PROJECT DOCUMENTATION REPORT

CS 410 TEXT INFORMATION SYSTEMS FALL 2020

TEXT CLASSIFICATION COMPETITION

Twitter Sarcasm Detection

Team Member	**Email**
Shreya Reddy Peesary	*peesary2@illinois.edu*

INTRODUCTION

Recognizing sarcasm in text is an important task for Natural Language processing to avoid misinterpretation of sarcastic statements as literal statements. The use of sarcasm is prevalent across all social media, micro-blogging and e-commerce platforms. Sarcasm detection is imperative for accurate sentiment analysis and opinion mining. It could contribute to enhanced automated feedback systems in the context of customer-based sites. Twitter is a micro-blogging platform extensively used by people to express thoughts, reviews, discussions on current events and convey information in the form of short texts. Twitter data provides a diverse corpus for sentences which implicitly contain sarcasm.

The aim of this project is to classify the tweets in the given dataset as SARCASM or NOT_SARCASM and beat the base line score of F1: 0.723.

DATASET FORMAT

Train.jsonl:

Shape: 5000 rows x 3 columns

Train dataset is balanced with 2500 SARCASM and 2500 NOT_SARCASM samples.

Test.jsonl:

Shape: 1800 rows x 3 columns

Column Definitions:

- **response** : The Tweet to be classified
- **context** : the conversation context of the ***response*** . The context is an ordered list of dialogue, i.e., if the context contains three elements, c1, c2, c3, in that order, then c2 is a reply to c1 and c3 is a reply to c2. Further, the Tweet to be classified is a reply to c3.
- **label** : SARCASM or NOT_SARCASM
- **id**: String identifier for sample. This id will be required when making submissions. (ONLY in test data)

For instance, for the following training example :

"label": "SARCASM", "response": "@USER @USER @USER I don't get this .. obviously you do care or you would've moved right along .. instead you decided to care and troll her ..", "context": ["A minor child deserves privacy and should be kept out of politics . Pamela Karlan ,

you should be ashamed of your very angry and obviously biased public pandering , and using a child to do it .", "@USER If your child isn't named Barron ... #BeBest Melania couldn't care less . Fact . 💯"]
The response tweet, "@USER @USER @USER I don't get this..." is a reply to its immediate context "@USER If your child isn't..." which is a reply to "A minor child deserves privacy...". Your goal is to predict the label of the "response" while optionally using the context (i.e, the immediate or the full context).

SYSTEM DESIGN

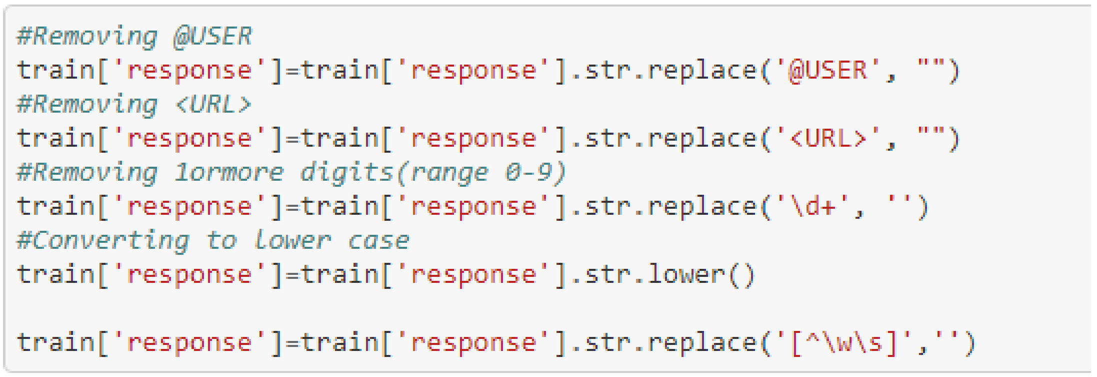
- The code for this project was done using Google Collaboratory (Using GPU Run Time Type).
- The source code can be directly run from Collaboratory using the link by executing the cells step by step from the jupyter notebook link below: https://colab.research.google.com/github/pshreyareddy/CourseProject/blob/main/FinalSubmissionForBERT.ipynb

