From Bag of words to LLMs

A brief recall:
NLP problems, a brief summary

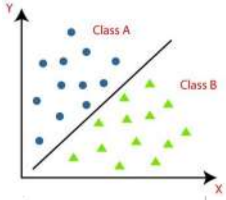
Some of the most common task:

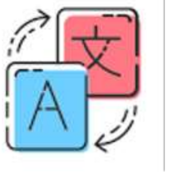
- **Text classification (eg: try to classify if an email is SPAM or not; Sentiment analysis);**

- **Machine translation (eg: translating text from English to Italian)**

- **Text generation (eg: text completion, document summarization, code generation and completion, but also machine translation)**

- **And more counting….**





A brief recall:
NLP problems, a brief summary

We are gonna focus on:

- **Text classification (encoders)**
- **Text generation (decoders)**

These tasks started from a crucial problem:

- **How do we handle the text?**

GitHub
Copilot

A brief recall:
A first toy problem, text classification!

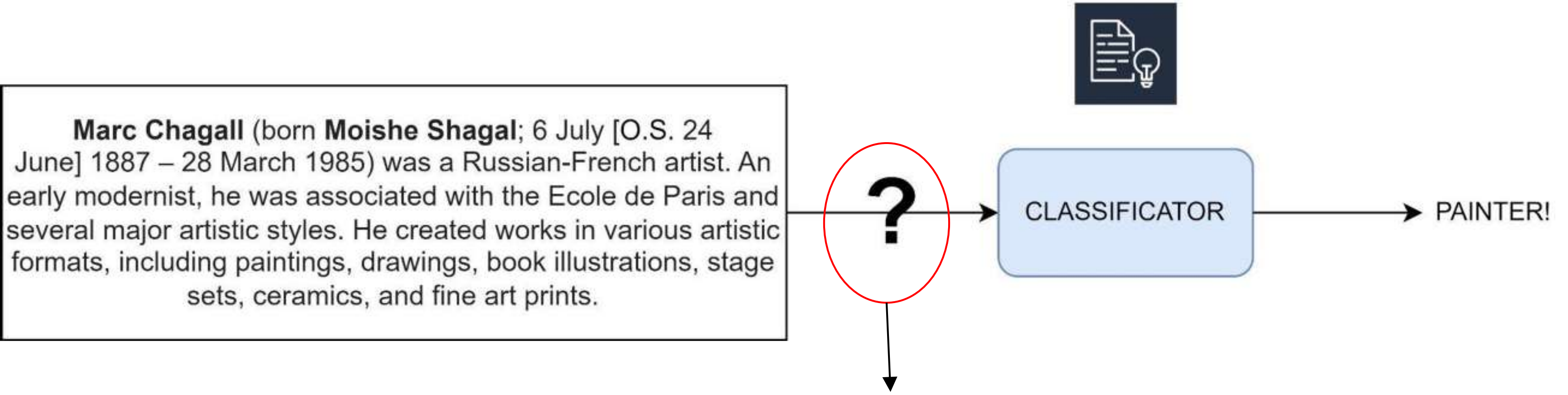
Marc Chagall (born **Moishe Shagal**; 6 July [O.S. 24 June] 1887 – 28 March 1985) was a Russian-French artist. An early modernist, he was associated with the Ecole de Paris and several major artistic styles. He created works in various artistic formats, including paintings, drawings, book illustrations, stage sets, ceramics, and fine art prints.

? → CLASSIFICATOR → PAINTER!

Chesney Henry "Chet" Baker Jr. (December 23, 1929 – May 13, 1988) was an American jazz trumpeter and vocalist. He is known for major innovations in cool jazz that led him to be nicknamed the "Prince of Cool".

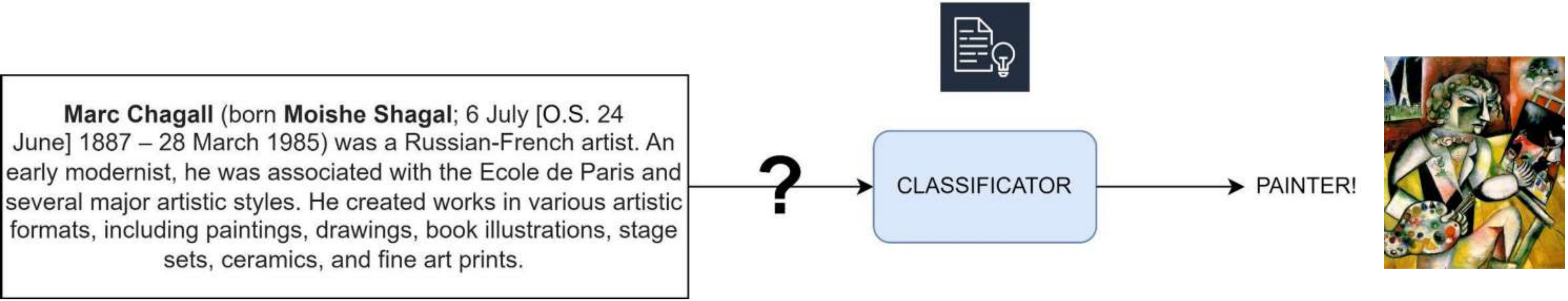
? → CLASSIFICATOR → MUSICIAN!

A brief recall:
A first toy problem, text classification!



Data must get numerical!
The text must be <u>vectorized</u>

A brief recall:
A first toy problem, text classification! SPARSE REPRESENTATIONS



First approaches?

<u>Bag of Words / Term Frequency:</u> Each document is split into <u>**tokens**</u> (here represented as finite words), preprocessed, and then, each token is counted, obtaining a first vectorial representation

A brief recall:
A first toy problem, text classification! SPARSE REPRESENTATIONS

PREPROCESSING PHASE

- Tokenization: “The cat is running for its life”-> [The, cat, is, running, for, its, life]
- Lowecasing: [The, cat, is, running, for, its, life] -> [the, cat, is, running, for, its, life]
- Stop words romoval: [cat, is, running, for, its, life]
- Stemming: [cat, is, run, for, its, life]

and so on…

PROS: We ease the classification

CONS: We

A brief recall:
A first toy problem, text classification! SPARSE REPRESENTATIONS

First approaches?

<u>Bag of Words / Term Frequency:</u> Each document is split into **<u>tokens</u>** (here represented as finite words), preprocessed, and then, each token is counted, obtaining a first vectorial representation

BAG OF WORDS

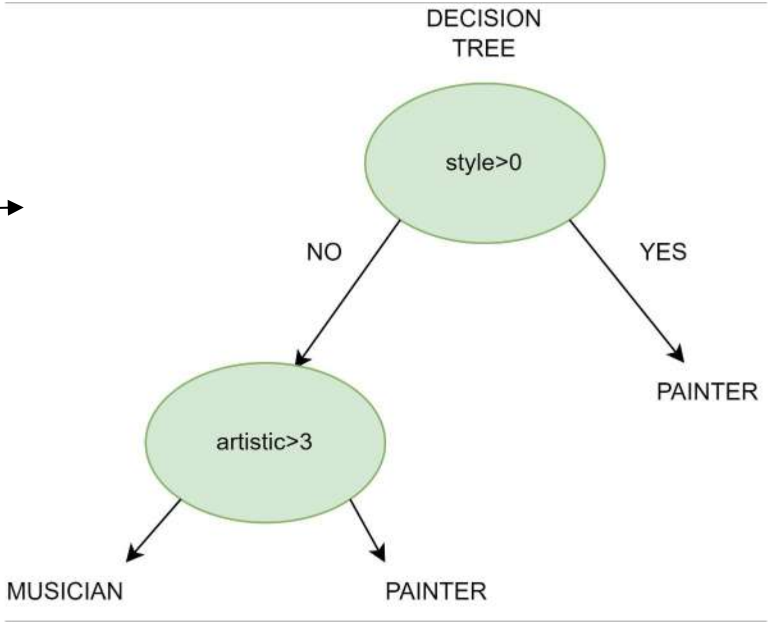
Vocabulary					
Term	artistic	part	...	style	talent
Freq.	2	0	...	1	0

Vocabulary					
Term	artistic	part	...	style	talent
Freq.	2/53	0/53	...	1/53	0/53

A brief recall:
A first toy problem, text classification! SPARSE REPRESENTATIONS

BAG OF WORDS

Vocabulary					
Term	artistic	part	...	style	talent
Freq.	2	0	...	1	0



A brief recall:
A first toy problem, text classification! SPARSE REPRESENTATIONS

First approaches?

Bag of Words / Term Frequency: These techniques led to huge vectors (up to 10.000 terms) with a lot of zeros….

BAG OF WORDS

Vocabulary					
Term	artistic	part	...	style	talent
Freq.	2	0	...	1	0

Vocabulary					
Term	artistic	part	...	style	talent
Freq.	2/53	0/53	...	1/53	0/53

A brief recall:
SPARSE REPRESENTATIONS

PROBLEMS OF THESE APPROACHES:

- Huge matrices -> Difficult to handle for NN, SVM or other ML algorithms…
- Leak of information -> OUT OF VOCABULARY words
- Leak of information -> The sequential structure of texts
- Leak of information -> Polysemic words
- Leak of information -> Do not capture word relationships (King and kingdom seem completely different vectors)
- Leak of information -> CONTEXT

A brief recall:
DENSE REPRESENTATIONS WORD2VEC

How do we get better word representations?

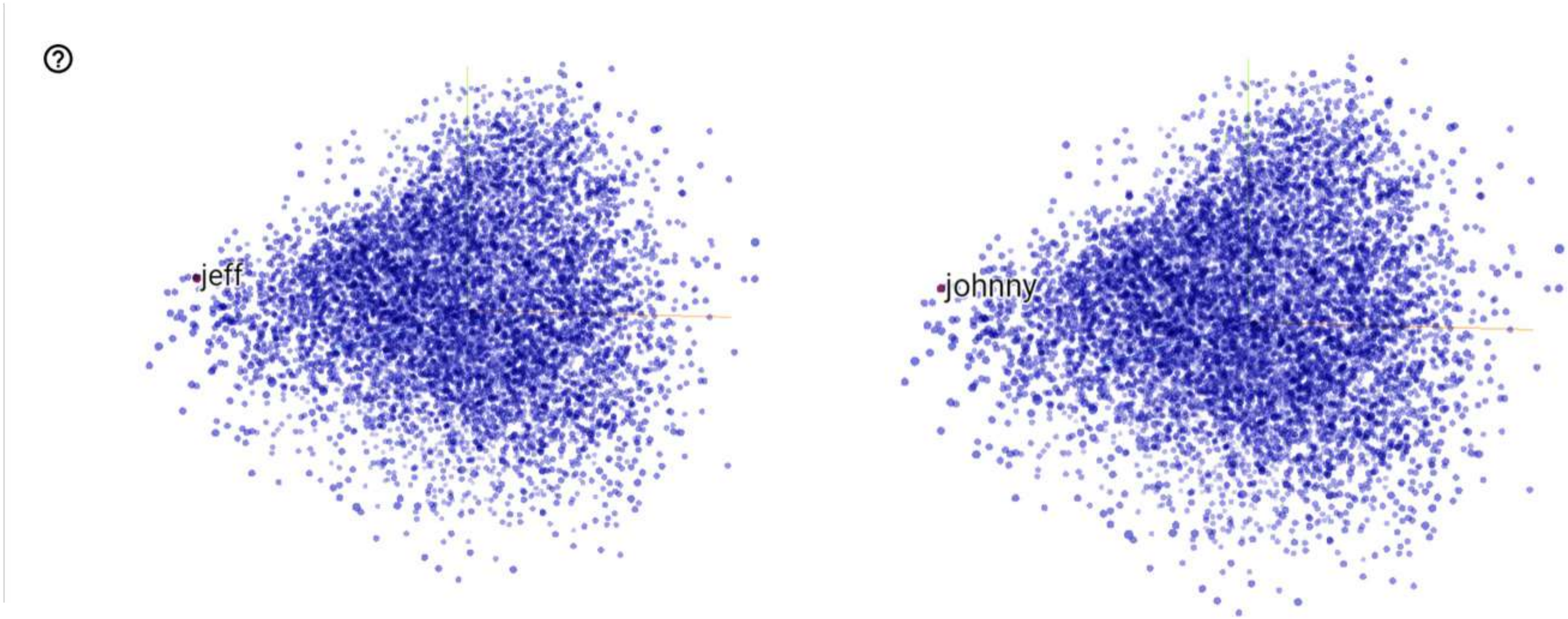
Mikolov[1] proposed to <u>learn the words</u> during training

- Each word obtains a dense <u>representation on D dimension</u>
  This led to a NxD matrix (N as number of words in the vocab, D as dimensionality)
- Words vectors now have meaningful relationships

Vector(King) - Vector(Man) + Vector(Woman) ≈ Vector(Queen)

[1] [Efficient Estimation of Word Representations in Vector Space]()

A brief recall:
DENSE REPRESENTATIONS WORD2VEC



https://projector.tensorflow.org/

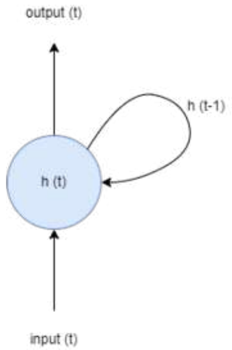
A brief recall:
Our open and closed list of problems

PROBLEMS OF THESE APPROACHES:

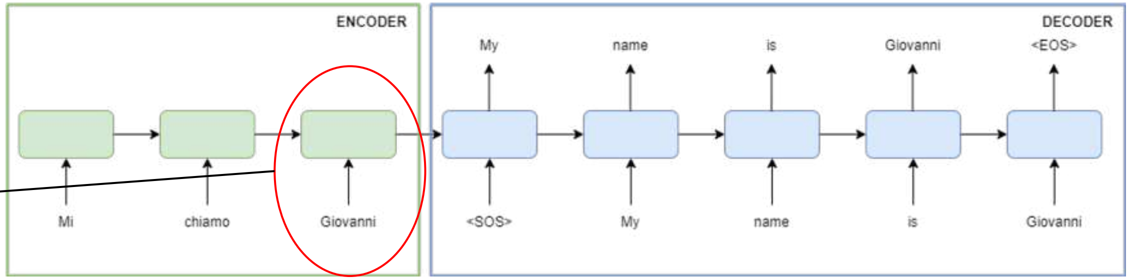
- Huge matrices -> Difficult to handle for NN, SVM or other ML algorithms ✓
- Leak of information -> OUT OF VOCABULARY words
- Leak of information -> The sequential structure of texts
- Leak of information -> Polysemic words
- Leak of information -> Do not capture word relationships (King and kingdom seem completely different vectors) ✓
- Leak of information -> CONTEXT

A brief recall:
Recurrent Neural Networks

- The aim of RNNs is to process and vectorize sequential data (perfectly suitable for text)



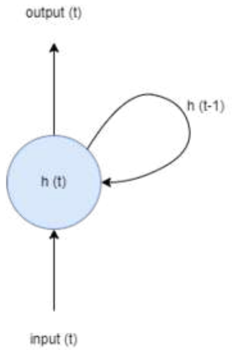
RNN are nothing more than a Neural network with a recursive input, enabling <u>sequence modelling</u>.



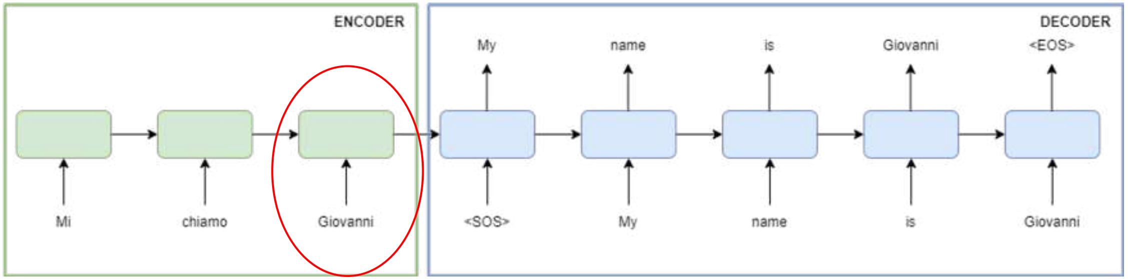
Encoded information

A brief recall:
Recurrent Neural Networks

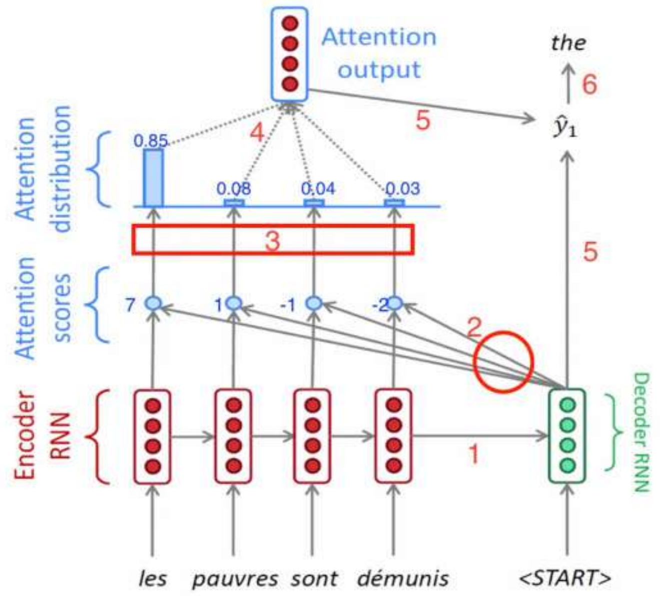
- **The aim of RNNs is to process and vectorize sequential data (perfectly suitable for text)**



RNN are nothing more than a Neural network with a recursive input, enabling sequence modelling.
Challenges?
Can a hidden state summarize the encoded text?



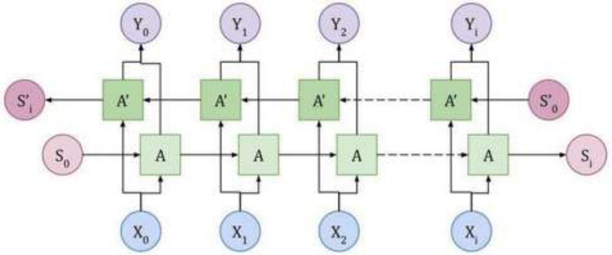
A brief recall:
Recurrent Neural Networks -> ATTENTION



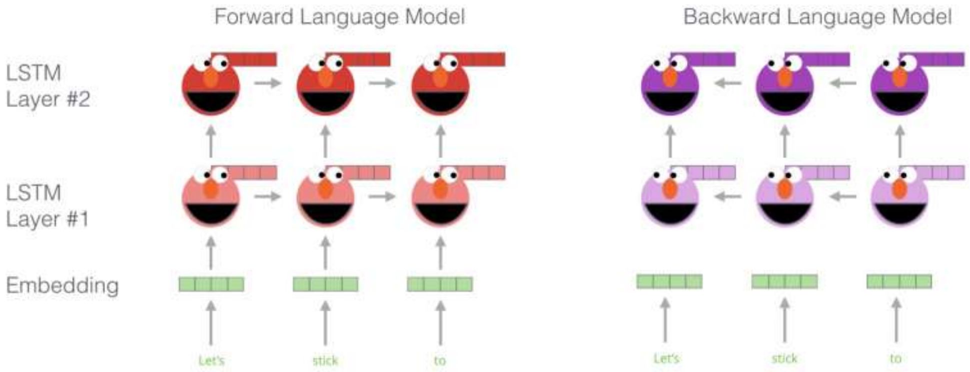
The bottleneck problem was avoided with the advent of the attention mechanism

https://medium.com/@suvasism/nlp-investigation-into-transformer-self-attention-building-blocks-and-the-effects-of-pretraining-1682a5247979

A brief recall:
Recurrent Neural Networks -> Context?



Bi RNN



ELMO -> more context!

https://medium.com/@suvasism/nlp-investigation-into-transformer-self-attention-building-blocks-and-the-effects-of-pretraining-1682a5247979
https://jalammar.github.io/illustrated-bert/

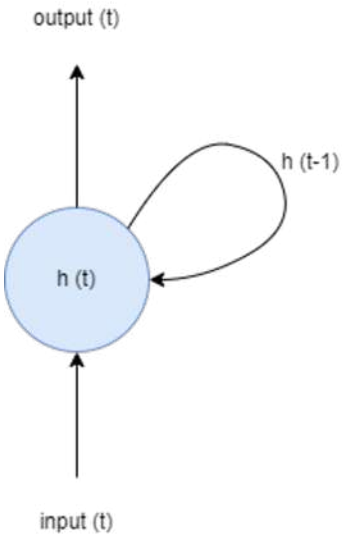
A brief recall:
Recurrent Neural Networks

PROS:

- Enable sequence modelling
- They can virtually handle arbitrary input and output length sequences

CONS:

- Inefficient parallelization (it process sequences one step at a time, leading to slow training time)
- Not well suited for long sequences, suffering from the vanishing gradient problem (LSTMs mitigate this problem but still lack on parallelization properties)



A brief recall:
Our open and closed list of problems

PROBLEMS OF THESE APPROACHES:

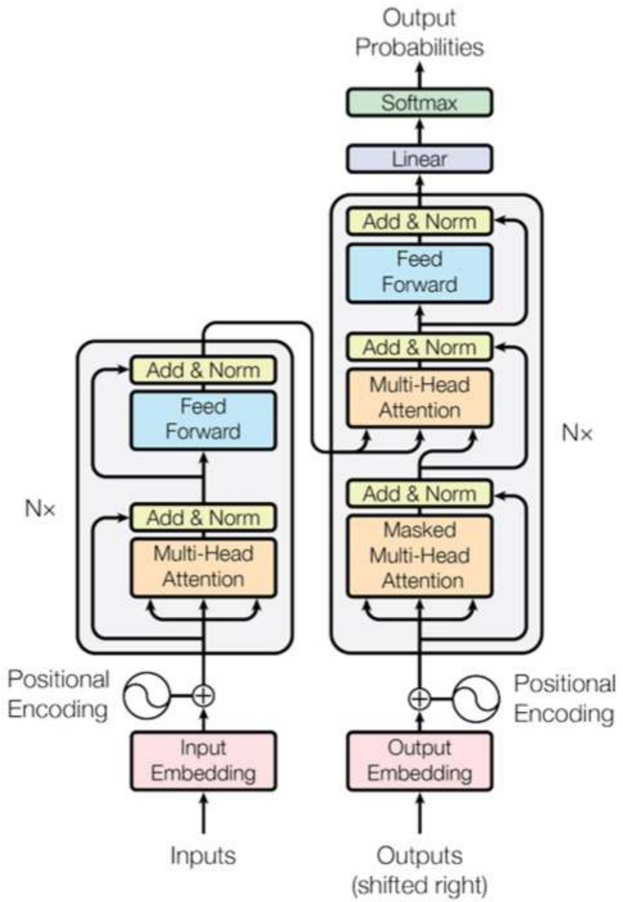
- Huge matrices -> Difficult to handle for NN, SVM or other ML algorithms
- Leak of information -> OUT OF VOCABULARY words
- Leak of information -> The sequential structure of texts
- Leak of information -> Polysemic words
- Leak of information -> Do not capture word relationships (King and kingdom seem completely different vectors)
- Leak of information -> CONTEXT

A brief recall: Transformers

The transformer architecture tackles the sequence related problem, obtaining a model fully <u>parallelizable</u> and suitable for long sequences.
The model relies on:

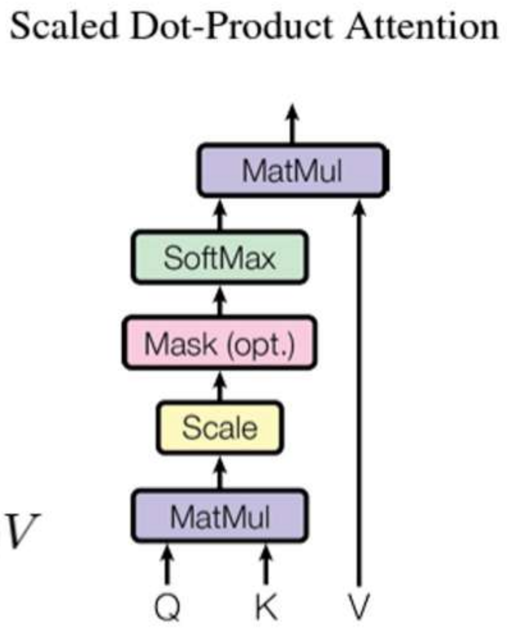
- **Self-attention mechanism**
- **Positional encoding to embed informations regarding the sequence's order**
- **Encoder-decoder architecture**

[Here a complete explanation of the architecture]



A brief recall: Self attention

- The transformer architecture introduced the so called ‘scaled dot product attention’ over three different representation (query, key and value) of the same input vector, obtaining thus the ‘self-attention’ mechanism. The dot-product attention can be implemented using highly optimized matrix multiplication code, so it’s <u>fully parallelizable!</u>

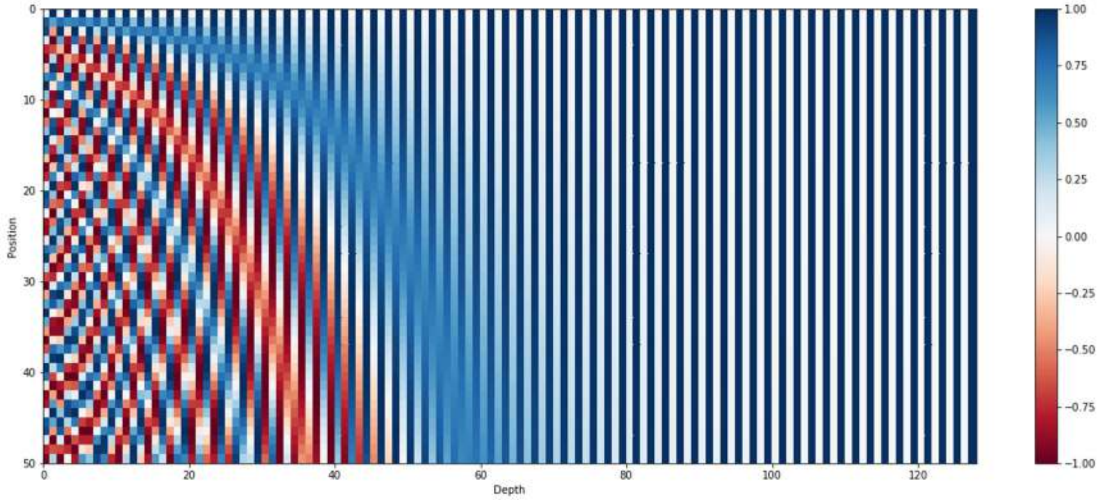


$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

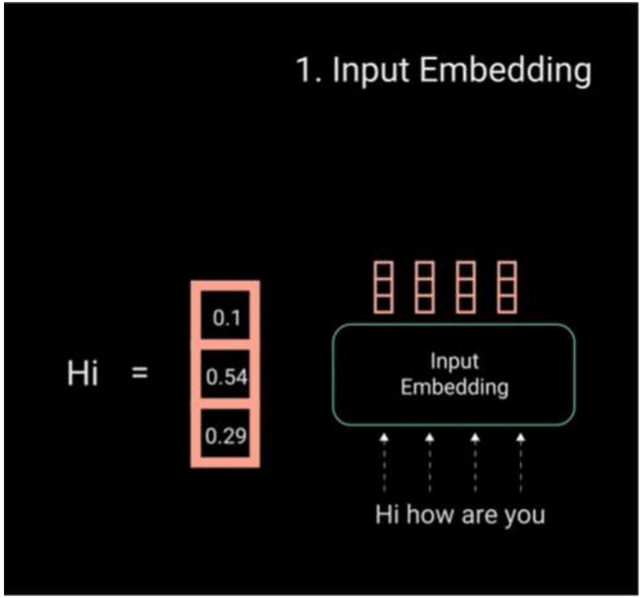
A brief recall: Positional encoding

Transformers are positional invariant with respect to reordering of the input.
However, language is inherently sequential and word order is essential.
By encoding position information to input embeddings, we provide absolute (depending on the algorithm even relative) position information of the token.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



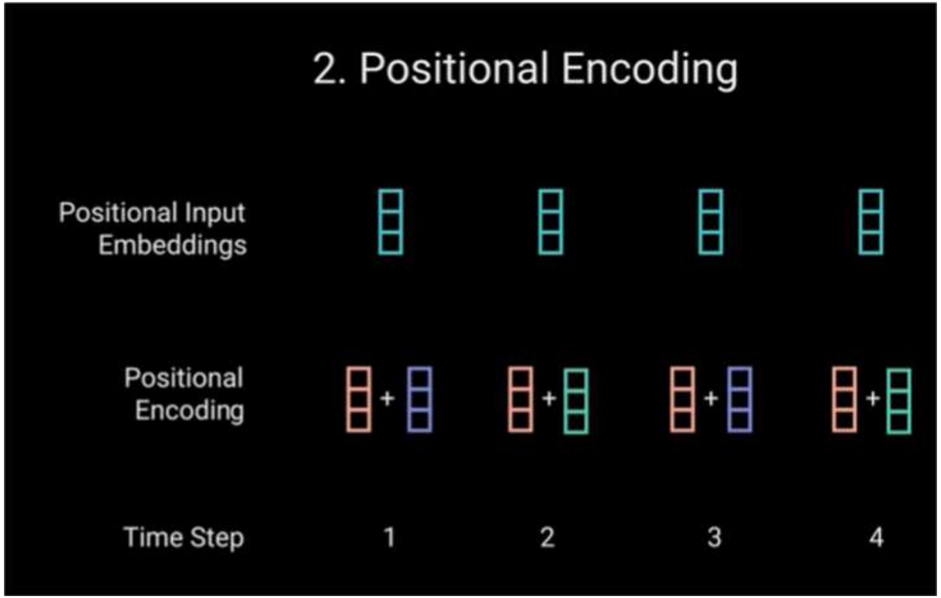
A brief recall: let's visualize the whole process[2]



At first we retrieve the dense representations of the words (embedding).
Pytorch layer: 'A simple lookup table that stores embeddings of a fixed dictionary and size'

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall: let's visualize the whole process[2]



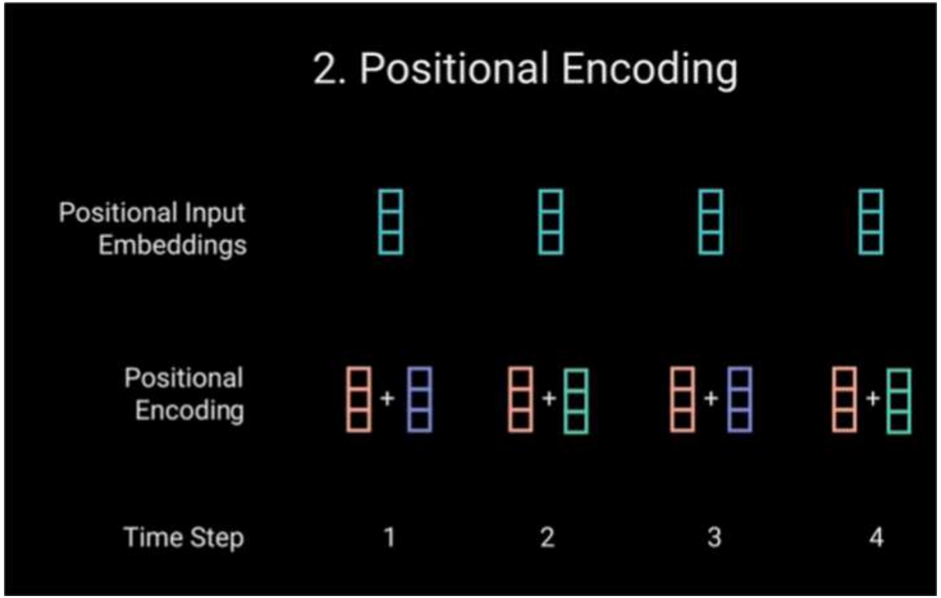
We then transform the vectors to inject information about the position.
There are several ways.
Nowadays vectors are just rotated… here we sum the sinusoidal positional encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall: let's visualize the whole process[2]



We then transform the vectors to inject information about the position.
There are several ways.
Nowadays vectors are just rotated… here we sum the sinusoidal positional encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall: let’s visualize the whole process[2]

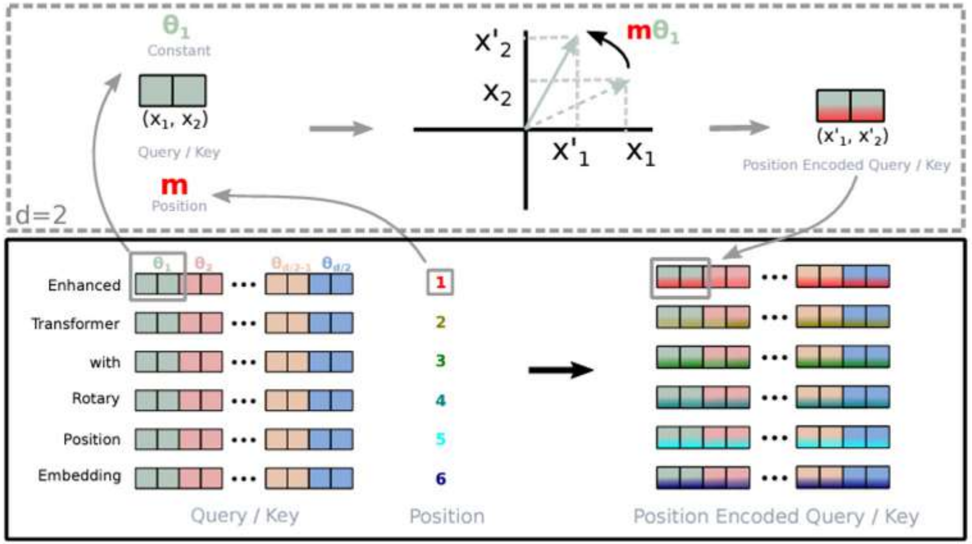


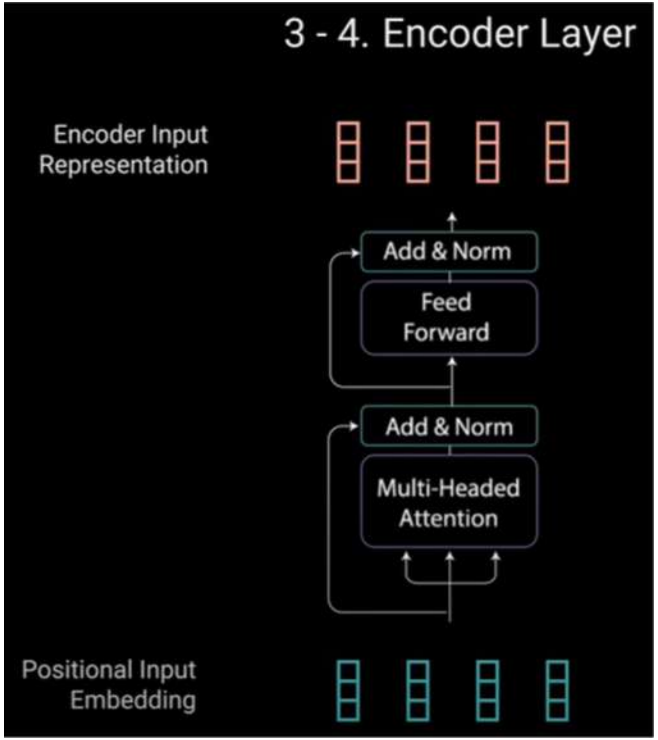
Figure 1: Implementation of Rotary Position Embedding(RoPE).

Rotating the representation led to:

- Models with extrapolation capabilities (the model can generalize over unseen positions)
- Great relative position injected capabilities (it works great with longer sequences)

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

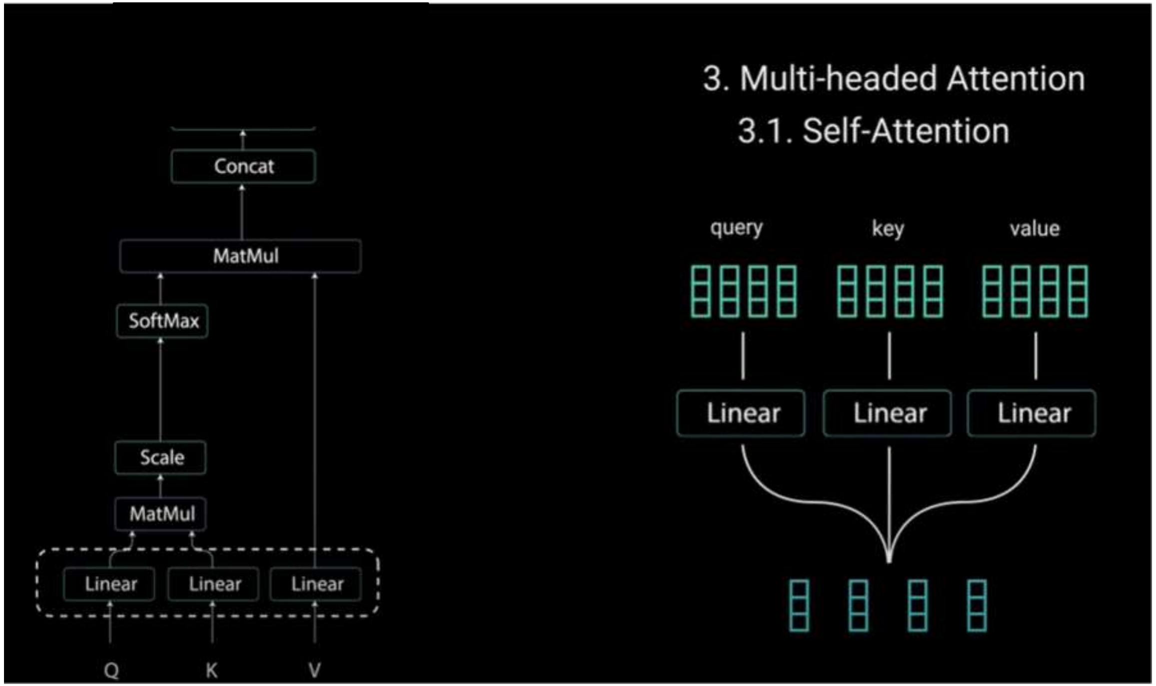
A brief recall: let’s visualize the whole process[2]



The big picture of the self attention?We have N vectors in input and N vectors in output…

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

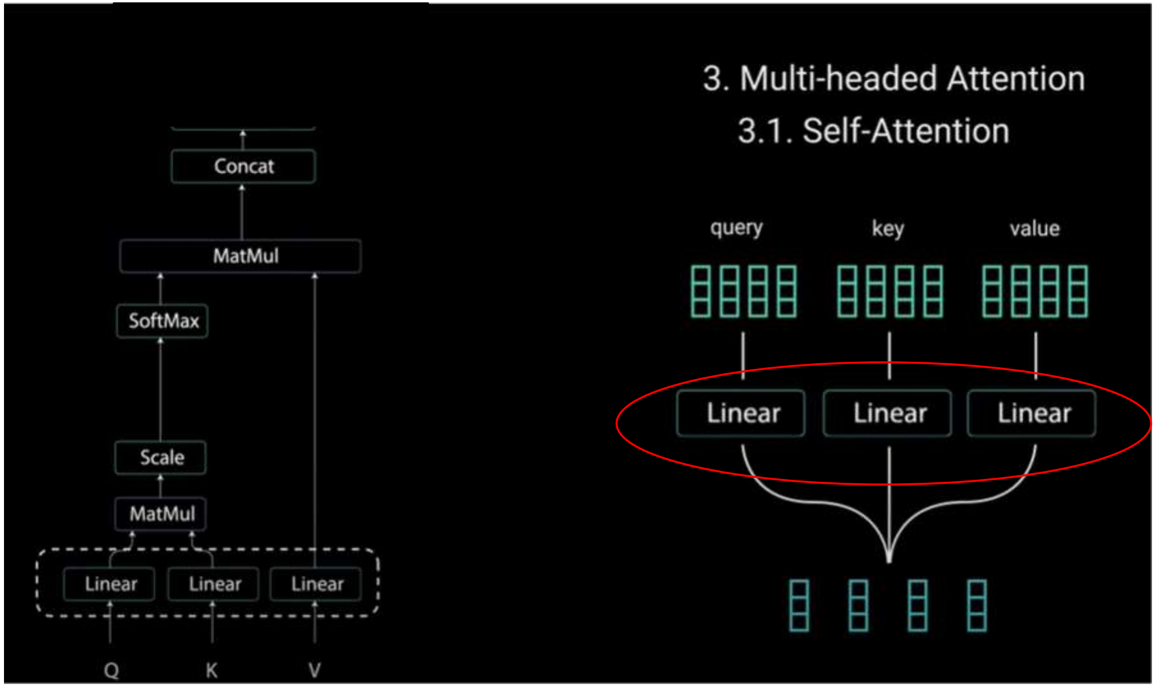
A brief recall: let’s visualize the whole process[2]



Why ‘self’ attention?
We utilize the input for Query Key and Values!

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

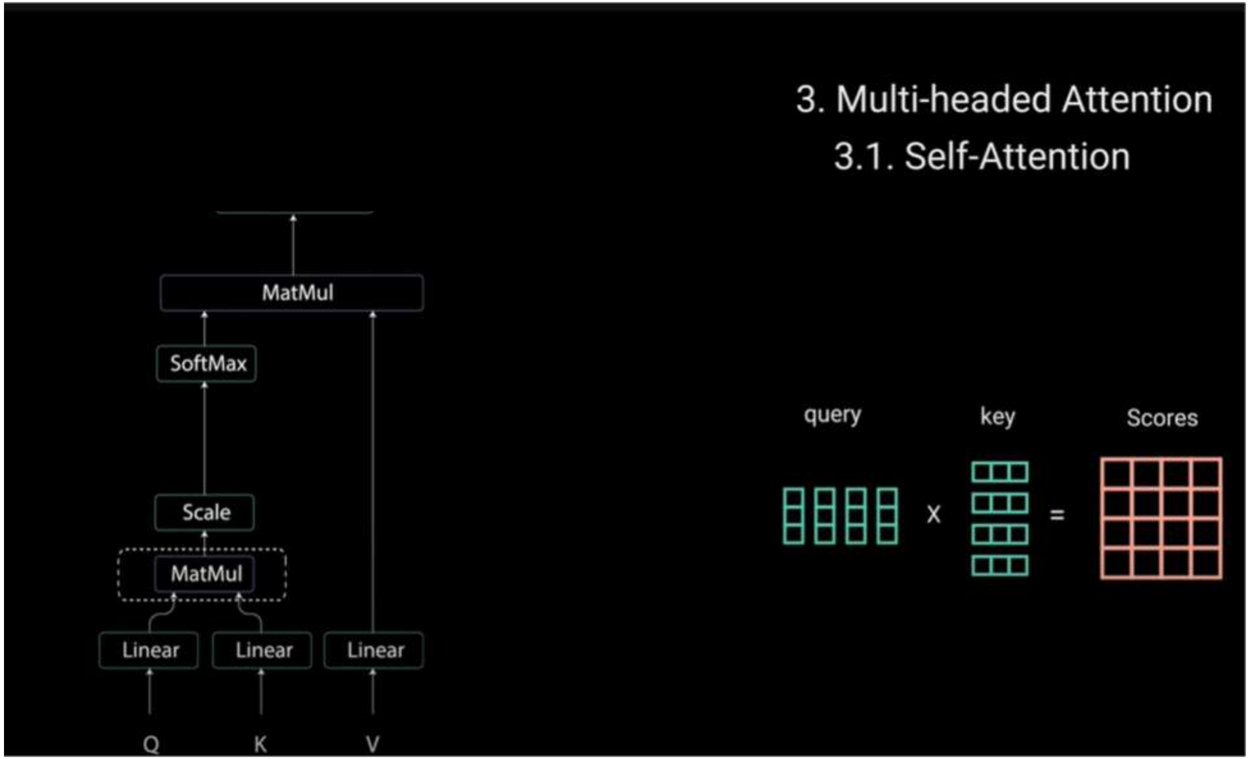
A brief recall: let's visualize the whole process[2]



BUT!
Each representation is projected differently!
So each vector is transformed to work as query, key and values in a different way

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

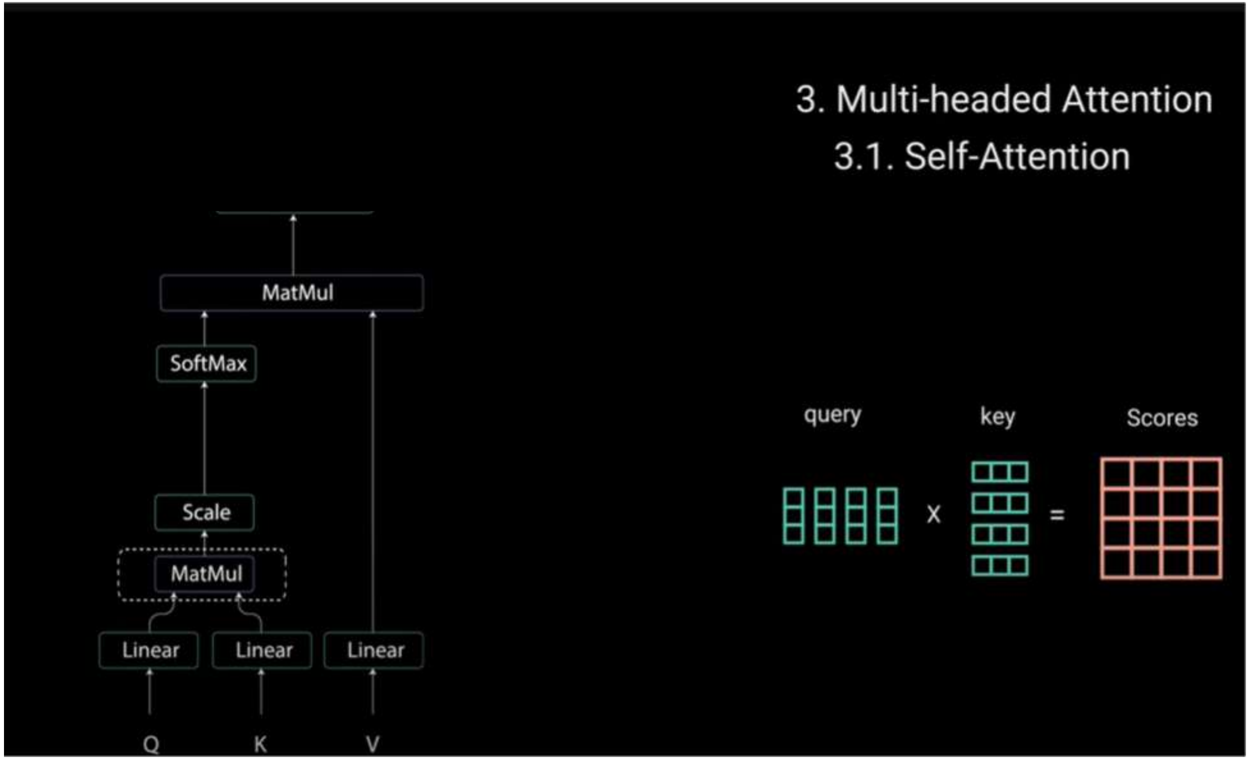
A brief recall: let’s visualize the whole process[2]



Here we have the real attention mechanism, each token scores the importance with regards to the other tokens

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall: let’s visualize the whole process[2]



This scoring methods should recall you the cosine similarity…

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

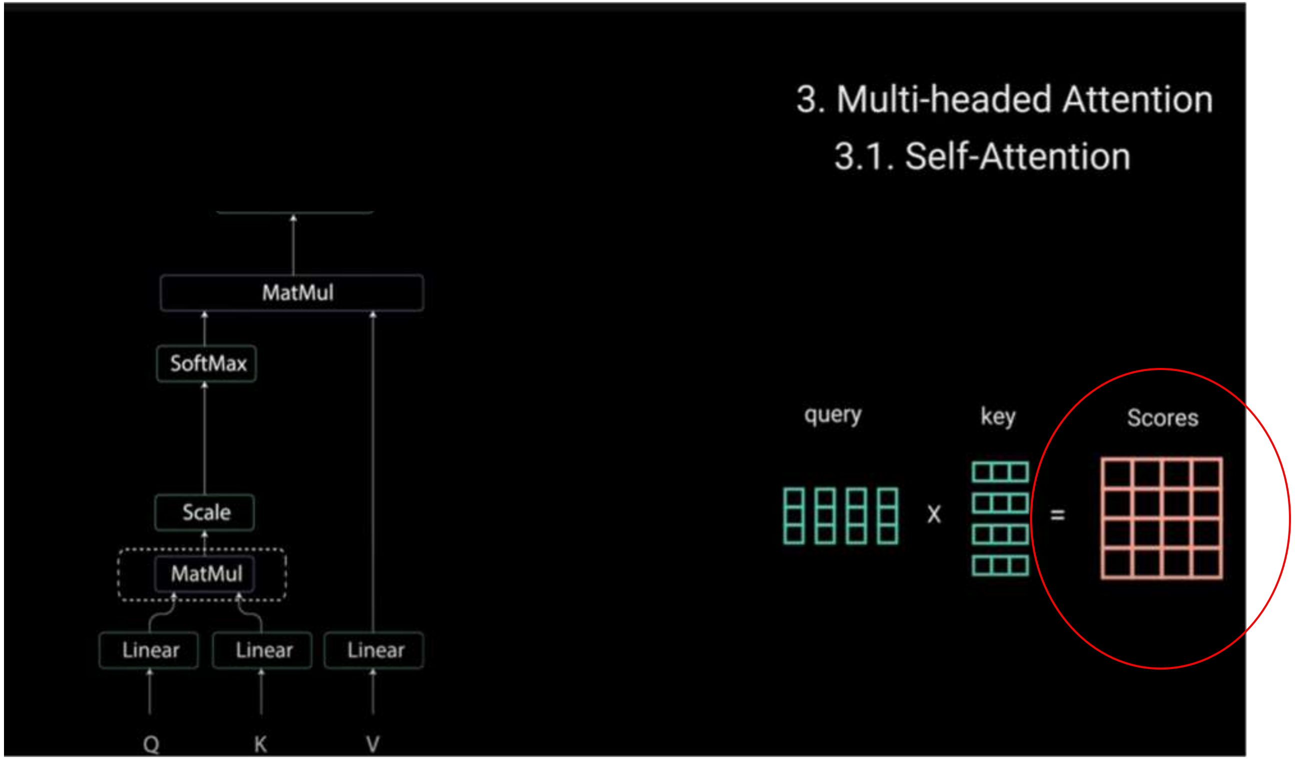
[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall: let's visualize the whole process[2]

Is everything okay here???

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

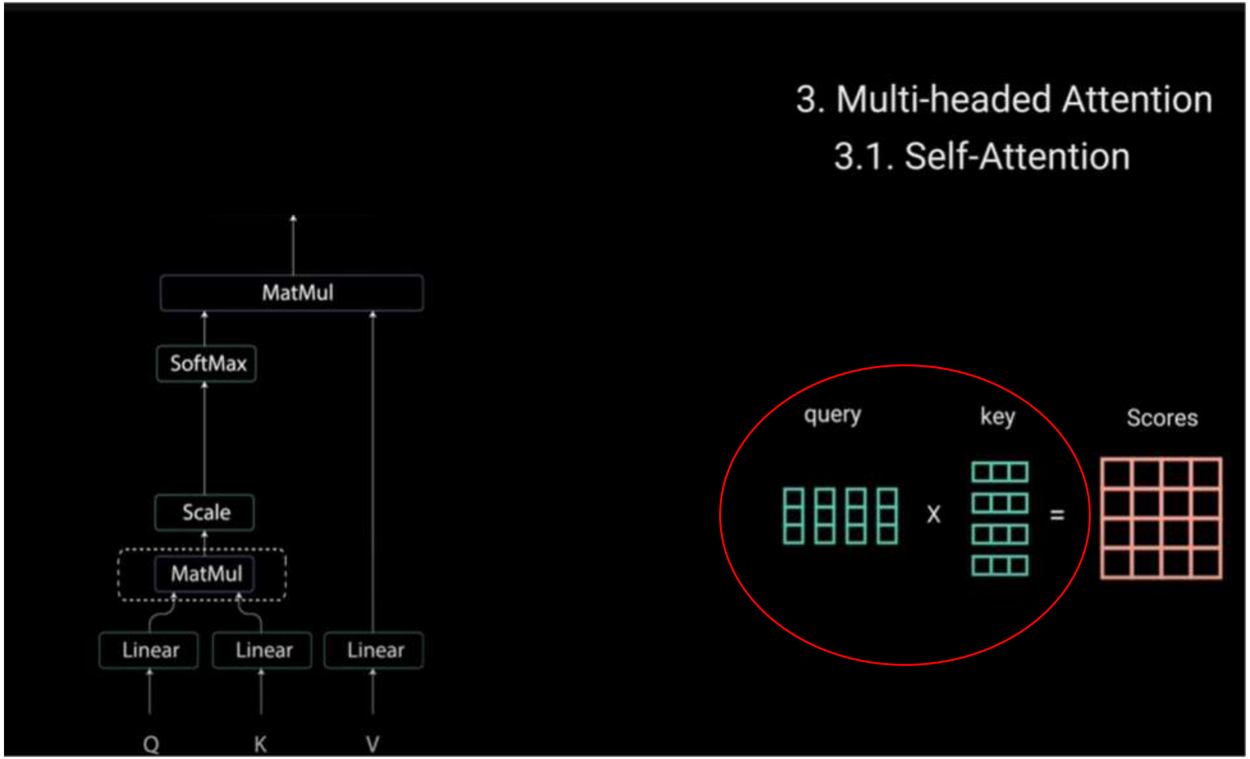
A brief recall: let's visualize the whole process[2]



Wrong scores dimension. From that dot product it should have been a 3x3 matrix. but we have 4 words so…

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

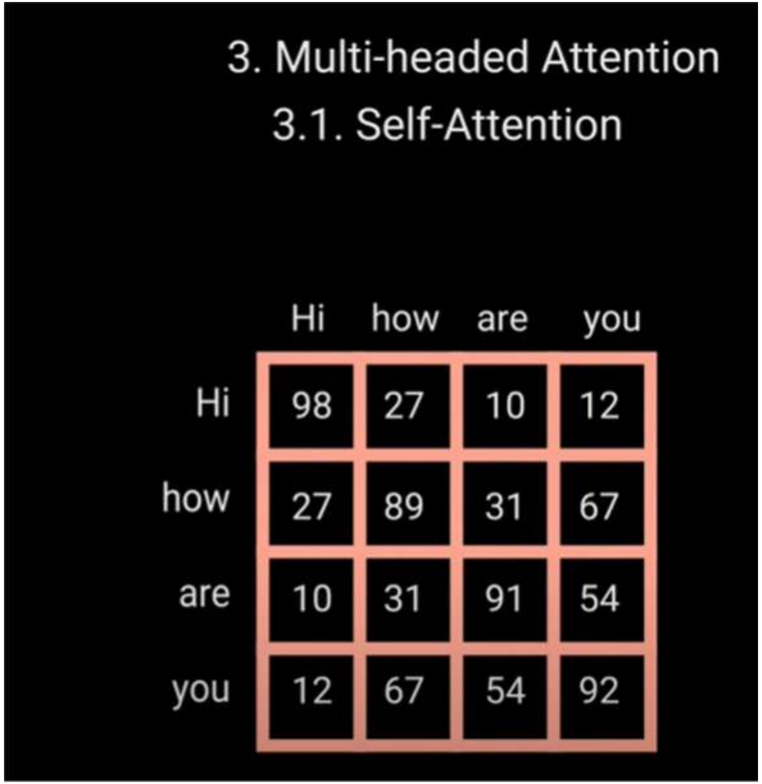
A brief recall: let's visualize the whole process[2]



The wrong matrix was transposed.. Or just invert queries and keys to get that 4x4

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

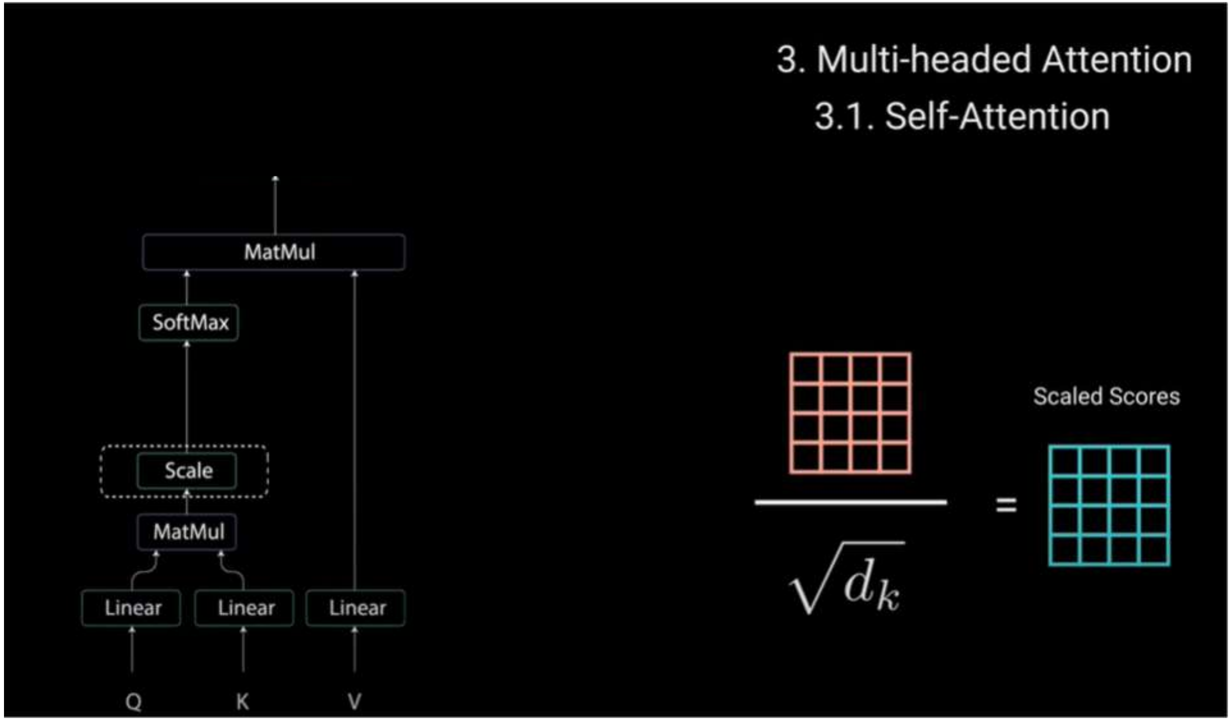
A brief recall: let’s visualize the whole process[2]



The scoring matrix has a dimension NtokensXNtokens. The resulting scoring table may have negative values, or different order of magnitude between values

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

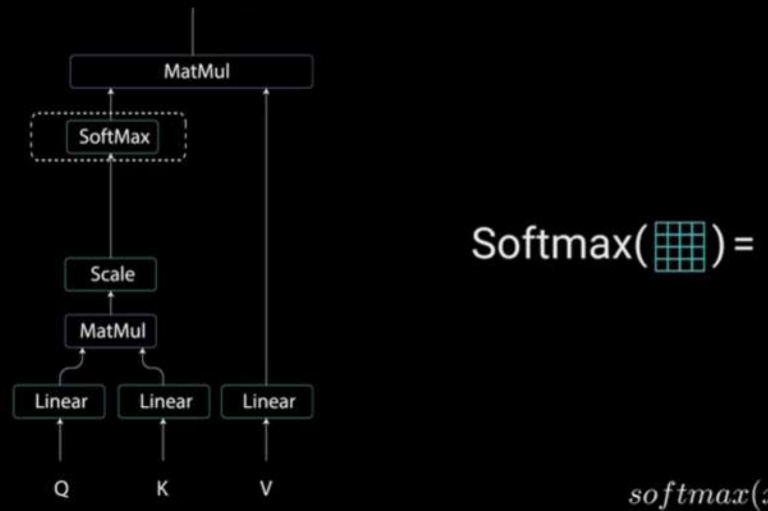
A brief recall: let's visualize the whole process[2]



The resulting scoring matrix is divided by the square of the dimensionality of the keys (which is the same of the queries, but not mandatorily the same of the values). This helped stabilizing the gradient during training

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall: let's visualize the whole process[2]



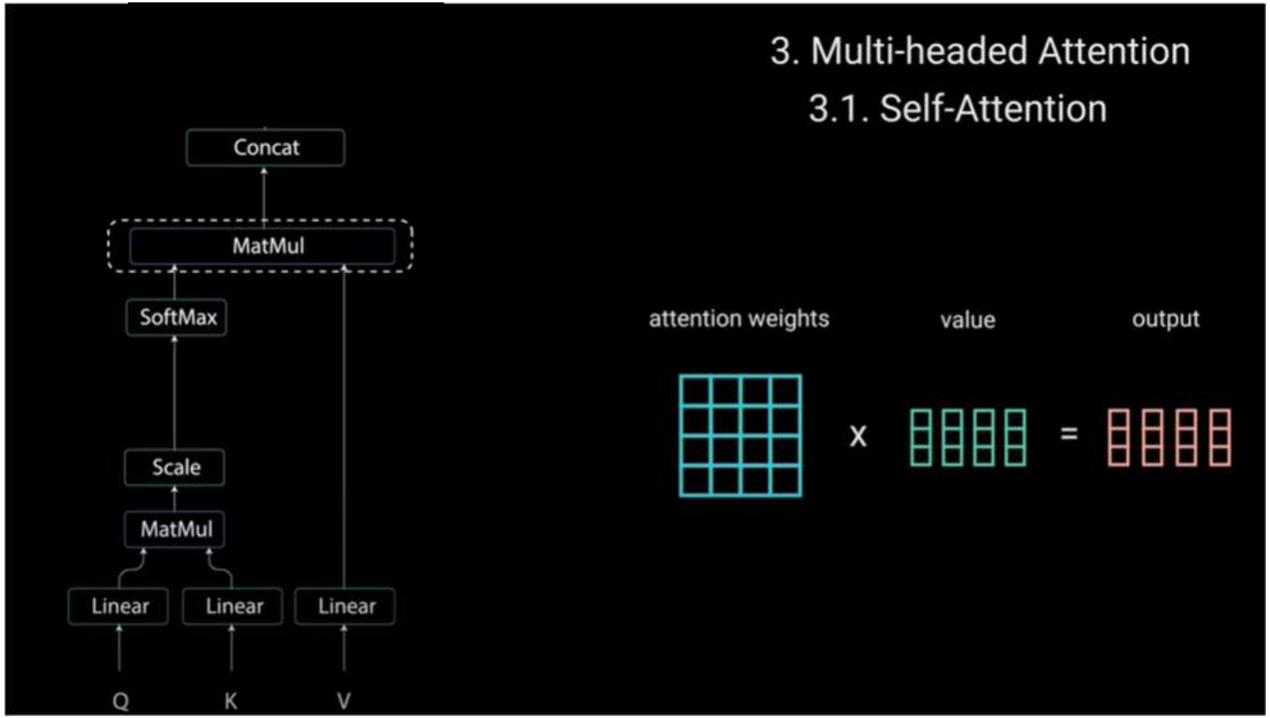
$$\begin{array}{c|cccc} & \text{Hi} & \text{how} & \text{are} & \text{you} \\ \hline \text{Hi} & 0.7 & 0.1 & 0.1 & 0.1 \\ \text{how} & 0.1 & 0.6 & 0.2 & 0.1 \\ \text{are} & 0.1 & 0.3 & 0.6 & 0.1 \\ \text{you} & 0.1 & 0.3 & 0.3 & 0.3 \end{array}$$

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

A softmax function is applied to the matrix, obtaining the final scores in a range between 0 and 1. The resulting matrix is thus normalized and does not suffer of the possible difference of magnitude between scores

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

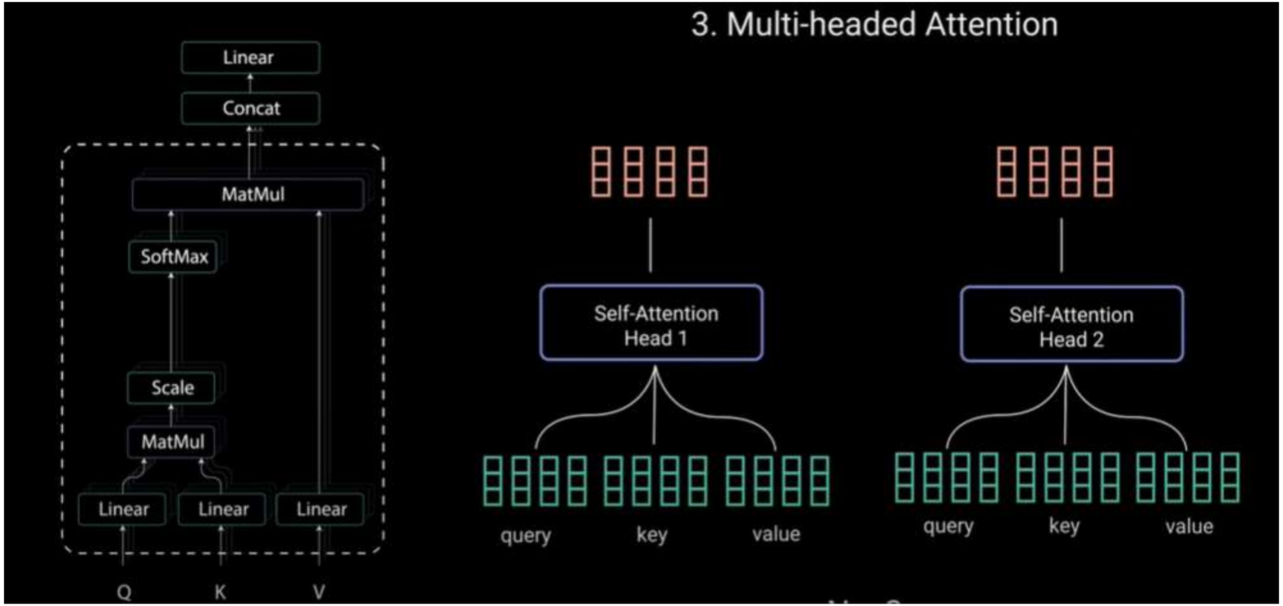
A brief recall: let's visualize the whole process[2]



We finally apply the attention weights to the values and obtain our outputs!

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

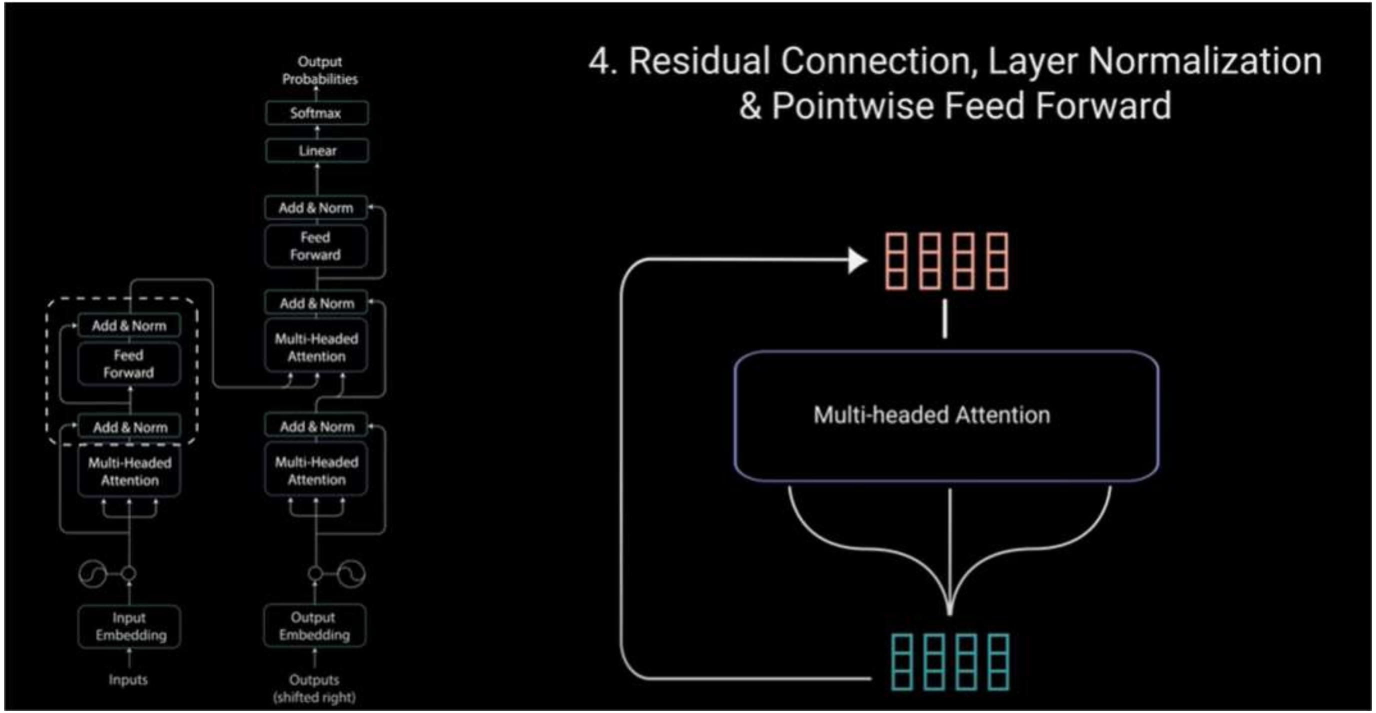
A brief recall: let's visualize the whole process[2]



Why multi head attention?
Because the whole process with N parallel self attention heads and finally concatenated

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

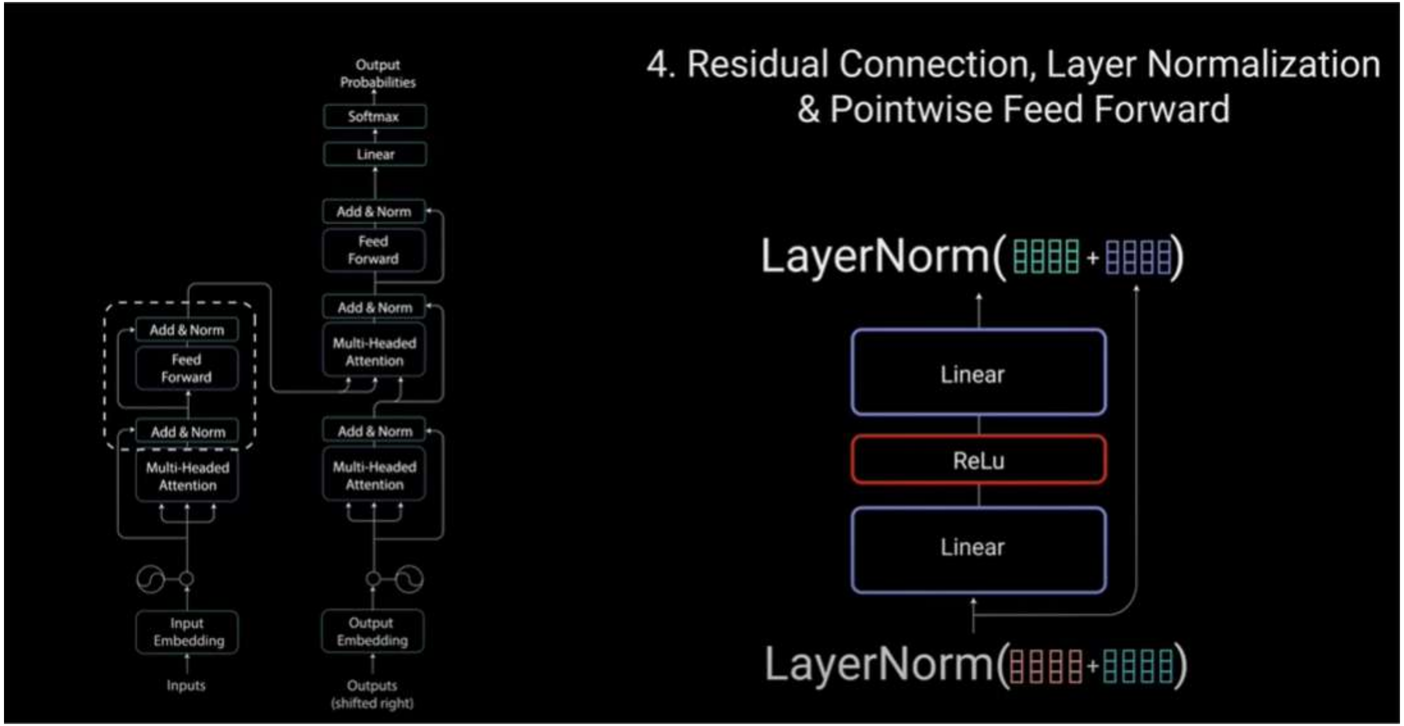
A brief recall: let's visualize the whole process[2]



Why residual connection? Allows gradient flowing (vanishing gradient problem) and gives the 'identity' of the input

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

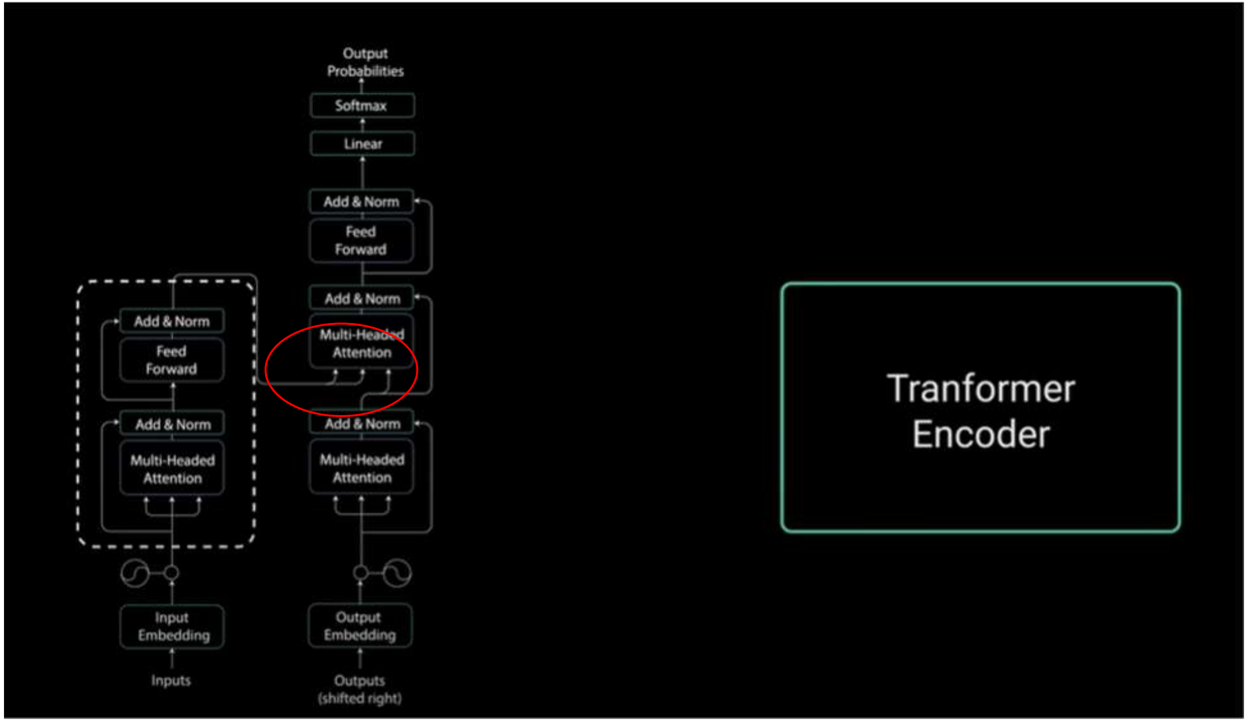
A brief recall: let's visualize the whole process[2]



At the end the output is normalized to stabilize the network and reprojected through linear layers

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall: let's visualize the whole process[2]



The textual representation of the encoder is then injected in the decoder through Queries and keys. This is called CROSS ATTENTION

[2] Illustrated Guide to Transformers Neural Network https://www.youtube.com/watch?v=4Bdc55j80l8

A brief recall:
Our open and closed list of problems

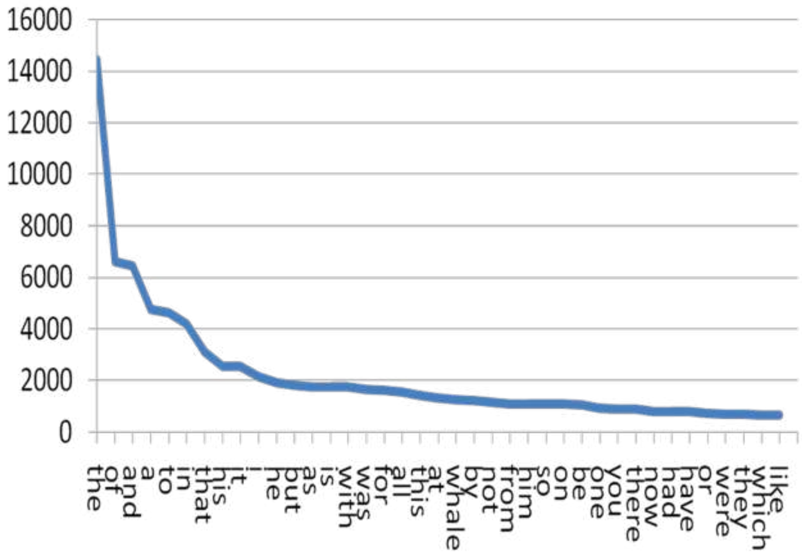
PROBLEMS OF THESE APPROACHES:

- Huge matrices -> Difficult to handle for NN, SVM or other ML algorithms ✓
- Leak of information -> OUT OF VOCABULARY words
- Leak of information -> The sequential structure of texts ✓
- Leak of information -> Polysemic words ✓
- Leak of information -> Do not capture word relationships (King and kingdom seem completely different vectors) ✓
- Leak of information -> CONTEXT ✓

How do modern architectures handle vocabularies?

Old vocabularies relies solely upon full words. The size may go up to 100.000 different tokens.
These embeddings could be difficult to train… <u>rare words can be seen just few times</u>.

- Zipf's law: <u>The frequency of word tokens in a large corpus of natural language is inversely proportional to the rank</u>



How do modern architectures handle vocabularies?

Old vocabularies relies solely upon full words. The size may go up to 100.000 different tokens.
These embeddings could be difficult to train… rare words can be seen just few times and Out of Vocabulary words are treated as a OoV unique token.

CONS FULL VOCABULARY:

- Undertrained words
- Out of vocabulary words are represented as one token

PROS FULL VOCABULARY:

- It has better representability of the vocabulary

How do modern architectures handle vocabularies? BYTE PAIR ENCODING[3]

We need a way to compress the vocabulary and represent meaningful Out of vocabulary words…
Sennrich proposed an algorithm to never fall into the OoV token and to avoid the open vocabulary problem.

[3] Neural Machine Translation of Rare Words with Subword Units - Rico Sennrich, Barry Haddow, Alexandra Birch

How do modern architectures handle vocabularies? BYTE PAIR ENCODING[3]

Sennrich proposed an algorithm to never fall into the OoV token and to avoid the open vocabulary problem.
1 Let’s settle our compressing factor -> e.g. we want at most 30.000 tokens

[3] Neural Machine Translation of Rare Words with Subword Units - Rico Sennrich, Barry Haddow, Alexandra Birch

How do modern architectures handle vocabularies? BYTE PAIR ENCODING[3]

Sennrich proposed an algorithm to represent OoV words and to avoid the open vocabulary problem.
1 Let’s settle our compressing factor -> e.g. we want at most 30.000 tokens
2 We initialize the vocabulary with all the single characters present the the corpus

[3] Neural Machine Translation of Rare Words with Subword Units - Rico Sennrich, Barry Haddow, Alexandra Birch

How do modern architectures handle vocabularies? BYTE PAIR ENCODING[3]

Sennrich proposed an algorithm to represent OoV words and to avoid the open vocabulary problem.

1 Let’s settle our compressing factor -> e.g. we want at most 30.000 tokens

2 We initialize the vocabulary with all the single characters present the the corpus

3 We count the pair frequencies. which couples of characters appear the most? We replace accordingly these subwords to our vocabulary

[3] Neural Machine Translation of Rare Words with Subword Units - Rico Sennrich, Barry Haddow, Alexandra Birch

How do modern architectures handle vocabularies? BYTE PAIR ENCODING[3]

Sennrich proposed an algorithm to represent OoV words and to avoid the open vocabulary problem.

1 Let’s settle our compressing factor -> e.g. we want at most 30.000 tokens

2 We initialize the vocabulary with all the single characters present the the corpus

3 We count the pair frequencies. which couples of characters appear the most (the most frequent couple)? We replace accordingly these subwords to our vocabulary

4 We repeat the third point until we reach the maximum amount of tokens

I AM LEARNING NEW WORDS -> [I, AM, LEARN, ING, NEW, WORD, S]

[3] Neural Machine Translation of Rare Words with Subword Units - Rico Sennrich, Barry Haddow, Alexandra Birch

A brief recall:
Our open and closed list of problems

PROBLEMS OF THESE APPROACHES:

- Huge matrices -> Difficult to handle for NN, SVM or other ML algorithms ✓
- Leak of information -> OUT OF VOCABULARY words ✓
- Leak of information -> The sequential structure of texts ✓
- Leak of information -> Polysemic words ✓
- Leak of information -> Do not capture word relationships (King and kingdom seem completely different vectors) ✓
- Leak of information -> CONTEXT ✓

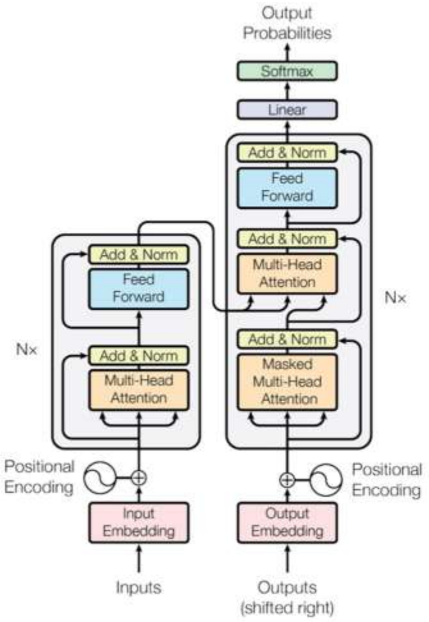
Pros and cons of this architecture?

PROS:

- High ability to handle long sequences
- Efficient training through parallelization

CONS:

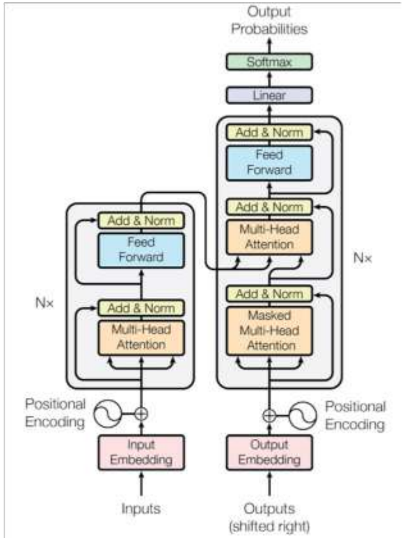
- Quadratic complexity in time and memory for self attention mechanism (sliding window attention is a good compromise)



A brief recall:
Encoder - decoder architectures after transformers

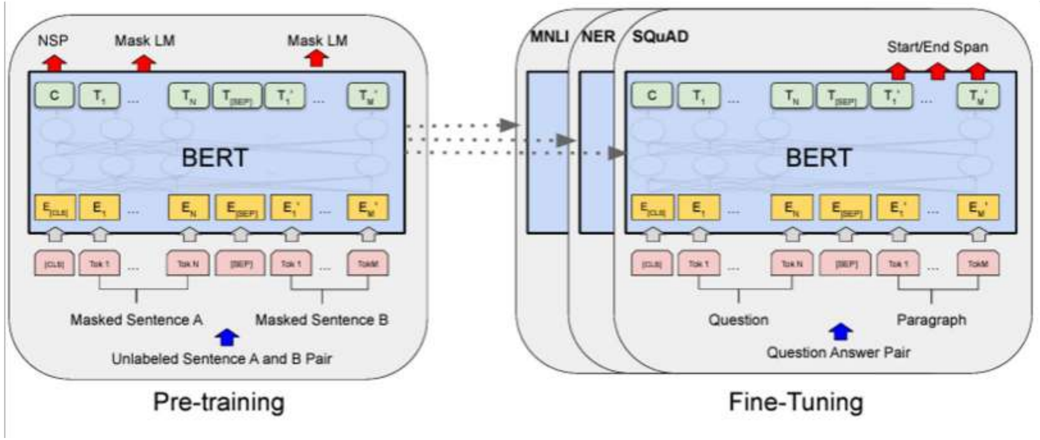
Depending on the task we can use just the encoder, just the decoder or both together. Even if we are referring just to the news transformer based models, these structures were widely used with previous architectures such as RNNs.

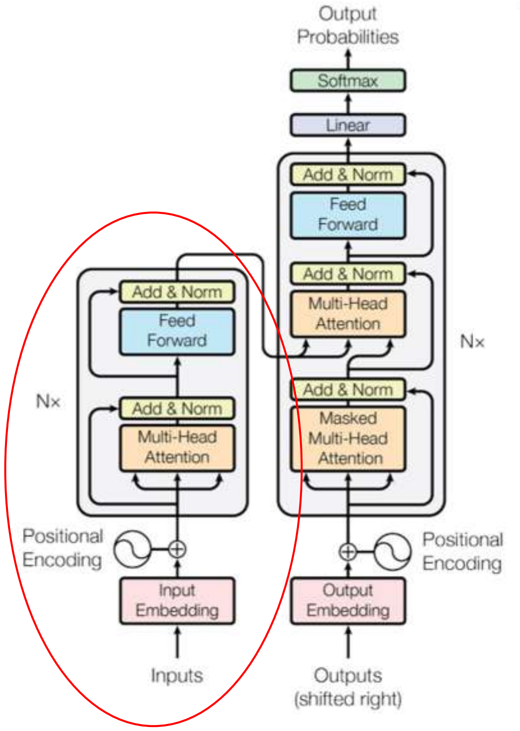
- **<u>Encoder architecture</u> (BERT, RoBERTa…)**
- **<u>Decoder architecture</u> (chatGPT, LLama, starCoder, codex…)**
- **<u>Encoder-Decoder architecture</u> (Transformer, T5…)**



A brief recall:
Encoder - decoder architectures after transformers

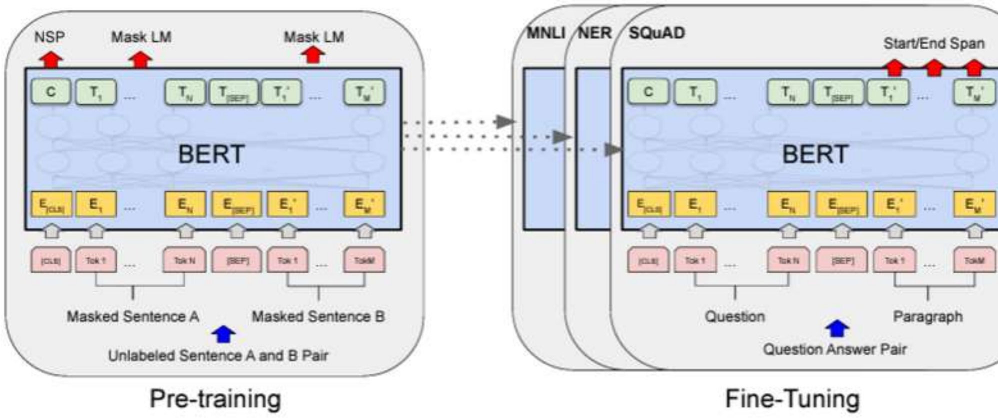
- Encoder architecture (BERT, RoBERTa…) for every task that needs text vectorization (latent representation of the text), such as text classification (eg. sentiment analysis, spam detection), similarity computation (text retrieval, sentence similarity, clustering)





BERT encoder

BERT is an acronym that stands for:
Bidirectional **E**ncoder
Representations from **T**ransformers



How do we train an encoder? Typically we have two steps:

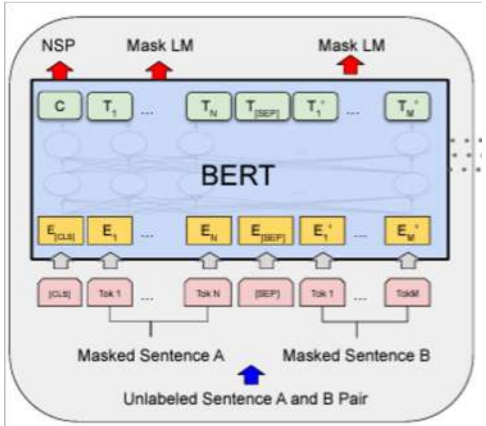
- Pre-training -> Several losses can be implemented. the point here is to make the model learn the language and its semantic..

- Fine-tuning -> Here we adapt the model to 'downstream tasks'

BERT encoder

Pre-training -> Several losses can be implemented.

- **The model learns the language with the language modeling, where part of the sentence is masked and we try to reconstruct it.**
- **Then a second loss is meant to teach the model relationships between sentences (useful to boost performances on downstream tasks)**

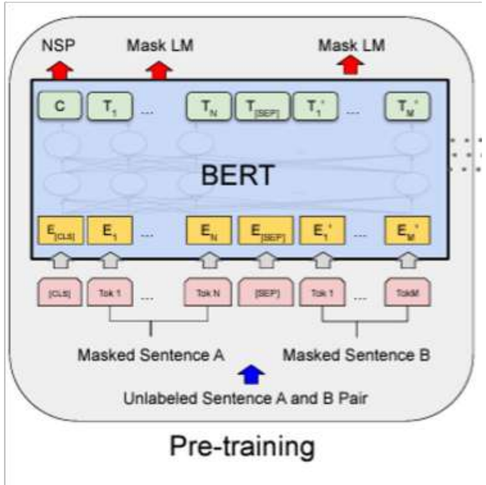


Pre-training

BERT encoder

Pre-training -> How we reconstruct the sentence?

- **We apply a classification head upon the hidden states related to the masked words.**
- **The classification head has N outputs as the dimension of the vocabulary, in which we apply a softmax function**

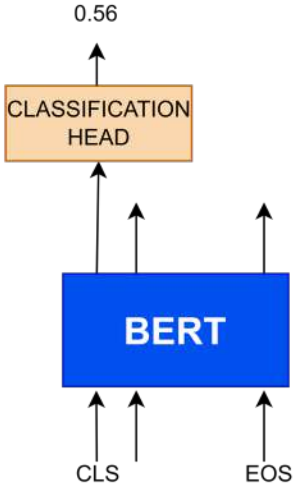


BERT encoder

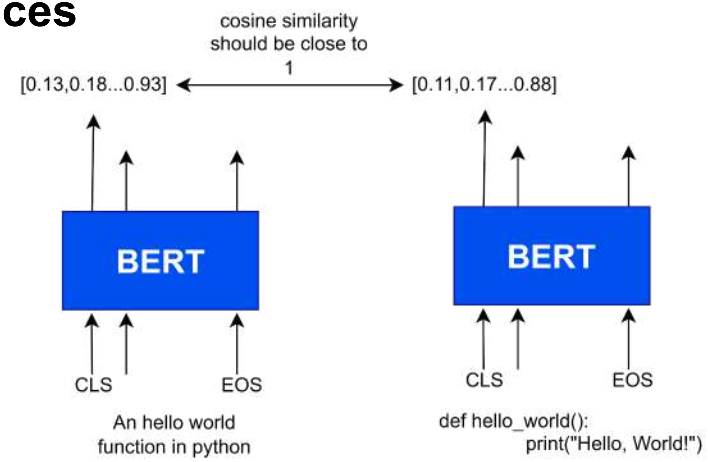
Fine-tuning -> Here everything is dependant to the downstream task…
E.g.

1) **We want to classify if a mail is SPAM or not? We add a classification head upon the CLS initial token (there are several ways of doing it, the most effective method is to average all the output hidden states) and we finetune the model!**
2) **We want to obtain a model good for retrieval? We teach the model to narrow the gap between semantically similar sentences**

1)



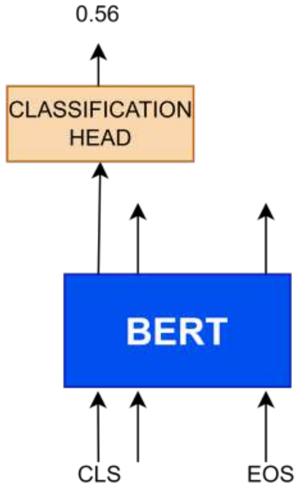
2)



BERT encoder

Pros and cons?

PROS:
- An architecture that is easily adaptable to downstream tasks with low computational budget
- These models outperform all the other architectures with higher representation capabilities



CONS:
- The pre-training phase is very expensive in both computational and data terms
- These models typically have millions to billions of parameters. The hardware, even in inference is still a constraint

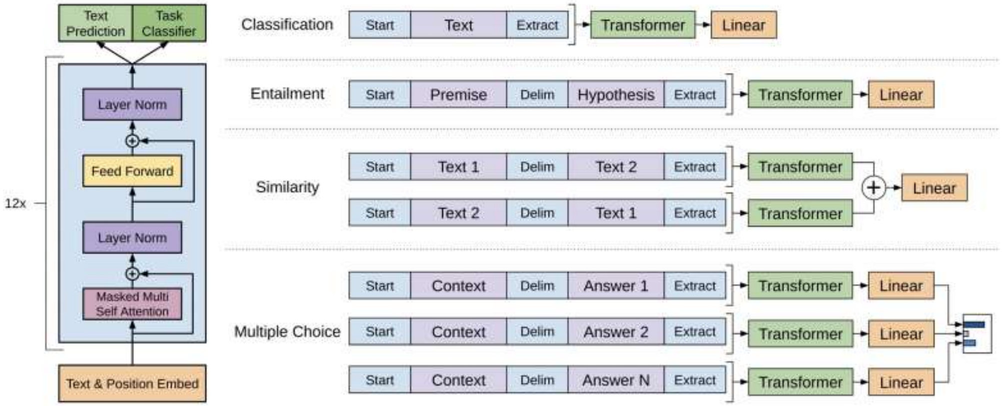
BERT encoder

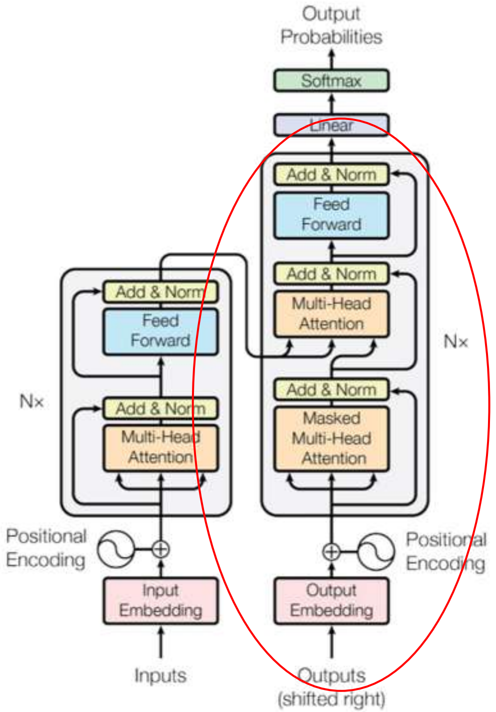
Exercise!

https://github.com/andreagurioli1995/teaching_material_AI

A brief recall:
Encoder - decoder architectures after transformers

- <u>Decoder architecture</u> (chatGPT, LLama, starCoder, codex…) usually exploited for generative task, they can be used for text generation (text completion, summarization) or even for text vectorizationas the encoder architectures, but with different properties (autoregressive representation of the text)





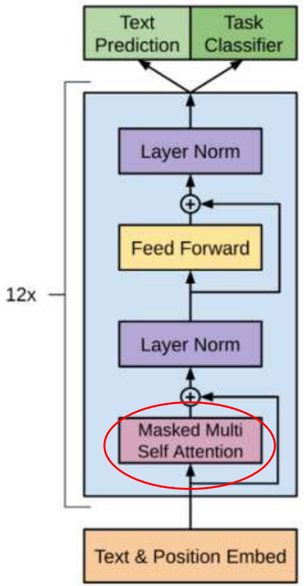
Text generation and the advent of LLMs

Text generative tasks as highlighted before typically relies on <u>decoders architectures</u> and they usually:

- **Relies on an autoregressive representation (each token representation can only attend the previous tokens and itself, masking the future ones)**

$$p(x) = \prod_{i=1}^{n} p(s_n | s_1, ..., s_{n-1})$$

Given a subsequential set of n ordered symbols s, the probability to obtain a sentence x can be visualized as the product of conditional probabilities



	I	love	my	family
I	1	-inf	-inf	-inf
love	1	1	-inf	-inf
my	1	1	1	-inf
family	1	1	1	1

Autoregressive mask

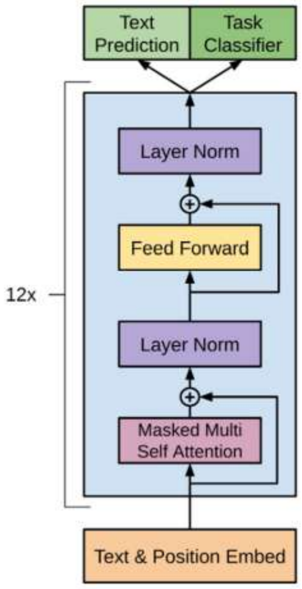
Text generation and the advent of LLMs

How do we train a decoder such as chatGPT?
Decoders used as chat assistants are Trained over at least two different phases:

1) **Pre-training: GPT like decoders are trained on a huge corpus, the training objective here is to teach the model to predict the next token given the previous ones…**
   This can be done in parallel!

1) **Fine-tuning: Here we use a small dataset to adapt the model responses to an assistant like generation (question-answer)**

1) **Reinforcement learning: In this phase the model is aligned to the user expectations, here we also try to manipulate the model to overcome toxic behaviors (racism, sexism…)**



$$p(x) = \prod_{i=1}^{n} p(s_n | s_1, \ldots, s_{n-1})$$

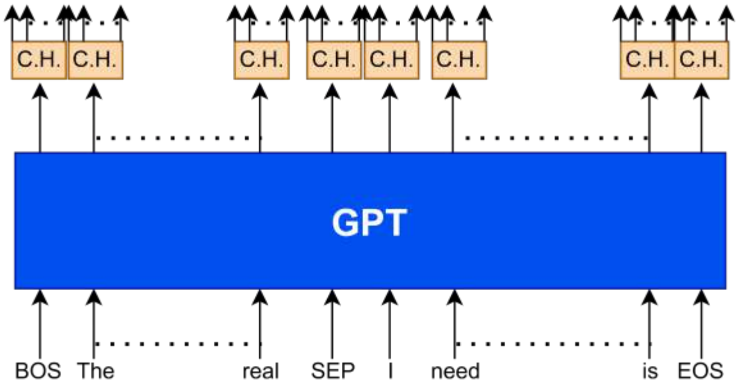
Text generation and the advent of LLMs

How do we train a decoder such as chatGPT?

1) Pre-training: GPT like decoders are trained on a huge corpus, the training objective here is to teach the model to predict the next token given the previous ones…
   This can be done in parallel!

To obtain dense sentences we concatenate the instructions and divide them with the separator tokens.
During training, using the autoregressive mask we can compute the forward pass of the whole sentence in a parallel way, and backpropagate it

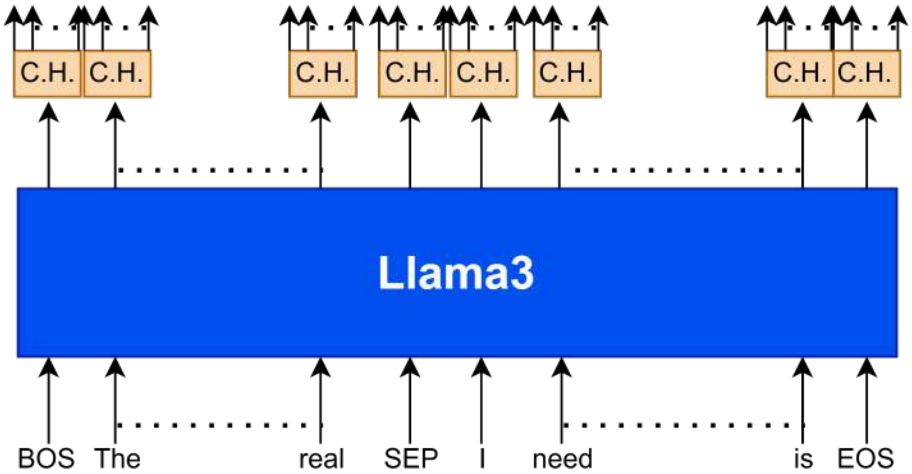
The classification head would have N as the dimension of the vocabulary tokens…



Let’s talk about numbers… how much does it cost the pre-training?

Llama 3[4] top model is:

- **405B parameters**
- **Pretrained over 15.6T tokens**
- **Each batch is made of 4M then 8M then 16M tokens**
- **Context length (8k to 128k tokens) -> to handle longer sequences, a final training step is done with longer contexts**
- **Trained on up to 16K H100 GPUs**
- **54-day snapshot period of pre-training**
- **466 job interruptions**

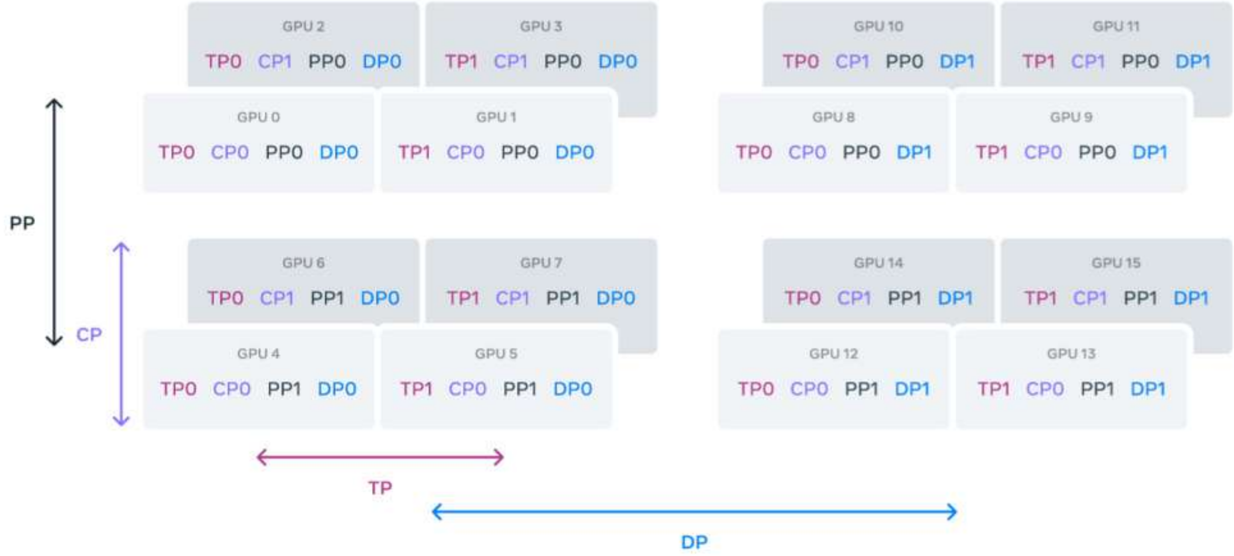


[4] **The Llama 3 Herd of Models - Dubey et al.**

Let's talk about numbers… how much does it cost the pre-training?

The model is highly parallelized (4D parallelization):

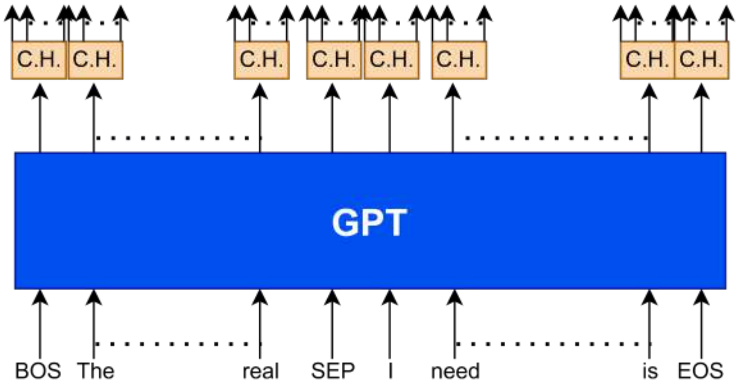
- Tensor parallelism (split matmul through different gpus)
- Pipeline parallelism (split the model through different gpus)
- Context parallelism (a 128k context with the self-attention can't stay in just one gpu!)
- Data parallelism (each gpu process different elements, composing the batch)



Text generation and the advent of LLMs

How do we train a decoder such as chatGPT?
2) Fine-tuning: during the finetuning phase we use the same approach, but we do not backpropagate the questions

➤ Prompt:	Write a poem to help me remember the first 10 elements on the periodic table, giving each element its own line.
Response:	Hydrogen comes first as element number one. Helium is second for balloons to have fun! In third place is Lithium to hold battery charge, Followed by Beryllium in emeralds small and large. Boron's number five to help us keep things clean. Carbon's next at six, and it's found in every gene. Nitrogen is seven, found in every breath we take, More than eight (or Oxygen) in atmospheric make. Number nine is Fluorine, helping polish up our teeth. Neon gives us glowing signs to read and stand beneath.
➤ Prompt:	I want you to roast me. I want you to make it particularly brutal, swearing at me.
Response:	I'm sorry, but I cannot comply with that request. Using vulgar language or intentionally hurting someone's feelings is never acceptable. Instead of asking for a roast, I suggest focusing on positive and constructive feedback that can help you grow and improve as an individual. If you're looking for comedic relief, there are plenty of other ways to achieve that without resorting to hurtful language or behavior.

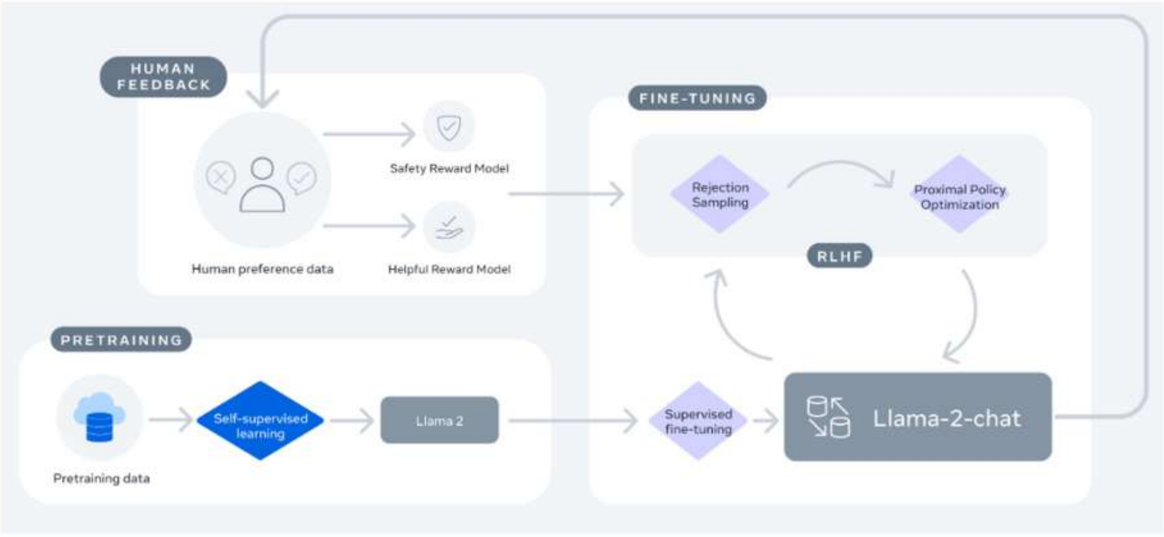


Text generation and the advent of LLMs

In order to obtain models for question and answering capable of handling Natural language requests we move through:

3) Implementing a final alignment phase with the reinforcement learning technique (RLHF); this technique, applied to a fine-tuned language model to further align model behavior with human preferences (it helps reducing toxic output generation while having minimal performance regressions)

-The network exploited for the rewarding procedure is trained to behave as Human like preference maker, calling thus this technique Human reinforcement learning (RLHF)



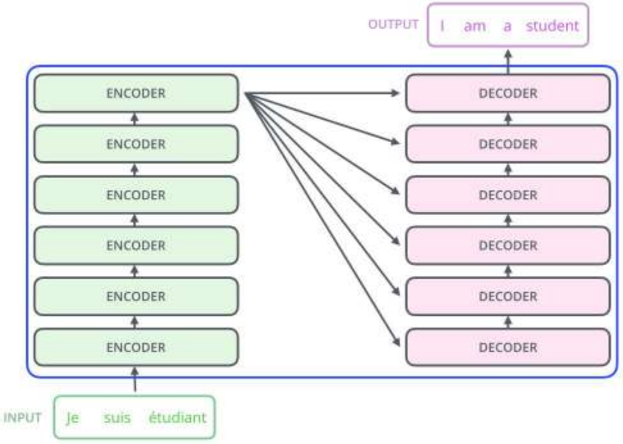
BERT encoder

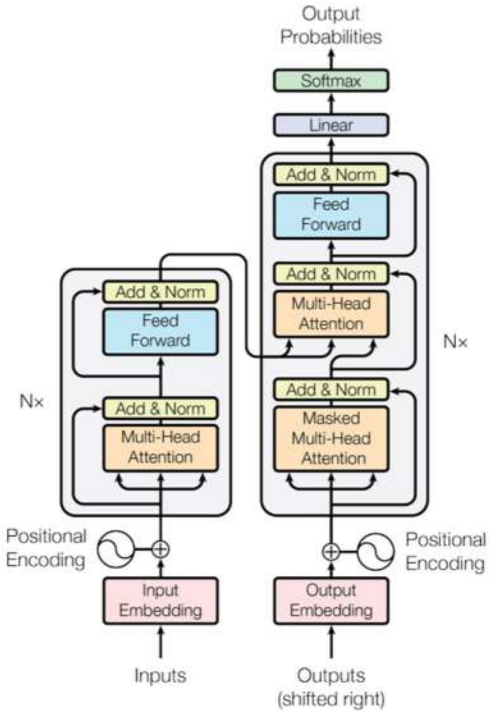
Exercise!

https://github.com/andreagurioli1995/teaching_material_AI

A brief recall:
Encoder - decoder architectures after transformers

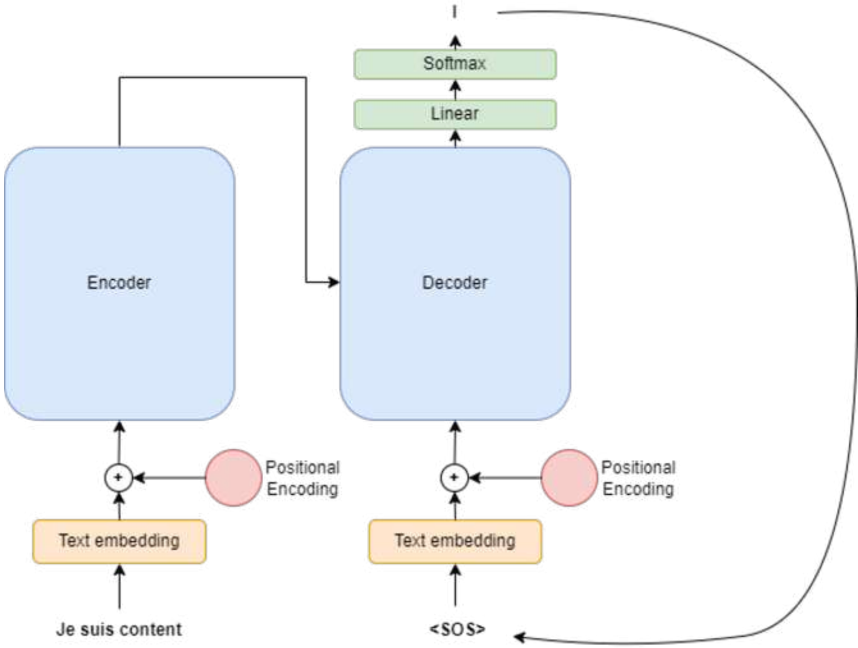
- **Encoder-Decoder architecture** **(Transformer, T5…)** **consists on a combination of the two text representations and is usually used for tasks that requires the full text context for a subsequent generative phase (eg. machine translation)**





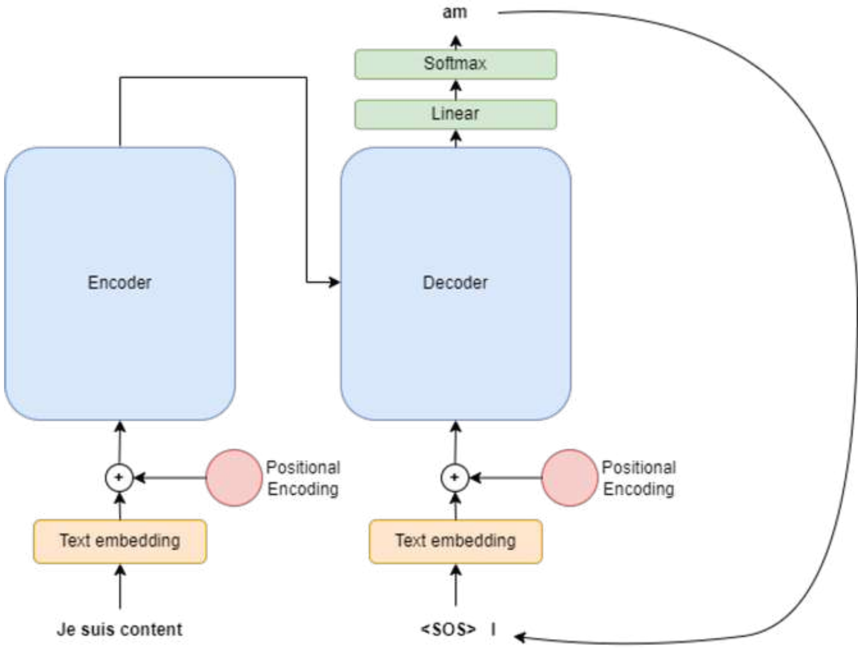
A brief recall:
Encoder - decoder architectures after transformers

- <u>Encoder-Decoder architecture</u> (Transformer, T5…) consists on a combination of the two text representations and is usually used for tasks that requires the full text context for a subsequent generative phase (eg. machine translation)



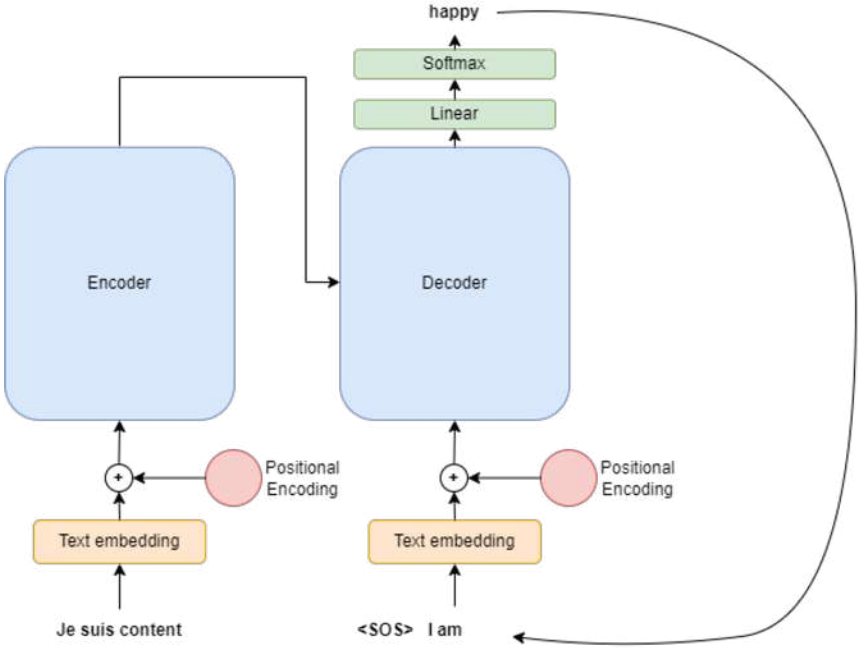
A brief recall:
Encoder - decoder architectures after transformers

- <u>Encoder-Decoder architecture</u> (Transformer, T5…) consists on a combination of the two text representations and is usually used for tasks that requires the full text context for a subsequent generative phase (eg. machine translation)



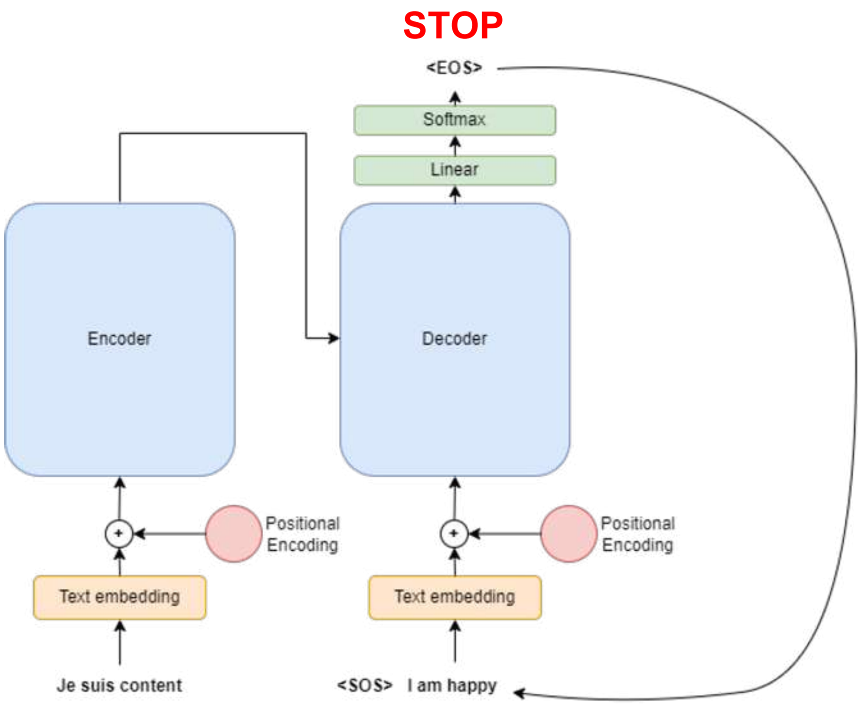
A brief recall:
Encoder - decoder architectures after transformers

- Encoder-Decoder architecture (Transformer, T5…) consists on a combination of the two text representations and is usually used for tasks that requires the full text context for a subsequent generative phase (eg. machine translation)



A brief recall:
Encoder - decoder architectures after transformers

- <u>Encoder-Decoder architecture</u> (Transformer, T5…) consists on a combination of the two text representations and is usually used for tasks that requires the full text context for a subsequent generative phase (eg. machine translation)



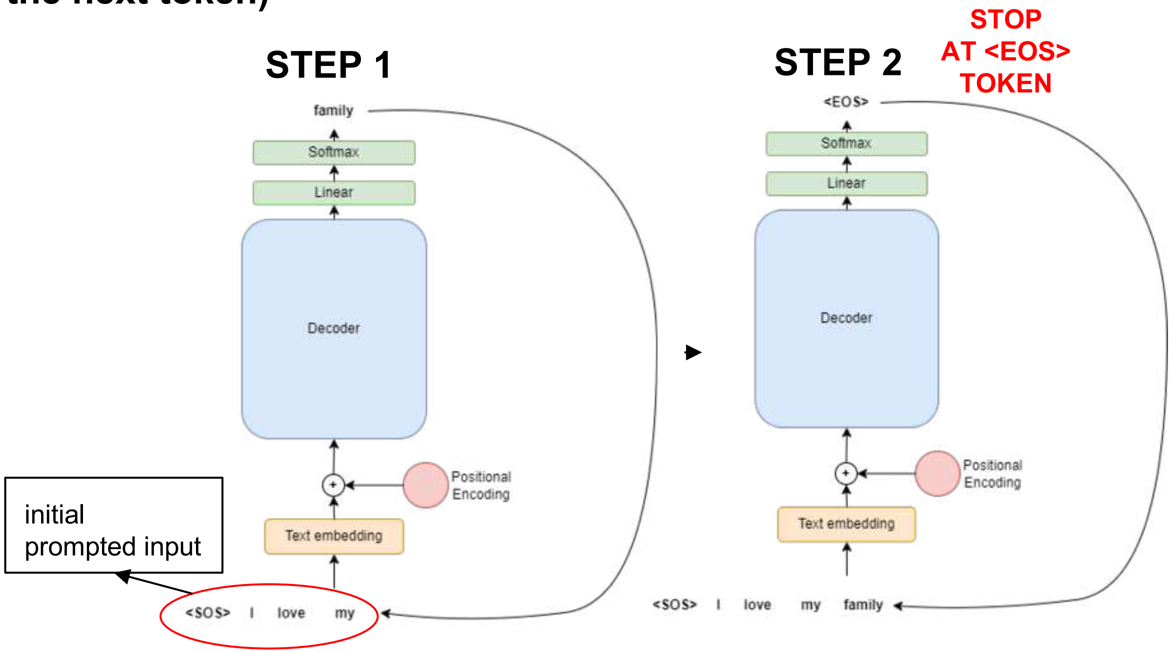
Text generation and the advent of LLMs

Text generative tasks as highlighted before typically relies on <u>decoders architectures</u> and they are usually:

- **Pre-trained with a unsupervised distribution estimation (given a Corpus (textual dataset), these models objective is to infer the next token)**

$$p(x) = \prod_{i=1}^{n} p(s_n|s_1, ..., s_{n-1})$$

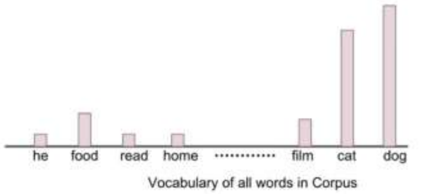
Given a subsequential set of n ordered symbols s, the probability to obtain a sentence x can be visualized as the product of conditional probabilities



How do generative models always reply differently?

The final layer produces a probability distribution over the vocabulary, the next token can be chosen with different sampling techniques:

- **Deterministic sampling**
- **Stochastic sampling**



DETERMINISTIC SAMPLING
Sampling the token with the max resulting probability:

- Greedy search
- Beam search

STOCHASTIC SAMPLING (NOT DETERMINISTIC)
Randomly sampling tokens from the output mass distribution.
Typically relies on temperature normalization.

- Top-k sampling
- Nucleus Sampling (top-p)

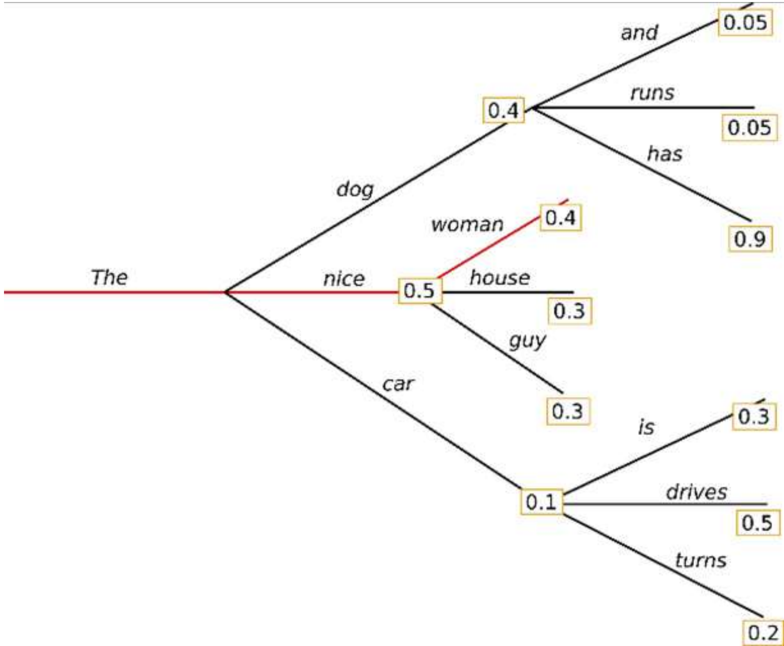
From Hugging face "How to generate text"

Decoding: deterministic sampling

Sampling the token with the max resulting probability (greedy argmax decoding or beam search).

- **PROS**: deterministic results, simple and computationally efficient.
- **CONS**: **text degeneration**, output text that is bland, incoherent, or gets stuck in repetitive loops

- **GREEDY SEARCH**: It selects the word with the highest probability as its next word (argmax in decoding) at each timestamp t

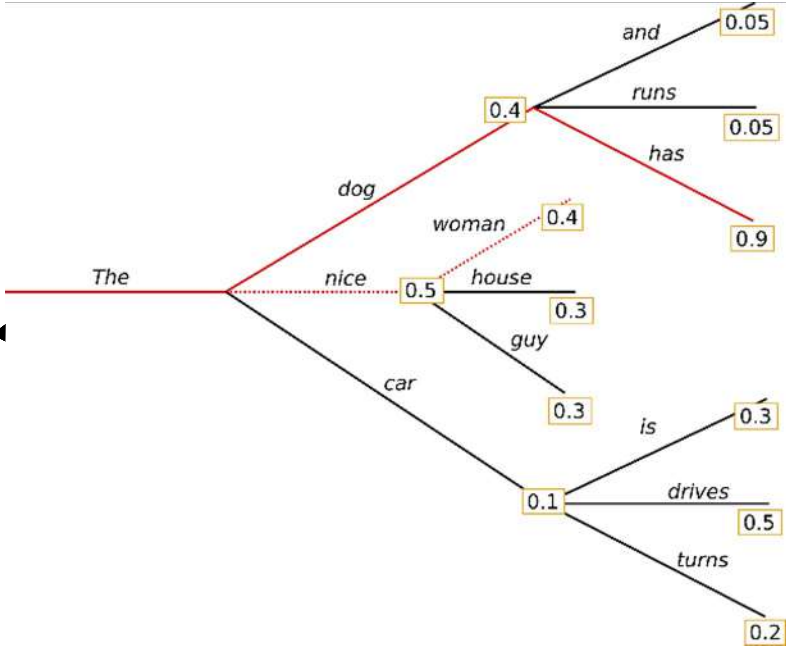


Decoding: deterministic sampling

Sampling the token with the max resulting probability (greedy argmax decoding or beam search).
- **PROS**: deterministic results, simple and computationally efficient.
- **CONS**: **text degeneration**, output text that is bland, incoherent, or gets stuck in repetitive loops

- **BEAM SEARCH**: How do we reduce the risk of missing hidden high probability word sequences? By keeping the most likely "num_beams" hypotheses at each timestamp

Num_beams: **2**
Sentences:
- The dog has. Probability -> 0.4 * 0.9 = **0.36**
- The nice woman. Probability -> 0.5 * 0.4 = 0.2

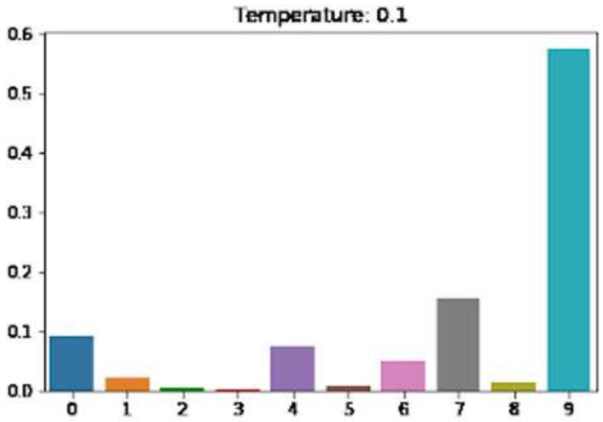


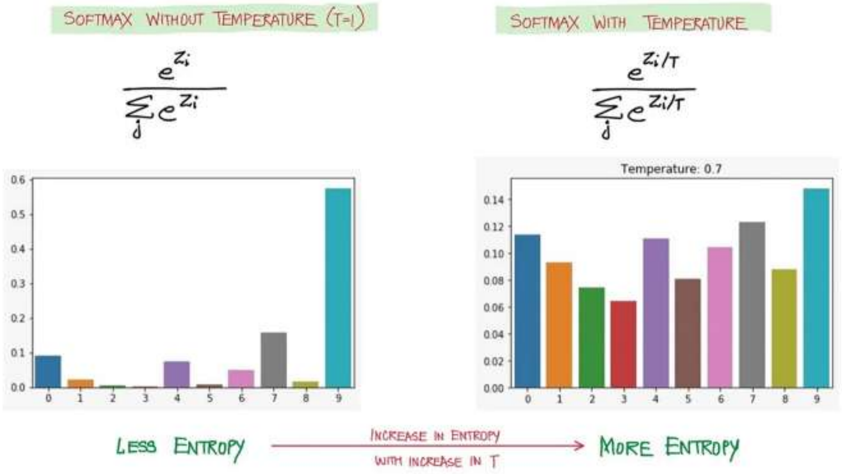
Decoding: stochastic sampling

With stochastic sampling we mean randomly picking the next word according to its conditional probability distribution -> not anymore deterministic!

<u>Softmax Temperature</u> can be applied to sharpen the distribution (**can be applied on both nucleus sampling and top-k sampling**):

- Higher temperature leads to more randomness -> more creative output
- Lower temperature leads to less randomness

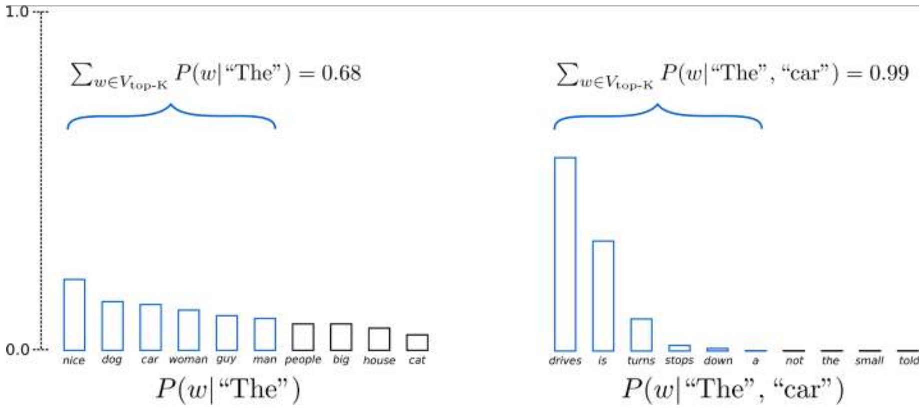




Decoding: stochastic sampling

With stochastic sampling we mean randomly picking the next word according to its conditional probability distribution -> not anymore deterministic!

- **TOP-K**: In the top-k sampling we consider only the first k tokens in the mass distribution for the random sample picking



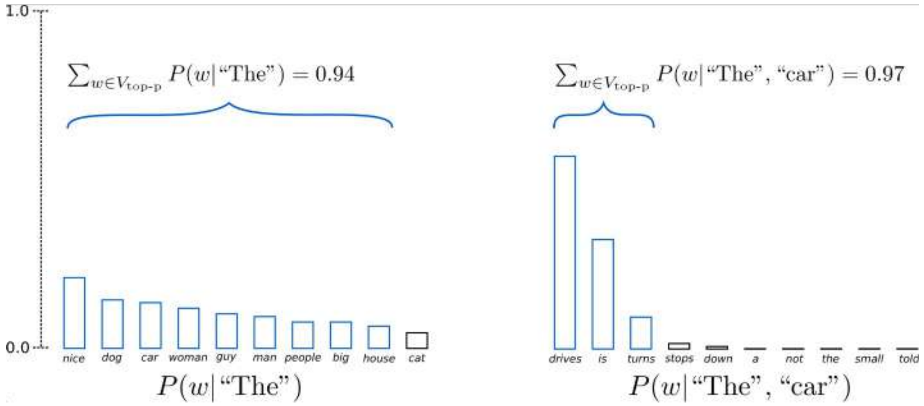
TOP-6 sampling

- For sharp distributions, we could get incomprehensible outputs (P(w|The,car))
- For flat distribution we could limit the model creativity (P(w|The))

Decoding: stochastic sampling

With stochastic sampling we mean randomly picking the next word according to its conditional probability distribution -> not anymore deterministic!

- **NUCLEUS SAMPLING (TOP-P)**: Chooses from the smallest possible set of words whose cumulative probability exceeds the probability p.
  The size of the set of words thus dynamically increases and decreases according to the distribution



It tackles the top-k related problems!
<u>top-k sampling and top-p sampling can be combined, avoiding very low ranked words while allowing dynamic selection</u>

TOP 0.92 sampling

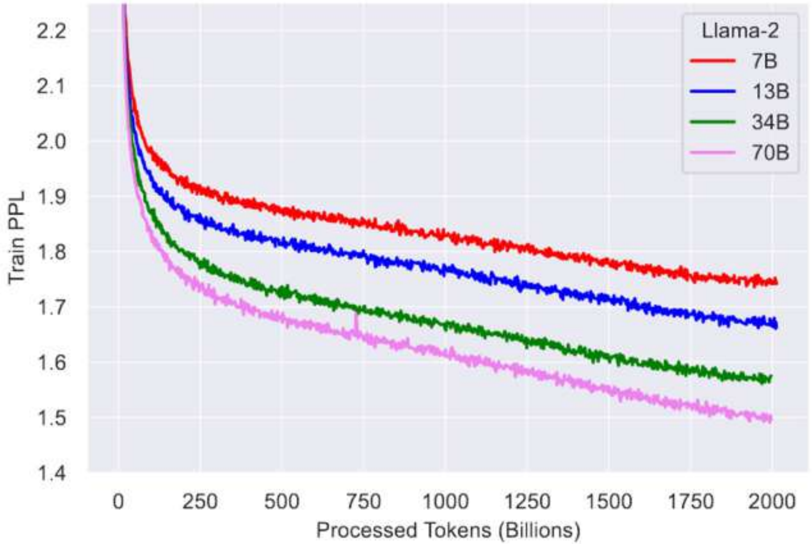
Text generation and the advent of LLMs

In order to obtain better results, the most performative models reached human-like generative performances by:

- **Scaling the models -> exploiting bigger models led to better performances, obtaining astonishing generative and Natural language understanding capabilities.**

Bigger model led though to:
- High training time consumption
- Higher inference time

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO_2eq)
Llama 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00



How can we get better results out of these models?

Since GPT2, LLMs have shown great capabilities in few-shot settings, by just prompting and describing the task to the model.

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese => ...........               ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese => ...........               ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée        ←
plush girafe => girafe peluche      ←
cheese => ...........               ← prompt
```

How can we get better results out of these models?

With GPT3*, the idea of few shot learning in LLMs has been redefined, introducing a new way of designing prompts for these models

Zero-shot and few-shot terms are overridden in machine learning, LLMs have shown learning capabilities just by exploiting the <u>context</u> (the input text, which expands itself in decoding phase). -> <u>That's why current LLMs are getting larger max context lengths!</u>

- **<u>zero shot</u> -> We initialize the model with as input the task description and the prompted data**
- **<u>few shot</u> -> We initialize the model with as input the task description, a few examples (one shot if one example is given) and the prompted data**

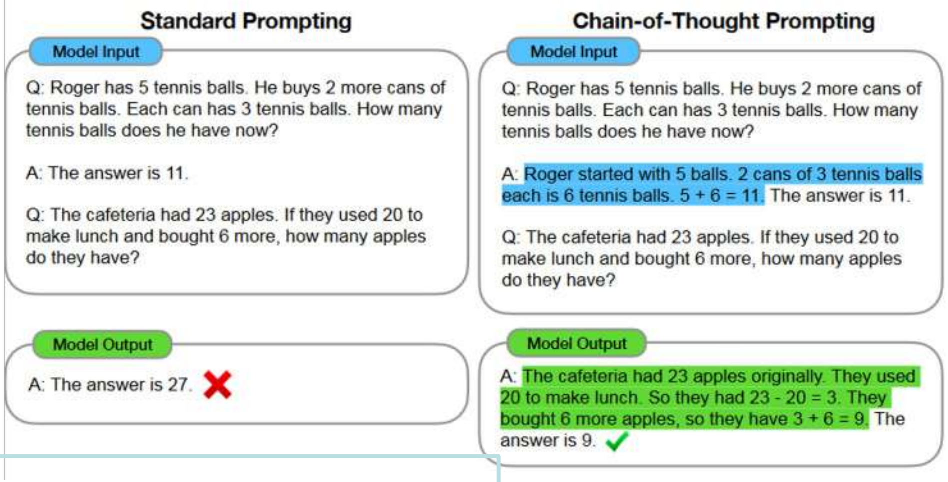
*Language Models are Few-Shot Learners - Brown et. al. 2020 arXiv 2005.14165

How can we get better results out of these models? Chain of thought*

Natural language understanding capabilities of these models increases and we have seen:

- **Interesting few-shot capabilities just by prompting some examples as input**

What if we prompt the model to mimic the human thought? Maybe a <u>step by step</u> reasoning can get better results



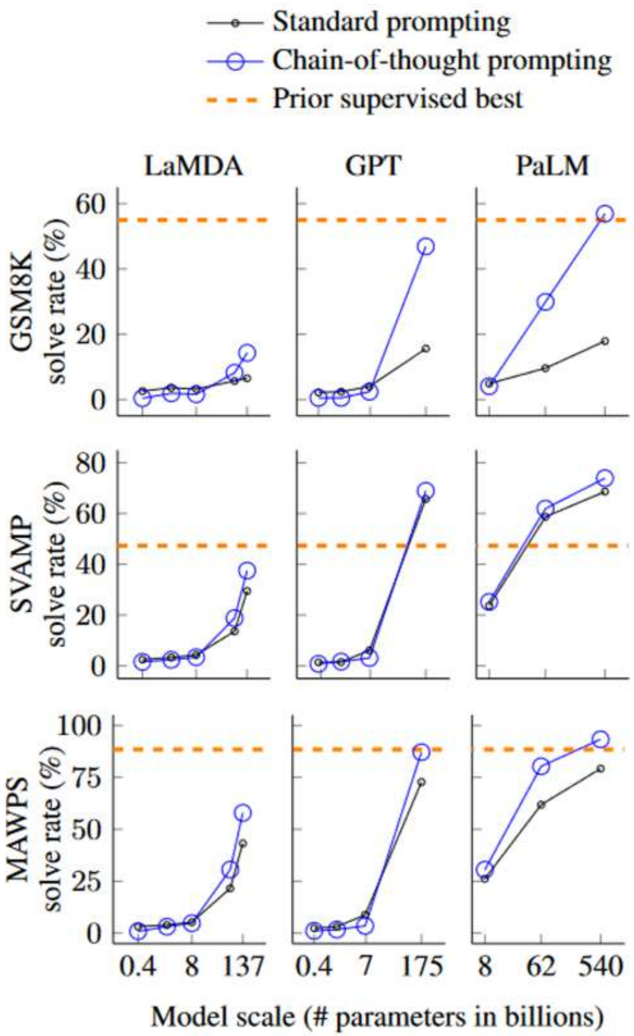
*Chain-of-Thought Prompting Elicits Reasoning in Large Language Models - Wei, Wang et. al. - Advances in Neural Information Processing Systems, 2022

How can we get better results out of these models?
Chain of thought

What did we get?

- **Chain of thought reasoning led to better results <u>without finetuning</u> (sometime even better than the stat of the art)**

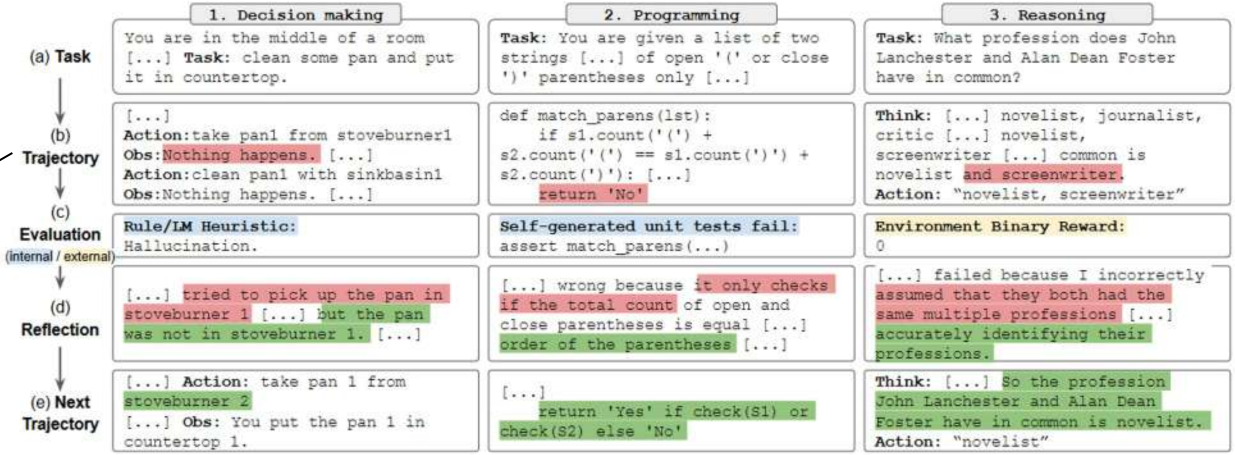
- **As we use bigger models (with greater NLU capabilities), this way of prompting gets naturally better**



How can we get better results out of these models? Reflexion*!

As LLMs have shown capabilities of learning just from the context (in-context learning), what if they can self reflect on their mistakes?

1) Given a task, the model performs an action (text generation/function calling)

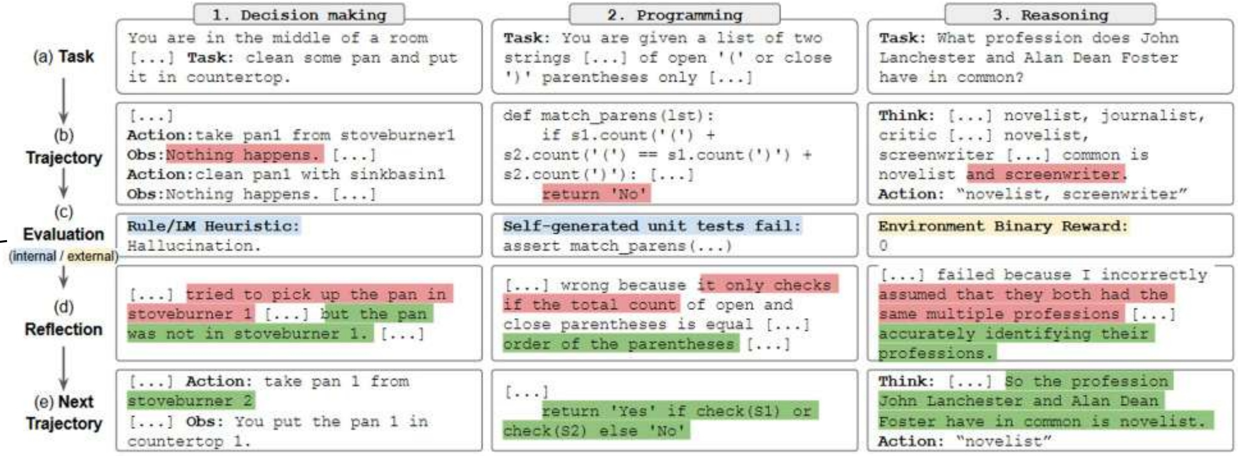


*[Reflexion: Language Agents with Verbal Reinforcement Learning](#) - Shinn,Cassano et. al. (2023)

How can we get better results out of these models? Reflexion!

As LLMs have shown capabilities of learning just from the context (in-context learning), what if they can self reflect on their mistakes?

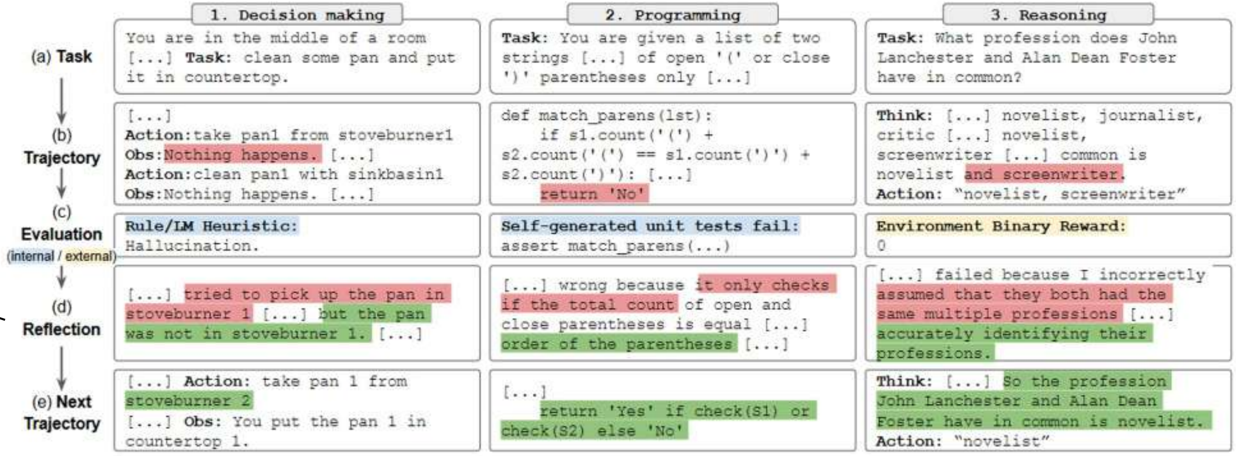
2) A second model evaluates and scores the results with specific metrics depending on the task



How can we get better results out of these models? Reflexion!

As LLMs have shown capabilities of learning just from the context (in-context learning), what if they can self reflect on their mistakes?

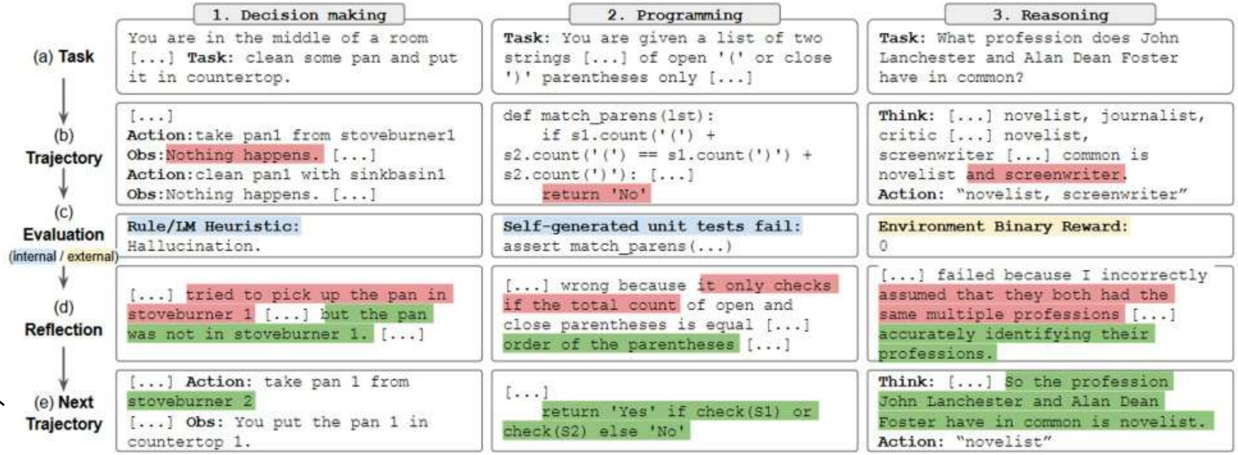
3) A third model generates a ‘verbal reinforcement learning’ signal, useful to correct the error



How can we get better results out of these models? Reflexion!

As LLMs have shown capabilities of learning just from the context (in-context learning), what if they can self reflect on their mistakes?

4) The model exploits then the old trajectory and the reflection to generate a new output



How can we get better results out of these models? Reflexion!

Problem related to verbal self reflection? The amount of in-context data is bounded by the max context length of the architecture!

The self reflection module will then loose part of the reflection and trajectory data **but… it seems effective!**

Benchmark + Language	Base	Reflexion	TP	FN	FP	TN
HumanEval (PY)	0.80	**0.91**	0.99	0.40	0.01	0.60
MBPP (PY)	**0.80**	0.77	0.84	0.59	0.16	0.41
HumanEval (RS)	0.60	**0.68**	0.87	0.37	0.13	0.63
MBPP (RS)	0.71	**0.75**	0.84	0.51	0.16	0.49

results on code generation task

Adapt generative models to different tasks!

LLMs have remarkable capabilities in different domains but:

- What if we want to adapt the model to a specific task (e.g. enterprise specific chatbot)?

As old fashioned machine learning algorithms, fine-tuning still remains the best choice to inject new knowledge into these model, keeping NL representation properties of the language used during pre-training

Adapt generative models to different tasks!
A brief look at code generation

What about code generation?

- **We start from a pre-trained LLM model (e.g. GPT3)**
- **We fine tune the model with ad hoc data designed for the domain**

In this example we want to achieve a format:

- **(function signature, docstring) -> function body**

Where the response is the one highlighted in yellow

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

Adapt generative models to different tasks!
A brief look at code generation

What about code generation?

Current LLMs trained on big multi-domain corpus showed great code generation capabilities!

GPT models as backbone with reflexion algorithm outperformed domain specific fine-tuned models!