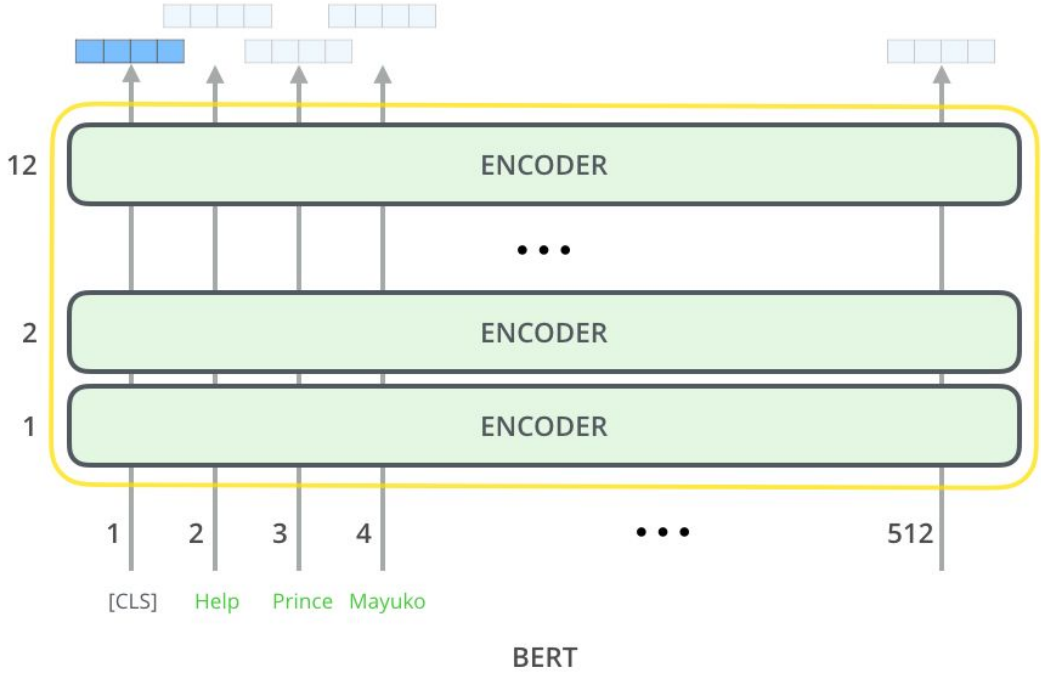



[Fuente: Jay Alammar]

# BERT

Por dentro, es un apilado de encoders que toman la concatenación de los embeddings de cada palabra.

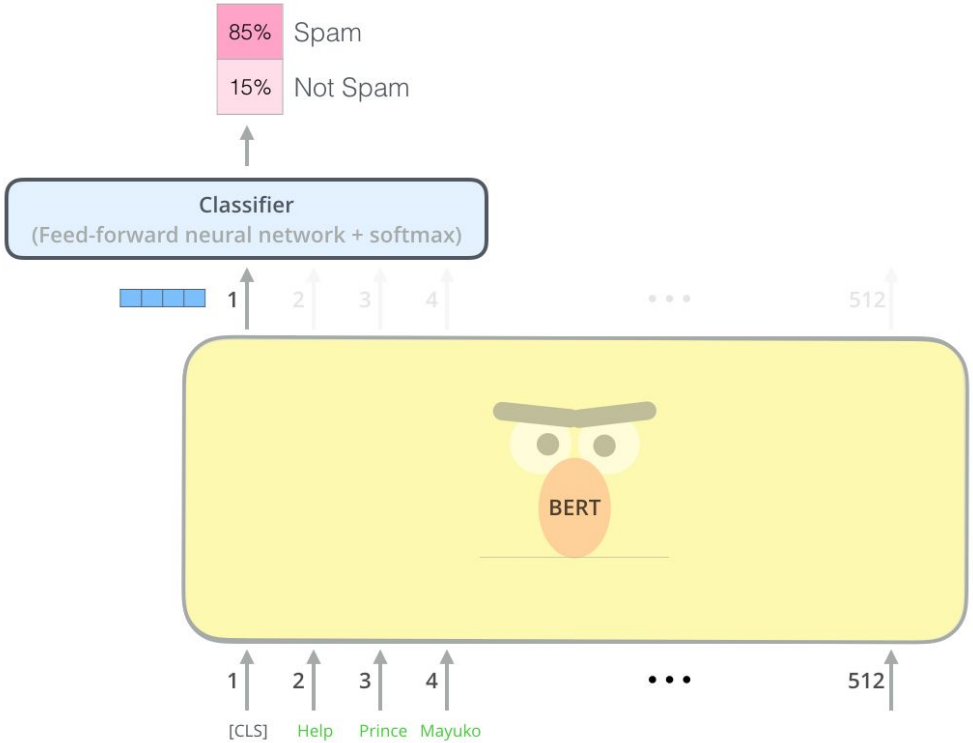
En este caso, se puede predecir un vector por cada entrada.




[Fuente: Jay Alammar]

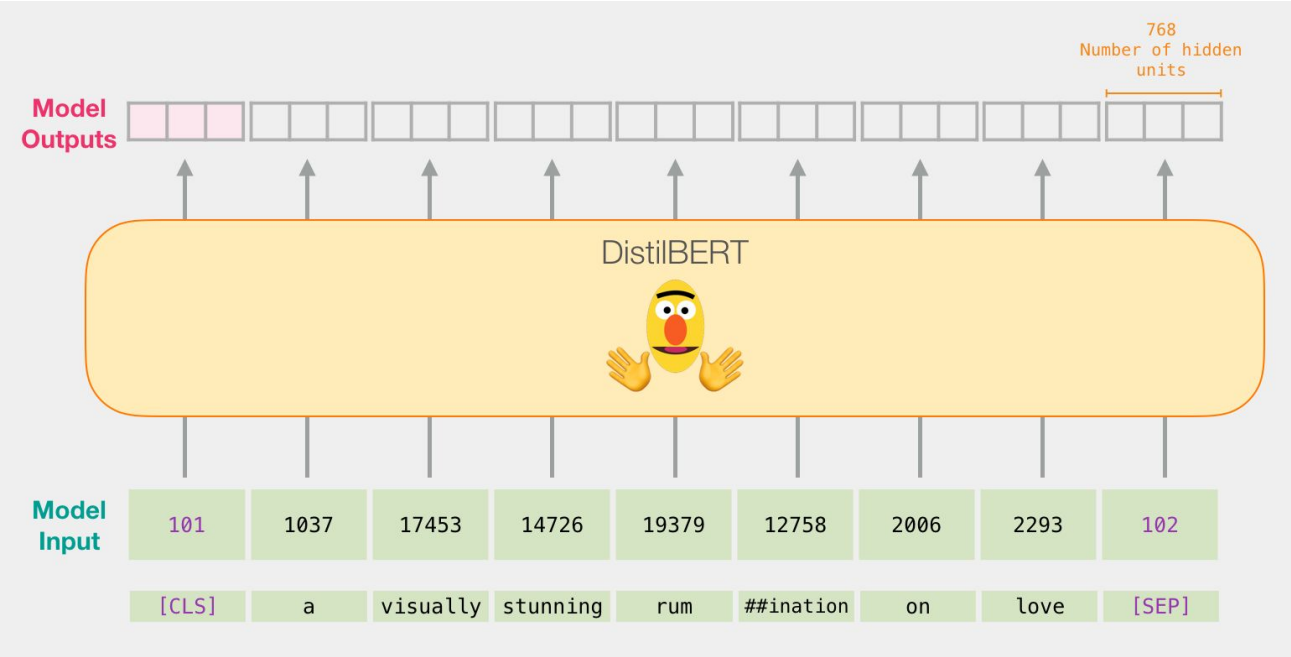
# BERT

O se puede utilizar sólo la salida del primer instante de tiempo para resolver  un problema de sequence classification




[Fuente: Jay Alammar]

# BERT – Cómo fluye una entrada




[Fuente: Jay Alammar]

# Embeddings con BERT vs Word2Vec

## BERT

- Ofrece embeddings en el contexto de la oración.
- Ej: embedding “banco” distinto en “me sentaré en el banco” que en “depositaré en el banco”.
- Toma en cuenta la posición de la palabra con los encodings posicionales.

## WORD2Vec

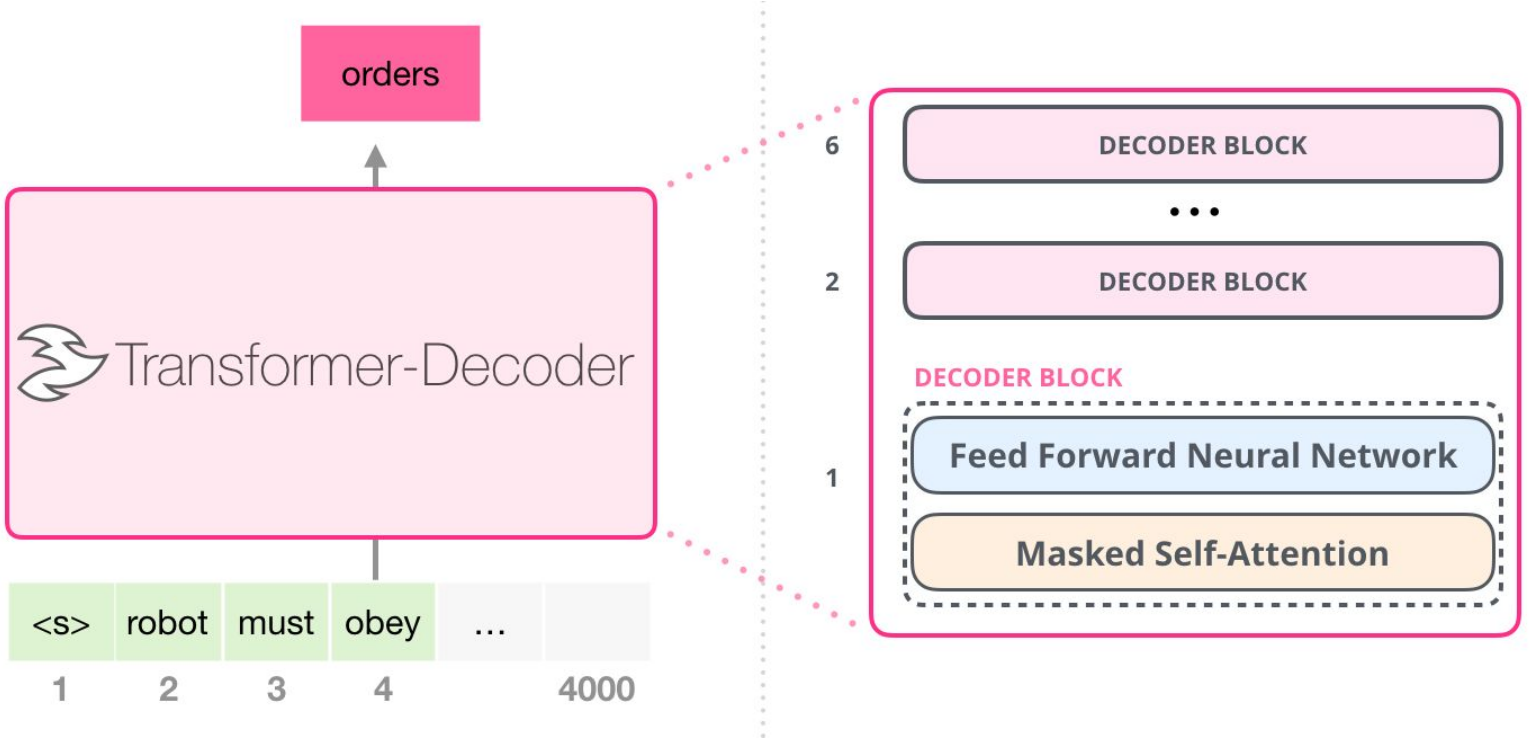
- Mismo embedding, en el contexto general.
- Ej: mismo embedding para “banco” sin importar si es asiento o entidad bancaria.
- Es agnóstico a la posición de la palabra.