PACKAGES USED

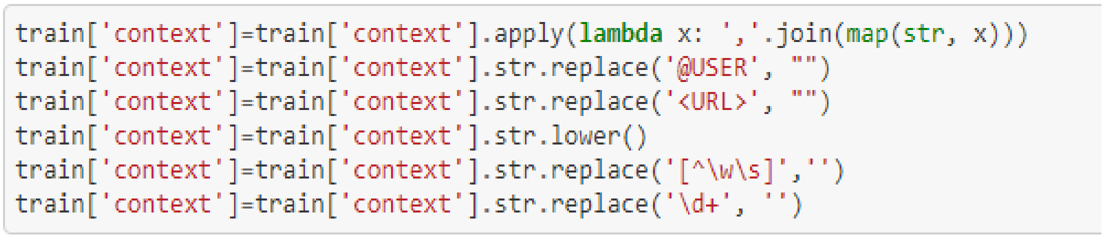
- Python 3.6.9
- Pandas: Python data analysis library.
- Numpy: Python library for working with arrays.
- Re: This is used for regular expression matching.
- String: For common string operations
- Sklearn: Python machine learning library
- TensorFlow2.1.0: an open source software library for high performance numerical computation
- Keras: a deep learning API written in Python, running on top of the machine learning platform TensorFlow.
- TensorFlow Hub: a repository of trained machine learning models.

DATA PREPROCESSING:

- Converted the context column into comma separated string.
- Removed @USER from response and context columns
- Removed <URL> from response and context columns.
- Removed digits (0-9)
- Removed special characters ,white space characters and allowing only alphanumeric characters.
- Converted the response and context columns to lowercase.



```
#Removing @USER
train['response']=train['response'].str.replace('@USER', "")
#Removing <URL>
train['response']=train['response'].str.replace('<URL>', "")
#Removing 1ormore digits(range 0-9)
train['response']=train['response'].str.replace('\d+', '')
#Converting to lower case
train['response']=train['response'].str.lower()

train['response']=train['response'].str.replace('[^\w\s]','')
```



```
train['context']=train['context'].apply(lambda x: ','.join(map(str, x)))
train['context']=train['context'].str.replace('@USER', "")
train['context']=train['context'].str.replace('<URL>', "")
train['context']=train['context'].str.lower()
train['context']=train['context'].str.replace('[^\w\s]','')
train['context']=train['context'].str.replace('\d+', '')
```

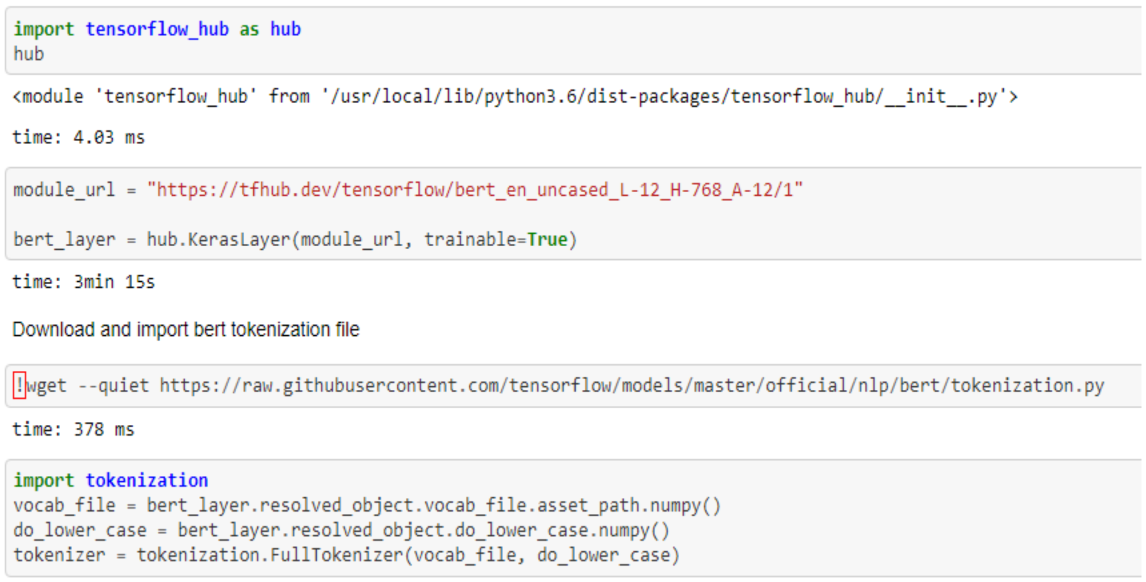
BERT MODEL:

In order to beat the base line score used state of the art language models in BERT (Bidirectional Encoder Representations from Transformers).
BERT relies on a Transformer (the attention mechanism that learns contextual relationships between words in a text). A basic Transformer consists of an encoder to read the text input and a decoder to produce a prediction for the task. Since BERT's goal is to generate a language representation model, it only needs the encoder part. The input to the encoder for BERT is a sequence of tokens, which are first converted into vectors and then processed in the neural network. But before processing can start, BERT needs the input to be massaged and decorated with some extra metadata:

1. **Token embeddings**: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. **Segment embeddings**: A marker indicating Sentence A or Sentence B is added to each token. This allows the encoder to distinguish between sentences.
3. **Positional embeddings**: A positional embedding is added to each token to indicate its position in the sentence.

PROCESS :

- Added a Keras Bert layer using Bert uncased L-12_H-768_A-12.
- Tokenization approach using Bert full tokenizer followed as per standard usage in tensor flow hub (https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1).



```
import tensorflow_hub as hub
hub
```

```
<module 'tensorflow_hub' from '/usr/local/lib/python3.6/dist-packages/tensorflow_hub/__init__.py'>

time: 4.03 ms
```

```
module_url = "https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1"

bert_layer = hub.KerasLayer(module_url, trainable=True)
```

```
time: 3min 15s
```

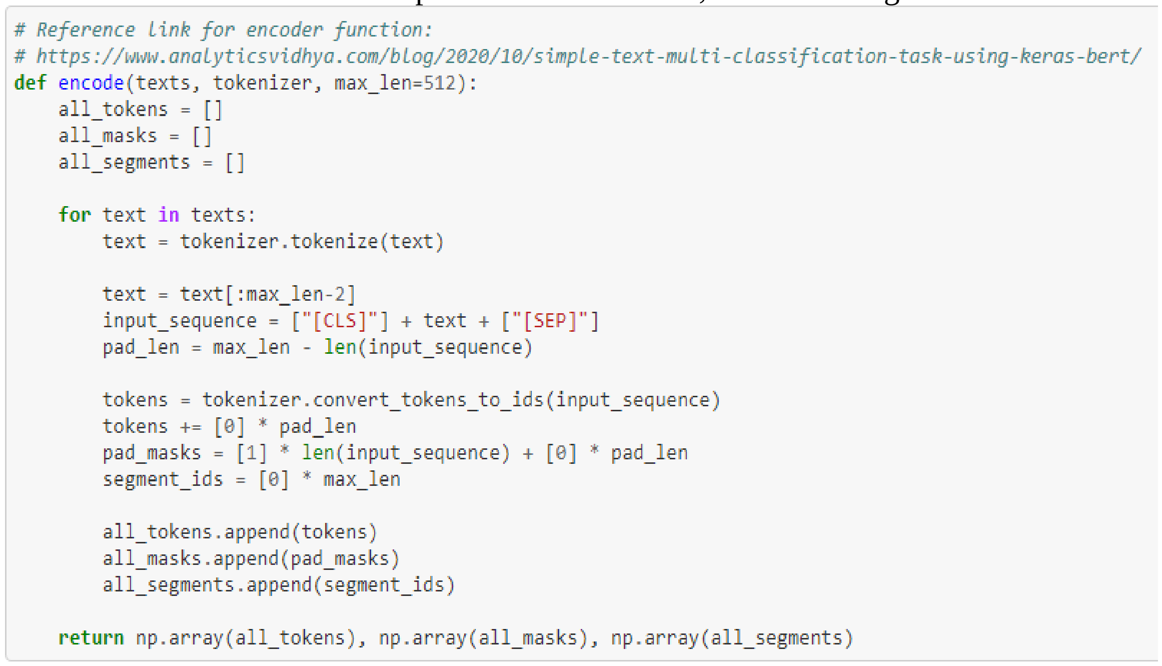
Download and import bert tokenization file

```
!wget --quiet https://raw.githubusercontent.com/tensorflow/models/master/official/nlp/bert/tokenization.py
```

```
time: 378 ms
```

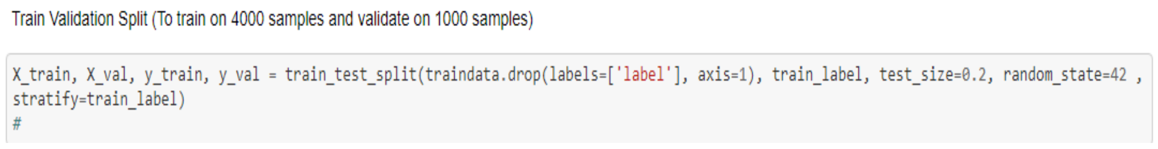
```
import tokenization
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
tokenizer = tokenization.FullTokenizer(vocab_file, do_lower_case)
```

- Define an Encode function to separate text into tokens, masks and segments



```
# Reference link for encoder function:
# https://www.analyticsvidhya.com/blog/2020/10/simple-text-multi-classification-task-using-keras-bert/
def encode(texts, tokenizer, max_len=512):
    all_tokens = []
    all_masks = []
    all_segments = []

    for text in texts:
        text = tokenizer.tokenize(text)

        text = text[:max_len-2]
        input_sequence = ["[CLS]"] + text + ["[SEP]"]
        pad_len = max_len - len(input_sequence)

        tokens = tokenizer.convert_tokens_to_ids(input_sequence)
        tokens += [0] * pad_len
        pad_masks = [1] * len(input_sequence) + [0] * pad_len
        segment_ids = [0] * max_len

        all_tokens.append(tokens)
        all_masks.append(pad_masks)
        all_segments.append(segment_ids)

    return np.array(all_tokens), np.array(all_masks), np.array(all_segments)
```

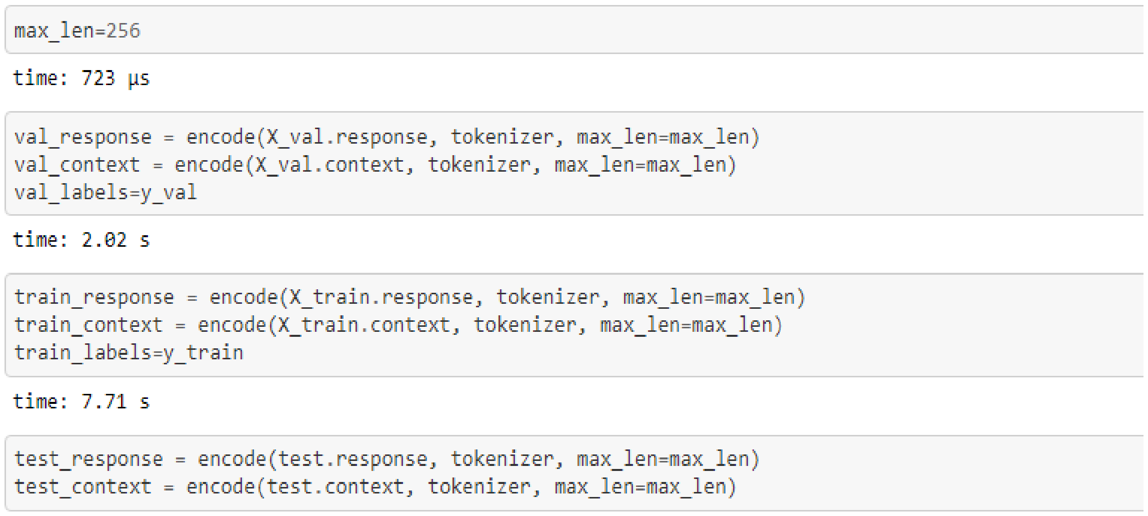
- Split the train data set into training and validation



Train Validation Split (To train on 4000 samples and validate on 1000 samples)

```
X_train, X_val, y_train, y_val = train_test_split(traindata.drop(labels=['label'], axis=1), train_label, test_size=0.2, random_state=42 ,
stratify=train_label)
#
```

- Apply the encode function on both response and context.



```
max_len=256
```

```
time: 723 µs
```

```
val_response = encode(X_val.response, tokenizer, max_len=max_len)
val_context = encode(X_val.context, tokenizer, max_len=max_len)
val_labels=y_val
```

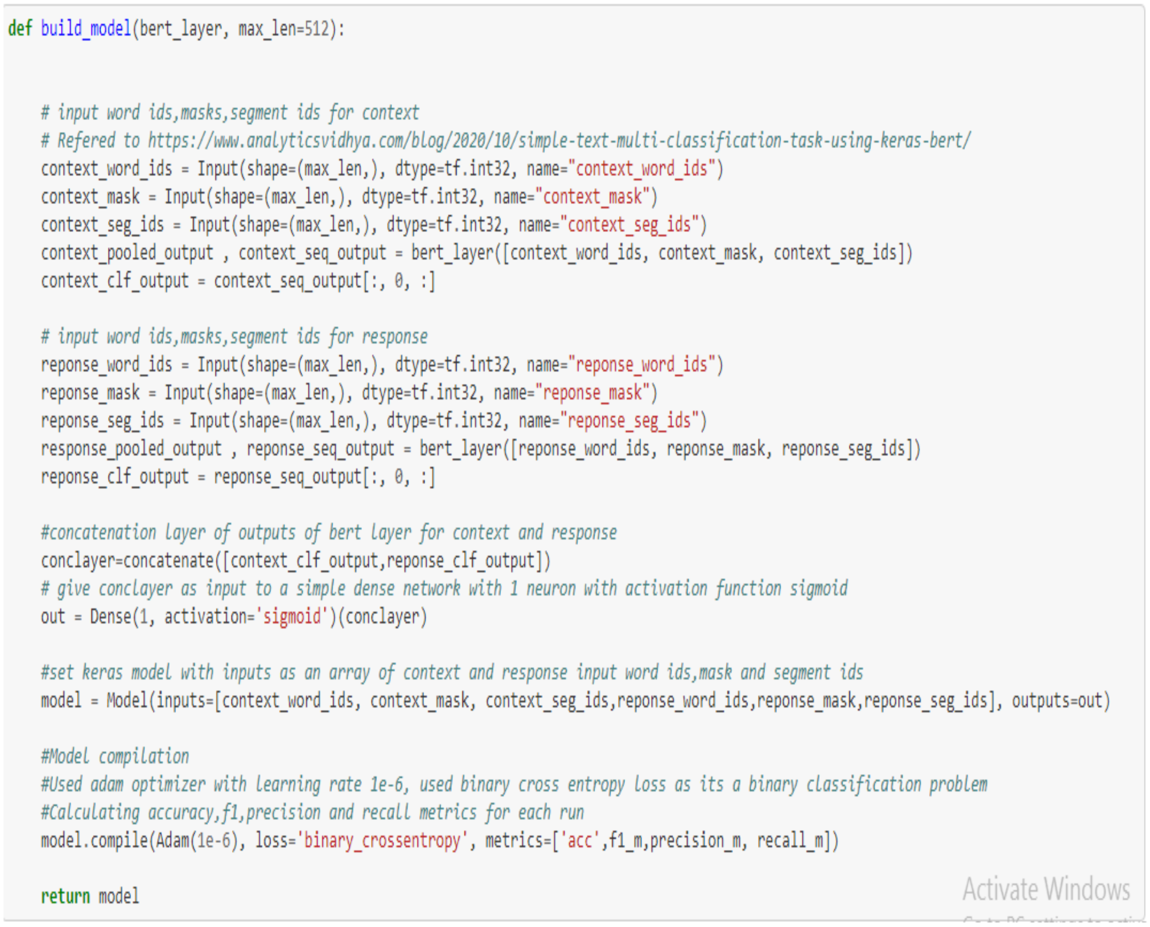
```
time: 2.02 s
```

```
train_response = encode(X_train.response, tokenizer, max_len=max_len)
train_context = encode(X_train.context, tokenizer, max_len=max_len)
train_labels=y_train
```

```
time: 7.71 s
```

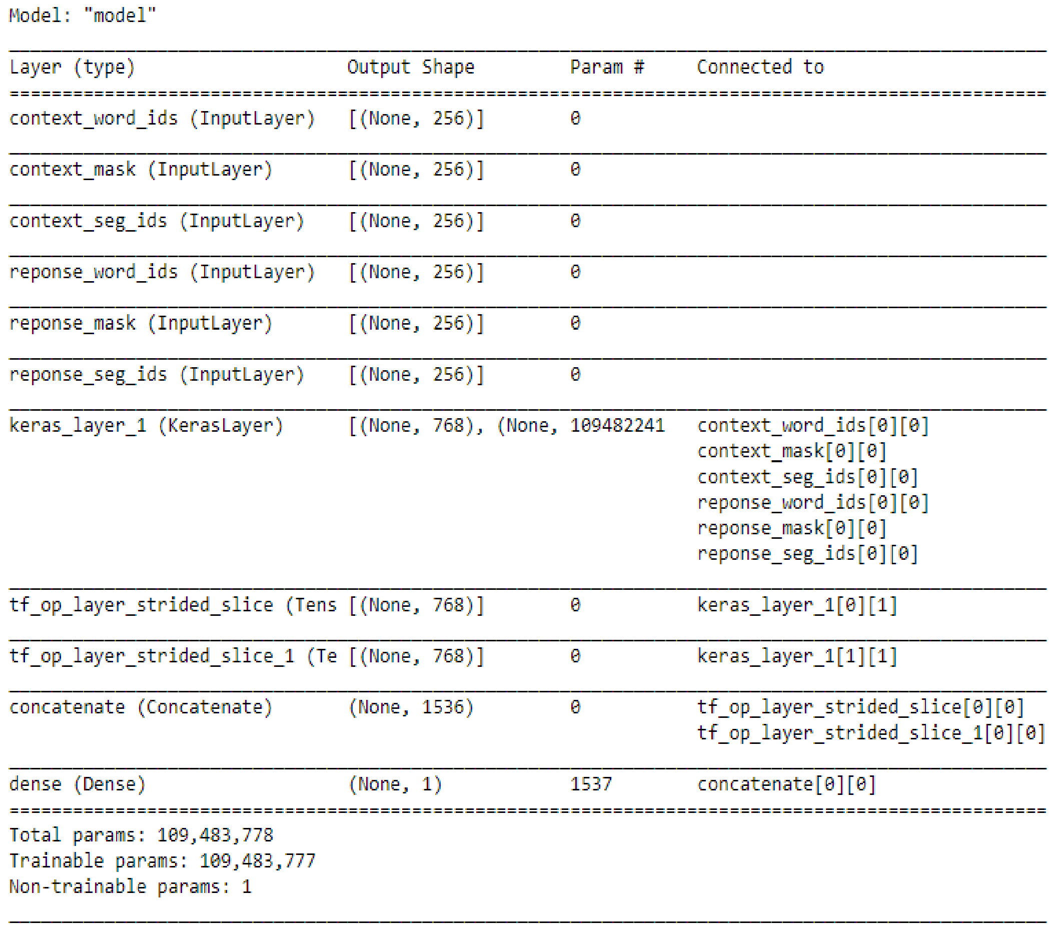
```
test_response = encode(test.response, tokenizer, max_len=max_len)
test_context = encode(test.context, tokenizer, max_len=max_len)
```

- Build Model with inputs as an array of context and response input word ids, mask and segment ids and output layer is a simple neural network with 1 neuron and activation function : sigmoid which is used for classification.
- **Hyperparameters**: Optimizer : Adam learning rate : 1e-6 , Loss : Binary Cross Entropy



```
def build_model(bert_layer, max_len=512):


    # input word ids,masks,segment ids for context
    # Refered to https://www.analyticsvidhya.com/blog/2020/10/simple-text-multi-classification-task-using-keras-bert/
    context_word_ids = Input(shape=(max_len,), dtype=tf.int32, name="context_word_ids")
    context_mask = Input(shape=(max_len,), dtype=tf.int32, name="context_mask")
    context_seg_ids = Input(shape=(max_len,), dtype=tf.int32, name="context_seg_ids")
    context_pooled_output , context_seq_output = bert_layer([context_word_ids, context_mask, context_seg_ids])
    context_clf_output = context_seq_output[:, 0, :]

    # input word ids,masks,segment ids for response
    reponse_word_ids = Input(shape=(max_len,), dtype=tf.int32, name="reponse_word_ids")
    reponse_mask = Input(shape=(max_len,), dtype=tf.int32, name="reponse_mask")
    reponse_seg_ids = Input(shape=(max_len,), dtype=tf.int32, name="reponse_seg_ids")
    response_pooled_output , reponse_seq_output = bert_layer([reponse_word_ids, reponse_mask, reponse_seg_ids])
    reponse_clf_output = reponse_seq_output[:, 0, :]

    #concatenation layer of outputs of bert layer for context and response
    conclayer=concatenate([context_clf_output,reponse_clf_output])
    # give conclayer as input to a simple dense network with 1 neuron with activation function sigmoid
    out = Dense(1, activation='sigmoid')(conclayer)

    #set keras model with inputs as an array of context and response input word ids,mask and segment ids
    model = Model(inputs=[context_word_ids, context_mask, context_seg_ids,reponse_word_ids,reponse_mask,reponse_seg_ids], outputs=out)

    #Model compilation
    #Used adam optimizer with learning rate 1e-6, used binary cross entropy loss as its a binary classification problem
    #Calculating accuracy,f1,precision and recall metrics for each run
    model.compile(Adam(1e-6), loss='binary_crossentropy', metrics=['acc',f1_m,precision_m, recall_m])

    return model
```

- **Model layout** :



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
context_word_ids (InputLayer)   [(None, 256)]        0
__________________________________________________________________________________________________
context_mask (InputLayer)       [(None, 256)]        0
__________________________________________________________________________________________________
context_seg_ids (InputLayer)    [(None, 256)]        0
__________________________________________________________________________________________________
reponse_word_ids (InputLayer)   [(None, 256)]        0
__________________________________________________________________________________________________
reponse_mask (InputLayer)       [(None, 256)]        0
__________________________________________________________________________________________________
reponse_seg_ids (InputLayer)    [(None, 256)]        0
__________________________________________________________________________________________________
keras_layer_1 (KerasLayer)      [(None, 768), (None, 109482241   context_word_ids[0][0]
                                                                 context_mask[0][0]
                                                                 context_seg_ids[0][0]
                                                                 reponse_word_ids[0][0]
                                                                 reponse_mask[0][0]
                                                                 reponse_seg_ids[0][0]
__________________________________________________________________________________________________
tf_op_layer_strided_slice (Tens [(None, 768)]        0           keras_layer_1[0][1]
__________________________________________________________________________________________________
tf_op_layer_strided_slice_1 (Te [(None, 768)]        0           keras_layer_1[1][1]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 1536)         0           tf_op_layer_strided_slice[0][0]
                                                                 tf_op_layer_strided_slice_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 1)            1537        concatenate[0][0]
==================================================================================================
Total params: 109,483,778
Trainable params: 109,483,777
Non-trainable params: 1
__________________________________________________________________________________________________
```

- **Training the model :**
  train_input (Array of train_context and train_reponse with in-turn contains input word ids,mask and segment ids)
  val_input (Array of val_context and val_reponse with in-turn contains input word ids,mask and segment ids)
  used **epochs = 3** and **batch_size = 3**
  Process takes around 30-40 min in Collaboratory using GPU.

```
train_history = model.fit(
    train_input, y_train,
    validation_data =(val_input,y_val),
    epochs=3,
    batch_size=3)
```

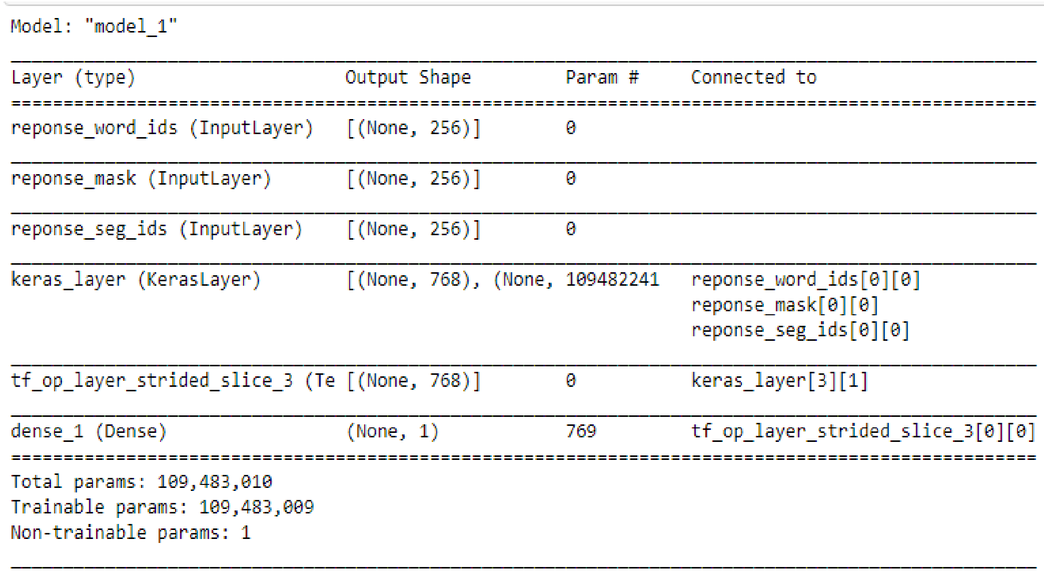
- **Predictions:**
  Predictions are saved in **answer.txt** with id and target label.

RESULTS:

Got an F1 score of **0.7402464065708418** using this approach beating the base line score.

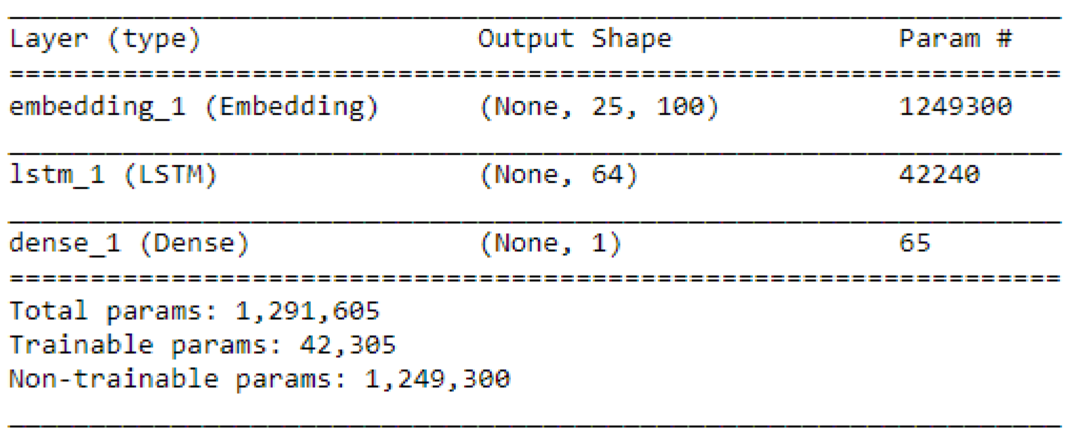
OTHER APPROACHES :

- Same Bert process with response only input without context.(F1score: **0.6905005107252298**



```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
reponse_word_ids (InputLayer)   [(None, 256)]        0
__________________________________________________________________________________________________
reponse_mask (InputLayer)       [(None, 256)]        0
__________________________________________________________________________________________________
reponse_seg_ids (InputLayer)    [(None, 256)]        0
__________________________________________________________________________________________________
keras_layer (KerasLayer)        [(None, 768), (None, 109482241   reponse_word_ids[0][0]
                                                                 reponse_mask[0][0]
                                                                 reponse_seg_ids[0][0]
__________________________________________________________________________________________________
tf_op_layer_strided_slice_3 (Te [(None, 768)]        0           keras_layer[3][1]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            769         tf_op_layer_strided_slice_3[0][0]
==================================================================================================
Total params: 109,483,010
Trainable params: 109,483,009
Non-trainable params: 1
__________________________________________________________________________________________________
```

- Simple LSTM with Stanford Glove embeddings (F1 score : **0.6645126548196015**)



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 25, 100)           1249300
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                42240
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,291,605
Trainable params: 42,305
Non-trainable params: 1,249,300
_________________________________________________________________
```

REFERENCES:

- Documentation of BERT on TFHub
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- https://www.analyticsvidhya.com/blog/2020/10/simple-text-multi-classification-task-using-keras-bert/
- https://datascience.stackexchange.com/questions/45165/how-to-get-accuracy-f1-precision-and-recall-for-a-keras-model