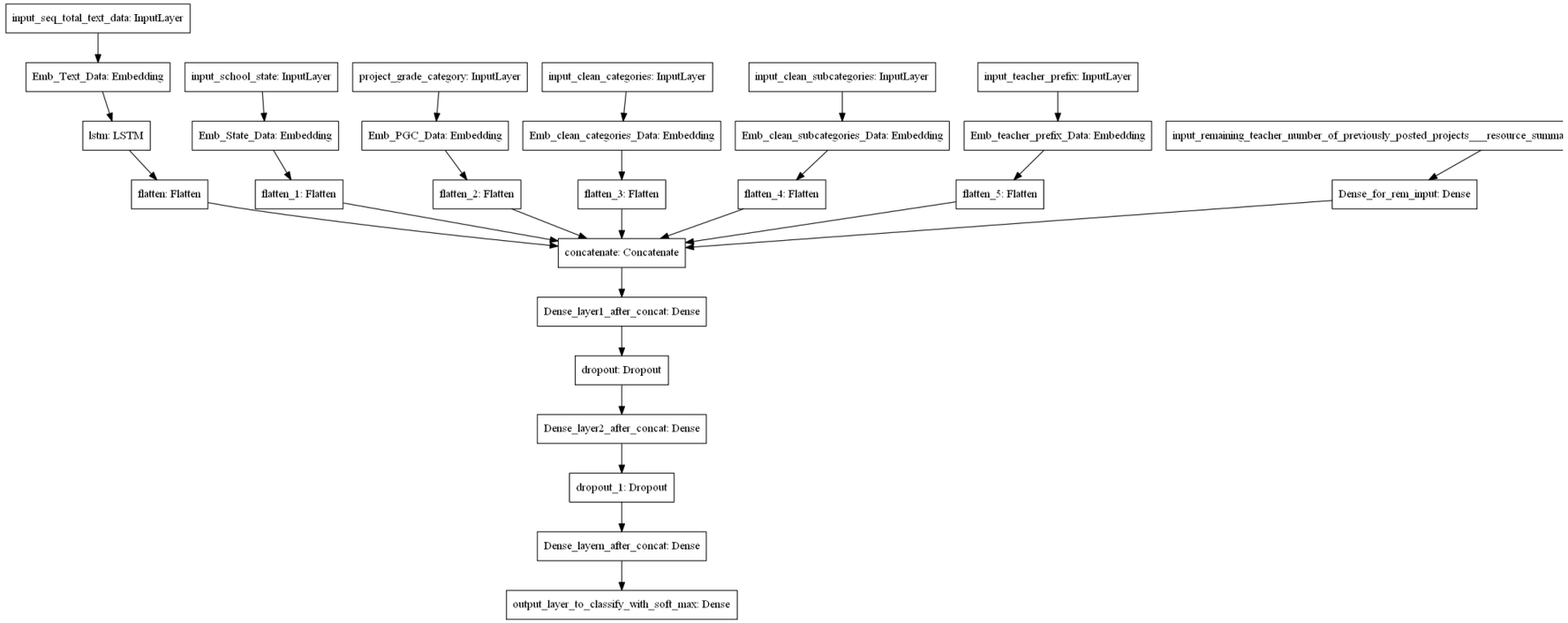
Assignment : 14

1. Preprocess all the Data we have in DonorsChoose Dataset (https://drive.google.com/drive/folders/1MIwK7BQMev8f5CbDDVNLPain.csv
2. Combine 4 essay's into one column named - 'preprocessed_essays'.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' (https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics) as a metric. chdatascience.stackexchange.com/a/20192) for using auc as a metric
5. You are free to choose any number of layers/hiddden units but you have to use same type of architectures shown below.
6. You can use any one of the optimizers and choice of Learning rate and momentum, resources: cs231n class notes (http://cural-networks-3/), cs231n class video (https://www.youtube.com/watch?v=hd_KFJ5ktUc).
7. For all the model's use TensorBoard (https://www.youtube.com/watch?v=2U6Jl7oqRkM) and plot the Metric value and Loss wibmitting, take a screenshot of plots and include those images in .ipynb notebook and PDF.
8. Use Categorical Cross Entropy as Loss to minimize.

Model-1

Build and Train deep neural network as shown below



ref: https://i.imgur.com/w395Yk9.png

- **Input_seq_total_text_data** --- You have to give Total text data columns. After this use the Embedding layer to get word vectors. Use given predefined g train any word vectors. After this use LSTM and get the LSTM output and Flatten that output.
- **Input_school_state** --- Give 'school_state' column as input to embedding layer and Train the Keras Embedding layer.
- **Project_grade_category** --- Give 'project_grade_category' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_categories** --- Give 'input_clean_categories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_clean_subcategories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_teacher_prefix' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_remaining_teacher_number_of_previously_posted_projects._resource_summary_contains_numerical_digits._price._quantity** ---concat and add a Dense layer after that.

- For LSTM, you can choose your sequence padding methods on your own or you can train your LSTM without padding, there is no restriction on that.

Below is an example of embedding layer for a categorical columns. In below code all are dummy values, we gave only for referance.

In [ ]:
```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount("/gdrive", force_remount=True).
/gdrive
```

In [ ]:
```
# Required Packages
import os
import numpy as np
import keras
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.regularizers import l2,l1_l2
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.layers import Dense, Dropout, Activation,Flatten,Conv1D,MaxPool1D,Input,BatchNormalization

from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import concatenate
from tqdm import tqdm
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding,GlobalMaxPooling1D,LSTM
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, TensorBoard
import tensorflow as tf
import datetime
import random as rn
from sklearn import metrics
from sklearn.metrics import auc as AUC_score
```

In [ ]:
```
df= pd.read_csv("/gdrive/My Drive/LSTM/LSTM Assignment/preprocessed_data.csv")
df.columns
```

Out[3]:
```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

In [ ]:
```
df.head(2)
```

Out[10]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subcate
0	ca	mrs	grades_prek_2	53	1	math_science	appliedsc health_life
1	ut	ms	grades_3_5	4	1	specialneeds	specia

In [ ]:
```
# Prepare Dependent and Independent varaiable
# split into input (X) and output (y) variables
X  =  df.drop(['project_is_approved'],axis=1)
y  =  df['project_is_approved']
```

In [ ]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 8 columns):
 #   Column                                        Non-Null Count   Dtype  
---  ------                                        --------------   -----  
 0   school_state                                  109248 non-null  object 
 1   teacher_prefix                                109248 non-null  object 
 2   project_grade_category                        109248 non-null  object 
 3   teacher_number_of_previously_posted_projects  109248 non-null  int64  
 4   clean_categories                              109248 non-null  object 
 5   clean_subcategories                           109248 non-null  object 
 6   essay                                         109248 non-null  object 
 7   price                                         109248 non-null  float64
dtypes: float64(1), int64(1), object(6)
memory usage: 6.7+ MB
```

In [ ]:
```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y,random_state=123)
```

In [ ]:
```
X_train.head(2)
```

Out[6]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	
88868	nc	ms	grades_3_5	8	math_science	health_lifescience mathematics	disad
30485	ca	ms	grades_prek_2	2	literacy_language appliedlearning	literature_writing other	eve

In [ ]:
```

```

In [ ]:

```
# First Essay Tokenizsation -  word i have used 300 dimension
def text_encoding(train,test):
    texts = train.essay.values.tolist()
    texts_train = test.essay.values.tolist()
    # Tokenizer
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    sequences_test = tokenizer.texts_to_sequences(texts_train)

    ''' tokenizer.word_index -  dic - key word value - index value'''
    word_index = tokenizer.word_index
    print('Found %s unique tokens.' % len(word_index))

    '''tokenizer.index_docs - Key index value value - occurances in # of documents'''
    index_doc_count = tokenizer.index_docs
    print('Index count',len(index_doc_count))

    # To select the best MAX_SEQ_LENGTH
    val_append = []
    for i in tqdm(range(len(sequences))):
        val_append.append(len(sequences[i]))

    print('Maximum length of essay in corpus',max(val_append))

    MAX_SEQUENCE_LENGTH = 500
    MAX_SEQUENCE_LENGTH
    print(MAX_SEQUENCE_LENGTH)

    data_essay = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH,padding='pre')
    data_essay_test = pad_sequences(sequences_test, maxlen=MAX_SEQUENCE_LENGTH,padding='pre')
    print('Shape of Essay Embeddin',data_essay.shape)


    BASE_DIR = ''
    GLOVE_DIR = os.path.join(BASE_DIR, r'/gdrive/My Drive/LSTM/LSTM Assignment')

    embeddings_index = {}
    with open(os.path.join(GLOVE_DIR, 'glove.6B.300d.txt'),encoding="utf8") as f:
        for line in tqdm(f):
            word, coefs = line.split(maxsplit=1)
            coefs = np.fromstring(coefs, 'f', sep=' ')
```

```
        embeddings_index[word] = coefs

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((len(word_index)+1, 300))
for word, i in tqdm(tokenizer.word_index.items()):

    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

print('Length of embedding matrix', len(embedding_matrix))
EMBEDDING_DIM = 300

embedding_layer_essay = Embedding(len(tokenizer.word_index)+1,
                    EMBEDDING_DIM,
                    weights=[embedding_matrix],
                    input_length=MAX_SEQUENCE_LENGTH,
                    trainable=False)

return embedding_layer_essay,data_essay,data_essay_test
```

In [ ]:
```
text_doc_embedding = text_encoding(X_train,X_test)
embedding_layer_text = text_doc_embedding[0]
print('TEXT EMBEDDING HAS COMPLETED')
x_train_text = text_doc_embedding[1]
print('shape of after text embedding',x_train_text.shape)
x_test_text = text_doc_embedding[2]
print('shape of after text embedding',x_test_text.shape)
```

```
100%|██████████| 73196/73196 [00:00<00:00, 2144647.40it/s]

Found 48341 unique tokens.
Index count 48341
Maximum length of essay in corpus 339
500


0it [00:00, ?it/s]

Shape of Essay Embeddin (73196, 500)

400001it [00:25, 15827.70it/s]
100%|██████████| 48341/48341 [00:00<00:00, 506342.74it/s]

Length of embedding matrix 48342
TEXT EMBEDDING HAS COMPLETED
shape of after text embedding (73196, 500)
shape of after text embedding (36052, 500)
```

In [ ]:
```
x_train_text
```

Out[10]:
```
array([[   0,    0,    0, ..., 2191,  578,   13],
       [   0,    0,    0, ..., 1169,   88,   13],
       [   0,    0,    0, ...,  322, 1245,   13],
       ...,
       [   0,    0,    0, ...,   27, 1161, 3123],
       [   0,    0,    0, ...,  164,  360,   13],
       [   0,    0,    0, ...,  316,  569,   13]], dtype=int32)
```

CATEGORICAL FEATURE PREPROCESSING

SCHOOL STATE

In [ ]:
```
def embedding_state(train,test):
      # school state
    len(set(df.school_state.values.tolist()))
    trainCategorical_state = train.school_state.values.tolist()
    testCategorical_state = test.school_state.values.tolist()
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(trainCategorical_state)
    trainCategorical_state = tokenizer.texts_to_sequences(trainCategorical_state)
    testCategorical_state = tokenizer.texts_to_sequences(testCategorical_state)

    val_append = []
    for i in tqdm(range(len(trainCategorical_state))):
        val_append.append(len(trainCategorical_state[i]))

    MAX_SEQUENCE_LENGTH_category = max(val_append)
    print('Embedding max_length',MAX_SEQUENCE_LENGTH_category)

    #padding
    # Train data
    max_length = 1
    x_train_sch_state = pad_sequences(trainCategorical_state, maxlen = max_length, padding='post')
    x_train_sch_state.shape
    # Test data
    x_test_sch_state = pad_sequences(testCategorical_state, maxlen = max_length, padding='post')
    x_test_sch_state.shape
    print('One hot encoding done - school state')

    unique_state = len(tokenizer.word_index)
    print('length uniques',unique_state)
    MAX_SEQUENCE_LENGTH_state = 1
    EMBEDDING_DIM_state = 2
    embedding_layer_state = Embedding(unique_state+1,
                              EMBEDDING_DIM_state,
                              weights=None,
                              input_length=MAX_SEQUENCE_LENGTH_state,
                              trainable=True)

    return x_train_sch_state,x_test_sch_state,embedding_layer_state
```

In [ ]:

```
State_embedding = embedding_state(X_train,X_test)
x_train_sch_state = State_embedding[0]
print('State embedding shape train',x_train_sch_state.shape)
x_test_sch_state = State_embedding[1]
print('State embedding shape test',x_test_sch_state.shape)
state_layer_model = State_embedding[2]
```

```
100%|██████████| 73196/73196 [00:00<00:00, 2187246.37it/s]

Embedding max_length 1
One hot encoding done - school state
length uniques 51
State embedding shape train (73196, 1)
State embedding shape test (36052, 1)
```

PROJECT GRADE

In [ ]:

```
def embedding_grade(train,test):
    trainCategorical_grade = train.project_grade_category.values.tolist()
    testCategorical_grade = test.project_grade_category.values.tolist()
    tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
    tokenizer.fit_on_texts(trainCategorical_grade)
    trainCategorical_grade = tokenizer.texts_to_sequences(trainCategorical_grade)
    testCategorical_grade = tokenizer.texts_to_sequences(testCategorical_grade)

    val_append = []
    for i in tqdm(range(len(trainCategorical_grade))):
        val_append.append(len(trainCategorical_grade[i]))

    MAX_SEQUENCE_LENGTH_category = max(val_append)
    print('Embedding max_length',MAX_SEQUENCE_LENGTH_category)

    #padding
    # Train data
    max_length = 1
    x_train_grade = pad_sequences(trainCategorical_grade, maxlen = max_length, padding='post')
    x_train_grade.shape
    # Test data
    x_test_grade = pad_sequences(testCategorical_grade, maxlen = max_length, padding='post')
    x_test_grade.shape

    print('One hot encoding done - Project grade')

    #Embedding _project grade
    unique_grade = len(tokenizer.word_index)
    print('length uniques',unique_grade)
    MAX_SEQUENCE_LENGTH_grade = 1
    EMBEDDING_DIM_grade = 2
    embedding_layer_grade= Embedding(unique_grade+1,
                            EMBEDDING_DIM_grade,
                            weights=None,
                            input_length=MAX_SEQUENCE_LENGTH_grade,
                            trainable=True)

    return x_train_grade,x_test_grade,embedding_layer_grade
```

In [ ]:

```
Grade_embedding = embedding_grade(X_train,X_test)
x_train_proj_grade = Grade_embedding[0]
print('Grade embedding shape train',x_train_proj_grade.shape)
x_test_proj_grade = Grade_embedding[1]
print('Grade embedding shape test',x_test_proj_grade.shape)
grade_layer_model = Grade_embedding[2]
```

```
100%|██████████| 73196/73196 [00:00<00:00, 2336869.84it/s]

Embedding max_length 1
One hot encoding done - Project grade
length uniques 4
Grade embedding shape train (73196, 1)
Grade embedding shape test (36052, 1)
```

In [ ]:

