Assignment : 14

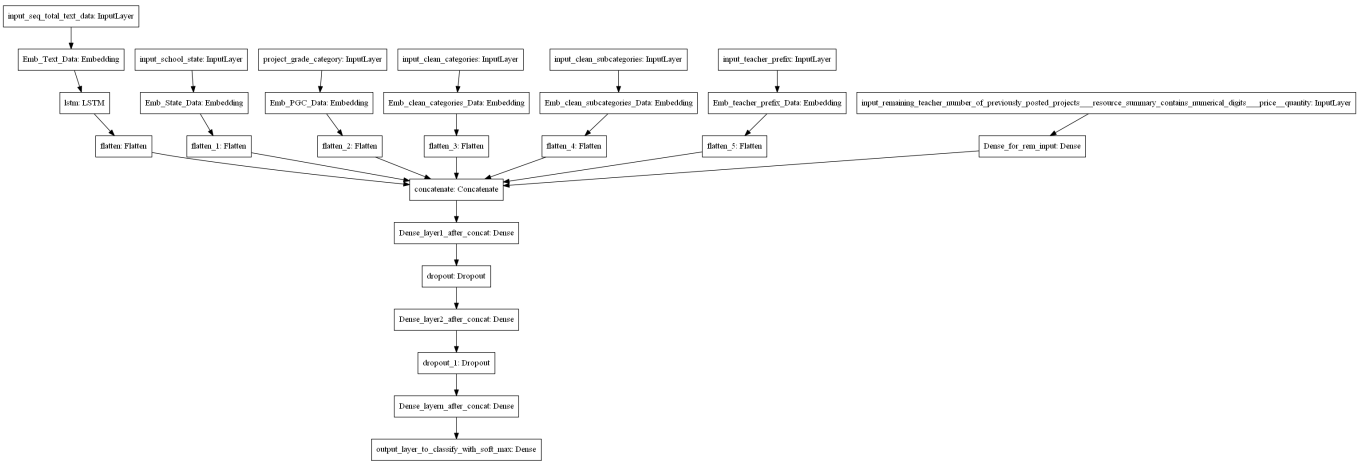
1. You can work with preprocessed_data.csv for the assignment. You can get the data from - Data folder
2. Load the data in your notebook.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check this and this for using auc as a metric
5. 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.

```
#importing libraries
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer,one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.layers import LSTM, BatchNormalization,concatenate,Flatten,Embedding,Dense,Dropout,MaxPooling2D,CuDNNLSTM,SpatialDropout1D
from keras.models import Sequential
from keras import Model,Input
from time import time
from tensorflow.python.keras.callbacks import TensorBoard,ModelCheckpoint
import warnings
warnings.filterwarnings("ignore")
import keras
from keras.regularizers import l2
import pickle
from keras.layers.convolutional import Conv2D,Conv1D
import keras.backend as k
from sklearn.metrics import roc_auc_score
import tensorflow as tf
from keras.initializers import he_normal
from keras.callbacks import Callback, EarlyStopping
```

```
#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
```

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.

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')
```

```
Mounted at /content/drive
```

```
#Reading the dataset
project_data = pd.read_csv('/content/drive/MyDrive/preprocessed_data.csv')
project_data.head(3)
```

	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

```
# defining the class label varibale
class_label = project_data['project_is_approved']
```

```
# droping class label data
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
X = project_data
project_data.shape
```

```
(109248, 8)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.2, stratify=y_train)

print('Train Data Set', X_train.shape, y_train.shape)
print('Cross Validate Data Set', X_cv.shape, y_cv.shape)
print('Test Data Set', X_test.shape, y_test.shape)
```

```
Train Data Set (69918, 8) (69918,)
Cross Validate Data Set (17480, 8) (17480,)
Test Data Set (21850, 8) (21850,)
```

```
project_data.replace(to_replace=np.NaN, value= str('nan'),inplace=True)
```

```
col = project_data.columns
col
```

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

```
col = ['teacher_prefix', 'school_state', 'project_grade_category',
       'clean_categories', 'clean_subcategories','essay',
       'price']
project_data = project_data[col]
```

```
# reading the pre trained word vectors file
dumb_file = open('/content/drive/MyDrive/Colab Notebooks/glove_vectors', 'rb')
dumb = pickle.load(dumb_file)
```

```
dumb['mallinson'].shape
```

```
(300,)
```

```
def word_ranking(dataframe):
    # performing train test split
    train, test, y_train, y_test = train_test_split(dataframe, class_label, stratify=class_label, train_size=0.7)
    train, cv, y_train, y_cv = train_test_split(train, y_train, stratify=y_train, train_size=0.8)

    # column names to consider
    col_names = dataframe.columns[:6]

    # list to store features for each column
    features = []

    # iterate over each column
    for col in col_names:
        # create CountVectorizer and fit on train data
        vectorizer = CountVectorizer(lowercase=False)
        bow_train = vectorizer.fit_transform(train[col])

        # get word frequencies and sort by frequency
        word_freqs = bow_train.sum(axis=0).A1
        word_indices = word_freqs.argsort()[::-1]
        words = vectorizer.get_feature_names()

        # assign ranks to words based on frequency
        word_rank = {word: rank+1 for rank, idx in enumerate(word_indices) for word in [words[idx]]}
        features.append(word_rank)

        # replace words with their ranks in train, test, and cv data
        for df in [train, test, cv]:
            ranks = []
            for sent in df[col].values:
                txt_row = [word_rank.get(word, 0) for word in sent.split()]
                ranks.append(txt_row)
            df[col] = ranks

    return train, test, cv, y_train, y_test, y_cv, features
```

```
train,test,cv,y_train,y_test,y_cv,feature_names = word_ranking(project_data)
```

```
print("Shape of the Train dataset: ", train.shape[0])
print("Shape of the Test dataset: ", test.shape[0])
print("Shape of the cv dataset:", cv.shape[0])
```

```
Shape of the Train dataset:  61178
Shape of the Test dataset:  32775
Shape of the cv dataset: 15295
```

```
#converting class labels to categorical variables
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_cv = to_categorical(y_cv)
```

```
from sklearn.utils import compute_class_weight
class_wght = compute_class_weight("balanced", classes= np.unique(class_label),y=class_label)
class_wght
```

```
array([3.30214001, 0.58921753])
```

```
feature_names[4]
```

```
{'literacy': 1,
 'mathematics': 2,
 'literature_writing': 3,
 'specialneeds': 4,
 'appliedsciences': 5,
 'health_wellness': 6,
 'visualarts': 7,
 'environmentalscience': 8,
 'gym_fitness': 9,
 'esl': 10,
 'health_lifescience': 11,
 'earlydevelopment': 12,
 'music': 13,
 'history_geography': 14,
 'college_careerprep': 15,
 'other': 16,
 'teamsports': 17,
 'charactereducation': 18,
 'performingarts': 19,
 'socialsciences': 20,
 'care_hunger': 21,
 'warmth': 22,
 'nutritioneducation': 23,
 'foreignlanguages': 24,
 'extracurricular': 25,
 'civics_government': 26,
 'parentinvolvement': 27,
 'financialliteracy': 28,
 'communityservice': 29,
 'economics': 30}
```

```
#Creating a matrix with rows as words and columns with 50 dim vectors for each word
def embedding_mat(word_index,embedding_dim = 300):
    embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
    for word, i in word_index.items():
        embedding_vector = dumb.get(word)
        if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix
```

Tokenizing the Text part

```
max_review_length = 250

X_train = pad_sequences(train['essay'], maxlen=max_review_length, padding='pre', truncating='pre')
X_test = pad_sequences(test['essay'], maxlen=max_review_length, padding='pre', truncating='pre')
X_cv = pad_sequences(cv['essay'], maxlen=max_review_length, padding='pre', truncating='pre')

print(X_train.shape)
print(X_train[256])
```

```
(61178, 250)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
   0    0    0    0    0    0    0    3    1   21   86  130  136  978
2211 3700   13  415   63  296   80   13   27  162  336 1605 2570 1918
 827  166   29   24    1  447    7   97   62  122   70  356    0   70
   5   24    2  373    1  177  441  228 7477 2924    1  749  270  568
 203    0  361   48   26  149  172  125  760   37   16  101 2435    4
 321  161  398  481  422  426  464  188   38    6  285 3026  997 2741
 128  473    1  483  101    3    1   18  321  325  228  321    1 1331
 125  101  213   16  101  298  648  116  309  808   19  516 1188   88
 101   37  321   44    1    9   26   17  166    4  886   12]
```

Tokenizing the school state

```
max_review_length = 1
# here we are using prepadding
X_train_school_state = pad_sequences(train['school_state'], maxlen=max_review_length, padding='pre', truncating='pre')
X_test_school_state = pad_sequences(test['school_state'], maxlen=max_review_length, padding='pre', truncating='pre')
X_cv_school_state = pad_sequences(cv['school_state'], maxlen=max_review_length, padding='pre', truncating='pre')

print(X_test_school_state.shape)
print(X_test_school_state[0])
```

```
(32775, 1)
[2]
```

Tokenizing the project_grade_category

```
max_review_length = 1
X_train_project_grade = pad_sequences(train['project_grade_category'], maxlen=max_review_length)  #padding zeros at the begining of each
X_test_project_grade = pad_sequences(test['project_grade_category'], maxlen=max_review_length)
X_cv_project_grade = pad_sequences(cv['project_grade_category'], maxlen=max_review_length)
print(X_train_project_grade.shape)
print(X_train_project_grade[0])
```

```
(61178, 1)
[1]
```

Tokenizing the clean_categories

```
max_review_length = 1
X_train_clean_categories = pad_sequences(train['clean_categories'], maxlen=max_review_length)  #padding zeros at the begining of each rev
X_test_clean_categories = pad_sequences(test['clean_categories'], maxlen=max_review_length)
X_cv_clean_categories = pad_sequences(cv['clean_categories'], maxlen=max_review_length)
print(X_train_clean_categories.shape)
print(X_train_clean_categories[0])
```

```
(61178, 1)
[2]
```

Tokenizing the clean_subcategories

```
max_review_length = 1
X_train_clean_subcategories = pad_sequences(train['clean_subcategories'], maxlen=max_review_length)  #padding zeros at the begining of ea
X_test_clean_subcategories = pad_sequences(test['clean_subcategories'], maxlen=max_review_length)
X_cv_clean_subcategories = pad_sequences(cv['clean_subcategories'], maxlen=max_review_length)
print(X_train_clean_subcategories.shape)
print(X_train_clean_subcategories[0])
```

```
(61178, 1)
[2]
```

Tokenizing the teacher prefix

```
max_review_length = 1
X_train_teacher_prefix = pad_sequences(train['teacher_prefix'], maxlen=max_review_length)  #padding zeros at the begining of each review
X_test_teacher_prefix = pad_sequences(test['teacher_prefix'], maxlen=max_review_length)
X_cv_teacher_prefix = pad_sequences(cv['teacher_prefix'], maxlen=max_review_length)
```

```
print(X_train_teacher_prefix.shape)
print(X_test_teacher_prefix[0])
```

```
(61178, 1)
[1]
```

```
train.head()
```

	teacher_prefix	school_state	project_grade_category	clean_categories	clean_subcategories	essay	price
56757	[2]	[22]	[1]	[2]	[2]	[3, 1, 21, 117, 139, 50, 58, 0, 48, 188, 38, 2...	60.59
5580	[2]	[6]	[3]	[5, 4]	[16, 4]	[105, 75, 42, 59, 1, 29, 11805, 7, 2483, 731, ...	269.99
18657	[1]	[5]	[1]	[4]	[4]	[0, 48, 501, 937, 546, 0, 1, 8366, 129, 214, 1...	693.06
						[0, 332, 1240, 366, 2, 29,	

Deep Learning Models

Model 1

```
# defining the auc score for model
def auc( y_true, y_pred ) :
    score = tf.numpy_function(lambda y_true, y_pred : roc_auc_score( y_true, y_pred ,average='weighted').astype('float32'),
                        [y_true, y_pred],
                        'float32',
                        name='sklearnAUC')
    return score


# defining the accuracy score for model

def accuracy(y_true, y_pred):
    y_pred = tf.argmax(y_pred, axis=1)
    accuracy = tf.keras.metrics.Accuracy()
    accuracy.update_state(y_true, y_pred)
    return accuracy.result().numpy()



# defining learning rate function with value of 0.0001
def step_decay(epoch):
    initial_lr = 0.0001
    lr_drop = 1e-6
    epochs_drop = 1
    lr = initial_lr * math.pow(lr_drop, math.floor((1 + epoch) / epochs_drop))
    return lr


#input 1
input1 = Input(shape=(250,))
x1 = Embedding(input_dim=44899,output_dim= 300,weights=[embedding_mat(feature_names[5])],trainable=False)(input1)
x1 = SpatialDropout1D(0.3)(x1)
x1 = LSTM(128,return_sequences=True)(x1)
x1 = Flatten()(x1)

#input 2
input2 = Input(shape=(1,))
x2 = Embedding(input_dim= 52,output_dim= 2)(input2)
x2 = Flatten()(x2)

#input 3
input3 = Input(shape=(1,))
x3 = Embedding(input_dim= 5,output_dim= 2)(input3)
x3 = Flatten()(x3)

#input 4
input4 = Input(shape=(1,))
x4 = Embedding(input_dim=50,output_dim= 2)(input4)
x4 = Flatten()(x4)

#input 5
input5 = Input(shape=(1,))
x5 = Embedding(input_dim= 385,output_dim= 50)(input5)
x5 = Flatten()(x5)

#input 6
input6 = Input(shape=(1,))
x6 = Embedding(input_dim= 6,output_dim= 5)(input6)
```

```
x6 = Flatten()(x6)

#input 7
input7 = Input(shape=(1,))
x7 = Dense(16,activation='sigmoid',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(input7)
# now will merge all the input layer using concatenate function
concated_layer = concatenate([x1,x2,x3,x4,x5,x6,x7])
#x = BatchNormalization()(concat)

x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(concated_layer)
x = Dropout(0.3)(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.3)(x)
output = Dense(2, activation = 'softmax')(x)

