

LLMs y AI Generativa

Docentes:

Esp. Ing Abraham Rodriguez - FIUBA

Esp. Ing Ezequiel Guinsburg - FIUBA

Programa de la materia

1. Repaso de Transformers, Arquitectura y Tokenizers.
2. Arquitecturas de LLMs, Transformer Decoder.
3. Ecosistema actual, APIs, costos, HuggingFace y OpenAI.
4. MoEs, técnicas de prompts, evaluación de LLMs.
5. Modelos locales y uso de APIs.
6. RAG, vector DBs, chatbots y práctica.
7. Agentes, fine-tuning y práctica.
8. Generación multimodal.

Github de la materia

[Repositorio](#)

Algunas Tecnologías y herramientas

 PyTorch






 **Hugging Face**

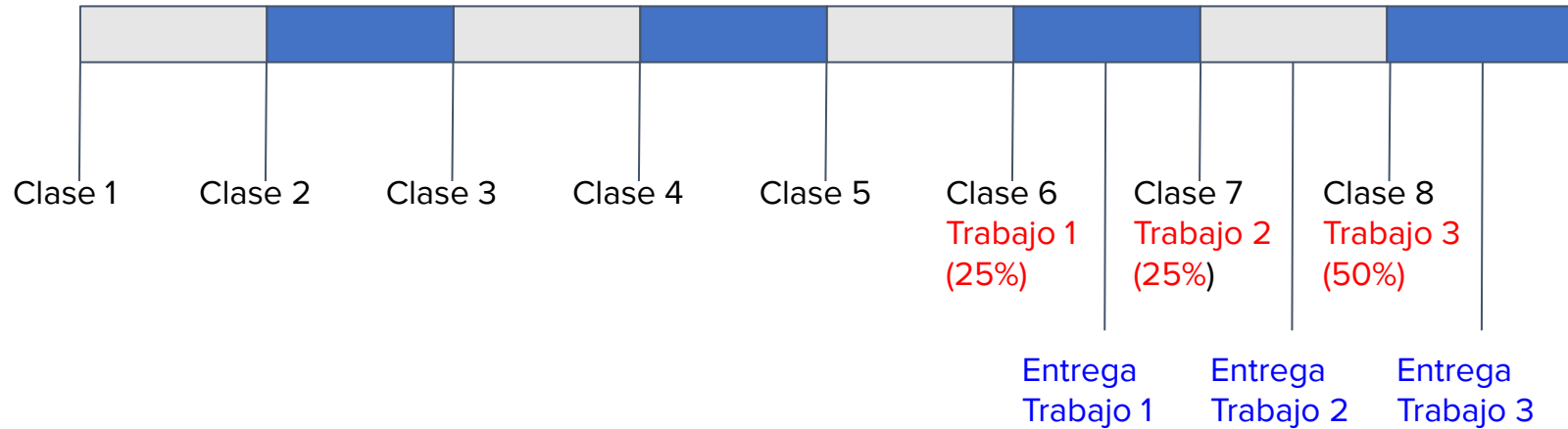
 **Chroma**



Cronograma de la materia (tentativo)

	1:20H	20M	1:20H
Teoría			
Break			
Teoría/Práctica			

Evaluación

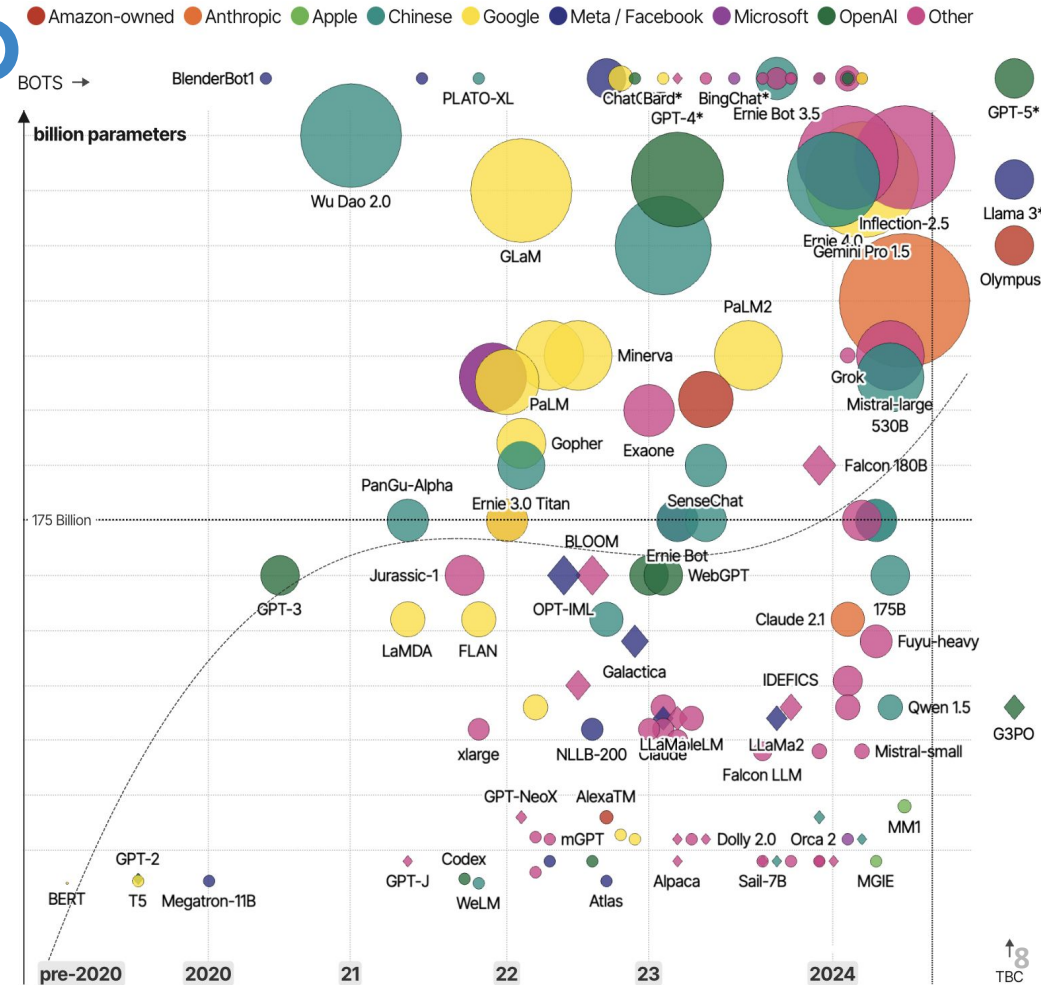


CLASE I (Transformers y Tokenizers)

