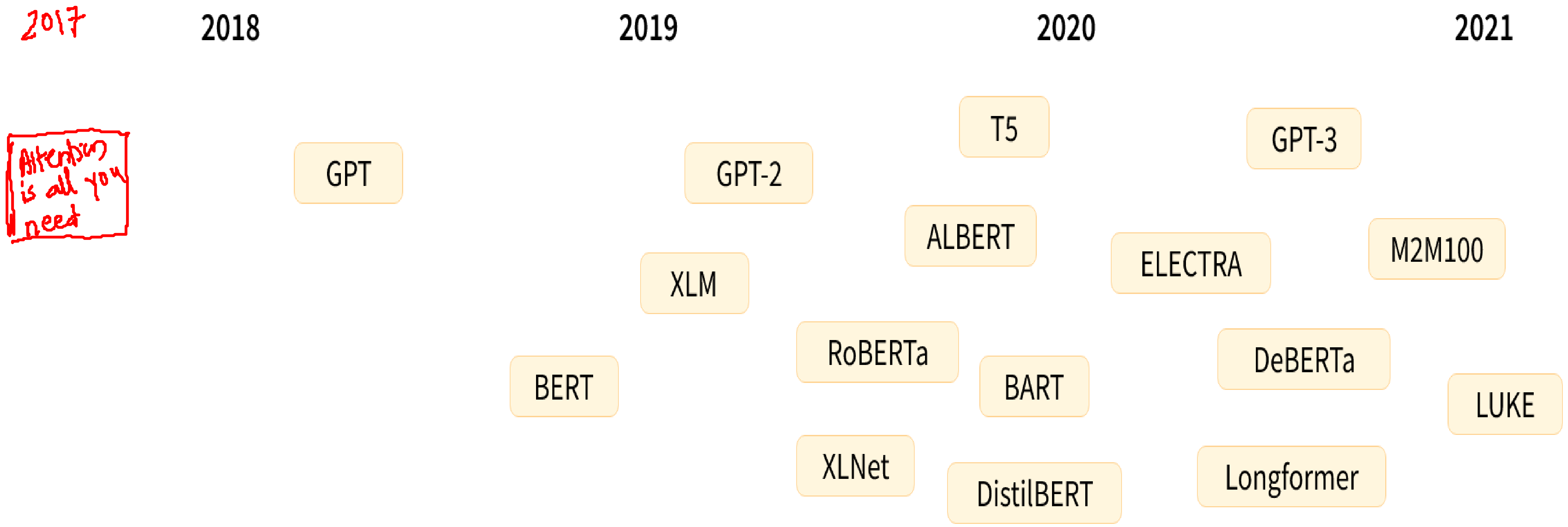
Transformers

Transformer History



Transformer History

The Transformer architecture was introduced in June 2017. The focus of the original research was on translation tasks. This was followed by the introduction of several influential models, including:

- **June 2018**: GPT, the first **G**enerative **P**retrained **T**ransformer model, used for fine-tuning on various NLP tasks and obtained state-of-the-art results
- **October 2018**: BERT, another large pretrained model, this one designed to produce better summaries of sentences (more on this in the next chapter!)
- **February 2019**: GPT-2, an improved (and bigger) version of GPT that was not immediately publicly released due to ethical concerns
- **October 2019**: DistilBERT, a distilled version of BERT that is 60% faster, 40% lighter in memory, and still retains 97% of BERT's performance
- **October 2019**: BART and T5, two large pretrained models using the same architecture as the original Transformer model (the first to do so)
- **May 2020**, GPT-3, an even bigger version of GPT-2 that is able to perform well on a variety of tasks without the need for fine-tuning (called *zero-shot learning*)

Transformer model Types

This list is far from comprehensive, and is just meant to highlight a few of the different kinds of Transformer models. Broadly, they can be grouped into three categories:

- GPT-like (also called *auto-regressive* Transformer models)
- BERT-like (also called *auto-encoding* Transformer models)
- BART/T5-like (also called *sequence-to-sequence* Transformer models)

Transformers are language models

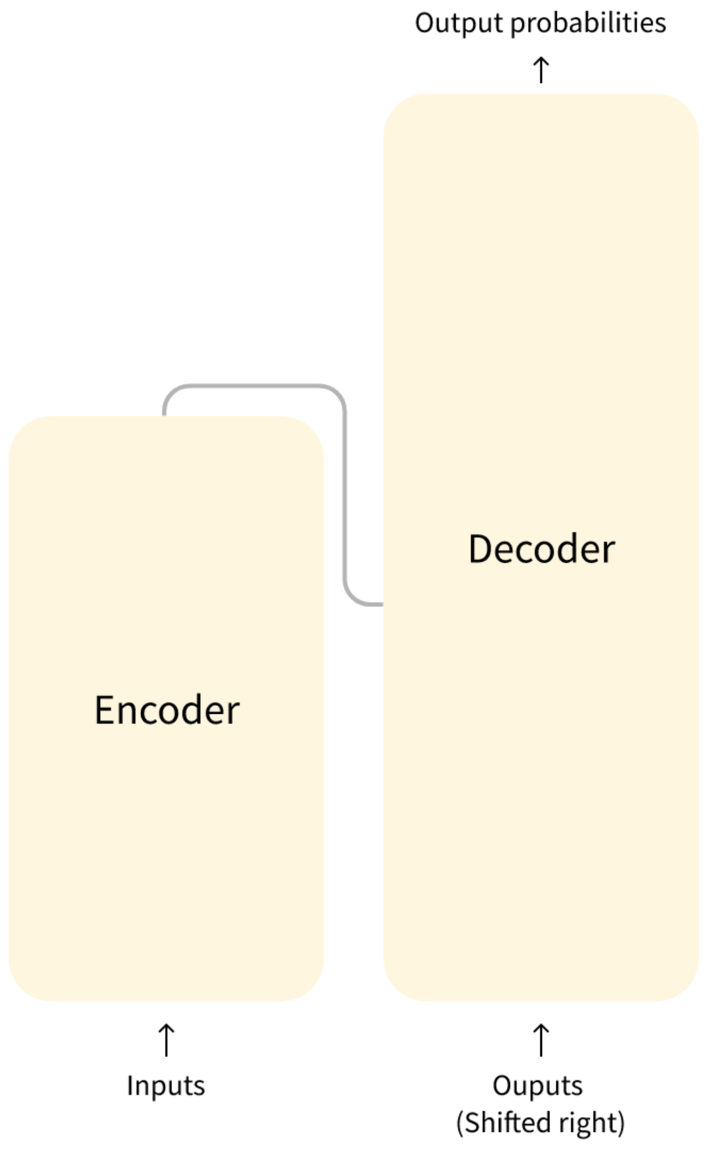
- Transformer models have been trained on large amounts of raw text in a self-supervised fashion
  - Self-supervised learning is a type of training in which the objective is automatically computed from the inputs of the model
  - That means that humans are not needed to label the data!

- This type of model develops a statistical understanding of the language it has been trained on
  - but it’s not very useful for specific practical tasks
  - Because of this, the general pretrained model then goes through a process called transfer learning
  - During this process, the model is fine-tuned in a supervised way — that is, using human-annotated labels — on a given task.

- An example of a task is predicting the next word in a sentence having read the n previous words
  - This is called causal language modeling because the output depends on the past and present inputs, but not the future ones

Transformer Model

- The model is primarily composed of two blocks:
  - Encoder (left)
    - The encoder receives an input and builds a representation of it (its features)
    - This means that the model is optimized to acquire understanding from the input.
  - Decoder (right)
    - The decoder uses the encoder's representation (features) along with other inputs to generate a target sequence
    - This means that the model is optimized for generating outputs.