# created the model with all inputs
model = Model([input1,input2,input3,input4,input5,input6,input7], output)
tensorboard = TensorBoard(log_dir='logs/{}'.format(time()))
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0006),metrics=['accuracy' ,auc])
print(model.summary())
```

```
 embedding_2 (Embedding)        (None, 1, 2)         104         ['input_3[0][0]']

 embedding_3 (Embedding)        (None, 1, 2)         10          ['input_4[0][0]']

 embedding_4 (Embedding)        (None, 1, 2)         100         ['input_5[0][0]']

 embedding_5 (Embedding)        (None, 1, 50)        19250       ['input_6[0][0]']

 embedding_6 (Embedding)        (None, 1, 5)         30          ['input_7[0][0]']

 input_8 (InputLayer)           [(None, 1)]          0           []

 flatten (Flatten)              (None, 32000)        0           ['lstm[0][0]']

 flatten_1 (Flatten)            (None, 2)            0           ['embedding_2[0][0]']

 flatten_2 (Flatten)            (None, 2)            0           ['embedding_3[0][0]']

 flatten_3 (Flatten)            (None, 2)            0           ['embedding_4[0][0]']

 flatten_4 (Flatten)            (None, 50)           0           ['embedding_5[0][0]']

 flatten_5 (Flatten)            (None, 5)            0           ['embedding_6[0][0]']

 dense (Dense)                  (None, 16)           32          ['input_8[0][0]']

 concatenate (Concatenate)      (None, 32077)        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, 128)          4105984     ['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, 2)            66          ['dropout_2[0][0]']

==================================================================================================
Total params: 17,825,516
Trainable params: 4,355,688
Non-trainable params: 13,469,828
__________________________________________________________________________________________________
None
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
#now fit and train the model
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
filepath="weights_2.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_auc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]

# Remove TensorBoard from the callbacks_list
model.fit([X_train, X_train_school_state, X_train_project_grade, X_train_clean_categories, X_train_clean_subcategories,
           X_train_teacher_prefix, train['price']], y_train, epochs=50, verbose=1, batch_size=256,
          validation_data=([X_cv, X_cv_school_state, X_cv_project_grade, X_cv_clean_categories, X_cv_clean_subcategories,
                            X_cv_teacher_prefix, cv['price']], y_cv), callbacks=callbacks_list)
```

```
Epoch 37/50
239/239 [==============================] - ETA: 0s - loss: 0.2369 - accuracy: 0.9846 - auc: 0.9955
Epoch 00037: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 54ms/step - loss: 0.2369 - accuracy: 0.9846 - auc: 0.9955 - val_loss: 0.9783 - va
Epoch 38/50
239/239 [==============================] - ETA: 0s - loss: 0.2373 - accuracy: 0.9851 - auc: 0.9957
Epoch 00038: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 54ms/step - loss: 0.2373 - accuracy: 0.9851 - auc: 0.9957 - val_loss: 0.9162 - va
Epoch 39/50
239/239 [==============================] - ETA: 0s - loss: 0.2408 - accuracy: 0.9846 - auc: 0.9956
Epoch 00039: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 56ms/step - loss: 0.2408 - accuracy: 0.9846 - auc: 0.9956 - val_loss: 0.9269 - va
Epoch 40/50
238/239 [============================>.] - ETA: 0s - loss: 0.2394 - accuracy: 0.9849 - auc: 0.9958
Epoch 00040: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 55ms/step - loss: 0.2395 - accuracy: 0.9849 - auc: 0.9958 - val_loss: 0.9743 - va
Epoch 41/50
238/239 [============================>.] - ETA: 0s - loss: 0.2383 - accuracy: 0.9855 - auc: 0.9956
Epoch 00041: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 54ms/step - loss: 0.2384 - accuracy: 0.9855 - auc: 0.9956 - val_loss: 0.9561 - va
Epoch 42/50
238/239 [============================>.] - ETA: 0s - loss: 0.2365 - accuracy: 0.9850 - auc: 0.9958
Epoch 00042: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 55ms/step - loss: 0.2365 - accuracy: 0.9850 - auc: 0.9958 - val_loss: 0.9474 - va
Epoch 43/50
239/239 [==============================] - ETA: 0s - loss: 0.2348 - accuracy: 0.9863 - auc: 0.9956
Epoch 00043: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 54ms/step - loss: 0.2348 - accuracy: 0.9863 - auc: 0.9956 - val_loss: 0.9339 - va
Epoch 44/50
238/239 [============================>.] - ETA: 0s - loss: 0.2329 - accuracy: 0.9859 - auc: 0.9954
Epoch 00044: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 55ms/step - loss: 0.2330 - accuracy: 0.9859 - auc: 0.9954 - val_loss: 0.9043 - va
Epoch 45/50
239/239 [==============================] - ETA: 0s - loss: 0.2313 - accuracy: 0.9867 - auc: 0.9960
Epoch 00045: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 54ms/step - loss: 0.2313 - accuracy: 0.9867 - auc: 0.9960 - val_loss: 0.9343 - va
Epoch 46/50
238/239 [============================>.] - ETA: 0s - loss: 0.2348 - accuracy: 0.9862 - auc: 0.9955
Epoch 00046: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 53ms/step - loss: 0.2350 - accuracy: 0.9861 - auc: 0.9955 - val_loss: 0.9360 - va
Epoch 47/50
238/239 [============================>.] - ETA: 0s - loss: 0.2292 - accuracy: 0.9872 - auc: 0.9956
Epoch 00047: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 54ms/step - loss: 0.2292 - accuracy: 0.9872 - auc: 0.9956 - val_loss: 0.8949 - va
Epoch 48/50
239/239 [==============================] - ETA: 0s - loss: 0.2334 - accuracy: 0.9862 - auc: 0.9953
Epoch 00048: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 54ms/step - loss: 0.2334 - accuracy: 0.9862 - auc: 0.9953 - val_loss: 0.9888 - va
Epoch 49/50
238/239 [============================>.] - ETA: 0s - loss: 0.2302 - accuracy: 0.9864 - auc: 0.9962
Epoch 00049: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 55ms/step - loss: 0.2302 - accuracy: 0.9863 - auc: 0.9962 - val_loss: 0.9803 - va
Epoch 50/50
239/239 [==============================] - ETA: 0s - loss: 0.2318 - accuracy: 0.9870 - auc: 0.9959
Epoch 00050: val_auc did not improve from 0.75975
239/239 [==============================] - 13s 55ms/step - loss: 0.2318 - accuracy: 0.9870 - auc: 0.9959 - val_loss: 0.9617 - va
<keras.callbacks.History at 0x7f6fd0097d90>
```

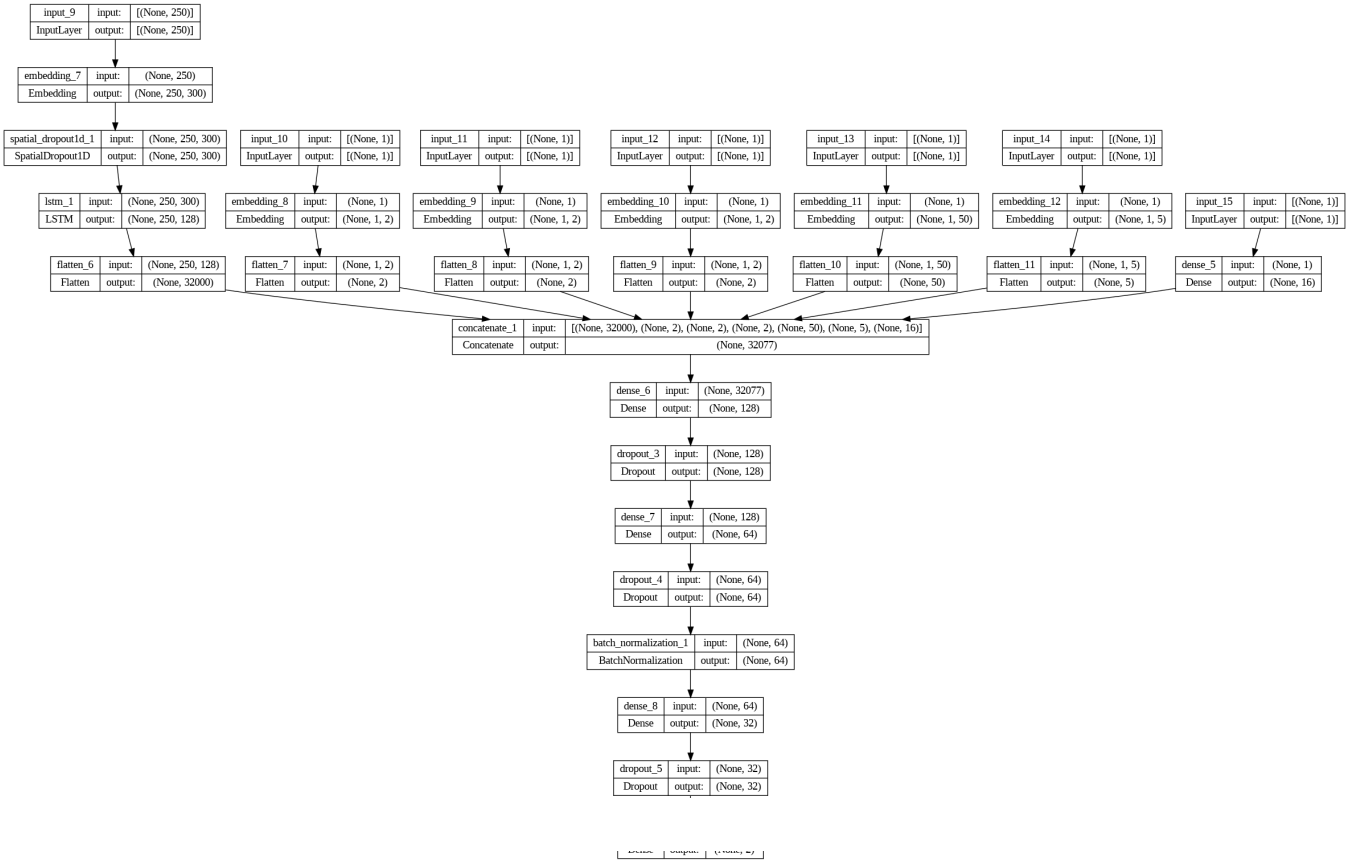
We have got the auc score 99.59 % and accuracy 98.70

```
#input 1
input1 = Input(shape=(250,))
x1 = Embedding(input_dim=44899,output_dim= 300,weights=[embedding_mat(feature_names[5])],trainable=False)(input1)
x1 = SpatialDropout1D(0.3)(x1)
x1 = LSTM(128,return_sequences=True)(x1)
x1 = Flatten()(x1)

#input 2
input2 = Input(shape=(1,))
x2 = Embedding(input_dim= 52,output_dim= 2)(input2)
```

```
x2 = Flatten()(x2)

#input 3
input3 = Input(shape=(1,))
x3 = Embedding(input_dim= 5,output_dim= 2)(input3)
x3 = Flatten()(x3)

#input 4
input4 = Input(shape=(1,))
x4 = Embedding(input_dim=50,output_dim= 2)(input4)
x4 = Flatten()(x4)

#input 5
input5 = Input(shape=(1,))
x5 = Embedding(input_dim= 385,output_dim= 50)(input5)
x5 = Flatten()(x5)

#input 6
input6 = Input(shape=(1,))
x6 = Embedding(input_dim= 6,output_dim= 5)(input6)
x6 = Flatten()(x6)

#input 7
input7 = Input(shape=(1,))
x7 = Dense(16,activation='sigmoid',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(input7)
# now will merge all the input layer using concatenate function
concated_layer = concatenate([x1,x2,x3,x4,x5,x6,x7])
#x = BatchNormalization()(concat)

x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(concated_layer)
x = Dropout(0.3)(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.3)(x)
output = Dense(2, activation = 'softmax')(x)

# created the model with all inputs
model = Model([input1,input2,input3,input4,input5,input6,input7], output)
tensorboard = TensorBoard(log_dir='logs/{}'.format(time()))
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0006),metrics=['accuracy' ,auc])
model.load_weights("weights_2.best.hdf5")


print("Auc for test data: %0.3f"%roc_auc_score(y_test,model.predict([X_test,X_test_school_state,X_test_project_grade,X_test_clean_categor
          X_test_teacher_prefix,test['price']])))
print("Auc for CV data: %0.3f"%roc_auc_score(y_cv,model.predict([X_cv,X_cv_school_state,X_cv_project_grade,X_cv_clean_categories,X_cv_cle
          X_cv_teacher_prefix,cv['price']])))
print("Auc for train data: %0.3f"%roc_auc_score(y_train,model.predict([X_train,X_train_school_state,X_train_project_grade,X_train_clean_c
          X_train_teacher_prefix,train['price']])))
```

```
1025/1025 [==============================] - 8s 7ms/step
Auc for test data: 0.752
478/478 [==============================] - 3s 6ms/step
Auc for CV data: 0.760
1912/1912 [==============================] - 12s 6ms/step
Auc for train data: 0.796
```

```
from tensorflow.keras.utils import plot_model



plot_model(model, to_file='model.png', show_shapes=True)
```



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.
5. Perform tokenization on the modified text data same as you have done for previous model.
6. Create embedding matrix for model 2 and then use the rest of the features similar to previous model.
7. Define the model, compile and fit the model.

```
class_label = project_data['project_is_approved']
```

```
#train test split
train,test,y_train,y_test = train_test_split(project_data, class_label , stratify = class_label, train_size = 0.7)
```

```
train,cv,y_train,y_cv = train_test_split(train,y_train,stratify = y_train,train_size = 0.8)
```

```
print("Shape of the Train dataset: ", train.shape[0])
print("Shape of the Test dataset: ", test.shape[0])
print("Shape of the CV Dataset:", cv.shape[0])
```

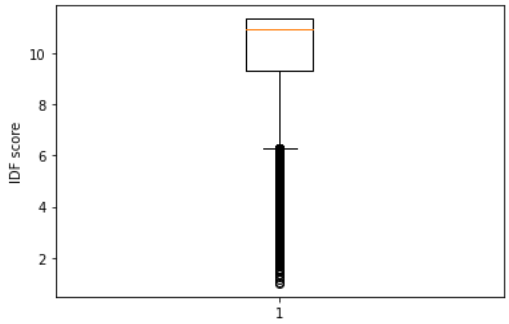
```
Shape of the Train dataset:  61178
Shape of the Test dataset:  32775
Shape of the CV Dataset: 15295
```

```
# refer : https://stackoverflow.com/questions/67018079/error-in-from-keras-utils-import-to-categorical

from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_cv = to_categorical(y_cv)
```

```
# now we are creating rows and columns as 50 dimensional vector

def embedding_mat(word_index,embedding_dim = 300):
    embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
    for word, i in word_index.items():
        embedding_vector =dumb.get(word)
        if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix
```

```
# Convert the data in the 'total_txt' column to a list of strings
#tfidf vectorization of text data

tfidf = TfidfVectorizer()
data_text = tfidf.fit_transform(train['essay'])
plt.boxplot(tfidf.idf_)
plt.ylabel("IDF score")
```

```
Text(0, 0.5, 'IDF score')
```




```
print("The 25 percentile of idf score is :", np.percentile(tfidf.idf_,[25]))
print("The 75 percentile of idf score is :",np.percentile(tfidf.idf_,[75]))
```

```
The 25 percentile of idf score is : [9.31350907]
The 75 percentile of idf score is : [11.32841209]
```

```
feature_idf = zip(tfidf.get_feature_names(),tfidf.idf_)

feature_name = []
for x,y in feature_idf:

    if y >=9.31350907 and 11.32841209 :
        feature_name.append(x)
    else:
        pass
```

```
def few_text(df, feature_name):
    processed_text = []
    feature_set = set(feature_name)
    for text in df:
        sent = " ".join([word for word in text.split() if word in feature_set])
        processed_text.append(sent)
    return processed_text

train['processed_essay'] = few_text(train['essay'], feature_name)
```

```
test['processed_essay'] = few_text(test['essay'], feature_name)
cv['processed_essay'] = few_text(cv['essay'], feature_name)


# now we are converting all the data to cs files
train.to_csv("model-train.csv", index=False)
test.to_csv("model-test.csv", index=False)
cv.to_csv("model-cv.csv", index=False)


train['essay'] =  train['essay']
test['essay'] = test['essay']
cv['essay'] = cv['essay']


y_train = train['project_is_approved']
y_test = test['project_is_approved']
y_cv = cv['project_is_approved']

# converting the class labels to one hot encoding for keras model evaluation
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_cv = to_categorical(y_cv)


train.head(2)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	cl
103461	pa	mrs	grades_3_5	100	1	
76709	ok	mrs	grades_3_5	9	1	

```
def word_ranking(train, test, cv):
    col_names = train.columns
    features = []

    for col in col_names[:6]:
        print(col)
        bag_of_words = CountVectorizer(lowercase=False)
        bow_words = bag_of_words.fit_transform(train[col])
        print(bow_words.shape)

        # Rank the words by frequency of occurrence
        word_freqs = dict(zip(bag_of_words.get_feature_names(), bow_words.sum(axis=0).A1))
        sorted_words = sorted(word_freqs, key=word_freqs.get, reverse=True)
        word_rank = dict(zip(sorted_words, range(1, len(sorted_words) + 1)))
        features.append(word_rank)

        # Replace words with their rank
        train[col] = [[word_rank[word] for word in sent.split() if word in word_rank] for sent in train[col].values]
        test[col] = [[word_rank[word] for word in sent.split() if word in word_rank] for sent in test[col].values]
        cv[col] = [[word_rank[word] for word in sent.split() if word in word_rank] for sent in cv[col].values]

    return train, test, cv, features


col = ['teacher_prefix', 'school_state', 'project_grade_category',
       'clean_categories', 'clean_subcategories','essay',
       'price']
train = train[col]
test = test[col]
test = test[col]
```

replacing nan value

```
train.replace(to_replace=np.NaN, value= str('nan'),inplace=True)
test.replace(to_replace=np.NaN, value= str('nan'),inplace=True)
```

```
cv.replace(to_replace=np.NaN, value= str('nan'),inplace=True)


train,test,cv,feature_names =  word_ranking(train,test,cv)
```

```
teacher_prefix
(61178, 5)
school_state
(61178, 51)
project_grade_category
(61178, 4)
clean_categories
(61178, 9)
clean_subcategories
(61178, 30)
essay
(61178, 45000)
```

