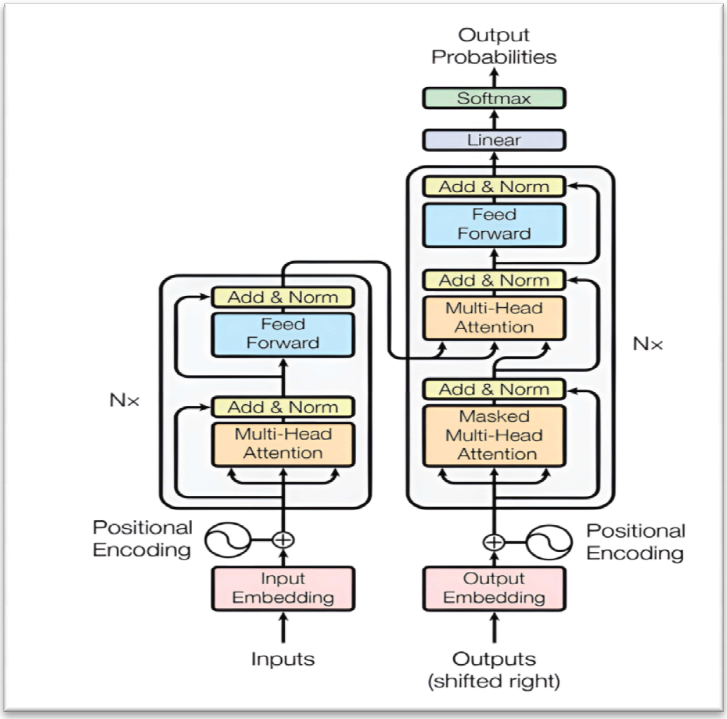
Breaking Down ‘Attention Is All You Need’: A Deep Dive into Transformers

Attention all you need is a Research paper published by people from Google Brain, Google Research, and the University of Toronto Explicitly speaking about the structure of the Transformers and the developments made in the field of Transformers. Transformer is a type of neural network architecture that excels at processing sequential data, like text or audio, by using a mechanism called self-attention to understand relationships between different parts of the input. In this paper, they have proposed a Transformer architecture that relies on the concept of self attenuation which helps in finding the dependencies between input and output. This Transformer was trained for 12 hours continuously with eight P100 GPUs.

Background:

Before transformers models like ByteNet, ConvS2S, etc., were more focused on processing the data parallelly, and due to this these models were suffering from long-term dependency problems. In The Transformers this issue was solved by introducing the concept of self-attention. This was the main difference Between the previous models and the transformers. Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence to compute a representation of the sequence. Self-attention captures the relation between the position of the input sequences in the data. Unlike traditional CNN, RNNs depend on self-attention only for sequence modeling These Transformers are entirely dependent on self-attention rather than RNNs and CNNs, which helps in improving the overall accuracy and capturing dependencies more accurately.



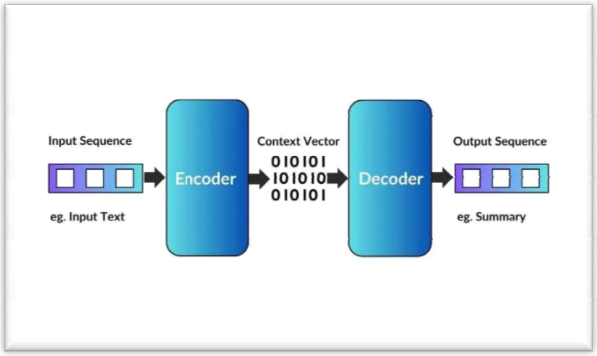
Transformer Architecture:

Let us look at the Model's overall Structure and let's understand the Whole architecture In the later stage.

This is the overall Structure of the Transformer Now let's break down the Transformer and understand the real Structure of it.

Encoder and Decoder:

The encoder and decoder are the main parts of the Transformer where the encoder helps to encode the data and the decoder decodes the data and brings it to the original form. Initially, the input is given to the encoder after using methods like embeddings. Embedding is the technique that converts the words into numerical or vector form so that the machine can understand and learn the patterns.

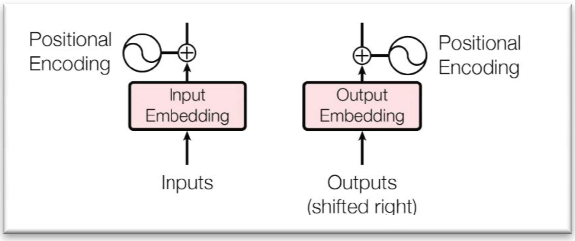


This is the whole structure behind a transformer But before learning about encoder and decoder architecture we should go through some of the main topics that are used to construct an encoder and decoder.

Topics:

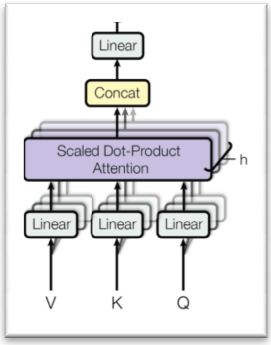
1. Positional Encoding
2. MultiHead Attention
3. Masked MultiHeadAttenction
4. Feed Forward NeuralNetwork
5. Layer Normalization
6. Linear & Softmax
7. Residual Connection

1. Positional Embeddings:



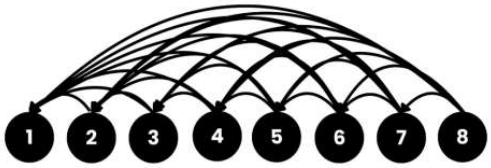
This is the place in the starting where the input embeddings are sent to the model the position Embeddings are implemented. These are the vectors implemented to find the position of the words and the distance between the two words.

2. Multi Head Attention:



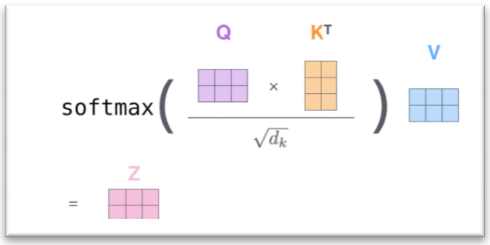
In MultiHead Attention we have to focus mainly on two things one is attention and the other is scalar dot product let's break it down.

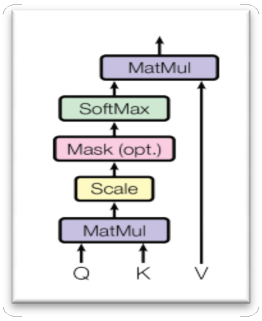
Attention:



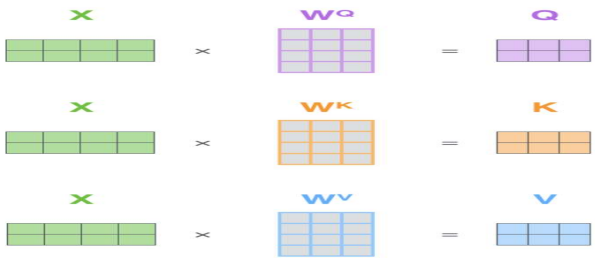
Before understanding Multi-Head Attention, it is important to first grasp the concept of attention. In a Transformer, attention is a mechanism that allows the model to focus on the most relevant words in a sentence by assigning different importance (weights) to each word. Self-attention enables each word in a sequence to look at all other words and determine their relevance, which helps the model capture contextual relationships more effectively. An attention function takes a query and a set of key-value pairs as input, where all elements are represented as vectors. The output is generated by computing a weighted sum of the values, with each weight determined by a compatibility function that measures how relevant the query is to each key.

Scaled dot product:





One of the fundamental components of attention is the Scaled Dot-Product Attention. This mechanism is primarily computed using the query, key, and value vectors, which are derived from the input embeddings. These vectors are generated by multiplying the embeddings with weight matrices W_q, W_k, W_v where these weights are initially randomized and learned during training. The key vector is obtained by multiplying the embeddings with W_k , the query vector is obtained by multiplying with W_q , and the value vector is obtained by multiplying with W_v . The scaled dot product computes attention scores by taking the dot product of the query and key vectors, then scaling the result to avoid extremely large values, and finally applying a softmax function to generate a probability distribution. These scores are then used to compute a weighted sum of the value vectors, producing the final attention output.



The Transformer model's power comes from its Attention Mechanism, which enables it to process sequences efficiently by focusing on relevant words. The foundation of this mechanism is Scaled Dot-Product Attention, which forms the core of how attention is computed. Multi-Head Attention extends this concept, allowing the model to capture a broader range of dependencies and representations within a sentence.

3. Masked MultiHead Attention:

Multi-head attenction is an important stage in transformer which is in both the encoder and decoder architectures. But masked multi-head attention exists only in the decoder. Now let us understand why it is implemented and why only in the decoder.
Masking is a technique which is used to prevent cheating in the decoder. This is done before applying the softmax layer where the future positions are masked or hidden ie by applying –inf as the value to the values in the upper triangular matrix..

$$\begin{bmatrix} 0.67 & -\infty & -\infty \\ 0.89 & 0.27 & -\infty \\ 0.30 & 0.25 & 0.37 \end{bmatrix}$$

After applying Softmax

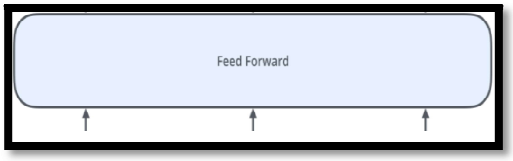
$$\begin{bmatrix} 1.00 & 0.00 & 0.00 \\ 0.65 & 0.35 & 0.00 \\ 0.33 & 0.31 & 0.35 \end{bmatrix}$$

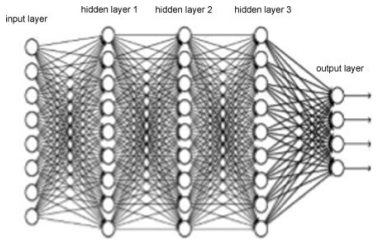
This is masking in the decoder. When the values are given to the encoder all the values are sent together but in the decoder, values are sent one by one and the next word that is to be predicted will not be sent to the decoder.

4. Position-wise FeedForward Networks:

Feed-forward networks are present in both the parts encoder and decoders. Position-wise feed-forward networks are fully connected neural networks that take the output from the Multihead attention layer after normalization. This consists of two linear transformations and the Relu activation function the hidden layer that is used is 2048 and it is a hyperparameter and can be changed as required.

$$FFN(x)=\max(0,xW_1+b_1)W_2+b_2$$





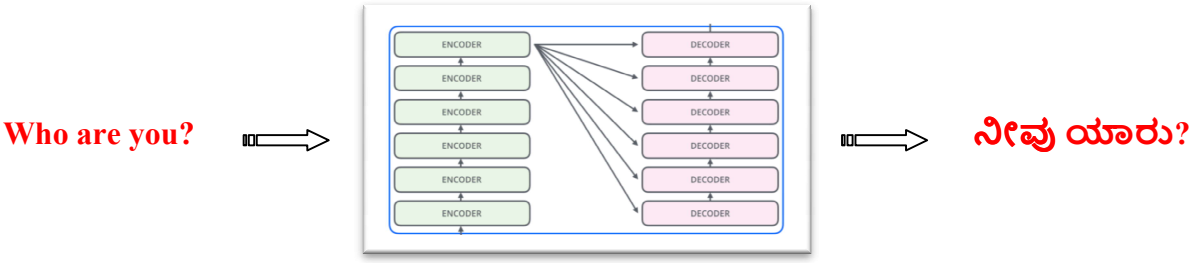
These networks operate independently on each position (token) in the sequence, hence the name "position-wise". This feed-forward network enhances feature representation, introduces non-linearity, and allows the Transformer to learn more complex patterns beyond what is captured by attention mechanisms. Variations of FFN use different activation functions (GELU), gated mechanisms, or dropout layers for improved efficiency and stability.

5. Layer Normalization:

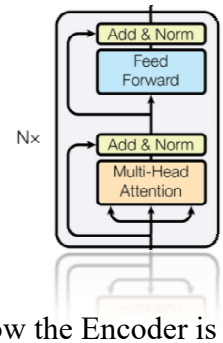
Layer normalization (LN) is a deep learning technique designed to stabilize training and enhance neural network performance by addressing the internal covariate shift (ICS) problem. During training, the distribution of activations within a layer can change, making it harder for the network to learn effectively. LN tackles this by normalizing activations within each layer independently across all features. It calculates the mean and variance of activations for each layer, then scales and shifts them to ensure they follow a standard normal distribution with a mean of 0 and variance of 1. This normalization helps improve training speed, ensures stable gradients, and allows the model to generalize better across different inputs.

These are the steps that are included in constructing a Transformer.

Encoder and Decoder:

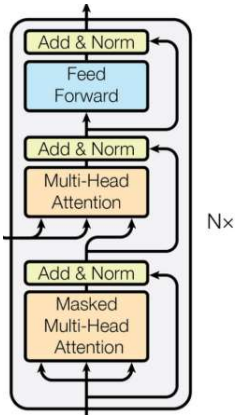


Let's see single Encoder architecture:



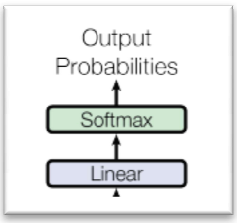
Let's Now understand how the Encoder is connected, The Encoder is made of Multihead attention and feedforward layers. In the starting stage, the embeddings are given to the multihead attention after the positional encoding after that same encodings are added to the output of multihead attention and sent to the layer normalization layer. After the layer normalization, the data is sent to the positional feed-forward network. The result is then added to the input and sent to the layer normalization layer. This is the one encoder structure and like the same, 6 encoders are implemented in one transformer.

Single Decoder architecture:



The decoder is for decoding the values obtained by the encoder, it consists of more layers than the encoder. The first layer is the masked multi-head attention layer the output labels are first sent to it one by one. Along with this, the output labels are given to the Add&Norm layer. Then the second layer is the MultiHead attention layer the outputs from masked attention and the encoder then there is an add&norm layer after this the results are sent to the feed forward network followed by the layer normalization.

6. Linear and softmax:



The Linear layer in a Transformer helps convert the model's internal features into a form that can be used for predictions. After processing through attention and feedforward layers, the decoder produces a high-dimensional vector. The Linear layer takes this vector and transforms it into a size that matches the vocabulary, using a weight matrix and bias. This step is necessary because the model needs to map its learned features to actual words before making a prediction.
The Softmax layer takes the output from the Linear layer and turns it into probabilities. Since the model needs to choose the most likely next word, Softmax ensures that each possible word gets a probability value between 0 and 1, with the total sum being 1. This makes it easy to select the word with the highest probability. Without Softmax, the model would just output raw scores instead of meaningful probabilities.

7. Residual Connections:

A residual layer in a Transformer helps improve learning by allowing the model to pass information from earlier layers directly to later layers. Each sub-layer in the encoder (such as self-attention and feedforward networks) has a residual connection, meaning the input to the sub-layer is added to its output before applying layer normalization. This helps in stabilizing training and prevents the problem of vanishing gradients.
In simpler terms, instead of relying only on the transformed output of a sub-layer, the Transformer also keeps a copy of the original input and adds it back. This ensures that important information from earlier layers is not lost and makes it easier for deeper models to learn. The same concept applies to the decoder as well, allowing it to retain information while refining it through multiple layers.

Conclusion:

The Transformer has revolutionized sequence modeling by replacing traditional recurrent layers with self-attention, making training faster and more efficient. It has set new benchmarks in translation tasks, proving the power of attention-based architectures. Moving forward, there is exciting potential to extend this model beyond text to areas like image, audio, and video processing. Future work will focus on improving efficiency, exploring localized attention, and making generation less dependent on sequential steps.

Reference:

https://jalammar.github.io/illustrated-transformer/
https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf