In the last few years, a new architecture has come to light and has taken the world of NLP by a storm, it was called Transformers. Transformers are based on the concept of attention -we are going to explain that in a while- this concept allowed the transformers to be quickly established as the leading architecture for almost all NLP applications beating existing state-of-the-art models in both performance and the quality of results.

Transformers in a nutshell are basically seq2seq models, they take a text sequence as an input and produce another text sequence as an output. This is mostly referred to as Neural Machine Translation. Now let’s get to how the transformers works, let’s first start with the inputs.

Embedding Layer

Like any NLP model the Transformers needs to know two things about inputs:

1. the embedding of each word
2. the word’s position in the sequence.

Each word in the sequence is first mapped to word IDs using vocabulary, it then maps each ID to an embedding vector that corresponds to that word.

Now we need the positional information of each word, recall that in RNNs since each word is processed sequentially it implicitly knows each position of each word. But in Transformers each sentence is processed as whole in parallel while this is a good thing, it still means that we lose positional information. To address this problem the Transformers, use Positional Encoding which can be thought of as constant values that each corresponds to a position, these values are then added to the Embeddings and we have our input ready to go.

Visualize Inputs

Diagram

Description automatically generated Deep learning models process training samples as batches, the figure below explains how the input get processed in the embedding layer and gives a visualization of the dimension of the inputs in each stage.

As shown in the figure above the output of the embedding layer is a third-dimension matrix with a shape of (Samples x Seq Length x Encoding Size).

Where:

1. Samples refers to number of samples (training instances) in a single batch.
2. Sequence Length refers to the maximum number of words in each sequence.
3. Encoding Size refers to the embedding dimension of each word.

The embedding output is then sent to the Encoder.

Encoder

Diagram

Description automatically generatedThe encoder stack consists of several identical encoders connected sequentially. The first encoder receives it’s input from the embedding layer. The rest receives their inputs from the previous encoder.

Each encoder in the encoder stacks is identical and consists of two sublayers, the first is a multi-headed self-attention mechanism and the second is a simple fully connected feed-forward network, in addition to residual skip connections and followed by layer normalization.

Attention

Let’s consider this sentence: “The boy was holding a blue ball”.

When we reach the word “holding” our brain knows that the coming words are probably an object that the boy is holding, in another words we can say that the words “blue” and “ball” relates to the word “holding”. Or the words “blue” “ball” should be given more attention if we were to extract a meaning from the word “holding” than the rest of the words in the sequence. This is basically what attention is all about it allows the model to focus on specific words that are closely related to each of the words in the sequence. This allows the model to generate word embeddings which take the context of the surrounding words.

Now back to our multi-Headed attention sub layer, we need a way to tell the model that given a word (X) in a sequence assign more weights to the other words in the same sequence that are closely related to the word (X). Recall that in a basic retrieving system the concept is essentially similar. The system is given a query which the system then maps against some keys (title, description, words) and then present the user with best matched results (values). This is exactly what happens in the multi-headed layer the embedding vector is passed through three separate linear layers to produce three results (Query, Key and Value). These three are then used to produce attention score for each word against every other word in the sequence using this formula.

We first compute the similarity between the Query and Key vectors using dot product. The results are then normalized by dividing over the square root of the dimension. We then apply a SoftMax function to make the values in [0,1] range so that the high scores are highlighted, and the smaller ones are depressed. All of this is then multiplied with the Value vector, and we have the final output.

In simpler terms, the way to think about this is to consider that each query that belongs to a word is asking a question about its context in the sentence. The key vector for each other word in the sentence becomes possible context to the query. It then uses Vector similarity (dot product) to tell which key vector best describes the context of the query or in another terms, the key vector that allow the model to understand what the query is about. The value vector is then the word we assign to the query having found its context.

Residual (Skip) Connections, Layer Normalization

The output of the multi-headed attention layer is then added to the original input to help avoid the vanishing/exploding gradient problem. The output of this addition is then normalized by LayerNorm layer to stabilize the network.

Feed-Forward neural network

Diagram

Description automatically generatedThe output then goes through a feed-forward layer to further process the attention output and give it richer representation.

*Figure(3): Transformers Input and Encoder layers*