Transformers Language Model

Dhruv Makwana

What is Transformer?

- Transformer is a type of neural network architecture used for natural language processing (NLP) tasks such as language translation, question-answering, and text summarization.

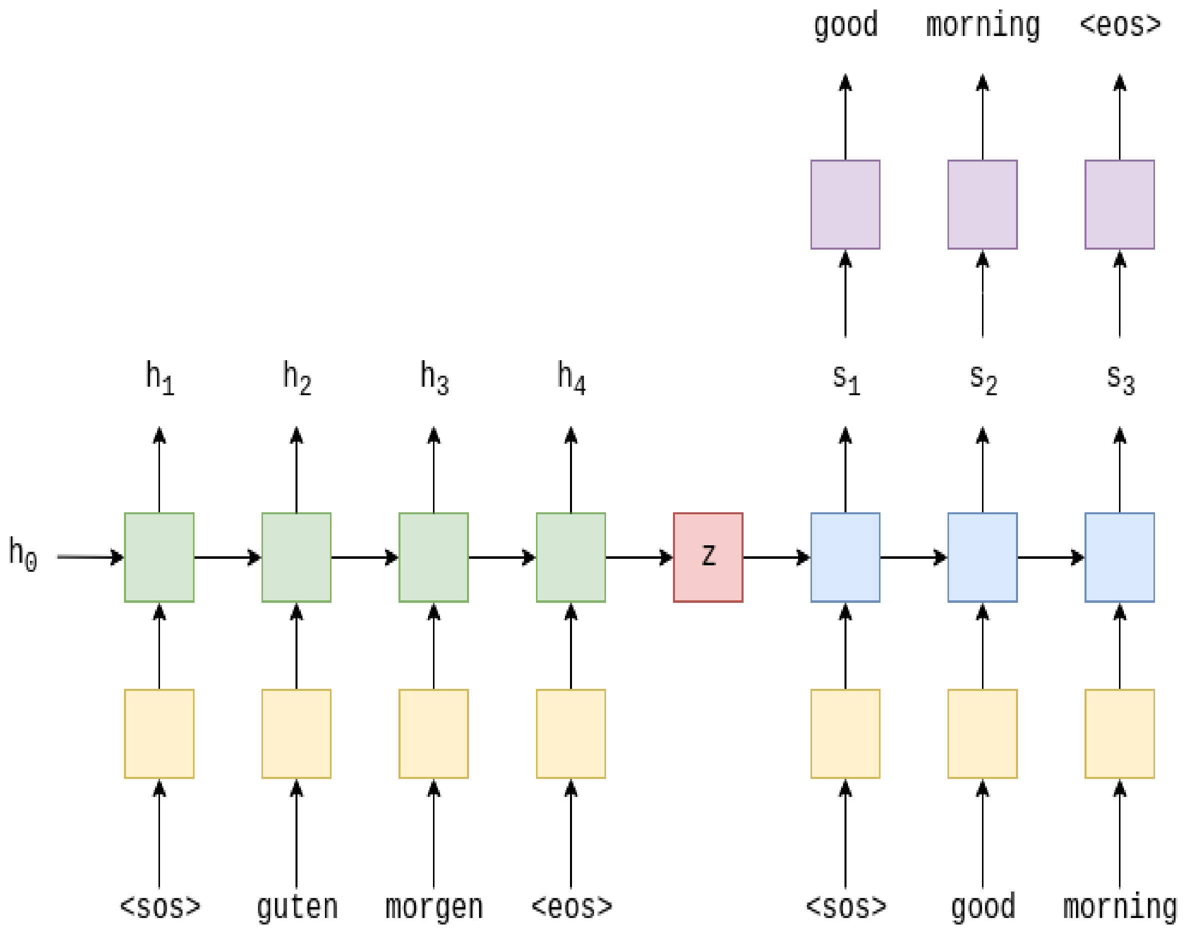
- The Transformer model was introduced in 2017 in a paper called "Attention is All You Need" by Vaswani et al.

- It is based on a self-attention mechanism that allows the model to attend to different parts of the input sequence and capture long-range dependencies more efficiently.

Paper: https://arxiv.org/abs/1706.03762

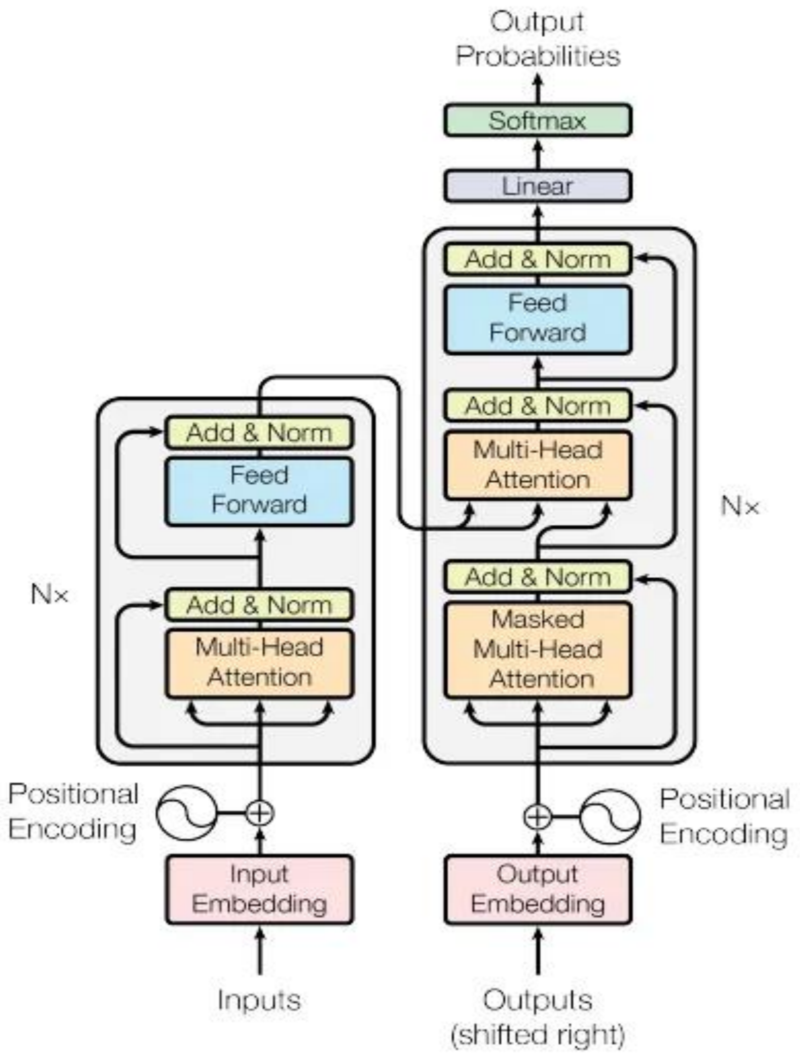
Why learn Transformer?

- Initial neural machine translation approaches used RNNs in an encoder-decoder design for sequence-to-sequence challenges.
- Their ability to retain information from the first elements was lost when new elements were incorporated into the sequence.
- The decoder will lose information about the sequence's earliest components if it only accesses the final concealed state.



Why learn Transformer?

- The Transformer model extract features for each word using a self-attention mechanism.
- Self attention is used to figure out how important all the other words in the sentence are w.r.t. to the current word.
- There is encoder in left side and decoder in right side. Both contains the attention



Transformer

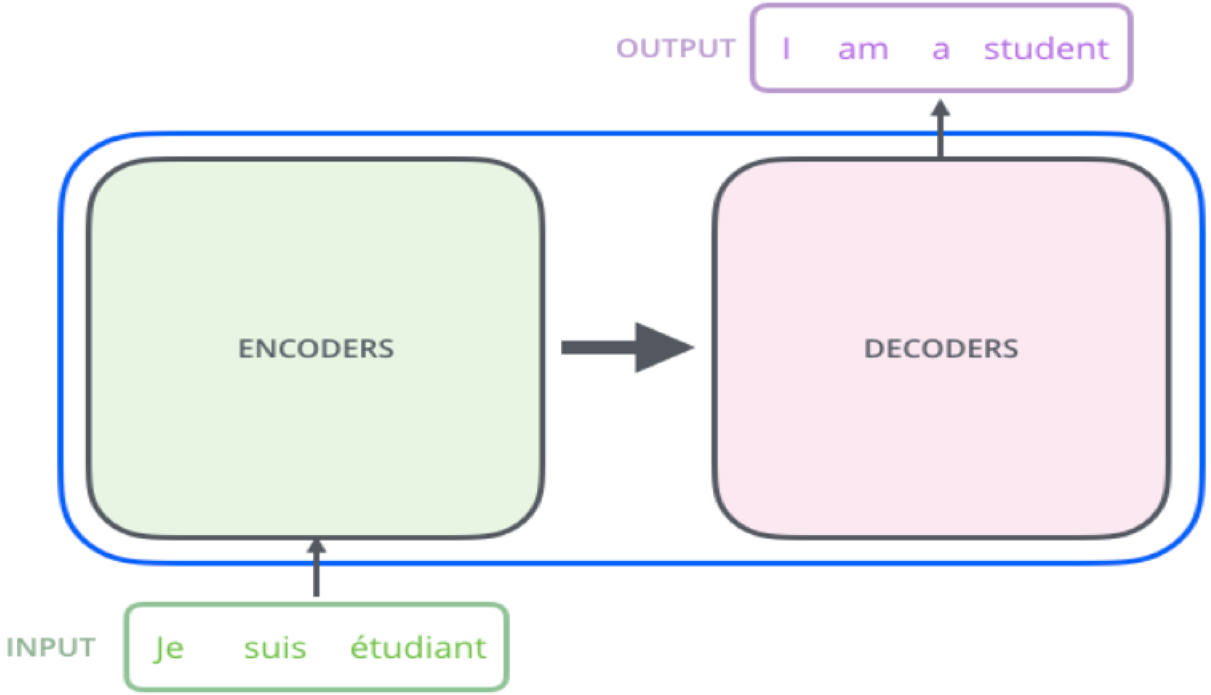
Higher level look

- It takes sentence in 1 language and returns in another language

Transformer

Higher level look

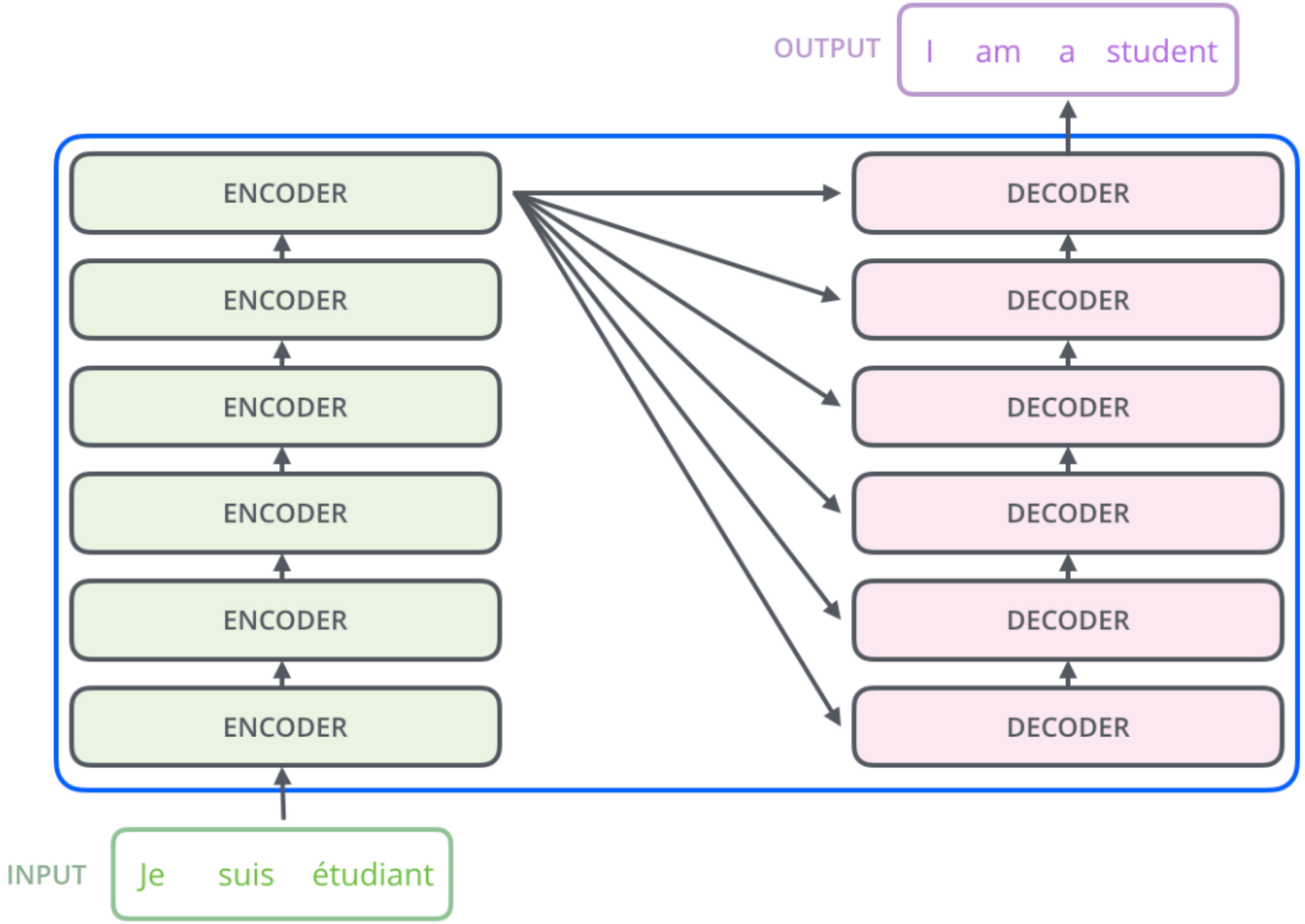
- Transformer contains set of encoders, set of decoders, and connection between them.



Transformer Deep Down

Higher level look

- Encoding part contains set of Encoders.
- Decoding part contains same number of set of Decoders.
- Paper uses 6 of them on encoding and decoding side.



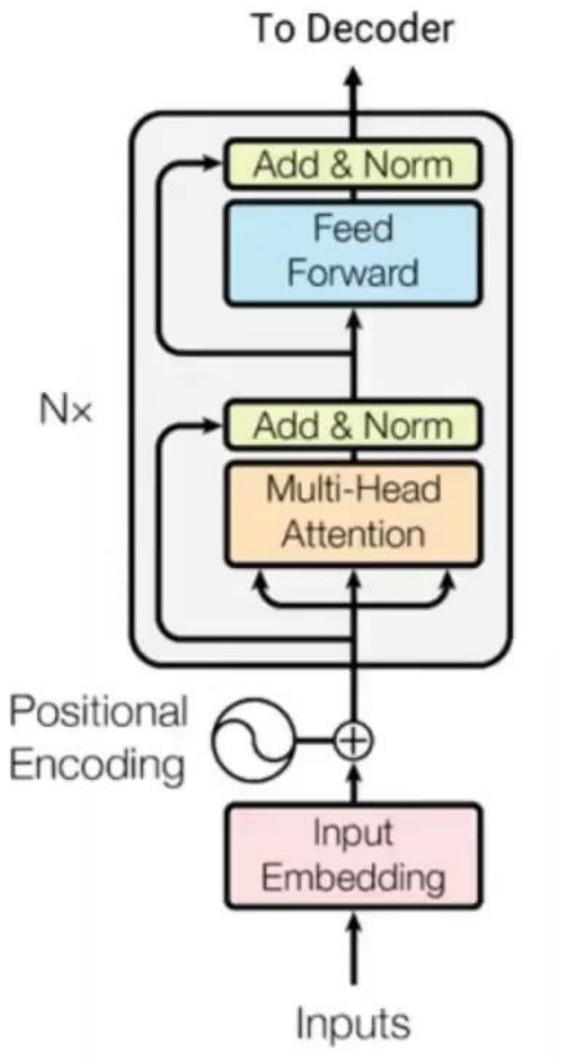
Transformer Deep Down

Encoder

- Encoder consists of two main components.

(1). **Multi-Head Attention:** It is a layer that helps the encoder look at other words in the input sentence as it encodes a specific word.

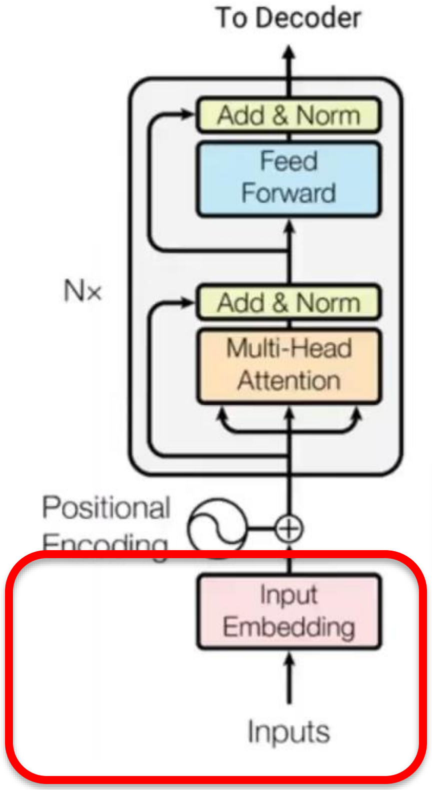
(2). **Feed-Forward Neural Network**: It is just like a simple Artificial Neural Network. The output of Multi-Head attention is passed to Feed Forward network as input.



Transformer Deep Down

Inputs

- Inputs are series of words.
- We turn each word into vector using an embedding algorithm.
- Simple approach is to assign different number to each word. However does not preserve similarities.
- These vectors are constructed in such a way that words with similar meanings have vectors that are closer to each other in the vector space.



Transformer Deep Down

Word Embeddings

- Word embeddings are usually learned from a large corpus of text data using techniques such as neural networks.
- During training, the model learns to predict the context words of a target word based on its position in the text.
- The weights of the hidden layer in the model are then used as the word embeddings.

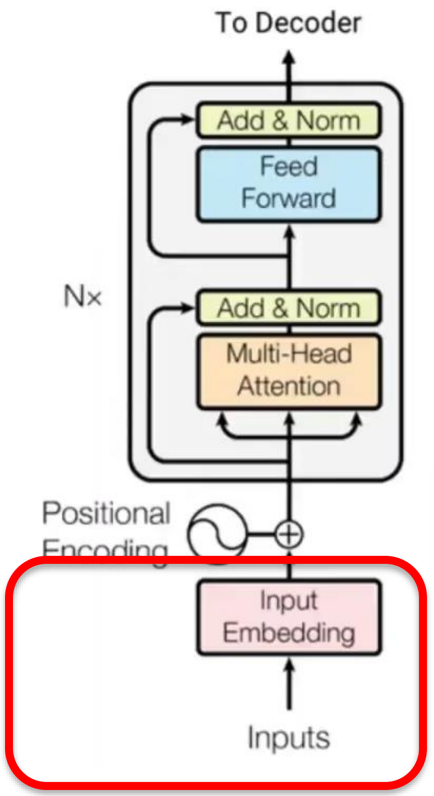
References:
https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca
https://youtu.be/gQddtTdmG_8

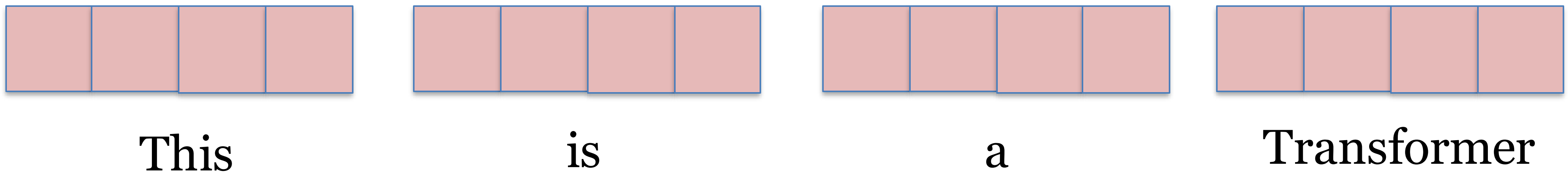
Transformer Deep Down

Inputs

Abstraction: Encoder receives a list of vectors each of size 512.

- Bottom most encoder takes word embedding as input.
- Other encoders takes output of the previous encoder.
- 512 dimension is hyperparameter, we can set.

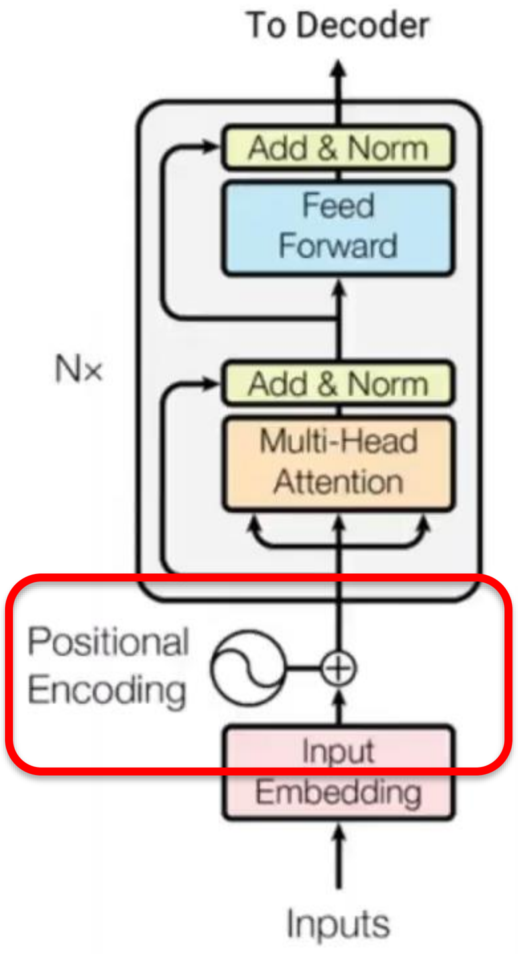




Transformer Deep Down

Positional Encoding

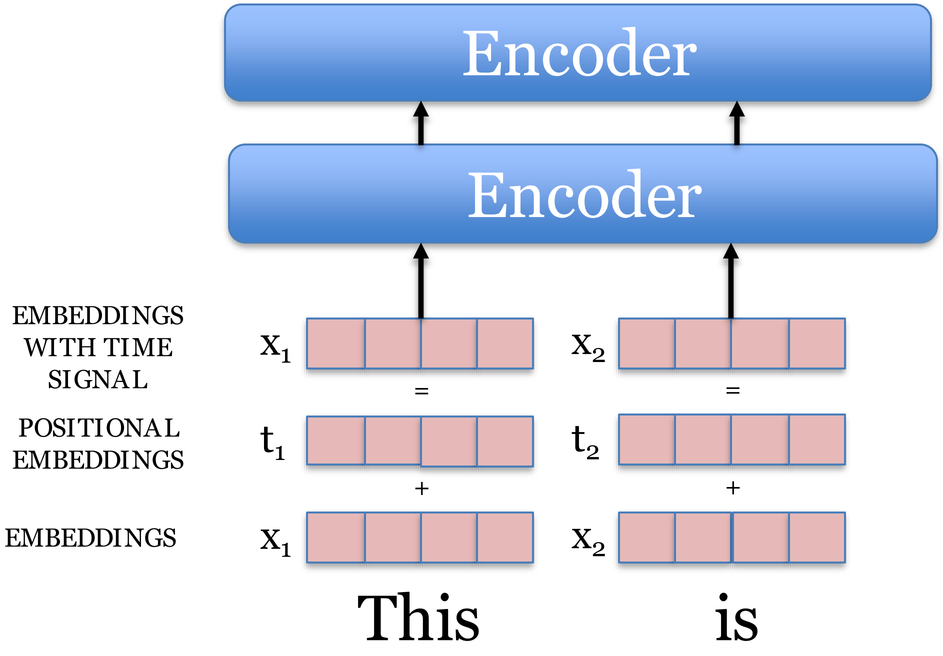
- Recurrent Neural Networks (RNNs) parse a sentence word by word in a sequential manner.
- Transformer solely relies on self-attention mechanism.
- As each word in a sentence simultaneously flows through the Transformer's encoder and decoder stack, The model itself doesn't have any sense of order for each word.



Transformer Deep Down

Positional Encoding

- Positional information is added by using the sum of current embedding vector with a new vector that contains the information on position of each word.
- We add 512 dimension input vector with 512 dimension positional encoding vector.



Transformer Deep Down

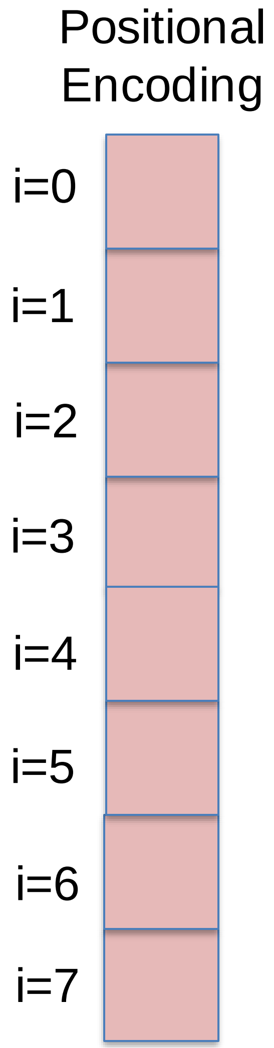
Positional Encoding

- To generate the positional encoding vector, sinusoidal function is used.
- There are two formula, 1 for even and 1 for odd dimensions.

Even dimension

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

Odd dimension



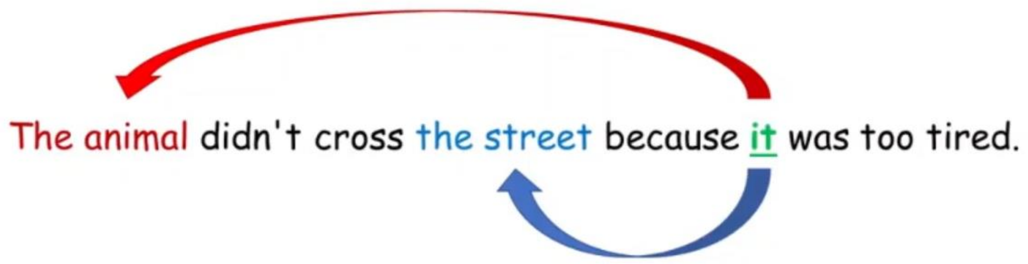
References: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Transformer Deep Down

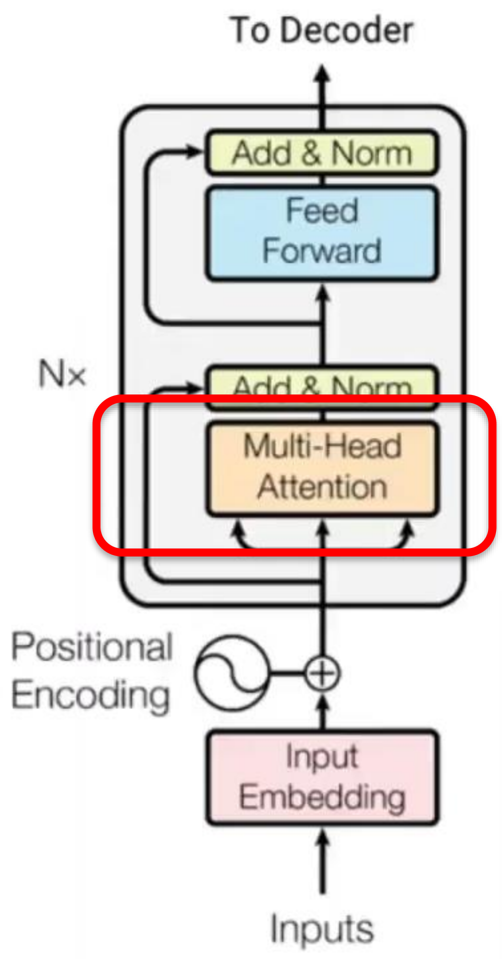
Multi-Head Attention

We need to understand self-attention before getting multi-head attention.

