ecnológico
Monterrey

NLP 2

Inteligencia artificial avanzada para la ciencia de datos II Modulo 5 NLP 2

transformers

Transformer

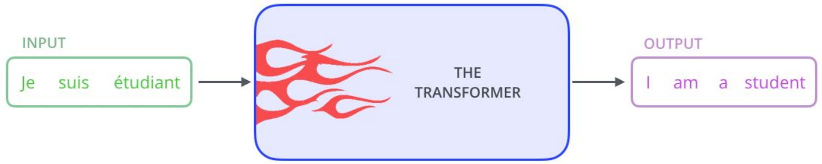
Transformers reuse the model encoder/decoder from seq2seq architecture.

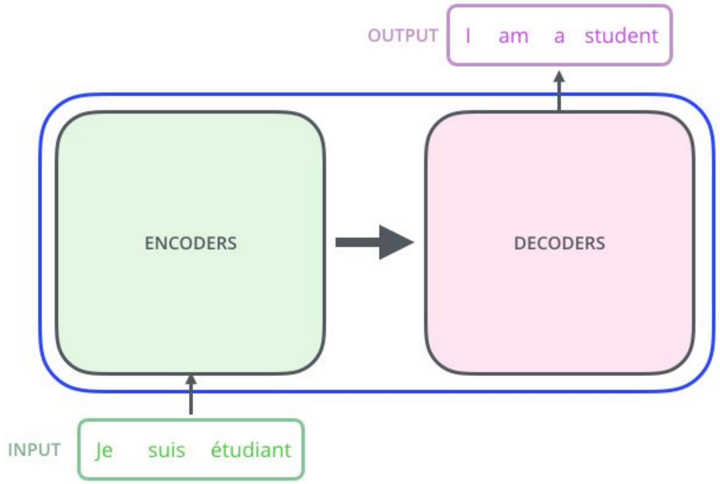
Seq2seq propose two networks (encoding and decoding) and an intermediate state with fix length (vector context) or dynamic (attention layer).

Transformer brings a new paradigm in 2017 with the paper

“Attention is all you need” by Google

https://arxiv.org/abs/1706.03762

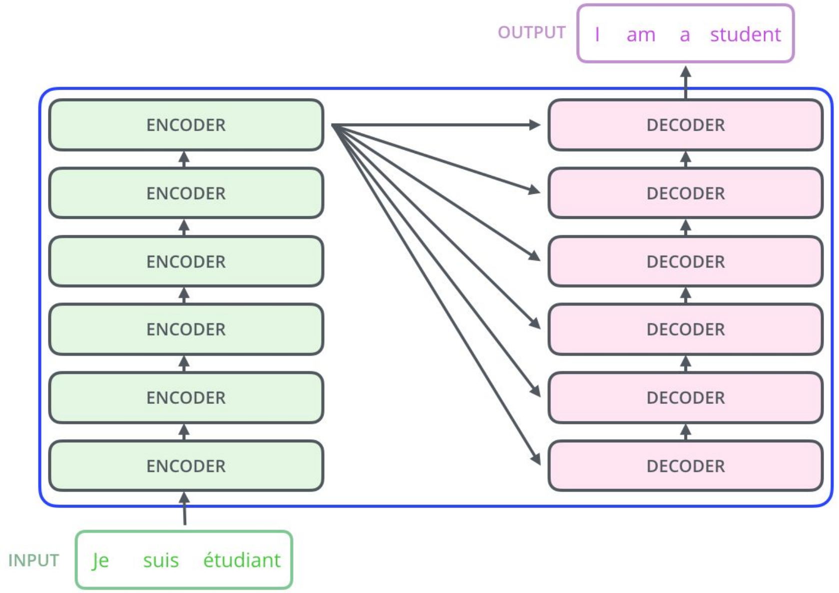




In high level

The high level architecture actually considers **not only one encoder/decoder**, actually there are several encoders/decoders connected by **putting them in an stack.**

The last encoder output will serve as input to the first decoder in the stack.

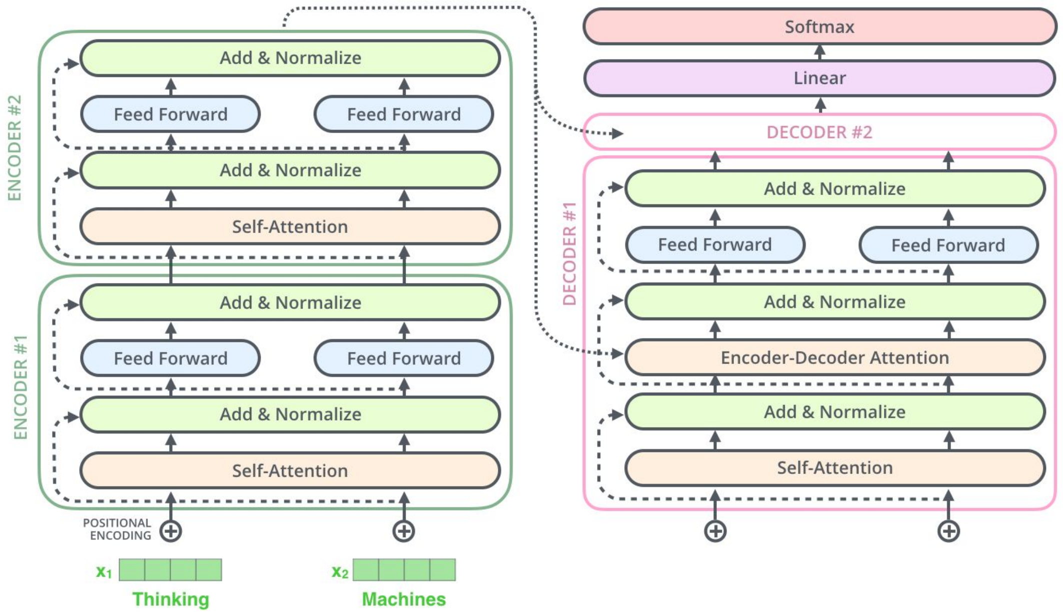


Low level architecture

Transformer model is quite complex but we will analyze every step.

Some important highlight to remember is the **self-attention** component, but we will go there later in this course.

Lets begin in the first step, you will notice a **positional encoding transformation** just in the very beginning.

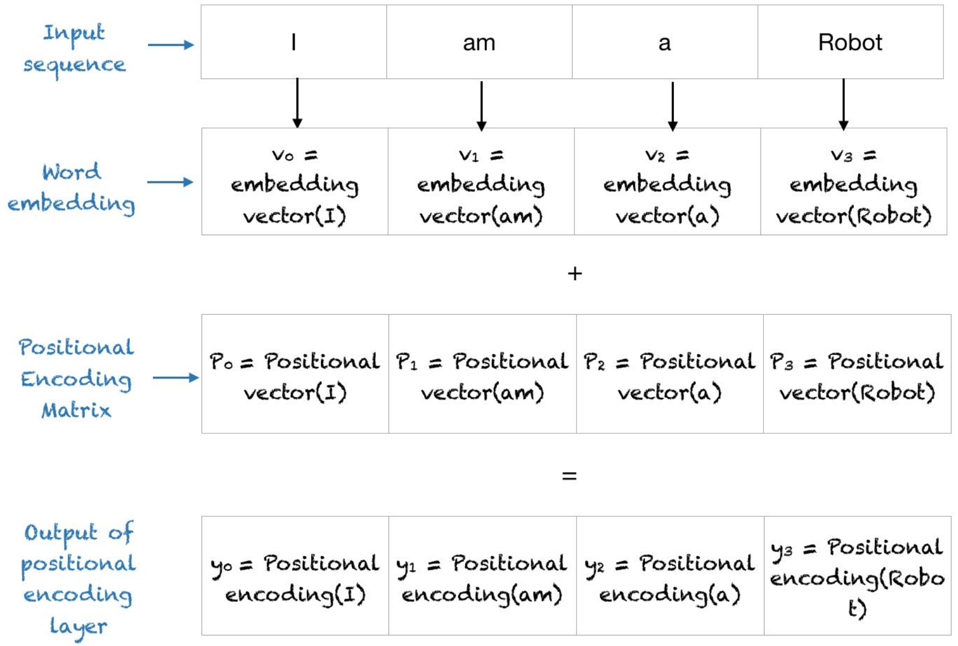


Positional encoding

A positional encoding is added to the **input embeddings of tokens.**

The purpose of positional encoding is to provide the model with information about the **relative positions of tokens** in the input sequence.

It allows the model to distinguish between tokens based on their position in **addition to their semantic content**.



Input sequence
I
am
a
Robot
Word embedding
v_0 = embedding vector(I)
v_1 = embedding vector(am)
v_2 = embedding vector(a)
v_3 = embedding vector(Robot)
+
Positional Encoding Matrix
P_0 = Positional vector(I)
P_1 = Positional vector(am)
P_2 = Positional vector(a)
P_3 = Positional vector(Robot)
=
Output of positional encoding layer
y_0 = Positional encoding(I)
y_1 = Positional encoding(am)
y_2 = Positional encoding(a)
y_3 = Positional encoding(Robo t)

How to encode position?

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Here:

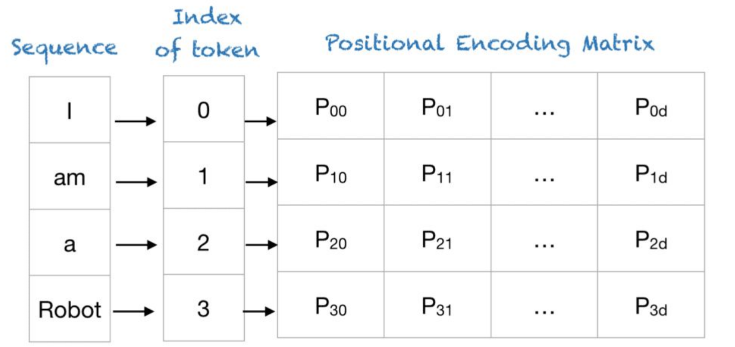
k: Position of an object in the input sequence, $0 \leq k < L/2$

d: Dimension of the output embedding space

$P(k, j)$: Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix

n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

i: Used for mapping to column indices $0 \leq i < d/2$, with a single value of i maps to both sine and cosine functions



Sequence	Index of token	Positional Encoding Matrix			
I	0	P_{00}	P_{01}	…	P_{0d}
am	1	P_{10}	P_{11}	…	P_{1d}
a	2	P_{20}	P_{21}	…	P_{2d}
Robot	3	P_{30}	P_{31}	…	P_{3d}

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

Example

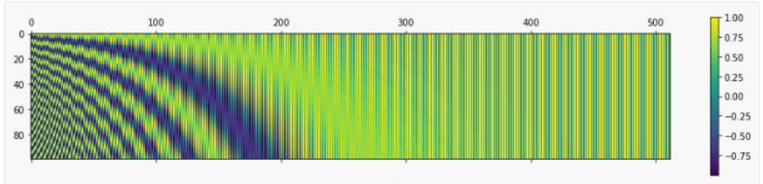
Positional encodings are typically generated using trigonometric functions or other mathematical functions that produce a unique encoding for **each position in the sequence**. These encodings are then added element-wise to the token embeddings.

The choice of functions and encoding methods may vary, but one common method is to use a combination of sine and cosine functions with different frequencies to create a distinct encoding for each position

Positional Encoding Matrix with d=4, n=100

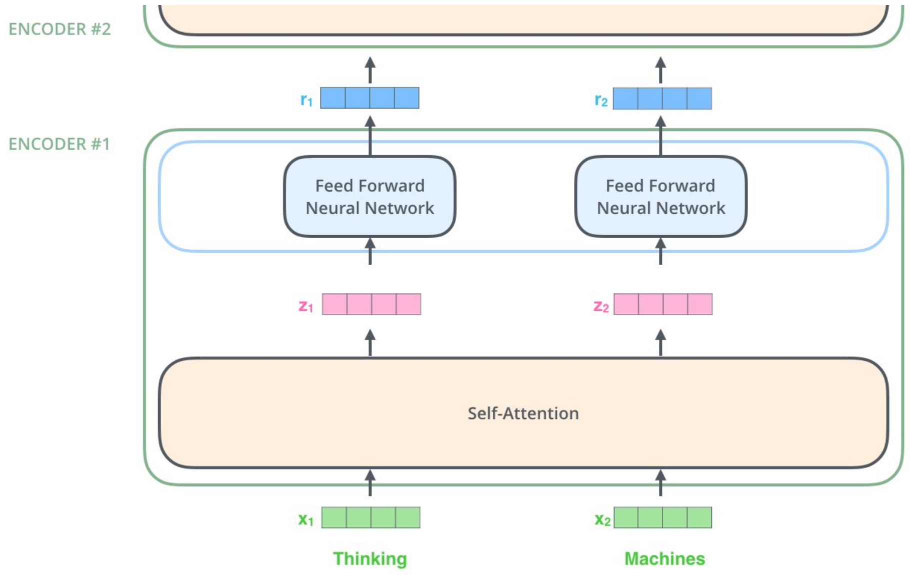
Sequence	Index of token, k	i=0	i=0	i=1	i=1
I	→ 0 →	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	→ 1 →	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	→ 2 →	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -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'



The positional encoding matrix for n=10,000, d=512, sequence length=100

Inside encoder



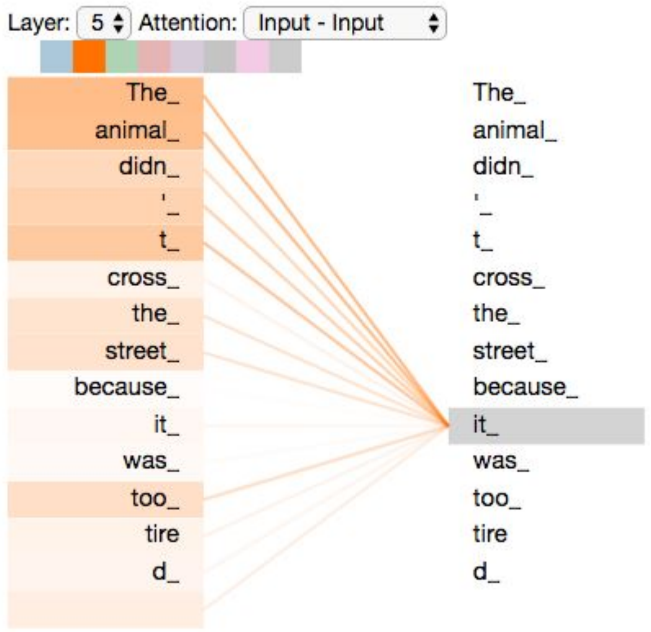
Now that we have our positional embedding , let's move inside each encoder layer.

Original paper considers an encoder size of 6, but this is an hyperparameter to explore. (GPT-2, which has 124M parameters, has 12 transformer layers in its architecture.).

Each encoder has two steps

- **Self-attention**
- **FFN**

Self-attention



As we discuss in last lecture, attention helped sec2sec decoders to produce a new prediction based on the attention step in the input.

Transformers bring self attention as a problem that matters in encoder too.

We need to focus on the relation of the tokens in the input.

For example

”`The animal didn't cross the street because it was too tired`”

Who was to tired, animal or street?

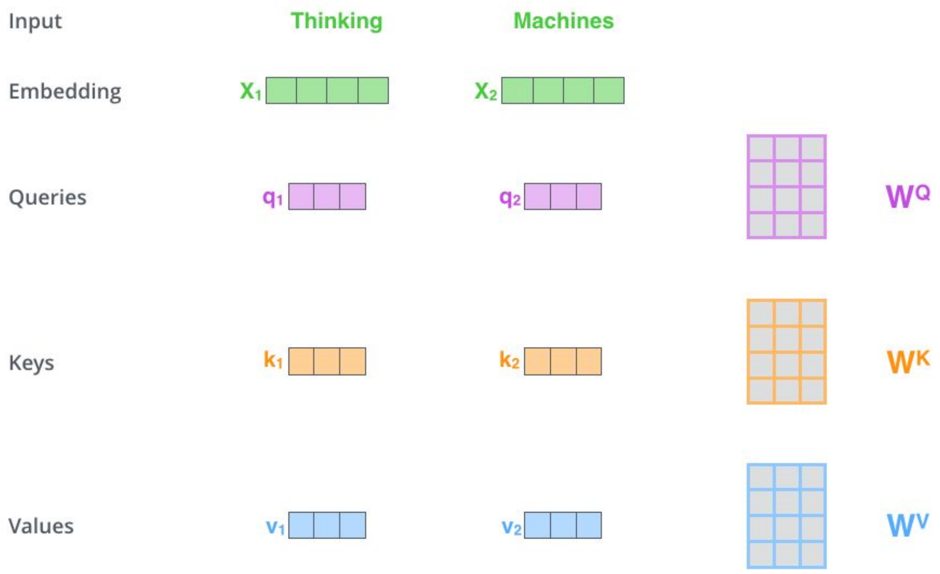
Self attention brings a better encoding that includes this information

Queries, keys, values

Let's put it into an intuitive way of think, we need a **final vector that best encode attention**, and we have a **N input set of tokens**.

For every token embedding we need the **"score relation" with all the other tokens in the input.**

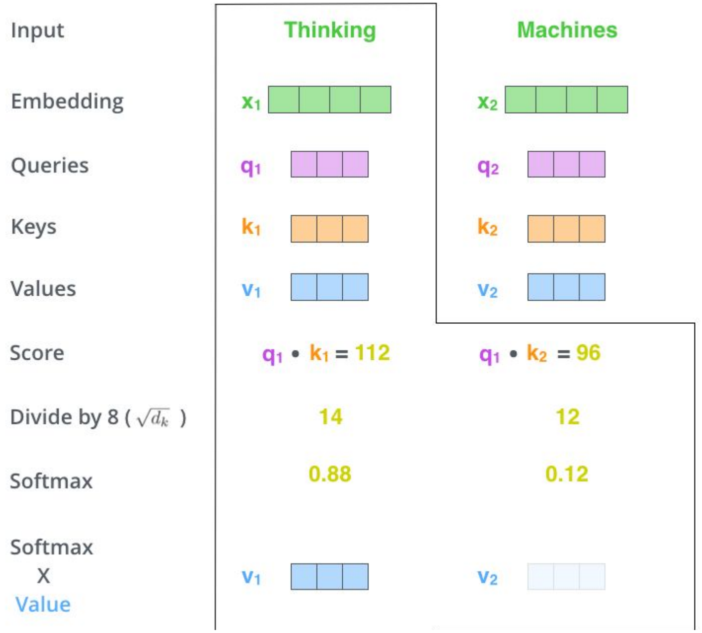
So, for every X vector we will produce three more vectors (usually less in dimension) q,k,v by using three weight matrix (Wq,Wk,Wv) in our model.

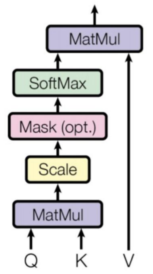


Self attention mechanism

This example shows how this 3 vectors helps input "thinking" to be related/scored with all the other vectors in 4 steps.

1. Q1 dot product with all the kn vectors
2. Then it's divided by 8 (the square root of the dimension of the key vectors used in the paper – 64. This leads to having more stable gradients)
3. Softmax normalizes the scores so they're all positive and add up to 1.
4. Multiply value with softmax (intuition here is get a X value transformed since not relevant will be multiplied by near 0 values)
5.

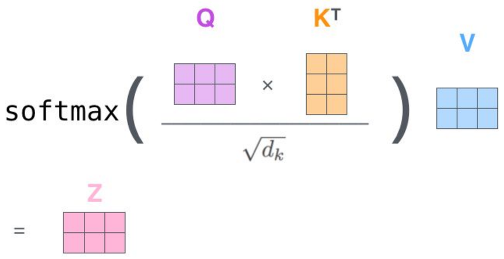


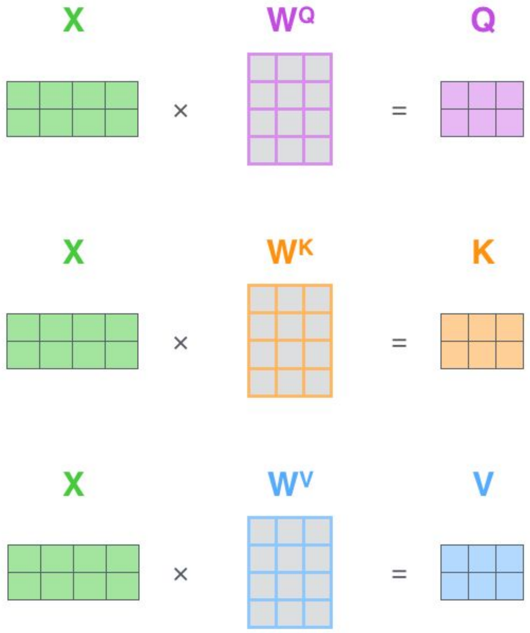


In matrix operations

The representation of this process in matrix might look as this:

- X represents all the input vectors ,
- W the matrices for get q,k,x
- Z is the resulting new set of input vectors in a better representation (with attention).



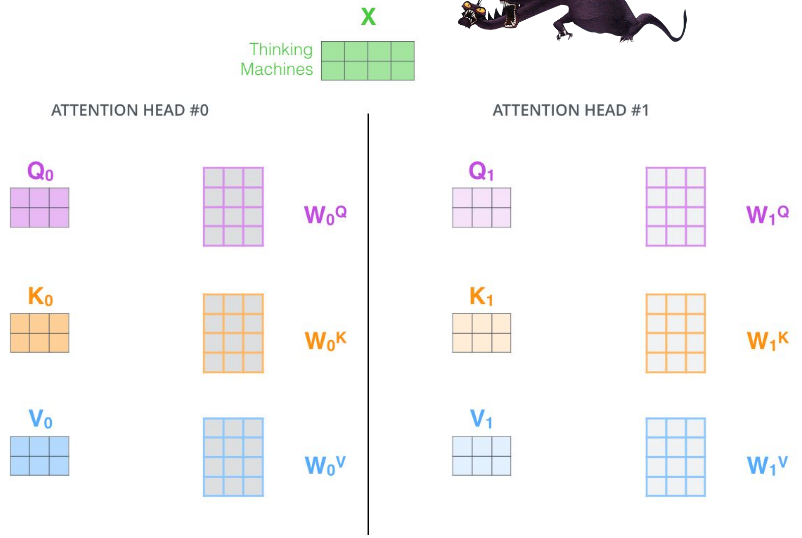


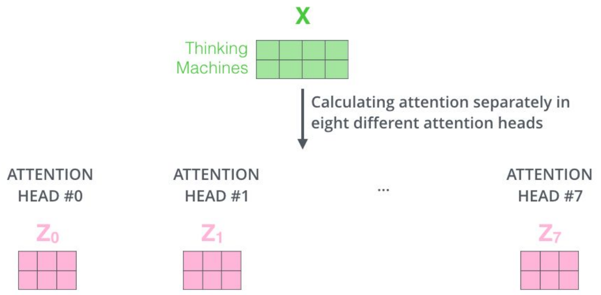
“multi-headed” attention

Transformers paper brings an optimization, since embeddings is a very high dimensional space we **should train this in many epochs.**

This was a problem in RNN models but now , we can do it in **parallel with the multi-headed mechanism,** this also helps to explore different types of semantic attention (pragmatics analysis).

Original paper propose 8 “heads” (simultaneous matrix weight calculation), however **this is another hyper-parameter** to explore (GPT 3 has 96).

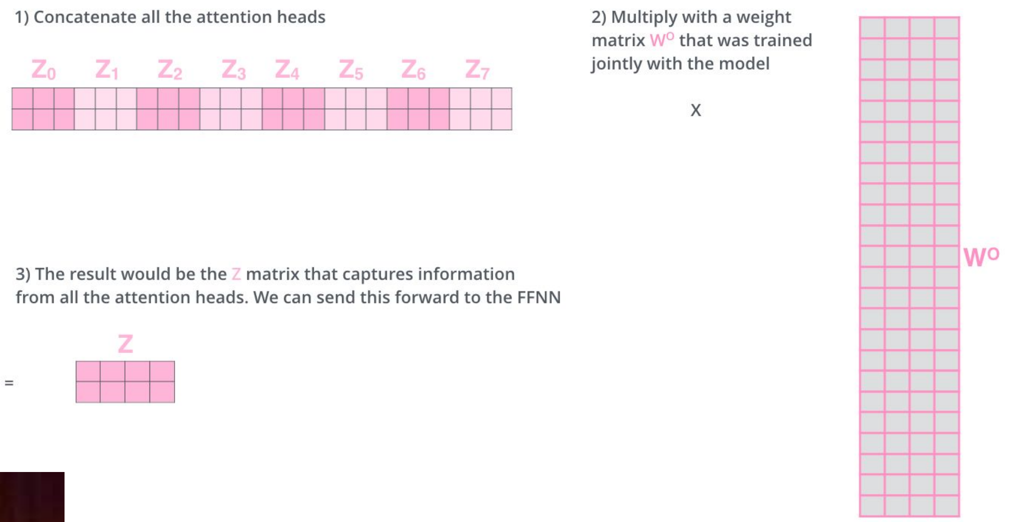




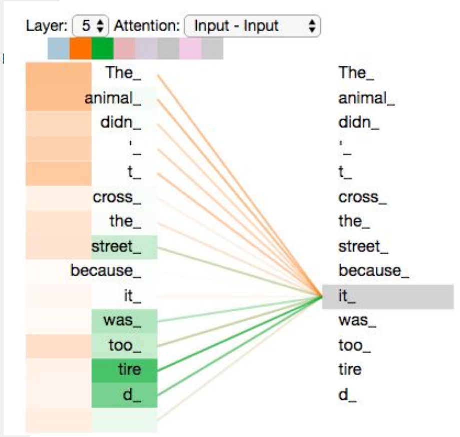
“multi-headed” issue

- Although this is a great idea, the next layer (FFNN) requires one vector per token ...

  The answer is to concatenate the output and reduce it by using another Wo matrix ... **yes another W matrix.**



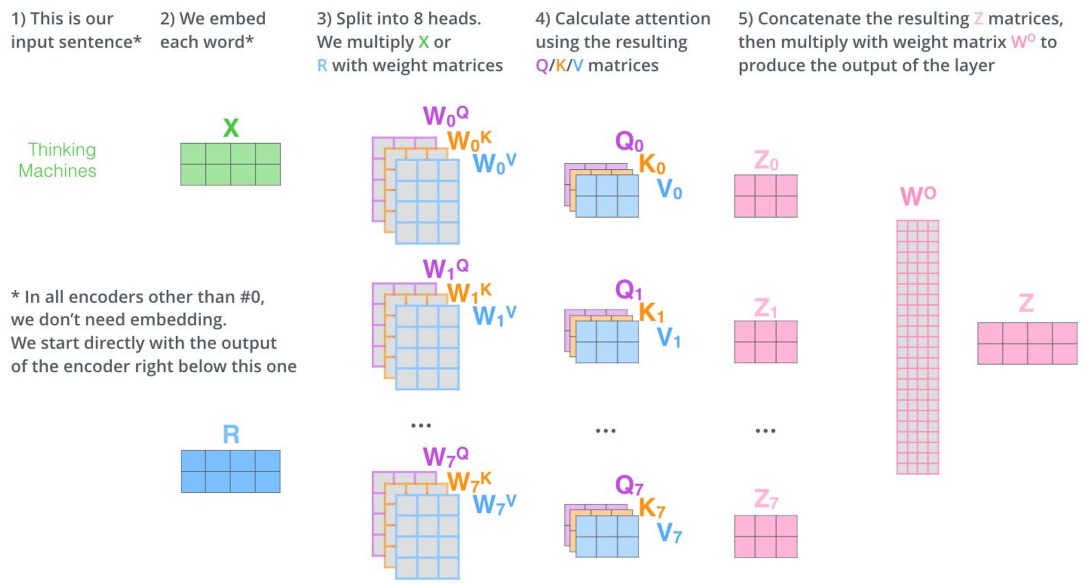
1) Concatenate all the attention heads
Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7
2) Multiply with a weight matrix W^O that was trained jointly with the model
X
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
=
Z



Layer: 5 Attention: Input - Input
The_ animal_ didn_ '_ t_ cross_ the_ street_ because_ it_ was_ too_ tire d_
The_ animal_ didn_ '_ t_ cross_ the_ street_ because_ it_ was_ too_ tire d_

Self-attention multi-headed recap

We are finishing all the necessary steps to continue to the second part of encoder **FFNN,** lets recap.



1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

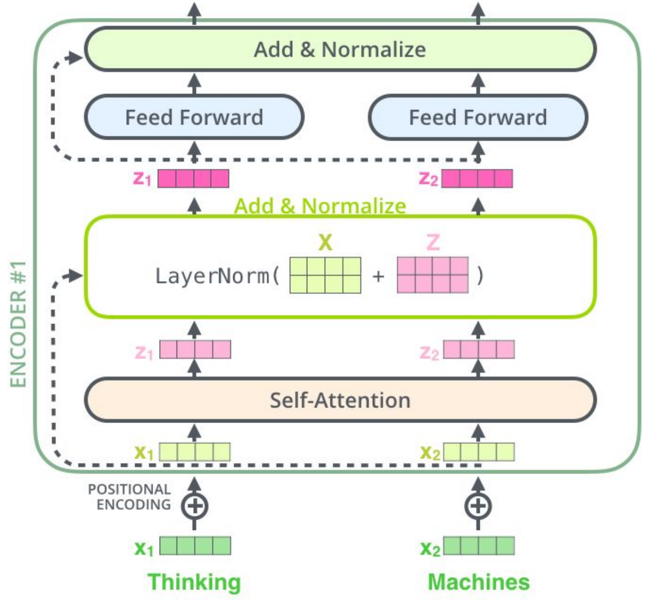
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Finishing encoding

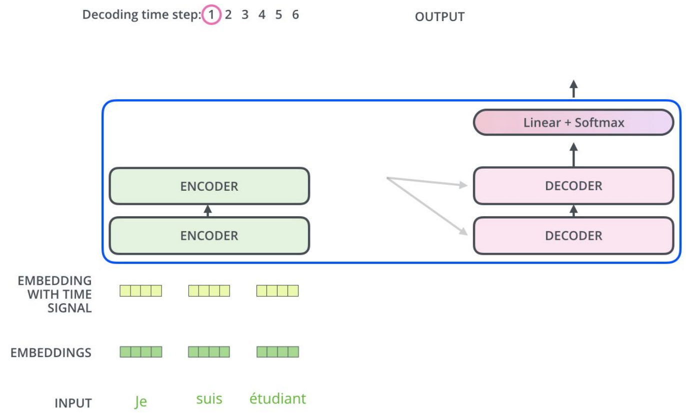
In models like the Transformer, which **relies heavily on self-attention** mechanisms, normalization plays a crucial **role in stabilizing the behavior of attention weights**, ensuring that they do not become too large or too small.

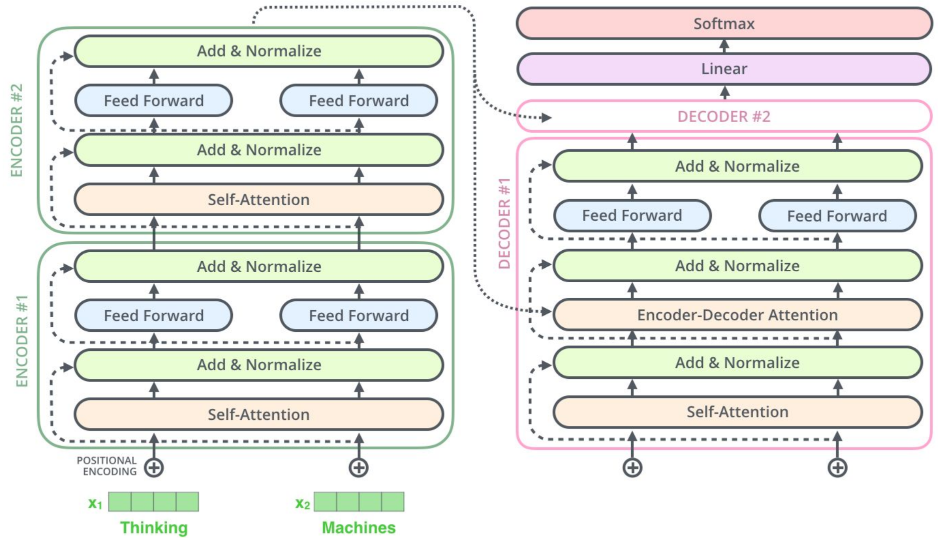
Finally FFN plays a crucial role in **extracting, transforming, and generalizing information from the input sequence**. It complements the self-attention mechanism by introducing non-linearity, flexibility, and feature extraction capabilities



Starting Decoding

At the end of the top decoder, the output is transformed into a k and v matrices to be use in the “encoder-decoder attencion layer”. This process helps the **decoder to focus on the important components of the input (attention)** to predict.



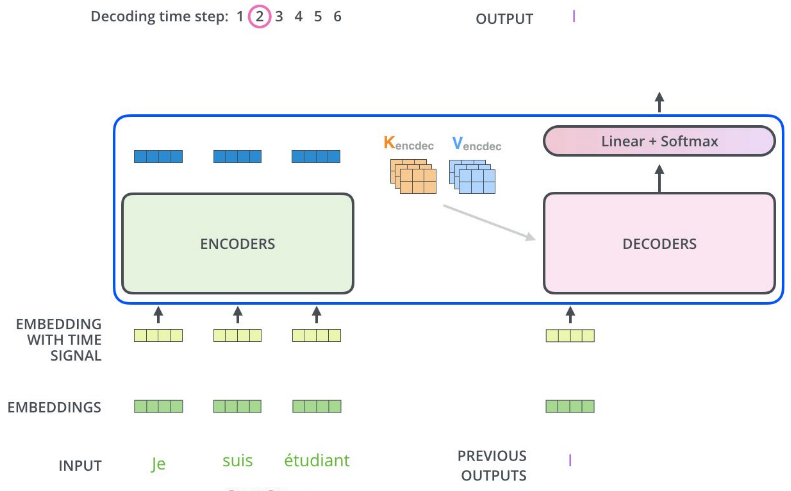


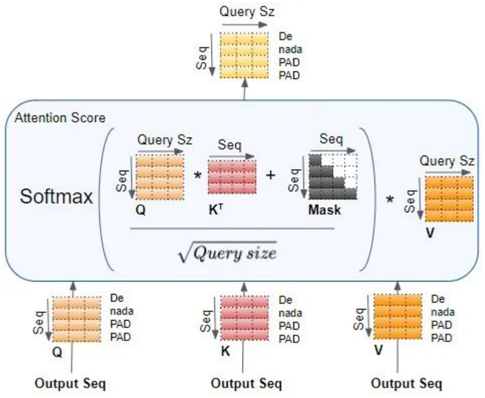
Decoder-self attention

In every step of the decoder an output is predicted taking into account **previous outputs.**

But now self **attention needs to include the previous outputs** (those are past now).

But there is a small change, attention it's only important from past to future. Not from future to past. This is called **no autoregressive behaviour** and it's donde by **masking the self-attention matrices**.



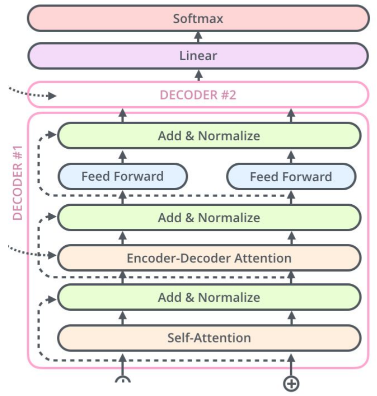


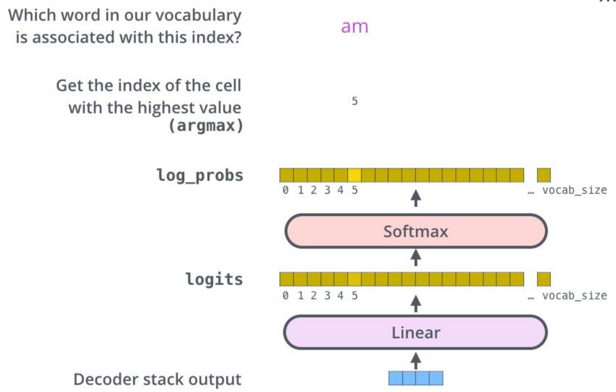
New vectors to words

Our original embeddings has change to much, in order to convert new vector into tokens we use the last two phases (linear and softmax).

Linear is a simple NN to convert vector to a “hot encode” vocabulary representation.

Softmax is a function to translate that vector into a probability output.




Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5 … vocab_size
Softmax
logits
0 1 2 3 4 5 … vocab_size
Linear
Decoder stack output

Training

To train we must include mask examples from our corpus, for example

“Thanks I am a student”

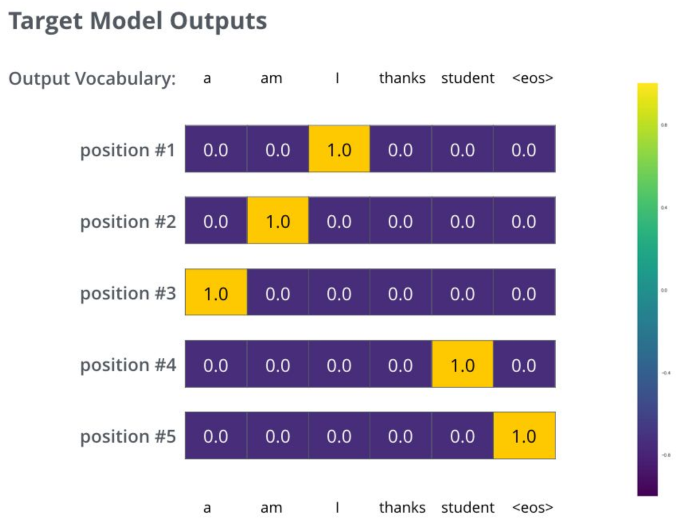
Will be trained as

X1 = “thanks I <mask> a student <EOS>”

y1= “am”

Notice the <mask> input needs to be transformed into am.

The function loss might now measures how well the transformer produces a correct hot encode vector for each position.



Target Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0
	a	am	I	thanks	student	<eos>

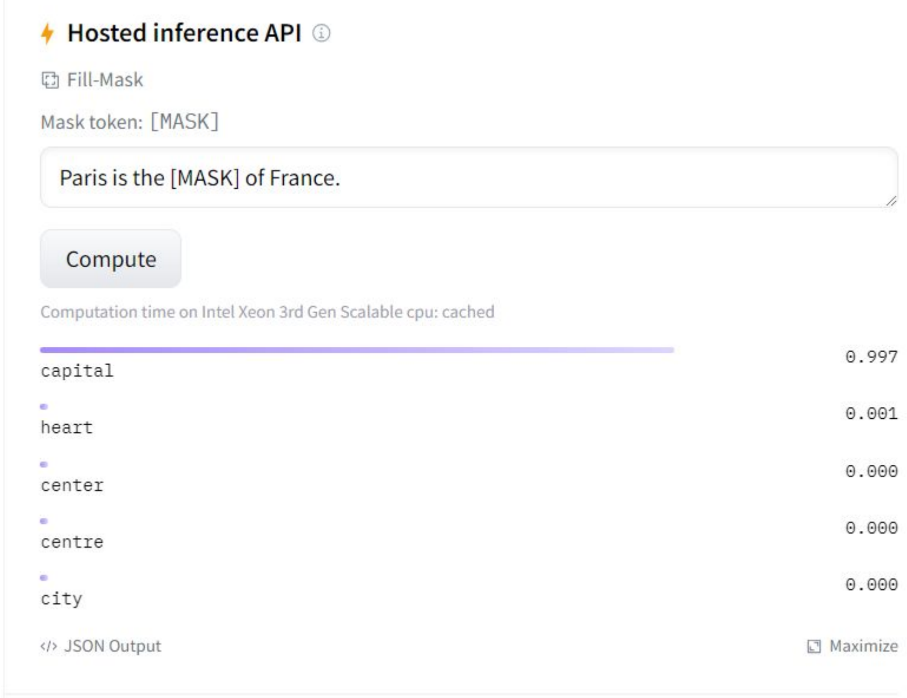
BERT

BERT, which stands for **Bidirectional Encoder Representations from Transformers**, is a deep learning model that has revolutionized the field of natural language processing (NLP).

Developed by researchers at Google AI in 2018, BERT is a type of transformer model, a neural network architecture that has proven highly effective for a wide range of NLP tasks.

Try it for free:

https://huggingface.co/bert-base-uncased?text=Paris+is+the+%5BMASK%5D+of+France.



Hosted inference API

Fill-Mask

Mask token: [MASK]

Paris is the [MASK] of France.

Compute

Computation time on Intel Xeon 3rd Gen Scalable cpu: cached

```
capital    0.997
heart      0.001
center     0.000
centre     0.000
city       0.000
```

JSON Output

Maximize

BERT

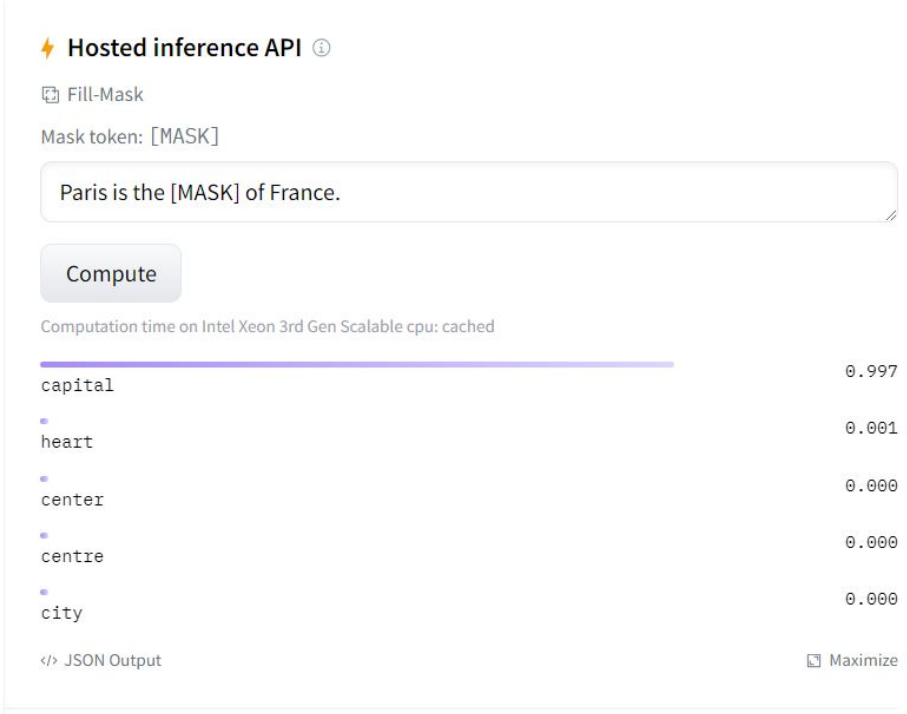
BERT

BERT, which stands for **Bidirectional Encoder Representations from Transformers**, is a deep learning model that has revolutionized the field of natural language processing (NLP).

Developed by researchers at Google AI in 2018, BERT is a type of transformer model, a neural network architecture that has proven highly effective for a wide range of NLP tasks.

Try it for free:

https://huggingface.co/bert-base-uncased?text=Paris+is+the+%5BMASK%5D+of+France.


Hosted inference API

Fill-Mask

Mask token: [MASK]

Paris is the [MASK] of France.

Compute

Computation time on Intel Xeon 3rd Gen Scalable cpu: cached

```
capital     0.997
heart       0.001
center      0.000
centre      0.000
city        0.000
```

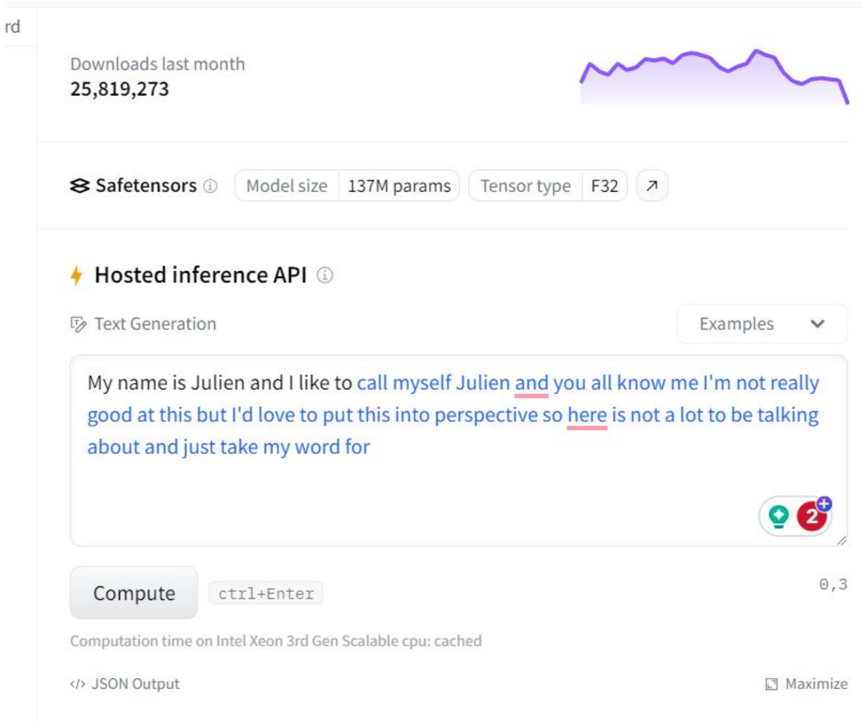
GPT

GPT, which stands for **Generative Pre-trained Transformer**, is a deep learning model that uses transformers just like bert.

Developed by researchers at Open AI in 2018, GTP focus more on decoding task (generative AI) than BERT which focus on understanding better (encoder).
Both has improve in time until reaching both (encoder and decoder) far much better.

Try it for free:

https://huggingface.co/gpt2



Downloads last month
25,819,273

Safetensors | Model size 137M params | Tensor type F32

Hosted inference API

Text Generation | Examples

My name is Julien and I like to call myself Julien and you all know me I'm not really good at this but I'd love to put this into perspective so here is not a lot to be talking about and just take my word for

Compute `ctrl+Enter` 0,3

Computation time on Intel Xeon 3rd Gen Scalable cpu: cached

JSON Output | Maximize

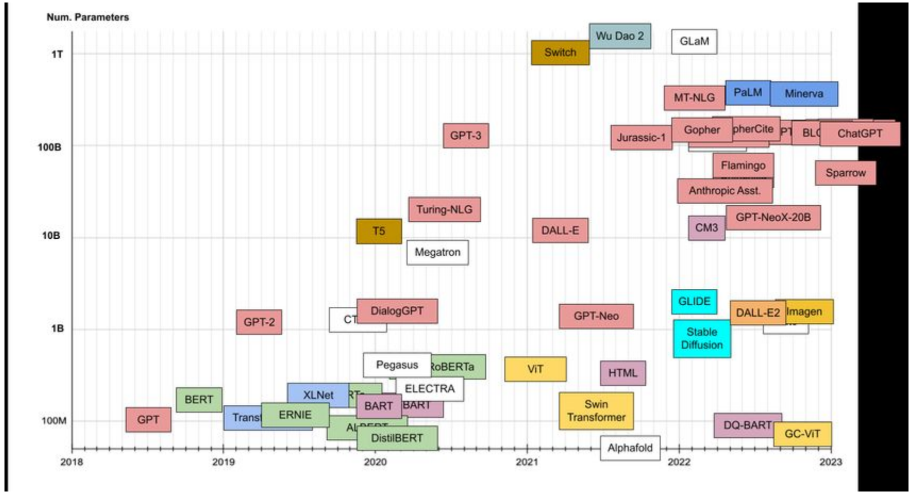
Transformers timeline

Since BERT and GPT, parameters has been greater and increasing the dataset until today.

Also, more fine tuning models based on top of this models.

This top models are focused on dealing with specific tasks to solve:

- QA
- Creating code
- Solving Maths
- Knowledge retrieve



Thanks

Do you have any questions?

emmanuel.paez@tec.mx
Slack #module-5-nlp-1