Assignment : 14

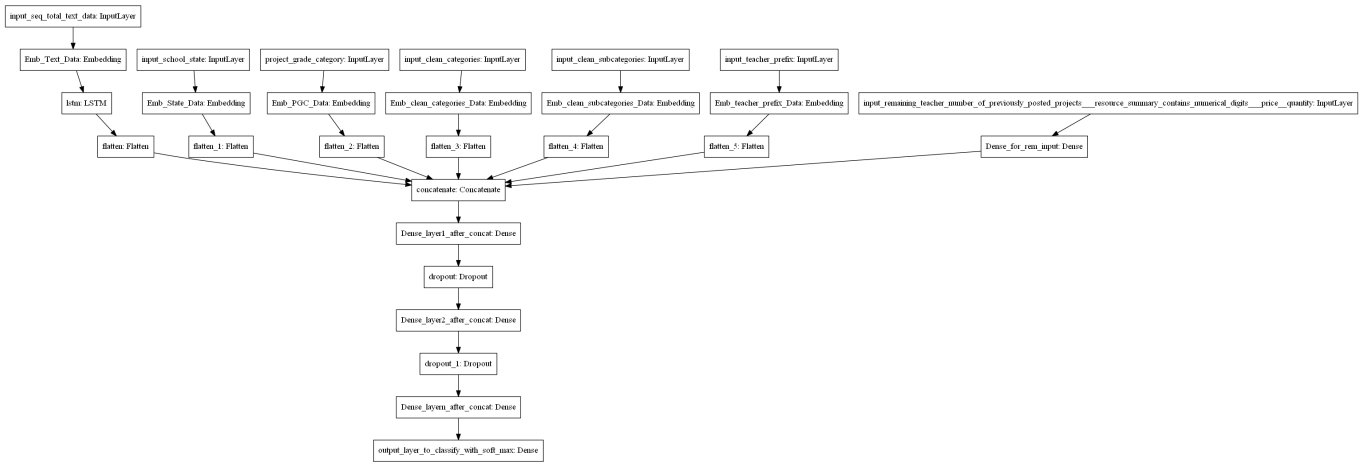
1. You can work with preprocessed_data.csv for the assignment. You can get the data from - Data folder
2. Load the data in your notebook.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check this and this for using auc as a metric
5. 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.
7. For all the model's use TensorBoard and plot the Metric value and Loss with epoch. While submitting, take a screenshot of plots
8. Make sure that you are using GPU to train the given models.

```
#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


#!gdown --id 1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
#!gdown --id 1pGd5tLwA30M7wkhJKdXHaae9tYVDICJ_
```

Saved successfully!

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

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
# import all the libraries
#make sure that you import your libraries from tf.keras and not just keras
import os
from tqdm import tqdm
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer,one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Saved successfully!

```
mport l2
Input,Dense,Conv1D,concatenate,Embedding,Flatten,Dropout,BatchNormalization,MaxPool1D,LSTM
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import EarlyStopping
import datetime
```

```
#read the csv file
import pandas as pd
df = pd.read_csv('drive/MyDrive/LSTM_donors_choose_26/preprocessed_data.csv')
df.head()
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_c
0	ca	mrs	grades_prek_2	53	1	ma
1	ut	ms	grades_3_5	4	1	sp
2	ca	mrs	grades_prek_2	10	1	literacy
3	ga	mrs	grades_prek_2	2	1	appl
4	wa	mrs	grades_3_5	2	1	literacy

```
print(df.columns)
print(df.shape)
```

```
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')
(109248, 9)
```

```
Y=df["project_is_approved"].values
df.drop("project_is_approved",axis = 1, inplace = True)
X=df
```

```
print(X.columns)
print(X.shape)
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price'],
      dtype='object')
(109248, 8)
```

```
# perform stratified train test split on the dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, stratify=Y)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

Saved successfully!

```
unique = set(X_train['essay'].values)
print("No of unique words in train essay",len(unique))
```

```
No of unique words in train essay 76022
```

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
```

```
#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/
```

Tokenizing the text

```
#Converts a text to a sequence of words (or tokens).
#A list of words (or tokens).
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train["essay"])
X_train['essay_tkn'] = tokenizer.texts_to_sequences(X_train["essay"])
X_test['essay_tkn'] = tokenizer.texts_to_sequences(X_test["essay"])
```

```
X_train.head()
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean
105919	fl	mrs	grades_3_5	48	math_science	envi
81820	nc	mrs	grades_3_5	8	math_science	
31306	ma	mrs	grades_6_8	0	specialneeds	
6489	ga	mrs	grades_3_5	0	math_science	

```
word_idx = tokenizer.word_index
                                     index)
                                     n(word_idx))
print(max_vocabulary)
```

Saved successfully!

```
Found 49027 unique tokens.
49027
```

```
max_review_length = 800
X_train_pad = pad_sequences(X_train['essay_tkn'].values, maxlen=max_review_length , padding='post' )
X_test_pad  = pad_sequences(X_test['essay_tkn'].values, maxlen=max_review_length , padding='post')
print(X_test_pad[0])
```

```
 256    1  227  115 2705  247   26   53 1634 1996  316   36
   5  496  168   18  121   52  147 1575    8    1   10 1996
   6   43    1  731  332   58 3969  320   59 1634 1996  316
  36    5   11  702  168   36    5  353  204  159   27  273
 121  469   36 1911  772  203   26  203  101   14   15  245
1802  406  151    1  168   78  372  283  157    1  548   87
 414 1376   25    1   15   10   47  430  414 4213 1992  548
   9   15   20  615 1064  102   27    9   28  475   41  354
1857 2156   18  151    2  103   50 1137  161    1  108   33
 386    5   13    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0
```

```
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0     0     0     0     0
          0     0     0     0     0     0     0     0]
```

```
with open('drive/MyDrive/LSTM_donors_choose_26/glove_vectors','rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
# create a weight matrix for words in training docs
from numpy import zeros
max_vocabulary = len(tokenizer.word_index)
embedding_matrix = zeros((max_vocabulary+1, 300))
for word, i in tokenizer.word_index.items():
  if word in glove_words:
    embedding_vector = model.get(word)
    embedding_matrix[i] = embedding_vector

print(max_vocabulary)
print(embedding_matrix.shape)
```

Saved successfully!

```
    (49028, 300)
```

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.preprocessing.OrdinalEncoder.html
# after categorical feature vectorization you will have column_train_data and column_test_data.
```

```
from sklearn.preprocessing import LabelEncoder

label_enc = LabelEncoder()
label_enc.fit(X['school_state'].values)

X_train_school_state = label_enc.transform(X_train['school_state'].values)

X_test_school_state = label_enc.transform(X_test['school_state'].values)
```

```
no_of_unique_states = X_train['school_state'].nunique()
print("Number of unique school state= ",no_of_unique_states)

embedding_size_school_state = min(np.ceil((no_of_unique_states)/2), 50 )
embedding_size_school_state = int(embedding_size_school_state)
print('Embedding size = ',embedding_size_school_state)
```

```
    Number of unique school state=  51
    Embedding size =  26
```

```
label_enc.fit(X['teacher_prefix'].values)

X_train_teacher_prefix = label_enc.transform(X_train['teacher_prefix'].values)

X_test_teacher_prefix = label_enc.transform(X_test['teacher_prefix'].values)
```

```
no_of_unique_teacher_prefix = X_train['teacher_prefix'].nunique()
print("Number of unique teacher_prefix= ",no_of_unique_teacher_prefix)

embedding_size_teacher_prefix = min(np.ceil((no_of_unique_teacher_prefix)/2), 50 )
embedding_size_teacher_prefix = int(embedding_size_teacher_prefix)
print('Embedding size = ',embedding_size_teacher_prefix)
```

```
    Number of unique teacher_prefix=  5
    Embedding size =  3
```

```
label_enc.fit(X['clean_categories'].values)

X_train_clean_categories = label_enc.transform(X_train['clean_categories'].values)

X_test_clean_categories = label_enc.transform(X_test['clean_categories'].values)


no_of_unique_clean_categories = X_train['clean_categories'].nunique()
print("Number of unique clean_categories= ",no_of_unique_clean_categories)

embedding_size_clean_categories = min(np.ceil((no_of_unique_clean_categories)/2), 50 )
embedding_size_clean_categories = int(embedding_size_clean_categories)
print('Embedding size = ',embedding_size_clean_categories)
```

```
Number of unique clean_categories=  51
Embedding size =  26
```

```
label_enc.fit(X['clean_subcategories'].values)

X_train_clean_subcategories = label_enc.transform(X_train['clean_subcategories'].values)

X_test_clean_subcategories = label_enc.transform(X_test['clean_subcategories'].values)
```



```
_train['clean_subcategories'].nunique()
egories= ",no_of_unique_clean_subcategories)

embedding_size_clean_subcategories = min(np.ceil((no_of_unique_clean_subcategories)/2), 50 )
embedding_size_clean_subcategories = int(embedding_size_clean_subcategories)
print('Embedding size = ',embedding_size_clean_subcategories)
```

```
Number of unique clean_subcategories=  394
Embedding size =  50
```

```
label_enc.fit(X['project_grade_category'].values)

X_train_project_grade_category = label_enc.transform(X_train['project_grade_category'].values)

X_test_project_grade_category = label_enc.transform(X_test['project_grade_category'].values)


no_of_unique_project_grade_category = X_train['project_grade_category'].nunique()
print("Number of unique project_grade_category= ",no_of_unique_project_grade_category)

embedding_size_project_grade_category = min(np.ceil((no_of_unique_project_grade_category)/2), 50 )
embedding_size_project_grade_category = int(embedding_size_project_grade_category)
print('Embedding size = ',embedding_size_project_grade_category)
```

```
Number of unique project_grade_category=  4
Embedding size =  2
```

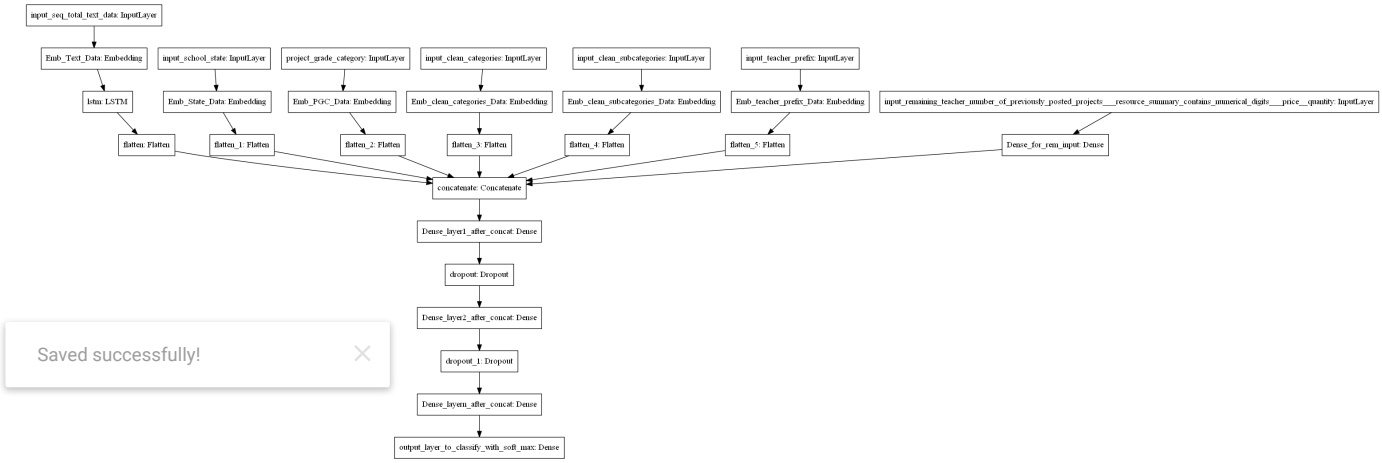
1.3 Numerical feature Vectorization

```
# 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


from sklearn.preprocessing import StandardScaler
stnd_scaler=StandardScaler()
stnd_scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_num_projects=stnd_scaler.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_test_num_projects=stnd_scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))



stnd_scaler.fit(X_train['price'].values.reshape(-1,1))

X_train_price = stnd_scaler.transform(X_train['price'].values.reshape(-1,1))

X_test_price = stnd_scaler.transform(X_test['price'].values.reshape(-1,1))
```

```
X_train_numeric_features = np.concatenate((X_train_num_projects , X_train_price) , axis = 1)

X_test_numeric_features= np.concatenate((X_test_num_projects , X_test_price) , axis = 1)

print(X_train_numeric_features.shape ,X_test_numeric_features.shape)
```

```
(76473, 2) (32775, 2)
```

1.4 Defining the model



Saved successfully!

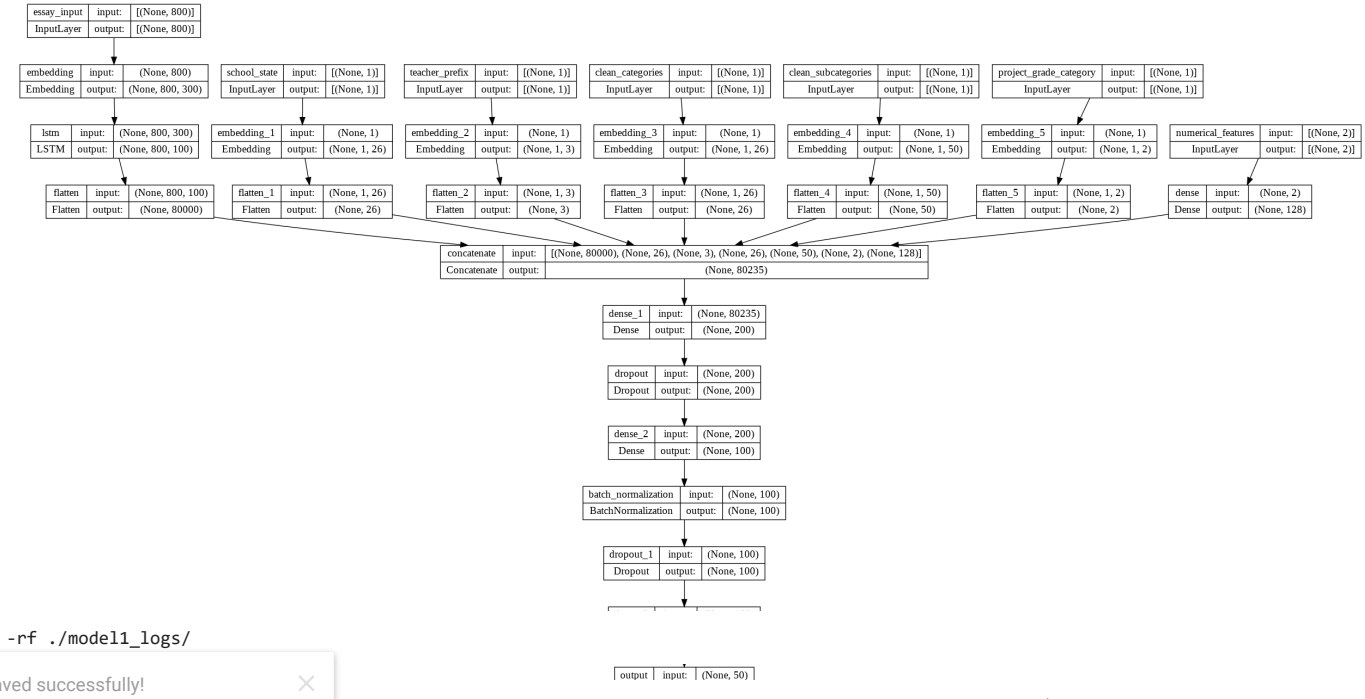
```
# as of now we have vectorized all our features now we will define our model.
# as it is clear from above image that the given model has multiple input layers and hence we have to use functional API
# Please go through - https://keras.io/guides/functional_api/
# it is a good programming practise 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 = Input(shape=(800,), name='essay_input')

X = Embedding(output_dim=300, input_dim=max_vocabulary+1, input_length=800 , weights=[embedding_matrix])(essay)
lstm_essay = LSTM(100,recurrent_dropout=0.5,kernel_regularizer=l2(0.001),return_sequences=True)(X)
flatten_1 = Flatten()(lstm_essay)

school_state = Input(shape=(1,), name='school_state')
X_school_state = Embedding(output_dim=embedding_size_school_state , input_dim=no_of_unique_states, input_length=1)(school_state)
flatten_2 = Flatten()(X_school_state)

teacher_prefix = Input(shape=(1,), name='teacher_prefix')
X_teacher_prefix = Embedding(output_dim=embedding_size_teacher_prefix , input_dim=no_of_unique_teacher_prefix, input_length=1)(teacher_pr
flatten_3 = Flatten()(X_teacher_prefix)

clean_categories = Input(shape=(1,), name='clean_categories')
X_clean_categories = Embedding(output_dim=embedding_size_clean_categories, input_dim=no_of_unique_clean_categories, input_length=1)(clear
flatten_4 = Flatten()(X_clean_categories)

clean_subcategories = Input(shape=(1,), name='clean_subcategories')
X_clean_subcategories = Embedding(output_dim=embedding_size_clean_subcategories, input_dim=no_of_unique_clean_subcategories, input_length
flatten_5 = Flatten()(X_clean_subcategories)

project_grade_category = Input(shape=(1,), name='project_grade_category')
X_project_grade_category = Embedding(output_dim=embedding_size_project_grade_category, input_dim=no_of_unique_project_grade_category, inp
flatten_6 = Flatten()(X_project_grade_category)

numeric_features = Input(shape=(2,) , name="numerical_features")
numeric_dense = Dense(128, activation='relu' , kernel_initializer='he_normal',kernel_regularizer=l2(0.001))(numeric_features )


X_concat = concatenate([flatten_1 , flatten_2 , flatten_3 ,flatten_4 , flatten_5 , flatten_6 , numeric_dense])
model = Dense(200, activation="relu", kernel_initializer="he_normal" ,kernel_regularizer=l2(0.001))(X_concat)

model = Dropout(0.5)(model)
```

```
model = Dense(100,activation="relu",kernel_initializer="glorot_normal" ,kernel_regularizer=l2(0.001))(model)

model = BatchNormalization()(model)

model = Dropout(0.5)(model)

model = Dense(50,activation="relu", kernel_initializer="glorot_normal" ,kernel_regularizer=l2(0.001))(model)


output = Dense(2, activation='softmax', name='output')(model)

model_1 = Model(inputs=[essay, school_state ,teacher_prefix,clean_categories,
                        clean_subcategories ,project_grade_category ,numeric_features ],outputs=[output])


print(model_1.summary())
```

```
lstm (LSTM)                    (None, 800, 100)     160400      ['embedding[0][0]']

embedding_1 (Embedding)        (None, 1, 26)        1326        ['school_state[0][0]']

embedding_2 (Embedding)        (None, 1, 3)         15          ['teacher_prefix[0][0]']

embedding_3 (Embedding)        (None, 1, 26)        1326        ['clean_categories[0][0]']

embedding_4 (Embedding)        (None, 1, 50)        19700       ['clean_subcategories[0][0]']

                               (None, 1, 2)         8           ['project_grade_category[0][0]']

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

flatten (Flatten)              (None, 80000)        0           ['lstm[0][0]']

flatten_1 (Flatten)            (None, 26)           0           ['embedding_1[0][0]']

flatten_2 (Flatten)            (None, 3)            0           ['embedding_2[0][0]']

flatten_3 (Flatten)            (None, 26)           0           ['embedding_3[0][0]']

flatten_4 (Flatten)            (None, 50)           0           ['embedding_4[0][0]']

flatten_5 (Flatten)            (None, 2)            0           ['embedding_5[0][0]']

dense (Dense)                  (None, 128)          384         ['numerical_features[0][0]']

concatenate (Concatenate)      (None, 80235)        0           ['flatten[0][0]',
                                                                 'flatten_1[0][0]',
                                                                 'flatten_2[0][0]',
                                                                 'flatten_3[0][0]',
                                                                 'flatten_4[0][0]',
                                                                 'flatten_5[0][0]',
                                                                 'dense[0][0]']

dense_1 (Dense)                (None, 200)          16047200    ['concatenate[0][0]']

dropout (Dropout)              (None, 200)          0           ['dense_1[0][0]']

dense_2 (Dense)                (None, 100)          20100       ['dropout[0][0]']

batch_normalization (BatchNorm  (None, 100)         400         ['dense_2[0][0]']
alization)

dropout_1 (Dropout)            (None, 100)          0           ['batch_normalization[0][0]']

dense_3 (Dense)                (None, 50)           5050        ['dropout_1[0][0]']

output (Dense)                 (None, 2)            102         ['dense_3[0][0]']

==================================================================================================
Total params: 30,964,411
Trainable params: 30,964,211
Non-trainable params: 200
__________________________________________________________________________________________________
None
```

Saved successfully!

```
plot_model(model_1,'image_1.png',show_shapes=True)
```




```
!rm -rf ./model1_logs/
```

Saved successfully!

```
_1.h5",monitor="val_auroc",mode="max",save_best_only = True,verbose=1)
earlystop1 = EarlyStopping(monitor = 'val_auroc', mode="max",min_delta = 0,patience = 3,verbose = 1)

tensorboard1 = TensorBoard(log_dir='Model1_visualization')

callbacks_1 = [checkpoint1,earlystop1,tensorboard1]


final_X_train_data = [X_train_pad,X_train_school_state,X_train_teacher_prefix,X_train_clean_categories,X_train_clean_subcategories,X_trai

final_X_test_data = [X_test_pad,X_test_school_state,X_test_teacher_prefix,X_test_clean_categories,X_test_clean_subcategories,X_test_proje



from keras.utils import np_utils

y_train = np_utils.to_categorical(Y_train, 2)
y_test = np_utils.to_categorical(Y_test, 2)
```

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


from sklearn.metrics import roc_auc_score
def auc1(y_true, y_pred):
    if len(np.unique(y_true[:,1])) == 1:
        return 0.5
    else:
        return roc_auc_score(y_true, y_pred)

def auroc(y_true, y_pred):
    return tf.compat.v1.py_func(auc1, (y_true, y_pred), tf.double)


model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auroc])



history = model_1.fit(final_X_train_data, y_train, batch_size=512, epochs=10, verbose=1,callbacks=callbacks_1, validation_data=(final_X_t


 Epoch 1/10
 WARNING:tensorflow:From /usr/local/lib/python3.8/dist-packages/tensorflow/python/autograph/impl/api.py:459: py_func (from tensorflo
 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
```

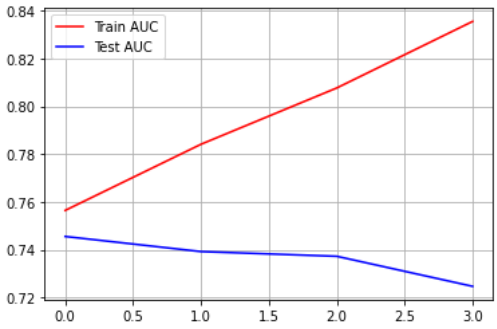
```
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

150/150 [==============================] - ETA: 0s - loss: 0.6355 - auroc: 0.7565
Epoch 1: val_auroc improved from -inf to 0.74557, saving model to model_1.h5
150/150 [==============================] - 456s 3s/step - loss: 0.6355 - auroc: 0.7565 - val_loss: 0.6641 - val_auroc: 0.7456
Epoch 2/10
150/150 [==============================] - ETA: 0s - loss: 0.5521 - auroc: 0.7842
Epoch 2: val_auroc did not improve from 0.74557
150/150 [==============================] - 456s 3s/step - loss: 0.5521 - auroc: 0.7842 - val_loss: 0.6347 - val_auroc: 0.7392
Epoch 3/10
150/150 [==============================] - ETA: 0s - loss: 0.4979 - auroc: 0.8077
Epoch 3: val_auroc did not improve from 0.74557
150/150 [==============================] - 456s 3s/step - loss: 0.4979 - auroc: 0.8077 - val_loss: 0.5388 - val_auroc: 0.7373
Epoch 4/10
150/150 [==============================] - ETA: 0s - loss: 0.4472 - auroc: 0.8356
Epoch 4: val_auroc did not improve from 0.74557
150/150 [==============================] - 460s 3s/step - loss: 0.4472 - auroc: 0.8356 - val_loss: 0.5273 - val_auroc: 0.7247
Epoch 4: early stopping
```

```
history_1 = model_1.fit(final_X_train_data, y_train, batch_size=900, epochs=10, verbose=1,callbacks=callbacks_1, validation_data=(final_X
```

```
Epoch 1/10
                                                  ======] - ETA: 0s - loss: 0.3053 - auroc: 0.9153
                                                  ove from 0.74557
                                                  ======] - 296s 3s/step - loss: 0.3053 - auroc: 0.9153 - val_loss: 0.5983 - val_auroc: 0.6878
Epoch 2/10
85/85 [==============================] - ETA: 0s - loss: 0.2718 - auroc: 0.9330
Epoch 2: val_auroc did not improve from 0.74557
85/85 [==============================] - 296s 3s/step - loss: 0.2718 - auroc: 0.9330 - val_loss: 0.5189 - val_auroc: 0.6660
Epoch 3/10
85/85 [==============================] - ETA: 0s - loss: 0.2479 - auroc: 0.9460
Epoch 3: val_auroc did not improve from 0.74557
85/85 [==============================] - 295s 3s/step - loss: 0.2479 - auroc: 0.9460 - val_loss: 0.5877 - val_auroc: 0.6630
Epoch 4/10
85/85 [==============================] - ETA: 0s - loss: 0.2296 - auroc: 0.9560
Epoch 4: val_auroc did not improve from 0.74557
85/85 [==============================] - 295s 3s/step - loss: 0.2296 - auroc: 0.9560 - val_loss: 0.6161 - val_auroc: 0.6419
Epoch 4: early stopping
```

Saved successfully!

```
from matplotlib import pyplot as plt
plt.plot(history.history['auroc'], 'r')
plt.plot(history.history['val_auroc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'g'})
plt.grid()
plt.show()
```




```
from matplotlib import pyplot as plt
plt.plot(history_1.history['auroc'], 'r')
plt.plot(history_1.history['val_auroc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'g'})
plt.grid()
plt.show()
```

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.

1. Fit TF-IDF vectorizer on the Train data

2. Get the idf value for each word we have in the train data. Please go through this

3. Do some analysis on the Idf values and based on those values choose the low and high threshold value. Because very frequent words and very very rare words don't give much information.
Hint - A preferable IDF range is 2-11 for model 2.

4.Remove the low idf value and high idf value words from the train and test data. You can go through each of the sentence of train and test data and include only those features(words) which are present in the defined IDF range.

Saved successfully!

fied text data same as you have done for previous model.

l 2 and then use the rest of the features similar to previous model.

7. Define the model, compile and fit the model.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(analyzer = 'word')
tfidf.fit(X_train['essay'].values)
```

```
TfidfVectorizer()
```

```
tfidf_vectorizer = TfidfVectorizer()
tfidf = tfidf_vectorizer.fit_transform(X_train["essay"])
```

```
data = {'word': tfidf_vectorizer.get_feature_names() , 'idf_value': tfidf_vectorizer.idf_}
tfidf_df = pd.DataFrame(data=data)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; ge
  warnings.warn(msg, category=FutureWarning)
```

```
print("Min tf-idf value is: ",min(tfidf_df['idf_value']))
print("Max tf-idf value is: ",max(tfidf_df['idf_value']))
```

```
Min tf-idf value is:  1.0074419070198535
Max tf-idf value is:  11.551558912204982
```

Remove low idf_ words from essays

```
filter =   (tfidf_df['idf_value']>=2) & (tfidf_df['idf_value'] <=11.21)
tfidf_best = tfidf_df[filter]
Best_tfidf = tfidf_best['word'].tolist()

print(tfidf_best.shape)
```

```
(29874, 2)
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(Best_tfidf)

X_train['essay_tok_2']  = tokenizer.texts_to_sequences(X_train['essay'].values)

X_test['essay_tok_2']   = tokenizer.texts_to_sequences(X_test['essay'].values)

max_vocabulary1 = len(tokenizer.word_index)
print("There are {} number of unique words in the entire text corpus".format(max_vocabulary1))

print(X_train['essay_tok_2'].values[0])
print(len(X_train['essay_tok_2'].values[0]))
```

```
There are 29874 number of unique words in the entire text corpus
[1619, 558, 11815, 16446, 23302, 12321, 29535, 25606, 21459, 11679, 2403, 8623, 9647, 1490, 21578, 11021, 15981, 3787, 16146, 25645
89
```

```
max_review_length = 800

X_train_pad1 = pad_sequences(X_train['essay_tok_2'].values, maxlen=max_review_length , padding='post' )
X_test_pad1  = pad_sequences(X_test['essay_tok_2'].values, maxlen=max_review_length , padding='post' )
```

```
X_train_pad1[0]
```

```
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0,     0,
        0,     0,     0,     0,     0,     0,     0,     0],
      dtype=int32)
```

Saved successfully!

```
with open('drive/MyDrive/LSTM_donors_choose_26/glove_vectors','rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
# create a weight matrix for words in training docs
from numpy import zeros
max_vocabulary_1 = len(tokenizer.word_index)
embedding_matrix_1 = zeros((max_vocabulary_1+1, 300))
for word, i in tokenizer.word_index.items():
  if word in glove_words:
    embedding_vector_1 = model.get(word)
    embedding_matrix_1[i] = embedding_vector_1
```

```
print(max_vocabulary_1)
print(embedding_matrix_1.shape)
```

```
29874
(29875, 300)
```

```
print(embedding_matrix_1[1].shape)
print(embedding_matrix_1[1])
```

```
(300,)
[ 0.38311    0.58955   -0.42684   -0.11505    0.23785    0.29834
 -0.71938    0.15237    0.085018   0.041687  -0.34072    0.97654
  0.24777    0.09065   -0.43767   -0.08772    0.0031667 -0.071644
  0.55351   -0.11632   -0.028157   0.32075    0.26208    0.37667
  0.26503   -0.11469    0.032618  -0.18537    0.10012    0.49492
  0.20423   -0.43822    0.48834   -0.43821    0.41716   -0.10504
  0.044853  -0.38108   -0.087677   0.94326   -0.28966   -0.0069608
 -0.33716   -0.18373    0.060478   0.27742   -0.035231   0.30744
 -0.66424    0.089716   0.14229    0.84869   -0.12057   -0.14589
 -0.22944    0.34677    0.062355  -0.48993    0.47824   -0.022702
  0.42403    0.31398    0.043538   0.35516    0.57668    0.47269
  0.14307    0.23131   -0.12372    0.25442   -0.23657   -0.41193
  0.10668   -0.66548   -0.70757    0.1851    -0.1038    -0.39208
  0.18296   -0.30977   -0.018167   0.32197    0.17513   -0.42211
 -0.13277    0.29645    0.021128  -0.34988    0.71415   -0.3805
 -0.22953   -0.27483   -0.015603  -0.11907    0.10681    0.90405
                                   0.22279   -0.17113   -0.29833
                                9  -0.45042    0.47973    0.88689
                                   -0.26627   -0.73931   -0.46068
  0.36344   -0.28019    0.41742    0.58817    0.10852   -0.0079874
 -0.11434    0.15054    0.63973    0.41721    0.24404   -0.68039
 -0.4055    -0.52723    0.33952   -0.11677   -0.089904   0.81436
  0.26772    0.073576  -0.52452   -0.50729   -0.54258   -0.14842
 -0.22435   -0.13641   -0.12253    0.43157   -0.71492   -0.31669
 -0.30325   -0.49072   -0.01959   -0.11519   -0.57455   -0.086298
  0.12211   -0.08537    0.2351     0.054954   0.17056    0.36587
  0.29804   -0.68196    0.63021   -0.25957   -0.44552    0.37638
  0.31014   -0.41451   -0.033984  -0.46487    0.31377   -0.20226
  0.48858    0.01402   -0.87869    0.27225   -0.13412   -0.35145
  0.07657    0.22391   -0.01095   -0.15935    0.18226    0.055708
 -0.038036   0.48037   -0.14483    0.32668    0.061728   0.22374
 -0.1482    -0.269     -0.65264    0.1343    -0.55079   -0.32581
 -0.46325    0.21897    0.29431   -0.98949    0.26147    0.68728
 -0.056174  -0.19671   -0.36356    0.25835   -0.36482    0.11671
  0.41547   -0.45227    0.22565    0.47386    0.27675    0.12709
  0.030362  -0.74851    0.33315   -0.16154    0.40523   -0.35153
  0.24219   -0.010269  -0.27914    0.4083     0.44669   -0.23572
 -0.2744     0.13094   -2.9057     0.41858    0.604      0.19251
 -0.1756     0.14679    0.2244    -0.1519     0.5022    -0.64604
  0.66071   -0.15955    0.49364    0.36689   -0.21363   -0.45818
 -0.18868    0.16895    0.59806   -0.72444    0.3018    -0.6566
  0.061542  -0.27434   -0.77936   -0.53357    0.65501    0.3633
 -0.17993    0.19085    0.23041    0.4376     0.39892    0.50818
  0.43578    0.23757   -0.5759    -0.11419    0.38459   -0.99393
  0.33491    0.20122    0.60021   -0.12203    0.30979   -0.29233
 -0.49053   -0.36853    0.074609   0.02444   -0.040684   0.065733
  0.56815   -0.4727    -0.22972   -0.39545   -0.2783     0.51589
  0.41044    0.044208   0.058739  -0.54321    0.27625   -0.24973
  0.6827    -0.67666   -0.34129   -0.38119   -0.12608    0.38659
 -0.36249    0.28039    0.087084   0.34844    0.28596   -0.17433  ]
```

Saved successfully!

```
in_pad1,X_train_school_state,X_train_teacher_prefix,X_train_clean_categories,X_train_clean_subcategories,X_train_project_grade_category,X
_pad1,X_test_school_state,X_test_teacher_prefix,X_test_clean_categories,X_test_clean_subcategories,X_test_project_grade_category,X_test_n
```

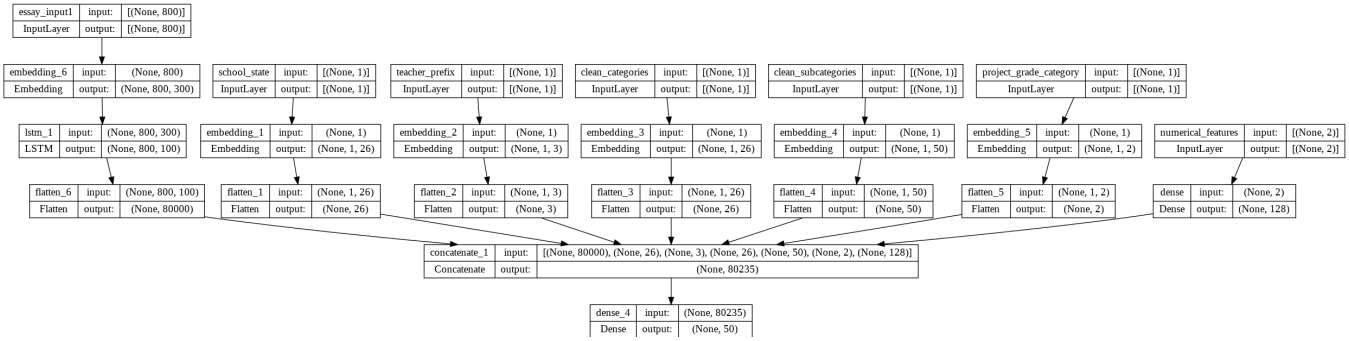
```
essay1 = Input(shape=(800,), name='essay_input1')

X = Embedding(output_dim=300, input_dim=max_vocabulary_1+1, input_length=800 , weights=[embedding_matrix_1])(essay1)
lstm_essay = LSTM(100,recurrent_dropout=0.5,kernel_regularizer= l2(0.001),return_sequences=True)(X)
flatten_new = Flatten()(lstm_essay)

X_concat = concatenate([flatten_new , flatten_2 , flatten_3 ,flatten_4 , flatten_5 , flatten_6 , numeric_dense])

model = Dense(50, activation="relu", kernel_initializer="he_normal" ,kernel_regularizer= l2(0.001))(X_concat)

model = Dropout(0.25)(model)

model = Dense(200,activation="relu",kernel_initializer="glorot_normal" ,kernel_regularizer= l2(0.001))(model)

model = BatchNormalization()(model)

model = Dropout(0.5)(model)
```

```
model = Dense(80,activation="sigmoid", kernel_initializer="glorot_normal" ,kernel_regularizer= l2(0.001))(model)

output = Dense(2, activation='softmax', name='output')(model)

model_2 = Model(inputs=[essay1, school_state ,teacher_prefix,clean_categories,clean_subcategories ,project_grade_category ,numeric_featur

print(model_2.summary())
```

```
lstm_1 (LSTM)                  (None, 800, 100)     160400      ['embedding_6[0][0]']

embedding_1 (Embedding)        (None, 1, 26)        1326        ['school_state[0][0]']

embedding_2 (Embedding)        (None, 1, 3)         15          ['teacher_prefix[0][0]']

embedding_3 (Embedding)        (None, 1, 26)        1326        ['clean_categories[0][0]']

embedding_4 (Embedding)        (None, 1, 50)        19700       ['clean_subcategories[0][0]']

embedding_5 (Embedding)        (None, 1, 2)         8           ['project_grade_category[0][0]']

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

flatten_6 (Flatten)            (None, 80000)        0           ['lstm_1[0][0]']

flatten_1 (Flatten)            (None, 26)           0           ['embedding_1[0][0]']

flatten_2 (Flatten)            (None, 3)            0           ['embedding_2[0][0]']

                               (None, 26)           0           ['embedding_3[0][0]']

                               (None, 50)           0           ['embedding_4[0][0]']

flatten_5 (Flatten)            (None, 2)            0           ['embedding_5[0][0]']

dense (Dense)                  (None, 128)          384         ['numerical_features[0][0]']

concatenate_1 (Concatenate)    (None, 80235)        0           ['flatten_6[0][0]',
                                                                 'flatten_1[0][0]',
                                                                 'flatten_2[0][0]',
                                                                 'flatten_3[0][0]',
                                                                 'flatten_4[0][0]',
                                                                 'flatten_5[0][0]',
                                                                 'dense[0][0]']

dense_4 (Dense)                (None, 50)           4011800     ['concatenate_1[0][0]']

dropout_2 (Dropout)            (None, 50)           0           ['dense_4[0][0]']

dense_5 (Dense)                (None, 200)          10200       ['dropout_2[0][0]']

batch_normalization_1 (BatchNo  (None, 200)         800         ['dense_5[0][0]']
rmalization)

dropout_3 (Dropout)            (None, 200)          0           ['batch_normalization_1[0][0]']

dense_6 (Dense)                (None, 80)           16080       ['dropout_3[0][0]']

output (Dense)                 (None, 2)            162         ['dense_6[0][0]']

==================================================================================================
Total params: 13,184,701
Trainable params: 13,184,301
Non-trainable params: 400
__________________________________________________________________________________________________
None
```

Saved successfully!

```
plot_model(model_2,'image_2.png',show_shapes=True)
```



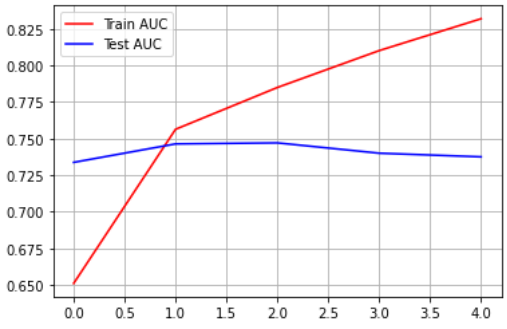

```
checkpoint_2 = ModelCheckpoint("model_2.h5",monitor="val_auroc",mode="max",save_best_only = True,verbose=1)
earlystop_2 = EarlyStopping(monitor = 'val_auroc', mode="max",min_delta = 0, patience = 2,verbose = 1)

tensorboard_2 = TensorBoard(log_dir='Model2_visualization')

callbacks_2 = [checkpoint_2,earlystop_2,tensorboard_2]
```

```
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auroc])
history_2 = model_2.fit(train_all, y_train, batch_size=900, epochs=30, verbose=1,callbacks=callbacks_2, validation_data=(test_all, y_test
```

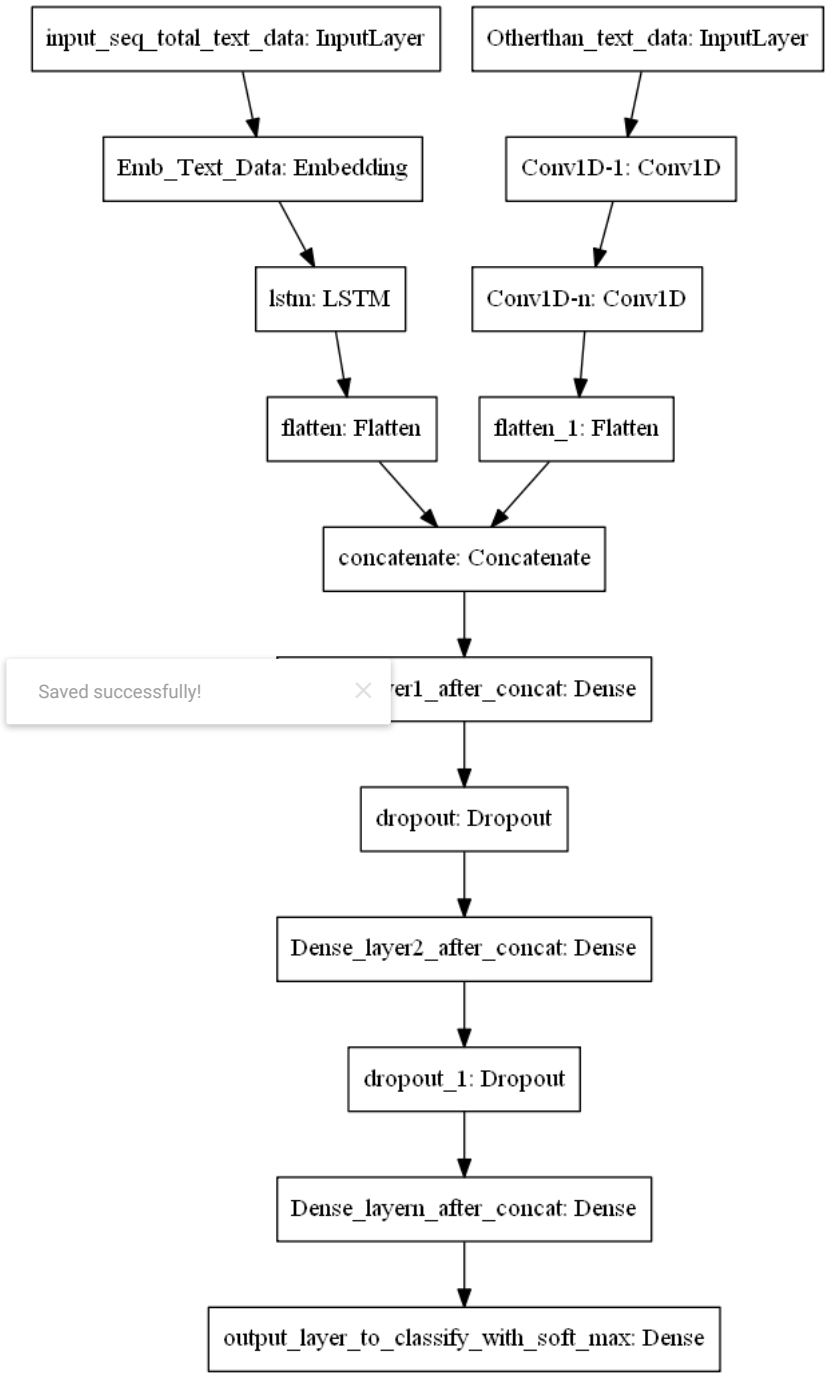
Saved successfully!

```
======] - ETA: 0s - loss: 0.6793 - auroc: 0.6511
m -inf to 0.73374, saving model to model_2.h5
85/85 [==============================] - 272s 3s/step - loss: 0.6793 - auroc: 0.6511 - val_loss: 0.5284 - val_auroc: 0.7337
Epoch 2/30
85/85 [==============================] - ETA: 0s - loss: 0.4672 - auroc: 0.7563
Epoch 2: val_auroc improved from 0.73374 to 0.74636, saving model to model_2.h5
85/85 [==============================] - 271s 3s/step - loss: 0.4672 - auroc: 0.7563 - val_loss: 0.4766 - val_auroc: 0.7464
Epoch 3/30
85/85 [==============================] - ETA: 0s - loss: 0.4156 - auroc: 0.7849
Epoch 3: val_auroc improved from 0.74636 to 0.74706, saving model to model_2.h5
85/85 [==============================] - 270s 3s/step - loss: 0.4156 - auroc: 0.7849 - val_loss: 0.4474 - val_auroc: 0.7471
Epoch 4/30
85/85 [==============================] - ETA: 0s - loss: 0.3833 - auroc: 0.8101
Epoch 4: val_auroc did not improve from 0.74706
85/85 [==============================] - 278s 3s/step - loss: 0.3833 - auroc: 0.8101 - val_loss: 0.4229 - val_auroc: 0.7400
Epoch 5/30
85/85 [==============================] - ETA: 0s - loss: 0.3574 - auroc: 0.8318
Epoch 5: val_auroc did not improve from 0.74706
85/85 [==============================] - 263s 3s/step - loss: 0.3574 - auroc: 0.8318 - val_loss: 0.4176 - val_auroc: 0.7375
Epoch 5: early stopping
```

```
from matplotlib import pyplot as plt
plt.plot(history_2.history['auroc'], 'r')
plt.plot(history_2.history['val_auroc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'g'})
plt.grid()
plt.show()
```



Model-3



Saved successfully!

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.
```

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)
X_train_state_one_hot = vectorizer.transform(X_train['school_state'].values)
X_test_state_one_hot = vectorizer.transform(X_test['school_state'].values)

print(X_train_state_one_hot.shape, y_train.shape)
print(X_test_state_one_hot.shape, y_test.shape)
```

```
(76473, 51) (76473, 2)
(32775, 51) (32775, 2)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)

X_train_categories_one_hot = vectorizer.transform(X_train['clean_categories'].values)
X_test_categories_one_hot = vectorizer.transform(X_test['clean_categories'].values)
```

```
print(vectorizer.get_feature_names())

print(X_train_categories_one_hot.shape, y_train.shape)
print(X_test_categories_one_hot.shape, y_test.shape)
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'specialne
(76473, 9) (76473, 2)
(32775, 9) (32775, 2)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; ge
  warnings.warn(msg, category=FutureWarning)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_subcategories_one_hot = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_subcategories_one_hot = vectorizer.transform(X_test['clean_subcategories'].values)
print(vectorizer.get_feature_names())

print(X_train_subcategories_one_hot.shape, y_train.shape)
print(X_test_subcategories_one_hot.shape, y_test.shape)
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'earlydevel
```

Saved successfully!

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)
X_train_teacher_prefix_one_hot = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_prefix_one_hot = vectorizer.transform(X_test['teacher_prefix'].values)

print(X_train_teacher_prefix_one_hot.shape, y_train.shape)
print(X_test_teacher_prefix_one_hot.shape, y_test.shape)
```

```
(76473, 5) (76473, 2)
(32775, 5) (32775, 2)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values)
X_train_project_grade_one_hot = vectorizer.transform(X_train['project_grade_category'].values)
X_test_project_grade_one_hot = vectorizer.transform(X_test['project_grade_category'].values)
print(vectorizer.get_feature_names())

print(X_train_project_grade_one_hot.shape, y_train.shape)
print(X_test_project_grade_one_hot.shape, y_test.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
(76473, 4) (76473, 2)
(32775, 4) (32775, 2)
```

```
from sklearn.preprocessing import StandardScaler
stnd_scaler=StandardScaler()
stnd_scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_num_projects=stnd_scaler.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_test_num_projects=stnd_scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print(X_train_num_projects.shape,X_test_num_projects.shape)
```

```
(76473, 1) (32775, 1)
```

```
stnd_scaler.fit(X_train['price'].values.reshape(-1,1))

X_train_price = stnd_scaler.transform(X_train['price'].values.reshape(-1,1))

X_test_price = stnd_scaler.transform(X_test['price'].values.reshape(-1,1))

print(X_train_price.shape,X_test_price.shape)
```

```
(76473, 1) (32775, 1)
```

```
X_train_numeric_features = np.concatenate((X_train_num_projects , X_train_price) , axis = 1)

X_test_numeric_features= np.concatenate((X_test_num_projects , X_test_price) , axis = 1)

print(X_train_numeric_features.shape ,X_test_numeric_features.shape)
```

```
(76473, 2) (32775, 2)
```

```
from scipy.sparse import hstack
train_features_without_text = hstack((X_train_state_one_hot, X_train_categories_one_hot, X_train_subcategories_one_hot, X_train_teacher_p
print(train_features_without_text.shape)

test_features_without_text = hstack((X_test_state_one_hot, X_test_categories_one_hot, X_test_subcategories_one_hot, X_test_teacher_prefix
print(test_features_without_text.shape)
```

```
(76473, 101)
(32775, 101)
```

```
print(X_train_pad.shape)
```

```
(76473, 800)
```

```
rest_features_train = np.expand_dims(train_features_without_text,2)
rest_features_test = np.expand_dims(test_features_without_text,2)
```

Saved successfully!

```
train_3 = [X_train_pad,X_train_pad1,train_features_without_text]
test_3= [X_test_pad,X_test_pad1,test_features_without_text]
```

```
from keras.layers import Input, Dense, Embedding, Flatten, concatenate, Dropout, Convolution1D, GlobalMaxPool1D, SpatialDropout1D, CuDNNG

essay = Input(shape=(800,), name='essay_input')
X = Embedding(output_dim=300, input_dim=max_vocabulary+1, input_length=800 , weights=[embedding_matrix])(essay)
x_words = LSTM(64,recurrent_dropout=0.3,kernel_regularizer=l2(0.001),return_sequences=True)(X)
flatten_1 = Flatten()(x_words)


essay1 = Input(shape=(800,), name='essay_input1')
X1 = Embedding(output_dim=300, input_dim=max_vocabulary_1+1, input_length=800 , weights=[embedding_matrix_1])(essay1)
x_words1 = LSTM(64,recurrent_dropout=0.3,kernel_regularizer=l2(0.001),return_sequences=True)(X1)
flatten_new = Flatten()(x_words1)


input_layer_other_than_text_data = Input(shape=(101,1),name="other_than_text_data")
conv1D_1 = Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer="he_normal")(input_layer_other_than_text_data)
conv1D_2 = Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer="he_normal")(conv1D_1)
flatten_other_than_text_data = Flatten()(conv1D_2)

X_concat = concatenate([flatten_1 ,flatten_new, flatten_other_than_text_data])

model = Dense(300, activation="relu", kernel_initializer="he_normal" ,kernel_regularizer=l2(0.001))(X_concat)

model = Dropout(0.25)(model)

model = Dense(150,activation="relu",kernel_initializer="glorot_normal" ,kernel_regularizer= l2(0.001))(model)

model = BatchNormalization()(model)

model = Dropout(0.5)(model)

model = Dense(80,activation="sigmoid", kernel_initializer="glorot_normal" ,kernel_regularizer= l2(0.001))(model)

output = Dense(2, activation='softmax', name='output')(model)

model_3 = Model(inputs=[essay,essay1,input_layer_other_than_text_data],outputs=[output])
```

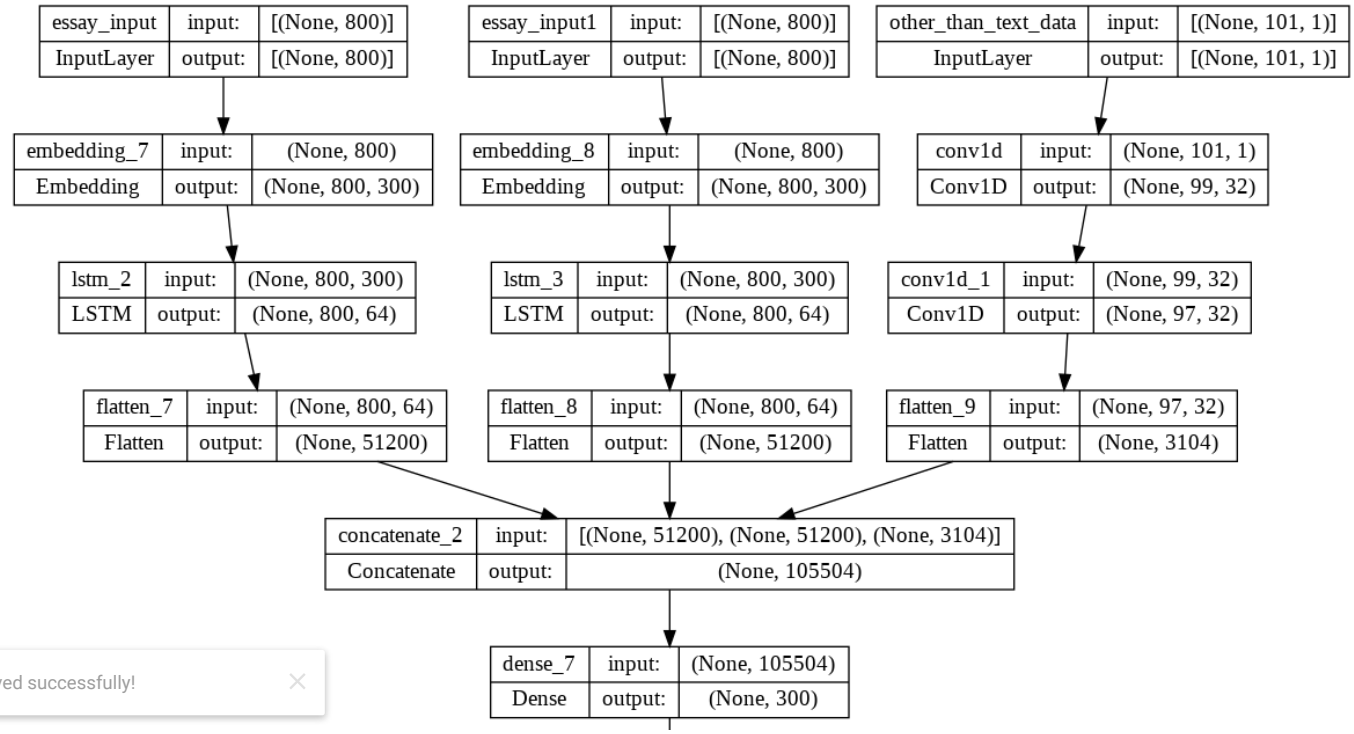
```
print(model_3.summary())
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 essay_input (InputLayer)       [(None, 800)]        0           []

 essay_input1 (InputLayer)      [(None, 800)]        0           []

 other_than_text_data (InputLay  [(None, 101, 1)]    0           []
 er)
```

```
embedding_7 (Embedding)        (None, 800, 300)     14708400    ['essay_input[0][0]']

embedding_8 (Embedding)        (None, 800, 300)     8962500     ['essay_input1[0][0]']

conv1d (Conv1D)                (None, 99, 32)       128         ['other_than_text_data[0][0]']

lstm_2 (LSTM)                  (None, 800, 64)      93440       ['embedding_7[0][0]']

lstm_3 (LSTM)                  (None, 800, 64)      93440       ['embedding_8[0][0]']

conv1d_1 (Conv1D)              (None, 97, 32)       3104        ['conv1d[0][0]']

flatten_7 (Flatten)            (None, 51200)        0           ['lstm_2[0][0]']

flatten_8 (Flatten)            (None, 51200)        0           ['lstm_3[0][0]']

flatten_9 (Flatten)            (None, 3104)         0           ['conv1d_1[0][0]']

concatenate_2 (Concatenate)    (None, 105504)       0           ['flatten_7[0][0]',
                                                                 'flatten_8[0][0]',
                                                                 'flatten_9[0][0]']

dense_7 (Dense)                (None, 300)          31651500    ['concatenate_2[0][0]']

dropout_4 (Dropout)            (None, 300)          0           ['dense_7[0][0]']

dense_8 (Dense)                (None, 150)          45150       ['dropout_4[0][0]']

                               (None, 150)          600         ['dense_8[0][0]']

dropout_5 (Dropout)            (None, 150)          0           ['batch_normalization_2[0][0]']

dense_9 (Dense)                (None, 80)           12080       ['dropout_5[0][0]']

output (Dense)                 (None, 2)            162         ['dense_9[0][0]']

==================================================================================================
Total params: 55,570,504
Trainable params: 55,570,204
Non-trainable params: 300
__________________________________________________________________________________________________
None
```

Saved successfully!

```
plot_model(model_3,'image_3.png',show_shapes=True)
```



Saved successfully!

```
checkpoint_3 = ModelCheckpoint("model_3_1.h5",
                               monitor="val_auroc",
                               mode="max",
                               save_best_only = True,
                               verbose=1)
earlystop_3 = EarlyStopping(monitor = 'val_auroc',
                            mode="max",
                            min_delta = 0,
                            patience = 2,
                            verbose = 1)

tensorboard_3 = TensorBoard(log_dir='Model3_1_visualization')

callbacks_3 = [checkpoint_3,earlystop_3,tensorboard_3]
```

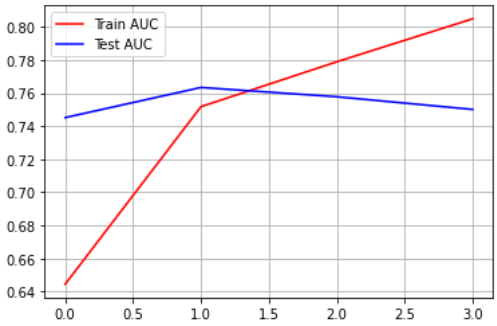
```
model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auroc])
```

| Dropout | output: | (None, 150) |

```
history_3 = model_3.fit(train_3, y_train, batch_size=500, epochs=25, verbose=1,callbacks=callbacks_3, validation_data=(test_3, y_test))
```

```
Epoch 1/25
WARNING:tensorflow:From /usr/local/lib/python3.8/dist-packages/tensorflow/python/autograph/impl/api.py:459: py_func (from tensorflo
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.

153/153 [==============================] - ETA: 0s - loss: 0.8628 - auroc: 0.6445
Epoch 1: val_auroc improved from -inf to 0.74516, saving model to model_3_1.h5
153/153 [==============================] - 4141s 27s/step - loss: 0.8628 - auroc: 0.6445 - val_loss: 0.5212 - val_auroc: 0.7452
Epoch 2/25
153/153 [==============================] - ETA: 0s - loss: 0.4685 - auroc: 0.7518
Epoch 2: val_auroc improved from 0.74516 to 0.76338, saving model to model_3_1.h5
153/153 [==============================] - 4203s 27s/step - loss: 0.4685 - auroc: 0.7518 - val_loss: 0.4506 - val_auroc: 0.7634
Epoch 3/25
153/153 [==============================] - ETA: 0s - loss: 0.4097 - auroc: 0.7789
Epoch 3: val_auroc did not improve from 0.76338
153/153 [==============================] - 4360s 29s/step - loss: 0.4097 - auroc: 0.7789 - val_loss: 0.4109 - val_auroc: 0.7578
Epoch 4/25
153/153 [==============================] - ETA: 0s - loss: 0.3785 - auroc: 0.8049
Epoch 4: val_auroc did not improve from 0.76338
153/153 [==============================] - 4370s 29s/step - loss: 0.3785 - auroc: 0.8049 - val_loss: 0.4171 - val_auroc: 0.7501
Epoch 4: early stopping
```

```
from matplotlib import pyplot as plt
plt.plot(history_3.history['auroc'], 'r')
plt.plot(history_3.history['val_auroc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'g'})
plt.grid()
plt.show()
```



Saved successfully!

✓ 0s completed at 2:05 AM