(Q). What does the word “it” refers in this sentence?



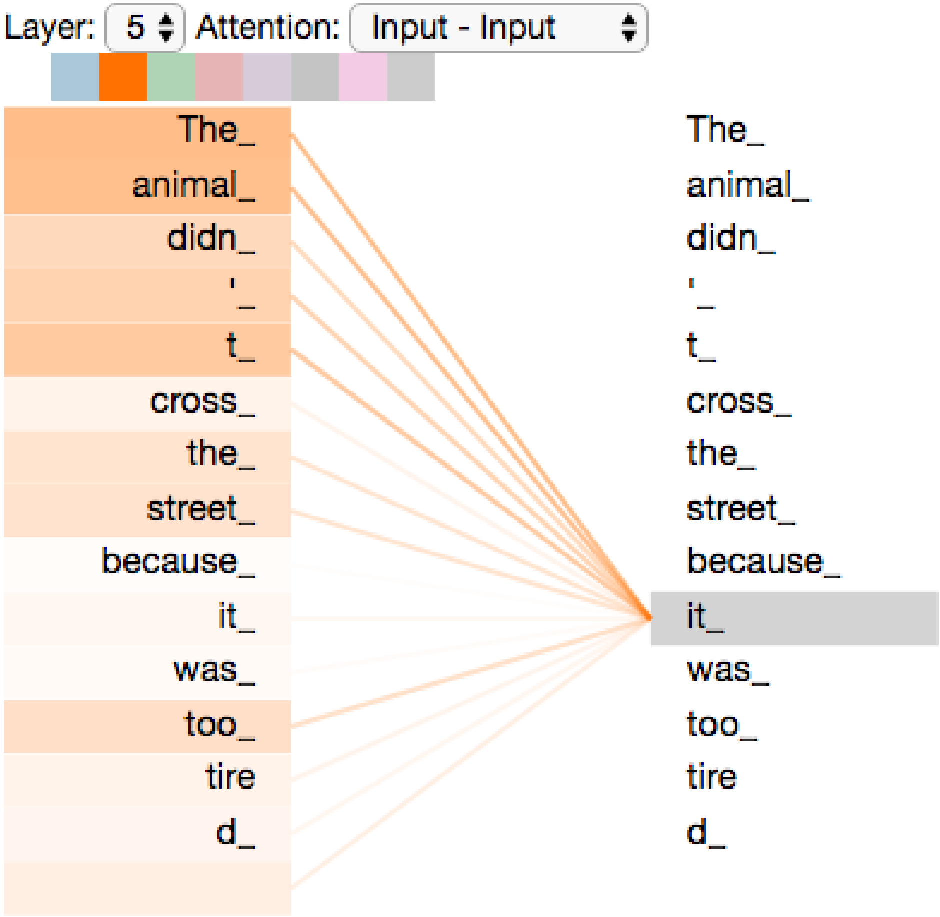
(A). self-attention allows the model to associate the word “it” with the word “animal”.



Transformer Deep Down

Self Attention

- Self-attention assist computers in comprehending these details about sentence.
- Its similar to how RNNs maintain hidden-states which allows an RNN to incorporate its representation of previous words/vectors.
- As seen from figure, when we are encoding word "it", part of attention process was focussed on the work "Animal"



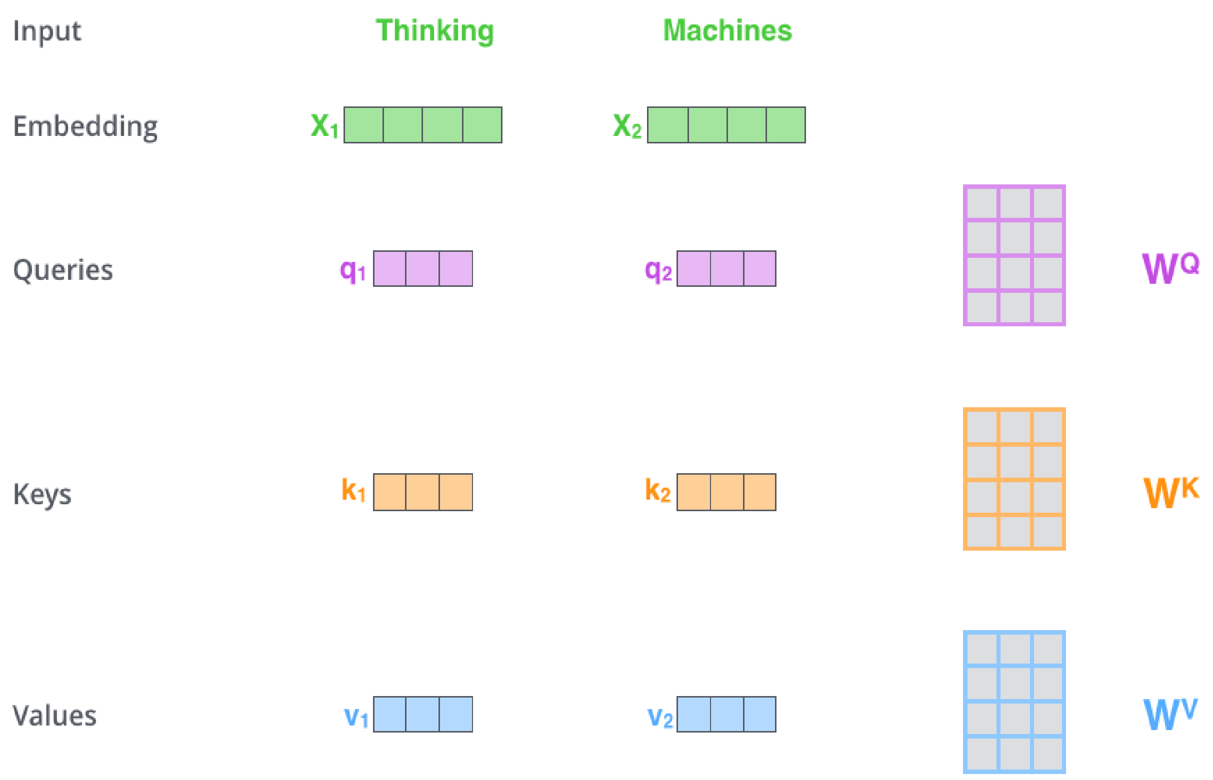
ignitarium Spark On.

Transformer Deep Down

How self-attention works?

Lets look at process of encoding single input embedding vector for the sake of simplicity.

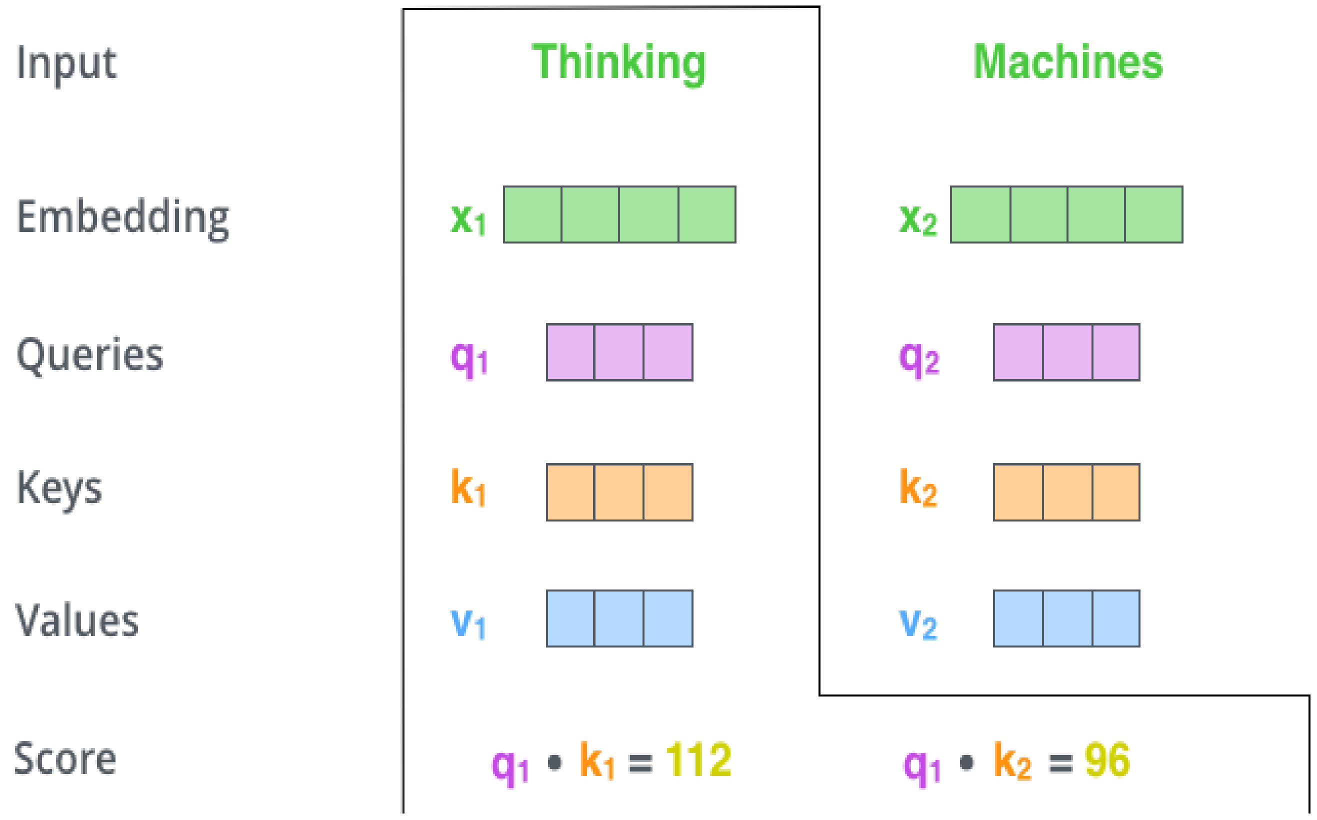
- The **first step** is to create three copies of input embedding.
- We multiply each input vector with a weight matrix which is learned through the training process.
- Output of these multiplication will be called as "Query", "Key", and "Value".
- Their dimensionality is 64, while embedding dimensionality is 512.



Transformer Deep Down

Self-Attention

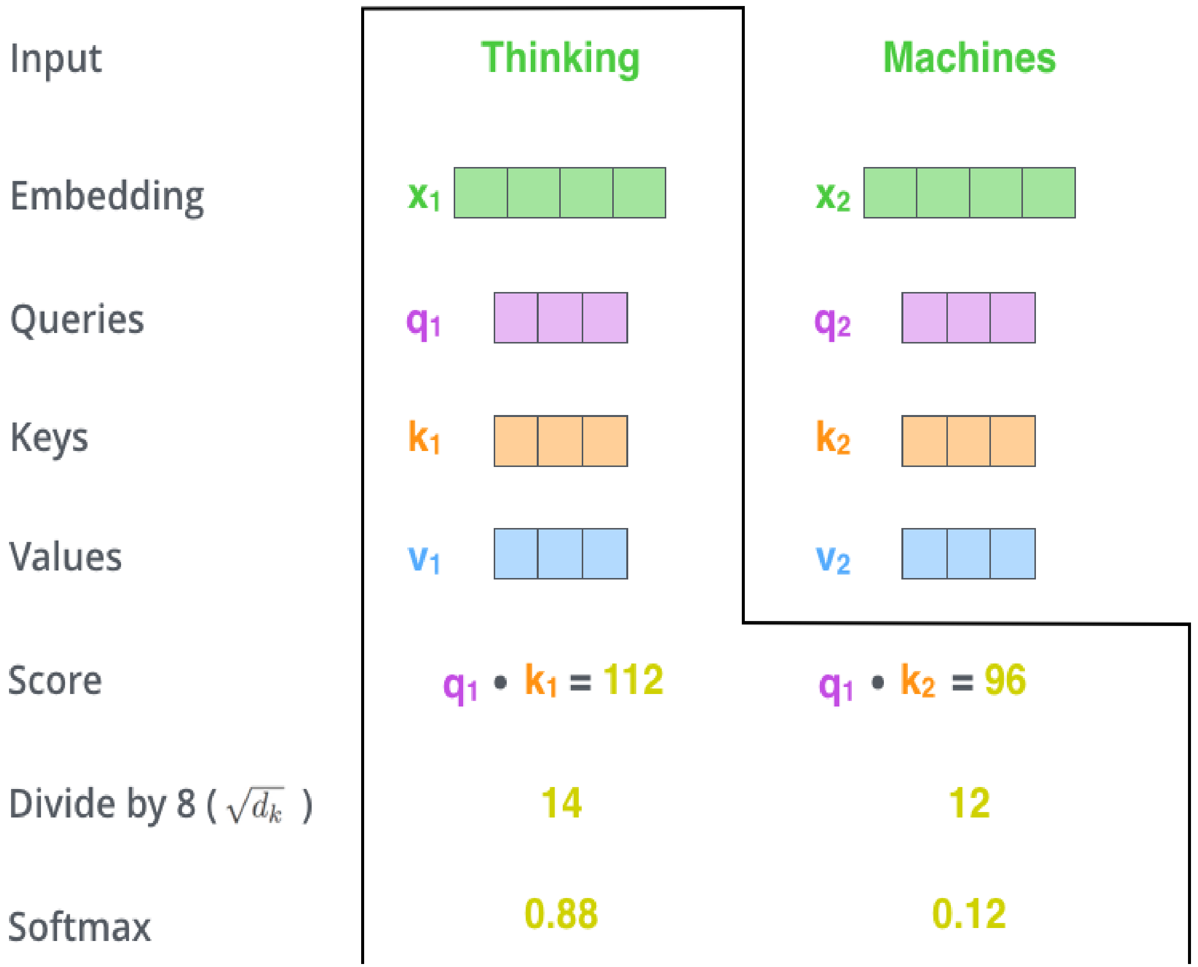
- The **second step** in calculating self-attention is to calculate a score.
- Consider we are calculating the self-attention for the first word "Thinking".
- We need to score each word in input sequence against this word.
- The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring.
- The score determines how much focus to place on other parts of the input sentence.



