## Assignment : 14

```
#you can use gdown modules to import dataset for the assignment
#for importing any file from drive to Colab you can write the syntax
as !gdown --id file_id
#you can run the below cell to import the required preprocessed
data.csv file and glove vector
```

## import section

```
import os
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from math import ceil
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from keras.utils import to_categorical
from sklearn.metrics import roc_auc_score,accuracy_score
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, Input, Dense, Flatten,
LSTM,Dropout, concatenate, Conv1D,BatchNormalization
from tensorflow.keras.callbacks import
ModelCheckpoint,EarlyStopping,LearningRateScheduler,ReduceLROnPlateau,
TensorBoard
from tensorflow.keras import
regularizers,initializers,optimizers,Model
from keras.callbacks import Callback, ModelCheckpoint
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
!pip install -U --no-cache-dir gdown --pre
from tqdm import tqdm
tqdm.pandas()
from tensorflow.keras.utils import plot_model
import datetime
from tensorflow.keras.optimizers import Adam
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gdown in /usr/local/lib/python3.8/dist-
packages (4.4.0)
Collecting gdown
  Downloading gdown-4.6.0-py3-none-any.whl (14 kB)
Requirement already satisfied: requests[socks] in
/usr/local/lib/python3.8/dist-packages (from gdown) (2.23.0)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-
packages (from gdown) (1.15.0)
Requirement already satisfied: beautifulsoup4 in
/usr/local/lib/python3.8/dist-packages (from gdown) (4.6.3)
Requirement already satisfied: filelock in
/usr/local/lib/python3.8/dist-packages (from gdown) (3.8.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-
packages (from gdown) (4.64.1)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown)
(2022.12.7)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
in /usr/local/lib/python3.8/dist-packages (from requests[socks]-
>gdown) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown)
(3.0.4)
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown)
(2.10)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in
/usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown)
(1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.4.0
    Uninstalling gdown-4.4.0:
      Successfully uninstalled gdown-4.4.0
Successfully installed gdown-4.6.0
```

## Lets import data

```
!gdown 1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
!gdown 1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_
```

```
Downloading...
From: https://drive.google.com/uc?id=1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
To: /content/preprocessed_data.csv
100% 124M/124M [00:01<00:00, 79.9MB/s]
Downloading...
From: https://drive.google.com/uc?id=1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_
To: /content/glove_vectors
100% 128M/128M [00:00<00:00, 229MB/s]
```

## Lets Load the processed data

```
preprocessed_data=pd.read_csv('/content/preprocessed_data.csv')
preprocessed_data.head()
```

```
  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2
1           ut             ms             grades_3_5
2           ca            mrs          grades_prek_2
3           ga            mrs          grades_prek_2
4           wa            mrs             grades_3_5

   teacher_number_of_previously_posted_projects
project_is_approved  \
0                                            53                    1

1                                             4                    1

2                                            10                    1

3                                             2                    1

4                                             2                    1


    clean_categories                clean_subcategories  \
0       math_science  appliedsciences health_lifescience
1       specialneeds                     specialneeds
2  literacy_language                         literacy
3    appliedlearning                  earlydevelopment
4  literacy_language                         literacy

                                              essay   price
0  i fortunate enough use fairy tale stem kits cl...  725.05
1  imagine 8 9 years old you third grade classroo...  213.03
2  having class 24 students comes diverse learner...  329.00
3  i recently read article giving students choice...  481.04
4  my students crave challenge eat obstacles brea...   17.74
```

## Lets load glove vectors

```
with open('/content/glove_vectors', 'rb') as f:
    glove_vector = pickle.load(f)
```

```
result_dataset = pd.DataFrame(data = np.zeros((3,3)),index =
['model1','model2','model3'],columns = ['AUC', 'Loss',"accuracy"])
result_dataset
```

```
        AUC  Loss  accuracy
model1  0.0   0.0       0.0
model2  0.0   0.0       0.0
model3  0.0   0.0       0.0
```

##Lets understand preprocessed data

```
preprocessed_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 9 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   project_is_approved                             109248 non-null
int64
 5   clean_categories                                109248 non-null
object
 6   clean_subcategories                             109248 non-null
object
 7   essay                                           109248 non-null
object
 8   price                                           109248 non-null
float64
dtypes: float64(1), int64(2), object(6)
memory usage: 7.5+ MB
```

## Lets split data into train and test sets

```
y = preprocessed_data.project_is_approved
X = preprocessed_data.drop(columns='project_is_approved')

X_train, X_test,y_train,y_test = train_test_split(X,y,stratify =
y,random_state = 45,test_size = 0.2)
X_train = X_train.reset_index(drop = True)
X_test = X_test.reset_index(drop = True)
y_train = y_train.reset_index(drop = True)
y_test = y_test.reset_index(drop = True)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(87398, 8)
(21850, 8)
(87398,)
(21850,)
```

## 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 glove word vectors, don't 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** ---concatenate remaining columns and add a Dense layer after that.

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

```
# https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
#input_layer = Input(shape=(n,))
#embedding = Embedding(no_1, no_2, input_length=n)(input_layer)
#flatten = Flatten()(embedding)
```

**1. Go through this blog, if you have any doubt on using predefined Embedding values in Embedding layer - https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/**

**2. Please go through this link https://keras.io/getting-started/functional-api-guide/ and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.**

## Model-1

### 1.1 Text Vectorization

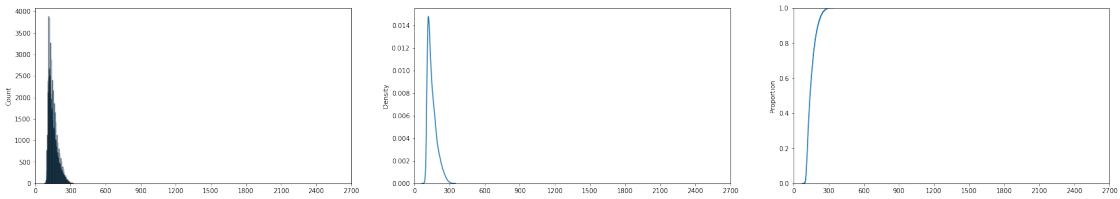
```
#since the data is already preprocessed, we can directly move to vectorization part
```

```
#first we will vectorize the text data
#for vectorization of text data in deep learning we use tokenizer, you
can go through below references
# https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenization-
text-data-prep.html
#https://stackoverflow.com/questions/51956000/what-does-keras-
tokenizer-method-exactly-do
# after text vectorization you should get train_padded_docs and
test_padded_docs
```

### Essay Vectorisation

```
train_essay_lengths = []
for essay in tqdm(X_train.essay):
  train_essay_lengths.append(len(essay.split()))
fig, axs = plt.subplots(ncols=3,figsize=(30,5))
a = sns.histplot(data = train_essay_lengths,ax=axs[0])
a.set_xticks(range(0,3000,300))
b = sns.kdeplot(data = train_essay_lengths,ax=axs[1])
b.set_xticks(range(0,3000,300))
c = sns.ecdfplot(data = train_essay_lengths,ax=axs[2])
c.set_xticks(range(0,3000,300))
plt.show()
```

```
100%|██████████| 87398/87398 [00:00<00:00, 121248.91it/s]
```



### 99.9 % percentage of data have words less than 350 hence we can select maximum word length as 350.

```
max_essay_length = 350
tokenizer_Essay = Tokenizer(oov_token='<oov>')
tokenizer_Essay.fit_on_texts(X_train.essay.to_list())
X_train_essay =
pad_sequences(tokenizer_Essay.texts_to_sequences(X_train.essay),
maxlen=max_essay_length,padding='post',truncating='post')
X_test_essay =
pad_sequences(tokenizer_Essay.texts_to_sequences(X_test.essay),
maxlen=max_essay_length,padding='post',truncating='post')

#after getting the padded_docs you have to use predefined glove
vectors to get 300 dim representation for each word
# we will be storing this data in form of an embedding matrix and will
use it while defining our model
# Please go through following blog's 'Example of Using Pre-Trained
GloVe Embedding' section to understand how to create embedding matrix
# https://machinelearningmastery.com/use-word-embedding-layers-deep-
```

```
learning-keras/


EMBEDDING_DIMS = 300     # glove vectors are 300 dims
VOCAB_SIZE = len(list(tokenizer_Essay.word_counts.keys()))
embedding_matrix = np.zeros((VOCAB_SIZE+1, EMBEDDING_DIMS))
for word, i in tokenizer_Essay.word_index.items():
    embedding_vector = glove_vector.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i-1] = embedding_vector
```

## 1.2 Categorical feature Vectorization

```
# for model 1 and model 2, we have to assign a unique number to each
feature in a particular categorical column.
# you can either use tokenizer,label encoder or ordinal encoder to
perform the task
# label encoder gives an error for 'unseen values' (values present in
test but not in train)
# handle unseen values with label encoder -
https://stackoverflow.com/a/56876351
# ordinal encoder also gives error with unseen values but you can use
modify handle_unknown parameter
# documentation of ordianl encoder
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessin
g.OrdinalEncoder.html
# after categorical feature vectorization you will have
column_train_data and column_test_data.
```