LÍNEA DE TIEMPO

The Rise and Rise of A.I. Large Language Models (LLMs)

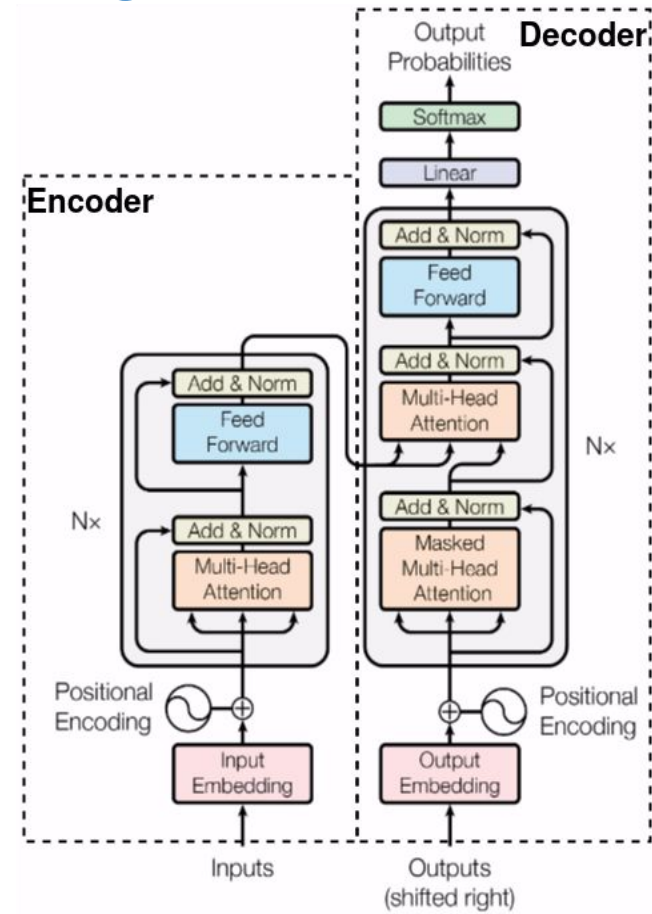
Todo empezó gracias al paper:
Attention is all you need (2017)



Arquitectura del Transformer

Propuesto por Vaswani et al. en su paper "[Attention Is All You Need](#)" (2017), es un modelo de arquitectura neural diseñado principalmente para tareas de procesamiento de lenguaje natural (NLP).

Su innovación clave es el uso del **mecanismo de atención**, que permite al modelo **enfocarse** en diferentes partes de la entrada de manera dinámica y eficiente, sin depender de la estructura secuencial de modelos como las redes recurrentes (RNNs) o LSTMs.

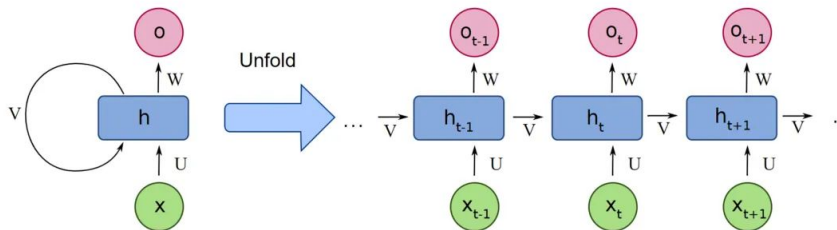


Transformer vs RNNs

LSTM is dead (Video)