Transformer Deep Down

Self-Attention

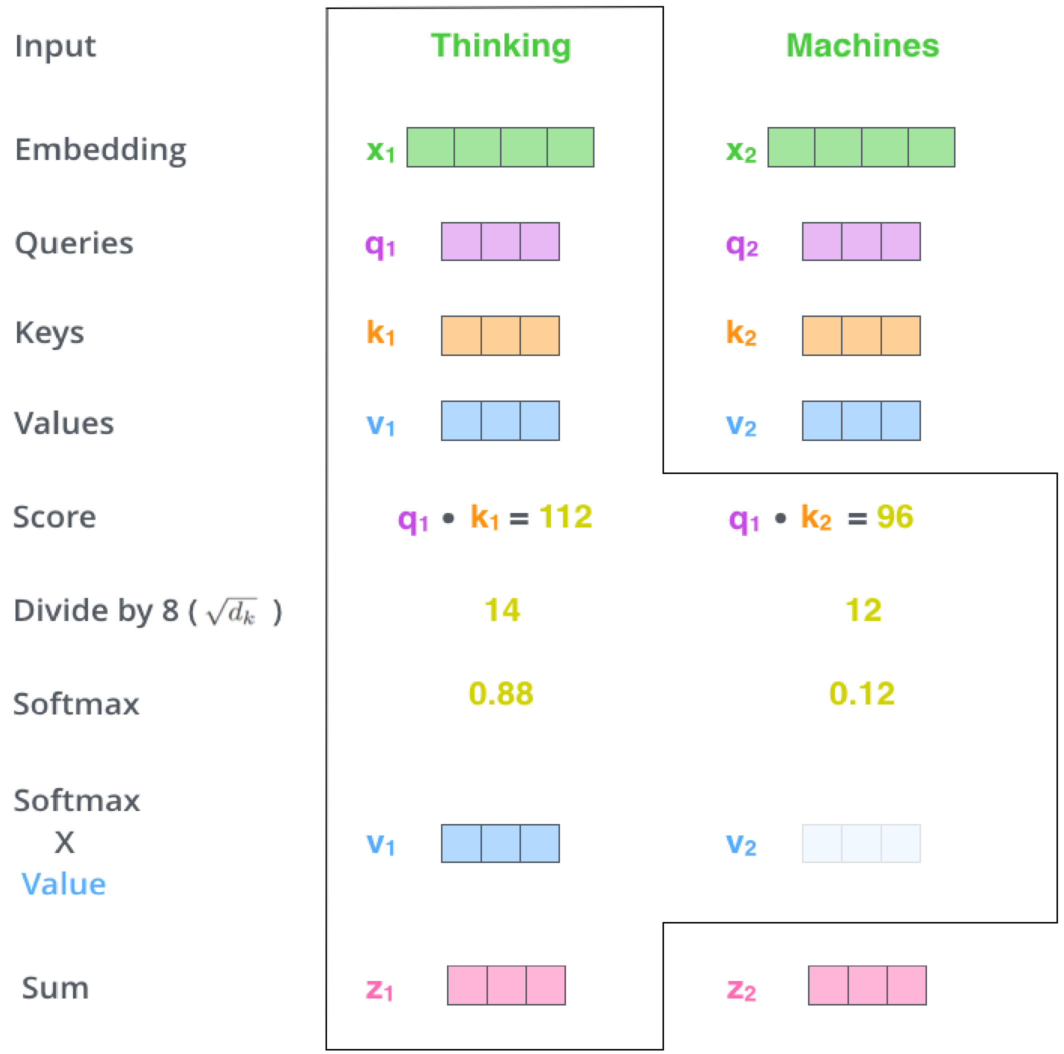
- The **third step** is to divide the scores by 8 (the square root of the dimension of the key vectors (64)).
- This step is performed for getting the more stable gradients.
- The **Fourth step** is to perform softmax on score, which normalizes the value between 0 and 1.



Transformer Deep Down

Self-Attention

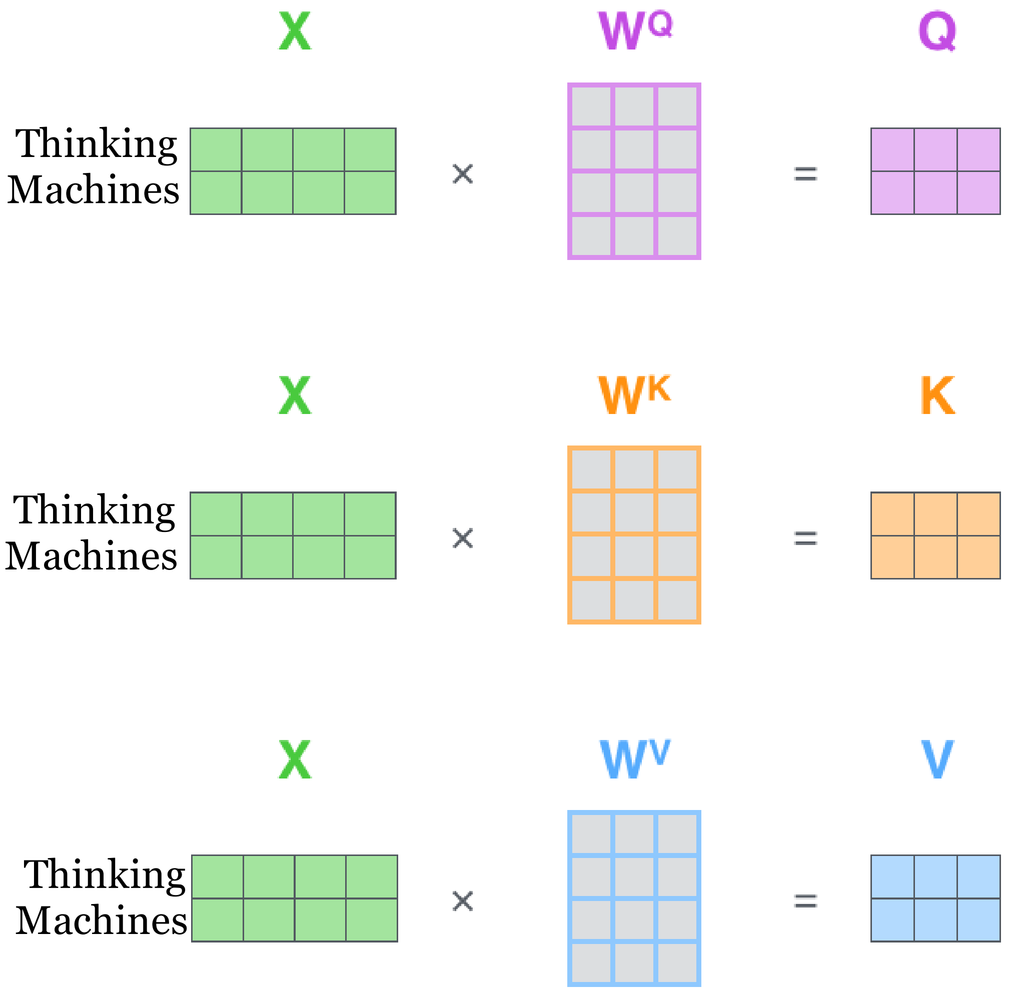
- The **fifth step** is to multiply each value vector by the softmax score.
- The intuition here is to keep intact the values of the word(s) we want to focus on and drown-out irrelevant words (by multiplying them by tiny numbers like 0.001, for example).
- The **sixth step** is to sum up the weighted value vectors.
- This produces the output of the self-attention layer at this position.



Transformer Deep Down

Self-Attention

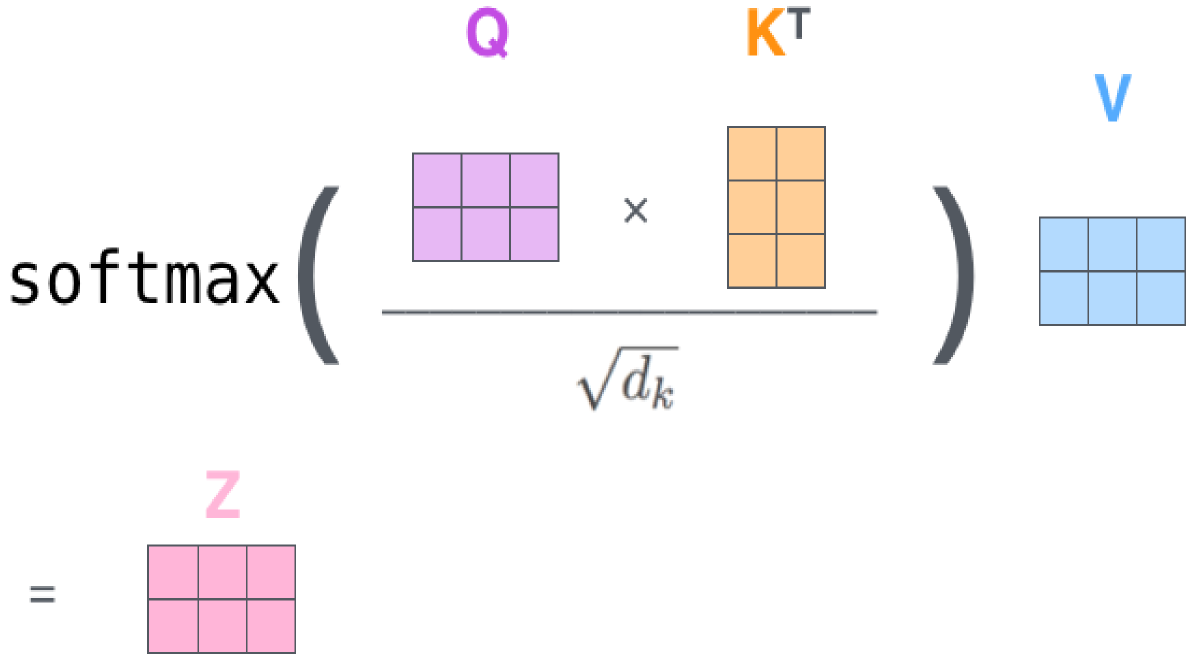
- In the actual implementation, this calculation is done in matrix form for faster processing.
- The right side diagram shows the same calculation in matrix format.
- The **first step** is to calculate the Query, Key, and Value matrices.



Transformer Deep Down

Self-Attention

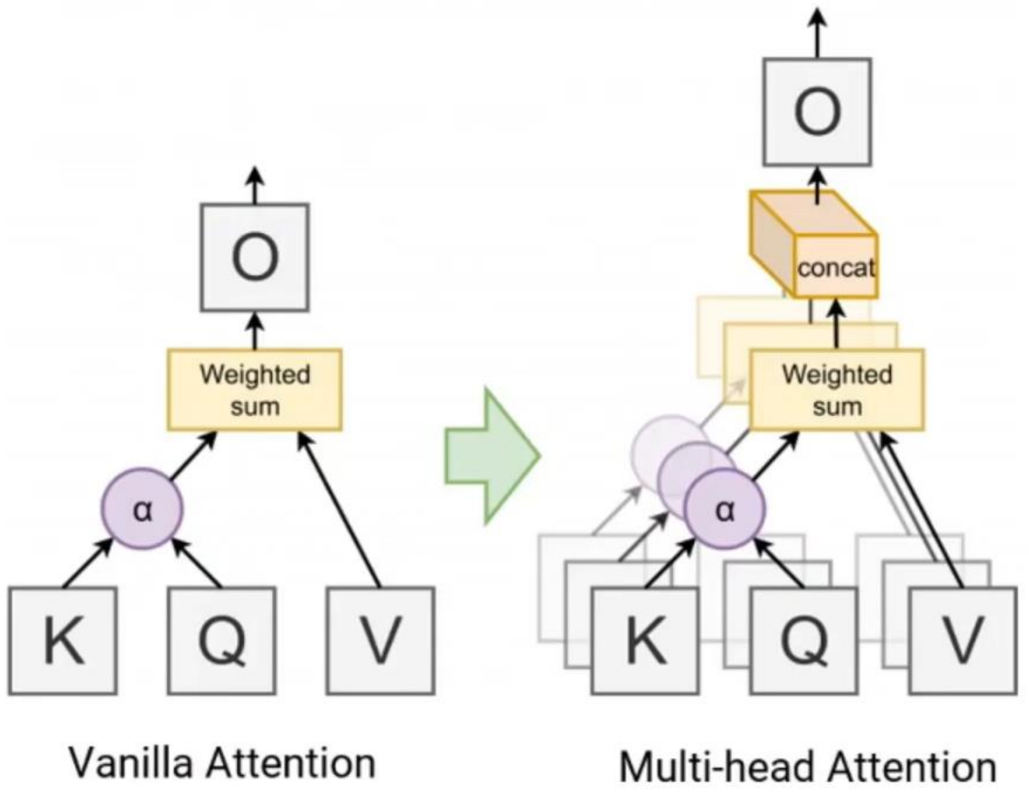
- The steps two to six can be performed using the right side showed formula.
- The entire process of calculating the output will be same as before, with the exception of using matrices instead of vectors.



Transformer Deep Down

Multi-Head Attention

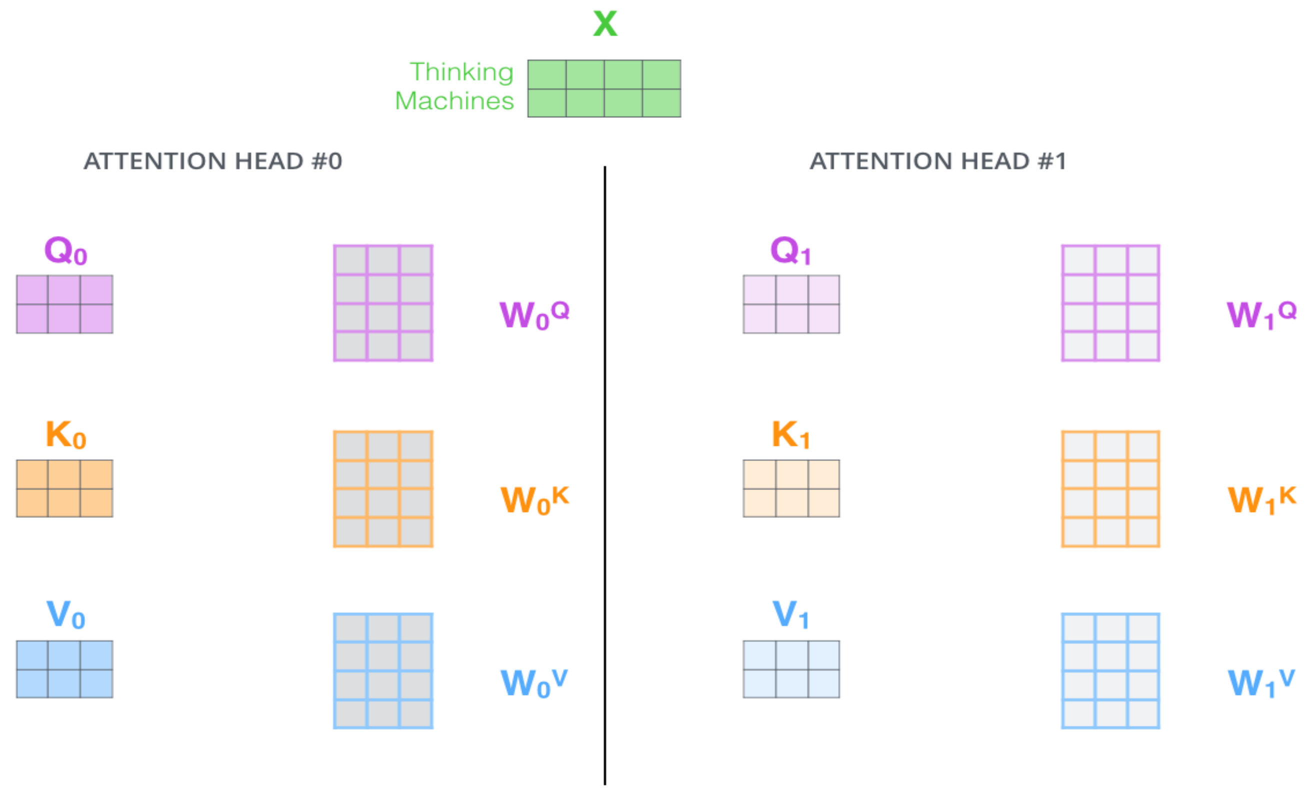
- Instead of performing a single attention with ‘d’ dimensional keys, values, and queries, the author found it beneficial to linearly project the queries, keys, and values 8 times with different projections.
- It expands the model’s ability to focus on different positions.



Transformer Deep Down

Multi-Head Attention

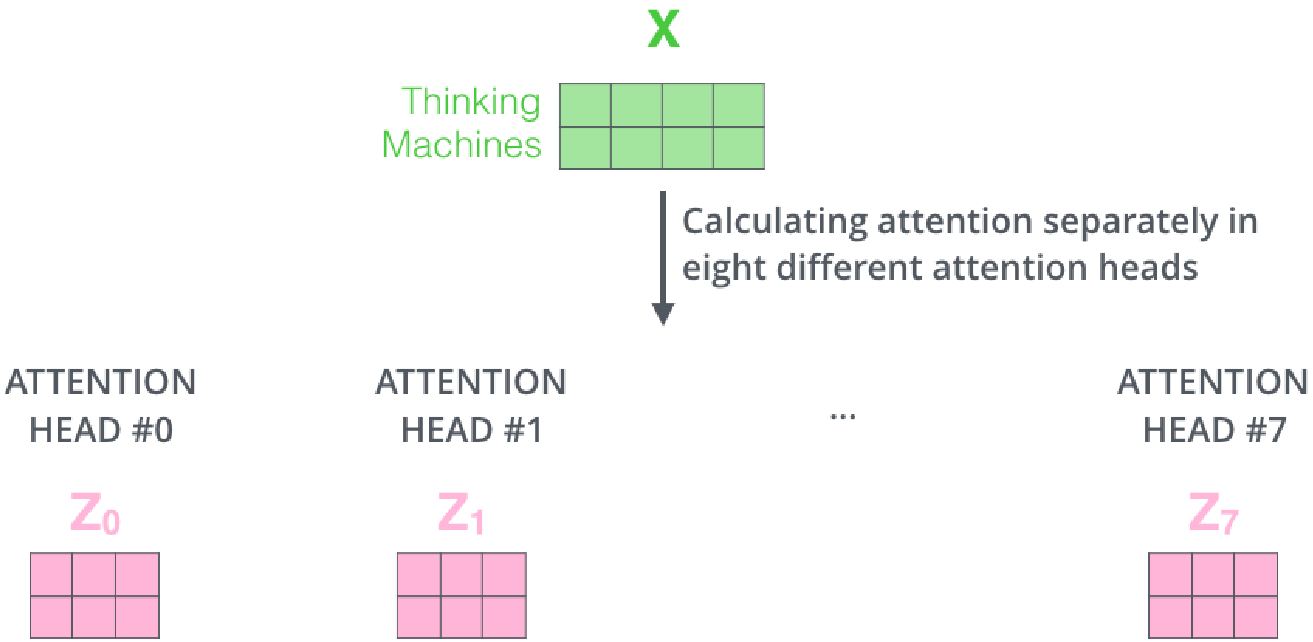
- We maintain separate query, key, and value weight matrices for each head, resulting in different query, key, and value matrices.
- As done before, we multiply embedding X, with W_0Q to obtain Q_0. Similarly we obtain W_0K, and W_0V by multiplying embedding X with K_0 and V_0 respectively.



Transformer Deep Down

Multi-Head Attention

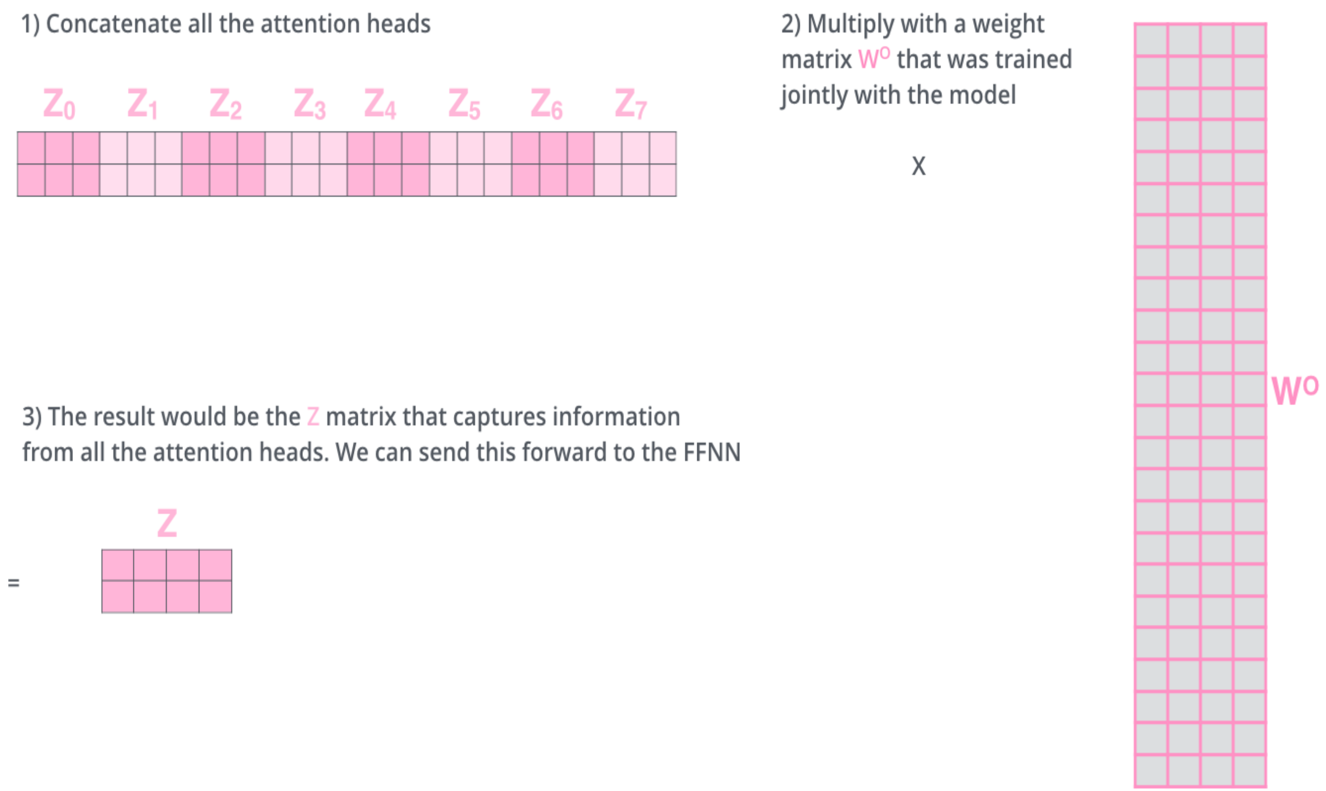
- On each of these projected version of queries, keys and values, we then perform the attention function parallel for eight times.
- After performing attention for eight times, we end up with eight different Z matrices.
- We need a way to combine these eight matrices into a single matrix.



Transformer Deep Down

Multi-Head Attention

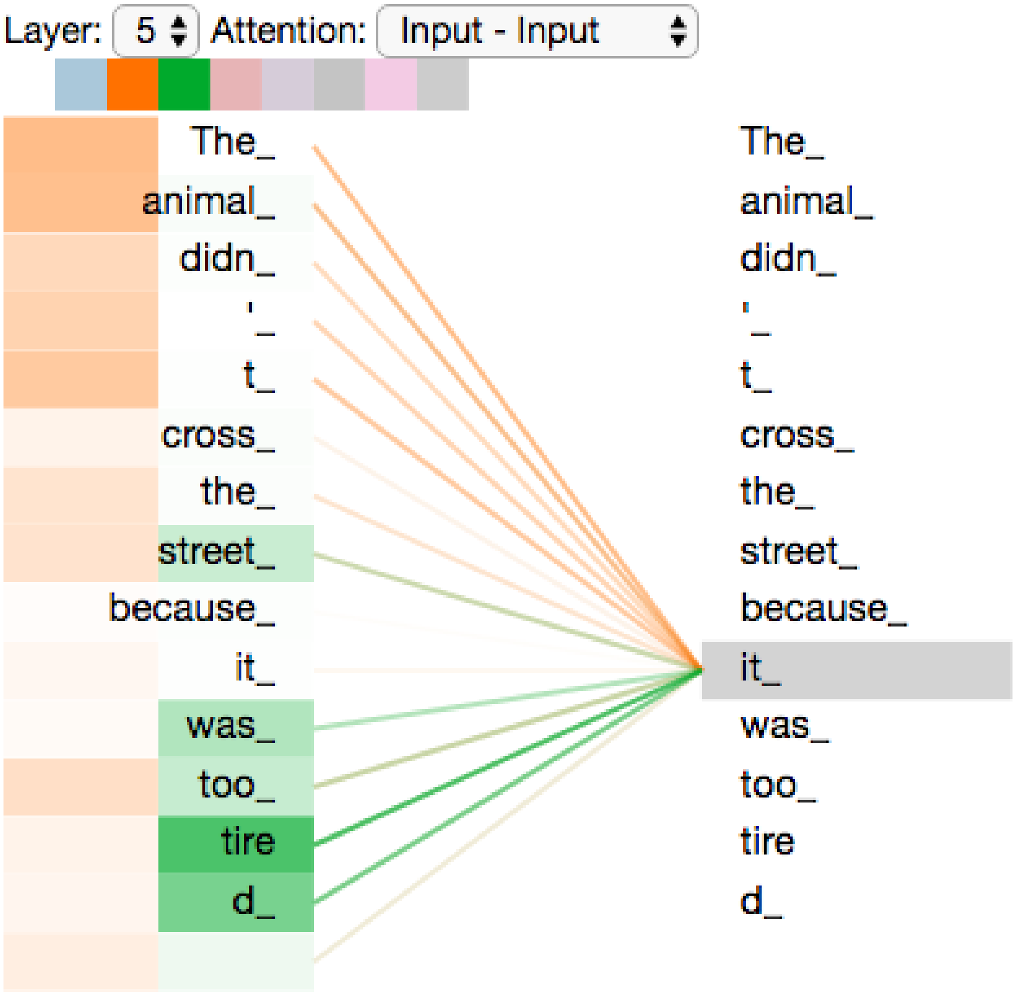
- We concatenate all the attention heads into single matrix.
- We multiply them with single weight matrix (shown by W^o in figure). This weight matrix is also learned in training phase.
- This results in single Z matrix which contains information about the attention.



Transformer Deep Down

Multi-Head Attention

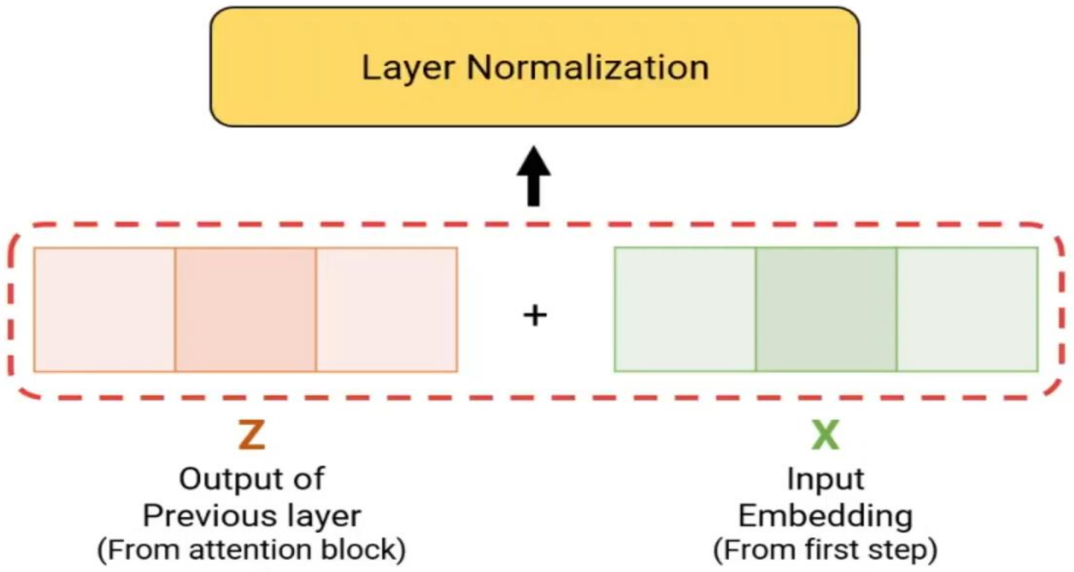
- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired“.
- In a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

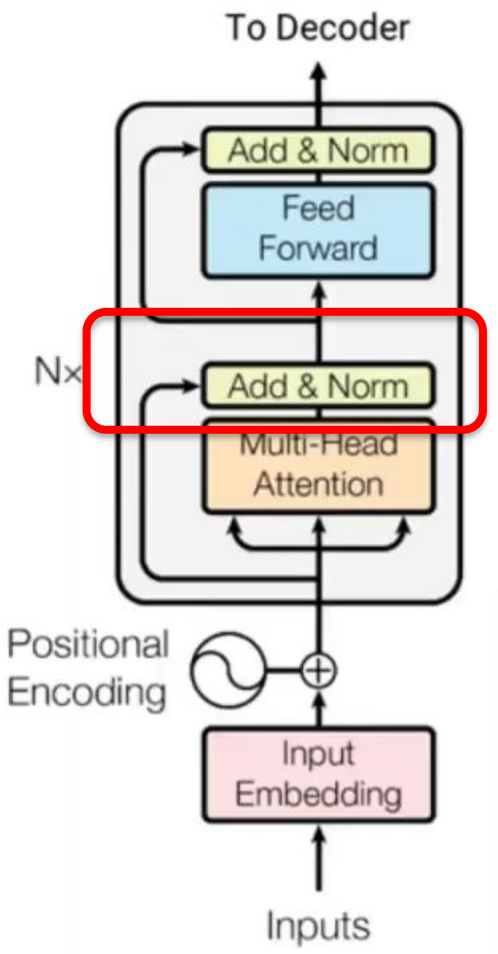


Transformer Deep Down

Add and Normalize

- Attention is followed by add and normalize layer.
- We first calculate the sum of output vector of attention block and the input embedding vector.
- Output of addition is passed through Layer Normalization layer.

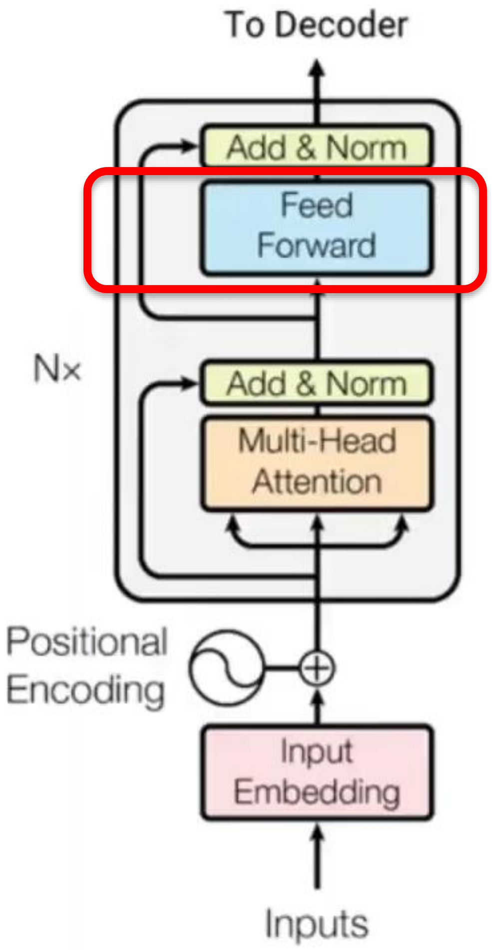




Transformer Deep Down

Feed Forward

- Feed Forward consists of two linear transformation with a ReLU activation in between.
- It is performed to process the output from one attention layer in a way to better fit the input for the next attention layer.

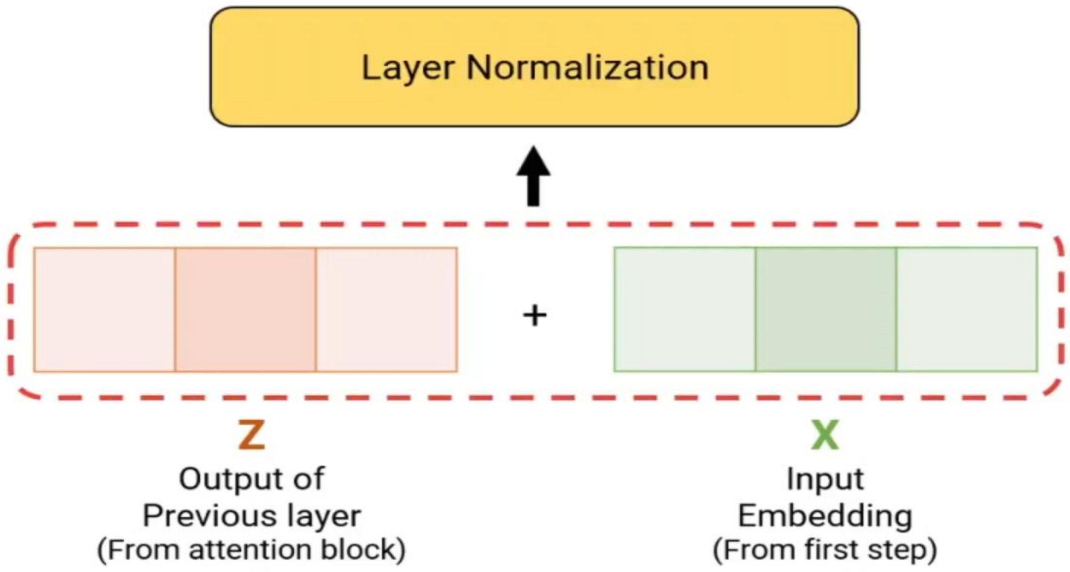


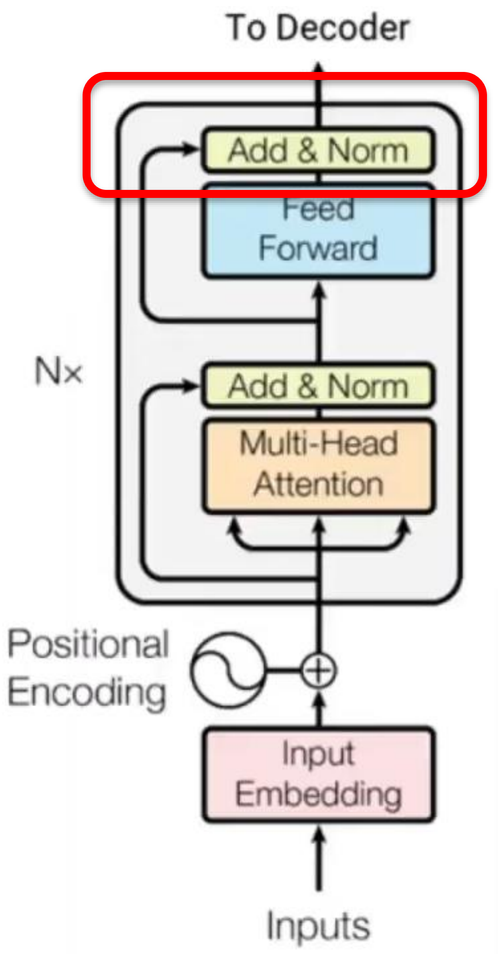
Reference: https://arxiv.org/abs/2012.14913

Transformer Deep Down

Add and Normalize

- Attention is followed by add and normalize layer.
- We first calculate the sum of output vector of attention block and the input embedding vector.
- Output of addition is passed through Layer Normalization layer.

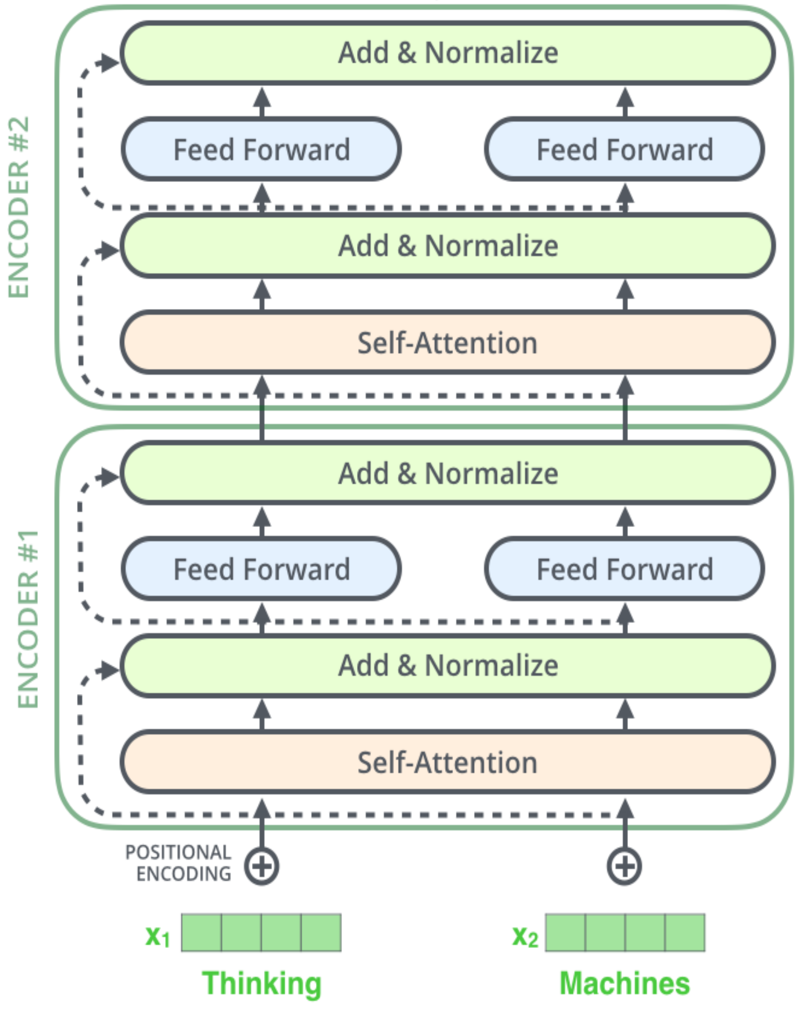




Transformer Deep Down

Encoder Summary

- Input is converted into embeddings and added with the positional embedding.
- Input with order information is passed through self-Attention.
- Residual is performed between self attention output and input embedding.
- Output of residual is passed through feed forward layers.
- Residual is performed between feed forward output and input embedding.
- Residual output is fed to next encoder.

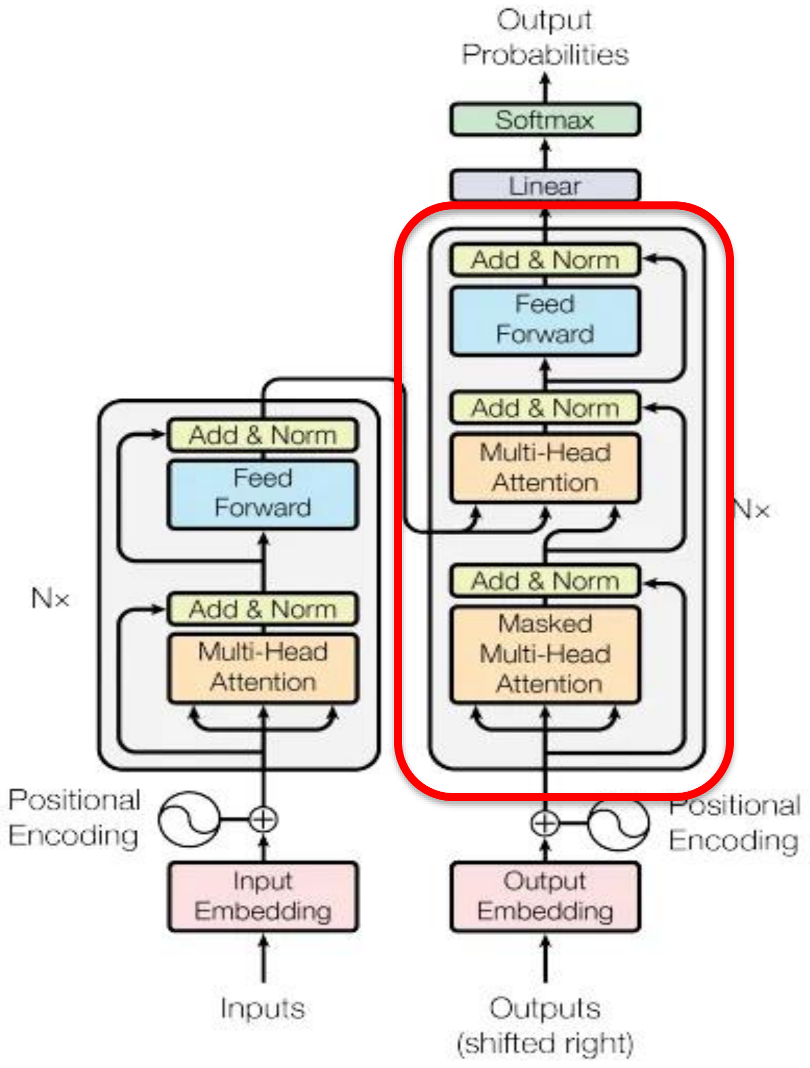


Transformer Deep Down

DECODER

The Decoder is used to determine the output sequence’s tokens one at a time by using:

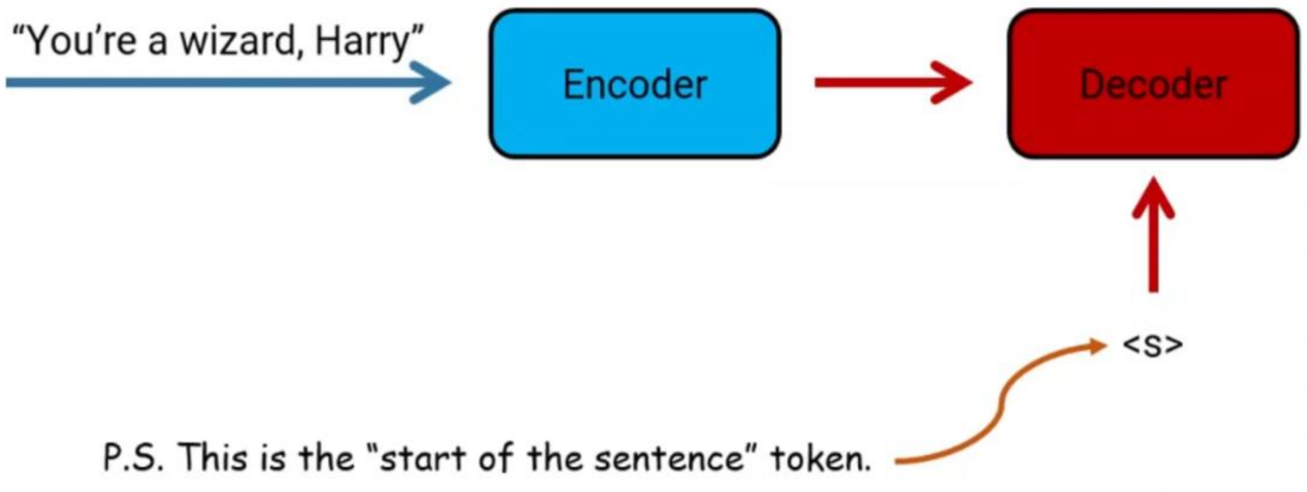
- Attention known for all tokens for from the Encoder.
- All predicted tokens of output sequence so far.
- Once a new token is predicted, it is considered to determine the next token.
- The Decoder works in two modes namely Train mode, and Test mode.



Transformer Deep Down

DECODER at test phase

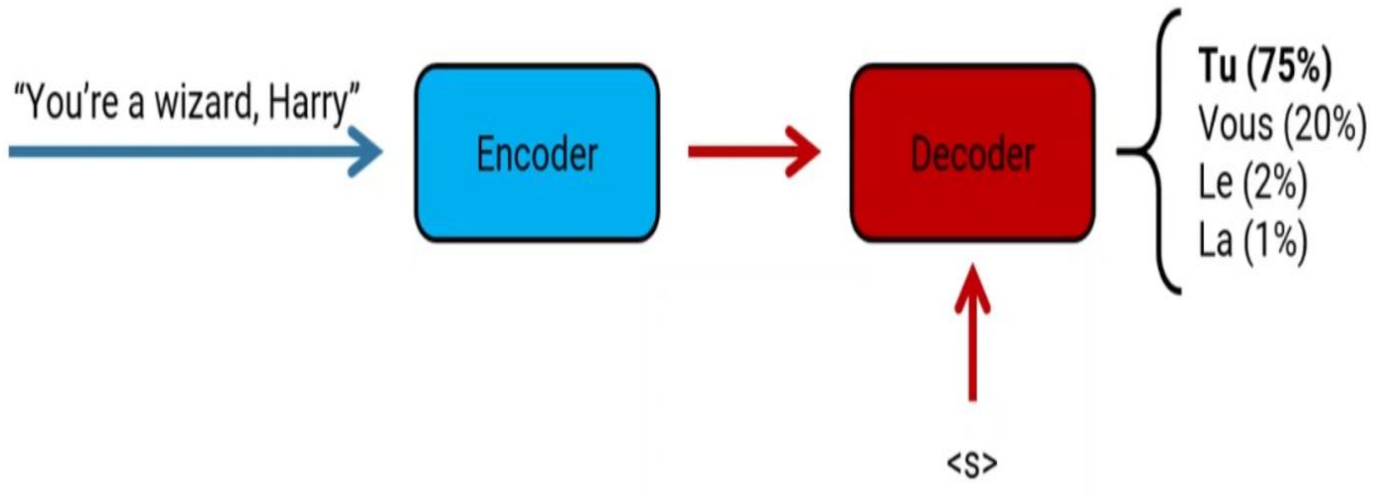
- Considering we are converting English sentence “You’re a wizard, Harry” into French sentence.
- We feed entire sentence into Encoder at once.
- Encoder’s output and special token (used for indicating start of the sentence) is fed to Decoder to generate the First word.



Transformer Deep Down

DECODER at test phase

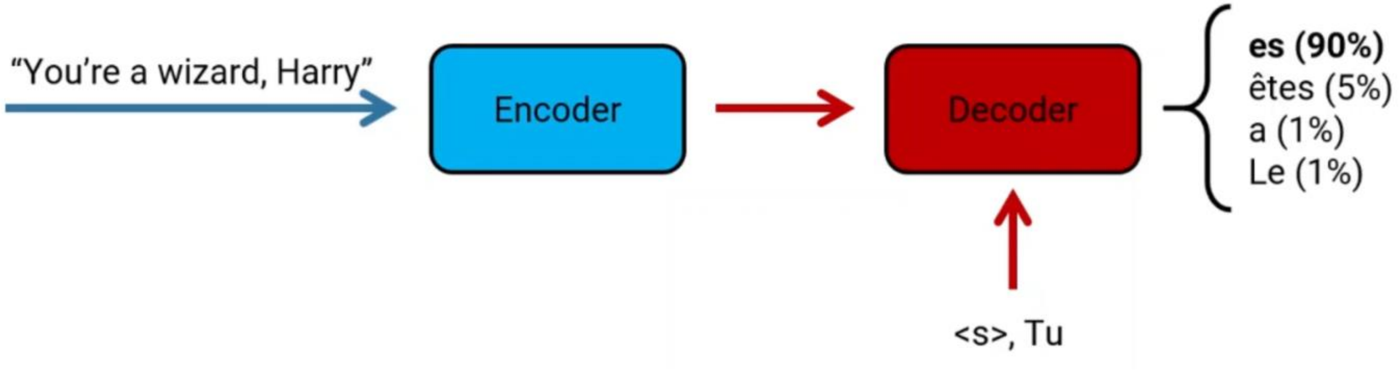
- Considering we are converting English sentence “You’re a wizard, Harry” into French sentence.
- We feed entire sentence into Encoder at once.
- Encoder’s output and special token <s> which is used for indicating start of the sentence is fed to Decoder to generate the First word.



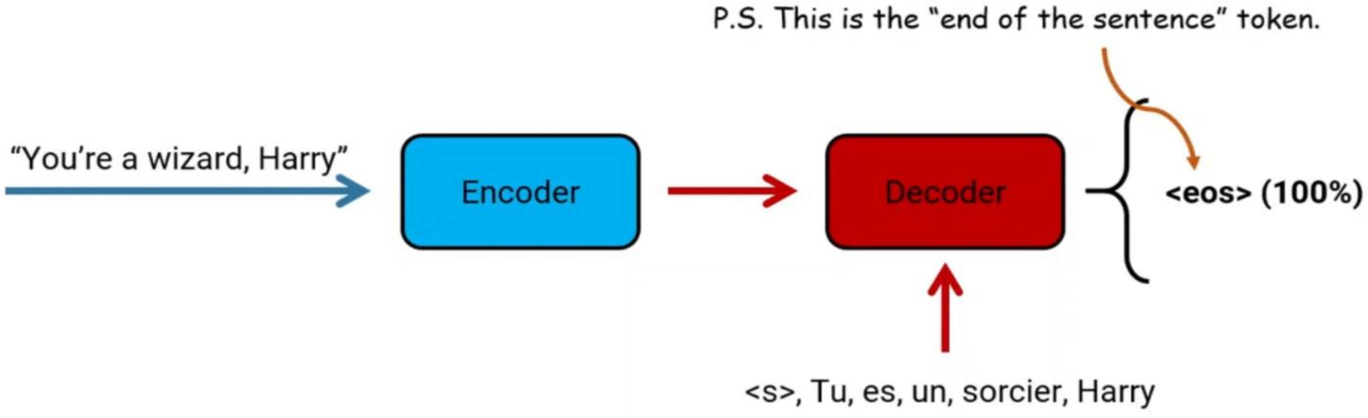
Transformer Deep Down

DECODER at test phase

- For next word prediction, decoder takes Encoder's output as well as the First word predicted with starting token(<s>).



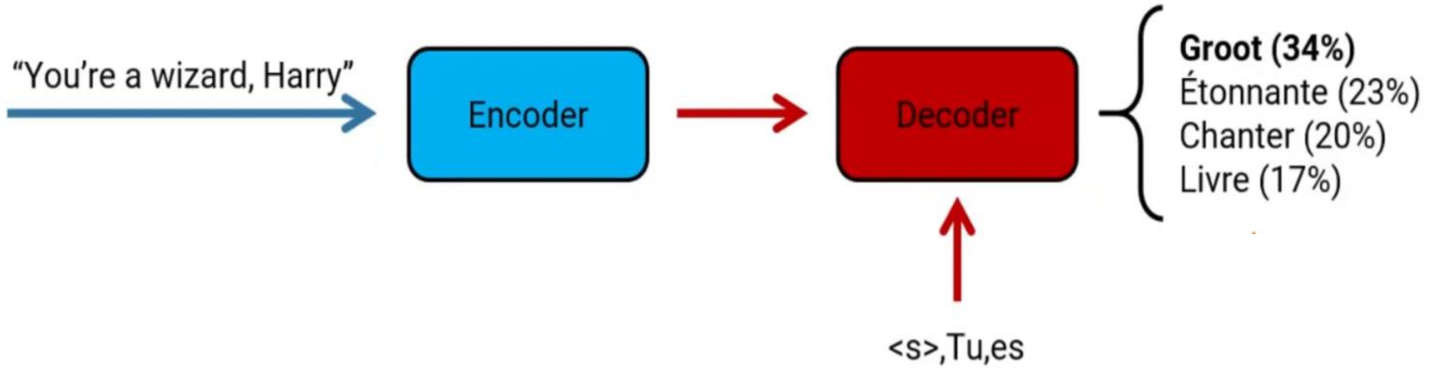
- This process continues until decoder predicts the special token end of the sentence <eos>.