###school_state

```
tokenizer_schoolState = Tokenizer(oov_token='<oov>')
tokenizer_schoolState.fit_on_texts(X_train.school_state.to_list())
max_schoolState_length = X_train.school_state.apply(lambda x :
len(x.split(' '))).max()
X_train_schoolState =
pad_sequences(tokenizer_schoolState.texts_to_sequences(X_train.school_
state),maxlen=max_schoolState_length,padding='post',truncating='post')
X_test_schoolState =
pad_sequences(tokenizer_schoolState.texts_to_sequences(X_test.school_s
tate),maxlen=max_schoolState_length,padding='post',truncating='post')

tokenizer_schoolState.word_index

{'<oov>': 1,
 'ca': 2,
 'tx': 3,
 'ny': 4,
 'fl': 5,
 'nc': 6,
 'il': 7,
```

```
 'sc': 8,
 'ga': 9,
 'mi': 10,
 'pa': 11,
 'in': 12,
 'mo': 13,
 'oh': 14,
 'ma': 15,
 'la': 16,
 'wa': 17,
 'ok': 18,
 'nj': 19,
 'az': 20,
 'va': 21,
 'wi': 22,
 'al': 23,
 'ut': 24,
 'tn': 25,
 'ct': 26,
 'md': 27,
 'nv': 28,
 'ms': 29,
 'ky': 30,
 'or': 31,
 'mn': 32,
 'co': 33,
 'ar': 34,
 'id': 35,
 'ia': 36,
 'ks': 37,
 'nm': 38,
 'dc': 39,
 'hi': 40,
 'me': 41,
 'wv': 42,
 'de': 43,
 'nh': 44,
 'ak': 45,
 'ne': 46,
 'sd': 47,
 'ri': 48,
 'mt': 49,
 'nd': 50,
 'wy': 51,
 'vt': 52}
```

###teacher_prefix

```
tokenizer_teacherprefix = Tokenizer(oov_token='<oov>')
tokenizer_teacherprefix.fit_on_texts(X_train.teacher_prefix.to_list())
max_teacherprefix_length = X_train.teacher_prefix.apply(lambda x :
```

```
len(x.split(' '))).max()
X_train_teacherprefix =
pad_sequences(tokenizer_teacherprefix.texts_to_sequences(X_train.teach
er_prefix),maxlen=max_teacherprefix_length,padding='post',truncating='
post')
X_test_teacherprefix =
pad_sequences(tokenizer_teacherprefix.texts_to_sequences(X_test.teache
r_prefix),maxlen=max_teacherprefix_length,padding='post',truncating='p
ost')

tokenizer_teacherprefix.word_index

{'<oov>': 1, 'mrs': 2, 'ms': 3, 'mr': 4, 'teacher': 5, 'dr': 6}
```

###project_grade_category

```
tokenizer_project_grade_category=
Tokenizer(oov_token='<oov>',filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\
t\n')
tokenizer_project_grade_category.fit_on_texts(X_train.project_grade_ca
tegory.to_list())
max_project_grade_category_length =
X_train.project_grade_category.apply(lambda x : len(x.split('
'))).max()
X_train_project_grade_category =
pad_sequences(tokenizer_project_grade_category.texts_to_sequences(X_tr
ain.project_grade_category),maxlen=max_project_grade_category_length,p
adding='post',truncating='post')
X_test_project_grade_category =
pad_sequences(tokenizer_project_grade_category.texts_to_sequences(X_te
st.project_grade_category),maxlen=max_project_grade_category_length,pa
dding='post',truncating='post')

tokenizer_project_grade_category.word_index

{'<oov>': 1,
 'grades_prek_2': 2,
 'grades_3_5': 3,
 'grades_6_8': 4,
 'grades_9_12': 5}
```

###clean_categories

```
tokenizer_clean_categories= Tokenizer(oov_token='<oov>',filters='!"#$
%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
tokenizer_clean_categories.fit_on_texts(X_train.clean_categories.to_li
st())
max_clean_categories_length = X_train.clean_categories.apply(lambda
x : len(x.split(' '))).max()
X_train_clean_categories =
pad_sequences(tokenizer_clean_categories.texts_to_sequences(X_train.cl
ean_categories),maxlen=max_clean_categories_length,padding='post',trun
```

```
cating='post')
X_test_clean_categories =
pad_sequences(tokenizer_clean_categories.texts_to_sequences(X_test.cle
an_categories),maxlen=max_clean_categories_length,padding='post',trunc
ating='post')

tokenizer_clean_categories.word_index

{'<oov>': 1,
 'literacy_language': 2,
 'math_science': 3,
 'health_sports': 4,
 'specialneeds': 5,
 'appliedlearning': 6,
 'music_arts': 7,
 'history_civics': 8,
 'warmth': 9,
 'care_hunger': 10}
```

### clean_subcategories

```
tokenizer_clean_subcategories=
Tokenizer(oov_token='<oov>',filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\
t\n')
tokenizer_clean_subcategories.fit_on_texts(X_train.clean_subcategories
.to_list())
max_clean_subcategories_length =
X_train.clean_subcategories.apply(lambda x : len(x.split(' '))).max()
X_train_clean_subcategories =
pad_sequences(tokenizer_clean_subcategories.texts_to_sequences(X_train
.clean_subcategories),maxlen=max_clean_subcategories_length,padding='p
ost',truncating='post')
X_test_clean_subcategories =
pad_sequences(tokenizer_clean_subcategories.texts_to_sequences(X_test.
clean_subcategories),maxlen=max_clean_subcategories_length,padding='po
st',truncating='post')

tokenizer_clean_subcategories.word_index

{'<oov>': 1,
 'literacy': 2,
 'mathematics': 3,
 'literature_writing': 4,
 'specialneeds': 5,
 'appliedsciences': 6,
 'health_wellness': 7,
 'visualarts': 8,
 'environmentalscience': 9,
 'gym_fitness': 10,
 'esl': 11,
 'health_lifescience': 12,
 'earlydevelopment': 13,
```

```
 'history_geography': 14,
 'music': 15,
 'college_careerprep': 16,
 'other': 17,
 'teamsports': 18,
 'charactereducation': 19,
 'performingarts': 20,
 'socialsciences': 21,
 'warmth': 22,
 'care_hunger': 23,
 'nutritioneducation': 24,
 'foreignlanguages': 25,
 'extracurricular': 26,
 'civics_government': 27,
 'parentinvolvement': 28,
 'financialliteracy': 29,
 'communityservice': 30,
 'economics': 31}
```

### 1.3 Numerical feature Vectorization

```
# you have to standardise the numerical columns
# stack both the numerical features
#after numerical feature vectorization you will have
numerical_data_train and numerical_data_test

X_train_numericals =
X_train[['teacher_number_of_previously_posted_projects','price']]
X_test_numericals =
X_test[['teacher_number_of_previously_posted_projects','price']]
scalar = MinMaxScaler()
X_train_numericals = scalar.fit_transform(X_train_numericals)
X_test_numericals = scalar.transform(X_test_numericals)
```

### 1.4 Defining the model

```
# as of now we have vectorized all our features now we will define our
model.
# as it is clear from above image that the given model has multiple
input layers and hence we have to use functional API
# Please go through - https://keras.io/guides/functional_api/
# it is a good programming practise to define your complete model i.e
all inputs , intermediate and output layers at one place.
# while defining your model make sure that you use variable names
while defining any length,dimension or size.
#for ex.- you should write the code as 'input_text =
Input(shape=(pad_length,))' and not as 'input_text =
Input(shape=(300,))'
# the embedding layer for text data should be non trainable
```

```
# the embedding layer for categorical data should be trainable
# https://stats.stackexchange.com/questions/270546/how-does-keras-
embedding-layer-work
# https://towardsdatascience.com/deep-embeddings-for-categorical-
variables-cat2vec-b05c8ab63ac0
#print model.summary() after you have defined the model
#plot the model using utils.plot_model module and make sure that it is
similar to the above image

essay block
essay_Inp =
Input(shape=(max_essay_length,),dtype='int32',name='essay_Inp')
essay_ouput_dimensions = 300
embedded_Essay =
Embedding(input_dim=len(tokenizer_Essay.word_index.items()),output_dim
=essay_ouput_dimensions,name='embedded_Essay',weights=[embedding_matri
x],trainable=False)(essay_Inp)
essay_LSTM = LSTM(units=128,return_sequences = True)(embedded_Essay)
essay_LSTM_2 = LSTM(units=60,return_sequences = True)(essay_LSTM)
essay_Out = tf.math.reduce_mean(essay_LSTM_2, axis = -1)
#essay_Out = Flatten()(average_out)

school_state block
school_state_inp = Input(shape=(max_schoolState_length,),
dtype='int32',name='school_state_inp')
embedded_school_state =
Embedding(input_dim=len(tokenizer_schoolState.word_index.items()),outp
ut_dim=6,name='embedded_school_state')(school_state_inp)
school_state_out = Flatten()(embedded_school_state)

teacher_prefix block
teacher_Pref_inp =
Input(shape=(max_teacherprefix_length,),dtype='int32',name='teacher_Pr
ef_inp')
embedded_teacher_Pref =
Embedding(input_dim=len(tokenizer_teacherprefix.word_index.items()),ou
tput_dim=2,name='embedded_teacher_Pref')(teacher_Pref_inp)
teacher_Pref_out = Flatten()(embedded_teacher_Pref)

project grade category block
pg_category_inp =
Input(shape=(max_project_grade_category_length,),dtype='int32',name='p
gCategory_Inp')
embedded_pgCategory =
Embedding(input_dim=len(tokenizer_project_grade_category.word_index.it
ems()),output_dim=2,name='embedded_pgCategory')(pg_category_inp)
pgCategory_Out = Flatten()(embedded_pgCategory)
```

*project clean category block*

```
cleanCategory_Inp =
Input(shape=(max_clean_categories_length,),dtype='int32',name='cleanCa
tegory_Inp')
embedded_cleanCategory =
Embedding(input_dim=len(tokenizer_clean_categories.word_index.items())
,output_dim=3,name='embedded_cleanCategory')(cleanCategory_Inp)
cleanCategory_Out = Flatten()(embedded_cleanCategory)
```

*project clean sub category block*

```
clean_subcategories_Inp =
Input(shape=(max_clean_subcategories_length,),dtype='int32',name='clea
n_subcategories_Inp')
embedded_cleanSubCategory =
Embedding(input_dim=len(tokenizer_clean_subcategories.word_index.items
()),output_dim=5,name='embedded_cleanSubCategory')
(clean_subcategories_Inp)
clean_subcategories_Out = Flatten()(embedded_cleanSubCategory)
```

*Numerical columns block*

```
numerical_input = Input(shape = (X_train_numericals.shape[1],),
dtype='float32', name='numericals_Inp')
numerical_dense_output = Dense(units =
16,activation='relu',kernel_initializer='he_normal')(numerical_input)
```

*concatenating all the above blocks outputs*

```
concatenated_Outs = concatenate([essay_Out,school_state_out,
teacher_Pref_out,pgCategory_Out,cleanCategory_Out,clean_subcategories_
Out,numerical_dense_output])

dense1 =
Dense(128,activation='relu',kernel_initializer='he_normal',kernel_regu
larizer=regularizers.l2(0.001))(concatenated_Outs)
dropout1 = Dropout(0.4)(dense1)
dense2 =
Dense(64,activation='relu',kernel_initializer='he_normal',kernel_regul
arizer=regularizers.l2(0.001))(dropout1)
dropout2 = Dropout(0.4)(dense2)
batchnorm1 = BatchNormalization()(dropout2)
dense3 =
Dense(32,activation='relu',kernel_initializer='he_normal',kernel_regul
arizer=regularizers.l2(0.001))(batchnorm1)
dropout3 = Dropout(0.4)(dense3)
output = Dense(1, activation = 'sigmoid')(dropout3)
```

## 1.5 Compiling and fititng your model

```
#define custom auc as metric , do not use tf.keras.metrics
# https://stackoverflow.com/a/46844409 - custom AUC reference 1
# https://www.kaggle.com/c/santander-customer-transaction-prediction/
```

```
discussion/80807  - custom AUC reference 2
# compile and fit your model
```

### Custom AUC Metric Function

```
def auc_temp(y_true, y_pred):
    if len(np.unique(y_true)) == 1:
        return 0.5
    else:
        return roc_auc_score(y_true, y_pred, average='micro')

def auc(y_true, y_pred):
    return tf.py_function(auc_temp, (y_true, y_pred), tf.double)

# create model with all the previously defined inputs
model1 =
Model([essay_Inp,school_state_inp,teacher_Pref_inp,pg_category_inp,cle
anCategory_Inp,clean_subcategories_Inp,numerical_input], output)
model1.compile(loss='binary_crossentropy',optimizer=Adam(lr =
0.001),metrics=[auc,'accuracy'])
print(model1.summary())
```

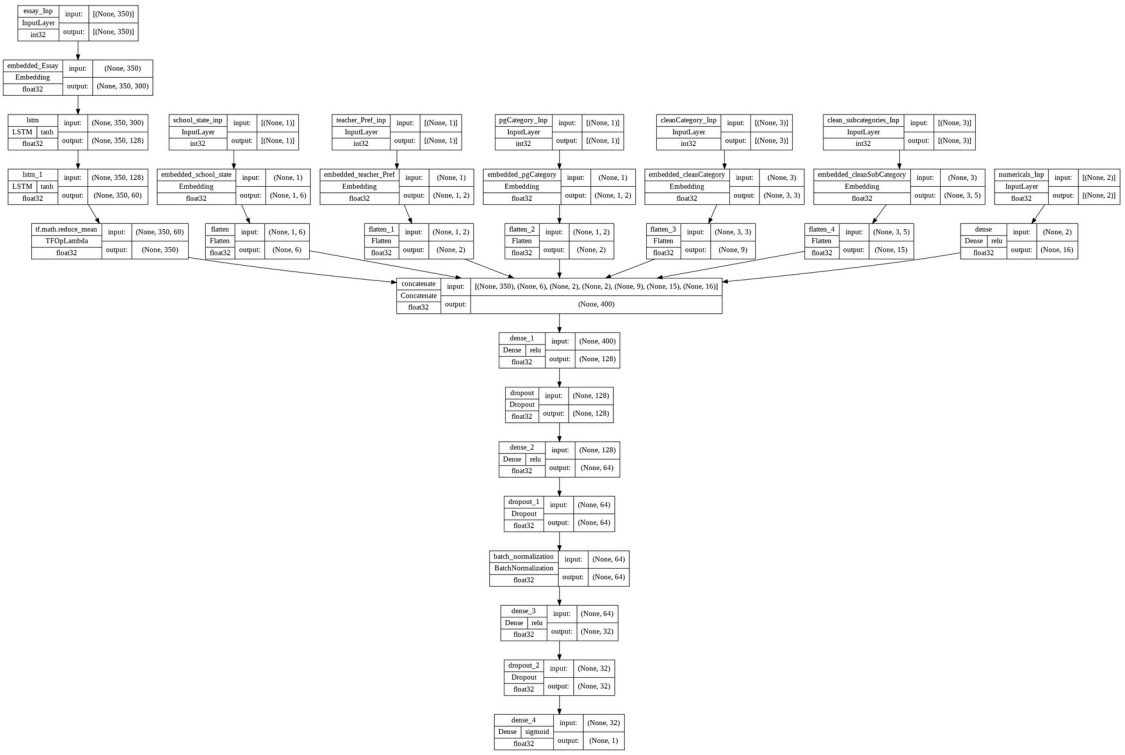
```
Model: "model"
__________________________________________________________________________
_____
 Layer (type)                   Output Shape         Param #
Connected to
==========================================================================
=============================
 essay_Inp (InputLayer)         [(None, 350)]        0           []



 embedded_Essay (Embedding)     (None, 350, 300)     15531600
['essay_Inp[0][0]']


 lstm (LSTM)                    (None, 350, 128)     219648
['embedded_Essay[0][0]']


 school_state_inp (InputLayer)  [(None, 1)]          0           []



 teacher_Pref_inp (InputLayer)  [(None, 1)]          0           []



 pgCategory_Inp (InputLayer)    [(None, 1)]          0           []
```

```
 cleanCategory_Inp (InputLayer)  [(None, 3)]         0           []


 clean_subcategories_Inp (Input  [(None, 3)]         0           []

 Layer)


 lstm_1 (LSTM)                  (None, 350, 60)      45360
['lstm[0][0]']


 embedded_school_state (Embeddi  (None, 1, 6)        312
['school_state_inp[0][0]']
 ng)


 embedded_teacher_Pref (Embeddi  (None, 1, 2)        12
['teacher_Pref_inp[0][0]']
 ng)


 embedded_pgCategory (Embedding  (None, 1, 2)        10
['pgCategory_Inp[0][0]']
 )


 embedded_cleanCategory (Embedd  (None, 3, 3)        30
['cleanCategory_Inp[0][0]']
 ing)


 embedded_cleanSubCategory (Emb  (None, 3, 5)        155
['clean_subcategories_Inp[0][0]']
 edding)
```

```
 numericals_Inp (InputLayer)    [(None, 2)]          0           []



 tf.math.reduce_mean (TFOpLambd  (None, 350)         0
['lstm_1[0][0]']
 a)



 flatten (Flatten)              (None, 6)            0
['embedded_school_state[0][0]']


 flatten_1 (Flatten)            (None, 2)            0
['embedded_teacher_Pref[0][0]']


 flatten_2 (Flatten)            (None, 2)            0
['embedded_pgCategory[0][0]']


 flatten_3 (Flatten)            (None, 9)            0
['embedded_cleanCategory[0][0]']


 flatten_4 (Flatten)            (None, 15)           0
['embedded_cleanSubCategory[0][0]
                                                                  ']



 dense (Dense)                  (None, 16)           48
['numericals_Inp[0][0]']


 concatenate (Concatenate)      (None, 400)          0
['tf.math.reduce_mean[0][0]',

'flatten[0][0]',

'flatten_1[0][0]',

'flatten_2[0][0]',

'flatten_3[0][0]',

'flatten_4[0][0]',
```

```
'dense[0][0]']


 dense_1 (Dense)                (None, 128)          51328
['concatenate[0][0]']


 dropout (Dropout)              (None, 128)          0
['dense_1[0][0]']


 dense_2 (Dense)                (None, 64)           8256
['dropout[0][0]']


 dropout_1 (Dropout)            (None, 64)           0
['dense_2[0][0]']


 batch_normalization (BatchNorm  (None, 64)          256
['dropout_1[0][0]']
 alization)



 dense_3 (Dense)                (None, 32)           2080
['batch_normalization[0][0]']


 dropout_2 (Dropout)            (None, 32)           0
['dense_3[0][0]']


 dense_4 (Dense)                (None, 1)            33
['dropout_2[0][0]']


==================================================================
============================
Total params: 15,859,128
Trainable params: 327,400
Non-trainable params: 15,531,728
__________________________________________________________________
_____________________________
None
```

```
plot_model(model1,to_file = 'model1.png',show_shapes =
True,show_layer_activations = True,show_dtype = True)
```




## Lets define few call backs

```
filepath="model_one_save/weights-{epoch:02d}-{val_auc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc',
verbose=1, save_best_only=True, mode='max')

earlystop = EarlyStopping(monitor='val_auc', min_delta=0.35,
patience=5, verbose=1)

reduce_lr = ReduceLROnPlateau(monitor='val_auc',
factor=0.1,patience=5, min_lr=0.0000001)

%load_ext tensorboard
log_dir = os.path.join("model_one","logs",'fits',
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback =
TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)
%reload_ext tensorboard
```

## Lets fit Model_1

```
model1_history =
model1.fit([X_train_essay,X_train_schoolState,X_train_teacherprefix,X_
train_project_grade_category,X_train_clean_categories,X_train_clean_su
bcategories,X_train_numericals], np.array(y_train),
           epochs=5,verbose=1,batch_size=256,
           validation_data =
```

```
([X_test_essay,X_test_schoolState,X_test_teacherprefix,X_test_project_
grade_category,X_test_clean_categories,X_test_clean_subcategories,X_te
st_numericals], np.array(y_test)),
            callbacks =[checkpoint,reduce_lr,tensorboard_callback])
#validation_data =
([X_test_essay,X_test_schoolState,X_test_teacherprefix,X_test_project_
grade_category,X_test_clean_categories,X_test_clean_subcategories,X_te
st_numericals], np.array(y_test)),

Epoch 1/5
342/342 [==============================] - ETA: 0s - loss: 0.3236 -
auc: 0.8154 - accuracy: 0.8765
Epoch 1: val_auc did not improve from 0.75190
342/342 [==============================] - 35s 101ms/step - loss:
0.3236 - auc: 0.8154 - accuracy: 0.8765 - val_loss: 0.3944 - val_auc:
0.7416 - val_accuracy: 0.8313 - lr: 1.0000e-06
Epoch 2/5
341/342 [============================>.] - ETA: 0s - loss: 0.3232 -
auc: 0.8176 - accuracy: 0.8764
Epoch 2: val_auc did not improve from 0.75190
342/342 [==============================] - 36s 105ms/step - loss:
0.3231 - auc: 0.8175 - accuracy: 0.8764 - val_loss: 0.3946 - val_auc:
0.7415 - val_accuracy: 0.8311 - lr: 1.0000e-06
Epoch 3/5
342/342 [==============================] - ETA: 0s - loss: 0.3234 -
auc: 0.8151 - accuracy: 0.8758
Epoch 3: val_auc did not improve from 0.75190
342/342 [==============================] - 34s 98ms/step - loss:
0.3234 - auc: 0.8151 - accuracy: 0.8758 - val_loss: 0.3945 - val_auc:
0.7415 - val_accuracy: 0.8312 - lr: 1.0000e-06
Epoch 4/5
342/342 [==============================] - ETA: 0s - loss: 0.3236 -
auc: 0.8155 - accuracy: 0.8771
Epoch 4: val_auc did not improve from 0.75190
342/342 [==============================] - 36s 104ms/step - loss:
0.3236 - auc: 0.8155 - accuracy: 0.8771 - val_loss: 0.3945 - val_auc:
0.7415 - val_accuracy: 0.8314 - lr: 1.0000e-06
Epoch 5/5
342/342 [==============================] - ETA: 0s - loss: 0.3235 -
auc: 0.8171 - accuracy: 0.8758
Epoch 5: val_auc did not improve from 0.75190
342/342 [==============================] - 33s 97ms/step - loss:
0.3235 - auc: 0.8171 - accuracy: 0.8758 - val_loss: 0.3947 - val_auc:
0.7415 - val_accuracy: 0.8316 - lr: 1.0000e-06

model1.load_weights('/content/model_one_save/weights-07-0.7519.hdf5')

model1_scores =
model1.evaluate([X_test_essay,X_test_schoolState,X_test_teacherprefix,
X_test_project_grade_category,X_test_clean_categories,X_test_clean_sub
categories,X_test_numericals], np.array(y_test))
```

```
print("Model 1 loss : ",model1_scores[0])
print("Model 1 auc : ",model1_scores[1])
print("Model 1 accuracy : ",model1_scores[2])
result_dataset['AUC']['model1'] = model1_scores[1]
result_dataset['Loss']['model1'] = model1_scores[0]
result_dataset['accuracy']['model1'] = model1_scores[2]
result_dataset
```

```
683/683 [==============================] - 13s 19ms/step - loss:
0.3819 - auc: 0.7532 - accuracy: 0.8470
Model 1 loss :  0.3818551003932953
Model 1 auc :  0.7531737685203552
Model 1 accuracy :  0.8469565510749817
```

| | AUC | Loss | accuracy |
|---|---|---|---|
| model1 | 0.753174 | 0.381855 | 0.846957 |
| model2 | 0.000000 | 0.000000 | 0.000000 |
| model3 | 0.000000 | 0.000000 | 0.000000 |

```
%tensorboard --logdir model_one/logs/fits
```

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

## 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.

```
tfIdf_vectorizer = TfidfVectorizer(min_df=3)
tfIdf_vectorizer.fit_transform(X_train.essay.to_list())
```

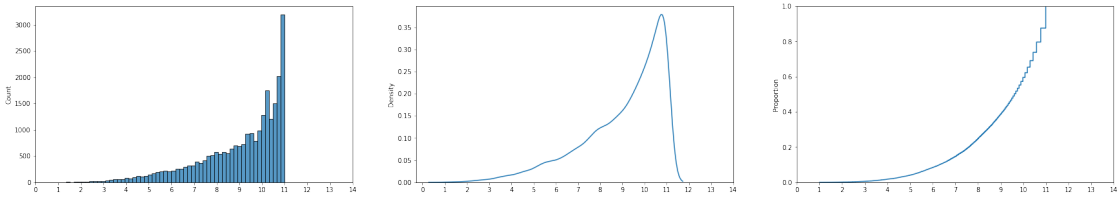
```
<87398x25598 sparse matrix of type '<class 'numpy.float64'>'
     with 9416361 stored elements in Compressed Sparse Row format>
```

```
feature_Idf_Map =
dict(zip(tfIdf_vectorizer.get_feature_names(),tfIdf_vectorizer.idf_))
```

```
len(tfIdf_vectorizer.get_feature_names())
```

```
25598
```

```
# lets see how idf values are distributed
fig, axs = plt.subplots(ncols=3,figsize=(30,5))
a = sns.histplot(data = tfIdf_vectorizer.idf_,ax=axs[0])
a.set_xticks(range(0,15,1))
b = sns.kdeplot(data = tfIdf_vectorizer.idf_,ax=axs[1])
b.set_xticks(range(0,15,1))
c = sns.ecdfplot(data = tfIdf_vectorizer.idf_,ax=axs[2])
c.set_xticks(range(0,15,1))
plt.show()
```

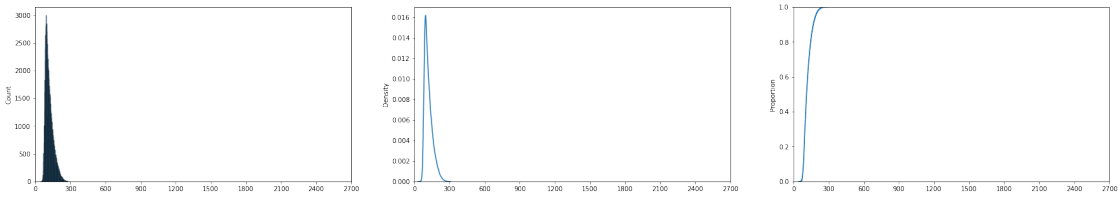


## from the above image it is clearly evident that idfvalues less than 2 are very less hence lets take idf values from 2 to 11

```
llimit, hlimit = 2, 11
def exclude_words(text):
  return ' '.join([word for word in text.split() if
feature_Idf_Map.get(word) is None or (feature_Idf_Map.get(word) and
(feature_Idf_Map.get(word)>=llimit and
feature_Idf_Map.get(word)<=hlimit))])

X_train['modified_essay']=X_train.essay.progress_apply(exclude_words)

100%|██████████| 87398/87398 [00:11<00:00, 7844.35it/s]

X_test['modified_essay'] = X_test.essay.progress_apply(exclude_words)

100%|██████████| 21850/21850 [00:02<00:00, 8312.79it/s]
```

### Essay Vectorisation for model 2

```
train_modified_essay_lengths = []
for essay in tqdm(X_train.modified_essay):
  train_modified_essay_lengths.append(len(essay.split()))
fig, axs = plt.subplots(ncols=3,figsize=(30,5))
a = sns.histplot(data = train_modified_essay_lengths,ax=axs[0])
a.set_xticks(range(0,3000,300))
b = sns.kdeplot(data = train_modified_essay_lengths,ax=axs[1])
b.set_xticks(range(0,3000,300))
c = sns.ecdfplot(data = train_modified_essay_lengths,ax=axs[2])
c.set_xticks(range(0,3000,300))
plt.show()

100%|██████████| 87398/87398 [00:00<00:00, 149098.28it/s]
```



## 99.9 % percentage of data have words less than 300 hence we can select maximum word length as 300.

```
max_modified_essay_length = 300
tokenizer_modified_Essay = Tokenizer(oov_token='<oov>')
tokenizer_modified_Essay.fit_on_texts(X_train.modified_essay.to_list()
```

```
)
X_train_modified_essay =
pad_sequences(tokenizer_modified_Essay.texts_to_sequences(X_train.modi
fied_essay),
maxlen=max_modified_essay_length,padding='post',truncating='post')
X_test_modified_essay =
pad_sequences(tokenizer_modified_Essay.texts_to_sequences(X_test.modif
ied_essay),
maxlen=max_modified_essay_length,padding='post',truncating='post')

EMBEDDING_DIMS = 300     # glove vectors are 300 dims
VOCAB_SIZE = len(list(tokenizer_modified_Essay.word_counts.keys()))
embedding_matrix_model2 = np.zeros((VOCAB_SIZE+1, EMBEDDING_DIMS))
for word, i in tokenizer_modified_Essay.word_index.items():
    embedding_vector_model2 = glove_vector.get(word)
    if embedding_vector_model2 is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix_model2[i-1] = embedding_vector_model2

# Modified essay
modified_essay_Inp =
Input(shape=(max_modified_essay_length,),dtype='int32',name='modified_
essay_Inp')
embedded_mod_Essay =
Embedding(input_dim=len(tokenizer_modified_Essay.word_index.items()),
                        output_dim=300,name='embedded_mod_Essay',
                        weights=[embedding_matrix_model2],
                        trainable=False)(modified_essay_Inp)
modified_essay_LSTM = LSTM(units=128,return_sequences = True)
(embedded_mod_Essay)
modified_essay_LSTM_2 = LSTM(units=60,return_sequences = True)
(modified_essay_LSTM)
modified_essay_Out = tf.math.reduce_mean(modified_essay_LSTM_2, axis =
-1)

concatenated_Outs_modified =
concatenate([modified_essay_Out,school_state_out,
teacher_Pref_out,pgCategory_Out,cleanCategory_Out,clean_subcategories_
Out,numerical_dense_output])

mod_dense1 =
Dense(128,activation='relu',kernel_initializer='he_normal',kernel_regu
larizer=regularizers.l2(0.001))(concatenated_Outs_modified)
mod_dropout1 = Dropout(0.4)(mod_dense1)
mod_dense2 =
Dense(64,activation='relu',kernel_initializer='he_normal',kernel_regul
arizer=regularizers.l2(0.001))(mod_dropout1)
mod_dropout2 = Dropout(0.4)(mod_dense2)
mod_batchnorm1 = BatchNormalization()(mod_dropout2)
mod_dense3 =
Dense(32,activation='relu',kernel_initializer='he_normal',kernel_regul
```

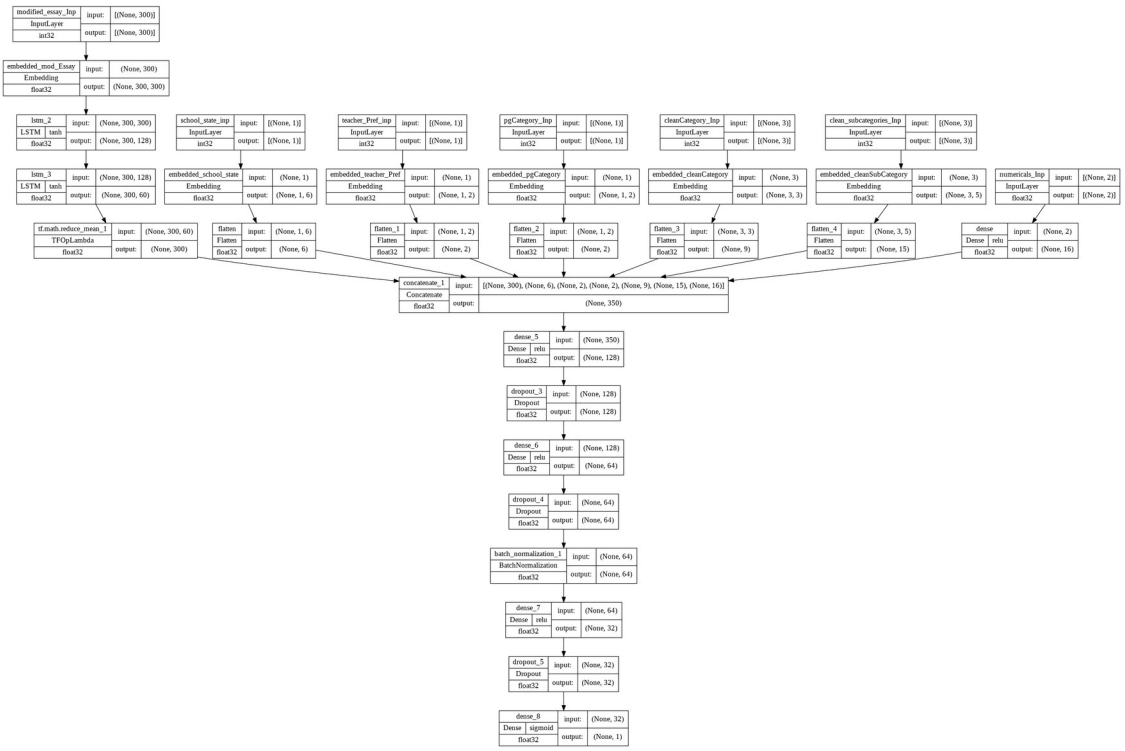
```
arizer=regularizers.l2(0.001))(mod_batchnorm1)
mod_dropout3 = Dropout(0.4)(mod_dense3)
mod_output = Dense(1, activation = 'sigmoid')(mod_dropout3)

# create model with all the previously defined inputs
model2 =
Model([modified_essay_Inp,school_state_inp,teacher_Pref_inp,pg_categor
y_inp,cleanCategory_Inp,clean_subcategories_Inp,numerical_input],
mod_output)
model2.compile(loss='binary_crossentropy',optimizer=Adam(lr =
0.001),metrics=[auc,'accuracy'])
print(model2.summary())

Model: "model_1"
_____________________________________________________________________
_____________________________
 Layer (type)                Output Shape              Param #
Connected to
=====================================================================
=============================
 modified_essay_Inp (InputLayer  [(None, 300)]       0           []

 )



 embedded_mod_Essay (Embedding)  (None, 300, 300)    15524100
['modified_essay_Inp[0][0]']


 lstm_2 (LSTM)               (None, 300, 128)     219648
['embedded_mod_Essay[0][0]']


 school_state_inp (InputLayer)  [(None, 1)]          0           []



 teacher_Pref_inp (InputLayer)  [(None, 1)]          0           []



 pgCategory_Inp (InputLayer)    [(None, 1)]          0           []



 cleanCategory_Inp (InputLayer)  [(None, 3)]         0           []
```

```
 clean_subcategories_Inp (Input  [(None, 3)]         0           []

 Layer)



 lstm_3 (LSTM)                  (None, 300, 60)      45360
['lstm_2[0][0]']


 embedded_school_state (Embeddi  (None, 1, 6)        312
['school_state_inp[0][0]']
 ng)



 embedded_teacher_Pref (Embeddi  (None, 1, 2)        12
['teacher_Pref_inp[0][0]']
 ng)



 embedded_pgCategory (Embedding  (None, 1, 2)        10
['pgCategory_Inp[0][0]']
 )



 embedded_cleanCategory (Embedd  (None, 3, 3)        30
['cleanCategory_Inp[0][0]']
 ing)



 embedded_cleanSubCategory (Emb  (None, 3, 5)        155
['clean_subcategories_Inp[0][0]']
 edding)



 numericals_Inp (InputLayer)    [(None, 2)]          0           []



 tf.math.reduce_mean_1 (TFOpLam  (None, 300)         0
['lstm_3[0][0]']
 bda)
```

```
 flatten (Flatten)              (None, 6)           0
['embedded_school_state[0][0]']


 flatten_1 (Flatten)            (None, 2)           0
['embedded_teacher_Pref[0][0]']


 flatten_2 (Flatten)            (None, 2)           0
['embedded_pgCategory[0][0]']


 flatten_3 (Flatten)            (None, 9)           0
['embedded_cleanCategory[0][0]']


 flatten_4 (Flatten)            (None, 15)          0
['embedded_cleanSubCategory[0][0]
                                                                  ']



 dense (Dense)                  (None, 16)          48
['numericals_Inp[0][0]']


 concatenate_1 (Concatenate)    (None, 350)         0
['tf.math.reduce_mean_1[0][0]',

'flatten[0][0]',

'flatten_1[0][0]',

'flatten_2[0][0]',

'flatten_3[0][0]',

'flatten_4[0][0]',

'dense[0][0]']


 dense_5 (Dense)                (None, 128)         44928
['concatenate_1[0][0]']
```

```
 dropout_3 (Dropout)            (None, 128)          0
['dense_5[0][0]']


 dense_6 (Dense)                (None, 64)           8256
['dropout_3[0][0]']


 dropout_4 (Dropout)            (None, 64)           0
['dense_6[0][0]']


 batch_normalization_1 (BatchNo  (None, 64)          256
['dropout_4[0][0]']
 rmalization)



 dense_7 (Dense)                (None, 32)           2080
['batch_normalization_1[0][0]']


 dropout_5 (Dropout)            (None, 32)           0
['dense_7[0][0]']


 dense_8 (Dense)                (None, 1)            33
['dropout_5[0][0]']


==================================================================
============================
Total params: 15,845,228
Trainable params: 321,000
Non-trainable params: 15,524,228
__________________________________________________________________
_____________________________
None
```

```
plot_model(model2,to_file = 'model2.png',show_shapes =
True,show_layer_activations = True,show_dtype = True)
```




## Lets define few call backs

```
filepath="model_two_save/weights-{epoch:02d}-{val_auc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc',
verbose=1, save_best_only=True, mode='max')

earlystop = EarlyStopping(monitor='val_auc', min_delta=0.35,
patience=5, verbose=1)

reduce_lr = ReduceLROnPlateau(monitor='val_auc',
factor=0.1,patience=5, min_lr=0.0000001)

%load_ext tensorboard
log_dir = os.path.join("model_two","logs",'fits',
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback =
TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)
%reload_ext tensorboard

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

## Lets fit Model_1

```
model2_history =
model2.fit([X_train_modified_essay,X_train_schoolState,X_train_teacher
prefix,X_train_project_grade_category,X_train_clean_categories,X_train
_clean_subcategories,X_train_numericals], np.array(y_train),
           epochs=10,verbose=1,batch_size=256,
           validation_data =
```

```
([X_test_modified_essay,X_test_schoolState,X_test_teacherprefix,X_test
_project_grade_category,X_test_clean_categories,X_test_clean_subcatego
ries,X_test_numericals], np.array(y_test)),
          callbacks =[checkpoint,reduce_lr,tensorboard_callback])
#validation_data =
([X_test_modified_essay,X_test_schoolState,X_test_teacherprefix,X_test
_project_grade_category,X_test_clean_categories,X_test_clean_subcatego
ries,X_test_numericals], np.array(y_test)),

Epoch 1/10
341/342 [============================>.] - ETA: 0s - loss: 0.6264 -
auc: 0.5780 - accuracy: 0.8434
Epoch 1: val_auc improved from -inf to 0.67219, saving model to
model_two_save/weights-01-0.6722.hdf5
342/342 [==============================] - 31s 91ms/step - loss:
0.6261 - auc: 0.5781 - accuracy: 0.8435 - val_loss: 0.5030 - val_auc:
0.6722 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 2/10
341/342 [============================>.] - ETA: 0s - loss: 0.4598 -
auc: 0.6601 - accuracy: 0.8485
Epoch 2: val_auc improved from 0.67219 to 0.70320, saving model to
model_two_save/weights-02-0.7032.hdf5
342/342 [==============================] - 29s 86ms/step - loss:
0.4598 - auc: 0.6601 - accuracy: 0.8485 - val_loss: 0.4533 - val_auc:
0.7032 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 3/10
342/342 [==============================] - ETA: 0s - loss: 0.4158 -
auc: 0.6989 - accuracy: 0.8487
Epoch 3: val_auc improved from 0.70320 to 0.71930, saving model to
model_two_save/weights-03-0.7193.hdf5
342/342 [==============================] - 29s 85ms/step - loss:
0.4158 - auc: 0.6989 - accuracy: 0.8487 - val_loss: 0.4115 - val_auc:
0.7193 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 4/10
341/342 [============================>.] - ETA: 0s - loss: 0.3944 -
auc: 0.7248 - accuracy: 0.8495
Epoch 4: val_auc improved from 0.71930 to 0.73611, saving model to
model_two_save/weights-04-0.7361.hdf5
342/342 [==============================] - 29s 85ms/step - loss:
0.3945 - auc: 0.7246 - accuracy: 0.8495 - val_loss: 0.3969 - val_auc:
0.7361 - val_accuracy: 0.8485 - lr: 0.0010
Epoch 5/10
341/342 [============================>.] - ETA: 0s - loss: 0.3807 -
auc: 0.7452 - accuracy: 0.8509
Epoch 5: val_auc improved from 0.73611 to 0.73959, saving model to
model_two_save/weights-05-0.7396.hdf5
342/342 [==============================] - 29s 85ms/step - loss:
0.3807 - auc: 0.7453 - accuracy: 0.8509 - val_loss: 0.3884 - val_auc:
0.7396 - val_accuracy: 0.8490 - lr: 0.0010
Epoch 6/10
```

```
341/342 [============================>.] - ETA: 0s - loss: 0.3715 -
auc: 0.7603 - accuracy: 0.8538
Epoch 6: val_auc did not improve from 0.73959
342/342 [==============================] - 29s 85ms/step - loss:
0.3716 - auc: 0.7600 - accuracy: 0.8537 - val_loss: 0.3831 - val_auc:
0.7381 - val_accuracy: 0.8505 - lr: 0.0010
Epoch 7/10
341/342 [============================>.] - ETA: 0s - loss: 0.3494 -
auc: 0.7912 - accuracy: 0.8629
Epoch 7: val_auc improved from 0.73959 to 0.74004, saving model to
model_two_save/weights-07-0.7400.hdf5
342/342 [==============================] - 29s 85ms/step - loss:
0.3495 - auc: 0.7912 - accuracy: 0.8629 - val_loss: 0.3868 - val_auc:
0.7400 - val_accuracy: 0.8427 - lr: 1.0000e-04
Epoch 8/10
341/342 [============================>.] - ETA: 0s - loss: 0.3409 -
auc: 0.8027 - accuracy: 0.8672
Epoch 8: val_auc did not improve from 0.74004
342/342 [==============================] - 29s 85ms/step - loss:
0.3410 - auc: 0.8026 - accuracy: 0.8672 - val_loss: 0.3905 - val_auc:
0.7371 - val_accuracy: 0.8373 - lr: 1.0000e-04
Epoch 9/10
341/342 [============================>.] - ETA: 0s - loss: 0.3353 -
auc: 0.8075 - accuracy: 0.8714
Epoch 9: val_auc did not improve from 0.74004
342/342 [==============================] - 29s 85ms/step - loss:
0.3352 - auc: 0.8079 - accuracy: 0.8714 - val_loss: 0.3959 - val_auc:
0.7360 - val_accuracy: 0.8363 - lr: 1.0000e-04
Epoch 10/10
342/342 [==============================] - ETA: 0s - loss: 0.3306 -
auc: 0.8129 - accuracy: 0.8729
Epoch 10: val_auc did not improve from 0.74004
342/342 [==============================] - 29s 85ms/step - loss:
0.3306 - auc: 0.8129 - accuracy: 0.8729 - val_loss: 0.3965 - val_auc:
0.7341 - val_accuracy: 0.8361 - lr: 1.0000e-04
```

```
model2.load_weights('/content/model_two_save/weights-07-0.7400.hdf5')
```

```
model2_scores =
model2.evaluate([X_test_modified_essay,X_test_schoolState,X_test_teach
erprefix,X_test_project_grade_category,X_test_clean_categories,X_test_
clean_subcategories,X_test_numericals], np.array(y_test))
print("Model 2 loss : ",model2_scores[0])
print("Model 2 auc : ",model2_scores[1])
print("Model 2 accuracy : ",model2_scores[2])
result_dataset['AUC']['model2'] = model2_scores[1]
result_dataset['Loss']['model2'] = model2_scores[0]
result_dataset['accuracy']['model2'] = model2_scores[2]
result_dataset
```

```
683/683 [==============================] - 11s 16ms/step - loss:
0.3868 - auc: 0.7369 - accuracy: 0.8427
Model 2 loss :  0.38676247000694275
Model 2 auc :  0.7368632555007935
Model 2 accuracy :  0.8427460193634033

             AUC      Loss  accuracy
model1  0.753174  0.381855  0.846957
model2  0.736863  0.386762  0.842746
model3  0.000000  0.000000  0.000000

%tensorboard --logdir model_two/logs/fits

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

## Model-3

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

```
#in this model you can use the text vectorized data from model1
#for other than text data consider the following steps
# you have to perform one hot encoding of categorical features. You
can use onehotencoder() or countvectorizer() for the same.
# Stack up standardised numerical features and all the one hot encoded
categorical features
#the input to conv1d layer is 3d, you can convert your 2d data to 3d
using np.newaxis
# Note - deep learning models won't work with sparse features, you
have to convert them to dense features before fitting in the model.

X_train_model3 = X_train.drop(columns = ['essay','modified_essay'])
X_test_model3 = X_test.drop(columns = ['essay','modified_essay'])
X_train_model3.head()

  school_state teacher_prefix project_grade_category  \
0           ky            mrs             grades_3_5
1           la             ms          grades_prek_2
2           ny             ms             grades_6_8
3           ma            mrs             grades_3_5
4           fl             ms          grades_prek_2

   teacher_number_of_previously_posted_projects  \
0                                             0
1                                             4
2                                             8
3                                             1
4                                             2

                clean_categories              clean_subcategories
price
```

```
0  literacy_language math_science             literacy mathematics
149.99
1  literacy_language math_science   literature_writing mathematics
349.00
2       math_science specialneeds         mathematics specialneeds
486.26
3  literacy_language specialneeds  literature_writing specialneeds
489.80
4               literacy_language                     esl literacy
503.55
```

```
X_transform = ColumnTransformer([('onehotencodinng',
OneHotEncoder(handle_unknown = 'ignore'),
['school_state','teacher_prefix','project_grade_category','clean_categ
ories','clean_subcategories']),('remaining_scaling',MinMaxScaler(),
['teacher_number_of_previously_posted_projects','price'])],
remainder='drop')
X_transform.fit(X_train_model3)
```

```
ColumnTransformer(transformers=[('onehotencodinng',

OneHotEncoder(handle_unknown='ignore'),
                                 ['school_state', 'teacher_prefix',
                                  'project_grade_category',
'clean_categories',
                                  'clean_subcategories']),
                                ('remaining_scaling', MinMaxScaler(),

['teacher_number_of_previously_posted_projects',
                                  'price'])])
```

```
X_train_model3 = X_transform.transform(X_train_model3).todense()
X_test_model3 = X_transform.transform(X_test_model3).todense()
```

```
X_train_model3=np.expand_dims(X_train_model3, axis=2)
X_test_model3=np.expand_dims(X_test_model3, axis=2)
```

```
X_train_model3[0].shape
```

```
(510, 1)
```

```
print(" Shape of X train after transformation :
",X_train_model3.shape)
print(" Shape of X test after transformation : ",X_test_model3.shape)
```

```
 Shape of X train after transformation :  (87398, 510, 1)
 Shape of X test after transformation :  (21850, 510, 1)
```

```
dims = X_train_model3.shape[1]
```

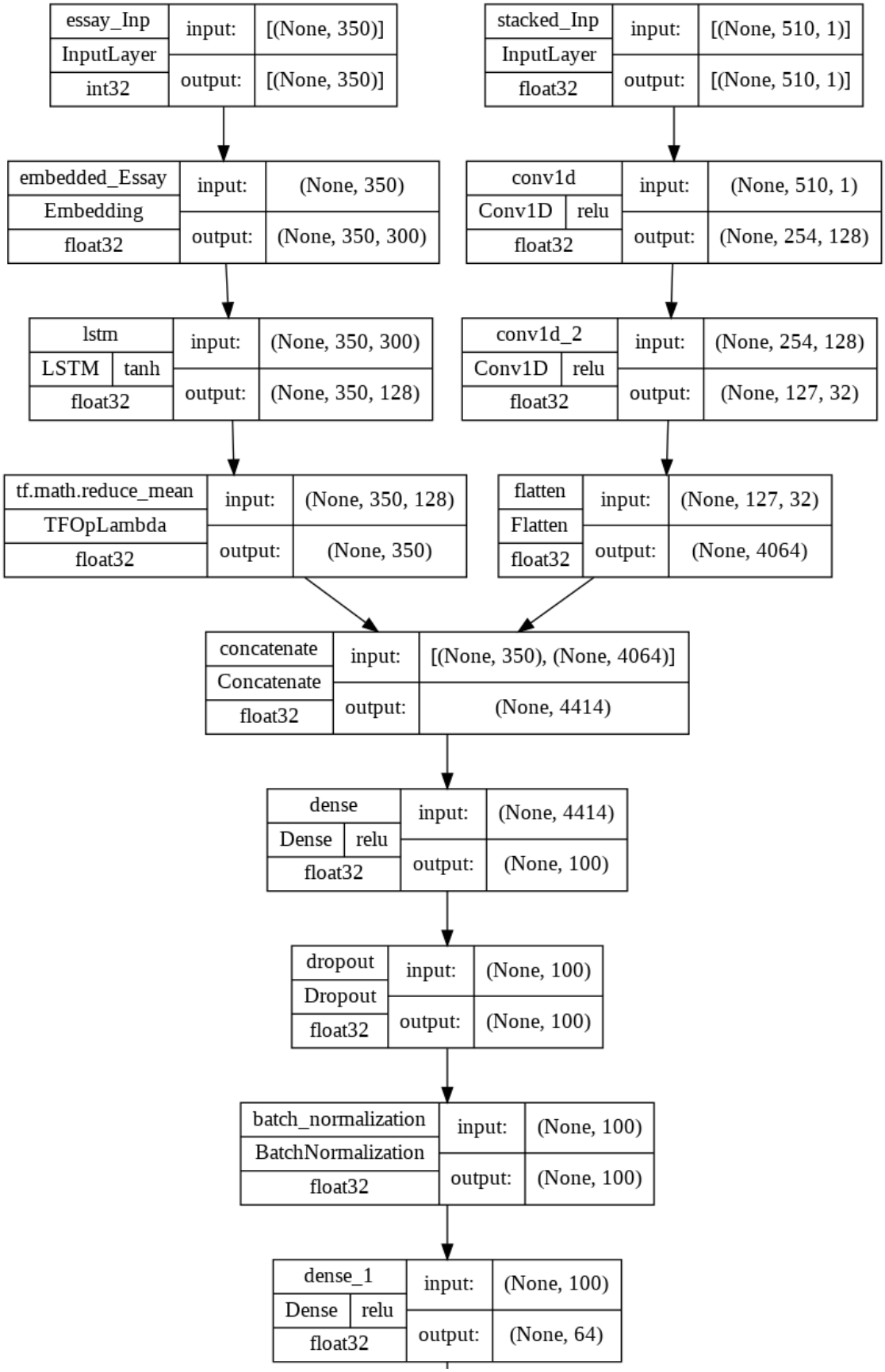
```
essay_Inp =
Input(shape=(max_essay_length,),dtype='int32',name='essay_Inp')
```

```
essay_ouput_dimensions = 300
embedded_Essay =
Embedding(input_dim=len(tokenizer_Essay.word_index.items()),output_dim
=essay_ouput_dimensions,name='embedded_Essay',weights=[embedding_matri
x],trainable=False)(essay_Inp)
essay_LSTM = LSTM(units=128,return_sequences = True)(embedded_Essay)
#essay_LSTM_2 = LSTM(units=60,return_sequences = True)(essay_LSTM)
essay_Out = tf.math.reduce_mean(essay_LSTM, axis = -1)
#essay_Out = Flatten()(average_out)

tf.keras.backend.clear_session()
stacked_Inp = Input(shape=(dims,1), name='stacked_Inp')
conv_layer1 = Conv1D(128, kernel_size=(3),strides =
2,activation='relu',kernel_initializer = 'he_normal')(stacked_Inp)
#128
conv_layer2 = Conv1D(64, kernel_size=(2),strides =
2,activation='relu',kernel_initializer = 'he_normal')(conv_layer1)  #
64
conv_layer3 = Conv1D(32, kernel_size=(2),strides =
2,activation='relu',kernel_initializer = 'he_normal')(conv_layer1)  #
32
conv_out = Flatten()(conv_layer3)

concatenated_Outs_model3=concatenate([essay_Out, conv_out])

# We are using the same configuration as the model1

dense_model3_1 = Dense(100,activation='relu',
               kernel_initializer=initializers.he_normal(),
               kernel_regularizer=regularizers.l2(0.001))
(concatenated_Outs_model3)
dropout_model3_1 = Dropout(0.5)(dense_model3_1)
batchnorm_model3_1 = BatchNormalization()(dropout_model3_1)
dense_model3_2 = Dense(64,activation='relu',
               kernel_initializer=initializers.he_normal(),
               kernel_regularizer=regularizers.l2(0.001))
(batchnorm_model3_1)
dropout_model3_2 = Dropout(0.5)(dense_model3_2)
batchnorm_model3_2 = BatchNormalization()(dropout_model3_2)
dense_model3_3 = Dense(32,activation='relu',
              kernel_initializer=initializers.he_normal(),
            kernel_regularizer=regularizers.l2(0.001))
(batchnorm_model3_2)
dropout_model3_3 = Dropout(0.5)(dense_model3_3)
model3_ouput = Dense(1, activation = 'sigmoid')(dropout_model3_3)

model3 = Model([essay_Inp, stacked_Inp], model3_ouput)
model3.compile(loss='binary_crossentropy',
               optimizer=Adam(lr = 0.001),
               metrics=[auc,'accuracy'])
print(model3.summary())
```

```
Model: "model"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #
Connected to
==========================================================================================
 essay_Inp (InputLayer)         [(None, 350)]        0           []


 stacked_Inp (InputLayer)       [(None, 510, 1)]     0           []


 embedded_Essay (Embedding)     (None, 350, 300)     15531600
['essay_Inp[0][0]']


 conv1d (Conv1D)                (None, 254, 128)     512
['stacked_Inp[0][0]']


 lstm (LSTM)                    (None, 350, 128)     219648
['embedded_Essay[0][0]']


 conv1d_2 (Conv1D)              (None, 127, 32)      8224
['conv1d[0][0]']


 tf.math.reduce_mean (TFOpLambd  (None, 350)         0
['lstm[0][0]']
 a)


 flatten (Flatten)              (None, 4064)         0
['conv1d_2[0][0]']


 concatenate (Concatenate)      (None, 4414)         0
['tf.math.reduce_mean[0][0]',

'flatten[0][0]']


 dense (Dense)                  (None, 100)          441500
```

```
['concatenate[0][0]']


 dropout (Dropout)              (None, 100)          0
['dense[0][0]']


 batch_normalization (BatchNorm  (None, 100)         400
['dropout[0][0]']
 alization)



 dense_1 (Dense)                (None, 64)           6464
['batch_normalization[0][0]']


 dropout_1 (Dropout)            (None, 64)           0
['dense_1[0][0]']


 batch_normalization_1 (BatchNo  (None, 64)          256
['dropout_1[0][0]']
 rmalization)



 dense_2 (Dense)                (None, 32)           2080
['batch_normalization_1[0][0]']


 dropout_2 (Dropout)            (None, 32)           0
['dense_2[0][0]']


 dense_3 (Dense)                (None, 1)            33
['dropout_2[0][0]']


==================================================================
============================
Total params: 16,210,717
Trainable params: 678,789
Non-trainable params: 15,531,928
__________________________________________________________________
_____________________________
None
```

```
plot_model(model3,to_file = 'model3.png',show_shapes =
True,show_layer_activations = True,show_dtype = True)
```


essay_Inp
InputLayer
int32
input:
[(None, 350)]
output:
[(None, 350)]
stacked_Inp
InputLayer
float32
input:
[(None, 510, 1)]
output:
[(None, 510, 1)]
embedded_Essay
Embedding
float32
input:
(None, 350)
output:
(None, 350, 300)
conv1d
Conv1D
relu
float32
input:
(None, 510, 1)
output:
(None, 254, 128)
lstm
LSTM
tanh
float32
input:
(None, 350, 300)
output:
(None, 350, 128)
conv1d_2
Conv1D
relu
float32
input:
(None, 254, 128)
output:
(None, 127, 32)
tf.math.reduce_mean
TFOpLambda
float32
input:
(None, 350, 128)
output:
(None, 350)
flatten
Flatten
float32
input:
(None, 127, 32)
output:
(None, 4064)
concatenate
Concatenate
float32
input:
[(None, 350), (None, 4064)]
output:
(None, 4414)
dense
Dense
relu
float32
input:
(None, 4414)
output:
(None, 100)
dropout
Dropout
float32
input:
(None, 100)
output:
(None, 100)
batch_normalization
BatchNormalization
float32
input:
(None, 100)
output:
(None, 100)
dense_1
Dense
relu
float32
input:
(None, 100)
output:
(None, 64)

### Lets define few call backs

```
filepath="model_three_save/weights-{epoch:02d}-{val_auc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc',
verbose=1, save_best_only=True, mode='max')

earlystop = EarlyStopping(monitor='val_auc', min_delta=0.35,
patience=5, verbose=1)

reduce_lr = ReduceLROnPlateau(monitor='val_auc',
factor=0.1,patience=5, min_lr=0.0000001)

%load_ext tensorboard
log_dir = os.path.join("model_three","logs",'fits',
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback =
TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)
%reload_ext tensorboard

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

### Lets fit Model_1

```
model3_history = model3.fit([X_train_essay,X_train_model3],
np.array(y_train),
           epochs=10,verbose=1,batch_size=256,
           validation_data = ([X_test_essay,X_test_model3],
np.array(y_test)),
            callbacks =[checkpoint,reduce_lr,tensorboard_callback])
#validation_data = ([X_test_essay,X_test_model3], np.array(y_test))

Epoch 1/10
341/342 [============================>.] - ETA: 0s - loss: 0.8334 -
auc: 0.5121 - accuracy: 0.7474
Epoch 1: val_auc improved from -inf to 0.54721, saving model to
model_three_save/weights-01-0.5472.hdf5
342/342 [==============================] - 32s 85ms/step - loss:
0.8331 - auc: 0.5119 - accuracy: 0.7475 - val_loss: 0.6154 - val_auc:
0.5472 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 2/10
341/342 [============================>.] - ETA: 0s - loss: 0.6003 -
auc: 0.5243 - accuracy: 0.8434
Epoch 2: val_auc improved from 0.54721 to 0.56307, saving model to
model_three_save/weights-02-0.5631.hdf5
342/342 [==============================] - 27s 80ms/step - loss:
0.6003 - auc: 0.5242 - accuracy: 0.8434 - val_loss: 0.5365 - val_auc:
0.5631 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 3/10
341/342 [============================>.] - ETA: 0s - loss: 0.5262 -
auc: 0.5293 - accuracy: 0.8480
Epoch 3: val_auc did not improve from 0.56307
342/342 [==============================] - 27s 79ms/step - loss:
```

```
0.5262 - auc: 0.5292 - accuracy: 0.8479 - val_loss: 0.4909 - val_auc:
0.5565 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 4/10
341/342 [============================>.] - ETA: 0s - loss: 0.4867 -
auc: 0.5288 - accuracy: 0.8485
Epoch 4: val_auc improved from 0.56307 to 0.57042, saving model to
model_three_save/weights-04-0.5704.hdf5
342/342 [==============================] - 27s 79ms/step - loss:
0.4866 - auc: 0.5288 - accuracy: 0.8485 - val_loss: 0.4617 - val_auc:
0.5704 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 5/10
341/342 [============================>.] - ETA: 0s - loss: 0.4632 -
auc: 0.5489 - accuracy: 0.8485
Epoch 5: val_auc improved from 0.57042 to 0.59445, saving model to
model_three_save/weights-05-0.5945.hdf5
342/342 [==============================] - 28s 82ms/step - loss:
0.4632 - auc: 0.5493 - accuracy: 0.8486 - val_loss: 0.4481 - val_auc:
0.5945 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 6/10
341/342 [============================>.] - ETA: 0s - loss: 0.4397 -
auc: 0.6208 - accuracy: 0.8486
Epoch 6: val_auc improved from 0.59445 to 0.68742, saving model to
model_three_save/weights-06-0.6874.hdf5
342/342 [==============================] - 28s 81ms/step - loss:
0.4397 - auc: 0.6207 - accuracy: 0.8486 - val_loss: 0.4811 - val_auc:
0.6874 - val_accuracy: 0.8486 - lr: 0.0010
Epoch 7/10
341/342 [============================>.] - ETA: 0s - loss: 0.4170 -
auc: 0.6852 - accuracy: 0.8487
Epoch 7: val_auc improved from 0.68742 to 0.70312, saving model to
model_three_save/weights-07-0.7031.hdf5
342/342 [==============================] - 27s 80ms/step - loss:
0.4170 - auc: 0.6852 - accuracy: 0.8486 - val_loss: 0.4087 - val_auc:
0.7031 - val_accuracy: 0.8486 - lr: 1.0000e-04
Epoch 8/10
341/342 [============================>.] - ETA: 0s - loss: 0.4101 -
auc: 0.6958 - accuracy: 0.8483
Epoch 8: val_auc improved from 0.70312 to 0.71018, saving model to
model_three_save/weights-08-0.7102.hdf5
342/342 [==============================] - 28s 81ms/step - loss:
0.4101 - auc: 0.6953 - accuracy: 0.8484 - val_loss: 0.4046 - val_auc:
0.7102 - val_accuracy: 0.8486 - lr: 1.0000e-04
Epoch 9/10
341/342 [============================>.] - ETA: 0s - loss: 0.4057 -
auc: 0.7023 - accuracy: 0.8488
Epoch 9: val_auc improved from 0.71018 to 0.71298, saving model to
model_three_save/weights-09-0.7130.hdf5
342/342 [==============================] - 27s 80ms/step - loss:
0.4056 - auc: 0.7023 - accuracy: 0.8488 - val_loss: 0.4039 - val_auc:
0.7130 - val_accuracy: 0.8492 - lr: 1.0000e-04
```

```
Epoch 10/10
341/342 [============================>.] - ETA: 0s - loss: 0.4020 -
auc: 0.7091 - accuracy: 0.8484
Epoch 10: val_auc improved from 0.71298 to 0.71695, saving model to
model_three_save/weights-10-0.7169.hdf5
342/342 [==============================] - 28s 81ms/step - loss:
0.4019 - auc: 0.7096 - accuracy: 0.8484 - val_loss: 0.3983 - val_auc:
0.7169 - val_accuracy: 0.8489 - lr: 1.0000e-04
```

```
model3.load_weights('/content/model_three_save/weights-10-
0.7169.hdf5')
```

```
model3_scores = model3.evaluate([X_test_essay,X_test_model3],
np.array(y_test))
print("Model 3 loss : ",model3_scores[0])
print("Model 3 auc : ",model3_scores[1])
print("Model 3 accuracy : ",model3_scores[2])
result_dataset['AUC']['model3'] = model3_scores[1]
result_dataset['Loss']['model3'] = model3_scores[0]
result_dataset['accuracy']['model3'] = model3_scores[2]
result_dataset
```

```
683/683 [==============================] - 9s 14ms/step - loss: 0.3983
- auc: 0.7152 - accuracy: 0.8489
Model 3 loss :  0.39834314584732056
Model 3 auc :  0.7152100205421448
Model 3 accuracy :  0.848924458026886
```

```
             AUC      Loss  accuracy
model1  0.753174  0.381855  0.846957
model2  0.736863  0.386762  0.842746
model3  0.715210  0.398343  0.848924
```

```
%tensorboard --logdir model_three/logs/fits
```

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

```
from tabulate import tabulate
print(tabulate(result_dataset, headers = 'keys', tablefmt = 'psql'))
```

```
+--------+----------+----------+------------+
|        |      AUC |     Loss |   accuracy |
|--------+----------+----------+------------|
| model1 | 0.753174 | 0.381855 |   0.846957 |
| model2 | 0.736863 | 0.386762 |   0.842746 |
| model3 | 0.71521  | 0.398343 |   0.848924 |
+--------+----------+----------+------------+
```