BERT Model

BERT is an open source machine learning framework for natural language processing (NLP). BERT is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context. The BERT framework was pre-trained using text from Wikipedia and can be fine-tuned with question and answer datasets.

BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. (In NLP, this process is called attention.)

BERT is currently being used at Google to optimize the interpretation of user search queries. BERT excels at several functions that make this possible, including:

- Sequence-to-sequence based language generation tasks such as:
    - Question answering
    - Abstract summarization
    - Sentence prediction
    - Conversational response generation
- Natural language understanding tasks such as:
    - Polysemy and Coreference (words that sound or look the same but have different meanings) resolution
    - Word sense disambiguation
    - Natural language inference
    - Sentiment classification

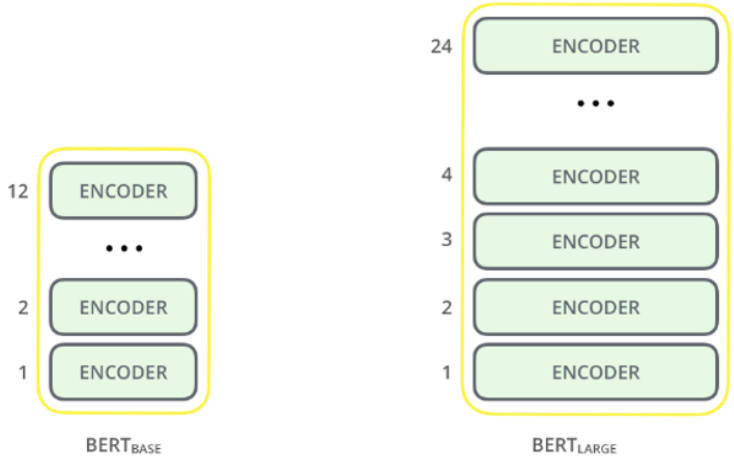
BERT Family:

- *ALBERT* by Google and more — This paper describes parameter reduction techniques to lower memory reduction and increase the training speed of BERT models.
- *RoBERTa* by Facebook — This paper for FAIR believes the original BERT models were under-trained and shows with more training/tuning it can outperform the initial results.

- *ERNIE*: Enhanced Representation through Knowledge Integration by Baidu — It is inspired by the masking strategy of BERT and learns language representation enhanced by knowledge masking strategies, which includes entity-level masking and phrase-level masking.
- *DistilBERT* — Smaller BERT using model distillation from Huggingface.

We currently have two variants available for BERT:

- **BERT Base** : 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
- **BERT Large** : 24 layers (transformer blocks), 16 attention heads, and, 340 million parameters



There are two steps in the BERT framework: ***pretraining and fine-tuning***. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For finetuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters.

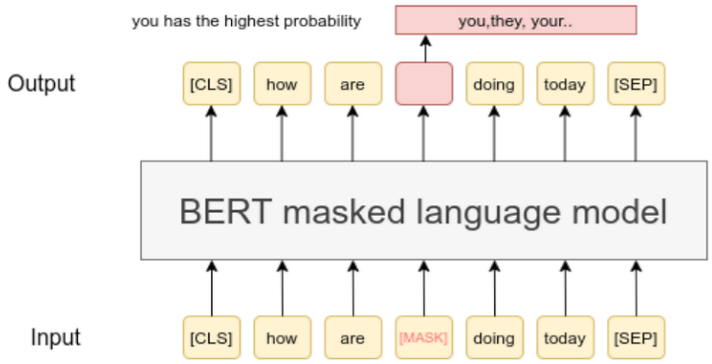
BERT pre-training uses two training strategies:

1. Masked Language Modeling (MLM):

   Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with

a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

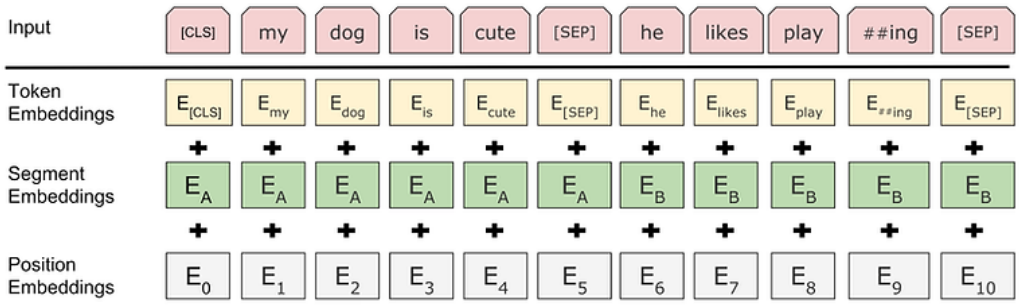
- Adding a classification layer on top of the encoder output.
- Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
- Calculating the probability of each word in the vocabulary with softmax.



2. Next Sentence Prediction (NSP):

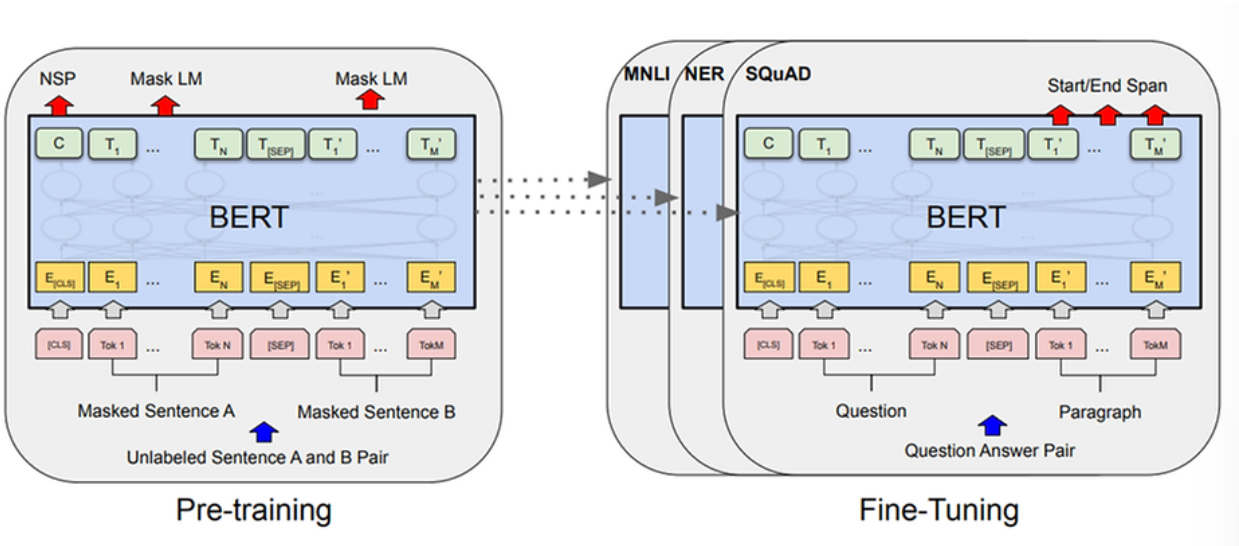
To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

- A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
- A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.



BERT Fine-Tuning is straightforward since the self-attention mechanism in the Transformer allows BERT to model many downstream tasks— whether they involve single text or text pairs—by swapping out the appropriate inputs and outputs.

For applications involving text pairs, a common pattern is to independently encode text pairs before applying bidirectional cross attention. BERT instead uses the self-attention mechanism to unify these two stages, as encoding a concatenated text pair with self-attention effectively includes bidirectional cross attention between two sentences.



CODE:

```
import tensorflow as tf

import tensorflow_hub as hub

!pip install tensorflow-text

import tensorflow_text as text


# Download the BERT preprocessor and encoder for generating the model

bert_preprocess =
hub.KerasLayer(“https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3”)

bert_encoder =
hub.KerasLayer(“https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4”)


# Bert layers

text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')

preprocessed_text = bert_preprocess(text_input)

outputs = bert_encoder(preprocessed_text)


# Neural network layers (binary text classification)

l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled_output'])

l = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(l)


# Use inputs and outputs to construct a final model

model = tf.keras.Model(inputs=[text_input], outputs = [l])


# Compile and fit model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X, y, epochs=epochs)
```