Attention layers

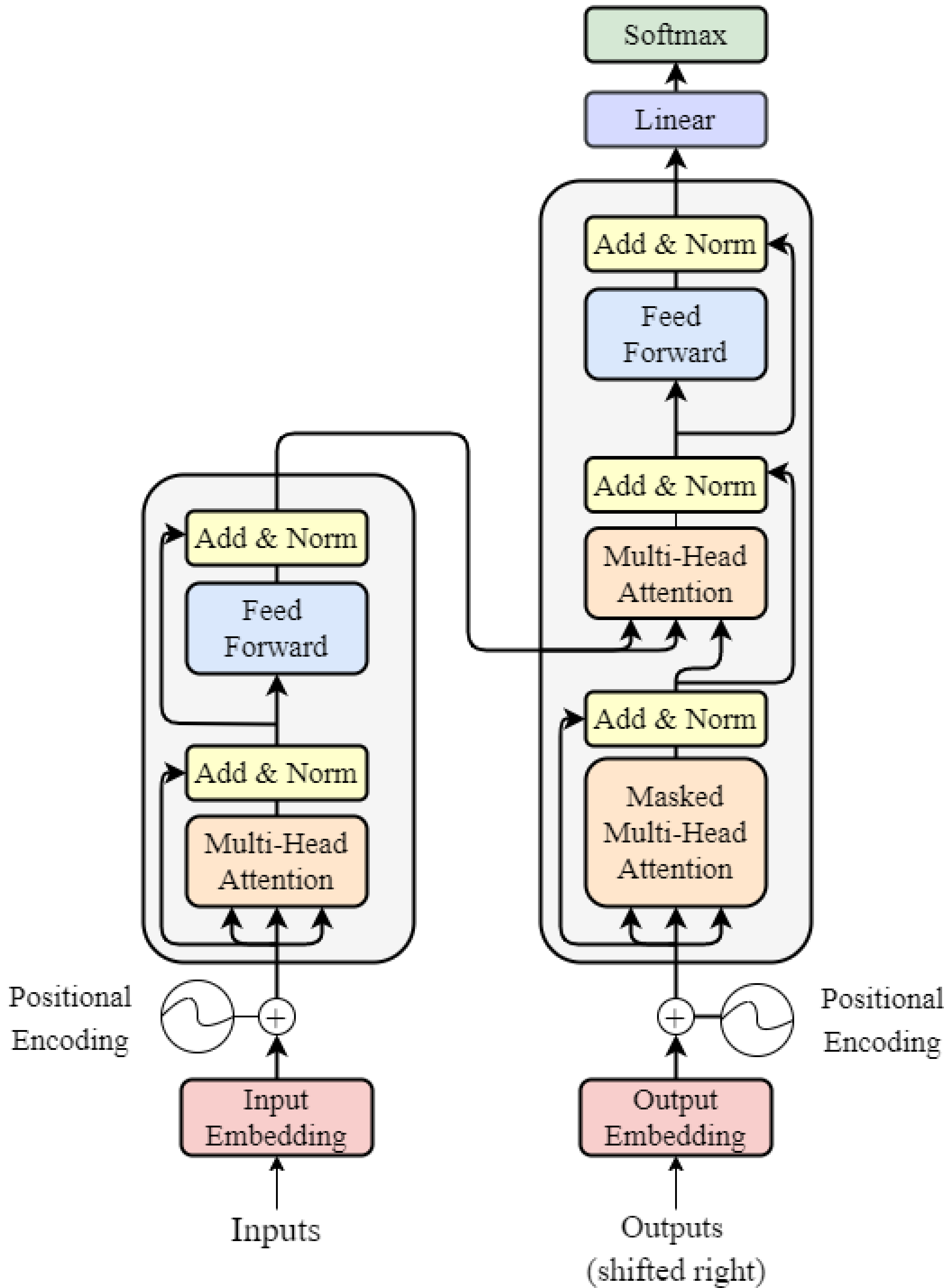
- Consider the task of translating text from English to French e.g. given “You like this course”
    - to get the proper **translation for the word “like”** a translation model will need to also attend to the adjacent word “You”
    - because in French the verb “like” is conjugated differently depending on the subject
    - the rest of the sentence, however, is not useful for the translation of that word
    - in the same vein, **when translating “this”** the model will also need to pay attention to the word “course”
    - because “this” translates differently depending on whether the associated noun is masculine or feminine
    - again, the other words in the sentence will not matter for the translation of “this”.
- With more complex sentences (and more complex grammar rules), the model would need to pay special attention to words that might appear farther away in the sentence to properly translate each word.
- A key feature of Transformer models is that they are built with special layers called attention layers
- This layer will tell the model to pay specific attention to certain words in the sentence you passed it (and more or less ignore the others) when dealing with the representation of each word.
- The same concept applies to any task associated with natural language
    - a word by itself has a meaning, but that meaning is deeply affected by the context
    - which can be any other word (or words) before or after the word being studied.

Transformer Architecture

- The Transformer architecture was originally designed for translation
- During training, the encoder receives inputs (sentences) in a certain language, while the decoder receives the same sentences in the desired target language
- In the encoder
    - the attention layers can use all the words in a sentence (the translation of a given word can be dependent on what is after as well as before it in the sentence)
- The decoder
    - however, works sequentially and can only pay attention to the words in the sentence that it has already translated
    - For example, to predict the 4th word of a sentence, the decoder uses the first three predicted words of the translated target, and all the inputs of the encoder
    - To speed things up during training (when the model has access to target sentences), the decoder is fed the whole target, but it is not allowed to use future words

Transformers Architecture

- The transformer is a relatively new network architecture that is solely based on attention mechanisms
- It consists of an encoder-decoder architecture
- **the encoder**
  - maps an input sequence of symbol representations (x1,…,xn) to a sequence z=(z1,…,zn)
- **the decoder**
  - given z , generates an output sequence (y1,…,ym)
  - At each time step the model consumes the previously generated symbols as additional input when generating the next (it is auto-regressive)



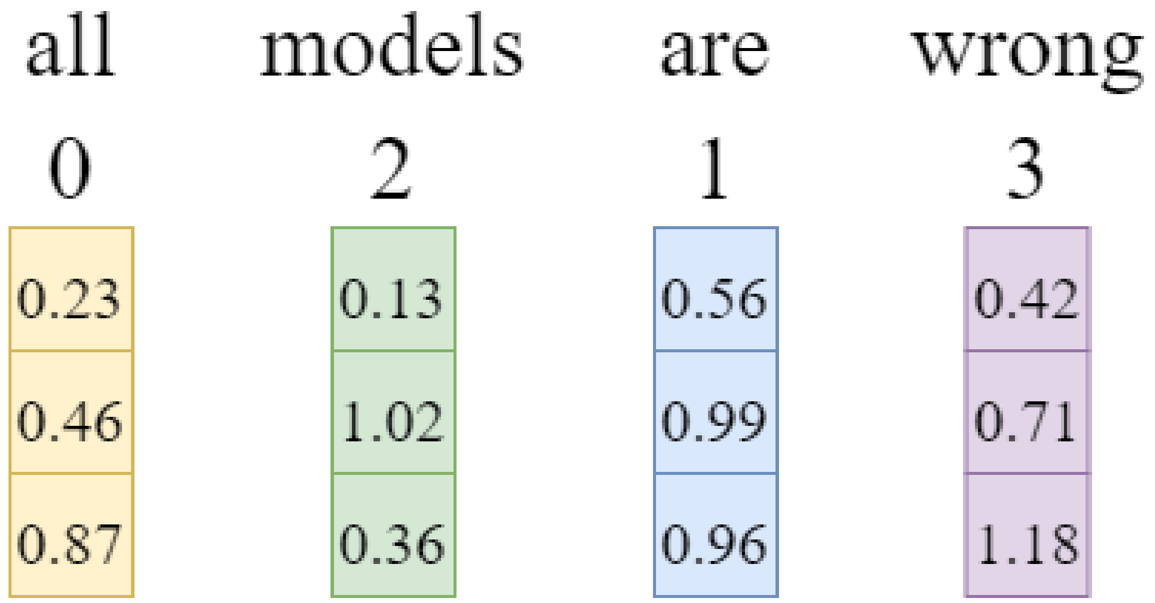
Encoding

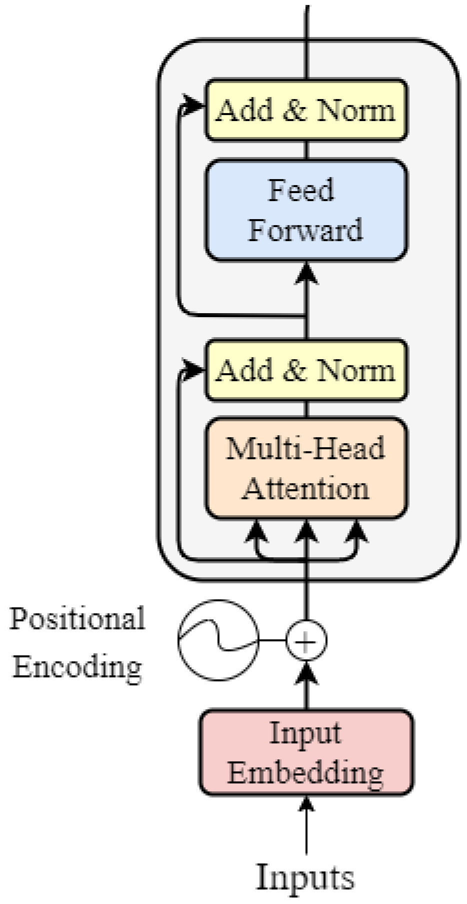
- The first thing the transformer does is transforming the input text into numbers
- To do that, we generate a basic vocabulary by taking all different words that are contained in the training data
- Each word will be represented by the index at which the word is stored in the vocabulary object
- For the sentence: [all are models wrong]
- Encoding will be: [0 2 1 3] (based on the dictionary)

https://quantdare.com/transformers-is-attention-all-we-need-in-finance-part-i/

Vector Embedding

- Next, we attach to each word a vector embedding that will allow to better represent the relations between words
- Similar words will be mapped with similar vectors