CLEAN CATEGORICAL

In [ ]:

```
def embedding_cleab_cat(train,test):
    trainCategorical_clean_cat = train.clean_categories.values.tolist()
    testCategorical_clean_cat = test.clean_categories.values.tolist()
    tokenizer = Tokenizer(num_words=None, filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
    tokenizer.fit_on_texts(trainCategorical_clean_cat)
    trainCategorical_clean_cat = tokenizer.texts_to_sequences(trainCategorical_clean_cat)
    testCategorical_clean_cat = tokenizer.texts_to_sequences(testCategorical_clean_cat)

    val_append = []
    for i in tqdm(range(len(trainCategorical_clean_cat))):
        val_append.append(len(trainCategorical_clean_cat[i]))

    MAX_SEQUENCE_LENGTH_category = max(val_append)
    print('Embedding max_length',MAX_SEQUENCE_LENGTH_category)
    #padding
    # Train data
    max_length = 3
    x_train_clean_cat = pad_sequences(trainCategorical_clean_cat, maxlen = max_length, padding='post')
    x_train_clean_cat.shape
    # Test data
    x_test_clean_cat = pad_sequences(testCategorical_clean_cat, maxlen = max_length, padding='post')
    x_test_clean_cat.shape

    print('One hot encoding done - Clean category')

     #Embedding _category
    unique_category = len(tokenizer.word_index)
    print('length uniques',unique_category)
    EMBEDDING_DIM_cat = 2
    embedding_layer_cat= Embedding(unique_category+1,
                            EMBEDDING_DIM_cat,
                            weights=None,
                            input_length=max_length,
                            trainable=True)

    return x_train_clean_cat,x_test_clean_cat,embedding_layer_cat
```

In [ ]:

```
Clean_cat_embedding = embedding_cleab_cat(X_train,X_test)
x_train_clean_cat  = Clean_cat_embedding[0]
print('Clean_cat embedding shape train',x_train_clean_cat.shape)
x_test_clean_cat  = Clean_cat_embedding[1]
print('Clean_cat embedding shape test',x_test_clean_cat.shape)
cat_layer_model = Clean_cat_embedding[2]
```

```
100%|██████████| 73196/73196 [00:00<00:00, 2337350.21it/s]

Embedding max_length 3
One hot encoding done - Clean category
length uniques 9
Clean_cat embedding shape train (73196, 3)
Clean_cat embedding shape test (36052, 3)
```

CLEAN SUB-CATEGORICAL

In [ ]:

```
def embedding_clean_sub_cat(train,test):
    #Clean  subcategory
    trainCategorical_clean_sub_cat = train.clean_subcategories.values.tolist()
    testCategorical_clean_sub_cat = test.clean_subcategories.values.tolist()
    tokenizer = Tokenizer(num_words=None, filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
    tokenizer.fit_on_texts(trainCategorical_clean_sub_cat)
    trainCategorical_clean_sub_cat = tokenizer.texts_to_sequences(trainCategorical_clean_sub_cat)
    testCategorical_clean_sub_cat = tokenizer.texts_to_sequences(testCategorical_clean_sub_cat)

    val_append = []
    for i in tqdm(range(len(trainCategorical_clean_sub_cat))):
        val_append.append(len(trainCategorical_clean_sub_cat[i]))

    MAX_SEQUENCE_LENGTH_category = max(val_append)
    print('Embedding max_length',MAX_SEQUENCE_LENGTH_category)

    #padding
    # Train data
    max_length = 3
    x_train_clean_sub_cat = pad_sequences(trainCategorical_clean_sub_cat, maxlen = max_length, padding='post')
    x_train_clean_sub_cat.shape
    # Test data
    x_test_clean_sub_cat = pad_sequences(testCategorical_clean_sub_cat, maxlen = max_length, padding='post')
    x_test_clean_sub_cat.shape

    print('One hot encoding done - Clean  subcategory')

      #Embedding _Subcategory
    unique_Subcategory = len(tokenizer.word_index)
    print('length uniques',unique_Subcategory)
    EMBEDDING_DIM_Subcategory = 2
    embedding_layer_Subcategory= Embedding(unique_Subcategory+1,
                            EMBEDDING_DIM_Subcategory,
                            weights=None,
                            input_length=max_length,
                            trainable=True)

    return x_train_clean_sub_cat,x_test_clean_sub_cat,embedding_layer_Subcategory
```

In [ ]:

```
Clean_Subcat_embedding = embedding_clean_sub_cat(X_train,X_test)
x_train_Subclean_cat  = Clean_Subcat_embedding[0]
print('Clean_sub_cat embedding shape train',x_train_Subclean_cat.shape)
x_test_Subclean_cat  = Clean_Subcat_embedding[1]
print('Clean_sub_cat embedding shape test',x_test_Subclean_cat.shape)
sub_cat_layer_model = Clean_Subcat_embedding[2]
```

```
100%|██████████| 73196/73196 [00:00<00:00, 1488587.45it/s]

Embedding max_length 3
One hot encoding done - Clean  subcategory
length uniques 30
Clean_sub_cat embedding shape train (73196, 3)
Clean_sub_cat embedding shape test (36052, 3)
```

TEACHER PREFIX

In [ ]:

```
def embedding_Teacher_prefix(train,test):
    len(set(df.teacher_prefix.values.tolist()))
    trainCategorical_prefix = train.teacher_prefix.values.tolist()
    testCategorical_prefix = test.teacher_prefix.values.tolist()
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(trainCategorical_prefix)
    trainCategorical_prefix = tokenizer.texts_to_sequences(trainCategorical_prefix)
    testCategorical_prefix = tokenizer.texts_to_sequences(testCategorical_prefix)

    val_append = []
    for i in tqdm(range(len(trainCategorical_prefix))):
        val_append.append(len(trainCategorical_prefix[i]))

    MAX_SEQUENCE_LENGTH_category = max(val_append)
    print('Embedding max_length',MAX_SEQUENCE_LENGTH_category)

    #padding
    # Train data
    max_length = 1
    x_train_teacher_prefix = pad_sequences(trainCategorical_prefix, maxlen = max_length, padding='post')
    x_train_teacher_prefix.shape
    # Test data
    x_test_teacher_prefix = pad_sequences(testCategorical_prefix, maxlen = max_length, padding='post')
    x_test_teacher_prefix.shape

    print('One hot encoding done - Teacher prefix')
    #Embedding _Teacher prefix
    unique_prefix = len(tokenizer.word_index)
    print('length uniques',unique_prefix)
    MAX_SEQUENCE_LENGTH_prefix = 1
    EMBEDDING_DIM_teacher_prefix = 2
    embedding_layer_prefix= Embedding(unique_prefix+1,
                            EMBEDDING_DIM_teacher_prefix,
                            weights=None,
                            input_length=MAX_SEQUENCE_LENGTH_prefix,
                            trainable=True)

    return x_train_teacher_prefix,x_test_teacher_prefix,embedding_layer_prefix
```

In [ ]:
```
Teacher_pre_embedding = embedding_Teacher_prefix(X_train,X_test)
x_train_teach_pre  = Teacher_pre_embedding[0]
print('Teacher_prefix embedding shape train',x_train_teach_pre.shape)
x_test_teach_pre  = Teacher_pre_embedding[1]
print('Teacher_prefix embedding shape test',x_test_teach_pre.shape)
prefix_layer_model = Teacher_pre_embedding[2]
```

```
100%|██████████| 73196/73196 [00:00<00:00, 2351133.24it/s]

Embedding max_length 1
One hot encoding done - Teacher prefix
length uniques 5
Teacher_prefix embedding shape train (73196, 1)
Teacher_prefix embedding shape test (36052, 1)
```

NUMERICAL FEATURE PRE-PROCESSING

TEACHER PREVIOUS SUBMITTED PROJECTS COUNT & PRICE

In [ ]:
```
def Numerical_preprocessing(train,test):
  continuous = ["teacher_number_of_previously_posted_projects", "price"]
  cs = MinMaxScaler()
  trainContinuous = cs.fit_transform(train[continuous])
  testContinuous = cs.transform(test[continuous])

  return trainContinuous,testContinuous
```

In [ ]:
```
numerical_feature = Numerical_preprocessing(X_train,X_test)
x_train_previously_posted_projects  = numerical_feature[0]
x_test_previously_posted_projects  =  numerical_feature[1]
```

In [ ]:
```
final_x_train  = [x_train_text,x_train_sch_state,x_train_proj_grade,x_train_clean_cat,x_train_Subclean_cat,x_train_teach_pre,x
final_x_test  = [x_test_text,x_test_sch_state,x_test_proj_grade,x_test_clean_cat,x_test_Subclean_cat,x_test_teach_pre,x_test_p
```

In [ ]:
```
len(final_x_train)
```

Out[18]: 7

CREATE MODEL

In [ ]:

```
# Create Model
os.environ['PYTHONHASHSEED'] = '0'

##https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development
## Have to clear the session. If you are not clearing, Graph will create again and again and graph size will increses.
## Varibles will also set to some value from before session
tf.keras.backend.clear_session()

## Set the random seed values to regenerate the model.
np.random.seed(0)
rn.seed(0)

#Input layers
# Text Data Input layer
input_layer_essay = Input(shape=(500,), dtype='int32',name='Essay_text')
e =embedding_layer_text(input_layer_essay)
lstm = LSTM(128,return_sequences=True)(e)
X1 = Flatten(data_format='channels_last',name='Flatten_essay')(lstm)

# Second Input layer - state
input_layer_state = Input(shape=(1,), dtype='int32',name='School_Sate')
e_state = state_layer_model(input_layer_state)
X2 = Flatten(data_format='channels_last',name='Flatten_state')(e_state)

# Third Input layer - grade
input_layer_grade= Input(shape=(1,), dtype='int32',name='Project_grade')
e_grade = grade_layer_model(input_layer_grade)
X3= Flatten(data_format='channels_last',name='Flatten_grade')(e_grade)


# Fourth  Input layer - category
input_layer_cat= Input(shape=(3,), dtype='int32',name='Project_category')
e_cat = cat_layer_model(input_layer_cat)
X4= Flatten(data_format='channels_last',name='Flatten_category')(e_cat)


# fifth  Input layer - sub_category
input_layer_sub_category= Input(shape=(3,), dtype='int32',name='SubCategory')
e_sub_category = sub_cat_layer_model(input_layer_sub_category)
X5= Flatten(data_format='channels_last',name='Flatten_sub_category')(e_sub_category)

# Sixth Input layer - Teacher prefix
```

```
input_layer_teacher_prefix = Input(shape=(1,), dtype='int32',name='Teacher_prefix')
e_teacher_prefix= prefix_layer_model(input_layer_teacher_prefix)
X6= Flatten(data_format='channels_last',name='Flatten_teacher_prefix')(e_teacher_prefix)

#input 7
input_7 = Input(shape=(2,))
X7 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(inp

'''input_8 = Input(shape=(1,))
X8 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(inp

merge = concatenate([X7,X8])'''

concat = concatenate([X1,X2,X3,X4,X5,X6,X7],axis=1)

# Fully connected Dense Layer

x = Dense(64, activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(cor
x = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.2)(x)
x = BatchNormalization()(x)
x = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.2)(x)
output = Dense(2, activation = 'softmax')(x)


all_inputs =  [input_layer_essay,input_layer_state,input_layer_grade,input_layer_cat,input_layer_sub_category,input_layer_tea
model = Model(inputs=all_inputs,outputs=output)
model.summary()
```

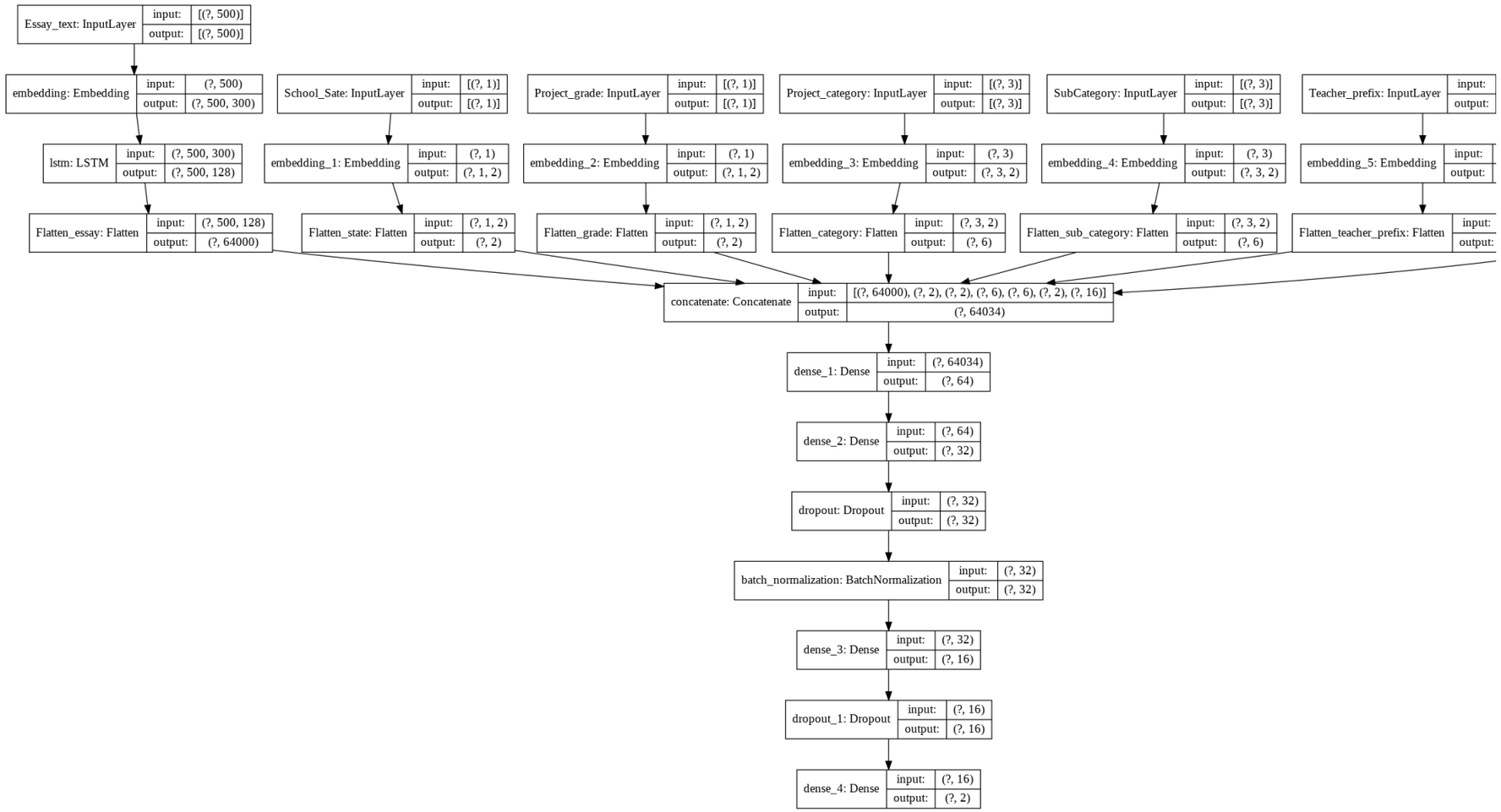
```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Essay_text (InputLayer)         [(None, 500)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 500, 300)     14512200    Essay_text[0][0]
__________________________________________________________________________________________________
School_Sate (InputLayer)        [(None, 1)]          0
__________________________________________________________________________________________________
Project_grade (InputLayer)      [(None, 1)]          0
__________________________________________________________________________________________________
```

Project_category (InputLayer)	[(None, 3)]	0	
SubCategory (InputLayer)	[(None, 3)]	0	
Teacher_prefix (InputLayer)	[(None, 1)]	0	
lstm (LSTM)	(None, 500, 128)	219648	embedding[0][0]
embedding_1 (Embedding)	(None, 1, 2)	104	School_Sate[0][0]
embedding_2 (Embedding)	(None, 1, 2)	10	Project_grade[0][0]
embedding_3 (Embedding)	(None, 3, 2)	20	Project_category[0][0]
embedding_4 (Embedding)	(None, 3, 2)	62	SubCategory[0][0]
embedding_5 (Embedding)	(None, 1, 2)	12	Teacher_prefix[0][0]
input_1 (InputLayer)	[(None, 2)]	0	
Flatten_essay (Flatten)	(None, 64000)	0	lstm[0][0]
Flatten_state (Flatten)	(None, 2)	0	embedding_1[0][0]
Flatten_grade (Flatten)	(None, 2)	0	embedding_2[0][0]
Flatten_category (Flatten)	(None, 6)	0	embedding_3[0][0]
Flatten_sub_category (Flatten)	(None, 6)	0	embedding_4[0][0]
Flatten_teacher_prefix (Flatten	(None, 2)	0	embedding_5[0][0]
dense (Dense)	(None, 16)	48	input_1[0][0]
concatenate (Concatenate)	(None, 64034)	0	Flatten_essay[0][0] Flatten_state[0][0] Flatten_grade[0][0] Flatten_category[0][0] Flatten_sub_category[0][0] Flatten_teacher_prefix[0][0] dense[0][0]
dense_1 (Dense)	(None, 64)	4098240	concatenate[0][0]

```
dense_2 (Dense)                 (None, 32)           2080        dense_1[0][0]
_________________________________________________________________________________________________
dropout (Dropout)               (None, 32)           0           dense_2[0][0]
_________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 32)           128         dropout[0][0]
_________________________________________________________________________________________________
dense_3 (Dense)                 (None, 16)           528         batch_normalization[0][0]
_________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 16)           0           dense_3[0][0]
_________________________________________________________________________________________________
dense_4 (Dense)                 (None, 2)            34          dropout_1[0][0]
=================================================================================================
Total params: 18,833,114
Trainable params: 4,320,850
Non-trainable params: 14,512,264
_________________________________________________________________________________________________
```

In [ ]:
```
dot_img_file = '/tmp/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

Out[20]:



In [ ]:
```
# Conevrt Target in to categorical
new_y_train = to_categorical(y_train)
new_y_test = to_categorical(y_test)
```

In [ ]:
```
filepath="/gdrive/My Drive/LSTM/LSTM Assignment/save_model/best_model-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc_score',  verbose=1, save_best_only=True, mode='max')
```

In [ ]:
```

ACCURACY_THRESHOLD_test = 0.75
class myCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs={}):

      if(logs.get('val_auc_score') > ACCURACY_THRESHOLD_test):
        print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD_test*100))
        self.model.stop_training = True



early_stop_auc_scores = myCallback()
```

In [ ]:
```
# Early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='auto',
    baseline=None, restore_best_weights=False
)
```

In [ ]:
```
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/LSTM/LSTM Assignment/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=0, write_graph=True)
```

In [ ]:
```
folder_name
```

Out[26]:
```
'20201001-050259'
```

In [ ]:
```
def auc_score(y_true, y_pred):

    auc_scoree =  tf.compat.v1.py_func(metrics.roc_auc_score, (y_true, y_pred), tf.double)
    return auc_scoree
```

In [ ]:
```
# compile
model.compile(loss='categorical_crossentropy',
            optimizer=keras.optimizers.Adam(lr=0.0008),
             metrics=[auc_score ])
```

In [ ]:
```
model.fit(final_x_train, new_y_train, epochs=40,verbose=1,batch_size=64, validation_data=(final_x_test, new_y_test),callbacks
```

```
Epoch 1/40
WARNING:tensorflow:From <ipython-input-27-af59d2ad3ac8>:3: py_func (from tensorflow.python.ops.script_ops) is deprecated and w
ture version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

   1/1144 [..............................] - ETA: 0s - loss: 1.0985 - auc_score: 0.5918WARNING:tensorflow:From /usr/local/lib/
s/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from tensorflow.python.eager.profiler) is deprecated and will be removed
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
   2/1144 [..............................] - ETA: 2:22 - loss: 1.2360 - auc_score: 0.4637WARNING:tensorflow:Callbacks method `
slow compared to the batch time (batch time: 0.0671s vs `on_train_batch_end` time: 0.1821s). Check your callbacks.
1144/1144 [==============================] - ETA: 0s - loss: 0.4643 - auc_score: 0.6232
Epoch 00001: val_auc_score improved from -inf to 0.72049, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/bes
1144/1144 [==============================] - 62s 54ms/step - loss: 0.4643 - auc_score: 0.6232 - val_loss: 0.4135 - val_auc_sco
Epoch 2/40
1143/1144 [============================>.] - ETA: 0s - loss: 0.4066 - auc_score: 0.7084
Epoch 00002: val_auc_score improved from 0.72049 to 0.74139, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 61s 53ms/step - loss: 0.4066 - auc_score: 0.7083 - val_loss: 0.3877 - val_auc_sco
Epoch 3/40
1143/1144 [============================>.] - ETA: 0s - loss: 0.3952 - auc_score: 0.7345
Epoch 00003: val_auc_score improved from 0.74139 to 0.74772, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 62s 54ms/step - loss: 0.3952 - auc_score: 0.7346 - val_loss: 0.3858 - val_auc_sco
Epoch 4/40
1143/1144 [============================>.] - ETA: 0s - loss: 0.3859 - auc_score: 0.7555
Epoch 00004: val_auc_score improved from 0.74772 to 0.75390, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/

Reached 75.00% accuracy, so stopping training!!
1144/1144 [==============================] - 62s 54ms/step - loss: 0.3858 - auc_score: 0.7556 - val_loss: 0.3923 - val_auc_sco
```

Out[29]:

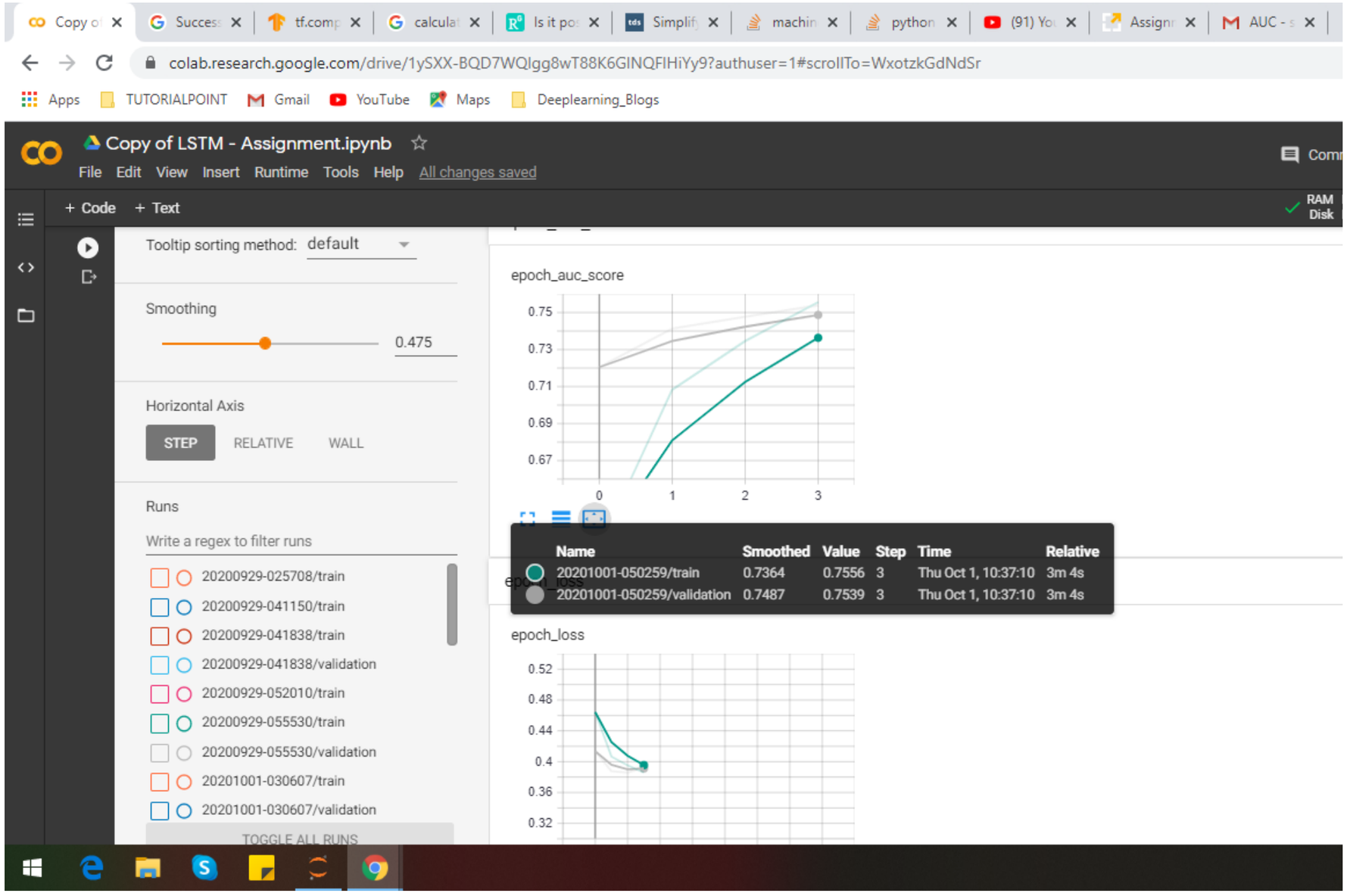
```
<tensorflow.python.keras.callbacks.History at 0x7f3f2b193278>
```

In [ ]:

In [ ]:
```
os.chdir('/gdrive/My Drive/LSTM/LSTM Assignment')
```

In [ ]:
```
%tensorboard --logdir logs/fit/
```

```
Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.
```


colab.research.google.com/drive/1ySXX-BQD7WQIgg8wT88K6GINQFIHiYy9?authuser=1#scrollTo=WxotzkGdNdSr
Copy of LSTM - Assignment.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
RAM Disk
Tooltip sorting method: default
Smoothing
0.475
Horizontal Axis
STEP RELATIVE WALL
Runs
Write a regex to filter runs
20200929-025708/train
20200929-041150/train
20200929-041838/train
20200929-041838/validation
20200929-052010/train
20200929-055530/train
20200929-055530/validation
20201001-030607/train
20201001-030607/validation
TOGGLE ALL RUNS
epoch_auc_score
0.75
0.73
0.71
0.69
0.67
0 1 2 3
Name Smoothed Value Step Time Relative
20201001-050259/train 0.7364 0.7556 3 Thu Oct 1, 10:37:10 3m 4s
20201001-050259/validation 0.7487 0.7539 3 Thu Oct 1, 10:37:10 3m 4s
epoch_loss
0.52
0.48
0.44
0.4
0.36
0.32

MODEL - 2

Use the same model as above but for 'input_seq_total_text_data' give only some words in the sentance not all the words. Filter the words as below.

In [ ]:
```
X_train, X_test, y_train, y_test
```

1. Train the TF-IDF on the Train data

2. Get the idf value for each word we have in the train data.

3. Remove the low idf value and high idf value words from our data. Do some analysis on the Idf values and based on those low and high threshold value. Because very frequent words and very very rare words don't give much information. (you can nd take only the idf scores within IQR range and corresponding words)

4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train on total data but in Model-2 train on ng some words based on IDF values)

In [ ]:
```
# Required Package for TFIDF
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [ ]:
```
# Encode Essay -  Set using BOW
print('Before encode shape of X_train : '+str(X_train.shape))
#print('Before encode shape of X_CV : '+str(X_cv.shape))
print('-'*110)

vectorizer = TfidfVectorizer(min_df=10,max_features=10000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data



# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)

#X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("-"*100)
```

```
Before encode shape of X_train : (73196, 8)
--------------------------------------------------------------------------------------------------------------
After vectorizations
(73196, 10000) (73196,)
(36052, 10000) (36052,)
----------------------------------------------------------------------------------------------------
```

In [ ]:
```
idf_value = vectorizer.idf_
df_idf_value = pd.DataFrame(data={'Feature_name': vectorizer.get_feature_names(),
                                  'IDF_value': idf_value})
print('shape of idf_value',df_idf_value.shape)
```

```
shape of idf_value (10000, 2)
```

In [ ]:
```
df_idf_value.tail(100)
```

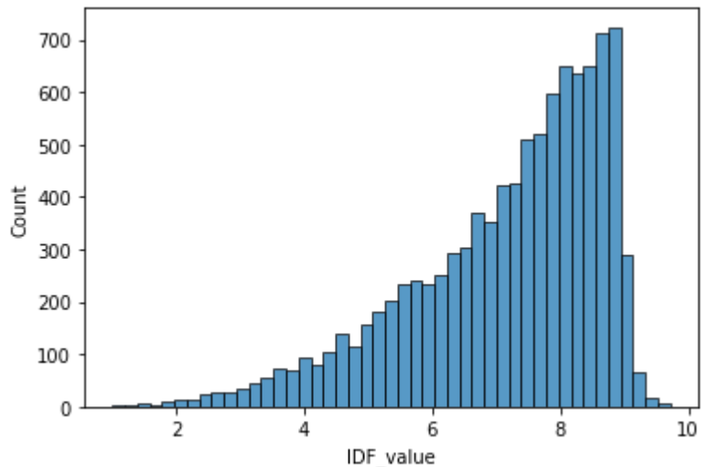
Out[20]:

	Feature_name	IDF_value
9900	workout	6.882790
9901	workouts	7.723573
9902	workplace	6.762830
9903	works	4.211349
9904	worksheet	6.816415
...	...	...
9995	zones	7.556519
9996	zoo	7.626199
9997	zoom	8.309089
9998	zoos	8.674549
9999	zumba	8.766923

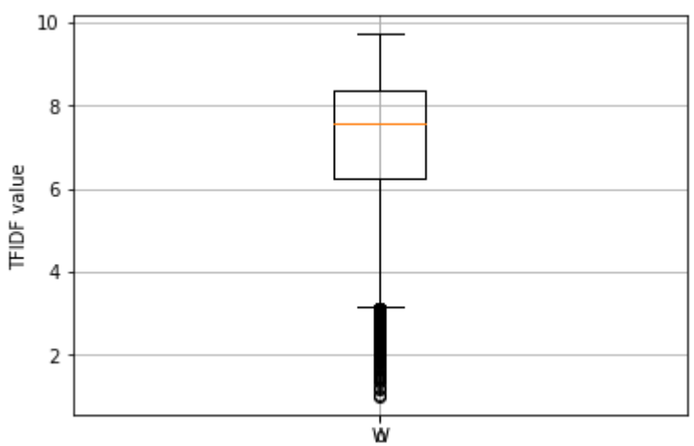
100 rows × 2 columns

In [ ]:
```
sns.histplot(df_idf_value.IDF_value)
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1102bf2940>
```



