Transformers
Documentation

BERT

URL

Bidirectional Encoder Representation from Transformer

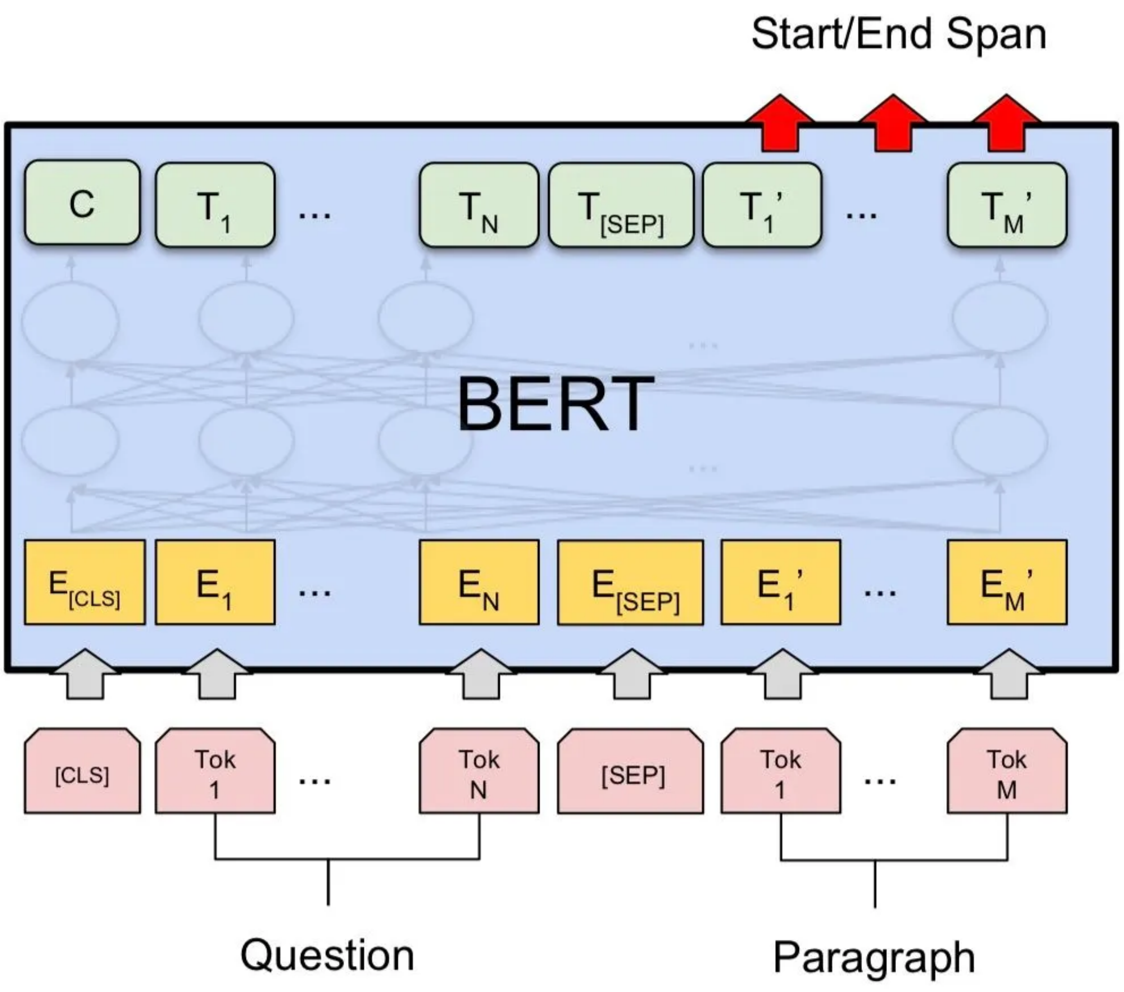
Pre-requisite : Need to have basic understanding of transformers.

Bidirectional Encoder Representation Transformers (BERT) were introduced in 2018 by Google.

This architecture is far superior than the LSTMs as BERT captures context of the words in a given sentence better than LSTMs.

Problems to Solve

- Neural Machine Translation
- Question Answering
- Sentiment Analysis
- Text Summarization



Training

Generally training of BERT models takes place in 2 parts

- Pre-Training : In this the entire Transformer is trained to understand Language.
- Fine-Tune : In this only the Encoder part of the transformer is trained on a very specific task (summarization).

Pre-Training

The main goal of pre-training is to enable the model to understand language and context. This is achieved by training BERT on two unsupervised tasks: masked language modeling and next sentence prediction.

1. Masked Language Modeling

   In this model receives a input sentence which has some of its words [masked] and the model has to predict these masked words. This allows the model to understand language of the sentence.

2. Next Sentence Prediction

   In this the model receives two input sentences and has to predict if the 2nd sentence will follow the 1st sentence. This allows the model to understand context of different sentences.

Fine-Tuning

Fine-tuning is the process of further training a deep learning model on a specific task after pre-training. It usually involves re-training the last few layers of a pre-trained model using the task-specific data. This allows the model to adapt to the new task and produce better results. By fine-tuning a pre-trained model, we can save time and resources, as the model is already trained on the general features of the dataset. The fine-tuned model can then be used for the specific task it was trained on.

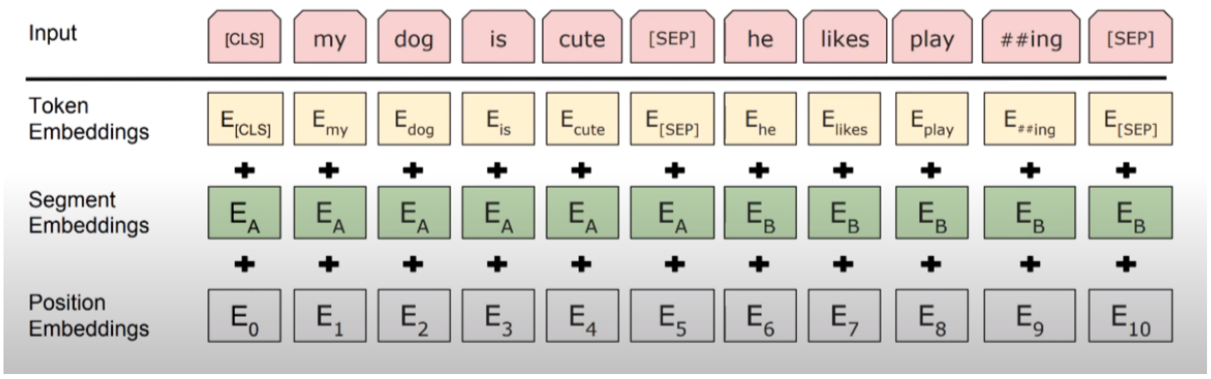
We con now further train BERT on specific NLP tasks like Summarization, Question-Answering. This is done by replacing the last few layers of the architecture with preferred ones and training the model on required dataset. As the model only needs to learn the last few layer params from scratch , the training time is very low.

Embeddings

Think of embeddings as compact and condensed representation of the original text, where each word or piece of text is mapped to a unique vector in a high-dimensional space.

The initial Embedding for BERT is made out of 3 embeddings

- Token Embeddings : encoding the words of the sentence into a vector (Word Piece)
- Segment Embeddings : Sentence number encoded into a vector
- Positional Embeddings : Position of that embedding in the sentence.


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

The Positional & Segment Embeddings are required for temporal understanding as all these vectors are fed into the model simultaneously.

Output

All the word vectors generated in the output side are generated simultaneously. But these numbers have no meaning to us humans as we can not understand these embeddings, therefore we add a linear layer with SoftMax activation at the output of the model. This layer size should be equal to the vocab size which was used to encode the inputs. (eg. Word Piece : 30k). This converts the outputs into a distribution. This `new_distribution` and the actual `distribution` are used to compute the loss of the model.