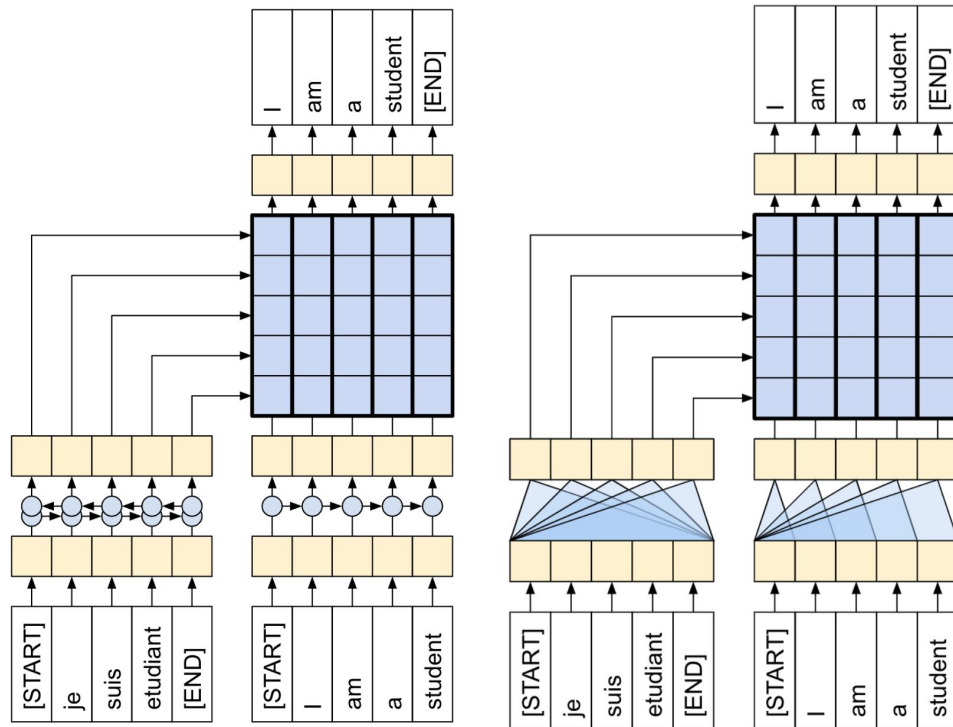
Advantages of Transformers

<https://www.tensorflow.org/text/tutorials/transformer>



The RNN+Attention model

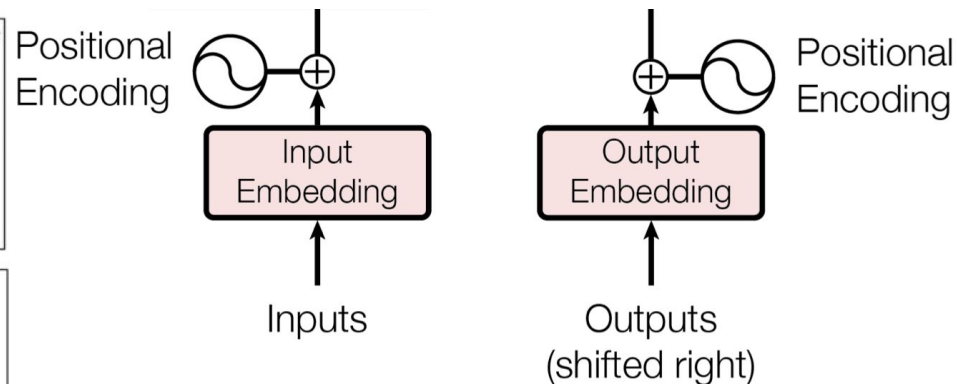
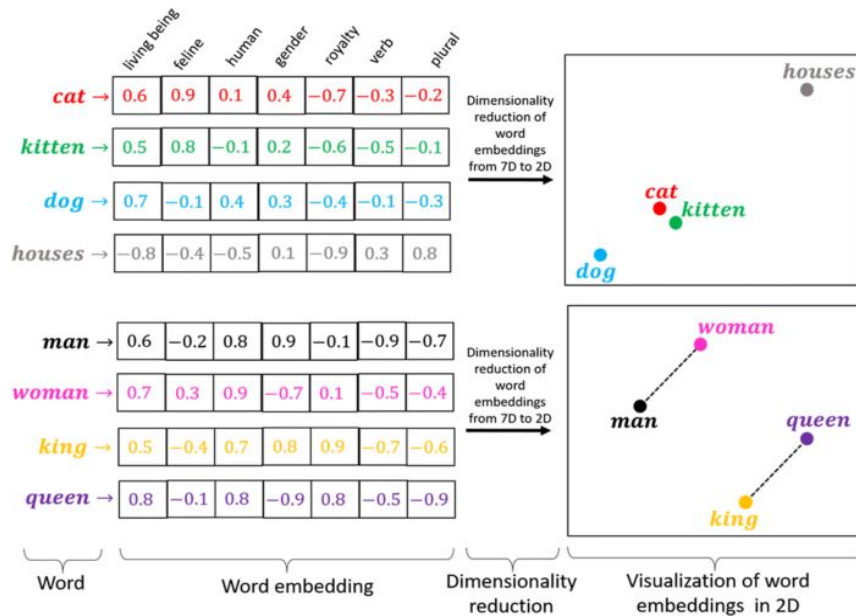
A 1-layer transformer



Transformers Embeddings

Embeddings: Los tokens se transforman en vectores y se convierten en embeddings. Estos embeddings son vectores de un espacio latente permitiendo la relación entre tokens.

Codificación Posicional (Positional Encoding): Los Transformers desconocen el orden secuencial. Se añaden embeddings posicionales para que el modelo sepa la posición de cada token dentro de la secuencia.

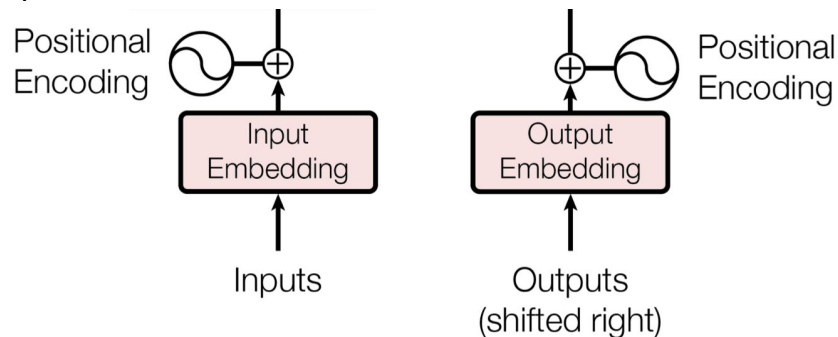


Positional Embeddings

El Transformer por naturaleza **desconoce el contexto espacial** de los datos de entrada. En NLP se sigue el orden de tokens de entrada siendo necesaria la información posicional de los tokens.

Sequence	Index of token, k	Positional Encoding Matrix with d=4, n=100			
		i=0	i=0	i=1	i=1
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'



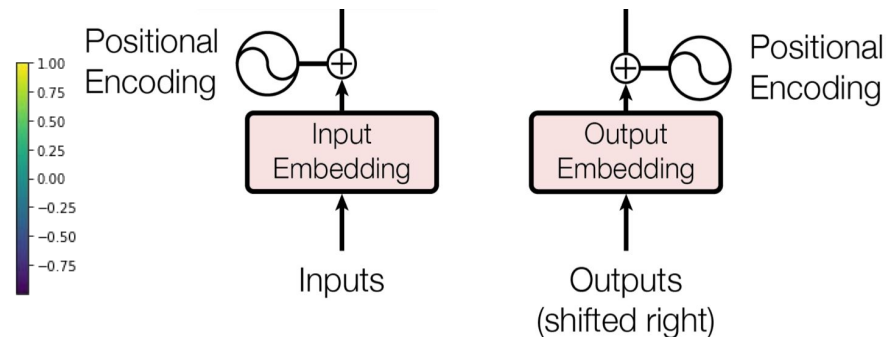
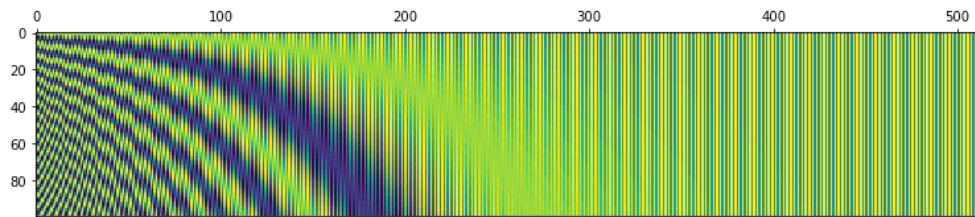
$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Positional Embeddings

Positional Embeddings in Transformers Explained

Gentle introduction to positional encoding in transformer models



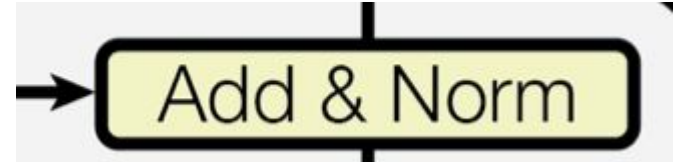
$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Transformers LayerNorm

Suma y Normalización (Add & Norm): Se realiza una operación de suma y normalización en los resultados para **estabilizar y mejorar** el aprendizaje. Esto estabiliza y optimiza la convergencia durante entrenamiento.

[Torch LayerNorm](#)

[Attention-is-all-you-need-layer-norm](#)

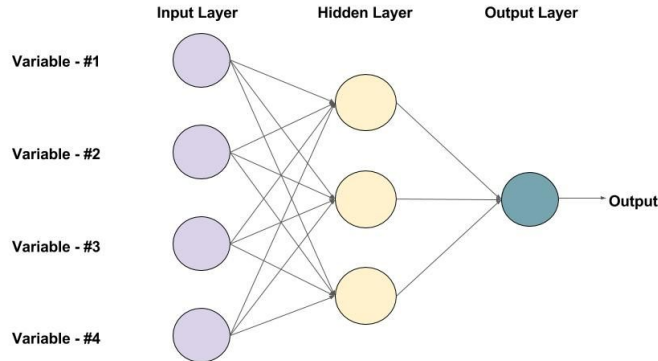


$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

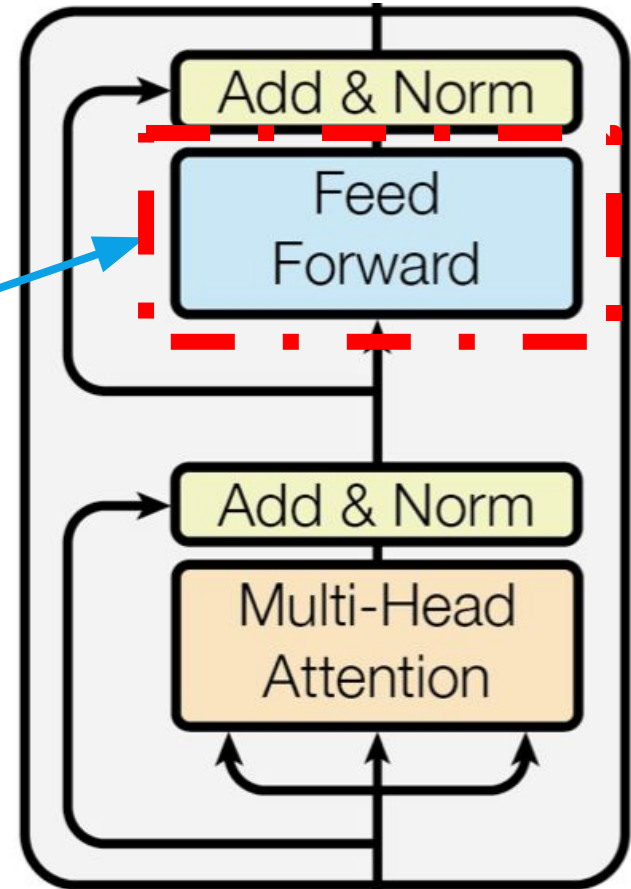
Transformers FeedForward

Red Lineal Feed-Forward (Feed Forward): Cada capa de atención es seguida por una **red lineal**, que extrae características complejas internas e introduce no-linearidad.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)



The Annotated Transformer

[The annotated Transformer](#) es una de las primeras guías de implementación y explicativas acerca del Transformer.

Recomendable leer antes de la clase II.

Mecanismo de Atención en Transformers

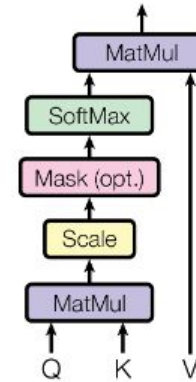
Scaled Dot-Product Attention: Es el mecanismo de atención que se utiliza para calcular la atención en base a los vectores Q (Queries), K (Keys), y V (Values).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

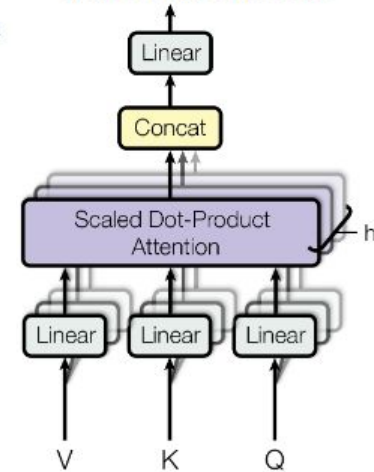
Multi-Head Attention: Es una extensión del mecanismo de atención que aplica múltiples capas de atención en paralelo. Esto implica realizar proyecciones lineales de los vectores de entrada en varios subespacios, cada uno alimentando una capa de atención independiente.

Los resultados de estas capas se concatenan y se proyectan nuevamente a una dimensión reducida, permitiendo que el modelo capture diferentes aspectos de las relaciones entre elementos en la secuencia.

Scaled Dot-Product Attention



Multi-Head Attention

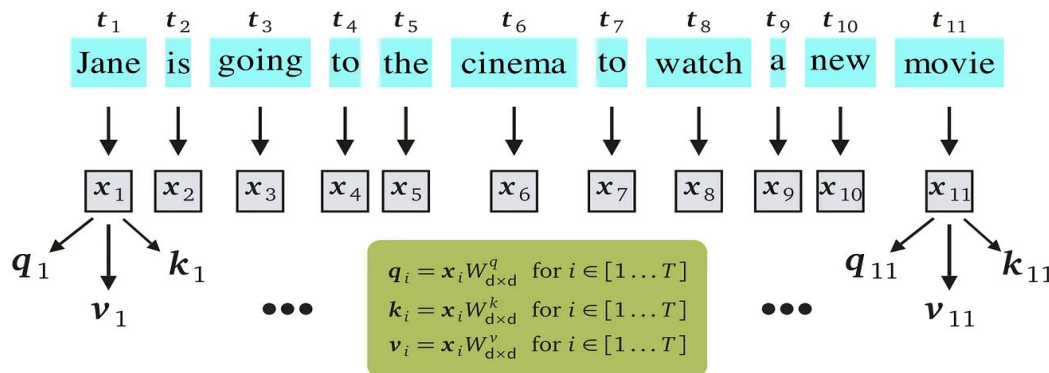


Scaled Dot-Product Attention

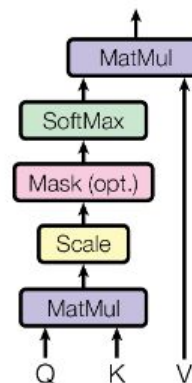
El mecanismo de atención consta de operaciones simples (multiplicación matricial, escalamiento y máscaras). Sin embargo es computacionalmente costoso $O(N^2)$.

Formalmente los vectores son:

- Q y $K \in \mathbb{R}^{dk}$
- $V \in \mathbb{R}^{dv}$

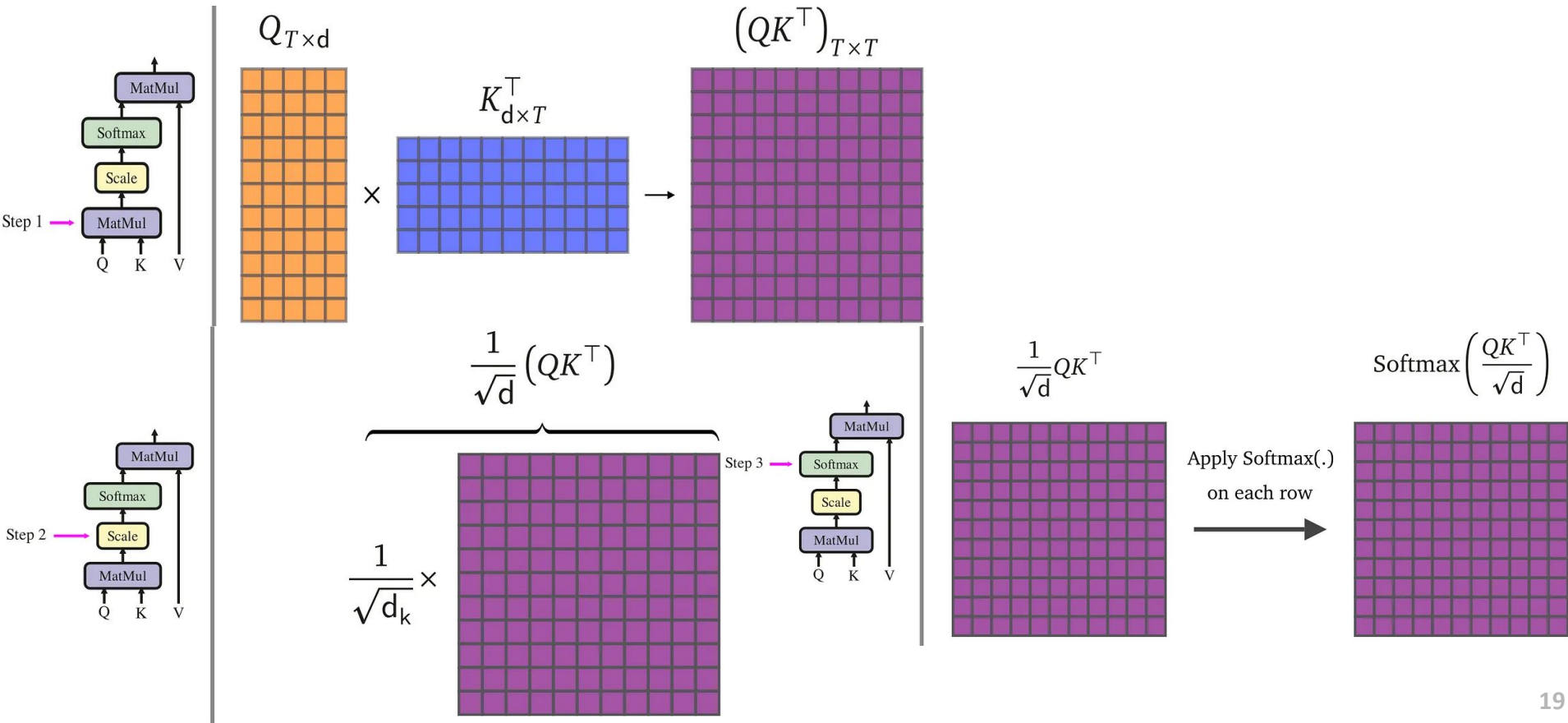


Scaled Dot-Product Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

Scaled Dot-Product Attention

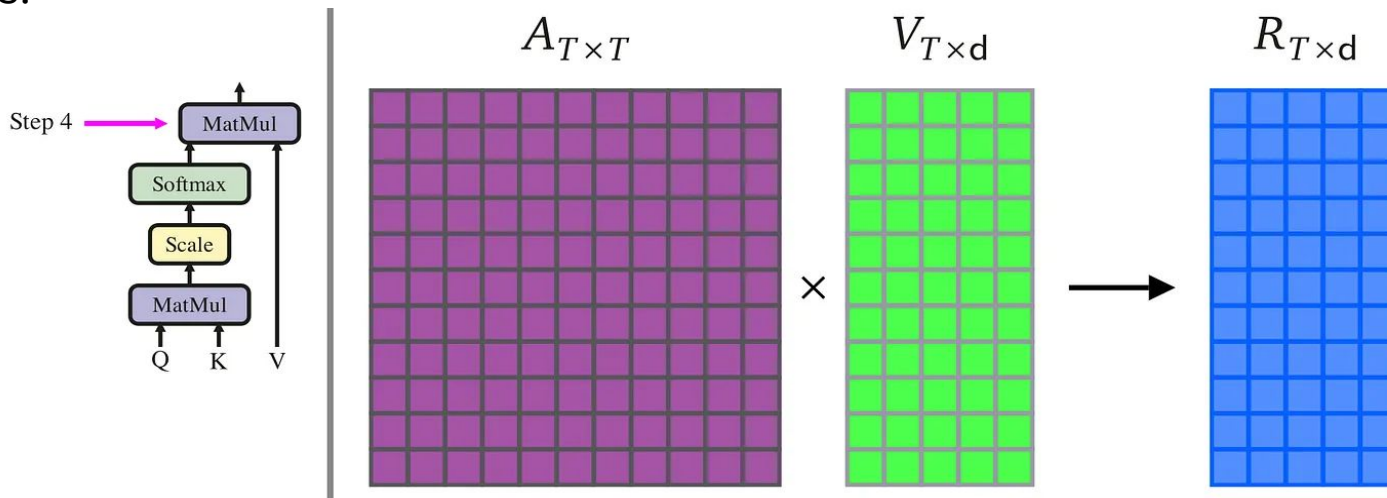


Scaled Dot-Product Attention

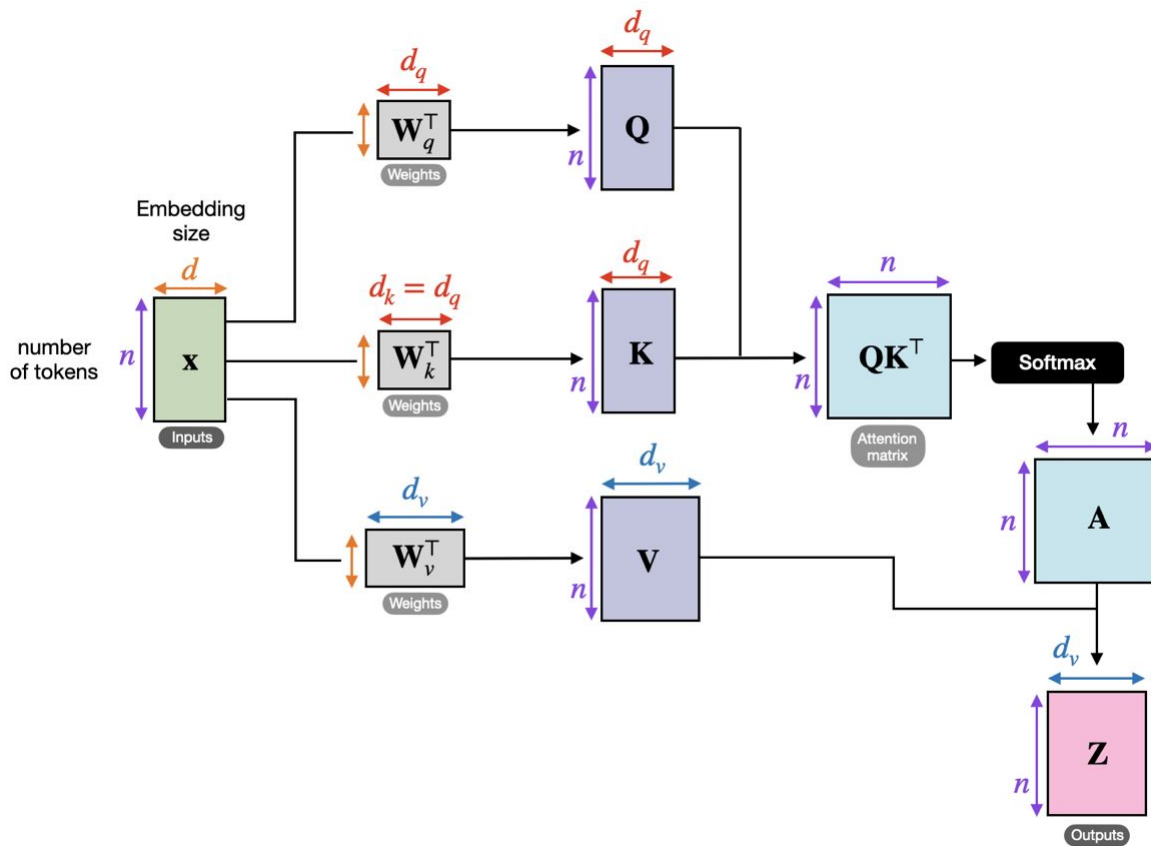
El resultado de la multiplicación, escalamiento y softmax entre las matrices Q y K dan como resultado a la **matriz A o matriz de Atención**.

La **matriz $A \times V = R$** , donde **R es la matriz de Contexto**.

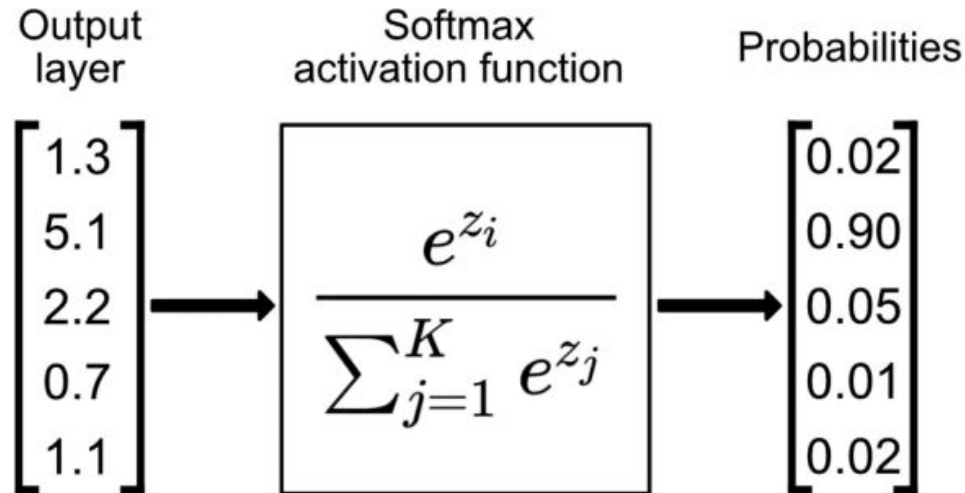
R es la salida de Attention, cada fila corresponde a la **correlación** entre tokens.



Scaled Dot-Product Attention



Escalamiento y Softmax



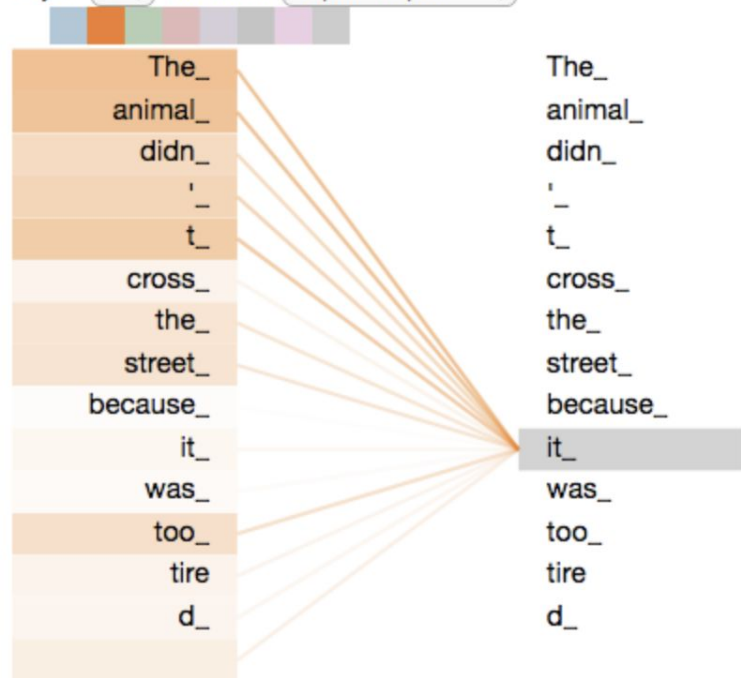
Si la dimensión d_k es grande, entonces $Q \cdot K$ puede dar como resultado valores muy altos. Eso hace que **softmax** tenga gradientes muy pequeños, dificultando el aprendizaje.

Los valores grandes pueden hacer que la *softmax* sea excesivamente “confiada” y puede desestabilizar el entrenamiento.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

Visualizando Self-Attention

Layer: 5 Attention: Input - Input



<https://jalammar.github.io/illustrated-transformer/>

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

Multihead Attention (MHA)

h es la cantidad de “heads” o bloques de Attention en paralelo, esto permite obtener el contexto de todos los elementos dentro de una secuencia en paralelo. En este caso, V, K y Q son proyectados linealmente a las matrices:

$$W_i^q \in \mathbb{R}^{d_{model} \times d_k}$$

$$W_i^k \in \mathbb{R}^{d_{model} \times d_k}$$

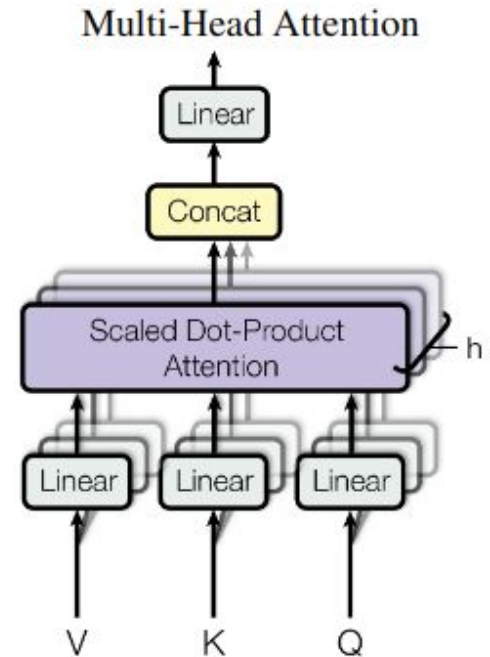
$$W_i^v \in \mathbb{R}^{d_{model} \times d_v}$$

Por ultimo el resultado de cada “head” es concatenado y proyectado a:

$$W^o \in \mathbb{R}^{h d_v \times d_{model}}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h) W^o$$



Guías sobre Self-Attention

[Self-Attention From Scratch](#)

[Attention and Self-Attention for NLP](#)

Veamos el Transformer

LLM Visualization

Tokenizers

Tokenizers

La tokenización es la descomposición de texto a palabras, letras, etc. Permite representar de manera compacta un texto.

En Transformers, se utiliza la vectorización de tokens para crear un espacio vectorial que define un vocabulario.

Ejemplo con [tiktoken](#).

Tokenizers

BPE (Subword)

WordPiece

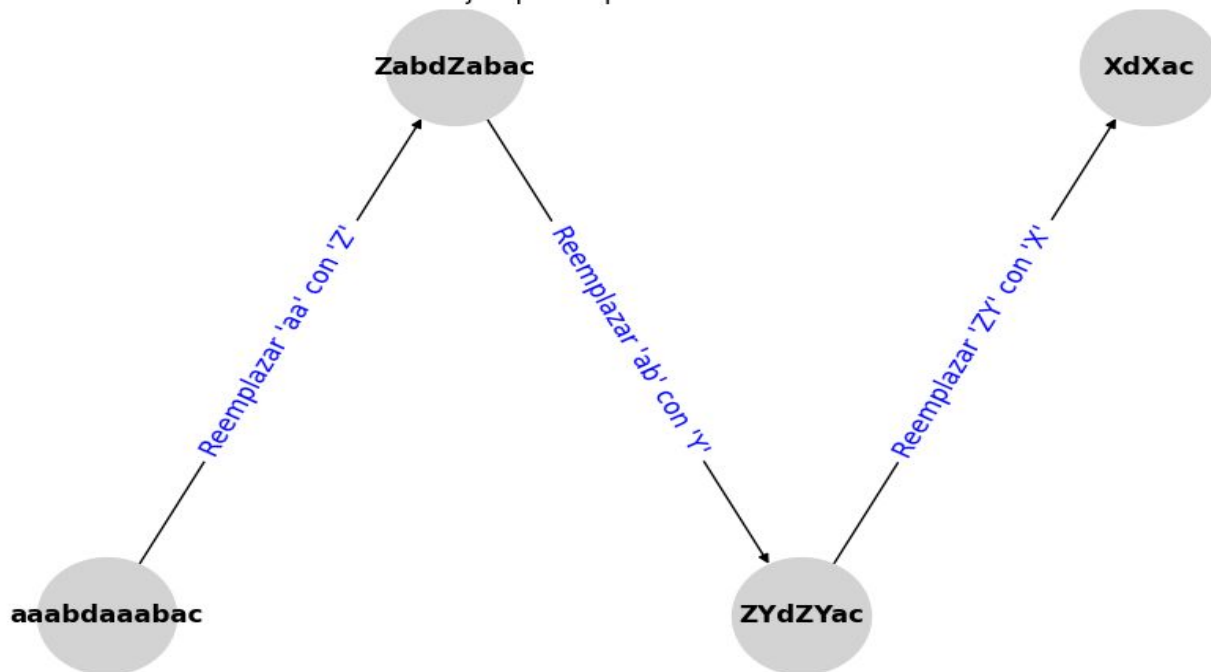
Unigram

[Guia de Huggingface](#)

Byte Pair Encoding (BPE)

El tipo de tokenizer más utilizado en LLMs es BPE.

Ejemplo de proceso de BPE



```
bpe = tokenizer.BPE()
```

```
encoded = bpe.encode("hello")
```

Output: [104, 101, 108, 108, 111]

^ ^ ^ ^ ^
h, e, l, l, o

Implementación de
ejemplo.

Byte Pair Encoding (BPE)

Aplicado por primera vez en 2015 en el paper [Neural Machine Translation of Rare Words with Subword Units](#) para compression de palabras

Algorithm 1 Learn BPE operations

```
import re, collections
```

```
def get_stats(vocab):  
    pairs = collections.defaultdict(int)  
    for word, freq in vocab.items():  
        symbols = word.split()  
        for i in range(len(symbols)-1):  
            pairs[symbols[i], symbols[i+1]] += freq  
    return pairs
```

```
def merge_vocab(pair, v_in):  
    v_out = {}  
    bigram = re.escape(' '.join(pair))  
    p = re.compile(r'(?!\S)' + bigram + r'(?!\S)')  
    for word in v_in:  
        w_out = p.sub(' '.join(pair), word)  
        v_out[w_out] = v_in[word]  
    return v_out
```

```
vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,  
         'n e w e s t </w>':6, 'w i d e s t </w>':3}  
num_merges = 10  
for i in range(num_merges):  
    pairs = get_stats(vocab)  
    best = max(pairs, key=pairs.get)  
    vocab = merge_vocab(best, vocab)  
    print(best)
```

Limitantes de la tokenización

La tokenización es un proceso iterativo, consume muchos recursos computacionales, y **no necesariamente converge** al resultado más óptimo, por lo cual es común entrenar múltiples veces. GPT-2 utilizó 50272 iteraciones.

El paso de entrenar un tokenizador, es una molestia para el desarrollo de LLMs, y es una de las razones por la cual hay diferencias de calidad de respuesta en varios modelos.

Se suele utilizar tokenizadores preentrenados, los más comunes son:

- GPT-2
- Llama 2
- BERT

Limitantes de la tokenización con Python

Python **no** es apto para realizar algoritmos complejos como BPE, por lo tanto el estándar es utilizar lenguajes más optimizados como Rust, Go o C y realizar bindings a Python.

[Tiktoken Github](#) (Rust)

[HuggingFace Tokenizers](#) (Rust)

[Weaviate Tiktoken](#) (Go)

[PyRusToken](#) (Rust, implementación propia)

[Counting words with Tiktoken example](#)

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Limitantes del Transformer (entrenamiento)

Los Transformers son computacionalmente costosos, requieren de Hardware especializado para lograr entrenar un modelo relativamente poderoso, GPT-3 (obsoleto) fue entrenado con un cluster de 1024 GPUs.

La mayor limitante del Transformer es MHA por su naturaleza $O(N^2)$, este ha sido sustituido por variantes como:

- [FlashAttention](#) (estandar),
- [Sliding-Window Attention \(SWA\)](#) (Mistral 7b),
- [Grouped Query Attention \(GQA\)](#) (LLama 2).

Algunas Optimizaciones en Transformers

Quantization (int4, int8...)

Variantes de Attention (FlashAttention)

Cache (KV-Cache)

Low Rank Projections (LoRA, QLoRA, Galore)

Detalles sobre Transformers

Pasos para preentrenar

El preentrenamiento es la etapa más difícil y costosa de realizar, hoy en día consta de:

- Aplicar las optimizaciones mencionadas (Rápida convergencia y menor uso de recursos)
- Obtener un dataset y utilizar dataloaders optimizados.
- Evaluación con GLUE, ROUGE, BLEU y benchmarks como SWAG, MMLU.
- Exportación de weights a formato eficiente (Safetensors)

GPT ≠ ChatBot

Un GPT preentrenado no realiza la tarea de QA (Question/Answer) en su lugar, solo completa texto por ejemplo:

La clase de AI generativa y LLMs es “una gran materia”

Completado por un GPT preentrenado

La tarea del modelo es **completar con n-tokens**.

QA GPT (Modelos de Chat)

Una vez realizados los pasos de preentrenamiento, para obtener un chat, se debe realizar fine-tuning sobre el modelo con datasets como:

- [Ultrachat](#)
- [Alpaca](#)

La función de pérdida es **cross-entropy**.

La estrategia de entrenamiento es comúnmente [Supervised Fine-tuning](#).

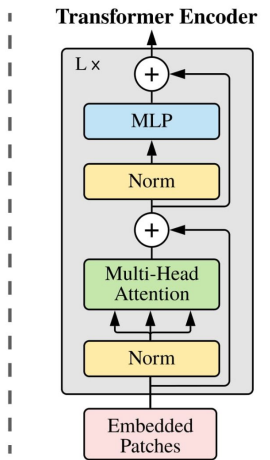
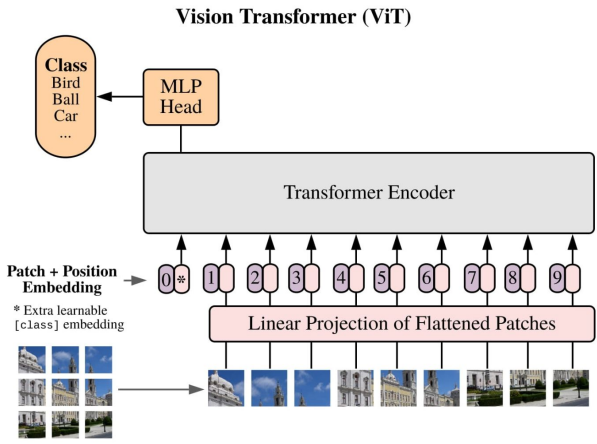
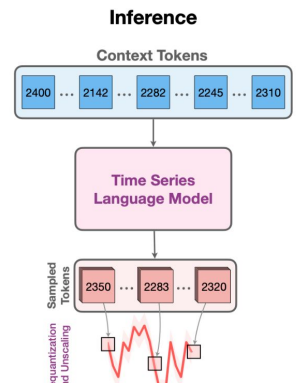
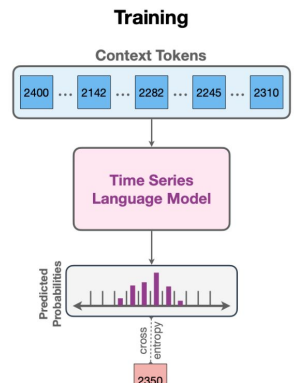
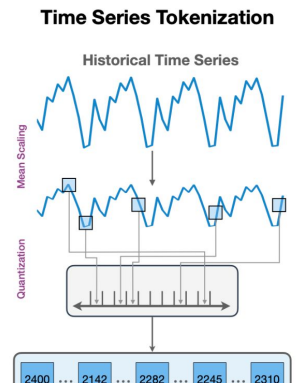
La evaluación es mediante **perplejidad** e interacción humana.

Durante el fine-tuning es buena práctica aplicar las optimizaciones mencionadas anteriormente.

Este paso requiere menos recursos computacionales que preentrenamiento.

Más allá de NLP

El Transformer está presente en visión artificial, análisis de series, audio, etc.



La arquitectura se mantiene **invariante**, el truco se encuentra en las **secuencias de entrada**.

Algunos Modelos del Estado del Arte

- ChatGPT-4
- Llama
- Mistral
- Qwen
- Gemma

Búsqueda de Papers

[Hugging Face Daily Papers](#)

[ArXiv](#)

[Meta Research...](#)

Preguntas?