In [ ]:
```
plt.boxplot(df_idf_value.IDF_value)
plt.xticks([1,1],('Words'))
plt.ylabel('TFIDF value')
plt.grid()
plt.show()
```



In [ ]:
```
print('Max value of TFIDF',max(df_idf_value.IDF_value))
print('Max value of TFIDF',min(df_idf_value.IDF_value))
```

```
Max value of TFIDF 9.716003065714995
Max value of TFIDF 1.0081619668601856
```

In [ ]:
```
# refer - https://www.geeksforgeeks.org/numpy-percentile-in-python/
for i in range (0,101,2):
    p = np.percentile(vectorizer.idf_, i)
    print(str(i)+" Percentile: "+ str(p))
```

```
0 Percentile: 1.0081619668601856
2 Percentile: 3.3039551507631786
4 Percentile: 3.8984263137625805
6 Percentile: 4.348556888778909
8 Percentile: 4.652838584829192
10 Percentile: 4.963850689378257
12 Percentile: 5.195011951339468
14 Percentile: 5.386366818243037
16 Percentile: 5.554519200655265
18 Percentile: 5.714748926558905
20 Percentile: 5.8819416017565604
22 Percentile: 6.046051621486577
24 Percentile: 6.204457626883974
26 Percentile: 6.333706155957574
28 Percentile: 6.467568438605249
30 Percentile: 6.591437920318036
32 Percentile: 6.707848272162446
34 Percentile: 6.807282169150633
36 Percentile: 6.9177059867650055
38 Percentile: 7.01912616521091
40 Percentile: 7.1108518340408615
42 Percentile: 7.210477128724258
44 Percentile: 7.303069915552083
46 Percentile: 7.388725360130577
48 Percentile: 7.473521896790654
50 Percentile: 7.546949365345471
52 Percentile: 7.6159422368324226
54 Percentile: 7.690050208986145
56 Percentile: 7.758258459012678
58 Percentile: 7.831461863035974
60 Percentile: 7.883421601966683
62 Percentile: 7.952414473453635
64 Percentile: 8.011254973476568
66 Percentile: 8.073775330457902
68 Percentile: 8.140466704956575
70 Percentile: 8.193576530270523
72 Percentile: 8.269084082778669
74 Percentile: 8.329708704595102
```

```
76 Percentile: 8.3722683190139
78 Percentile: 8.439709599809433
80 Percentile: 8.487337648798686
82 Percentile: 8.56332355577661
84 Percentile: 8.617390777046886
86 Percentile: 8.674549190886832
88 Percentile: 8.735173812703268
90 Percentile: 8.76692251101785
92 Percentile: 8.83361388551652
94 Percentile: 8.86870520532779
96 Percentile: 8.905072849498666
98 Percentile: 9.02285588515505
100 Percentile: 9.716003065714995
```

From the above percentile result, best therashold is between 10th and 80 percentile, IDF value less than and higher than thershold is more frequent and rare much informative to text.

In [ ]:
```
min_the = 3.3039551507631786 # 2th percentile
max_the = 8.905072849498666 # 96th percentile
print('Total # words is effective to the model',len(df_idf_value[(df_idf_value['IDF_value']>= min_the) & (df_idf_value['IDF_va
```

```
Total # words is effective to the model 9418
```

In [ ]:
```
rare_word_count = len(df_idf_value[(df_idf_value['IDF_value']< min_the)])
Frequent_word_count = len(df_idf_value[(df_idf_value['IDF_value']> max_the)] )
print('Removed word count - rare and frequent count is ', rare_word_count+ Frequent_word_count)
```

```
Removed word count - rare and frequent count is  582
```

In [ ]:
```
df_x_train = df_idf_value.copy()
```

In [ ]:
```
# Shape of x_train
before = df_x_train.shape[0]
column  = 'IDF_value'
_2th = np.percentile(df_x_train[column], 2)
_96th = np.percentile(df_x_train[column], 96)

print('Minimum Therashold',_2th)
print('Maximum Therashold',_96th )
df_x_train.drop(df_x_train[df_x_train[column] < _2th ].index, inplace=True)
df_x_train.drop(df_x_train[df_x_train[column] > _96th ].index, inplace=True)
count = str(before - df_x_train.shape[0])
print("Number of rare and frequent words removed in column '{0}' : {1}".format(column, count))
```

```
Minimum Therashold 3.3039551507631786
Maximum Therashold 8.905072849498666
Number of rare and frequent words removed in column 'IDF_value' : 582
```

In [ ]:
```
print('Before removel shape of X_train',df_idf_value.shape)
print('After removeal of frequent and non-frequent words',df_x_train.shape)
```

```
Before removel shape of X_train (10000, 2)
After removeal of frequent and non-frequent words (9418, 2)
```

In [ ]:
```
# Now check MAX and MIN values
print('Max value of TFIDF',max(df_x_train.IDF_value))
print('Min value of TFIDF',min(df_x_train.IDF_value))
```

```
Max value of TFIDF 8.905072849498666
Min value of TFIDF 3.304047971022603
```

In [ ]:
```
# take the word , which add important to the model
best_word = list(df_x_train.Feature_name)
print('sample best words',best_word[-50:-45])
```

```
sample best words ['written', 'wrong', 'wrote', 'wwii', 'www']
```

In [ ]:
```
# Remove a word not in best word list (removing low and high idf words from dataset)
def remove_high_low_tfidf(text,best_word):
  best_essay = []
  for i in tqdm(text):
    #print(i)
    remove = [t for t in i.split() if t.lower() in best_word]
    join_word = ' '.join(remove)
    best_essay.append(join_word)

  return best_essay


#tt = list(X_train.essay)[100:101]
#eeee = remove_high_low_tfidf(tt,best_word)
best_essay = remove_high_low_tfidf(X_train.essay,best_word)
```

```
100%|██████████| 73196/73196 [17:08<00:00, 71.16it/s]
```

In [ ]:
```
print('sample essay text \n', best_essay[100])
```

```
sample essay text
 4th exposure non fiction text what magazine super please funding magazine gateway exposure magazine encourage pick topics pre
classes benefit interesting articles topics inspire power point presentations rest based upon topics magazines magazine infuse
mately honor funded since located socio economically disadvantaged funds limited kicking exposed articles broaden horizon
```

In [ ]:
```
#save the text for future use
best_essays = pd.DataFrame(data = best_essay,columns=['tfidf_essay'])
best_essays.head()
best_essays.to_csv('/gdrive/My Drive/LSTM/LSTM Assignment/best_essay_2.csv',index=None)
```

In [ ]:
```
#read csv file
essay_df= pd.read_csv('/gdrive/My Drive/LSTM/LSTM Assignment/best_essay_2.csv')
essay_df.columns
essay_df.head(2)
```

Out[42]:

	tfidf_essay
0	economically disadvantaged living totally floo...
1	walk lot eagerness faces holds for setting if ...

In [ ]:
```
essay_df['tfidf_essay'].values
```

Out[43]:
```
array(['economically disadvantaged living totally flooded shelters hotels living basic essentials self esteem educational ment
career college their whole depend getting back solid ground clear consistent schedules reinforce academics town flooded lost e
n educations needed furniture homey items permanent homes clothing furniture inviting fluency critical nonexistent chance succ
ggest asset',
       'walk lot eagerness faces holds for setting if ask reply to change change whether big serves privileged oakland speak m
l schooling rather scary despite hardships smiles sense ownership tables sharing readily available at ages managing large desk
tables age appropriate space along building peers your generous donation allowing gain space large desks change ownership free
       'inner city elementary large metropolitan computers unique aspect serve large population country majority east africa ]
higher position resource specialist serves 650 personal joys transition elementary middle donations printing color sharing fri
rs family friends digitally publish color gives sense pride color toner print certificates honor successes those successes inc
ompleting somali immigrants never formal month honor roll',
       ...,
       'hardworking funny passionate everything exciting topics require challenge discovering aspects history finding connecti
ticularly interesting 8th programs material accessible translation programs information ease expand understanding history gove
al computers ensure information equally peers computer laptop greatly increase understanding current events issues facing toda
t ell fully understand curriculum',
       'tend basic granted longer basic realize vital york city consists general write draw due current situation budget cuts
r must supply paper erasable pens enable done lot wasted sharpening pencils erasable pens engage smallest biggest difference r
lease necessity',
       'there times challenge tradition conventional wisdom comes tradition conventional wisdom long sit desks feet ground col
12th public charter taught question everything courses author bias perspective text organization effective questioning led inc
ng effective productive passively sitting typical desk black chrome drafting stools stools standing height desks actively cour
annotating texts drafting revising editing formal essays taking essays and according substantial discovered stools standing de
uctivity courses college prep stand funding stand challenge tradition conventional wisdom major impact please stand today'],
      dtype=object)
```

TEXT EMBEDDING - BEST TFIDF TEXT

In [ ]:
```
new_x_train = X_train.copy()
#new_x_train.drop('essay',axis=1,inplace=True)
new_x_train['tfidf_essay'] = essay_df['tfidf_essay'].values
new_x_train['Target'] = y_train
new_x_train.head(2)
```

Out[44]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	e
88868	nc	ms	grades_3_5	8	math_science	health_lifescience mathematics	my econom disadvant living p
30485	ca	ms	grades_prek_2	2	literacy_language appliedlearning	literature_writing other	my stu walk everyd eagerne

In [ ]:
```
list(new_x_train.essay.values)[0]
```

Out[45]:
```
'my kids economically disadvantaged living poverty school totally flooded they shelters hotels living day day they need basic
steem educational mentality i want get spark back eyes get reading writing daily career college ready their whole lives depenc
ound clear consistent materials schedules reinforce academics our school town flooded lost everything our kids trying return ]
eeded supplies books furniture homey items many students still without permanent homes clothing furniture school safe invitinę
eel safe reading fluency critical future success materials home nonexistent i want kids chance succeed with resources feel lik
uccess enrichment activities kids biggest asset nannan'
```

In [ ]:
```
list(new_x_train.tfidf_essay.values)[0]
```

Out[46]:
```
'economically disadvantaged living totally flooded shelters hotels living basic essentials self esteem educational mentality s
college their whole depend getting back solid ground clear consistent schedules reinforce academics town flooded lost everythi
tions needed furniture homey items permanent homes clothing furniture inviting fluency critical nonexistent chance succeed day
sset'
```

In [ ]:

In [ ]:
```
new_x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73196 entries, 88868 to 86194
Data columns (total 10 columns):
 #   Column                                        Non-Null Count  Dtype  
---  ------                                        --------------  -----  
 0   school_state                                  73196 non-null  object 
 1   teacher_prefix                                73196 non-null  object 
 2   project_grade_category                        73196 non-null  object 
 3   teacher_number_of_previously_posted_projects  73196 non-null  int64  
 4   clean_categories                              73196 non-null  object 
 5   clean_subcategories                           73196 non-null  object 
 6   essay                                         73196 non-null  object 
 7   price                                         73196 non-null  float64
 8   tfidf_essay                                   73196 non-null  object 
 9   Target                                        73196 non-null  int64  
dtypes: float64(1), int64(2), object(7)
memory usage: 6.1+ MB
```

In [ ]:
```
# tfidf_essay, 4 rows has null value so we going to
#new_x_train['tfidf_essay'].replace('', np.nan, inplace=True)
#new_x_train.dropna(subset=['tfidf_essay'], inplace=True)
#new_x_train.info()
```

In [ ]:

```
# First Essay Tokenizsation
def text_encoding(train,test):
    texts = train.tfidf_essay.values.tolist()
    texts_train = test.essay.values.tolist()
    print(texts[:10])
    # Tokenizer
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(texts)

    sequences = tokenizer.texts_to_sequences(texts)
    sequences_test = tokenizer.texts_to_sequences(texts_train)

    ''' tokenizer.word_index -  dic - key word value - index value'''
    word_index = tokenizer.word_index
    print('Found %s unique tokens.' % len(word_index))

    '''tokenizer.index_docs - Key index value value - occurances in # of documents'''
    index_doc_count = tokenizer.index_docs
    print('Index count',len(index_doc_count))

    # To select the best MAX_SEQ_LENGTH
    val_append = []
    for i in tqdm(range(len(sequences))):
        val_append.append(len(sequences[i]))

    print('Maximum length of essay in corpus',max(val_append))

    MAX_SEQUENCE_LENGTH = 210
    MAX_SEQUENCE_LENGTH
    print(MAX_SEQUENCE_LENGTH)

    data_essay = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH,padding='pre')
    data_essay_test = pad_sequences(sequences_test, maxlen=MAX_SEQUENCE_LENGTH,padding='pre')
    print('Shape of Essay Embeddin',data_essay.shape)


    BASE_DIR = ''
    GLOVE_DIR = os.path.join(BASE_DIR, r'/gdrive/My Drive/LSTM/LSTM Assignment')

    embeddings_index = {}
    with open(os.path.join(GLOVE_DIR, 'glove.6B.300d.txt'),encoding="utf8") as f:
        for line in tqdm(f):
```

```
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, 'f', sep=' ')
        embeddings_index[word] = coefs

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((len(word_index)+1, 300))
for word, i in tqdm(tokenizer.word_index.items()):

    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

print('Length of embedding matrix', len(embedding_matrix))
EMBEDDING_DIM = 300

embedding_layer_essay = Embedding(len(tokenizer.word_index)+1,
                    EMBEDDING_DIM,
                    weights=[embedding_matrix],
                    input_length=MAX_SEQUENCE_LENGTH,
                    trainable=False)

return embedding_layer_essay,data_essay,data_essay_test
```

In [ ]:
```
text_doc_embedding = text_encoding(new_x_train,X_test)
embedding_layer_text = text_doc_embedding[0]
print('TEXT EMBEDDING HAS COMPLETED')
x_train_text_tfidf = text_doc_embedding[1]
print('shape of after text embedding',x_train_text_tfidf.shape)
x_test_texttfidf = text_doc_embedding[2]
print('shape of after text embedding',x_test_texttfidf.shape)
```

```
['economically disadvantaged living totally flooded shelters hotels living basic essentials self esteem educational mentality
college their whole depend getting back solid ground clear consistent schedules reinforce academics town flooded lost everythi
tions needed furniture homey items permanent homes clothing furniture inviting fluency critical nonexistent chance succeed day
sset', 'walk lot eagerness faces holds for setting if ask reply to change change whether big serves privileged oakland speak m
l schooling rather scary despite hardships smiles sense ownership tables sharing readily available at ages managing large desk
tables age appropriate space along building peers your generous donation allowing gain space large desks change ownership free
entary large metropolitan computers unique aspect serve large population country majority east africa latin america status 98
ce specialist serves 650 personal joys transition elementary middle donations printing color sharing friends relatives compute
tally publish color gives sense pride color toner print certificates honor successes those successes included completing 8th c
rants never formal month honor roll', 'wide levels but thing common walk store ask who whole reply believe expect nothing less
c sand electric pencil sharpener single items complete worth return sand sensory items emotionally behavior calm focused longe
ner frustrating near eye return allows pencil sharp waste hand held sharpeners', 'unique end succeed despite accommodations re
believe ict over past months noticed struggling fine gross motor adhd dyslexia emotional anxiety greatly benefit additional th
floor puzzles caddy bingo games shift energy back topic hand really gain confidence if believe believe anything provided neces
ing request inclusion each learns unique co fit essential each key component difference wobble stool perfect fit struggling st
is sight word floor puzzle journal interactive basic sight words stay caddies essential crafts organized convenient anything c
0 chromebooks technological literacy develop keyboarding prepare line assessments educated digital citizenship enthusiastic ar
ced computer skill learned right visit computer lab three weeks due number classes trying fit technological literacy achieve h
igniting passion helping achieve personal wonderful supportive administration impressive lovely smile faces desire addition ch
pen programs purchased personalized reinforce end assessment taken line chromebook prepare navigate perform chromebooks develc
arly start write code develop keyboarding computer addition chromebooks ensure prepared upper grades career choices comes neec
nging rapidly technological prepared higher eventually developing job market there jobs tech graduate let', 'middle located cc
consists 125 wide range socioeconomic for past co embarking journey call nature centered outdoor fortunate pond lake right reg
um focuses natural local issues hope creating generation newfound respect after searching inspiration approach turned natural
rve background nature centered outdoor weekly basis various including testing water samples identifying trees species collecti
link curriculum back inside found jackets collecting water samples', 'walk awaits tell going population ranges gifted disabili
family spend lot passion job leaders tomorrow learned job welcomes embraces differences flexible seating family eight hours ex
chairs entire shows short bouts physical activity task behavior leads performance giving choice seating role process flexible
choices purchase choose try seating options learned truly frequently maximize interaction performance', 'groove moving element
there usually 20 24 supportive willing qualify funding due for setting typically trouble sitting chairs benefit alternate seat
takes remembered movement involved seating options increase today everything moves faster pace choices seating taking please a
ating choices', 'today buzz excitement strive nurturing proper percentage living comes certain to basic breakfast provided una
y believe capable given proper positive encouragement walks door matters long term chrome currently chrome shared 25 another c
providing greater must keyboarding today elementary rigor state testing requires type performance tasks essay form challenge a
```

```
disadvantage cases computer computers opens possibilities keyboarding aspect information possible assignments less computers t
y fluency benefit additional needed chrome']

100%|██████████| 73196/73196 [00:00<00:00, 2084819.00it/s]

Found 9418 unique tokens.
Index count 9418
Maximum length of essay in corpus 209
210


1920it [00:00, 19193.43it/s]

Shape of Essay Embeddin (73196, 210)

400001it [00:20, 19659.22it/s]
100%|██████████| 9418/9418 [00:00<00:00, 446940.65it/s]

Length of embedding matrix 9419
TEXT EMBEDDING HAS COMPLETED
shape of after text embedding (73196, 210)
shape of after text embedding (36052, 210)
```

In [ ]:

Run the categorical and Numerical type column using model 1

In [ ]:

```
State_embedding = embedding_state(new_x_train,X_test)
x_train_sch_state = State_embedding[0]
print('State embedding shape train',x_train_sch_state.shape)
x_test_sch_state = State_embedding[1]
print('State embedding shape test',x_test_sch_state.shape)
state_layer_model = State_embedding[2]


# Grade
Grade_embedding = embedding_grade(new_x_train,X_test)
x_train_proj_grade = Grade_embedding[0]
print('Grade embedding shape train',x_train_proj_grade.shape)
x_test_proj_grade = Grade_embedding[1]
print('Grade embedding shape test',x_test_proj_grade.shape)
grade_layer_model = Grade_embedding[2]

#clean category
Clean_cat_embedding = embedding_cleab_cat(new_x_train,X_test)
x_train_clean_cat  = Clean_cat_embedding[0]
print('Clean_cat embedding shape train',x_train_clean_cat.shape)
x_test_clean_cat  = Clean_cat_embedding[1]
print('Clean_cat embedding shape test',x_test_clean_cat.shape)
cat_layer_model = Clean_cat_embedding[2]

#clean sub cat
Clean_Subcat_embedding = embedding_clean_sub_cat(new_x_train,X_test)
x_train_Subclean_cat  = Clean_Subcat_embedding[0]
print('Clean_sub_cat embedding shape train',x_train_Subclean_cat.shape)
x_test_Subclean_cat  = Clean_Subcat_embedding[1]
print('Clean_sub_cat embedding shape test',x_test_Subclean_cat.shape)
sub_cat_layer_model = Clean_Subcat_embedding[2]

# teacher prefix
Teacher_pre_embedding = embedding_Teacher_prefix(new_x_train,X_test)
x_train_teach_pre  = Teacher_pre_embedding[0]
print('Teacher_prefix embedding shape train',x_train_teach_pre.shape)
x_test_teach_pre  = Teacher_pre_embedding[1]
print('Teacher_prefix embedding shape test',x_test_teach_pre.shape)
prefix_layer_model = Teacher_pre_embedding[2]

#numeric
numerical_feature = Numerical_preprocessing(new_x_train,X_test)
```

```
x_train_previously_posted_projects  = numerical_feature[0]
x_test_previously_posted_projects  =  numerical_feature[1]
```

```
100%|██████████| 73196/73196 [00:00<00:00, 2280031.75it/s]

Embedding max_length 1
One hot encoding done - school state
length uniques 51
State embedding shape train (73196, 1)
State embedding shape test (36052, 1)

100%|██████████| 73196/73196 [00:00<00:00, 2397287.88it/s]

Embedding max_length 1
One hot encoding done - Project grade
length uniques 4
Grade embedding shape train (73196, 1)
Grade embedding shape test (36052, 1)

100%|██████████| 73196/73196 [00:00<00:00, 2142252.99it/s]

Embedding max_length 3
One hot encoding done - Clean category
length uniques 9
Clean_cat embedding shape train (73196, 3)
Clean_cat embedding shape test (36052, 3)

100%|██████████| 73196/73196 [00:00<00:00, 1906717.32it/s]

Embedding max_length 3
One hot encoding done - Clean  subcategory
length uniques 30
Clean_sub_cat embedding shape train (73196, 3)
Clean_sub_cat embedding shape test (36052, 3)

100%|██████████| 73196/73196 [00:00<00:00, 2265595.21it/s]

Embedding max_length 1
One hot encoding done - Teacher prefix
length uniques 5
Teacher_prefix embedding shape train (73196, 1)
Teacher_prefix embedding shape test (36052, 1)
```

In [ ]:
```
final_x_train  = [x_train_text_tfidf,x_train_sch_state,x_train_proj_grade,x_train_clean_cat,x_train_Subclean_cat,x_train_teach
final_x_test  = [x_test_texttfidf,x_test_sch_state,x_test_proj_grade,x_test_clean_cat,x_test_Subclean_cat,x_test_teach_pre,x_t
```

In [ ]:

```
# Create Model
os.environ['PYTHONHASHSEED'] = '0'

##https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development
## Have to clear the session. If you are not clearing, Graph will create again and again and graph size will increses.
## Varibles will also set to some value from before session
tf.keras.backend.clear_session()

## Set the random seed values to regenerate the model.
np.random.seed(0)
rn.seed(0)

#Input layers
# Text Data Input layer
input_layer_essay = Input(shape=(210,), dtype='int32',name='Essay_text')
e =embedding_layer_text(input_layer_essay)
lstm = LSTM(128,return_sequences=True)(e)
X1 = Flatten(data_format='channels_last',name='Flatten_essay')(lstm)

# Second Input layer - state
input_layer_state = Input(shape=(1,), dtype='int32',name='School_Sate')
e_state = state_layer_model(input_layer_state)
X2 = Flatten(data_format='channels_last',name='Flatten_state')(e_state)

# Third Input layer - grade
input_layer_grade= Input(shape=(1,), dtype='int32',name='Project_grade')
e_grade = grade_layer_model(input_layer_grade)
X3= Flatten(data_format='channels_last',name='Flatten_grade')(e_grade)


# Fourth  Input layer - category
input_layer_cat= Input(shape=(3,), dtype='int32',name='Project_category')
e_cat = cat_layer_model(input_layer_cat)
X4= Flatten(data_format='channels_last',name='Flatten_category')(e_cat)


# fifth  Input layer - sub_category
input_layer_sub_category= Input(shape=(3,), dtype='int32',name='SubCategory')
e_sub_category = sub_cat_layer_model(input_layer_sub_category)
X5= Flatten(data_format='channels_last',name='Flatten_sub_category')(e_sub_category)

# Sixth Input layer - Teacher prefix
```

```
input_layer_teacher_prefix = Input(shape=(1,), dtype='int32',name='Teacher_prefix')
e_teacher_prefix= prefix_layer_model(input_layer_teacher_prefix)
X6= Flatten(data_format='channels_last',name='Flatten_teacher_prefix')(e_teacher_prefix)

#input 7
input_7 = Input(shape=(2,))
X7 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(inp

'''input_8 = Input(shape=(1,))
X8 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(inp

merge = concatenate([X7,X8])'''

concat = concatenate([X1,X2,X3,X4,X5,X6,X7],axis=1)

# Fully connected Dense Layer

x = Dense(64, activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l1_l2(l1=1e-5,
          bias_regularizer= l2(1e-4),activity_regularizer=l2(1e-5))(concat)
x = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l1_l2(l1=1e-5, l
          bias_regularizer= l2(1e-4),activity_regularizer=l2(1e-5))(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l1_l2(l1=1e-5, l
          bias_regularizer= l2(1e-4),activity_regularizer=l2(1e-5))(x)
x = Dropout(0.5)(x)
output = Dense(2, activation = 'softmax')(x)


all_inputs =  [input_layer_essay,input_layer_state,input_layer_grade,input_layer_cat,input_layer_sub_category,input_layer_teac
model = Model(inputs=all_inputs,outputs=output)
model.summary()
```

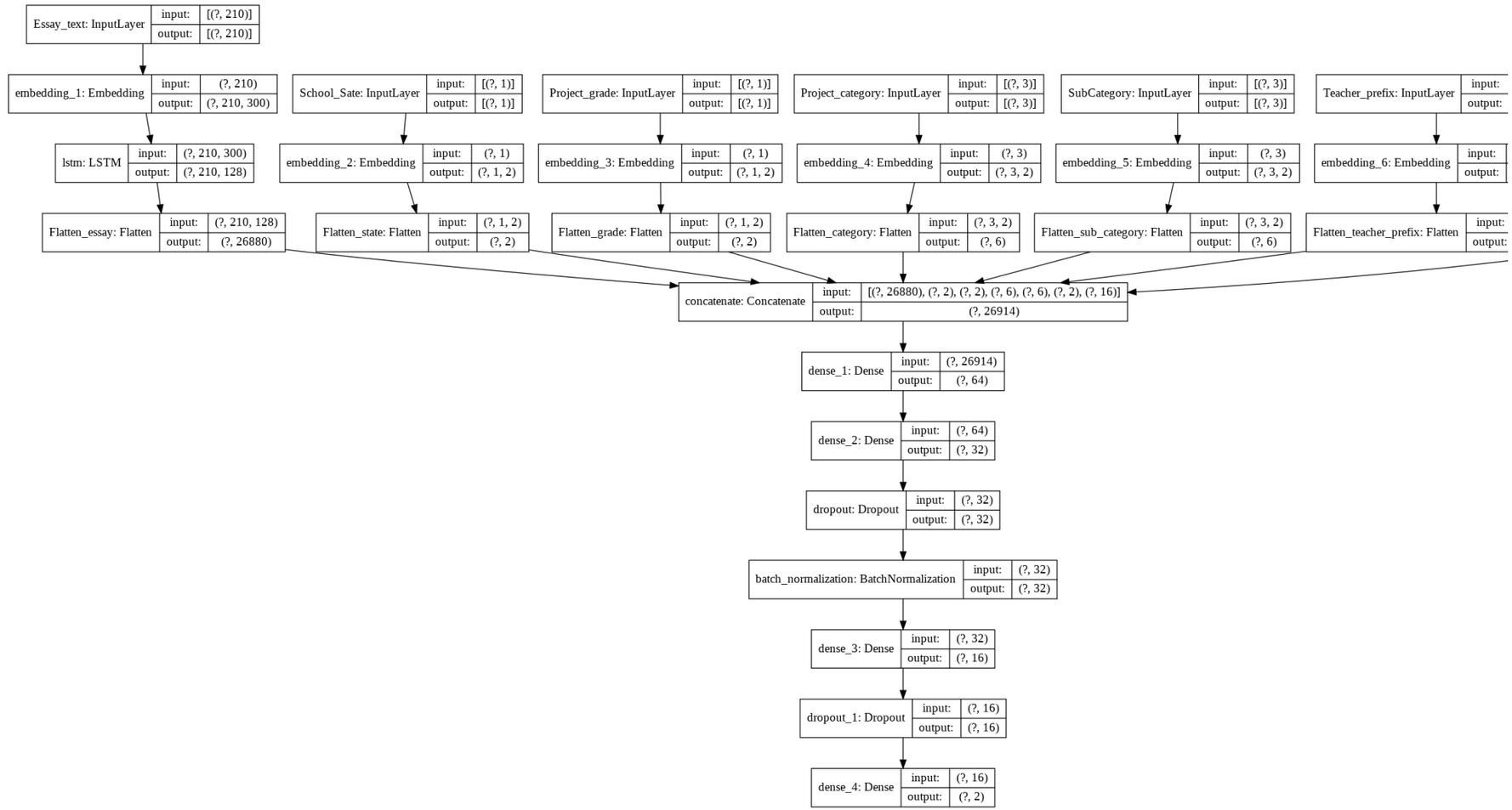
```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Essay_text (InputLayer)         [(None, 210)]        0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 210, 300)     2825700     Essay_text[0][0]
__________________________________________________________________________________________________
School_Sate (InputLayer)        [(None, 1)]          0
```

Project_grade (InputLayer)	[(None, 1)]	0	
Project_category (InputLayer)	[(None, 3)]	0	
SubCategory (InputLayer)	[(None, 3)]	0	
Teacher_prefix (InputLayer)	[(None, 1)]	0	
lstm (LSTM)	(None, 210, 128)	219648	embedding_1[4][0]
embedding_2 (Embedding)	(None, 1, 2)	104	School_Sate[0][0]
embedding_3 (Embedding)	(None, 1, 2)	10	Project_grade[0][0]
embedding_4 (Embedding)	(None, 3, 2)	20	Project_category[0][0]
embedding_5 (Embedding)	(None, 3, 2)	62	SubCategory[0][0]
embedding_6 (Embedding)	(None, 1, 2)	12	Teacher_prefix[0][0]
input_1 (InputLayer)	[(None, 2)]	0	
Flatten_essay (Flatten)	(None, 26880)	0	lstm[0][0]
Flatten_state (Flatten)	(None, 2)	0	embedding_2[4][0]
Flatten_grade (Flatten)	(None, 2)	0	embedding_3[4][0]
Flatten_category (Flatten)	(None, 6)	0	embedding_4[4][0]
Flatten_sub_category (Flatten)	(None, 6)	0	embedding_5[4][0]
Flatten_teacher_prefix (Flatten	(None, 2)	0	embedding_6[4][0]
dense (Dense)	(None, 16)	48	input_1[0][0]
concatenate (Concatenate)	(None, 26914)	0	Flatten_essay[0][0] Flatten_state[0][0] Flatten_grade[0][0] Flatten_category[0][0] Flatten_sub_category[0][0] Flatten_teacher_prefix[0][0]

```
                                                                 dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           1722560     concatenate[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 32)           2080        dense_1[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 32)           0           dense_2[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 32)           128         dropout[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 16)           528         batch_normalization[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 16)           0           dense_3[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 2)            34          dropout_1[0][0]
==================================================================================================
Total params: 4,770,934
Trainable params: 1,945,170
Non-trainable params: 2,825,764
__________________________________________________________________________________________________
```

In [ ]:
```
dot_img_file = '/tmp/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

Out[117]:



In [ ]:
```
# Convert Target in to categorical
new_y_train = to_categorical(new_x_train.Target.values)
new_y_test = to_categorical(y_test)
```

In [ ]:
```
filepath="/gdrive/My Drive/LSTM/LSTM Assignment/save_model/best_model-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc_score',  verbose=1, save_best_only=True, mode='max')
```

In [ ]:
```

ACCURACY_THRESHOLD_test = 0.71
class myCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs={}):


      if(logs.get('val_auc_score') > ACCURACY_THRESHOLD_test) and (logs.get('auc_score') > ACCURACY_THRESHOLD_test) :
        print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD_test*100))
        self.model.stop_training = True



early_stop_auc_scores = myCallback()
```

In [ ]:
```
# Early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_auc_score', min_delta=0, patience=10, verbose=0, mode='auto',
    baseline=None, restore_best_weights=False

)
```

In [ ]:
```

reduce_lr = ReduceLROnPlateau(monitor='val_auc_score', factor=0.2,
                              patience=1, min_lr=0.0001)
```

In [ ]:
```
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/LSTM/LSTM Assignment/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=0, write_graph=True)
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [ ]:
```
folder_name
```

Out[124]:
```
'20201002-124401'
```

In [ ]:
```
def auc_score(y_true, y_pred):

    auc_scoree =  tf.compat.v1.py_func(metrics.roc_auc_score, (y_true, y_pred), tf.double)
    return auc_scoree
```

In [ ]:
```
# compile
model.compile(loss='categorical_crossentropy',
            optimizer=keras.optimizers.Adam(lr = 0.0006),
             metrics=[auc_score ])
```

In [ ]:
```
model.fit(final_x_train, new_y_train, epochs=20,batch_size=64, validation_data=(final_x_test, new_y_test),
          callbacks =[checkpoint,tensorboard_callback,early_stop_auc_scores,reduce_lr])
```

```
Epoch 1/20
   2/1144 [..............................] - ETA: 3:08 - loss: 0.8904 - auc_score: 0.5107WARNING:tensorflow:Callbacks method `
slow compared to the batch time (batch time: 0.0377s vs `on_train_batch_end` time: 0.2920s). Check your callbacks.
1143/1144 [============================>.] - ETA: 0s - loss: 0.5948 - auc_score: 0.5472
Epoch 00001: val_auc_score improved from -inf to 0.52700, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/bes
1144/1144 [==============================] - 40s 35ms/step - loss: 0.5948 - auc_score: 0.5472 - val_loss: 0.5555 - val_auc_scc
Epoch 2/20
1144/1144 [==============================] - ETA: 0s - loss: 0.4916 - auc_score: 0.6312
Epoch 00002: val_auc_score improved from 0.52700 to 0.69309, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 39s 34ms/step - loss: 0.4916 - auc_score: 0.6312 - val_loss: 0.4609 - val_auc_scc
Epoch 3/20
1144/1144 [==============================] - ETA: 0s - loss: 0.4489 - auc_score: 0.6833
Epoch 00003: val_auc_score improved from 0.69309 to 0.70759, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 39s 34ms/step - loss: 0.4489 - auc_score: 0.6833 - val_loss: 0.4403 - val_auc_scc
Epoch 4/20
1143/1144 [============================>.] - ETA: 0s - loss: 0.4366 - auc_score: 0.6964
Epoch 00004: val_auc_score improved from 0.70759 to 0.71514, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 39s 34ms/step - loss: 0.4366 - auc_score: 0.6962 - val_loss: 0.4310 - val_auc_scc
Epoch 5/20
1142/1144 [============================>.] - ETA: 0s - loss: 0.4310 - auc_score: 0.7084
Epoch 00005: val_auc_score improved from 0.71514 to 0.71642, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 40s 35ms/step - loss: 0.4309 - auc_score: 0.7082 - val_loss: 0.4302 - val_auc_scc
Epoch 6/20
1142/1144 [============================>.] - ETA: 0s - loss: 0.4271 - auc_score: 0.7222
Epoch 00006: val_auc_score improved from 0.71642 to 0.71676, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/

Reached 71.00% accuracy, so stopping training!!
1144/1144 [==============================] - 40s 35ms/step - loss: 0.4271 - auc_score: 0.7220 - val_loss: 0.4295 - val_auc_scc
```

Out[127]:
```
<tensorflow.python.keras.callbacks.History at 0x7f10f866edd8>
```

In [ ]:
```
```

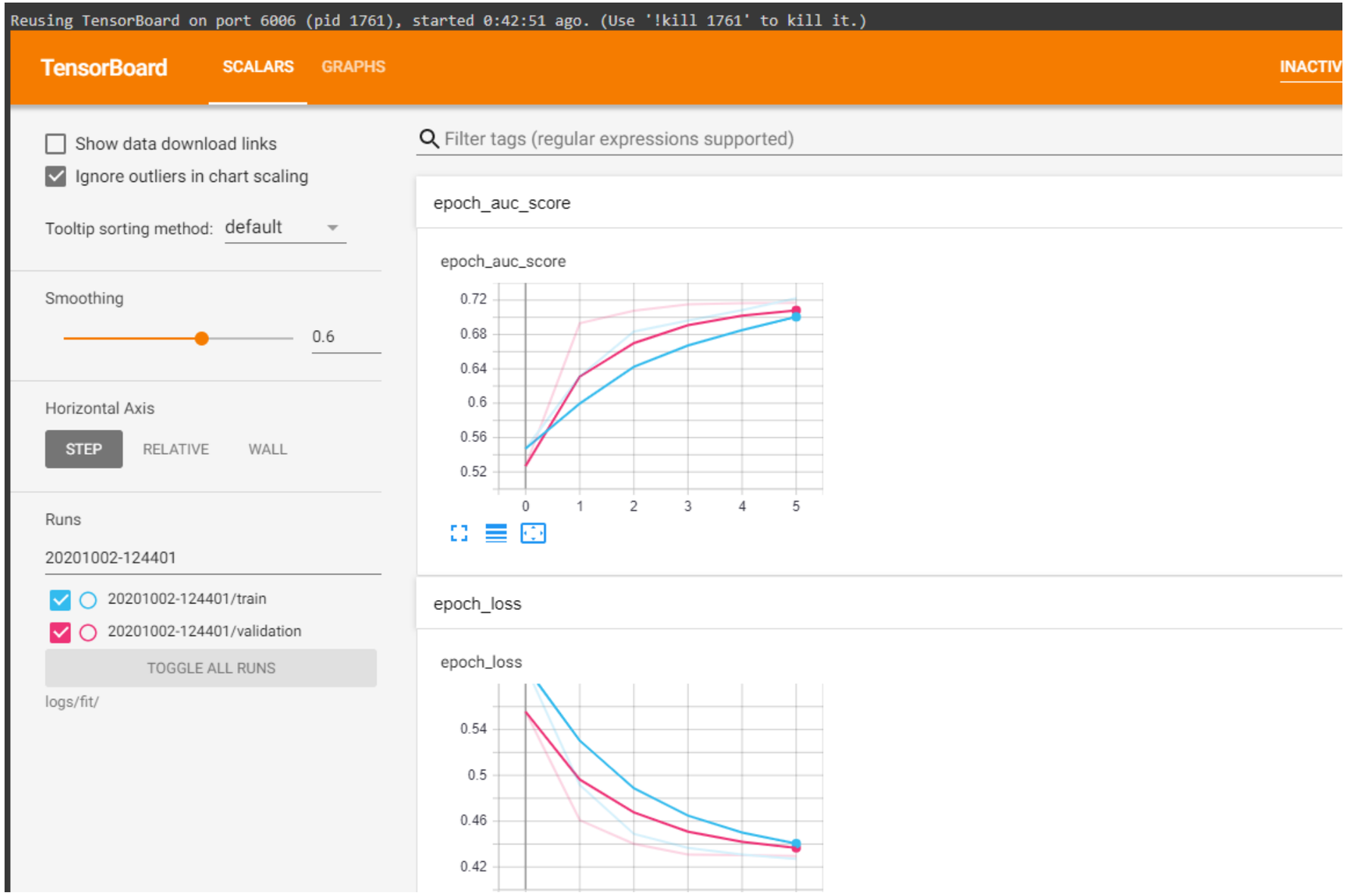
In [ ]:
```
# Evaluate
predict = model.predict(final_x_test)
score = metrics.roc_auc_score(new_y_test,predict)
print('roc_auc_score', score)
```

```
roc_auc_score 0.7140296363413339
```

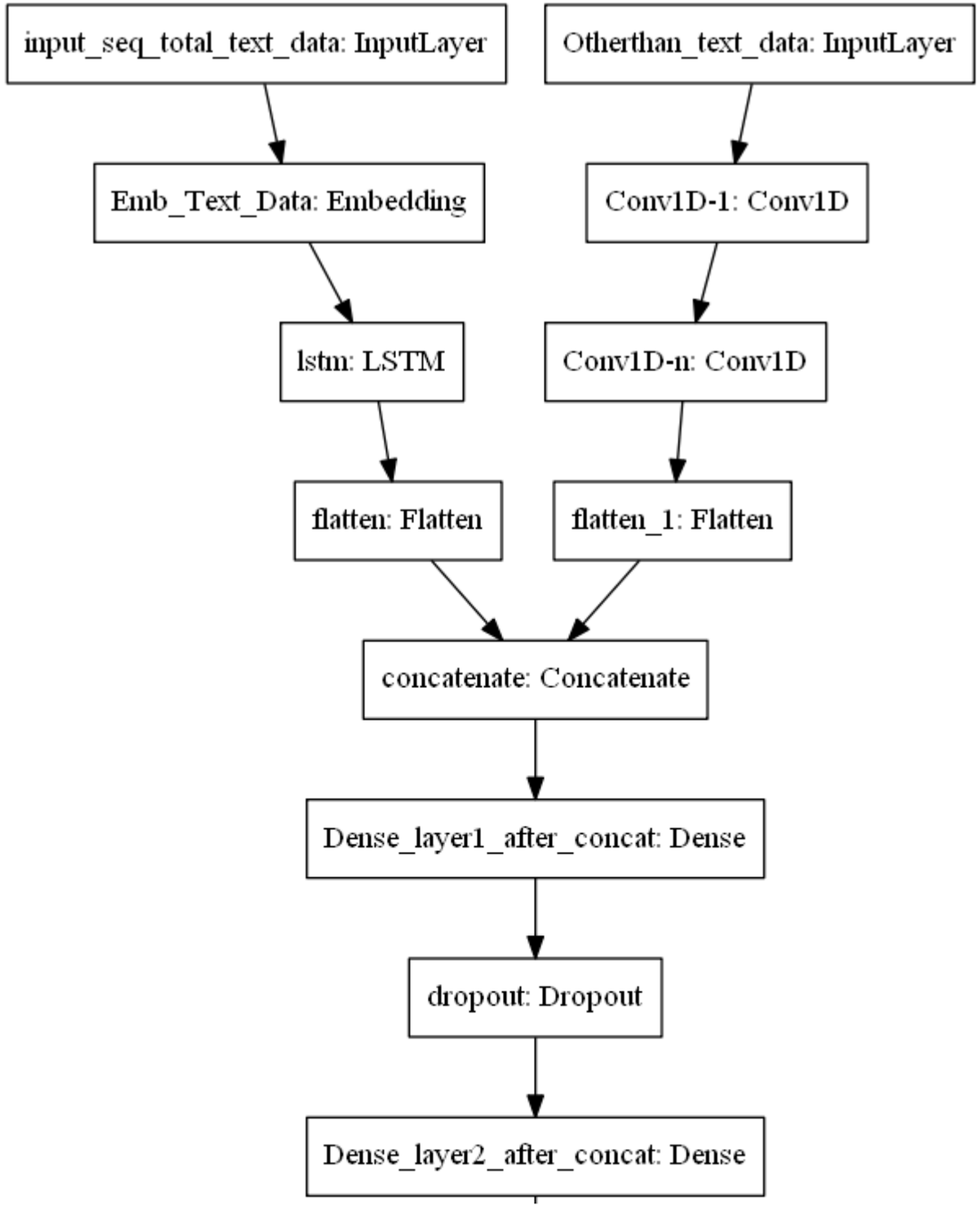
In [ ]:
```
os.chdir('/gdrive/My Drive/LSTM/LSTM Assignment')
```

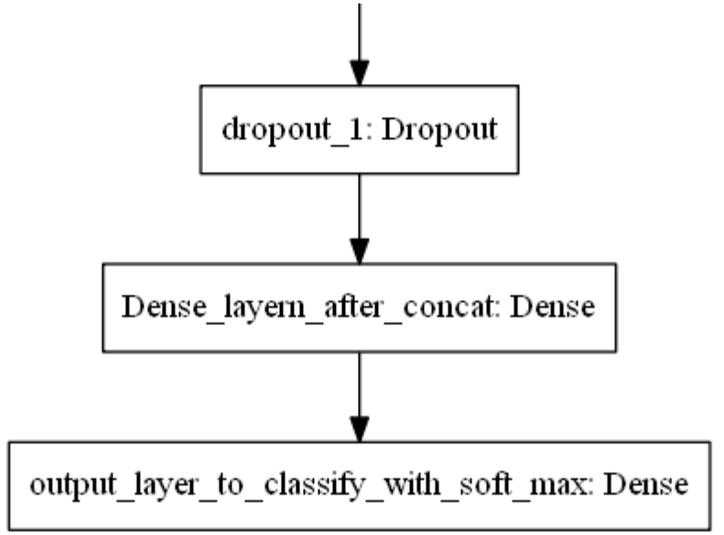
In [ ]:
```
%tensorboard --logdir logs/fit/
```

```
Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.
```


Reusing TensorBoard on port 6006 (pid 1761), started 0:42:51 ago. (Use '!kill 1761' to kill it.)
TensorBoard
SCALARS
GRAPHS
INACTIV
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
20201002-124401
20201002-124401/train
20201002-124401/validation
TOGGLE ALL RUNS
logs/fit/
Filter tags (regular expressions supported)
epoch_auc_score
epoch_auc_score
0.72
0.68
0.64
0.6
0.56
0.52
0
1
2
3
4
5
epoch_loss
epoch_loss
0.54
0.5
0.46
0.42

Model-3


input_seq_total_text_data: InputLayer
Otherthan_text_data: InputLayer
Emb_Text_Data: Embedding
Conv1D-1: Conv1D
lstm: LSTM
Conv1D-n: Conv1D
flatten: Flatten
flatten_1: Flatten
concatenate: Concatenate
Dense_layer1_after_concat: Dense
dropout: Dropout
Dense_layer2_after_concat: Dense



ref: https://i.imgur.com/fkQ8nGo.png

- **input_seq_total_text_data**:

```
. Use text column('essay'), and use the Embedding layer to get word vectors.

. Use given predefined glove word vectors, don't train any word vectors.

. Use LSTM that is given above, get the LSTM output and Flatten that output.

. You are free to preprocess the input text as you needed.
```

- **Other_than_text_data**:

. Convert all your Categorical values to onehot coded and then concatenate all these onehot vectors

. Neumerical values and use CNN1D (https://keras.io/getting-started/sequential-model-guide/#sequence-classification-
s) as shown in above figure.

