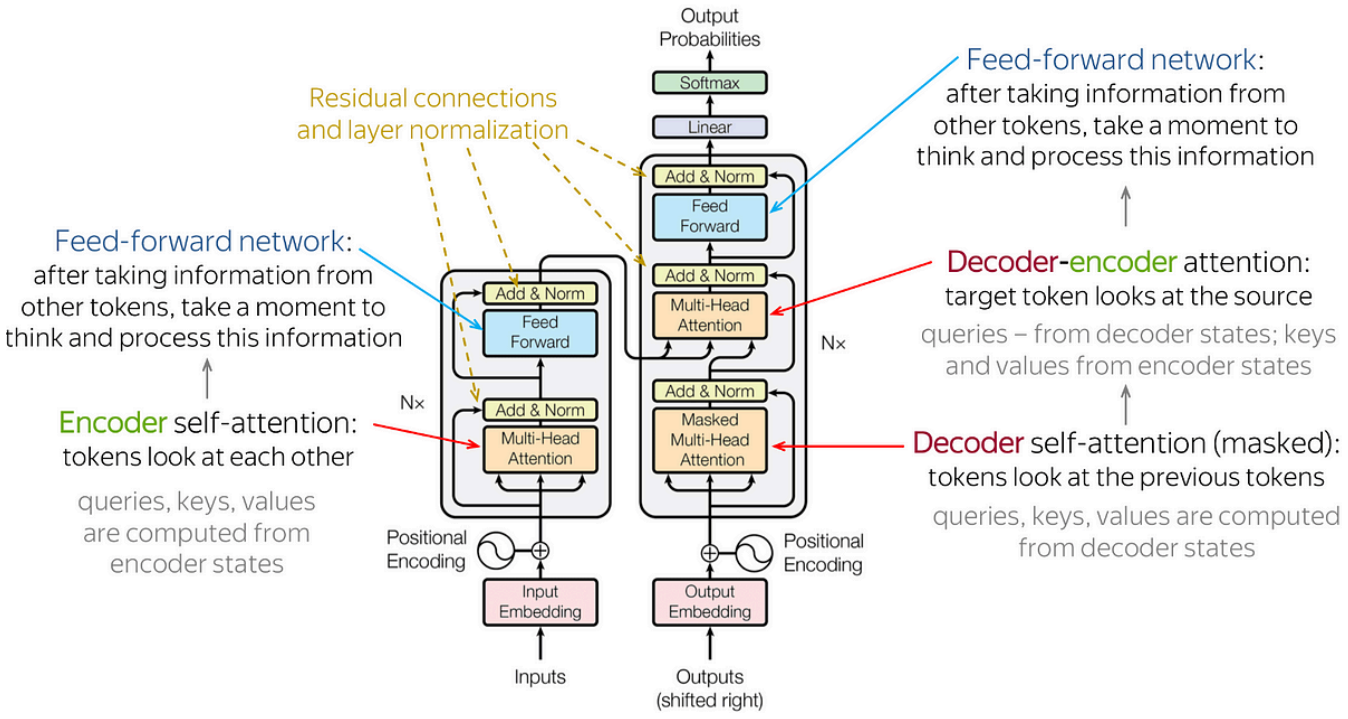
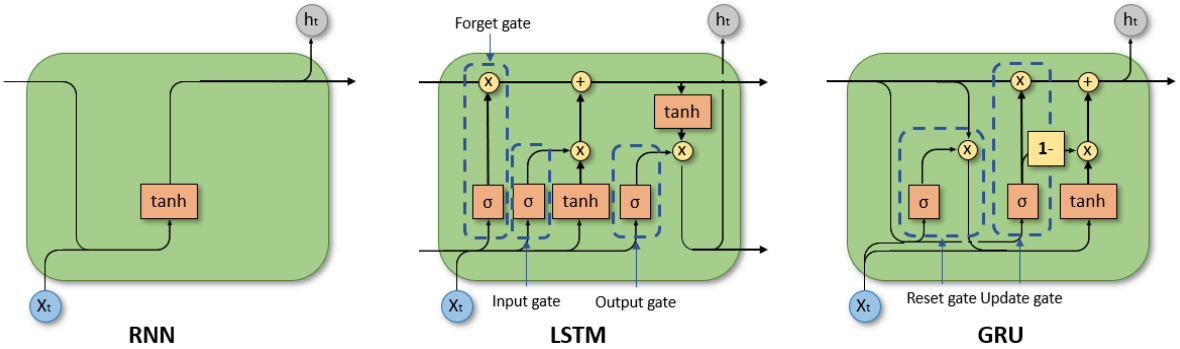
Notes by Izam Mohammed
Special thanks to Jay Alamar ❤️🥹

Transformers

The transformer was introduced in the paper Attention is all you need.pdf in 2017. It relies on a mechanism of self-attention, which allows the model to weigh the importance of different words in a sentence when processing each word.



The problem with the Traditional approach

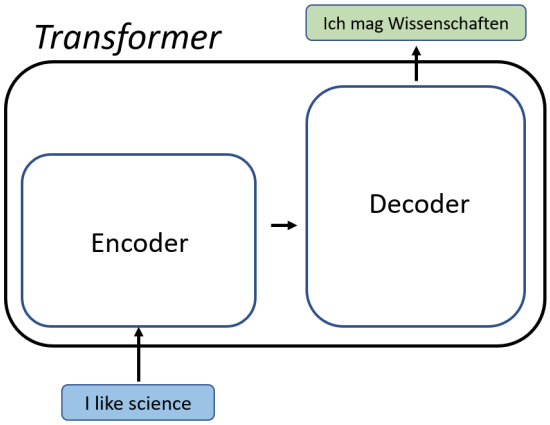


Before transformers, we were using RNN to deal with text data. But it can't capture the long-term dependency. Later LSTM and GRU came into the picture. Still, it has a lot of issues …

- ➢ *LSTM process input sequentially which makes them slower compared to transformers which have parallel processing capabilities*

- *LSTM struggles to capture the long-term dependency.*
- *It is challenging in deep LSTM and struggling with vanishing gradient.*
- *Context learning is insufficient in LSTM since it uses a hidden state*
- *Due to their sequential processing nature, LSTMs and GRUs might struggle to scale effectively to huge datasets.*

Why Transformers



- *It will solve all of the problems mentioned above that occur in the LSTM and GRU.*
- *The core is a self-attention mechanism, which enables the model to weigh the importance of different tokens in the input sequence when generating output.*
- *incorporate positional encodings to provide information about the position of tokens in the input sequence*
- *Can do transfer learning and pre-training very well.*
- *offer interpretability through attention visualization*
- *can be adapted to various tasks through modifications*

Advantages

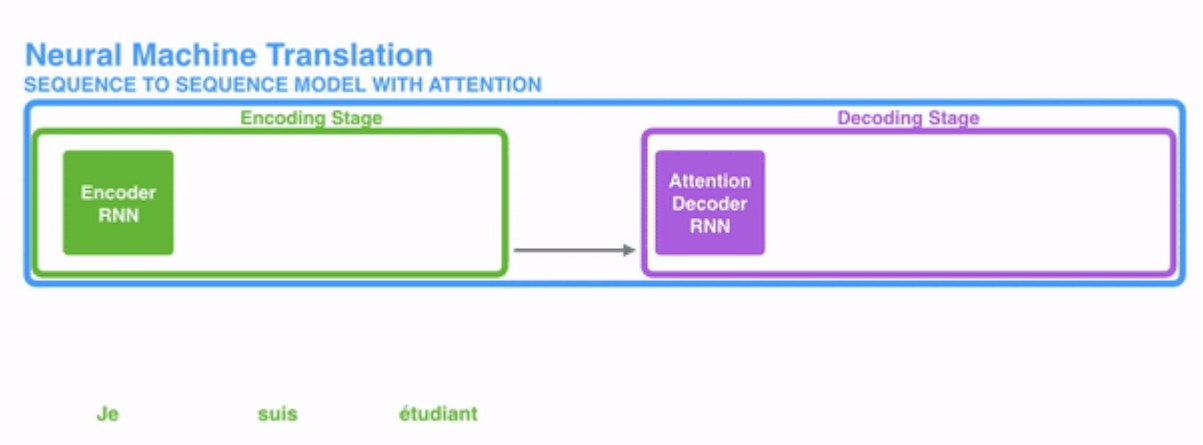
- *Parallel processing*
- *Scaling*
- *Improved contextual understanding*
- *Wide range of application*
- *Transfer learning possibilities*
- *Interpretability with visualization*

Disadvantages

- *High computational cost*
- *Vulnerability to Adversarial Attacks*
- *High memory requirements*
- *Interpretability challenges*
- *Limited Generalization to Out-of-Distribution Data*

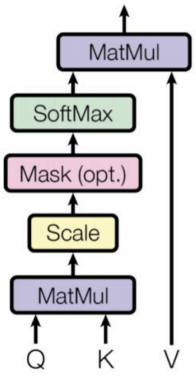
Attention in detail

It enhances deep learning models by selectively focusing on important input elements, improving prediction accuracy and computational efficiency.



Types of attention

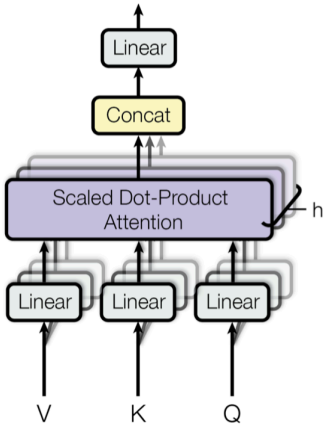
- ➢ *Self-attention (Scaled Dot product Attention)*
  - *Self-attention is a mechanism that allows a model to focus on different parts of the input sequence when processing each element. It calculates attention weights by measuring the similarity between pairs of elements in the sequence and then combines them to produce a weighted sum.*

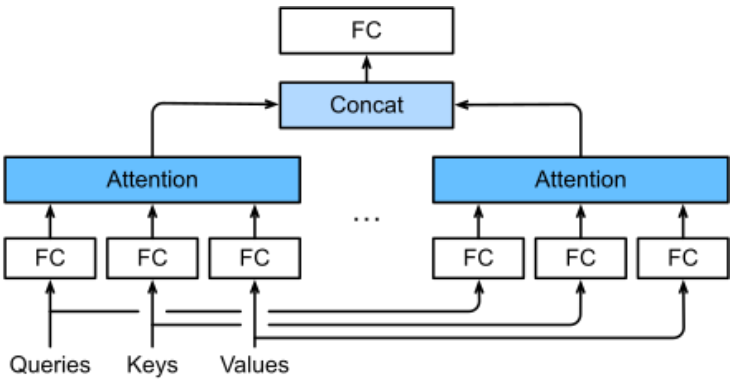


$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K and V are Query, Key and Value vectors.

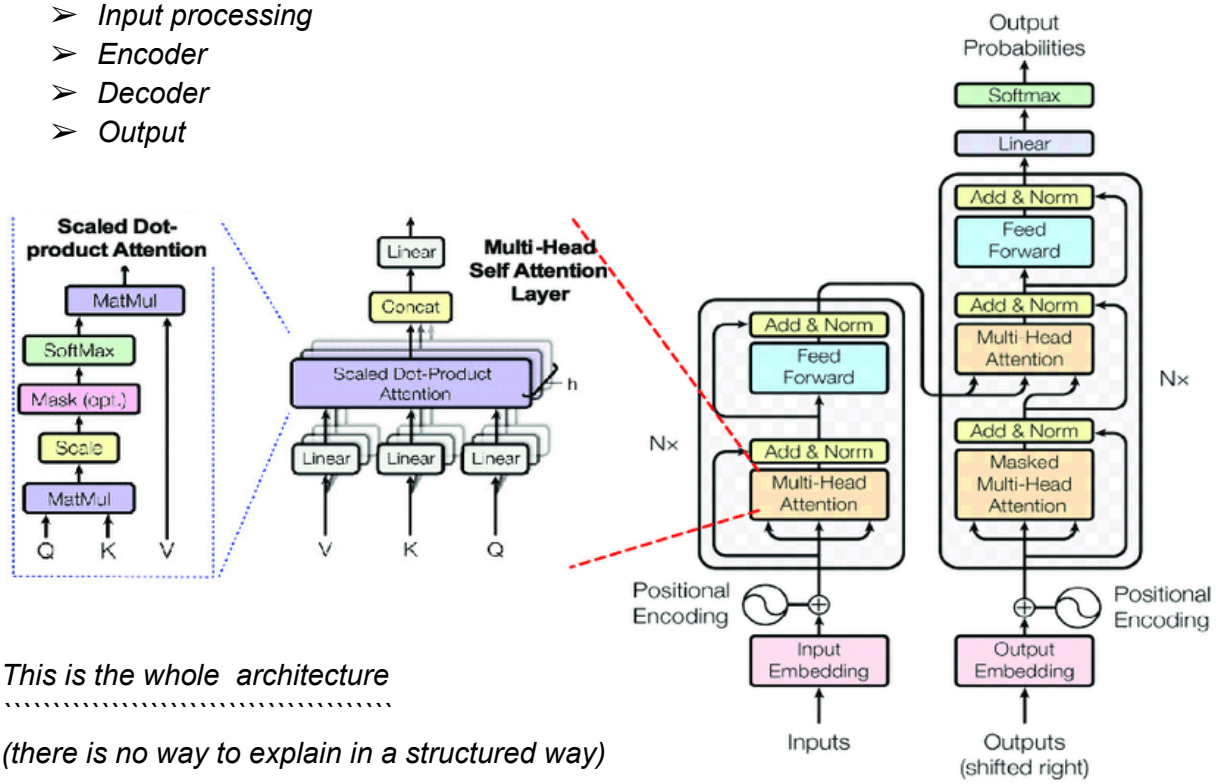
- ➢ *Multi-Head attention*
  - *Multi-head attention is a mechanism that extends self-attention by computing attention in parallel across multiple heads or sets of parameters. This allows the model to attend to different parts of the input sequence simultaneously and learn diverse representations.*





How it works

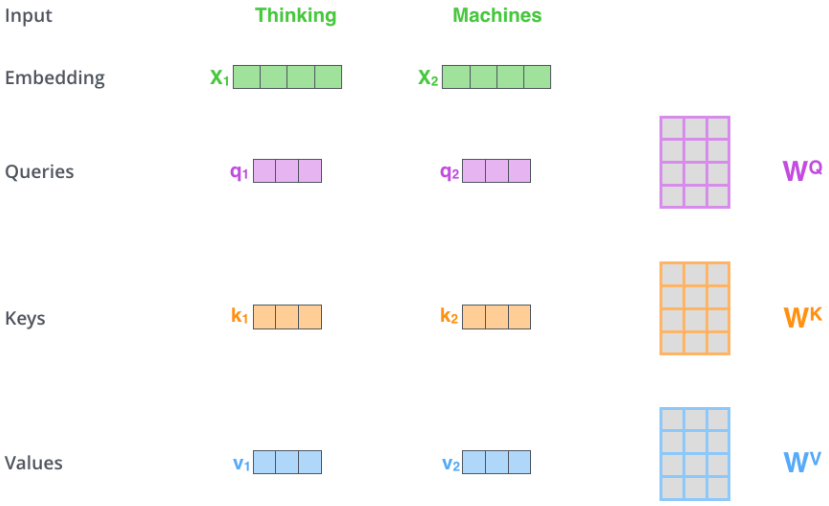
- ➢ *Input processing*
- ➢ *Encoder*
- ➢ *Decoder*
- ➢ *Output*



This is the whole architecture

(there is no way to explain in a structured way)

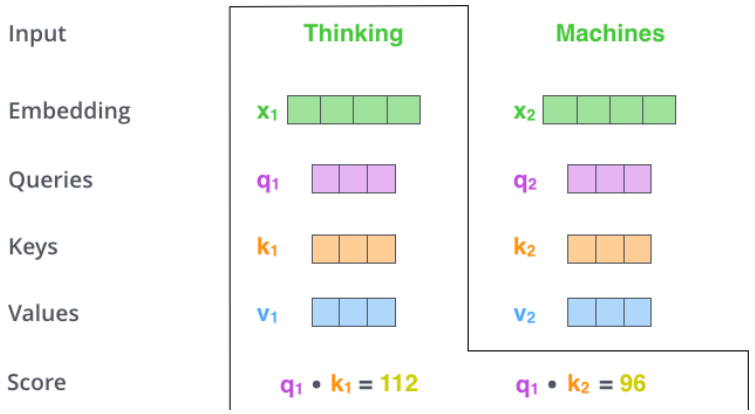
- ➢ *In a high-level view, the transformer just takes a sequence and outputs a sequence.*
- ➢ *Internally it has 6 encoders and 6 decoders (6 is a hyperparameter)*
- ➢ *The inputs are encoded using learned vectors and are in 512 dimensions (512 is mentioned in the paper). Each word will be a vector.*
- ➢ *After a positional encoding is added to the encoded text*
- ➢ *The positionally encoded vector will pass into the attention layer*
- ➢ *In the attention layer, there are 3 weights → for query(wq), key(wk), value(wv)*
- ➢ *The embedding vector will multiply with wq, wk, and wv. It like this*



Here `thinking` and `machines` are two words that give input

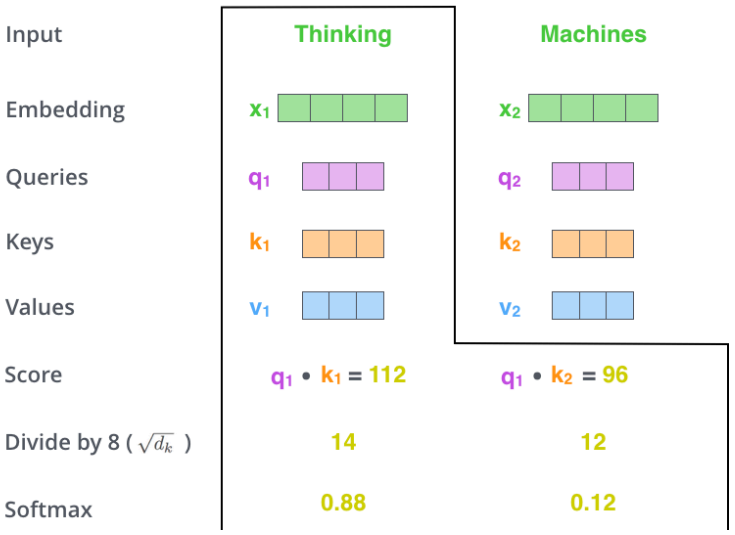
- ➢ *The query will be 64 dimensions after multiplying with the weight*

➢ *The query and key will be multiplied. It is called the score. Here, for the word* `thinking` *have 2 scores q1.k1 and q1.k2 and the other words will also have that. To multiply, we have to take the transpose of key*



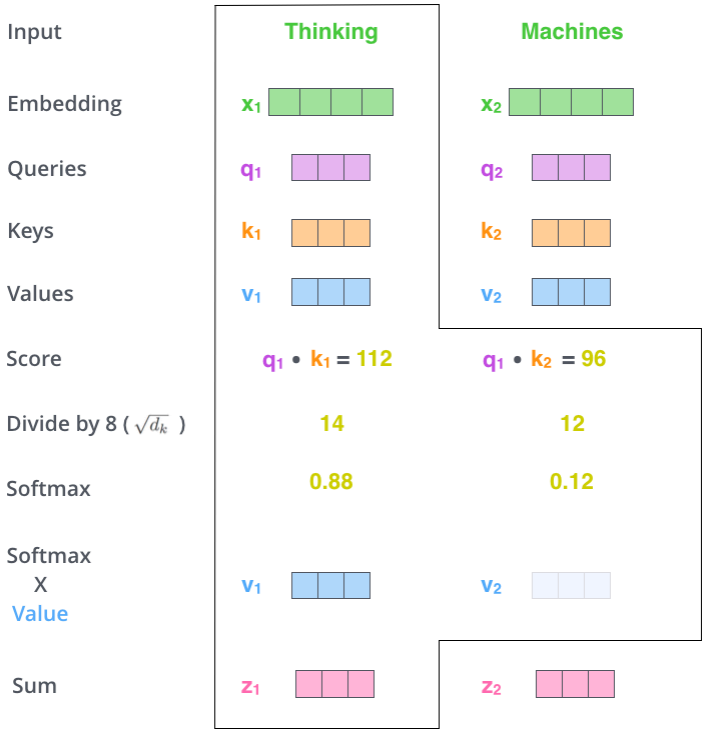
➢ *Now we are dividing the score by 8. So the 8 is root(dk). dk is the dimension of keys. The dimensions of the keys, queries, and values are 64.*

➢ *After that apply a softmax function to all of the scores of a word*

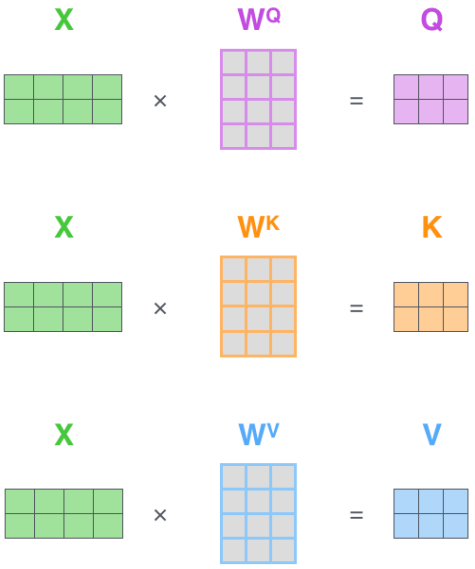


➢ *Now we will multiply the value that we calculated with softmax. It will be a vector of 64 dimensions.*

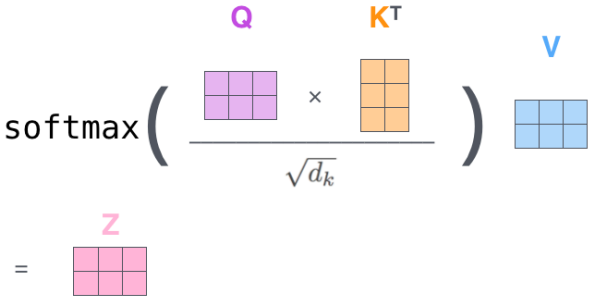
➢ *After that, sum up all of the vectors that we got.*



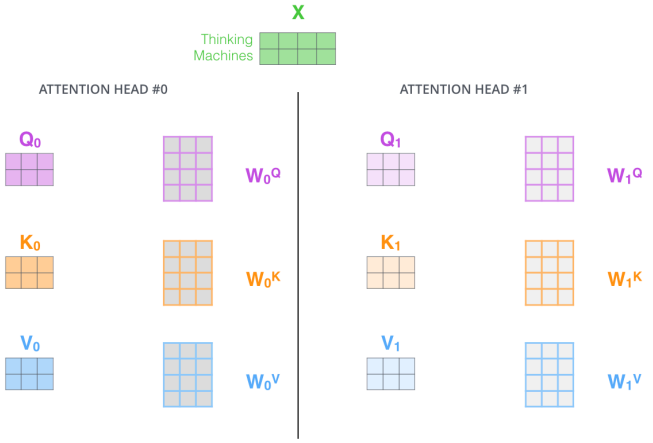
- *Finally, we got the output as z as the output of the single head attention.*
- ***Once again, I am explaining the attention***
- *First, we have to do the matrix multiplication of key, query, and value like this*



- *In the x, each row is the vector of each word. The number of rows in the x will be the number of words or tokens and the number of columns will be 512.*
- *After this step, we have to find the z by doing this*

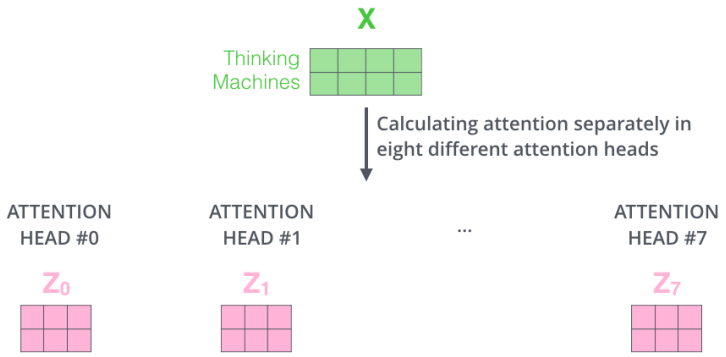


- *Now we can discuss the **multi-head attention***
- *Here all of the weights of key, query, and value are the same. So we can consider it as one head of attention. In the transformer, there will be multiple heads.*



- *There will be w1, w2 …. for each q,k and v*

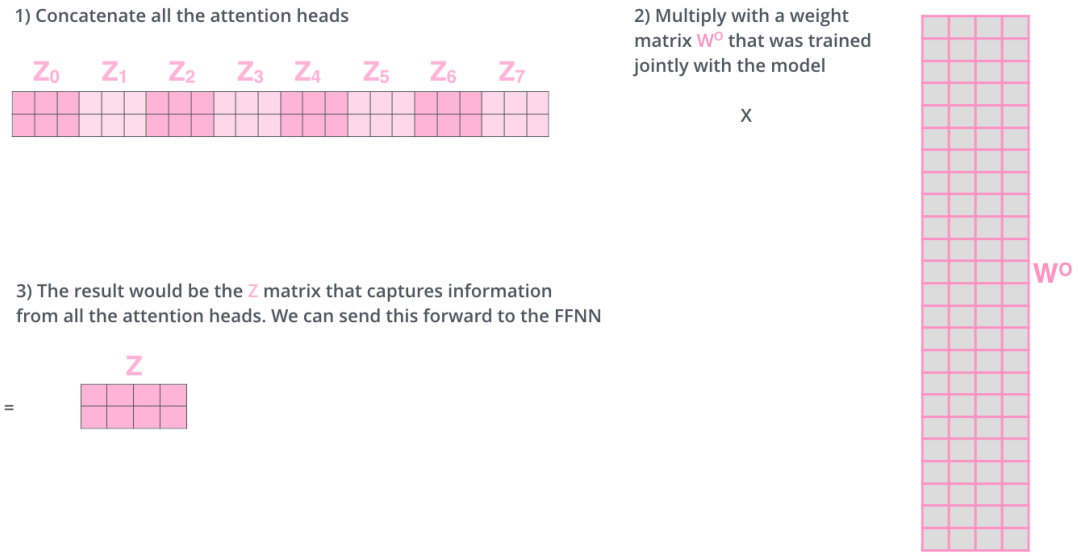
➢ *So that there will be various z-matrix*



➢ *Now we have multiple z values (in here, 8 z values are there). So we have to combine all of that*