Fuente: Aman AI

# GPT – Generative Pre-training

- Decoder-only transformer (*autoregressive*)
- Utiliza un apilado de decoders (ej. 12)
- A diferencia de encoders, tienen acceso sólo a palabras a la izquierda (o derecha, para el caso de idiomas con escritura de derecha a izquierda)
- Por esto, son modelos excelentes para generar. Pueden predecir, pero es más costoso

# GPT – Generative Pre-training




[Fuente: Jay Alammar]

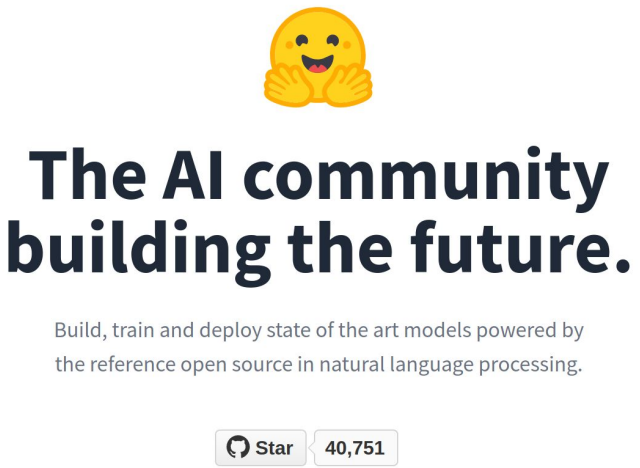
# Hugging Face Transformers

# Hugging Face: modelos, datasets y librerías

Proyecto open source que contiene modelos pre-entrenados para distintas áreas de PLN, además de datasets y librerías.

La iniciativa apunta a hacer accesible modelos de estado del arte para investigadores, estudiantes y practicantes de ML, así como ayudar a reducir la huella de carbono producto de tener que re-entrenar un modelo.

Hugging Face provee implementaciones propias de dichos modelos a partir de lo expuestos en sus respectivos artículos.



# Hugging Face: Transformers

Hugging face provee una librería llamada **transformers** que implementa los mecanismos de self-attention, además de habilitar el acceso a los modelos pre-entrenados desde sus servidores. Soportan **pytorch** y **tensorflow.**

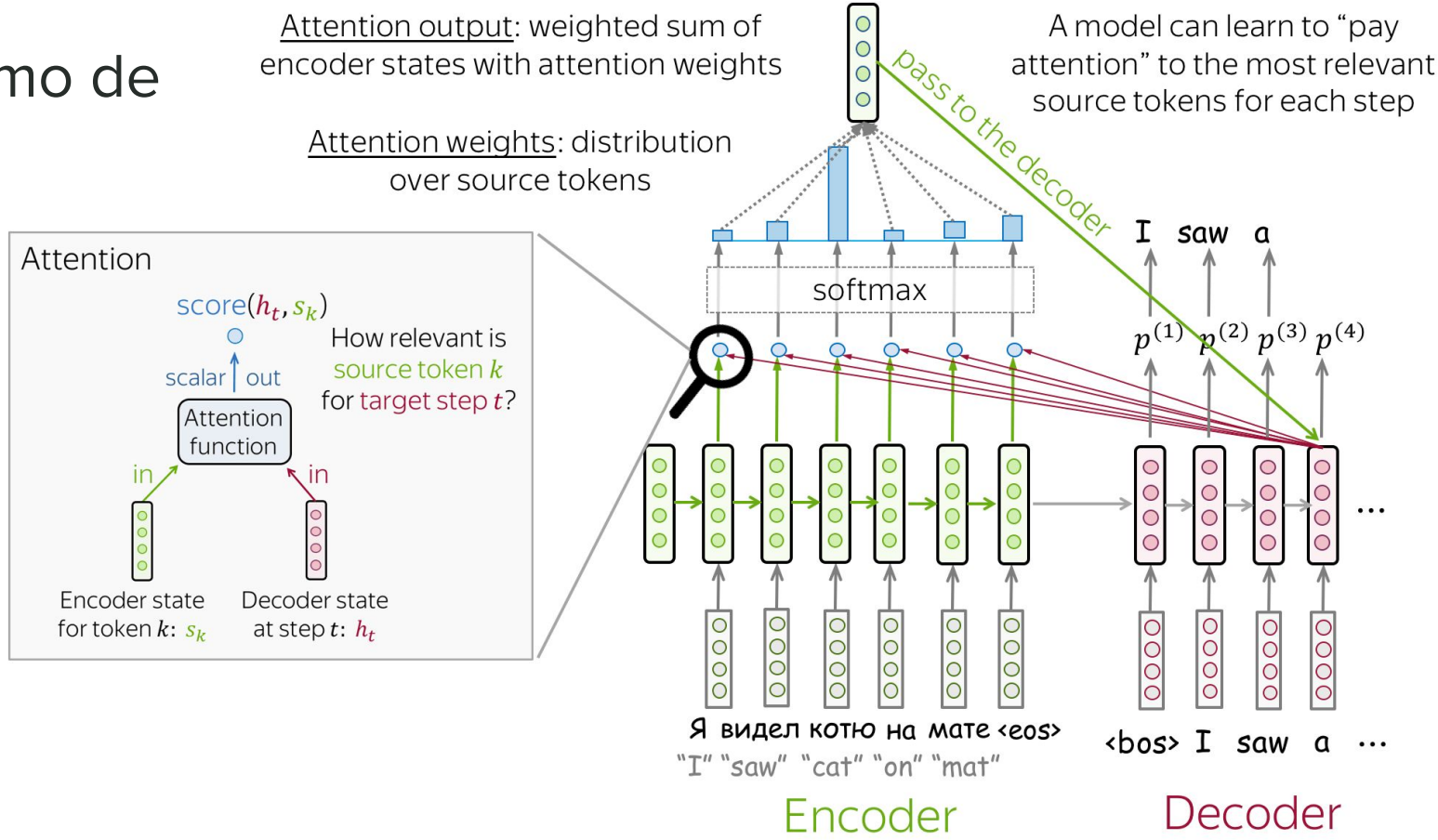
```
1 from transformers import AutoTokenizer, AutoModelForMaskedLM
2 tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
3 model = AutoModelForMaskedLM.from_pretrained("bert-base-uncased")
```

# LET’S CODE

## 8. Redes Transformers

# Apéndice: arquitectura en mayor detalle y material extra

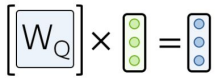
# Mecanismo de Atención




[Fuente: Lena Voita]

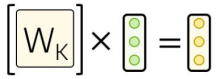
# Función de atención

Each vector receives three representations ("roles")



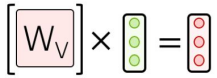
Query: vector **from** which the attention is looking

"Hey there, do you have this information?"



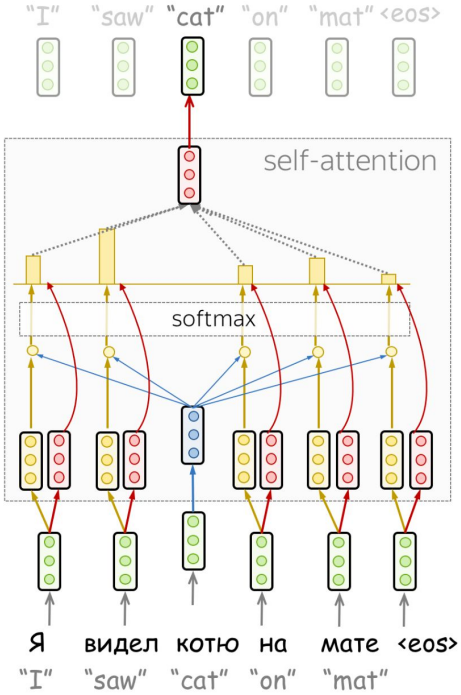
Key: vector **at** which the query looks to compute weights

"Hi, I have this information – give me a large weight!"



Value: their weighted sum is attention output

"Here's the information I have!"




Attention output: $c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \cdots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$

↑ (weighted sum)

"source context for decoder step $t$"

Attention weights: $a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \text{k} = 1..\text{m}$

↑ (softmax)

"attention weight for source token $k$ at decoder step $t$"

Attention scores: $\text{score}(h_t, s_k), \text{k} = 1..\text{m}$

↑

"How relevant is source token $k$ for target step $t$?"

Attention input: $s_1, s_2, \ldots, s_m$ (all encoder states) $h_t$ (one decoder state)

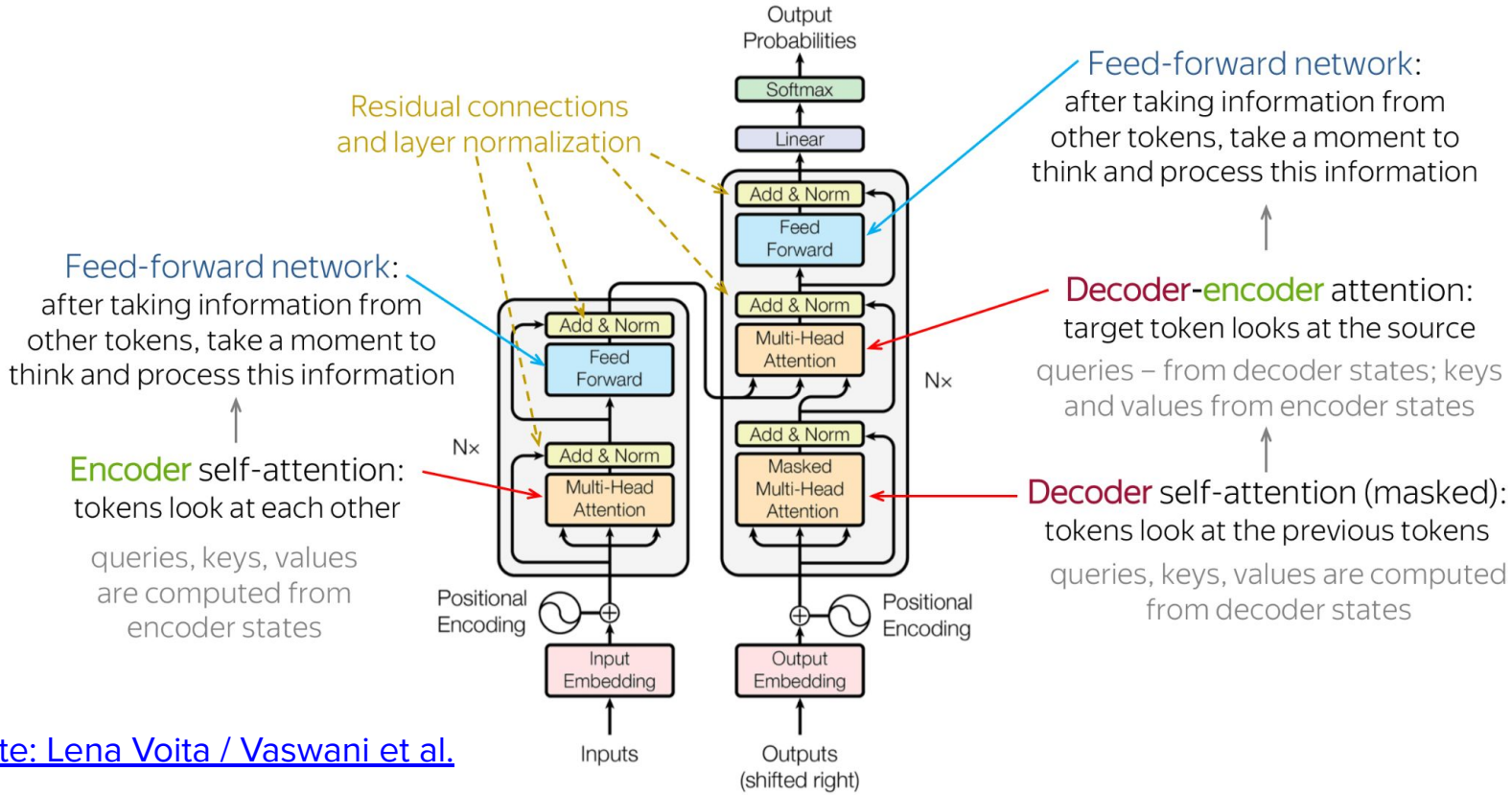
$$\overbrace{Attention(q, k, v) = softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)v}^{\text{Attention weights}}$$

from to

vector dimensionality of K, V

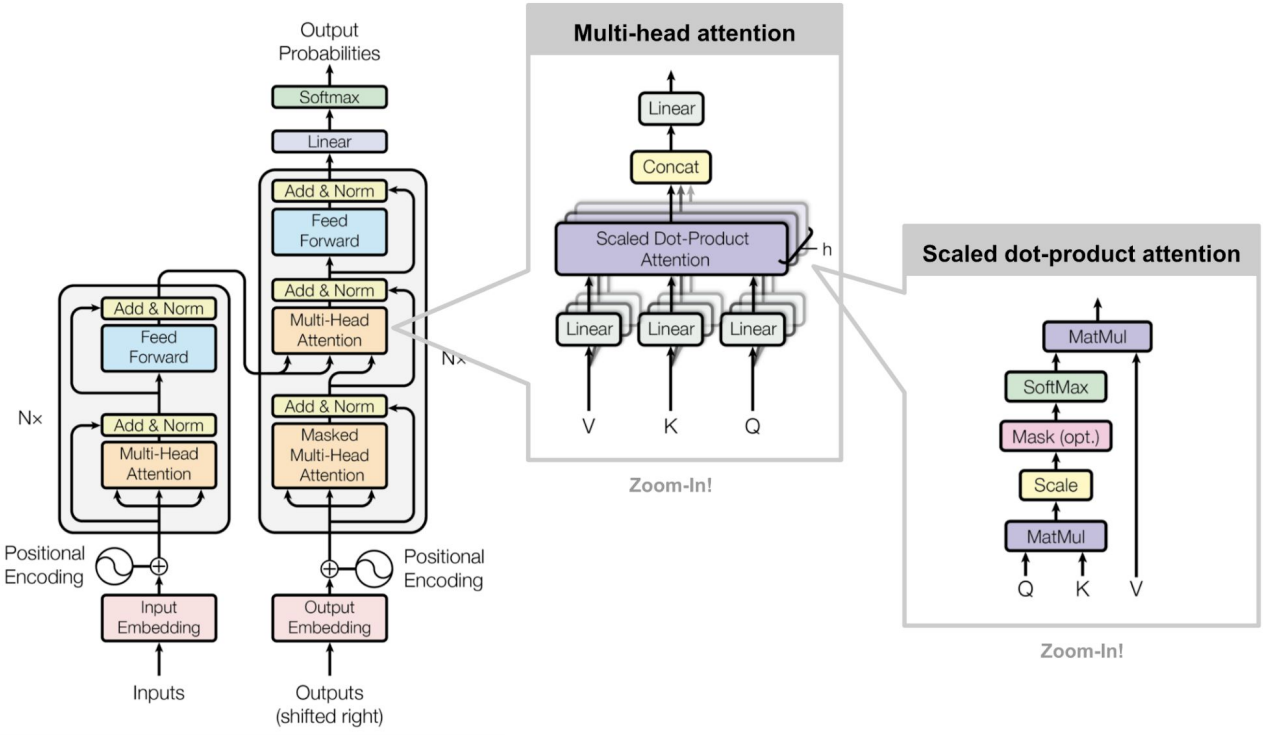
[Fuente: Lena Voita]

# Transformer – arquitectura completa




Fuente: Lena Voita / Vaswani et al.

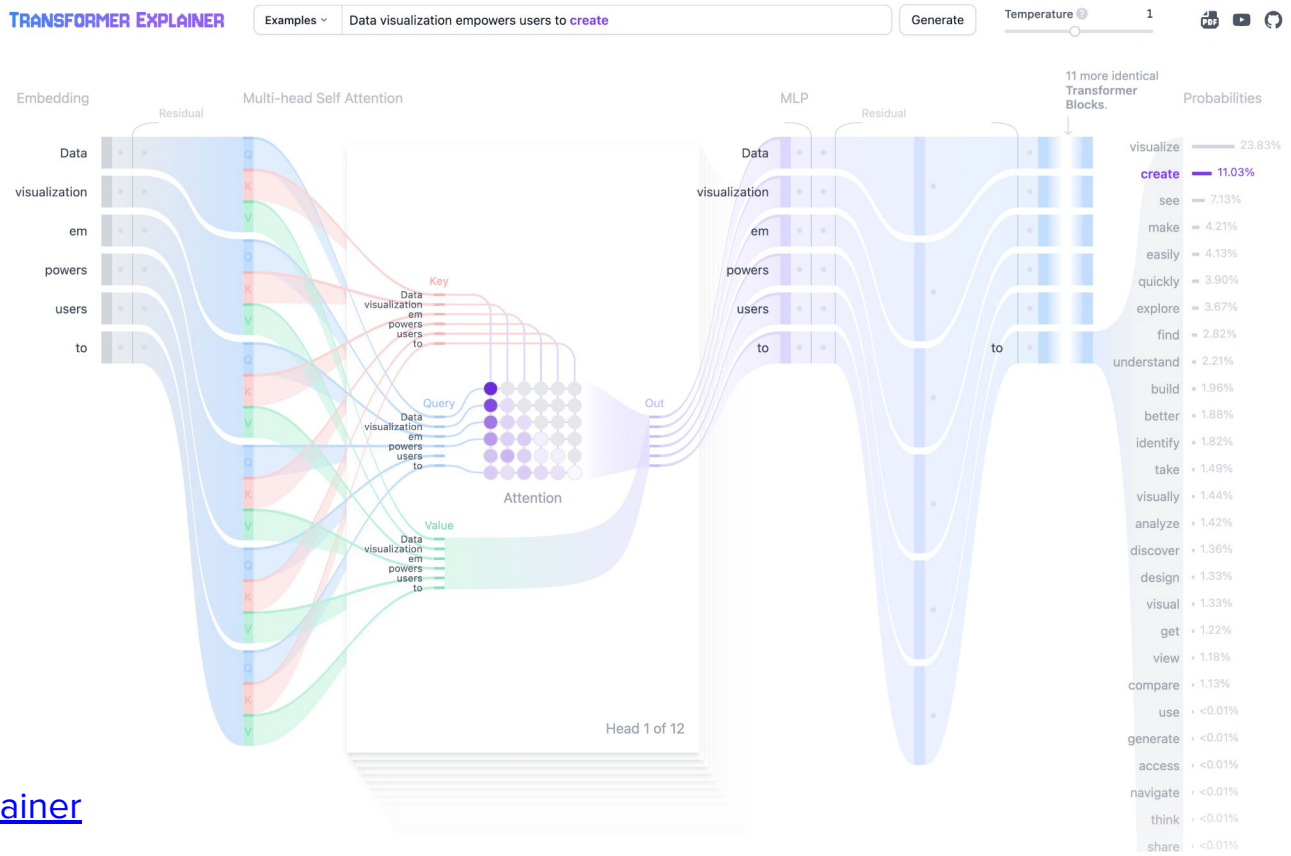
# Transformer – arquitectura completa




[Fuente: Lilian Weng / Vaswani et al.]

# GPT – Visualizado

Vemos desde
un visualizador




[Fuente: Transformer Explainer]

# Vision Transformers

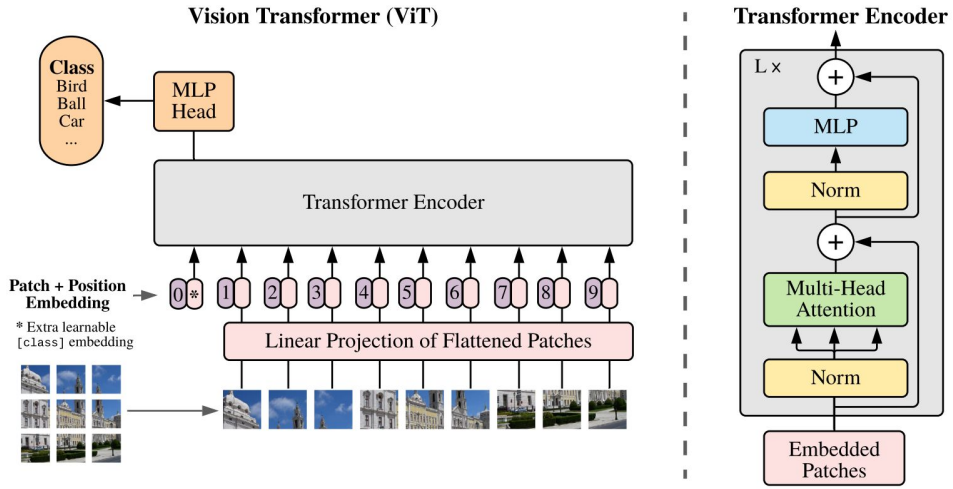



Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Fuente: Dosovitskiy et al. 2020

# Material extra

- Curso de NLP de Lena Voita: Transformers.
- Blogs ilustrados de Jay Alammar: Seq2seq, Transformers, BERT 1, 2, GPT-2 y GPT-3.
- Explicador de transformers: GPT-2.
- The Vision Transformer Model.