. You are free to choose all CNN parameters like kernel sizes, stride.

In [ ]:
```
#X_train, X_test, y_train, y_test
```

TEXT EMBEDDING

In [ ]:
```
text_doc_embedding = text_encoding(X_train,X_test)
embedding_layer_text = text_doc_embedding[0]
print('TEXT EMBEDDING HAS COMPLETED')
x_train_text = text_doc_embedding[1]
print('shape of after text embedding',x_train_text.shape)
x_test_text = text_doc_embedding[2]
print('shape of after text embedding',x_test_text.shape)
```

```
100%|██████████| 73196/73196 [00:00<00:00, 2229691.88it/s]

Found 48167 unique tokens.
Index count 48167
Maximum length of essay in corpus 339
500


1903it [00:00, 19021.40it/s]

Shape of Essay Embeddin (73196, 500)

400001it [00:21, 18994.69it/s]
100%|██████████| 48167/48167 [00:00<00:00, 478704.17it/s]

Length of embedding matrix 48168
TEXT EMBEDDING HAS COMPLETED
shape of after text embedding (73196, 500)
shape of after text embedding (36052, 500)
```

CATEGORICAL ONE HOT ENCODING

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [ ]:

```
# Categorial Feature Encoding


print('1. State Encoding')
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
#X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
#print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

print('2. Teachers prefix Encoding')

vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
#print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

print('3. Project_grade_category')

vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
```

```
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
#X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
#print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

print('4. Clean_category')

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
#X_cv_clean_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
#print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

print('5. Clean_subcategory')

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcat_ohe = vectorizer.transform(X_train['clean_categories'].values)
#X_cv_clean_subcat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_subcat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_subcat_ohe.shape, y_train.shape)
#print(X_cv_clean_subcat_ohe.shape, y_cv.shape)
print(X_test_clean_subcat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
print("="*100)
```

```
1. State Encoding
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md',
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'v
'wv', 'wy']
====================================================================================================
2. Teachers prefix Encoding
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
3. Project_grade_category
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
4. Clean_category
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'spec
====================================================================================================
5. Clean_subcategory
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'early
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness', 'health_lifesci
s', 'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvol
ts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

NUMERICAL

In [ ]:
```
numerical_feature = Numerical_preprocessing(X_train,X_test)
x_train_previously_posted_projects  = numerical_feature[0]
x_test_previously_posted_projects  =  numerical_feature[1]
```

Merge other than text data

In [ ]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,X_train_clean_cat_ohe,X_train_clean_subcat_ohe, x_tra
X_te = hstack(( X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,X_test_clean_cat_ohe,X_test_clean_subcat_ohe, x_test_pr
```

In [ ]:
```
print('2dimesional - X_train',X_tr.shape)
print('2dimesional - X_test',X_te.shape)
X_tr = np.array(X_tr).reshape(X_tr.shape[0],X_tr.shape[1],1)
X_te = np.array(X_te).reshape(X_te.shape[0],X_te.shape[1],1)
print('2dimesional - X_train',X_tr.shape)
print('2dimesional - X_test',X_te.shape)
```

```
2dimesional - X_train (73196, 101)
2dimesional - X_test (36052, 101)
2dimesional - X_train (73196, 101, 1)
2dimesional - X_test (36052, 101, 1)
```

In [ ]:
```
final_x_train = [x_train_text,X_tr]
final_x_test = [x_test_text,X_te]
```

CREATE MODEL

In [ ]:

```
# Create Model
os.environ['PYTHONHASHSEED'] = '0'

##https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development
## Have to clear the session. If you are not clearing, Graph will create again and again and graph size will increses.
## Varibles will also set to some value from before session
tf.keras.backend.clear_session()

## Set the random seed values to regenerate the model.
np.random.seed(0)
rn.seed(0)

#Input layers
# Text Data Input layer
input_layer_essay = Input(shape=(500,), dtype='int32',name='Essay_text')
e =embedding_layer_text(input_layer_essay)
lstm = LSTM(128,return_sequences=True)(e)
X1 = Flatten(data_format='channels_last',name='Flatten_essay')(lstm)

input_layer_cat_num = Input(shape=(101,1),name='Cat_numeric')
conv_1 = Conv1D(128, 3, activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45))(input_layer_cat_num)
conv_2 = Conv1D(64, 3, activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45))(conv_1)
X2 = Flatten(data_format='channels_last',name='Flatten_Cat_num')(conv_2)

concat = concatenate([X1,X2],axis=1)

x = Dense(64, activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(cor
x = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.2)(x)
x = BatchNormalization()(x)
x = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.2)(x)
output = Dense(2, activation = 'softmax')(x)

all_inputs = [input_layer_essay,input_layer_cat_num]
model = Model(inputs=all_inputs,outputs=output)
model.summary()
```

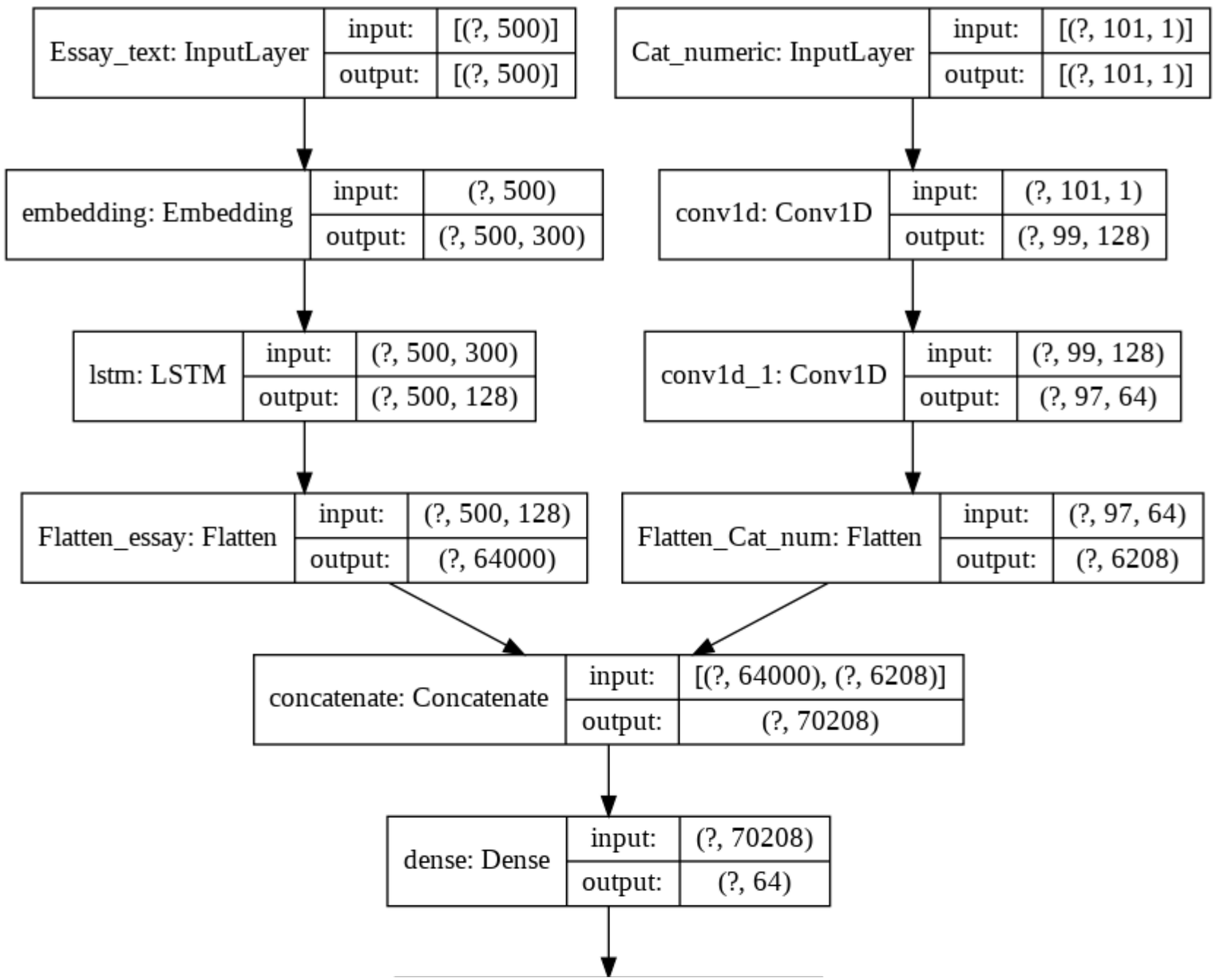
```
Model: "functional_1"
```

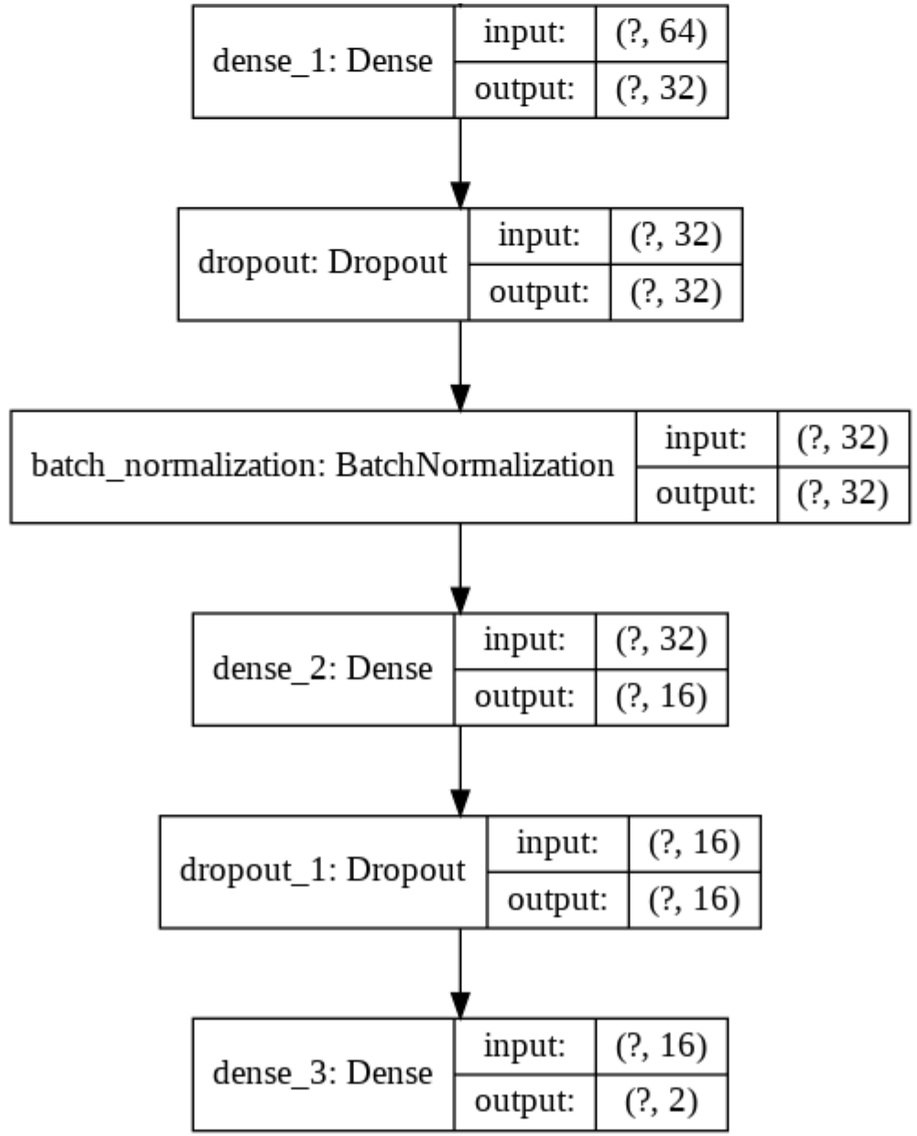
Layer (type)	Output Shape	Param #	Connected to
Essay_text (InputLayer)	[(None, 500)]	0	
Cat_numeric (InputLayer)	[(None, 101, 1)]	0	
embedding (Embedding)	(None, 500, 300)	14450400	Essay_text[0][0]
conv1d (Conv1D)	(None, 99, 128)	512	Cat_numeric[0][0]
lstm (LSTM)	(None, 500, 128)	219648	embedding[1][0]
conv1d_1 (Conv1D)	(None, 97, 64)	24640	conv1d[0][0]
Flatten_essay (Flatten)	(None, 64000)	0	lstm[0][0]
Flatten_Cat_num (Flatten)	(None, 6208)	0	conv1d_1[0][0]
concatenate (Concatenate)	(None, 70208)	0	Flatten_essay[0][0] Flatten_Cat_num[0][0]
dense (Dense)	(None, 64)	4493376	concatenate[0][0]
dense_1 (Dense)	(None, 32)	2080	dense[0][0]
dropout (Dropout)	(None, 32)	0	dense_1[0][0]
batch_normalization (BatchNorma	(None, 32)	128	dropout[0][0]
dense_2 (Dense)	(None, 16)	528	batch_normalization[0][0]
dropout_1 (Dropout)	(None, 16)	0	dense_2[0][0]
dense_3 (Dense)	(None, 2)	34	dropout_1[0][0]

Total params: 19,191,346
Trainable params: 4,740,882
Non-trainable params: 14,450,464

In [ ]:
```
dot_img_file = '/tmp/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

Out[36]:





In [ ]:
```
# Conevrt Target in to categorical
new_y_train = to_categorical(y_train)
new_y_test = to_categorical(y_test)
```

In [ ]:
```
ACCURACY_THRESHOLD_test = 0.75
class myCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs={}):


      if(logs.get('val_auc_score') > ACCURACY_THRESHOLD_test) and (logs.get('auc_score') > ACCURACY_THRESHOLD_test) :
        print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD_test*100))
        self.model.stop_training = True



early_stop_auc_scores = myCallback()
```

In [ ]:
```
# Early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_auc_score', min_delta=0, patience=10, verbose=0, mode='auto',
    baseline=None, restore_best_weights=False