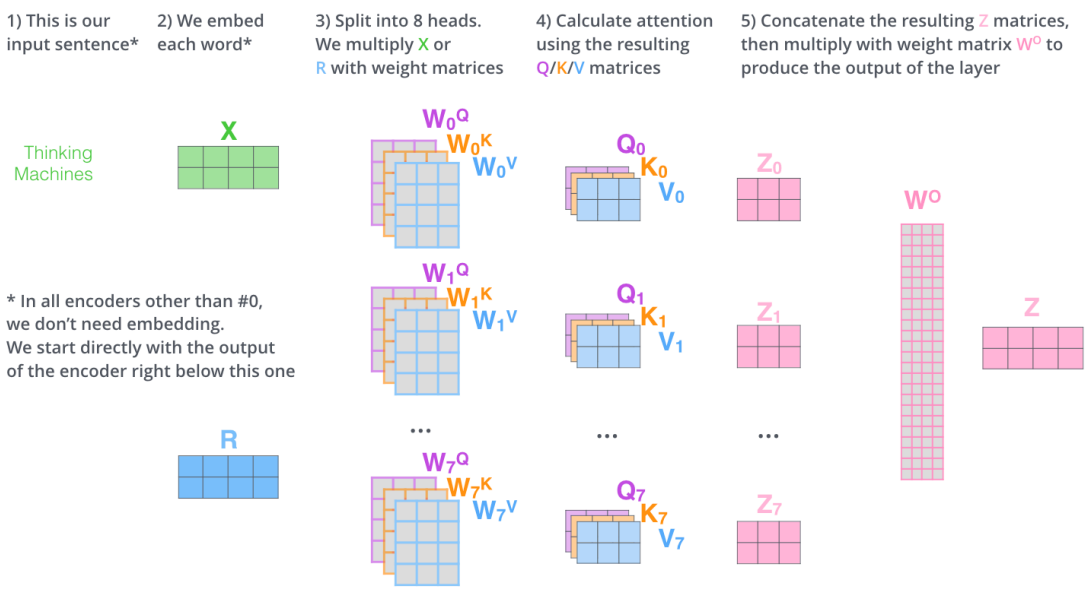
➢ *Concatenate all of the z, its number of rows is the number of words and the number of columns is 64 x the number of heads (8) = 512*

➢ *The output Z will have all of the heads or all of the z values*

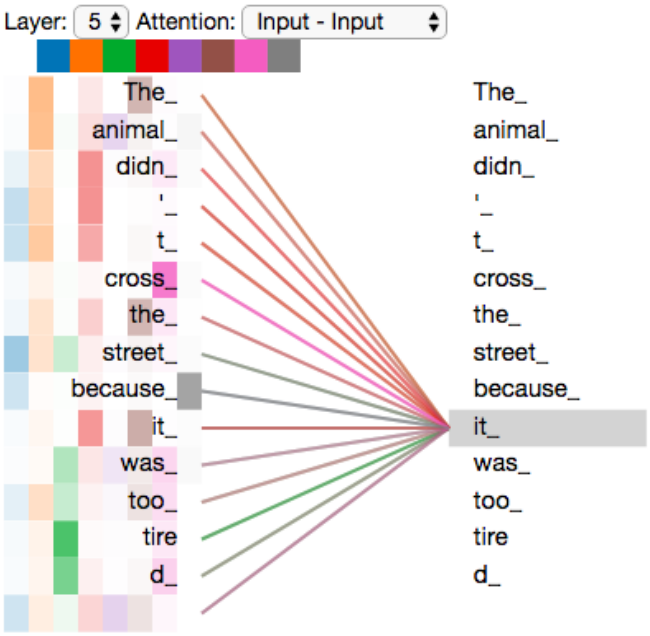


➢ *We can give this output to a feed-forward neural network*

➢ *Lemme just summarise the entire multi-head attention*



➢ *If we visualize all of the heads, it will look like this*

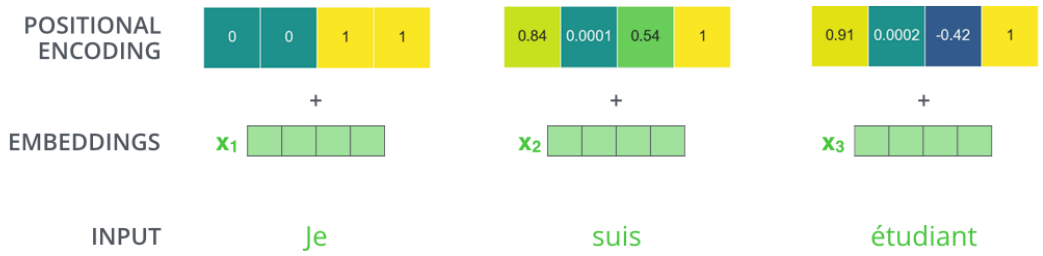


➢ *Here the word '`it_`' has multiple attentions from the 8 attention heads*

➢ *So far we only discussed the **Encoder**, we missed one portion before that which is the positional encoding*

➢ *Positional encoding means we will have some values for each embedding based on its position. It is like this*

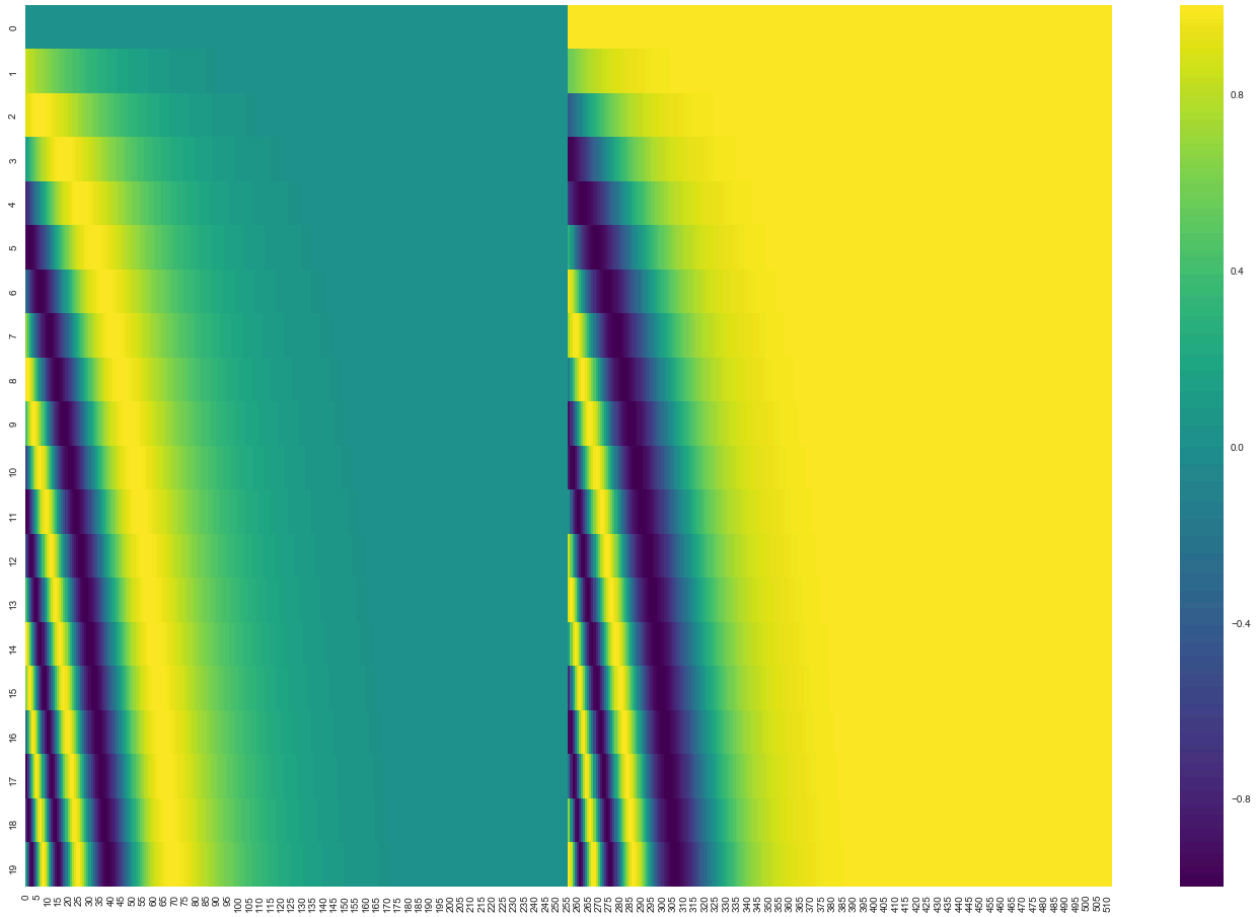
➢ *If the embedding is in 4 dimensions,*



➢ *By seeing this image, you may think what is the value of the positional encoding, I will show you another image that has the pattern of the positional encoding*

➢ *A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.*

**the picture is right below →*



➢ *There is a way to calculate these positional values*

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

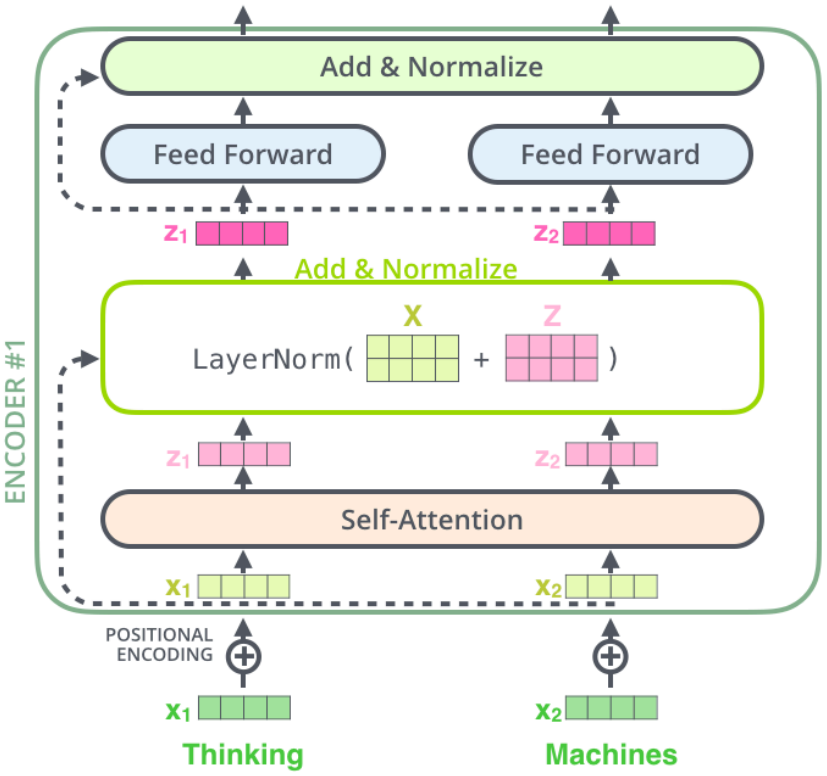
Fig. Formula to calculate positional encoding (Source: Author)

In the above formula,

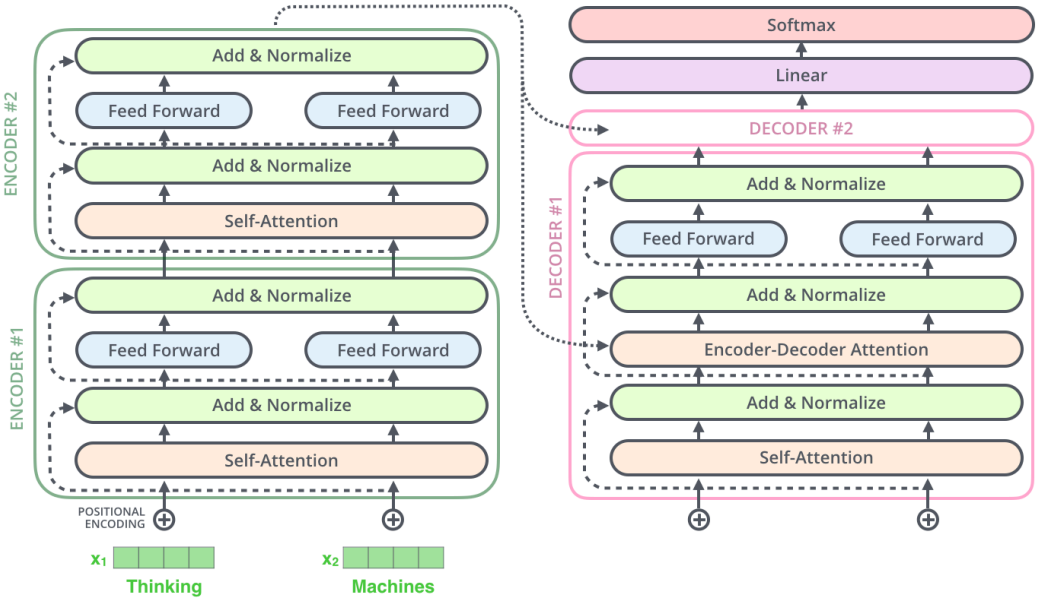
- **pos** = position of the word in the sentence
- **d** = dimension of the word/token embedding
- **i** = represents each dimension in the embedding

➢ *There another important thing about Encoder is that ….*

➢ *After the self-attention layer and before the feed-forward network There is a skip connection (residual connection) and a layer normalization*



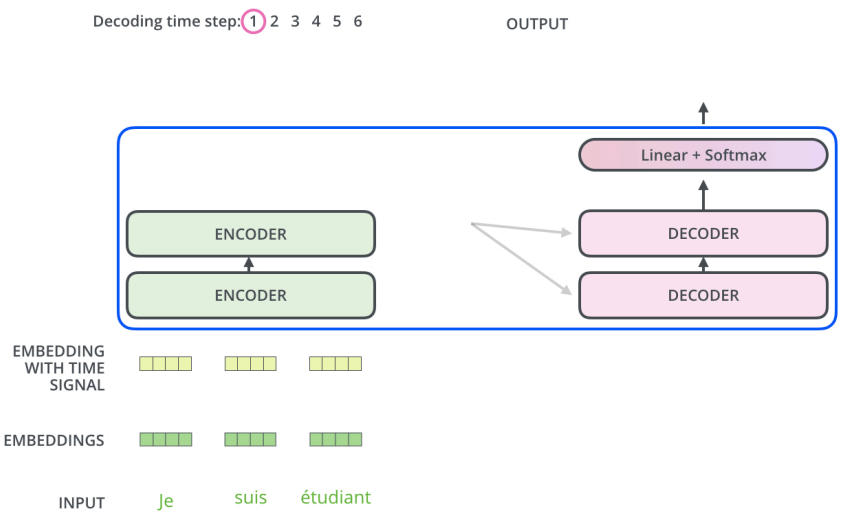
➢ *Now we can go to the **Decoder** …*



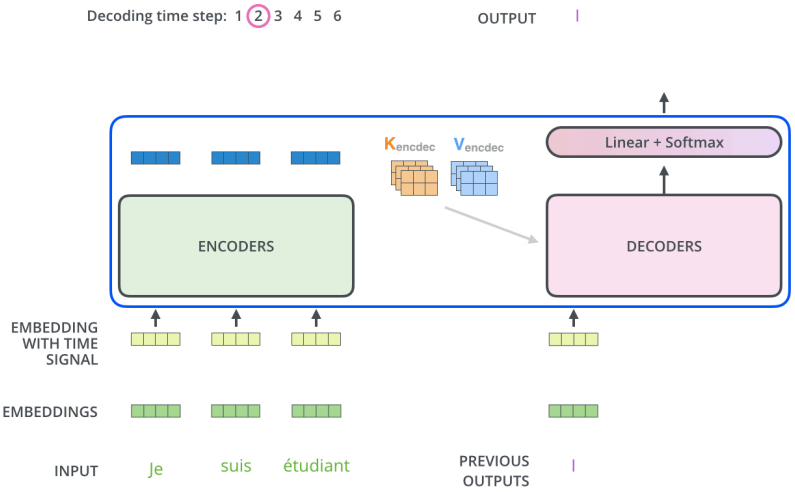
➢ *The output of the top encoder is transformed into a set of attention vectors K and V.*

➢ *The first step will be like this*

**scroll buddy 🙂 →*

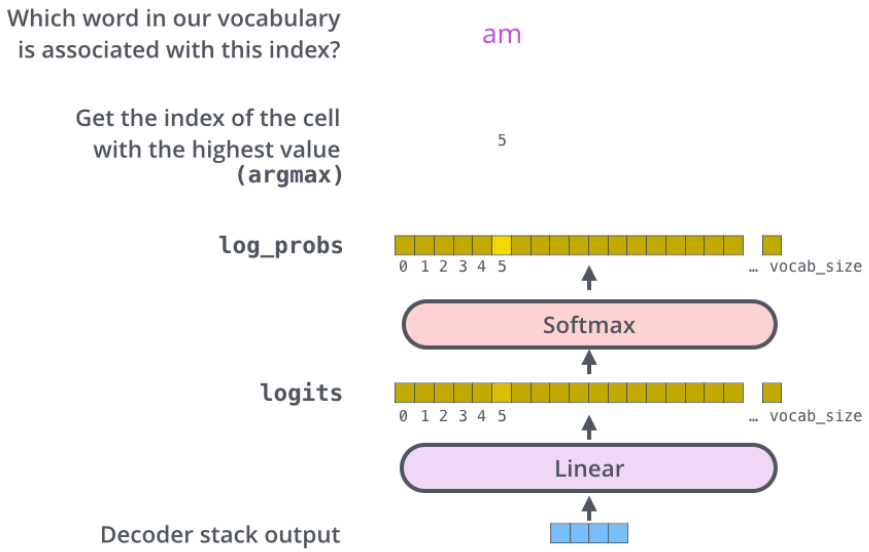


➢ *The decoder will generate the sequence until it gets to the end of the sentence.*



➢ *In the decoder, instead of multi-head attention, we are using masked multi-head attention. It is slightly different than the other one.*
➢ *It incorporates a mask into the attention calculation, which prevents tokens from attending to future tokens.*
➢ *It is done by applying a triangular mask to the attention score matrix, setting all of the entries above diagonal to negative infinity.*
➢ *That's all about the decoder. Now Let's talk about the final **Linear and Softmax** Layer*
➢ *The linear layer is a simple fully connected neural network that projects the vector produced by the decoder into a logit vector.*
➢ *Assuming that the model knows 10,000 words, then it would make a logit of size 10k with 10k cells wide.*
➢ *The softmax layer turns those values into probabilities.*
➢ *The cell which has the highest will be chosen …*

**hehe, there is not enough space here baby 🤗. Scroll plssss*



- ➢ *So that is all about the architecture of the Transformers* 🩵🥹

Terms

- ➢ *Teacher forcing*
  - *In teacher forcing, during training, the actual previous output from the transformer's decoder is used as the input for the next step, instead of the predicted output from the previous step.*
- ➢ *Label smoothing*
  - *Label smoothing introduces a small amount of noise to the training labels, making them less one-hot encoded and more probabilistic.*
  - *This encourages the model to learn more generalizable representations and avoid overfitting the specific training data.*
- ➢ *LLMs*
  - *LLMs are large, powerful neural networks trained on massive amounts of text data.*
  - *They can generate text, translate languages, write different kinds of creative content, and answer your questions in an informative way.*
- ➢ *Pre-training*
  - *Pre-training involves training a transformer on a large, general-purpose text corpus before fine-tuning it for a specific task.*
- ➢ *Finetuning*
  - *Fine-tuning takes a pre-trained transformer and adapts it to a specific task by training it on task-specific data.*
  - *This leverages the general language knowledge from pre-training while specializing the model for the target task, often resulting in significant performance improvements.*
- ➢ *RAG model*
  - *Retrieval-augmented generation combines retrieval and generation techniques.*

    - *During inference, they first retrieve relevant passages from a knowledge base using a retriever, then feed those passages and the input prompt to a decoder to generate the final response.*
- *Perplexity*
    - *Perplexity is a measure of how well a language model predicts the next word in a sequence.*

Evaluation

- *BLEU → for translations*
- *Accuracy → in the case of classification*
- *Perplexity → next word prediction*
- *Human Evaluation → for all kinda problems*
- *Attention Visualization*

Implementation in PyTorch

steps

- *Embedding*
- *Positional Embedding*
- *Multihead Attention*
- *Transformer Block*
- *Encoder block*
- *Decoder Block*
- *Decoder transformer*
- *Transformer*

Notebook → Tranformers_from_scratch.ipynb

Libraries

- *Pytorch*
- *Tensorflow*
- *Transformers*
- *OpenLLM*
- *Langchain*

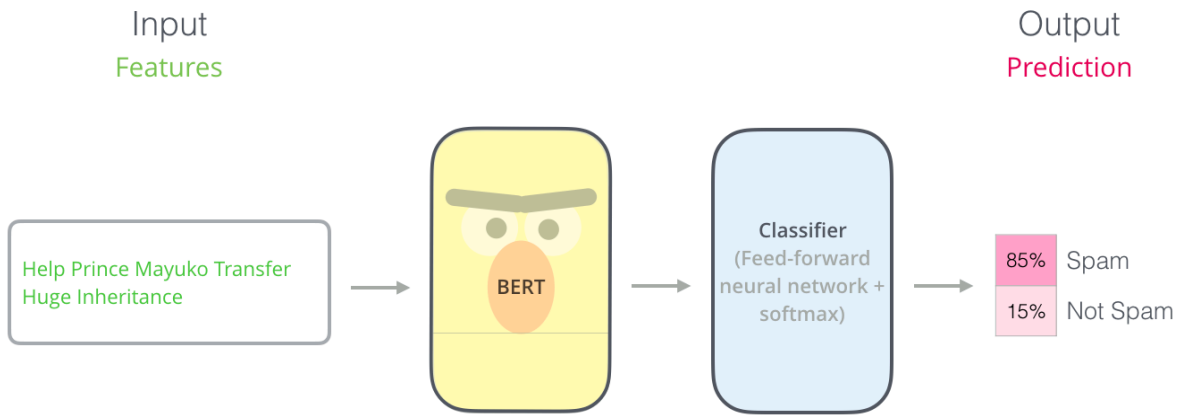
BERT

Referring to the research paper [BERT.pdf]

Bidirectional Encoder Representations from Transformers (BERT) is built on the Transformer architecture, a neural network design introduced in 2017 that utilizes "self-attention" mechanisms. It is an encoder-only model.

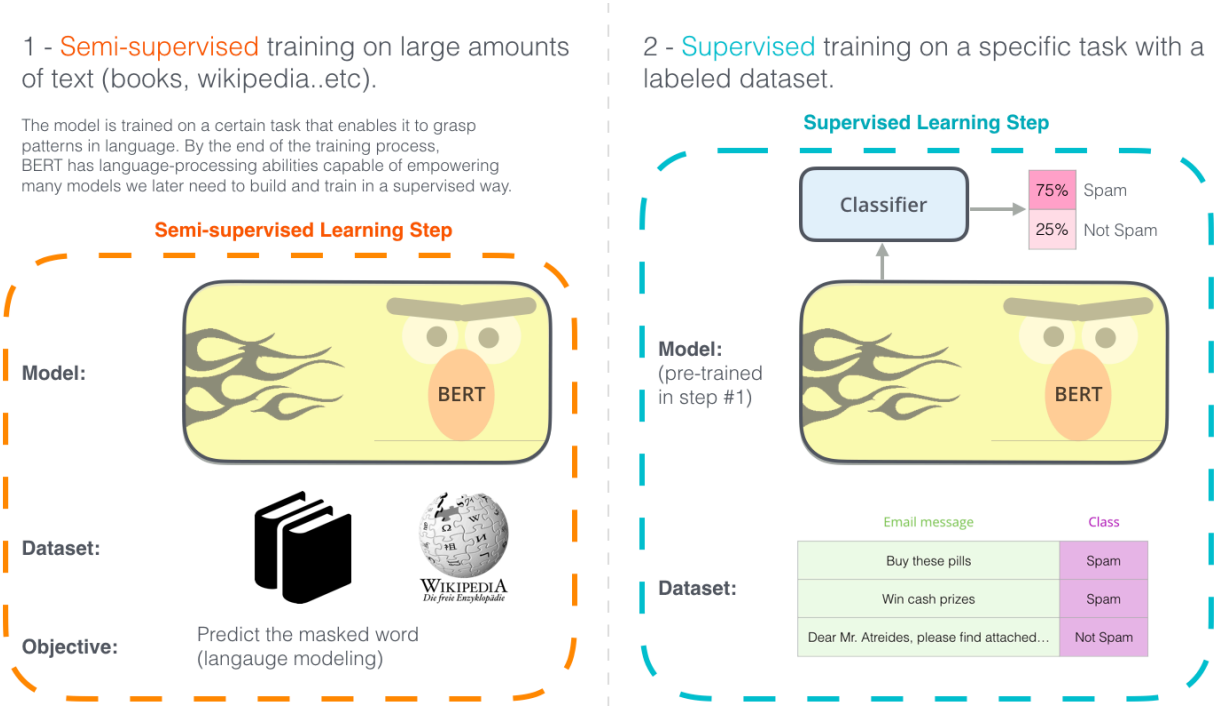
The sequence of tokens is fed to the Transformer encoder. These tokens are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors, each corresponding to an input token, providing contextualized representations.

Eg:-



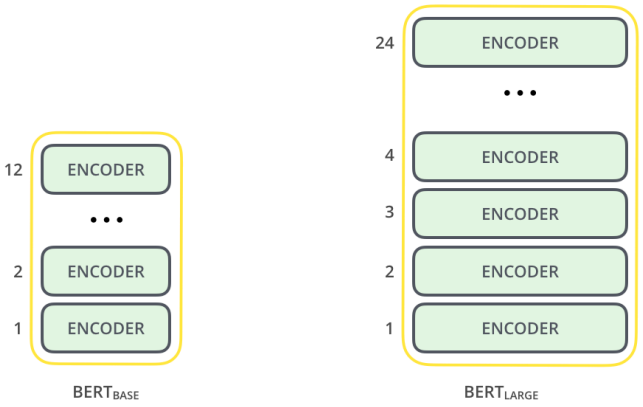
The BERT model undergoes a two-step process:

- ➢ *Pre-training on Large amounts of unlabeled text to learn contextual embeddings.*
- ➢ *Fine-tuning labeled data for specific NLP tasks.*



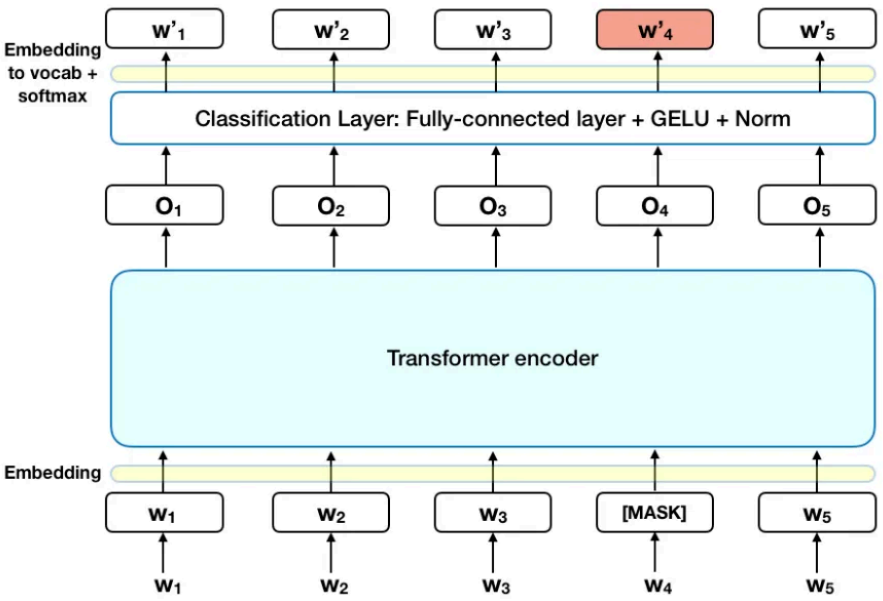
The paper presents 2 model sizes for the BERT model

- ➢ *BERT BASE – Comparable in size to the OpenAI Transformer to compare the performance*
  - *Have 12 encoder layers*
  - *Have 768 unit large feedforward neural network*
  - *Have 12 attention heads*
  - *110M params*
- ➢ *BERT LARGE – A ridiculously huge model that achieved the state-of-the-art results reported in the paper*
  - *Have 24 encoder layer*
  - *Have 1024 hidden units of FNN*
  - *Have 16 attention heads*
  - *340M params*



When it comes to the training of the model, defining the prediction goal is the challenging part. BERT addresses this challenge with two innovative training strategies:

- ➢ *Masked Language Model (MLM)*
  - *Randomly mask out 15% of the words in the input, replacing them with [MASK] token, run the entire sequence through the BERT model, and then predict only the masked words.*
  - *However, there is a problem with this naive masking approach — the model only tries to predict when the [MASK] token is present in the input. At the same time, we want the model to try to predict the correct tokens regardless of what token is present in the input. To deal with this issue, out of the 15% of the tokens selected for masking:*
    - *80% of the tokens are replaced with the token [MASK].*
    - *10% of the time tokens are replaced with a random token.*
    - *10% of the time tokens are left unchanged.*



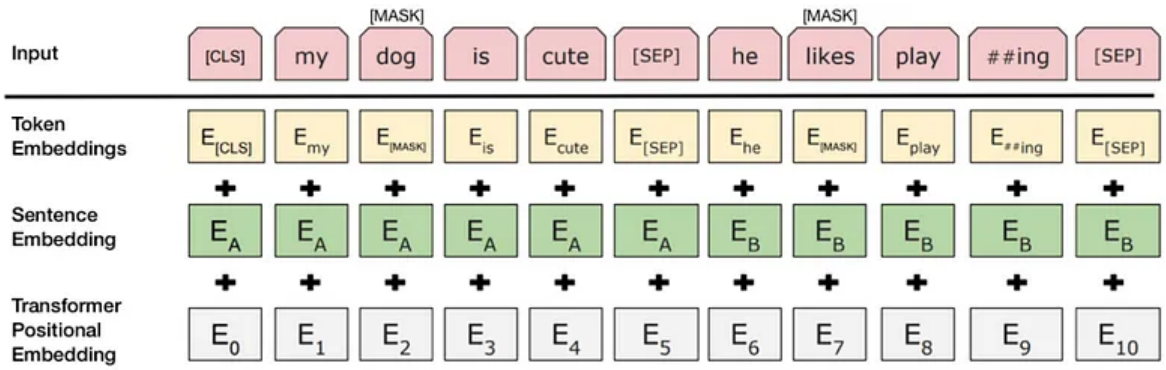
➢ *Next sentence prediction (NSP)*

- *The BERT training process also uses next-sentence prediction.*
- *As we have seen earlier, BERT separates sentences with a special [SEP] token. During training the model is fed with two input sentences at a time such that*
  - *50% of the time the second sentence comes after the first one.*
  - *50% of the time it is a a random sentence from the full corpus.*

$$\text{Input} = \texttt{[CLS] the man went to [MASK] store [SEP]}$$
$$\texttt{he bought a gallon [MASK] milk [SEP]}$$
$$\text{Label} = \texttt{IsNext}$$

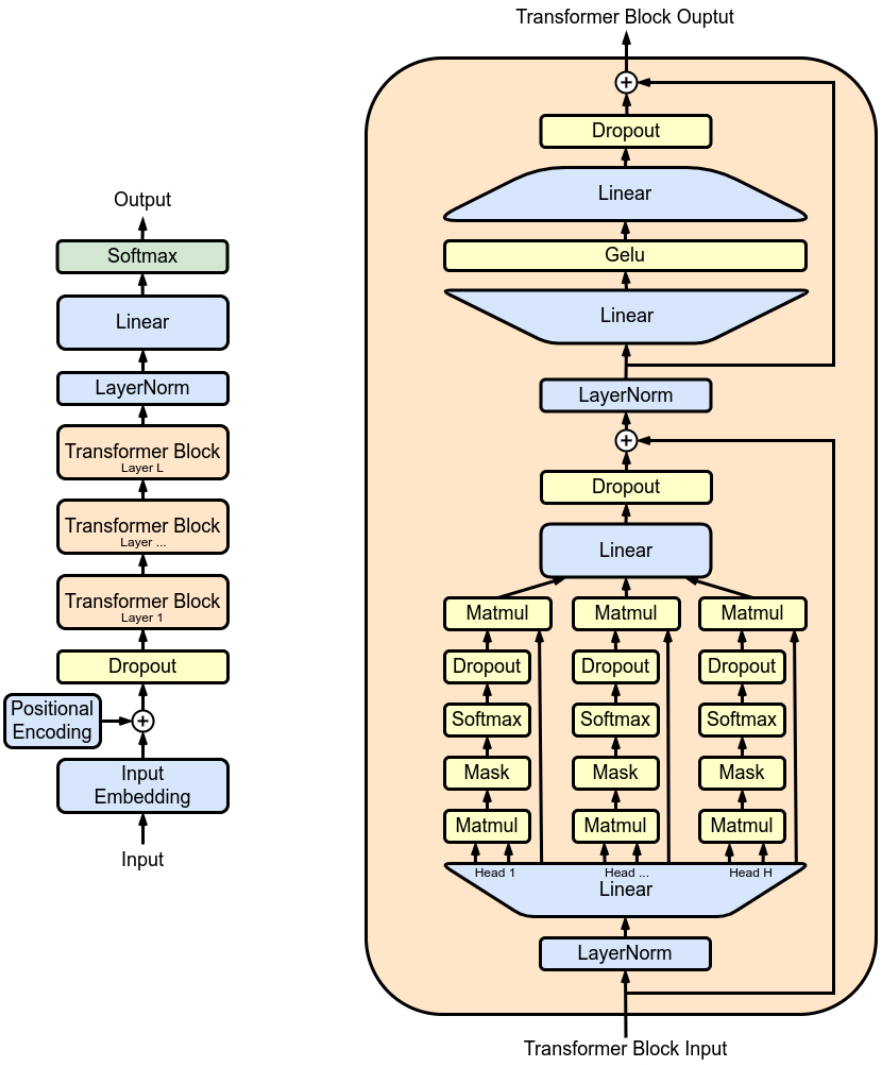
$$\text{Input} = \texttt{[CLS] the man [MASK] to the store [SEP]}$$
$$\texttt{penguin [MASK] are flight ##less birds [SEP]}$$
$$\text{Label} = \texttt{NotNext}$$

- *To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:*
  - *A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.*
  - *A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.*
  - *A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.*



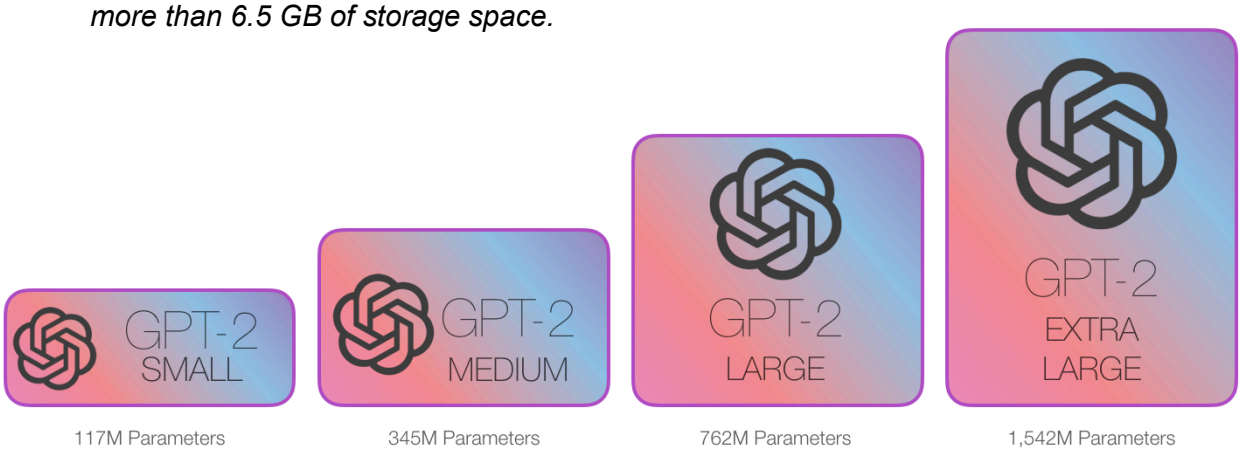
GPT

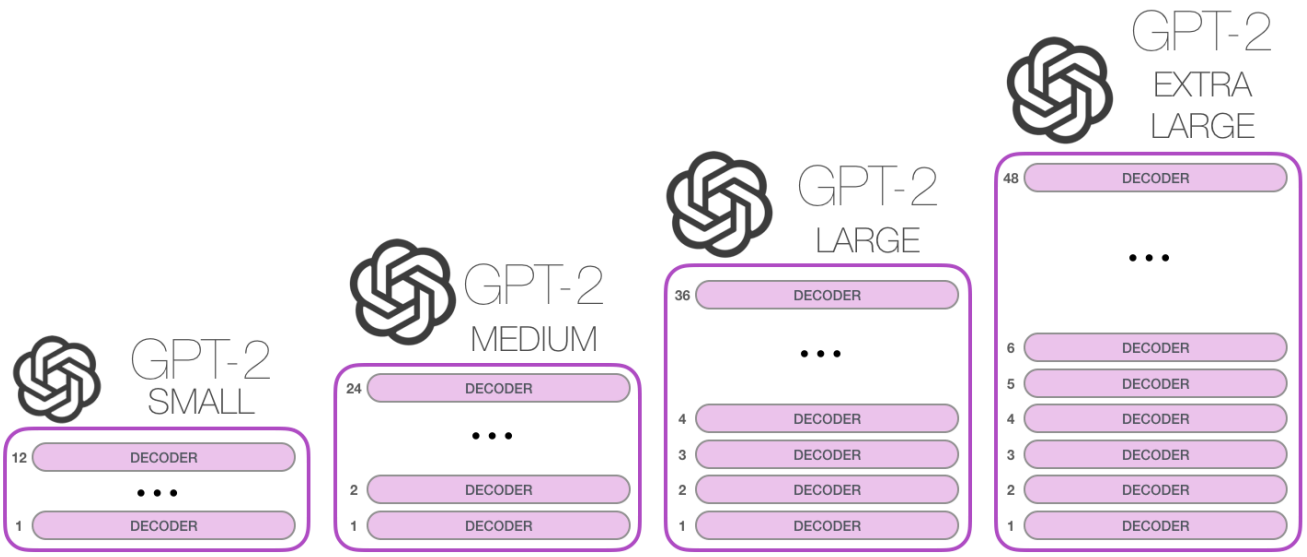
GPT uses an unmodified Transformer decoder, except that it lacks the encoder attention part. We can see this visually in the above diagrams. The GPT, GPT2, and GPT 3 are built using transformer decoder blocks.



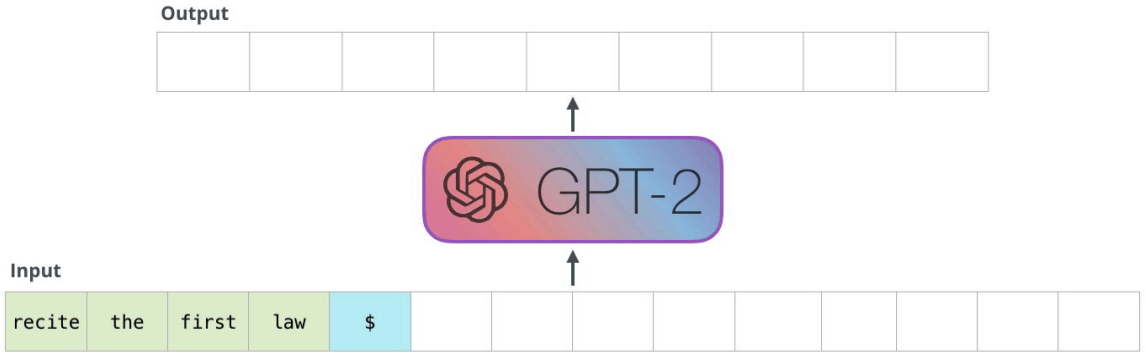
Here I would like to discuss the GPT-2. Referring to the paper [language-models.pdf]

- *The GPT-2 was trained on a massive 40GB dataset called WebText that the OpenAI researchers crawled from the internet as part of the research effort.*
- *The smallest variant of the trained GPT-2, takes up 500MBs of storage to store all of its parameters. The largest GPT-2 variant is 13 times the size so it could take up more than 6.5 GB of storage space.*

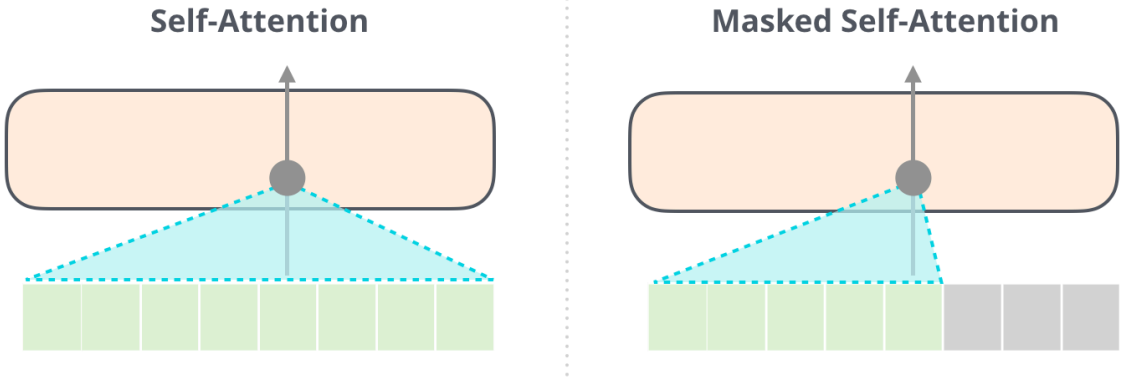




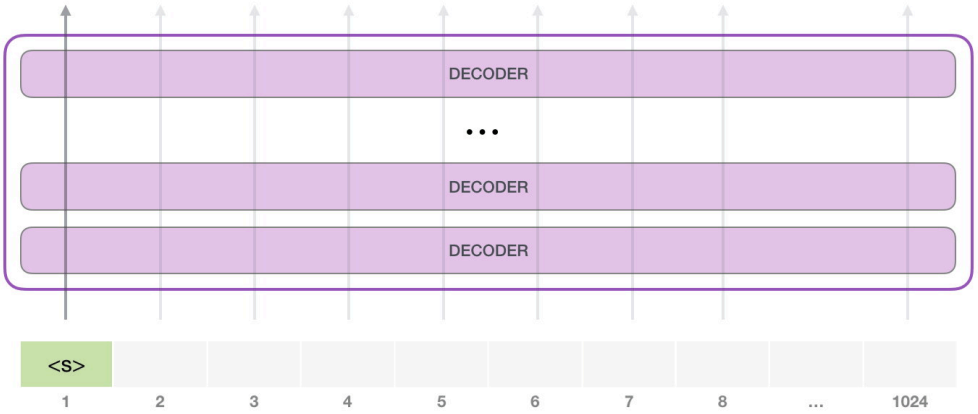
➢ *Like the traditional language models, GPT outputs one token at a time. So the new input to the model is its next step. This idea is called 'Auto-regression'*



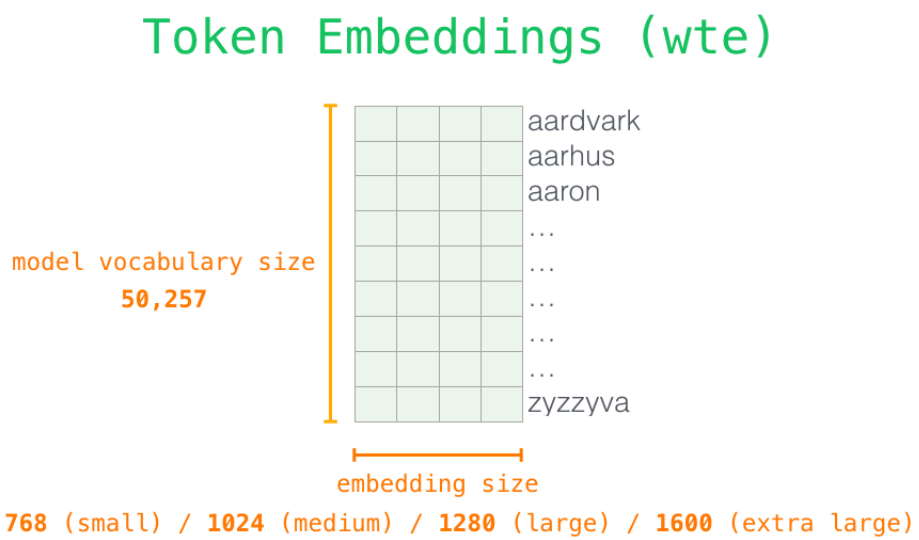
➢ *Because it is a decoder-only model, instead of self-attention, GPT uses masked self-attention. A normal self-attention block allows a position to peak at tokens to its right. Masked self-attention prevents that from happening.*



➢ *When running the GPT-2 model, the model only has input tokens. So the path will be the only active one.*



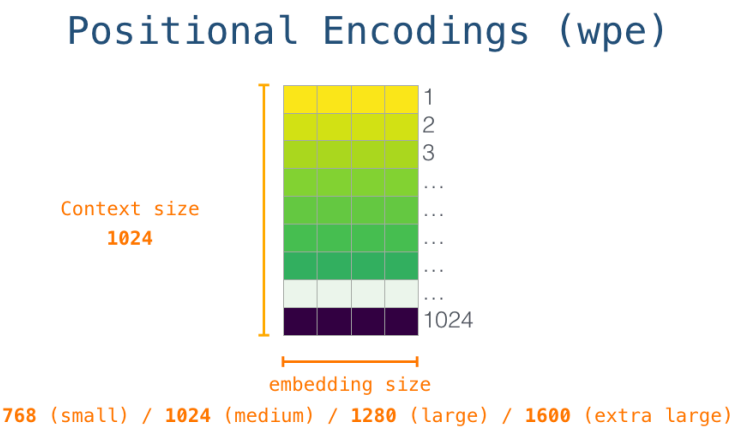
➢ *Let's look in a more detailed way, what a coconut is happening inside that. What is the embedding by the way?*



➢ *Each row is a word embedding: a list of numbers representing a word and capturing some of its meaning. The size of that list is different in different GPT2 model sizes. The smallest model uses an embedding size of 768 per word/token.*

➢ *So at first, we look up embedding with the starting token of* `<s>` *in the embedding matrix*

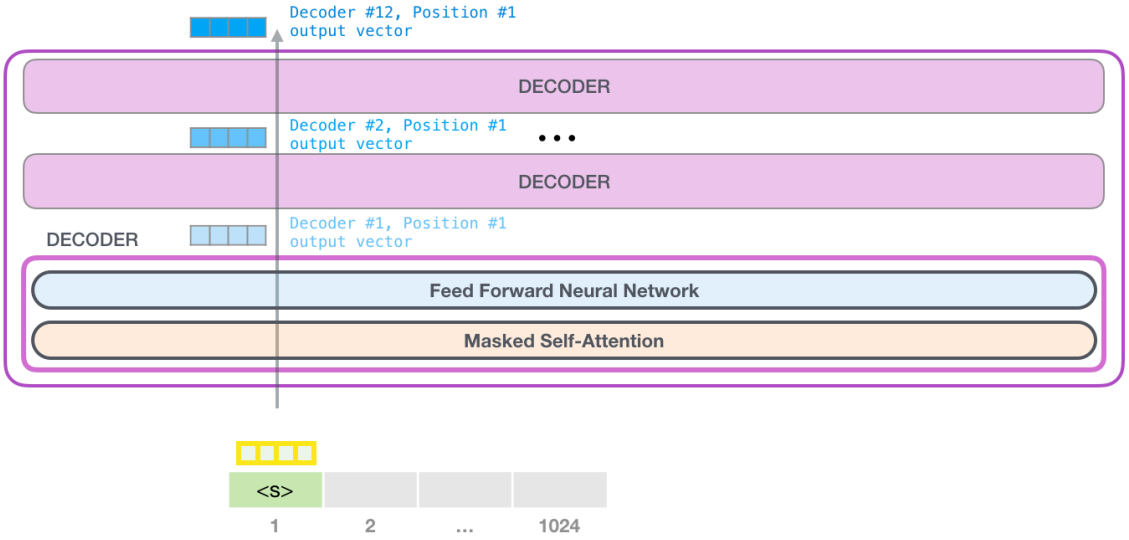
➢ *Before giving the embedding to the decoder block, it will be multiplied with a positional encoding, a positional encoding is like this*



➢ *With this, we've covered how input words are processed before being handed to the first transformer block*

➢ *Now we can discuss how it goes through the entire model. It will feel like too much. But trust me, I will get you there*

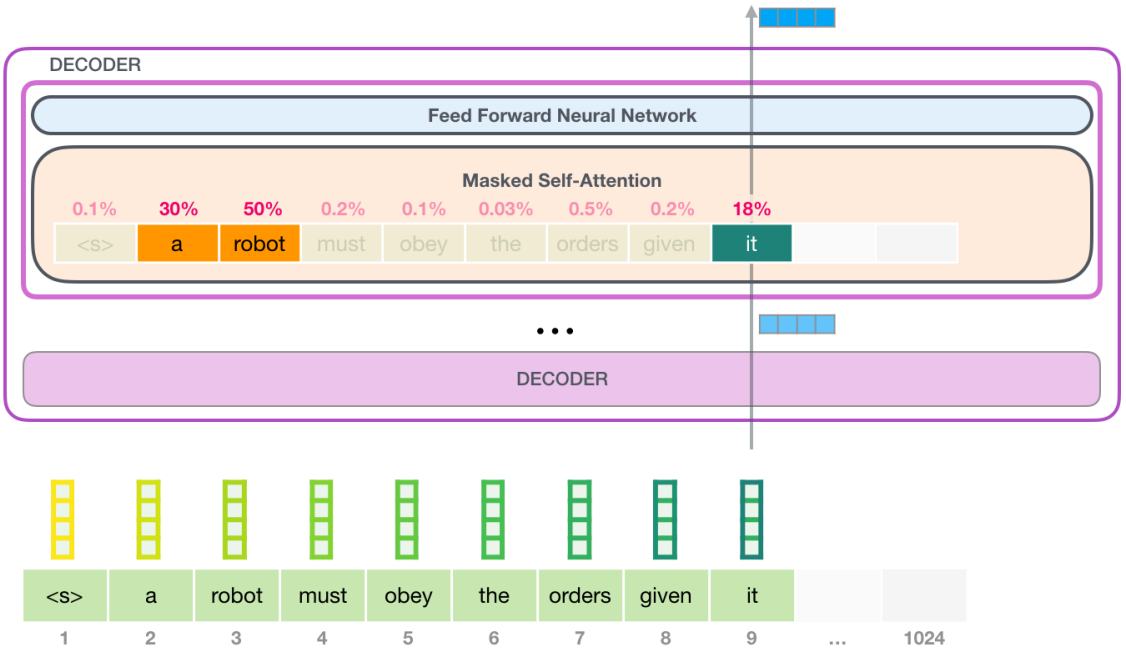
➢ *The first block can now process the token by first passing it through the self-attention process, and then passing it through its network layer. The process is identical for each of the blocks but has different weights and attention.*



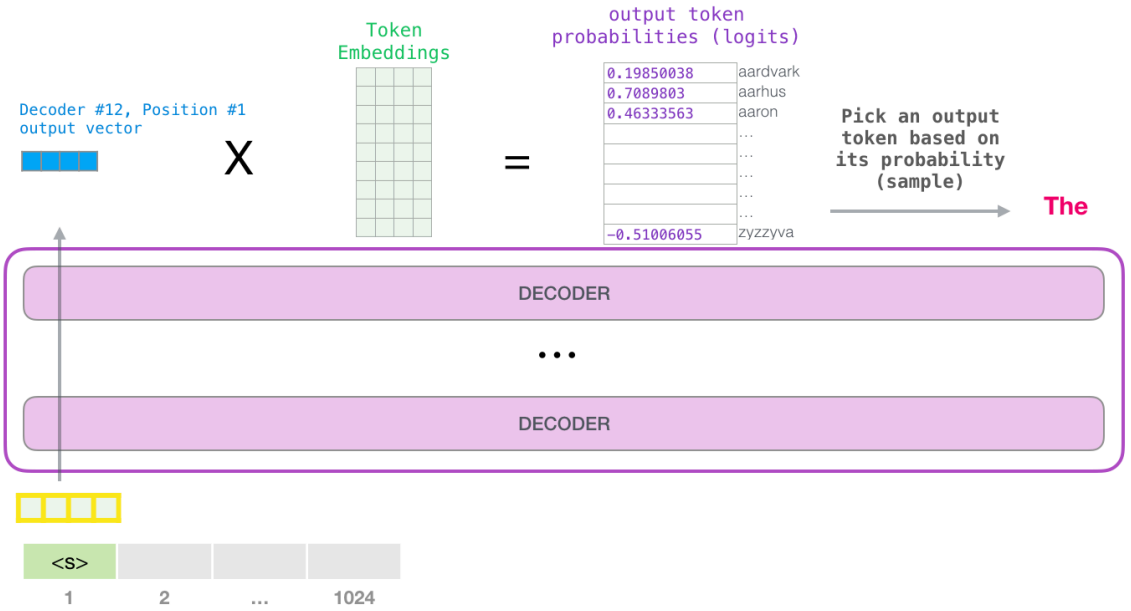
➢ *The language heavily relies on context. The model will understand relevant and associated words by assigning a score to how relevant each word is.*

➢ *For example, here is the word, and the attention of the word* `it`

**not here, down there*



- *This is the masked self-attention, we discussed this earlier.*
- *Let's talk about the output part of the decoder models*
- *When the top block in the model produces its output vector (the result of its self-attention followed by its neural network), the model multiplies that vector by the embedding matrix.*

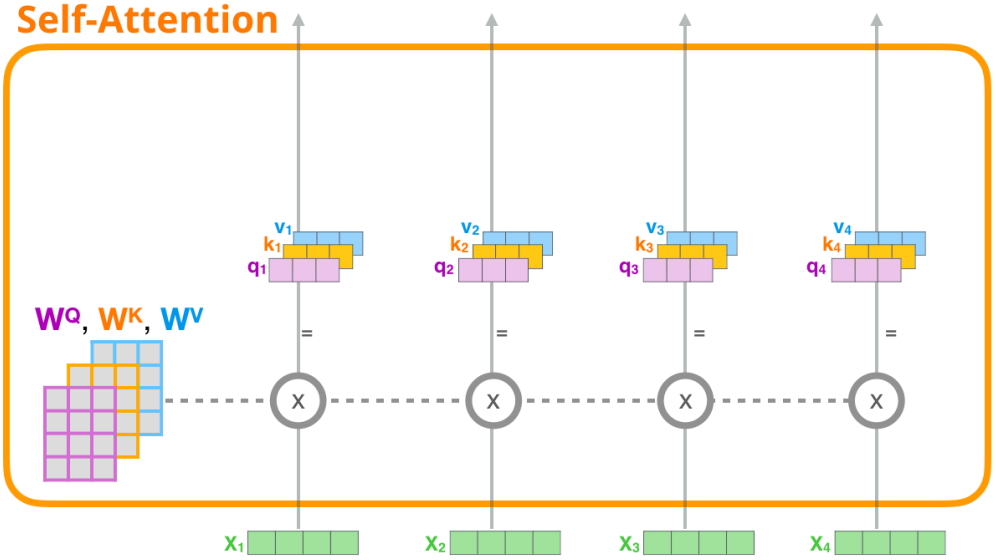


- *We can simply select the token with the highest score (tok_k = 1). The model continues iterating until the entire context is generated (1024 tokens) or until an end-of-sequence token is produced.*

- *If you understand self-attention and masked-self attention, It's greatttt. If not, I will explain in another way.*
- *Begin the self-attention, it has multiple steps,*

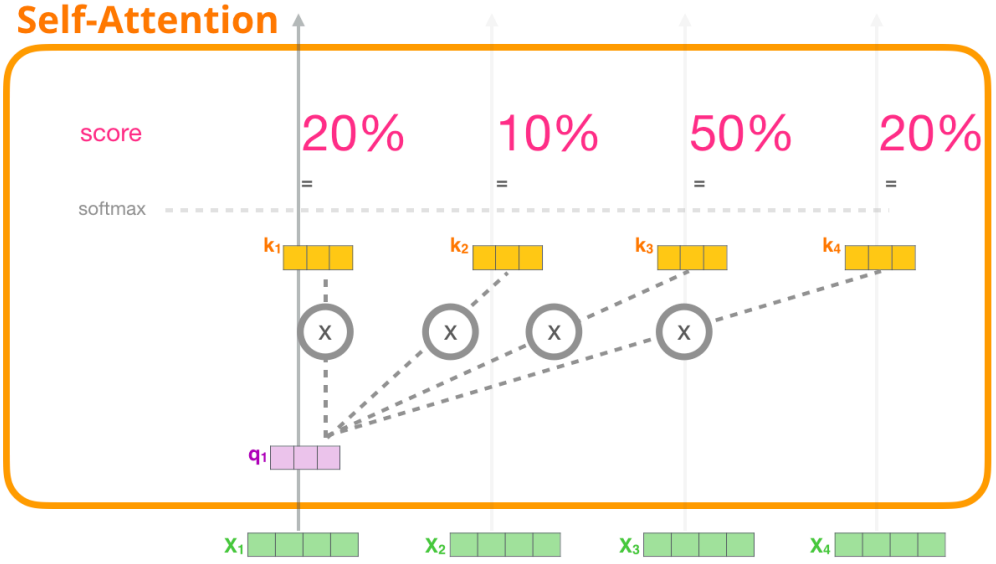
➢ *Create the Query, Key, and Value vectors for each path.*

1) For each input token, create a query vector, a key vector, and a value vector by multiplying by weight Matrices W^Q, W^K, W^V



➢ *Now calculate the* `score`

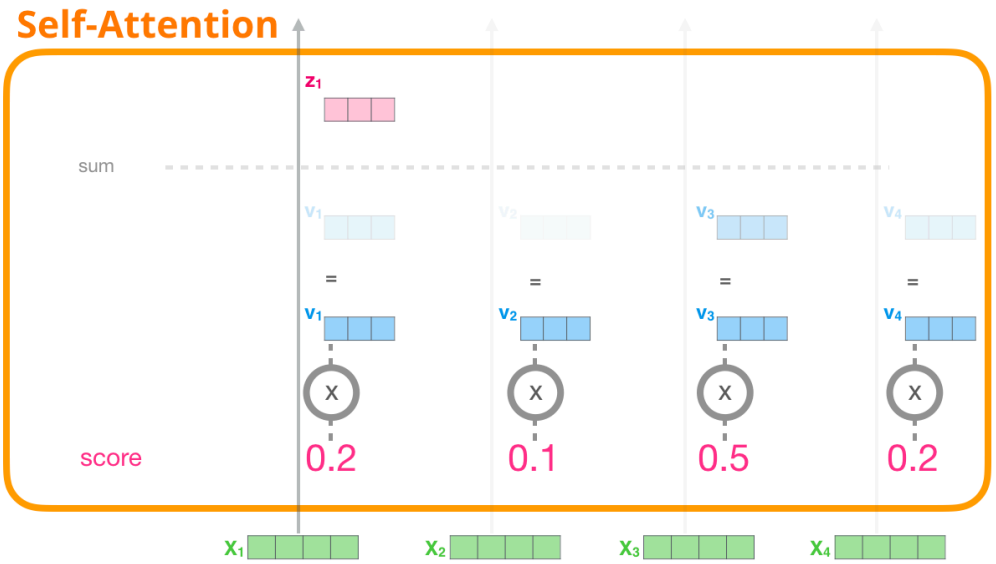
2) Multiply (dot product) the current query vector, by all the key vectors, to get a score of how well they match



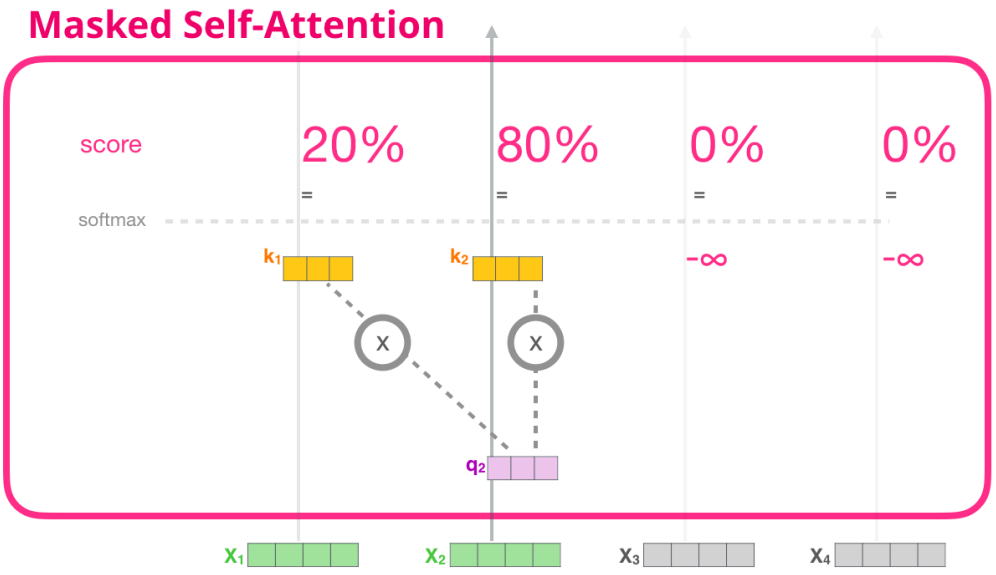
➢ *It is the time to sum it up*

**shhhh, down there …..*

3) Multiply the value vectors by the scores, then sum up



- *That's all, I have explained in 2 ways, you can use either of them and learn*
- *Let's see the **masked self-attention** also*
- *Here, it always scores the future tokens as 0 so the model can't peak to future words*



- *So, in normal attention, the score is calculated as follows*

Queries

robot	must	obey	orders

X

Keys

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

=

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

➢ *But in the case of masked attention, there will be one more step also*

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask →

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

➢ *And after that, we will apply the softmax layer*

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax (along rows) →

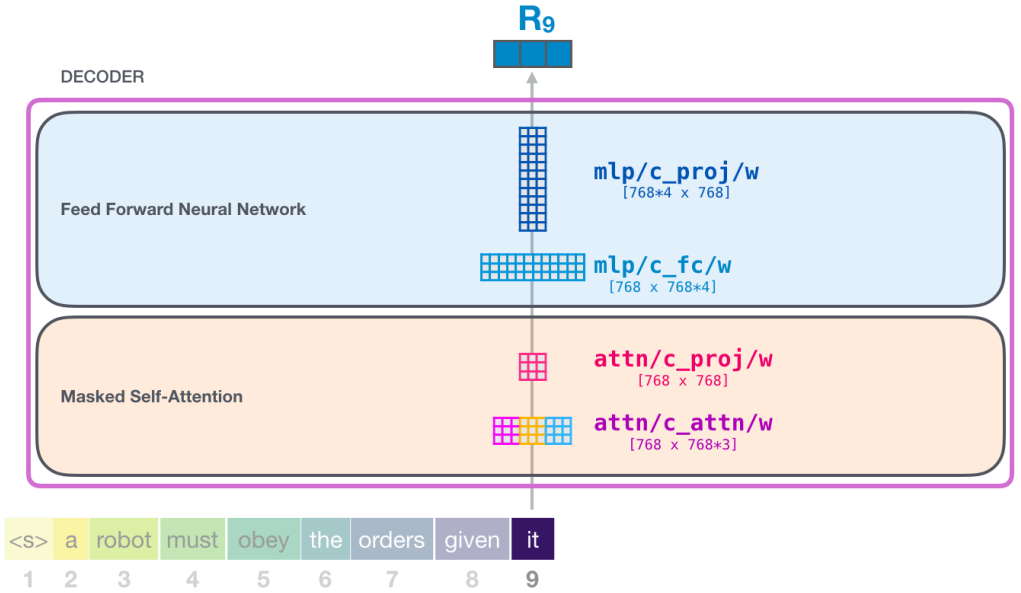
Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

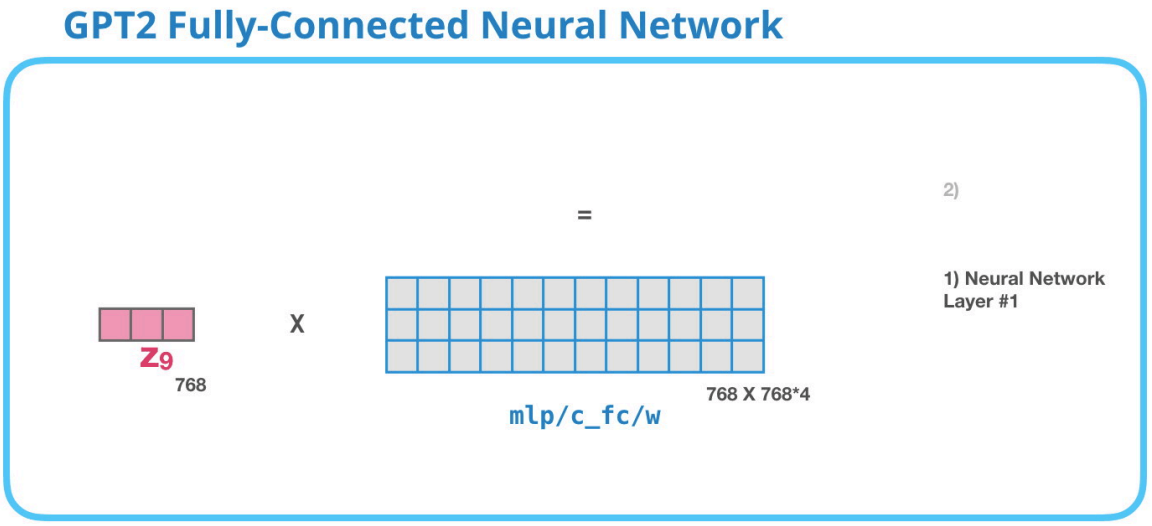
➢ *That's all about the attention part.*

➢ *I just wanna add one more thing to the output layer*
➢ *The output from the last decoder is passed to a fully connected layer.*
➢ *It is made up of two layers. The first layer is four times the size of the model (Since GPT2 small is 768, this network would have 768*4 = 3072 units).*
➢ *Why four times? That's just the size the original transformer rolled with (the model dimension was 512 and layer #1 in that model was 2048).*
➢ *So both fully connected layers look like this*

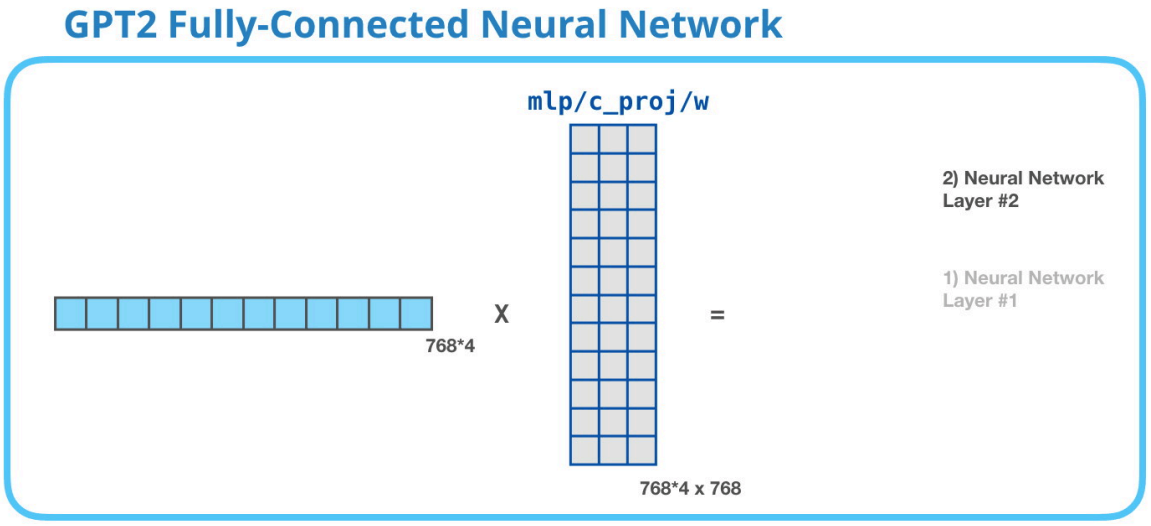
First the decoder …..



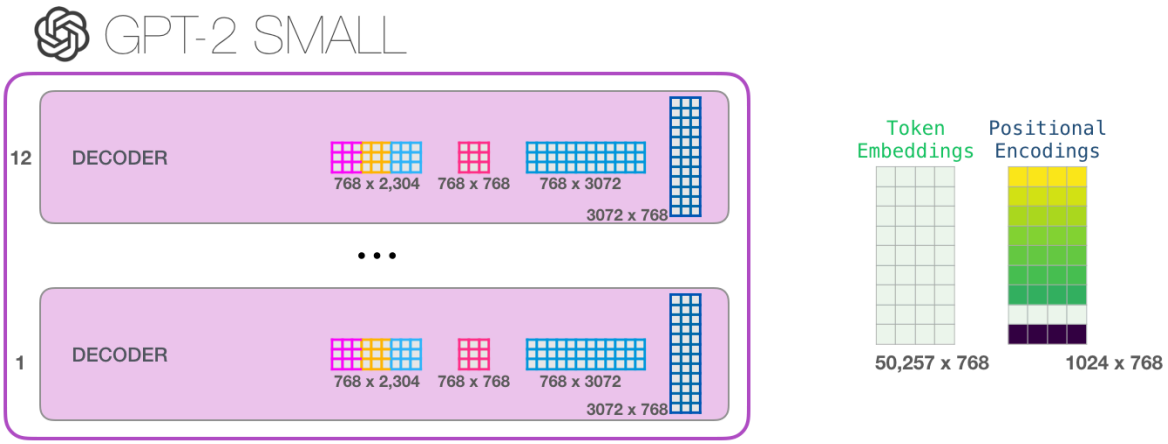
The first fully connected one …..



The second fully connected one ……

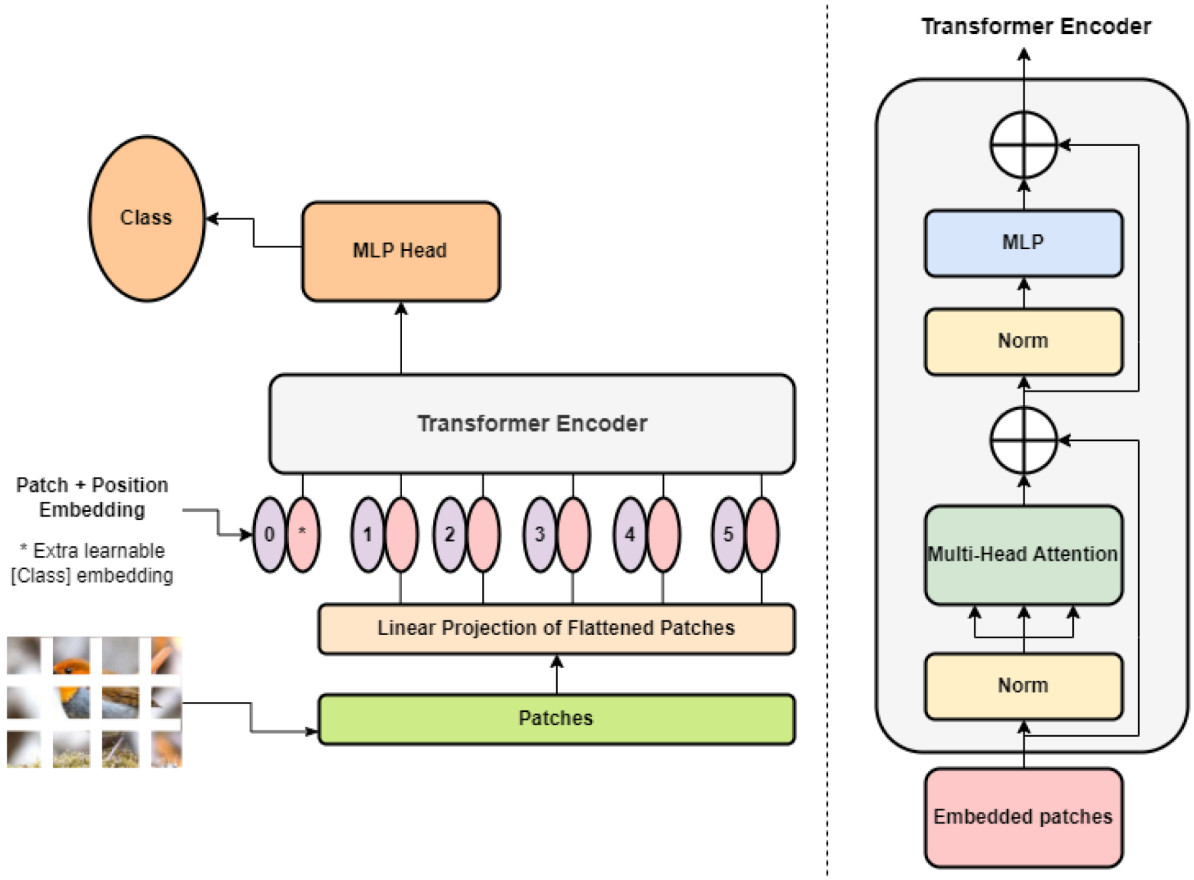


➢ *Overall here is the recap of the GPT-2 small*



Transformer vision (ViT)

Referring to the paper [An Image is Worth 16x16 Words.pdf]



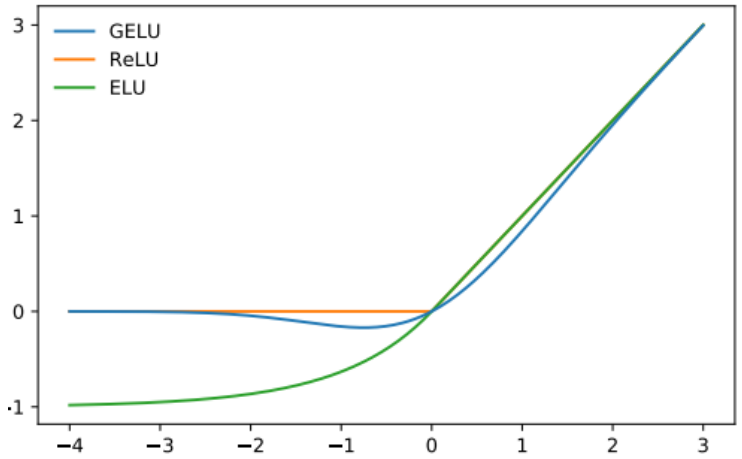
Vision transformers are types of transformers used for visual tasks such as image processing. Let's dive into the working part of that

- ➢ *The core concepts are always* `attention` *and* `multi-head attention`*.*
- ➢ *Usually, we use the CNN models for this task. But the transformers are working well in these tasks*
- ➢ *Here are the steps of ViT:*
  - *Split an image into patches (fixed size)*
  - *Flatten the image patches*
  - *Create lower-dimensional linear embedding from these flattened image patches.*
  - *Include positional embedding*
  - *Feed the sequence as an input to a state-of-the-art transformer encoder*
  - *Pre-train the ViT model with image labels, which is then fully supervised on a big dataset*
  - *Fine-tune the downstream dataset for image classification*

**again , just scroll*

*Here is a simple example of the **ViT***

- *Mainly this architecture has 2 blocks*
  - *Layer Norm*
    - *keeps the training process on track and lets the model adapt to the variations among the training images.*
  - *Multi-Head Attention Network (MSP)*
    - *is a network responsible for generating attention maps from the given embedded visual tokens.*
  - *Multi-Layer Perceptron (MLP)*
    - *a two-layer classification network with GELU (Gaussian Error Linear Unit) at the end.*
- *Here you may have seen a thing called GELU, it is nothing but an activation just like ReLu, ELU, etc. It is used in GPT and BERT.*



➢ *If you doubt why we split the data into patches, in the text we tokenized at first, it is like that.*

Sentence to word tokens:

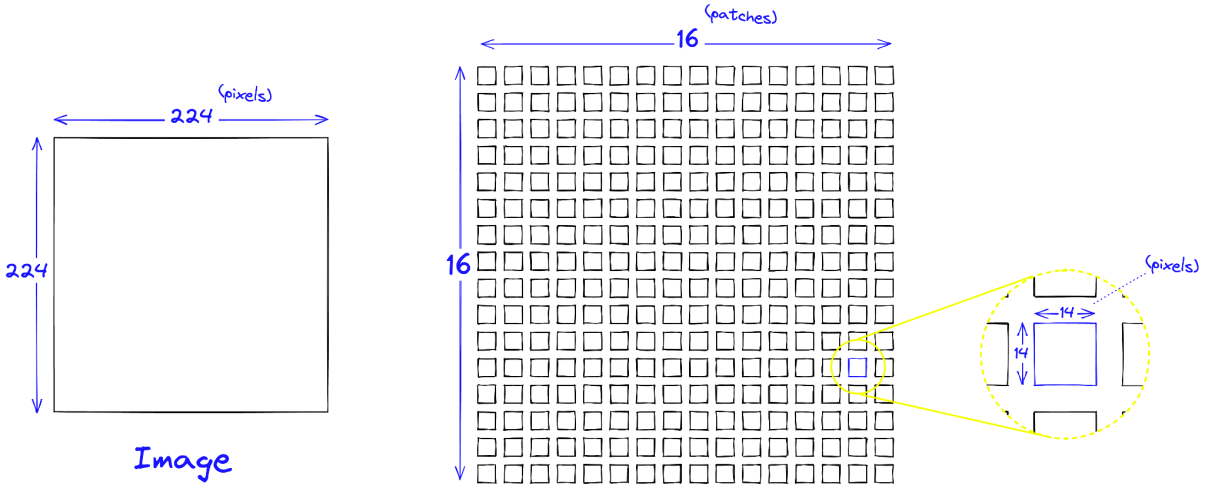
"hi, I am a short sentence"

↓

'hi' ',' 'I' 'am' 'a' 'short' 'sentence'

Image to image patches:

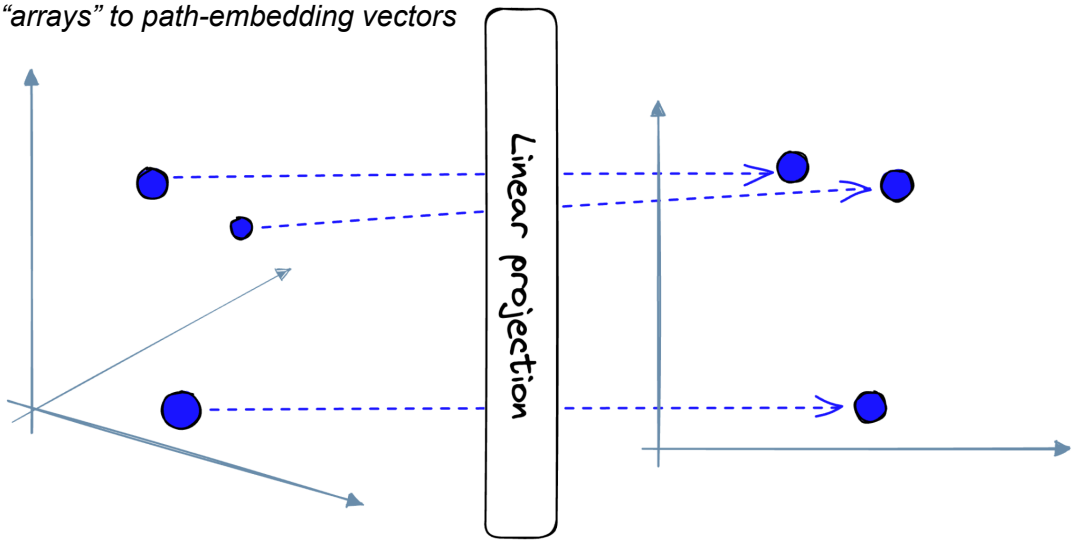
➢ *In more detail →*



➢ *Conversion of 224x224 pixel image into 256 14x14 pixel image patches.*

➢ *One more crazy coconut thing that you may not understood is the linear projection*

➢ *After building the image patches, a linear projection layer is used to map the image patch "arrays" to path-embedding vectors*



ViT vs. Convolutional Neural Networks

- ➢ *Unlike CNNs, ViT obtains the global representation from the shallow layers, but the local representation obtained from the shallow layers is also important.*
- ➢ *ViT retains more spatial information than ResNet*
- ➢ *ViT can learn high-quality intermediate representations with large amounts of data.*
- ➢ *Skip connections in ViT are even more influential than in CNNs (ResNet) and substantially impact the performance and similarity of representations.*

Reference

**Sorry, there are a hell lot of blogs that I referred to, I lost the count itself 🥵*
But I need to express my love to Jay Alamar, most of the things in here are his work 🖤

`.....`

And also I appreciate myself for preparing this within 2 days 😅😂😂
**cause nobody is here to clap for my efforts and sleepless nights other than me 🥹💔*
**and I know that is life …. 😎🫶*

With this, my little heart is stolen by the Transformers. It is really crazy 🤎