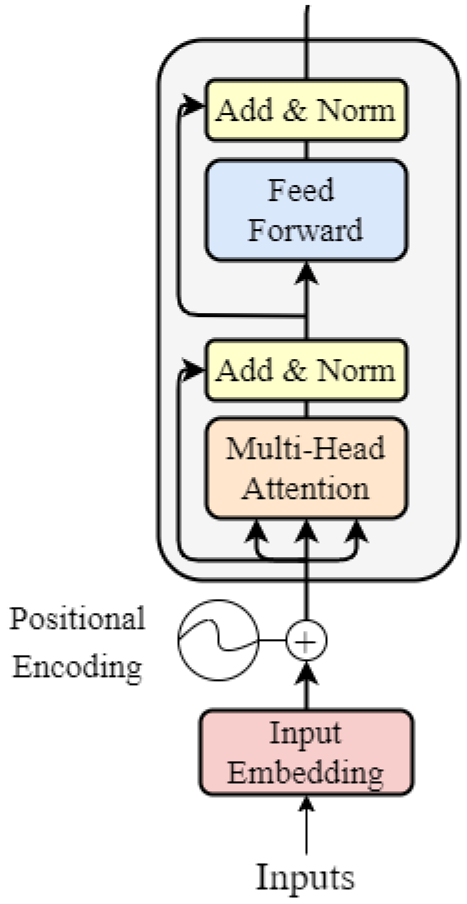




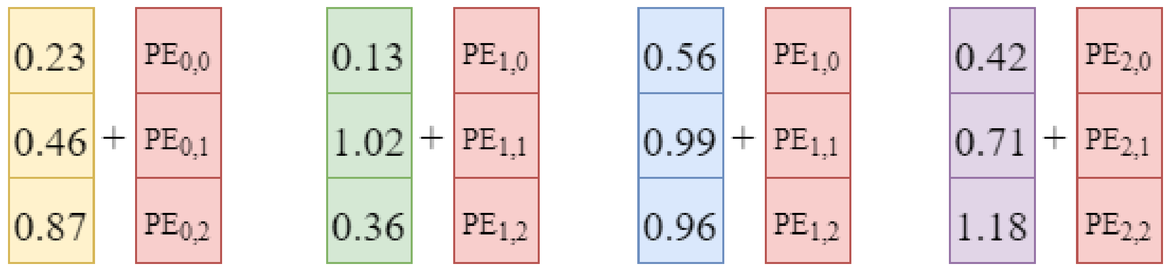
Positional Encoding

- We need to specify the order of words before we feed to the model
- For RNN, the words are processed sequentially, so no order needed
- But the transformer inputs all the words in parallel and uses an attention mechanism.
- The solution of the authors was to add a vector representing the position of the words to the embedding vectors.

$$\begin{bmatrix} 0.23 \\ 0.46 \\ 0.87 \end{bmatrix} + \begin{bmatrix} PE_{0,0} \\ PE_{0,1} \\ PE_{0,2} \end{bmatrix} \qquad \begin{bmatrix} 0.13 \\ 1.02 \\ 0.36 \end{bmatrix} + \begin{bmatrix} PE_{1,0} \\ PE_{1,1} \\ PE_{1,2} \end{bmatrix} \qquad \begin{bmatrix} 0.56 \\ 0.99 \\ 0.96 \end{bmatrix} + \begin{bmatrix} PE_{1,0} \\ PE_{1,1} \\ PE_{1,2} \end{bmatrix} \qquad \begin{bmatrix} 0.42 \\ 0.71 \\ 1.18 \end{bmatrix} + \begin{bmatrix} PE_{2,0} \\ PE_{2,1} \\ PE_{2,2} \end{bmatrix}$$



Positional Embedding



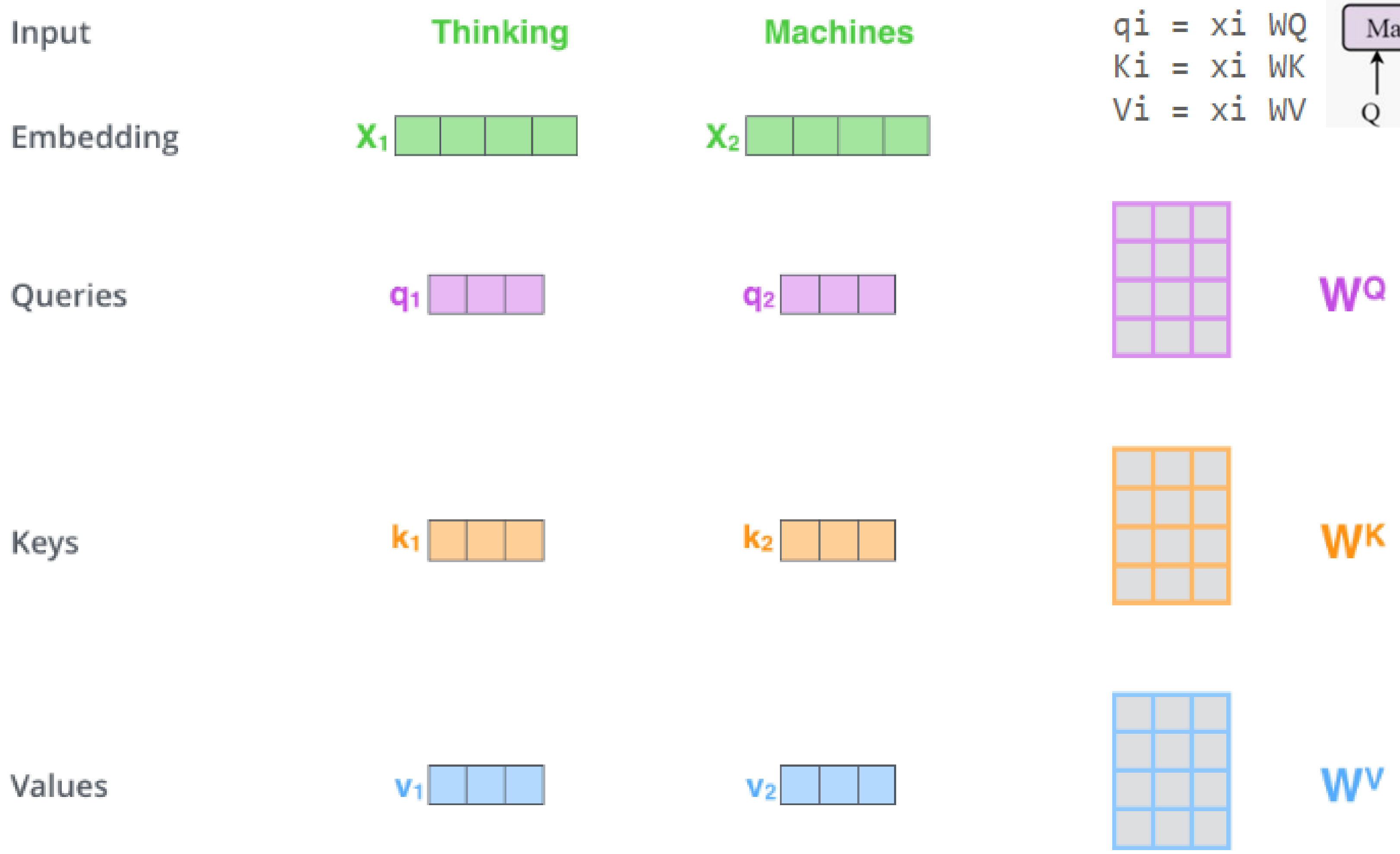
- Using Fourier analysis, the authors proposed an index-dependent function that encodes the position of each word as a sinusoidal wave while keeping low values
- For even and odd positions:

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

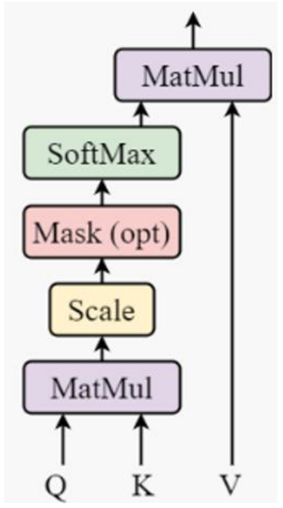
where pos is the position of the word, i is the i-th dimension of the word embedding and d_{model} is the number of dimensions in the embeddings.

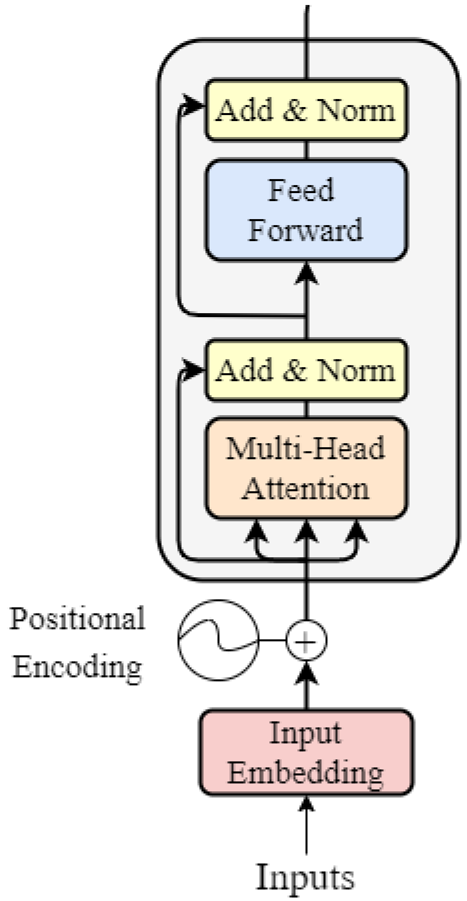
$$\text{Attention}(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

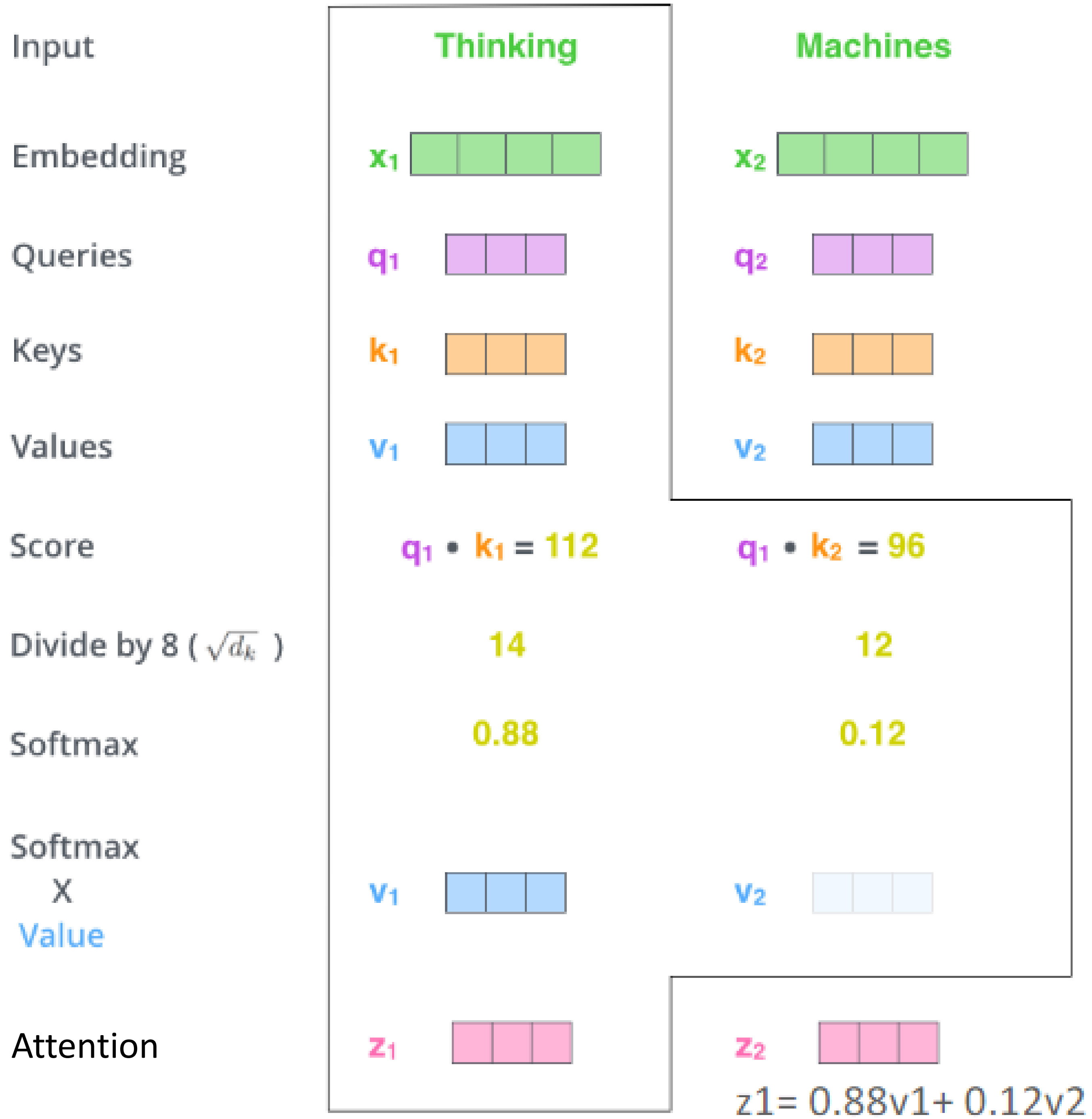
where Q is the query, K is the key, V is the value and d_k are the number of dimensions of K. The division by the square root of d_k is introduced to scalate the values and reduce possible problems with gradients.



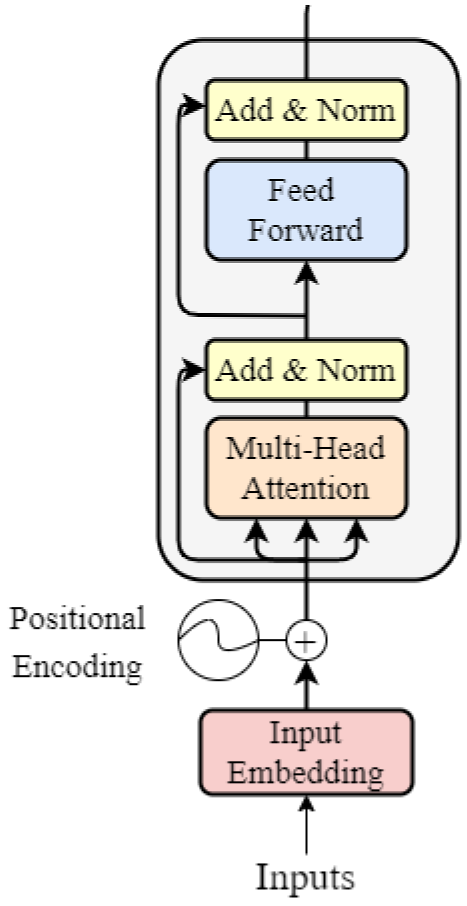
```
qi = xi WQ
Ki = xi WK
Vi = xi WV
```







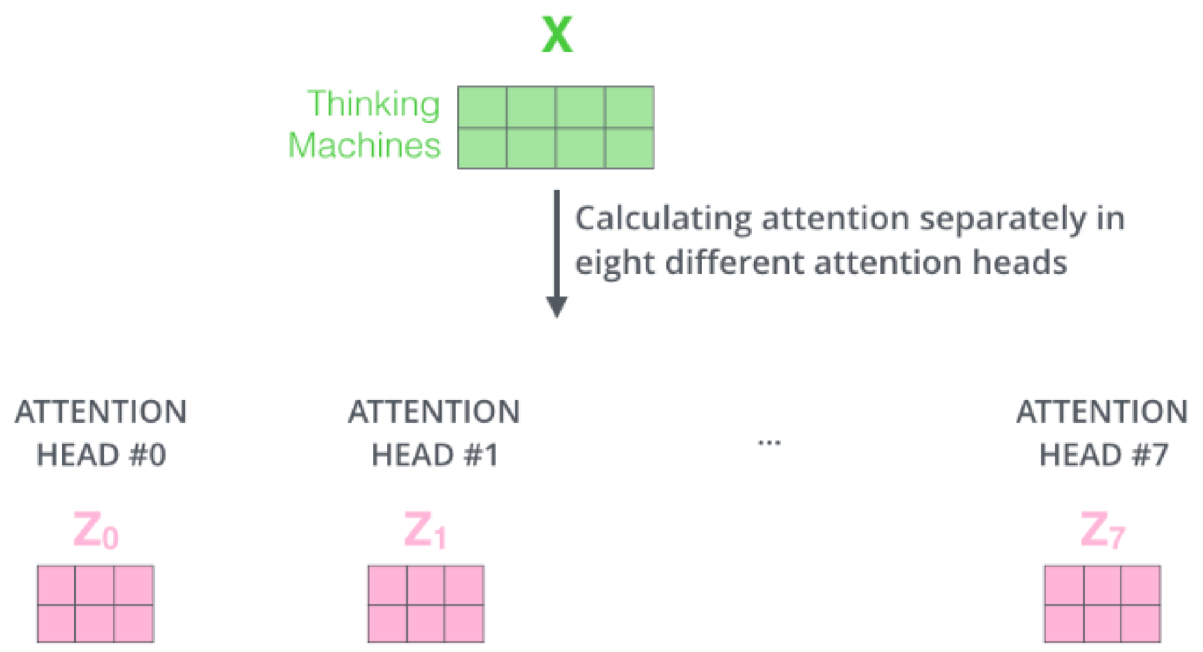
z1= 0.88v1+ 0.12v2

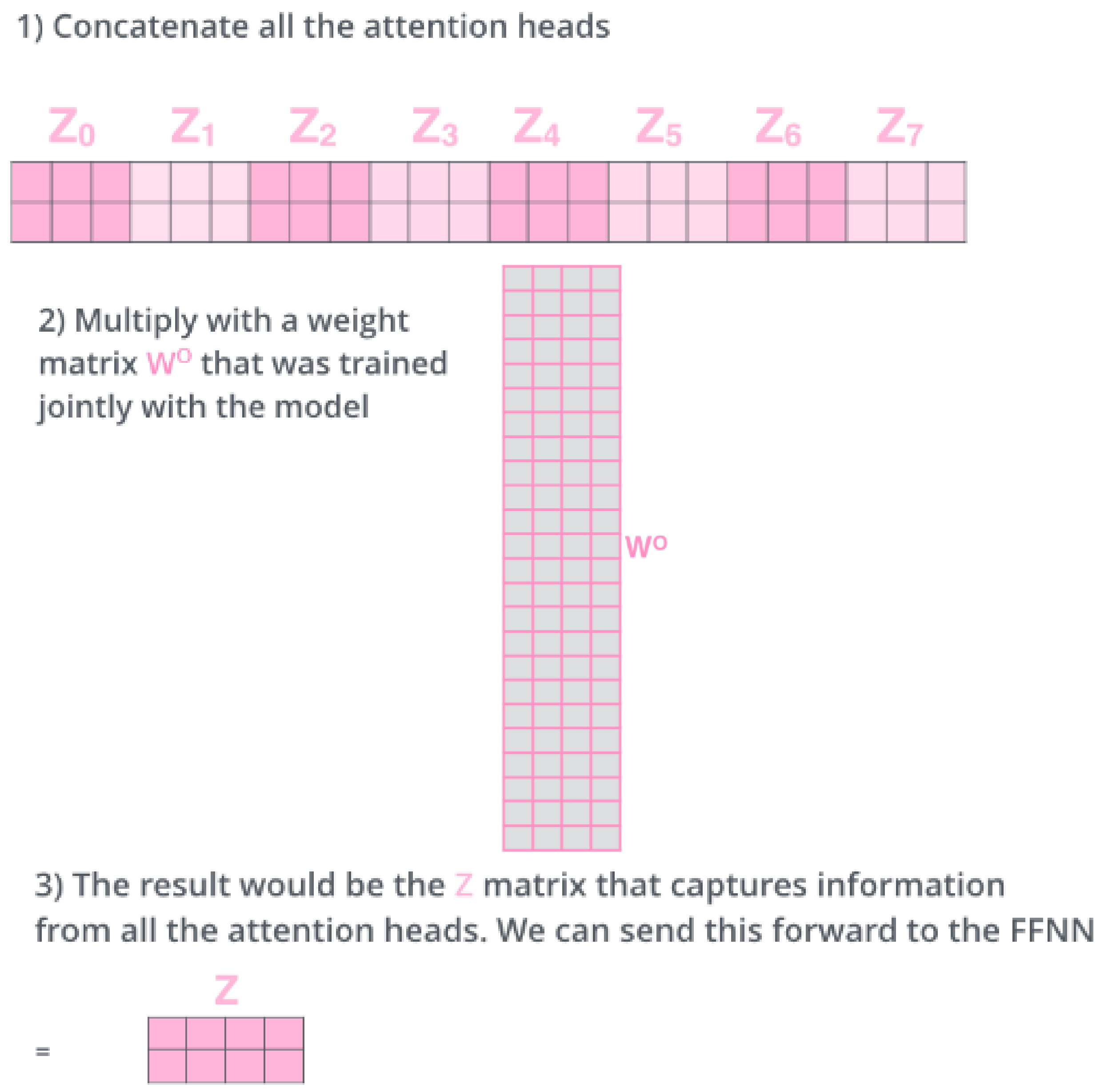


$$\text{Attention}(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q is the query, K is the key, V is the value and d_k are the number of dimensions of K. The division by the square root of d_k is introduced to scalate the values and reduce possible problems with gradients.

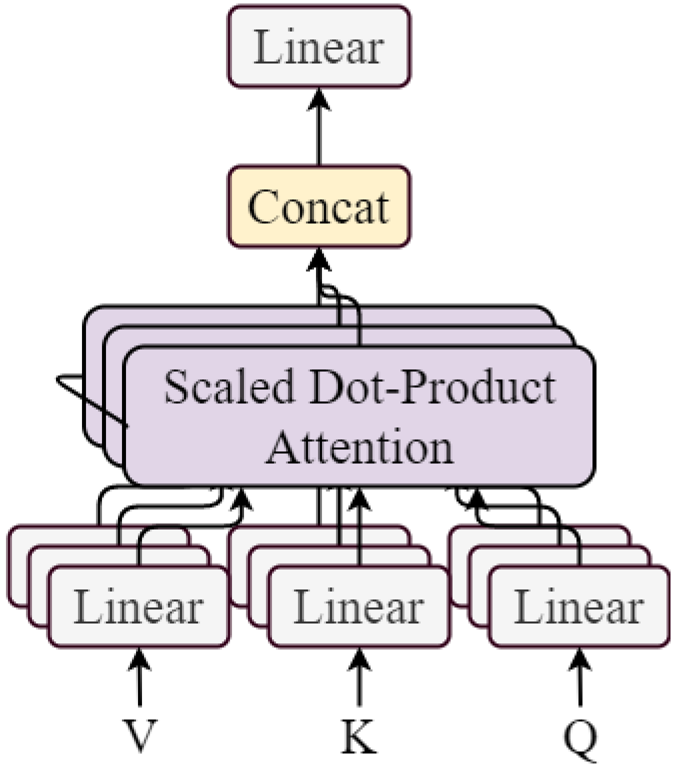
Multi Head Attention



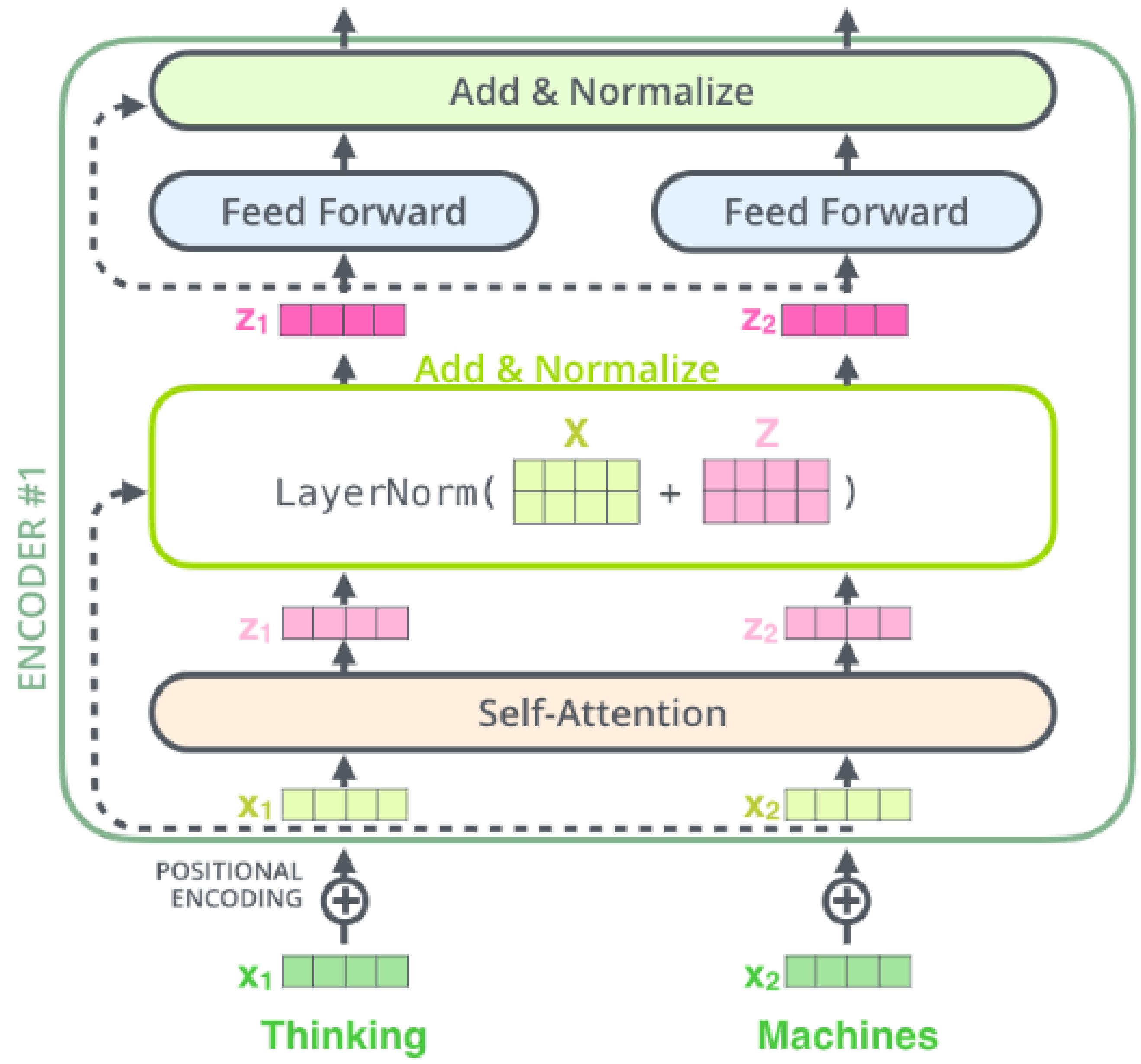

1) Concatenate all the attention heads
Z0 Z1 Z2 Z3 Z4 Z5 Z6 Z7
2) Multiply with a weight matrix W^O that was trained jointly with the model
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
=
Z

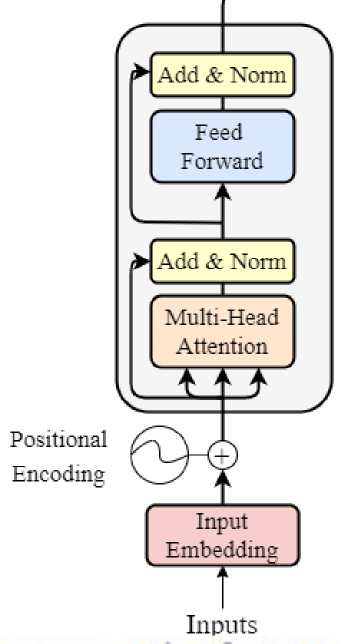
Multi head attention

- The transformer has more than one self-attention module
- Using these “heads” the model will hopefully be able to capture different features
- All attention scores are then concatenated in a single matrix, whose dimensions are reduced through the use of a linear mapping.

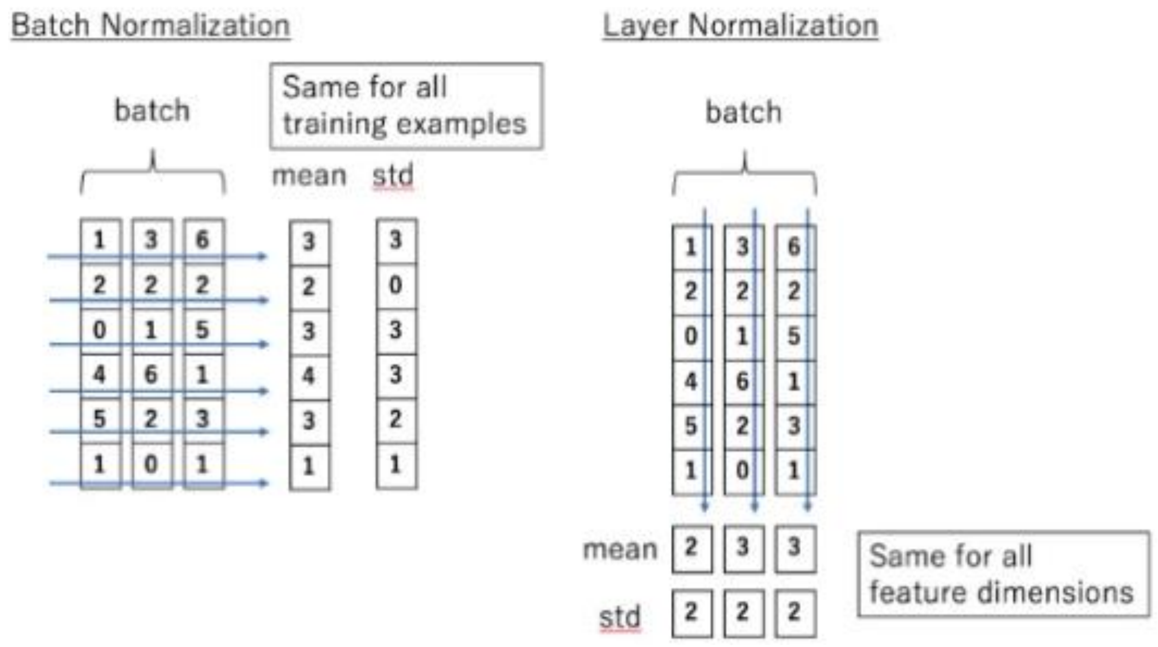


Add & Normalize





Layer normalization normalizes the inputs across the features.



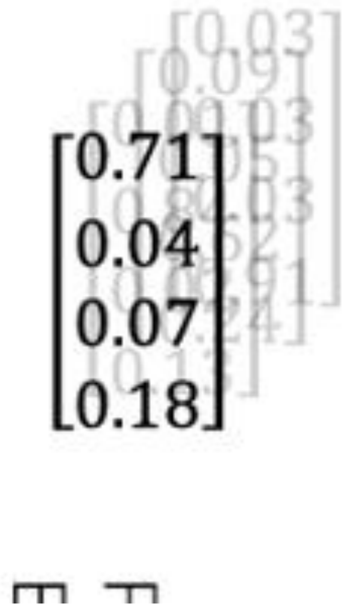
Batch Normalization

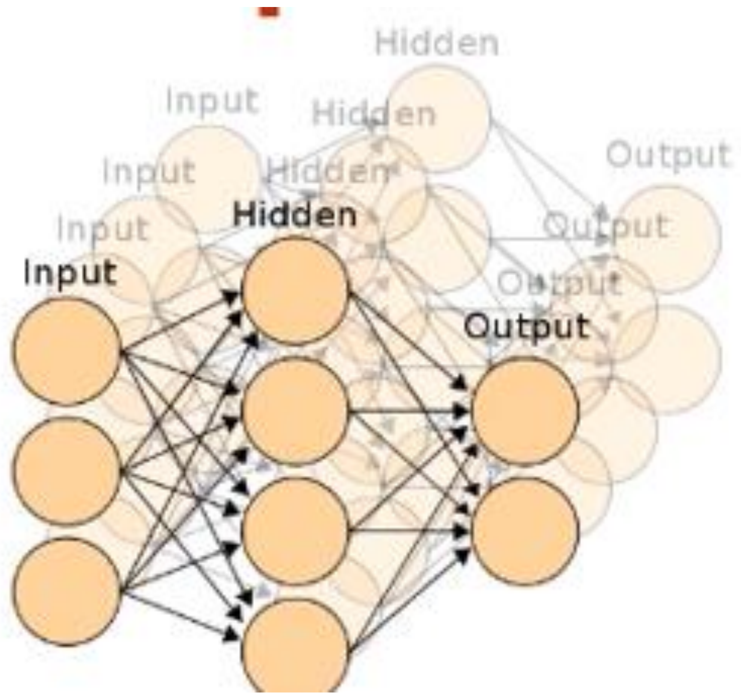
			mean	std
1	3	6	3	3
2	2	2	2	0
0	1	5	3	3
4	6	1	4	3
5	2	3	3	2
1	0	1	1	1