)
```

In [ ]:
```
filepath="/gdrive/My Drive/LSTM/LSTM Assignment/save_model/best_model-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc_score',  verbose=1, save_best_only=True, mode='max')
```

In [ ]:
```

reduce_lr = ReduceLROnPlateau(monitor='val_auc_score', factor=0.2,
                              patience=1, min_lr=0.0001)
```

In [ ]:
```
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/LSTM/LSTM Assignment/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=0, write_graph=True)
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [ ]:
```
folder_name
```

Out[43]:
```
'20201002-142741'
```

In [ ]:
```
def auc_score(y_true, y_pred):

  try:
    auc_scoree =  tf.compat.v1.py_func(metrics.roc_auc_score, (y_true, y_pred), tf.double)
    return auc_scoree
  except ValueError:
    pass
```

In [ ]:
```
# compile
model.compile(loss='categorical_crossentropy',
            optimizer=keras.optimizers.Adam(lr = 0.0008),
             metrics=[auc_score ])
```

In [ ]:
```
model.fit(final_x_train, new_y_train, epochs=20,batch_size=64, validation_data=(final_x_test, new_y_test),
          callbacks =[checkpoint,tensorboard_callback,early_stop_auc_scores,reduce_lr])
```

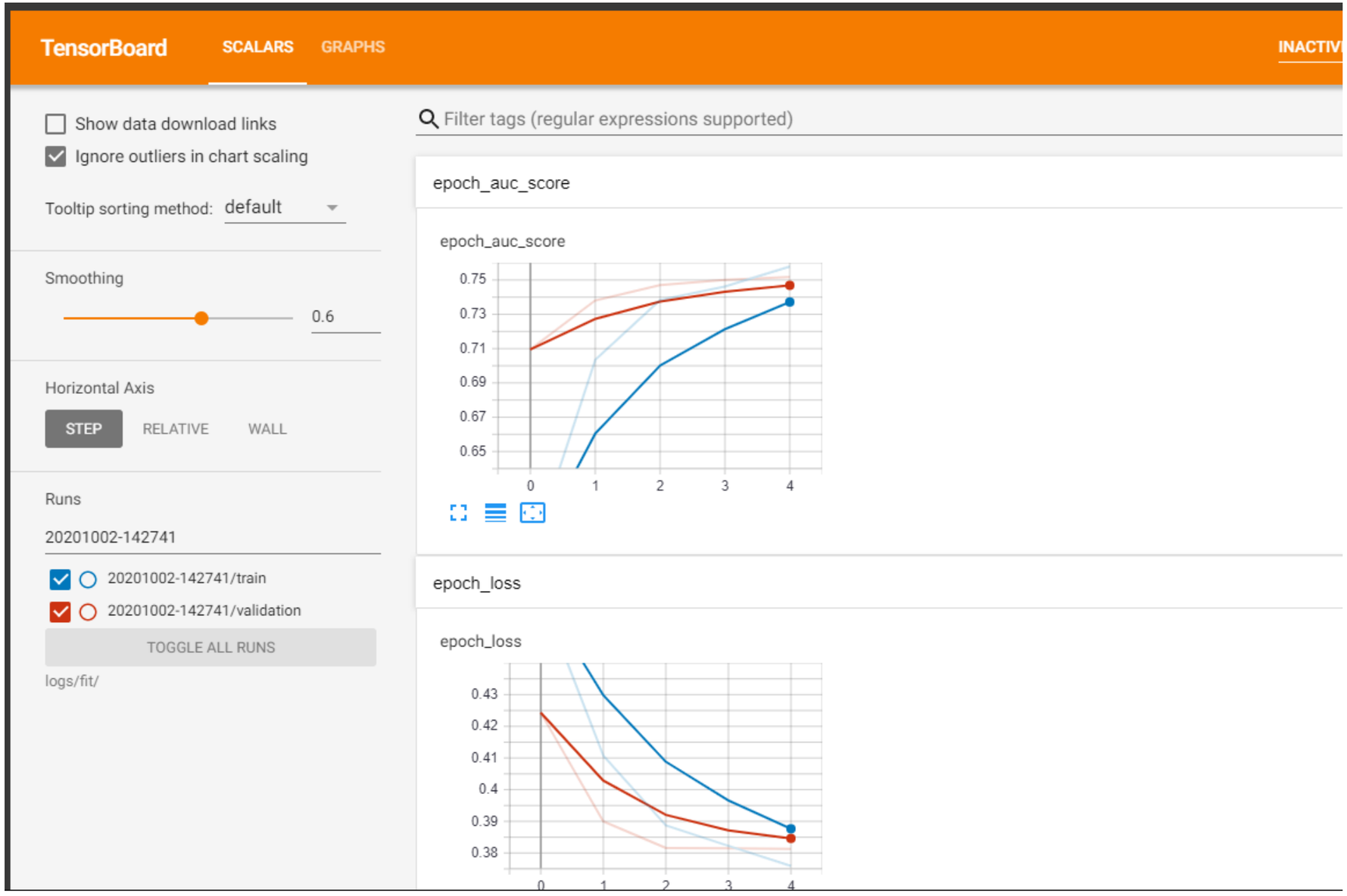
```
Epoch 1/20
   2/1144 [..............................] - ETA: 3:16 - loss: 1.1469 - auc_score: 0.4311WARNING:tensorflow:Callbacks method `
slow compared to the batch time (batch time: 0.0971s vs `on_train_batch_end` time: 0.2473s). Check your callbacks.
1143/1144 [============================>.] - ETA: 0s - loss: 0.4617 - auc_score: 0.5887
Epoch 00001: val_auc_score improved from -inf to 0.70954, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/bes
1144/1144 [==============================] - 68s 60ms/step - loss: 0.4617 - auc_score: 0.5888 - val_loss: 0.4243 - val_auc_scc
Epoch 2/20
1143/1144 [============================>.] - ETA: 0s - loss: 0.4107 - auc_score: 0.7034
Epoch 00002: val_auc_score improved from 0.70954 to 0.73805, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 68s 59ms/step - loss: 0.4107 - auc_score: 0.7035 - val_loss: 0.3900 - val_auc_scc
Epoch 3/20
1143/1144 [============================>.] - ETA: 0s - loss: 0.3886 - auc_score: 0.7384
Epoch 00003: val_auc_score improved from 0.73805 to 0.74695, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 68s 59ms/step - loss: 0.3887 - auc_score: 0.7383 - val_loss: 0.3816 - val_auc_scc
Epoch 4/20
1143/1144 [============================>.] - ETA: 0s - loss: 0.3823 - auc_score: 0.7461
Epoch 00004: val_auc_score improved from 0.74695 to 0.75004, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/
1144/1144 [==============================] - 70s 62ms/step - loss: 0.3822 - auc_score: 0.7462 - val_loss: 0.3815 - val_auc_scc
Epoch 5/20
1144/1144 [==============================] - ETA: 0s - loss: 0.3759 - auc_score: 0.7578
Epoch 00005: val_auc_score improved from 0.75004 to 0.75167, saving model to /gdrive/My Drive/LSTM/LSTM Assignment/save_model/

Reached 75.00% accuracy, so stopping training!!
1144/1144 [==============================] - 69s 60ms/step - loss: 0.3759 - auc_score: 0.7578 - val_loss: 0.3813 - val_auc_scc
```

Out[46]:
```
<tensorflow.python.keras.callbacks.History at 0x7f0bd0ddfc50>
```

In [ ]:
```
os.chdir('/gdrive/My Drive/LSTM/LSTM Assignment')
```

In [ ]:
```
%tensorboard --logdir logs/fit/
```

```
<IPython.core.display.Javascript object>
```


TensorBoard
SCALARS
GRAPHS
INACTIV
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
20201002-142741
20201002-142741/train
20201002-142741/validation
TOGGLE ALL RUNS
logs/fit/
Filter tags (regular expressions supported)
epoch_auc_score
epoch_auc_score
0.75
0.73
0.71
0.69
0.67
0.65
0
1
2
3
4
epoch_loss
epoch_loss
0.43
0.42
0.41
0.4
0.39
0.38
0
1
2
3
4

In [ ]:

In [ ]:

ROC AUC CURVE ROC graph drawn using True postive rate and True Postive rate to summarizse confusion matrix, which helps u
thershold value, and AUC which helps us to which catgorization is better. eg we drawn ROC using Decision tree and Random f
g both ROC we will predict skilled model

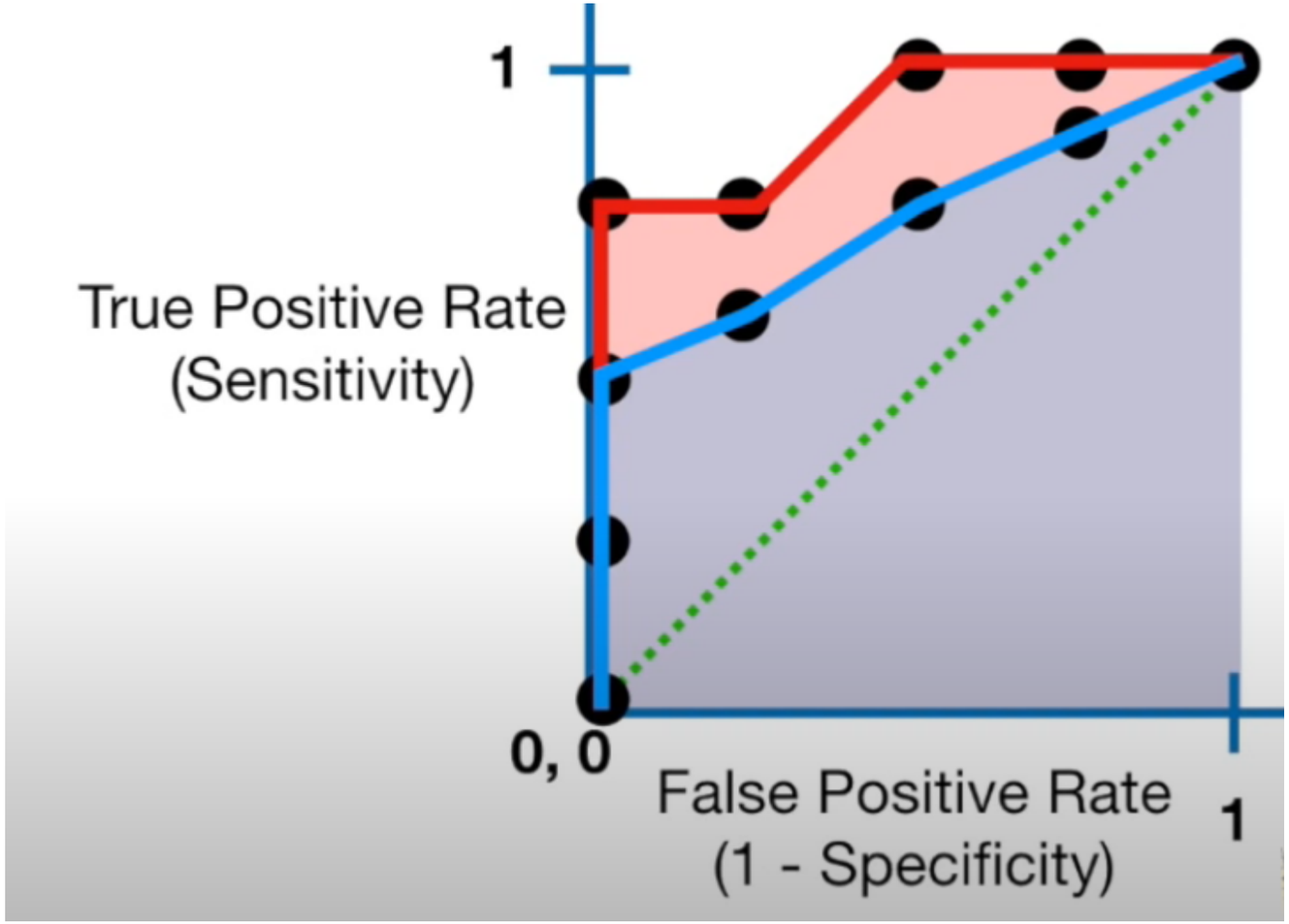
1. Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives.
2. Larger values on the y-axis of the plot indicate higher true positives and lower false negatives.

refer below snaps

True Positive Rate = True Positives / (True Positives + False Negatives)

False Positive Rate = False Positives / (False Positives + True Negatives)

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as
ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negativ



1. Model imporoved lot after using L2 regularizsation in layer level, because without using Regularizsation model overfit easliy
2. Played with Adam learning rate, 0.0008 gave the best result, as part of this task high learning doesnt help us much.
3. Tried learning rate scheduler my minimizse the adam learning rate after each 5 epoch model improved well but after 10 epoch model leads to overfit, be
4. We have highly balanced dataset so using ROC_AUC score to determine the perfromance.

```
[36] df.project_is_approved.value_counts()

1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [3]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Architecture", "Train AUC", "Test AUC"]
x.add_row(["Model-1", "0.755", "0.753"])
x.add_row(["Model-2", "0.722", "0.716"])
x.add_row(["Model-3", "0.757", "0.751"])
print(x)
```

```
+--------------+-----------+----------+
| Architecture | Train AUC | Test AUC |
+--------------+-----------+----------+
|   Model-1    |   0.755   |  0.753   |
|   Model-2    |   0.722   |  0.716   |
|   Model-3    |   0.757   |  0.751   |
+--------------+-----------+----------+
```

All three model did good job as part of classification , from my understanding model 3 will be more effective among th

Refer
https://datascience.stackexchange.com/questions/29851/one-hot-encoding-vs-word-embeding-when-to-choose-one-or-anot

MODEL 3 - BEST why? Here we using One hot encoding for categorical, One-Hot Encoding is a general method that can vectoriz features. It is simple and fast to create and update the vectorization, just add a new entry in the vector with a one for y. However, that speed and simplicity also leads to the "curse of dimensionality" by creating a new dimension for each cat

Why model 3 is better than model 1 ? One-Hot Encoding is a general method that can vectorize any categorical features. It to create and update the vectorization, just add a new entry in the vector with a one for each new category. However, that ity also leads to the "curse of dimensionality" by creating a new dimension for each category.

Embedding is a method that requires large amounts, both in the total amount of data and repeated occurrences of individual ng training time. The result is a dense vector with a fixed, arbitrary number of dimensions.

They also differ at the prediction stage a One-Hot Encoding tells you nothing of the semantics of the items. Each vectoriz onal representation in another dimension. Embeddings will group commonly co-occurring items together in the representation

If you have enough training data, enough training time, and the ability to apply the more complex training algorithm (e.g. e), go with Embeddings. Otherwise, fall back to One-Hot Encoding.

Why model 3 is better than model 2 ? Here we are removing more frequent and rare word from text esay, which may leads to p en we selected wrong thershold TFIDF value, and same as model 1 for catgorical representation we using using word embeddin

In [ ]: