Transformer Architecture

A transformer model basically helps in transforming one sequence of input into another depending on the problem statement. e.g. Translation from one Language to another

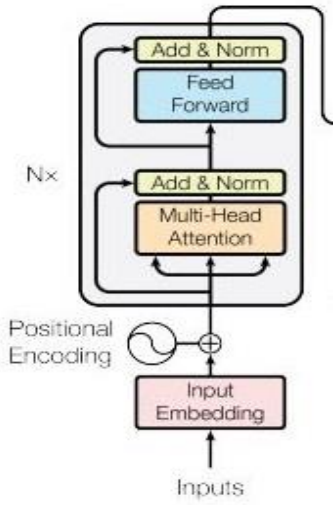
Two major Neural Networks in Transformer

- Encoder
- Decoder

Let's go through the Architecture from bottom to top as seen in the Images (Usually on the right) for Encoder and Decoder.

A. Encoder: decide which token in the input text is important.

- it tries to provide some attention to some tokens coming from the input sentence.



1. Input Embedding:

- Embedding of raw text. change to some numerical r epresentation (naïve example: OneHotEncoding)
- Any effective method can be used.

2. Positional Encoding:

- The input Embedding above generally doesn't acco unt for the position at which tokens (similar) appear
- Context could be lost.

i = index of the i-th unit in the embedding;
pos = Position of text
d_model = embedding dimension for each token e.g. 512

Even: [0,2,…,510]; Odd:[1,3,…,511]

`Output= n x 512`

$$PE_{(pos\ 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos\ 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Attention:

Attention is a mechanism that basically provides importance to a few key tokens in the inpu t sequence by altering the token embeddings.

Why not Seq2Seq Model?
The model needs to remember more than an LStM's or RNN's Memory Capacity which Atte ntion does easily.

3. Multi-Head Attention:

Multiple attention layers parallel to each other. e.g. Input Embedding =20 tokens, divide th e embedding into multiple segments and feed to multi-attention layers.

Why? To have different perspective and get a generalized perspective

How Attention Layer Work:

Important Matrices used in Attention Layer:

- *Query and Query_weight*
- *Key and Key_weight*
- *Value and Value_weight*

dimensions can be selected arbitrarily.

n = inputs

d_model = dimension of embedding for each token

d_k = dimension of Query and Key Vector

d_v = dimension of value vector

Query Weight = d_model X d_k

Key weight = d_model X d_k

Value Weight = d_model x d_v

Query = Output_of_Embeding (n x d_k) X Query_Weight (d_model x d_k) = (n x d_k)

Key = Output_of_Embeding (n x d_k) X Key_Weight (d_model x d_k) = (n x d_k)

Value = Output_of_Embeding (n x d_k) X Value_Weight (d_model x d_v) = (n x d_v)

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

Attention(Q,K,V) = dim(n X d_v)

SoftMax: A function that convert input values to 0-1 values that sums up to 1.

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

For Multi-Head, concatenate all *Attention(Q,K,V)* matrices from each head.

Multiply Output matrix by a **Weight matrix** such that the output is of dimension n x d_model

4. Post Layer Normalization (Add & Norm):

Normalization uses residual connections to look back at the input of the previous layer and its output simultaneously (idea: not to lose on any information in the input layer).

- Add Input and Output of Multi-Head Attention = V (n x d_model)
- Normalize the Added matrix V (this is done with each row):

$$\gamma\frac{v-\mu}{\sigma}+\beta$$

$\gamma = damping\ factor$, μ = mean ; β = Regularization constant ; $\sigma = standard\ deviation$

5. Feed Forward Network

straight forward shallow neural network (2 Layered NN) and applying ReLU activation function.

$$FFX(x) = \max(0, xw_1 + b_1)\, w_2 + b_2$$

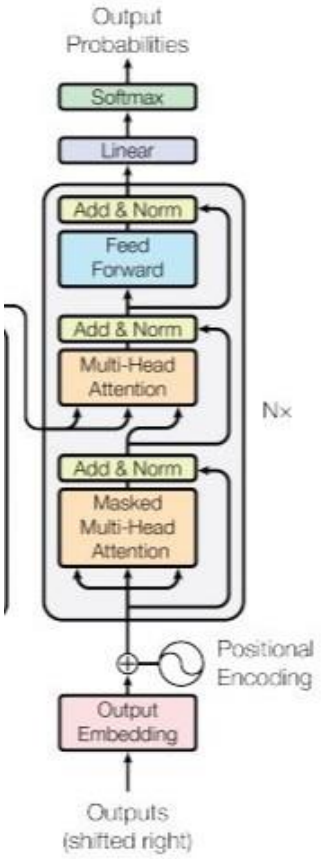
Again, we do a Post Layer Normalization with the input and output of FNN.

This iteration is repeated for N iterations and the output of the last iteration is passed to the **Decoder.**

B. Decoder:

The major aim of using a Decoder is to determine the output sequence's tokens one at a time by using:

- Attention known for all tokens from Encoder
- All predicted tokens of output sequence so far



Some layers not already explained from Encoder:

- Outputs: numerical representation of the output sequence generated using a tokenizer. This representation is right shifted. Why?
    - Encoder uses previous prediction and attention to predict the next sequence. Since there is no previous token for the first token, the output sequence is shifted & a "Beginning of Sentence (BOS)" is inserted at the beginning.
- Masked Multi-Head Attention: Similar to Multi-Head Attention but Attention is only calculated on tokens up to the current position and not on future tokens.

Important to understand that the Multi Head Attention layer in the decoder does not require any training because Attention is calculated using Query Matrix from Masked "MHA" and Key, Value from the output of the encoder.

This layer is using pretrained information. Since Query vector will only be available for just 'seen' tokens, this layer can't see beyond what is predicted in the output sequence.

All layers are repeated for N iterations just as done with Encoder after which We have Linear Layer followed by a SoftMax function giving us the probability for the most suitable token in the predicted sequence.

The predicted token goes in the tail of the output sequence and a new iteration is started until the predicted token is "End Of Sentence" (EOS).