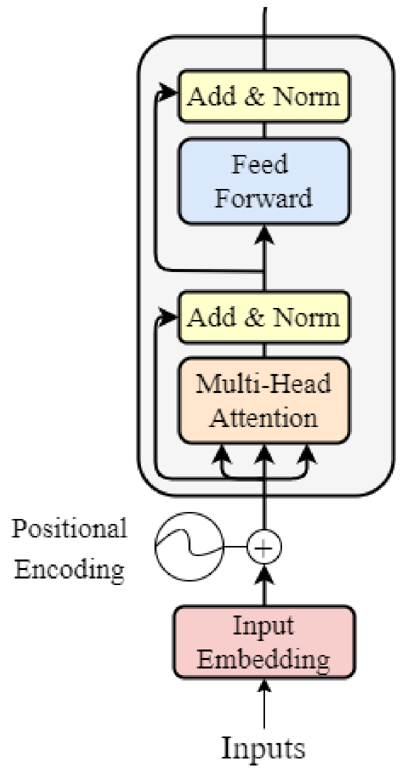
Layer Normalization

	1	3	6
	2	2	2
	0	1	5
	4	6	1
	5	2	3
	1	0	1
mean	2	3	3
std	2	2	2

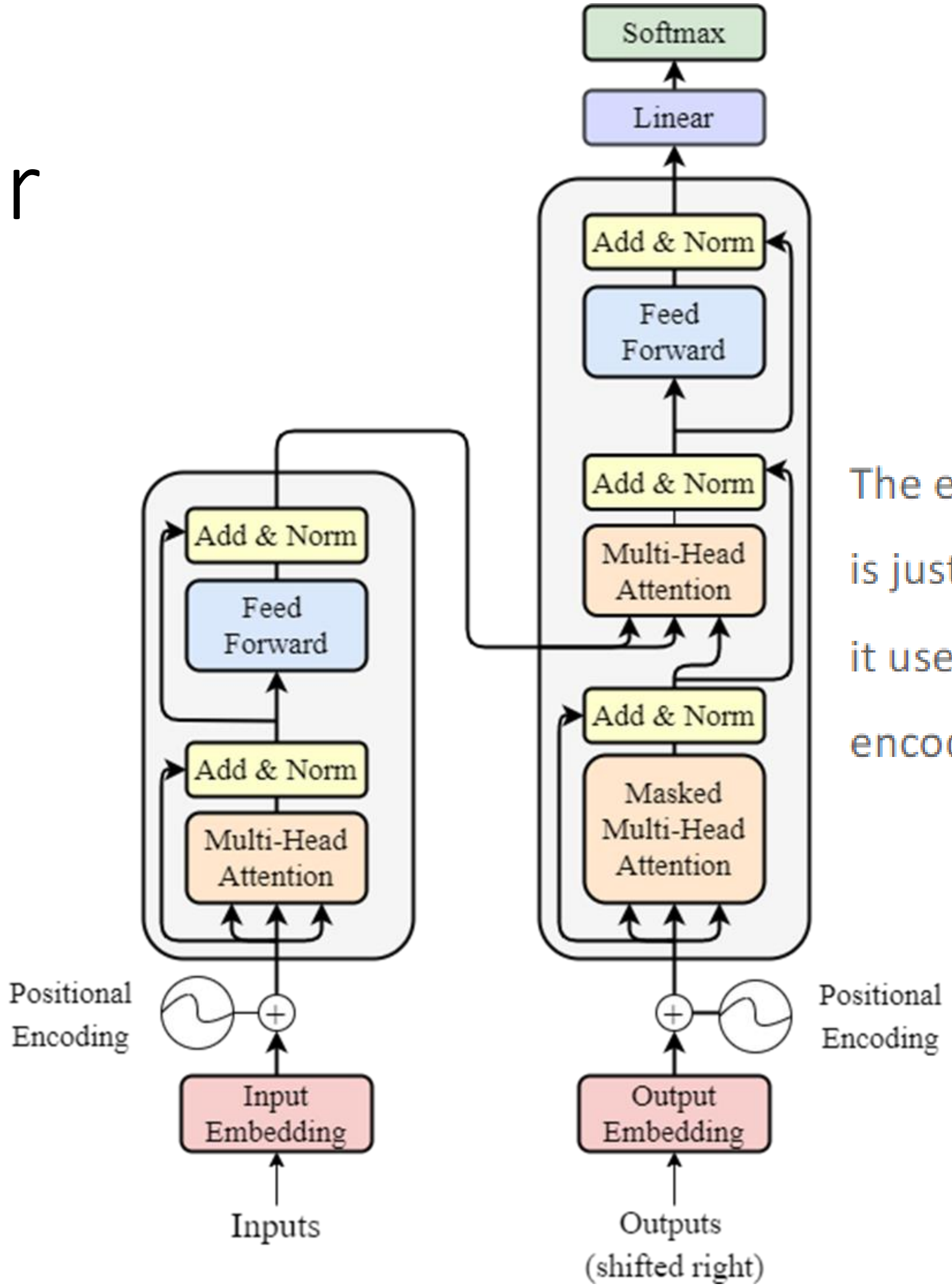
Feed Forward







The Transformer

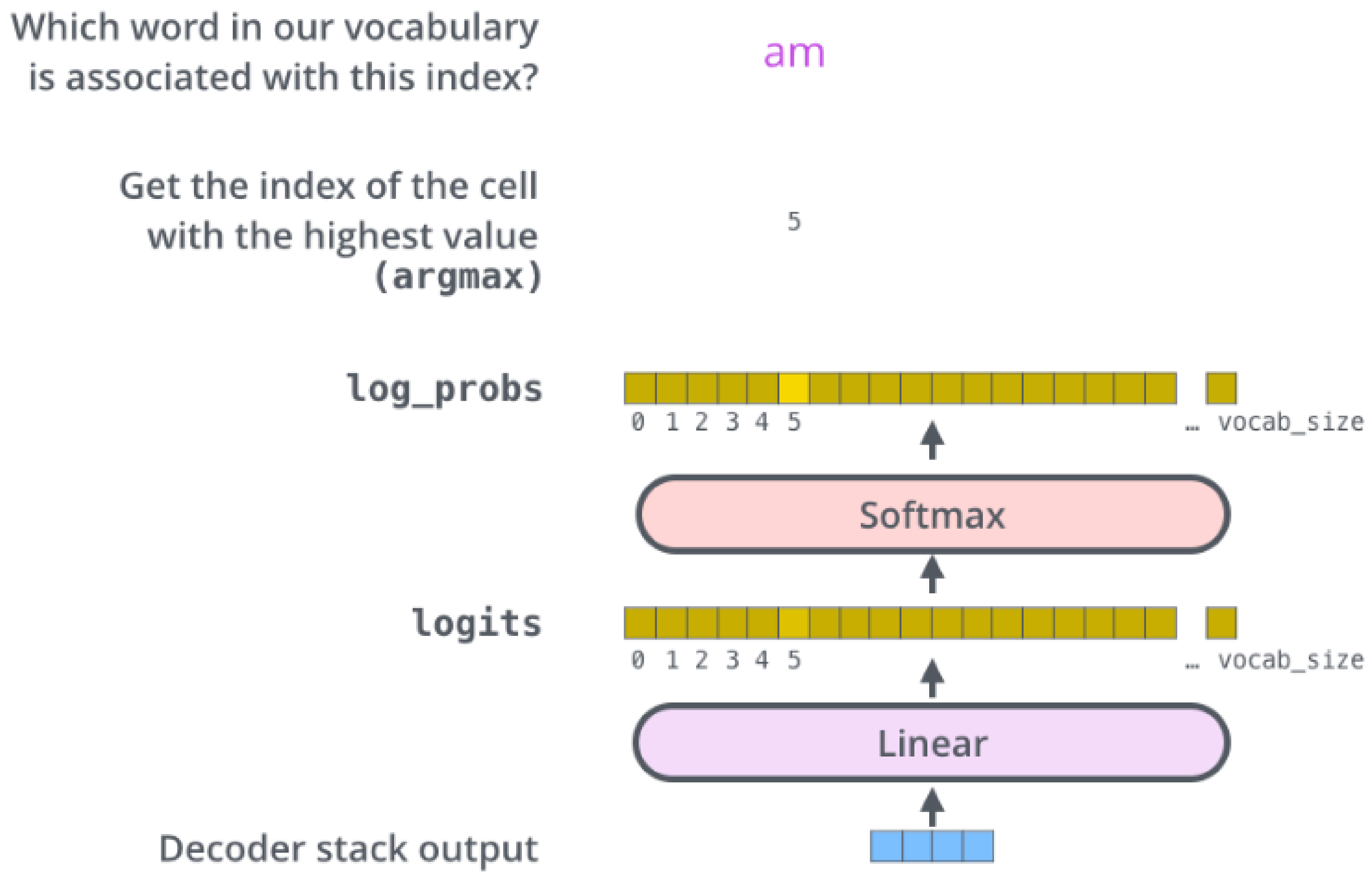


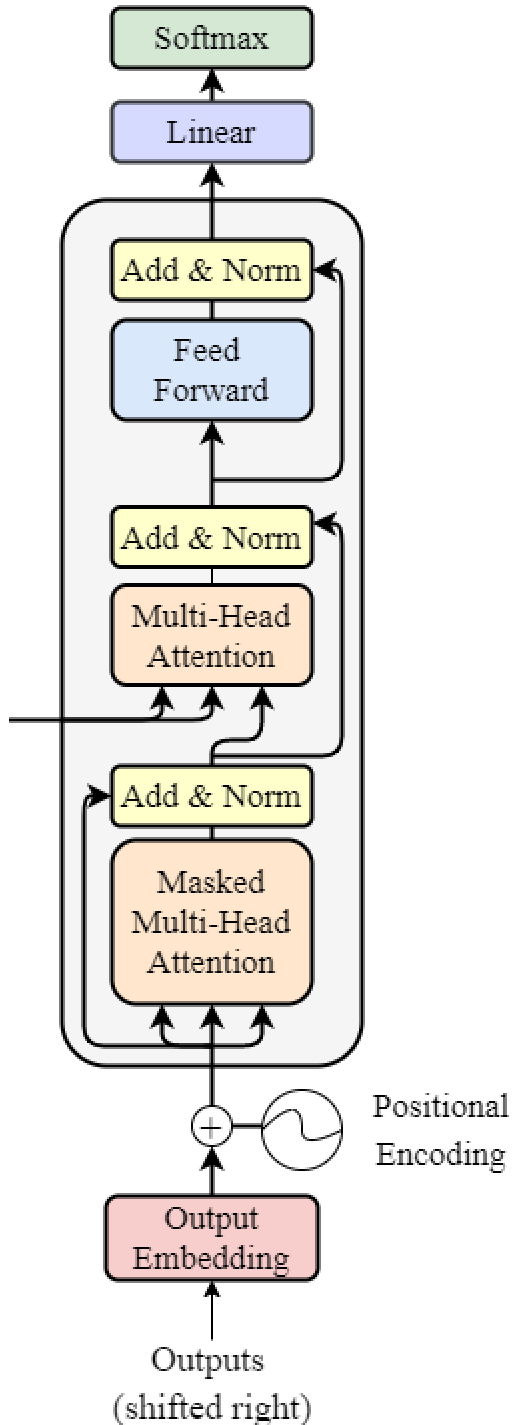
The encoder-decoder attention is just like self attention, except it uses K, V from the top of encoder output, and its own Q

