CS529

Topic and Tools on Social Media Data Mining

(Assignment #3)

Created by:-
(Tech Pirates)
Kevin Savsani (160101063)
Akul Agrawal (160101085)
Aakash Agrawal (160122001)

Problem Statement

To evaluate the BERT over language identification, the following experiments are to be conducted:

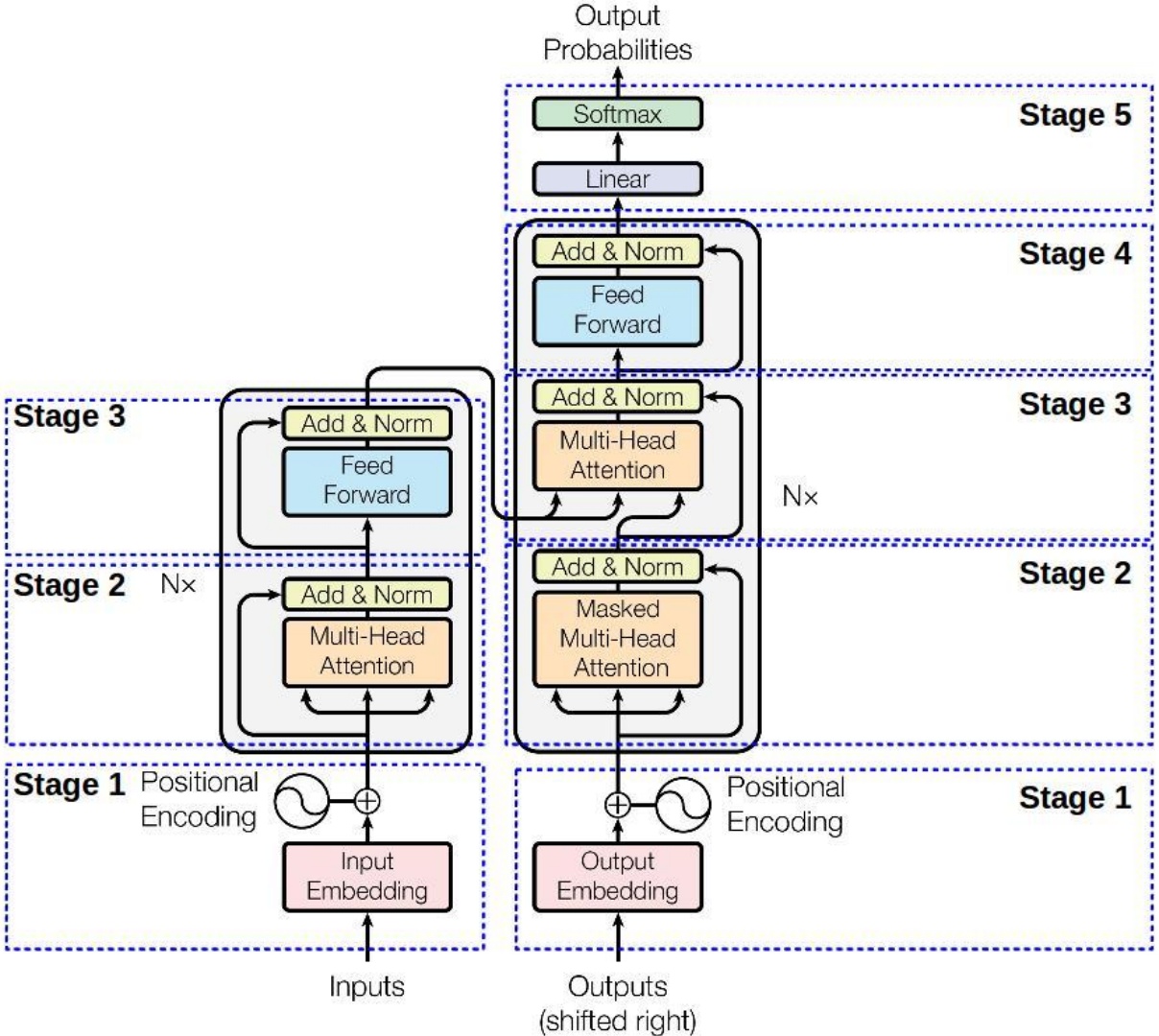
1. Train a BERT model end-to-end for language identification to find the language labels for an entire sequence of words.
2. Pretrain the BERT model and use the pre-trained embeddings into different word classification frameworks like :
    a. Sequence classification models like LSTM models, Transformer model etc. wherein the pre-trained BERT embeddings will be used to initialize the embeddings and the model will predict the language labels for the entire sequence.
    b. Feed-forward networks like MLP and CNN wherein instead of predicting the language labels for every word in the sequence, the model will predict the label for a particular word in a given context
3. Compare the performance obtained to that obtained using traditional word representation models like skip-gram models . Models a. and b. above can be used for comparison with the difference being the embeddings used.
4. You are also encouraged to explore other models for language identification as well as other contextualized word representation models like context2vec, ElMo etc.

BERT - INTRODUCTION

BERT is a deep learning model that has given state-of-the-art results on a wide variety of natural language processing tasks. It stands for **Bidirectional Encoder Representations for Transformers**.

It has been **pre-trained** on Wikipedia and **BooksCorpus** and requires task-specific fine-tuning.

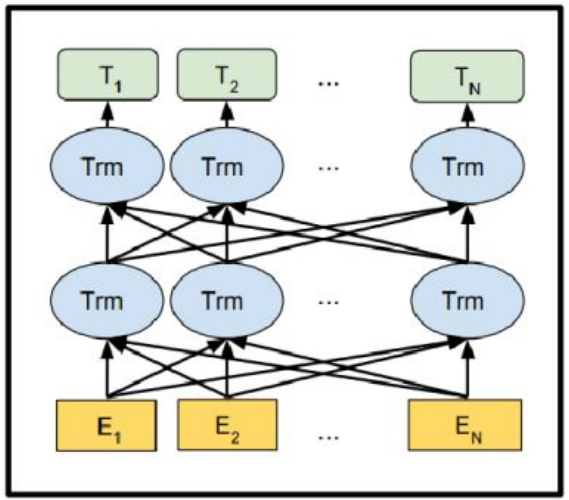
Transformer



Model Architecture BERT

BERT is a multi-layer bidirectional Transformer encoder. There are two models introduced in the paper.

- BERT base – 12 layers (transformer blocks), 12 attention heads, and 110 million parameters.
- BERT Large – 24 layers, 16 attention heads and, 340 million parameters.
- Flow of information of word in BERT is described below



A word starts with its embedding representation from the embedding layer. Every layer does some multi-headed attention computation on the word representation of the previous layer to create a new intermediate representation. All these intermediate representations are of the same size. In the figure above, **E1** is the embedding representation, **T1** is the final output and **Trm** are the intermediate representations of the same token. In a 12-layers BERT model a token will have 12 intermediate representations.

Pretraining of BERT

Masked Language Modeling - Language Modeling is the task of predicting the next word given a sequence of words. In **masked language modeling** instead of predicting every next token, a percentage of input tokens is masked at random and only those masked tokens are predicted.

Next Sentence Prediction - Next sentence prediction task is a binary classification task in which, given a pair of sentences, it is predicted if the second sentence is the actual next sentence of the first sentence.
This task can be easily generated from any monolingual corpus. It is helpful because many downstream tasks such as Question and Answering and Natural Language Inference require an understanding of the relationship between two sentences.

Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

Masked Language Modeling

MASKED LANGUAGE MODELING OVER STANDARD LANGUAGE MODELING

- Bi-directional models are more powerful than uni-directional language models. But in a multi-layered model bi-directional models do not work because the lower layers leak information and allow a token to see itself in later layers.

MASKED LANGUAGE MODELING IMPLEMENTED IN BERT

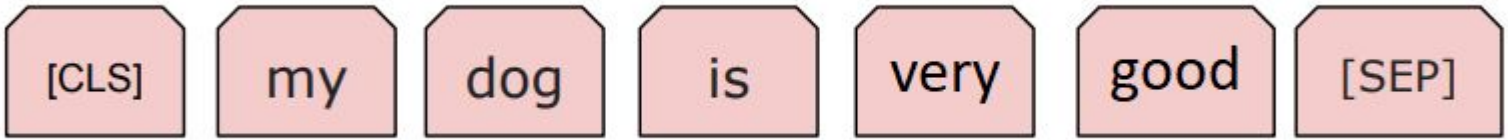
- The masked words are not always replaced with the masked token – **[MASK]** because then the masked tokens would never be seen before fine-tuning. Therefore, 15% of the tokens are chosen at random and –
    a. 80% of the time tokens are actually replaced with the token [MASK].
    b. 10% of the time tokens are replaced with a random token.
    c. 10% of the time tokens are left unchanged.

THE INPUT TEXT REPRESENTED BEFORE FEEDING TO BERT

BERT can be used for a wide variety of tasks. The two pre-training objectives allow it to be used on any **single sequence** and **sequence-pair** tasks without substantial task-specific architecture modifications.

The input representation used by BERT is able to represent a single text sentence as well as a pair of sentences (eg., [Question, Answer]) in a single sequence of tokens.

- The first token of every input sequence is the special classification token – **[CLS]**. This token is used in classification tasks as an aggregate of the entire sequence representation. It is ignored in non-classification tasks.
- For single text sentence tasks, this **[CLS]** token is followed by the WordPiece tokens and the separator token – **[SEP].**



THE INPUT TEXT REPRESENTED BEFORE FEEDING TO BERT

- For sentence pair tasks, the WordPiece tokens of the two sentences are separated by another **[SEP]** token. This input sequence also ends with the **[SEP]** token.

- A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar to token/word embeddings with a vocabulary of 2.
- A positional embedding is also added to each token to indicate its position in the sequence.

TOKENIZATION STRATEGY IS USED BY BERT

BERT uses **WordPiece tokenization**. The vocabulary is initialized with all the individual characters in the language, and then the most frequent/likely combinations of the existing words in the vocabulary are iteratively added.

The maximum sequence length of the Input is 512 tokens.

BERT HANDLE OOV WORDS - Any word that does not occur in the vocabulary is broken down into sub-words greedily. For example, if **play**, **##ing**, and **##ed** are present in the vocabulary but **playing** and **played** are OOV words then they will be broken down into **play + ##ing** and **play + ##ed** respectively. (**##** is used to represent sub-words).

FINE-TUNING

No layers are frozen during fine-tuning. All the pre-trained layers along with the task-specific parameters are trained simultaneously.

Discriminative fine-tuning is not used. All the parameters are tuned with the same learning rate.

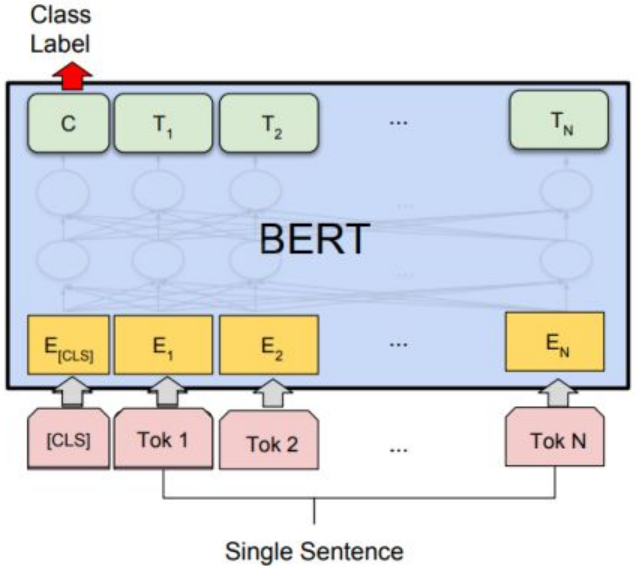
The optimal hyperparameter values used in fine-tuning are task-specific. But, the authors found that the following range of values works well across all tasks –

- Dropout – 0.1
- Batch Size – 16, 32
- Learning Rate (Adam) – 5e-5, 3e-5, 2e-5
- Number of epochs – 3, 4

The authors also observed that large datasets (> 100k labeled samples) are less sensitive to hyperparameter choice than smaller datasets.

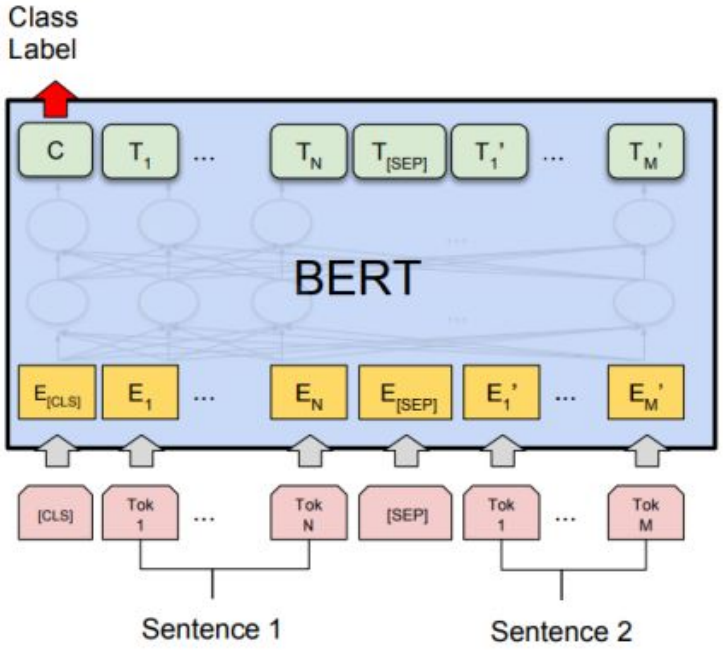
THE FINE-TUNING PROCEDURE FOR SEQUENCE CLASSIFICATION TASKS

The final hidden state of the **[CLS]** token is taken as the fixed-dimensional pooled representation of the input sequence. This is fed to the classification layer. The classification layer is the only new parameter added and has a dimension of **K x H**, where **K** is the number of classifier labels and **H** is the size of the hidden state. The label probabilities are computed with a standard softmax.



THE FINE-TUNING PROCEDURE FOR SENTENCE PAIR CLASSIFICATION TASKS

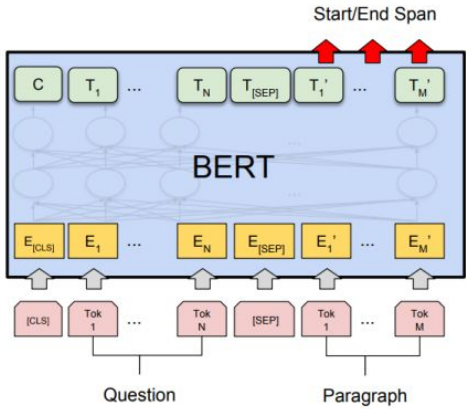
This procedure is exactly similar to the single sequence classification task. The only difference is in the input representation where the two sentences are concatenated together.



THE FINE-TUNING PROCEDURE FOR QUESTION ANSWERING TASKS

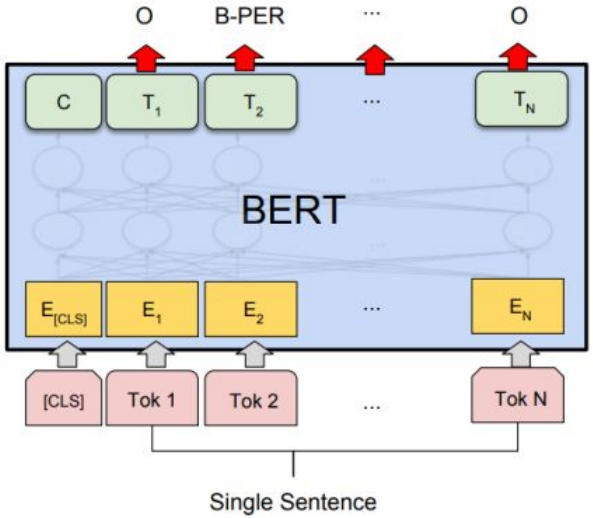
Question answering is a prediction task. Given a question and a context paragraph, the model predicts a start and an end token from the paragraph that most likely answers the question.

Just like sentence pair tasks, the question becomes the first sentence and paragraph the second sentence in the input sequence. There are only two new parameters learned during fine-tuning a **start vector** and an **end vector** with size equal to the hidden shape size. The probability of token **i** being the start of the answer span is computed as – **softmax(S . K)**, where **S** is the start vector and **K** is the final transformer output of token **i**. The same applies to the end token.



THE FINE-TUNING PROCEDURE FOR SINGLE SENTENCE TAGGING TASKS

In single sentence tagging tasks such as named entity recognition, a tag must be predicted for every word in the input. The final hidden states (the transformer output) of every input token is fed to the classification layer to get a prediction for every token. Since WordPiece tokenizer breaks some words into sub-words, the prediction of only the first token of a word is considered.



Plan For Final Submission

Complete all the parts of the problem statement i.e. try the following:

1. Fine tune BERT end-to-end.
2. Word-level classification using supervised machine learning with SVMs but no contextual information.
3. Word-level classification using supervised machine learning with SVMs employing contextual information.
4. Compare the differences in all approaches, and if time permits, apply some other approaches too.