Transformer Deep Down

DECODER at train phase

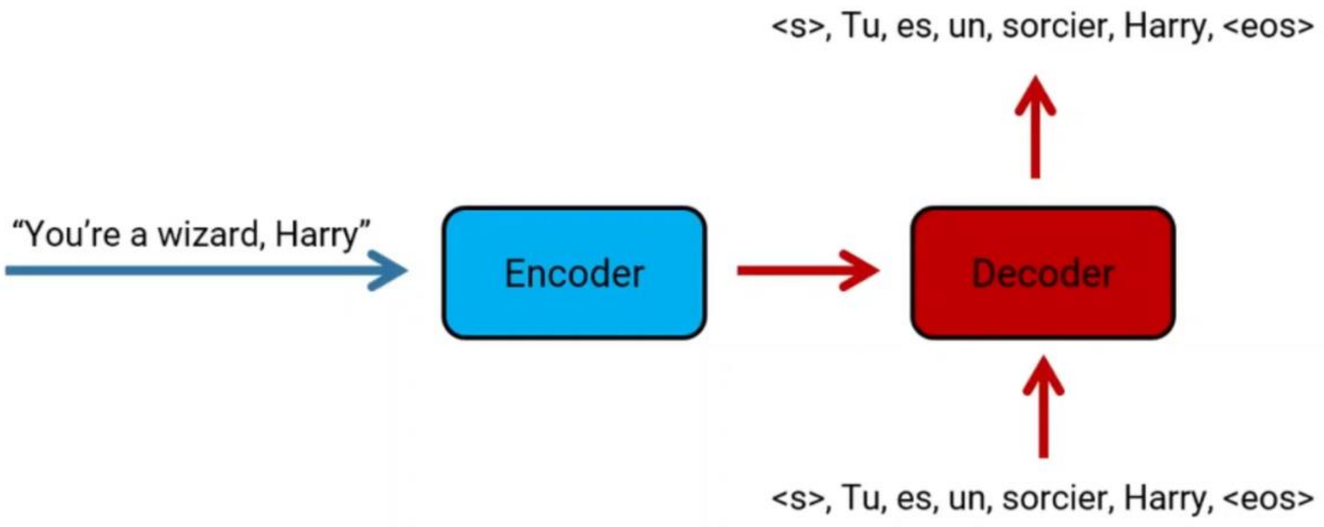
- Wrong starting of prediction token can generate the whole sentence wrong.
- So during Training phase, we fed entire input sentence to the encoder.
- Instead of giving only the “start of sentence” special token, we also give it a part of the target sentence.
- This strategy is called Teacher forcing.

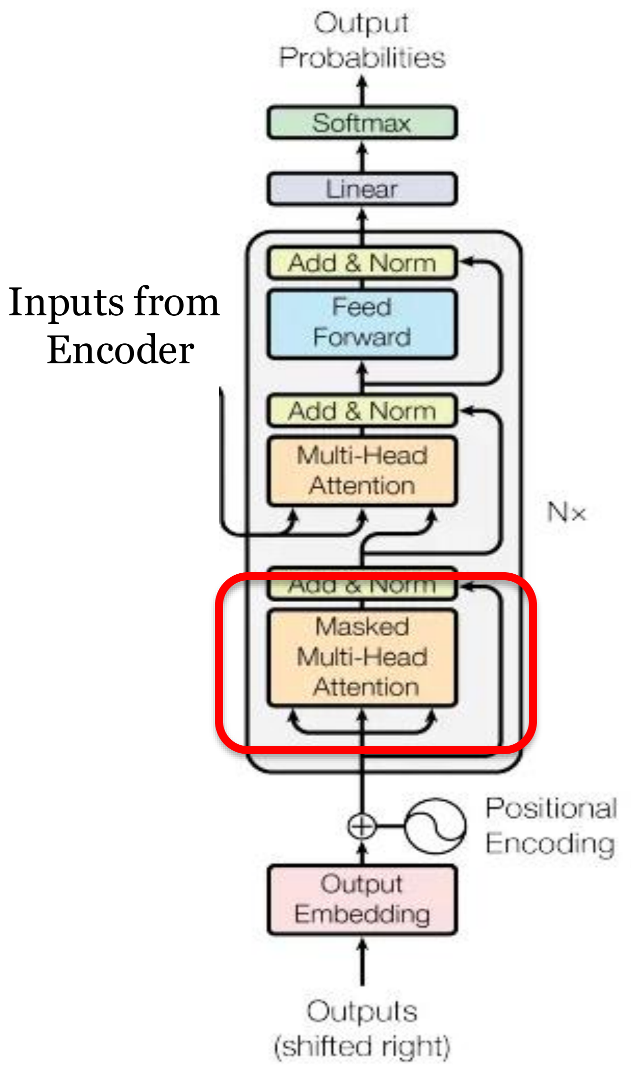


Transformer Deep Down

Masked Multi-Head attention

- The first layer in decoder is Masked Multi-head attention.
- We can feed to whole target sentence to decoder together with the Encoder’s output so that it can train faster.

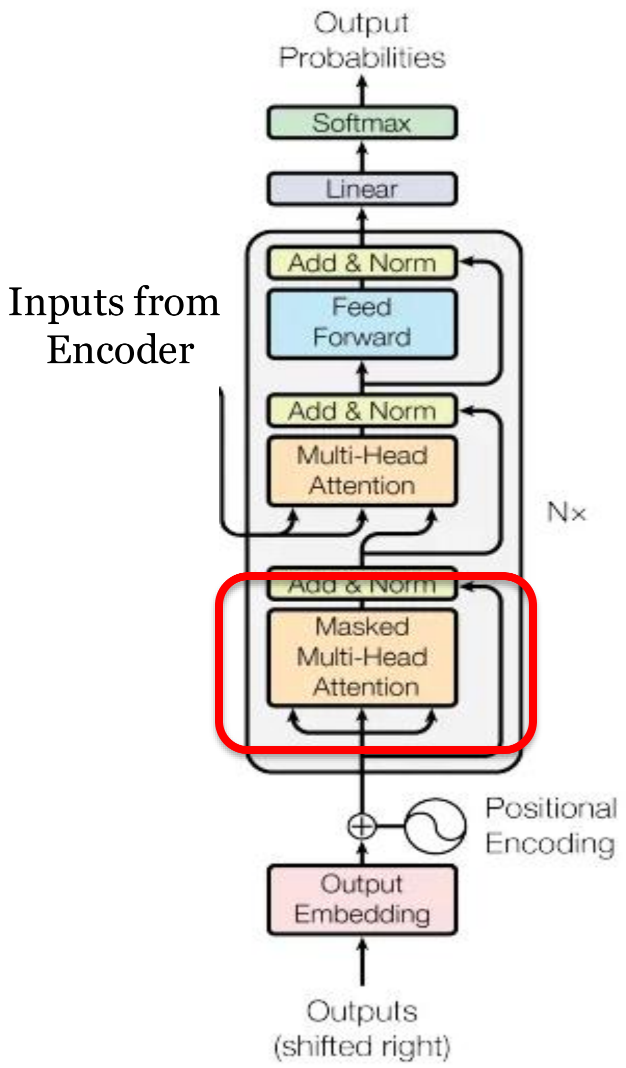




Transformer Deep Down

Masked Multi-Head attention

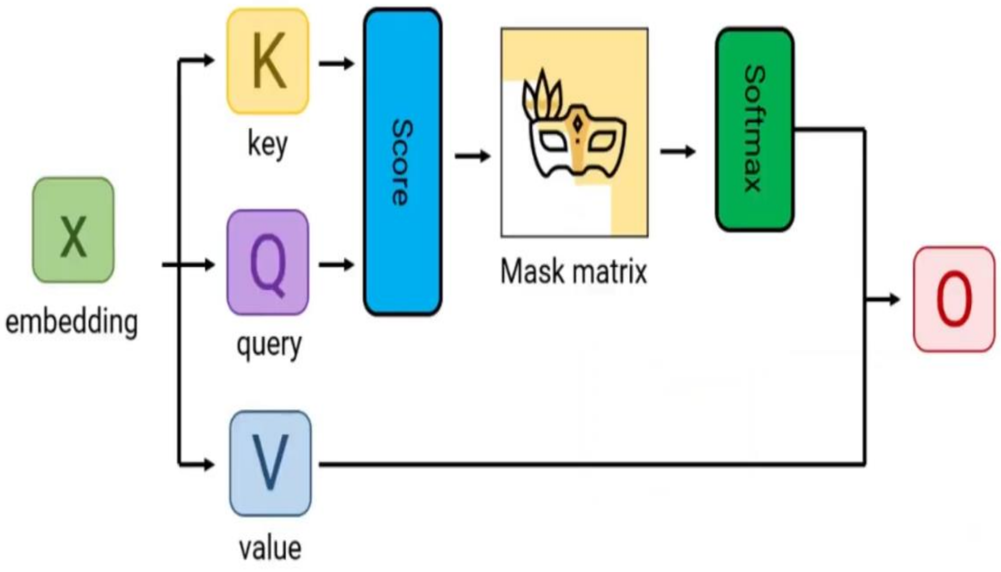
- Decoder is suppose to predict the output which needs to be as much as similar to the ground truth.
- This way Decoder knows what's it next word should be and it harm decoder's ability to generalize.
- In order to prevent Decoder from "cheating", we use masking.
- For example when decoder is predicting fourth output token, we should mask every word from index 4 until the end of target sentence.



Transformer Deep Down

Masked Multi-Head attention

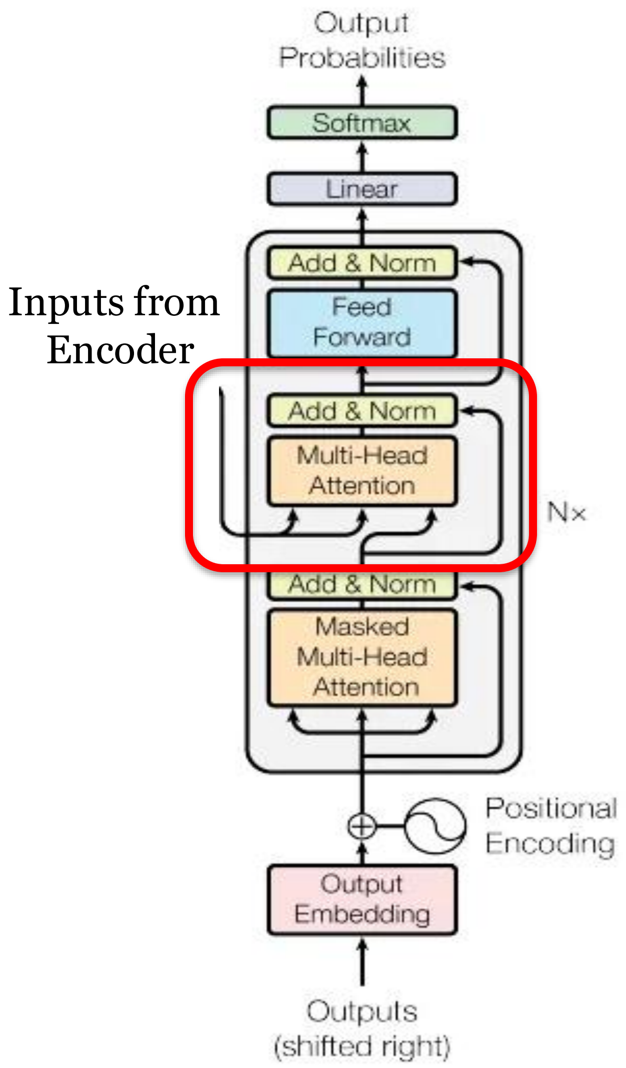
- Self attention mechanism in Masked multi-head attention is same as encoder's self attention mechanism with one difference.
- Mask matrix indicates that we apply masking on score achieved from dot product between Key and Query.
- Output of masking is then fed to the softmax and this whole dataflow remains same as in encoder.



Transformer Deep Down

Multi-Head Attention

- Self attention mechanism uses 3 inputs in order to produce the Key, Query and Value matrices.
- For performing self attention in Decoder, Keys and Values comes from encoder side.
- Encoder takes entire input sentence, and produces the output.
- We takes this output and make two copies of it by linear transformation.
- One copy will be the "Key" and other will be "value".
- "Query" comes from decoder's Masked Multi-head attention output.



Transformer Deep Down

References

- https://jalammar.github.io/illustrated-transformer/
- https://arxiv.org/abs/1706.03762
- https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- https://peterbloem.nl/blog/transformers
- https://www.youtube.com/watch?v=U0s0f995w14

The Ignitarium logo represents a stylized Delta - the classical symbol for fire. The Delta logo is created from the amalgamation of smaller deltas signifying the stages of transition from spark to ember to flame to fire.

THANK YOU

email: info@ignitarium.com

Ignitarium Technology Solutions

Bangalore, India
#2615, 3rd Floor,
27th Main Road, Sector 1,
HSR Layout,
Bangalore – 560102
Phone: +91-80-42054217

Kochi, India
Office No. 1A-1,
Jyothirmaya IT Building P.O,
Infopark Phase 2, SEZ, Kochi,
Kerala 682303
Phone: +91-484-4876089

San Jose, CA
2570 N. First Street,
Suite 200 San Jose,
CA 95131
Phone: +1 512 640 3488

Austin, TX
555 Round Rock West Drive,
Suite E217,
Round Rock, TX 78681
Phone: +1-512-669-9214

Yokohama, Japan
7-25-22 Okamura, Isogo-ku,
Yokohama 235-0021, Japan
Phone: +81-901-707-4661
Tel/Fax: +81-45-350-1757