Masked Multi-Head Attention

- The masked multi-head attention module ensures the model does not have access to future information in order to make the predictions.
- We know the transformer is an encoder-decoder architecture
    - to train the model, we provide the encoder with the first text sequence and we expect the decoder to generate the correct next sequence, one word at a time
    - We then show the decoder which word was the right one in order for the model to learn from its mistakes.
- The masking allows us to hide the future words by adding $-\infty$ values to the attention filter before feeding the matrix to the softmax layer
- The negative infinity elements will have a value of 0 after applying the softmax function, removing any illegal connections
- The masking will change at every time step, hiding less and less words. With this system the model is able to preserve the auto-regressive property

Decoder output

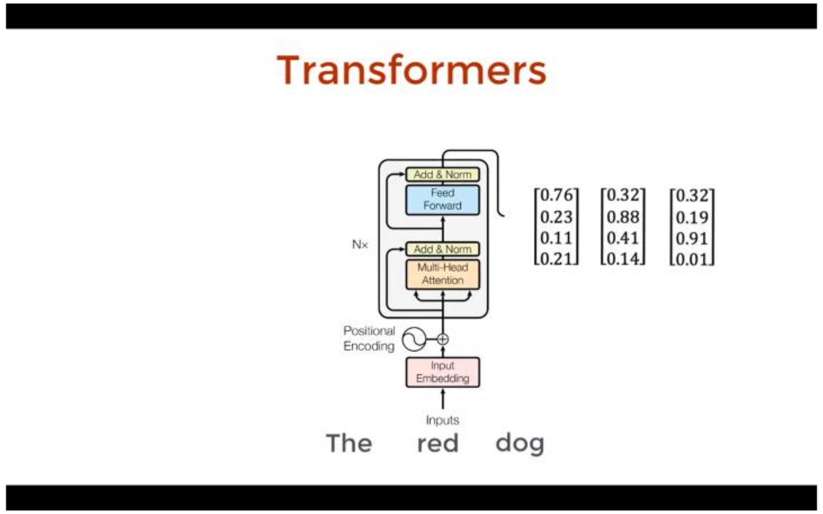

Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5 … vocab_size
Softmax
logits
0 1 2 3 4 5 … vocab_size
Linear
Decoder stack output



Summarizing

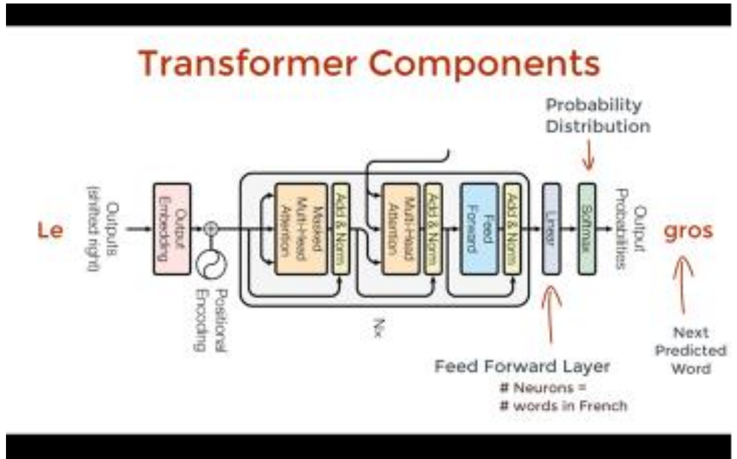
- The encoder takes the text and converts it to a vectorized representation
- The decoder takes this vectorized representation and the output generated so far (outputs) and makes a new prediction, that will be feed again
- The vectorized representation that is passed to the decoder are just copies of the final output of the encoder
    - We call them (again) key and value
- The output embedding that is transformed through a masked multi-head attention and then passed to the multi-head attention is called, to no one’s surprise, query.
- The process is repeated for each word until we reached an end token and… volilá! We have our transformer.

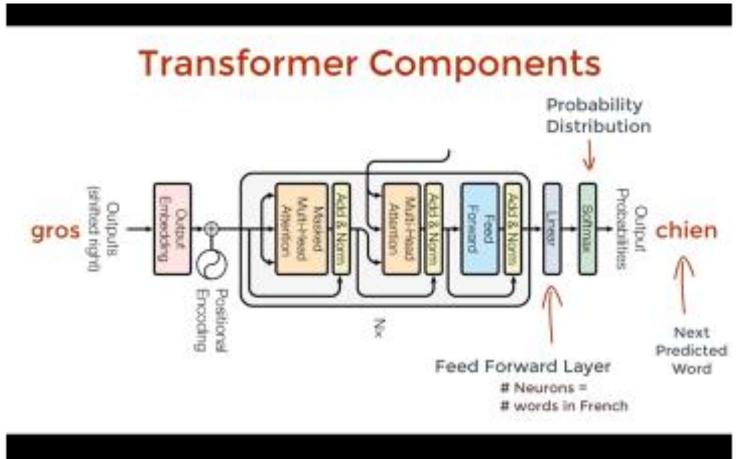
https://quantdare.com/transformers-is-attention-all-we-need-in-finance-part-i/

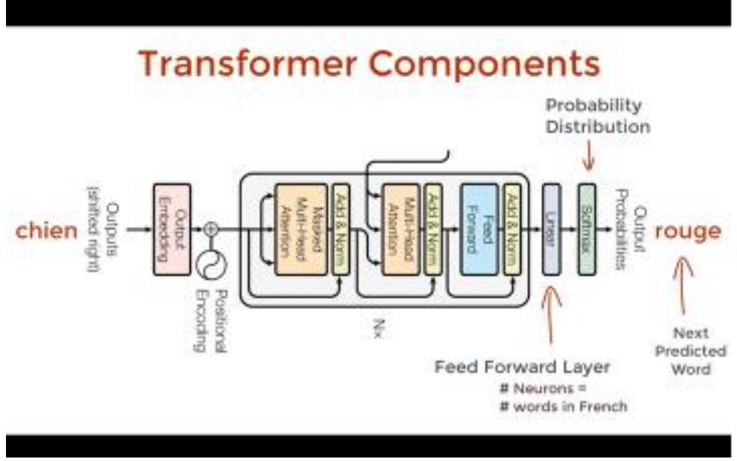


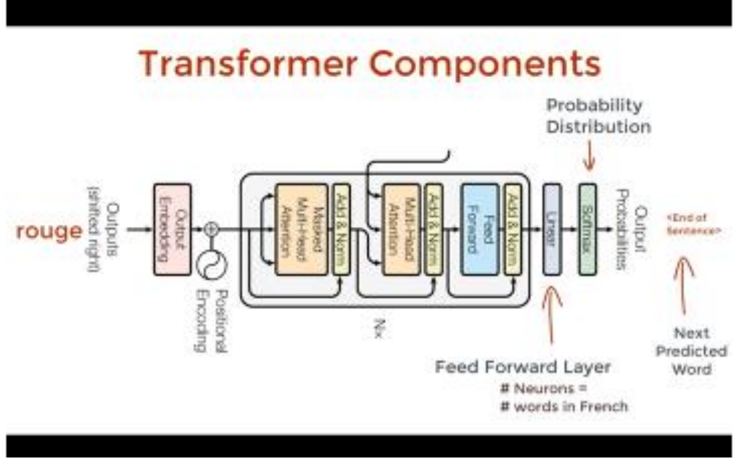
Encoding

Decoding


Transformer Components
Probability Distribution
Le
Outputs (shifted right)
Output Embedding
Positional Encoding
Masked Multi-Head Attention
Add & Norm
Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Nx
Linear
Softmax
Output Probabilities
gros
Next Predicted Word
Feed Forward Layer
# Neurons =
# words in French


Transformer Components
Probability Distribution
gros
Outputs (shifted right)
Output Embedding
Positional Encoding
Masked Multi-Head Attention
Add & Norm
Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Nx
Linear
Softmax
Output Probabilities
chien
Next Predicted Word
Feed Forward Layer
# Neurons =
# words in French


Transformer Components
Probability Distribution
chien
Outputs (shifted right)
Output Embedding
Positional Encoding
Masked Multi-Head Attention
Add & Norm
Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Nx
Linear
Softmax
Output Probabilities
rouge
Next Predicted Word
Feed Forward Layer
# Neurons =
# words in French


Transformer Components
Probability Distribution
rouge
Outputs (shifted right)
Output Embedding
Positional Encoding
Masked Multi-Head Attention
Add & Norm
Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Nx
Linear
Softmax
Output Probabilities
<End of Sentence>
Next Predicted Word
Feed Forward Layer
# Neurons =
# words in French