Tokenizing the Test part

```
max_review_length = 250
X_train = pad_sequences(train['essay'], maxlen=max_review_length)  #padding zeros at the begining of each review to make max len as 200
X_test = pad_sequences(test['essay'], maxlen=max_review_length)
X_cv = pad_sequences(cv['essay'], maxlen=max_review_length)
print(X_test.shape)
print(X_train[256])
```

```
(32775, 250)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    3    1   21   11  103  139   34 1282  280   67
  126    5   24    5  322 1540  104   60  412  194 3214  202  579   59
  212    1 1200 2701    3    1 2744  143   84   29  314  497 1532  104
   60   57 3026   77 1200   24    2 1198  158 1523  123    4   73  125
 1274   25 2596 1543  205 1475 1383    1   24    2 2068  412 8199   59
   57  113  212  497 1532  104   60    1   84  437  227   52 4565  460
 2615   30 3565   42    1   65 1127  117  457   15  152  682  437  126
    2   11    1 6209  638  223  626  437  227    5  174  275  283    1
  286   19   10    9  151  514 1122  564 2350   19  550  437  751   83
  291  212    1 2275   30  626 1874  227   66  209  460 1119  177  117
  437  227    5   42    1  396  358   69   41  142   20   12]
```

Tokenizing the school state

```
max_review_length = 1
X_train_school_state = pad_sequences(train['school_state'], maxlen=max_review_length)  #padding zeros at the begining of each review to m
X_test_school_state = pad_sequences(test['school_state'], maxlen=max_review_length)
X_cv_school_state = pad_sequences(cv['school_state'], maxlen=max_review_length)
print(X_test_school_state.shape)
print(X_test_school_state[0])
```

```
(32775, 1)
[1]
```

Tokenizing the project_grade_category

```
max_review_length = 1
X_train_project_grade = pad_sequences(train['project_grade_category'], maxlen=max_review_length)  #padding zeros at the begining of each
X_test_project_grade = pad_sequences(test['project_grade_category'], maxlen=max_review_length)
X_cv_project_grade = pad_sequences(cv['project_grade_category'], maxlen=max_review_length)
print(X_test_project_grade.shape)
print(X_train_project_grade[0])
```

```
(32775, 1)
[2]
```

Tokenizing the project categories

```
max_review_length = 1
X_train_clean_categories = pad_sequences(train['clean_categories'], maxlen=max_review_length)  #padding zeros at the begining of each rev
X_test_clean_categories = pad_sequences(test['clean_categories'], maxlen=max_review_length)
X_cv_clean_categories = pad_sequences(cv['clean_categories'], maxlen=max_review_length)
print(X_test_clean_categories.shape)
print(X_train_clean_categories[0])
```

```
(32775, 1)
[7]
```

Tokenizing the project subcategories

```
max_review_length = 1
X_train_clean_subcategories = pad_sequences(train['clean_subcategories'], maxlen=max_review_length)  #padding zeros at the begining of ea
X_test_clean_subcategories = pad_sequences(test['clean_subcategories'], maxlen=max_review_length)
X_cv_clean_subcategories = pad_sequences(cv['clean_subcategories'], maxlen=max_review_length)
print(X_test_clean_subcategories.shape)
print(X_train_clean_subcategories[0])
```

```
(32775, 1)
[28]
```

Tokenizing the teacher prefix

```
max_review_length = 1
X_train_teacher_prefix = pad_sequences(train['teacher_prefix'], maxlen=max_review_length)  #padding zeros at the begining of each review
X_test_teacher_prefix = pad_sequences(test['teacher_prefix'], maxlen=max_review_length)
X_cv_teacher_prefix = pad_sequences(cv['teacher_prefix'], maxlen=max_review_length)
print(X_test_teacher_prefix.shape)
print(X_test_teacher_prefix[0])
```

```
(32775, 1)
[1]
```

```
train.head()
```

	teacher_prefix	school_state	project_grade_category	clean_categories	clean_subcategories	essay	price
103461	[1]	[10]	[2]	[7]	[28]	[210, 511, 517, 11, 110, 67, 298, 3129, 1289, ...	132.20
76709	[1]	[17]	[2]	[1, 2]	[3, 2]	[1, 657, 38, 5, 188, 359, 73, 24, 73, 5449, 91...	9.09
15554	[1]	[38]	[3]	[2]	[2]	[3, 1, 147, 4335, 693, 1128, 211, 4, 879, 234,...	202.87
						[217, 1, 420, 964, 18, 31,	

Deep Learning Models

Model 1

```
# defining the auc score for model
def auc( y_true, y_pred ) :
    score = tf.numpy_function(lambda y_true, y_pred : roc_auc_score( y_true, y_pred ,average='weighted').astype('float32'),
                        [y_true, y_pred],
                        'float32',
                        name='sklearnAUC')
    return score


# defining the accuracy score for model

def accuracy(y_true, y_pred):
    y_pred = tf.argmax(y_pred, axis=1)
    accuracy = tf.keras.metrics.Accuracy()
    accuracy.update_state(y_true, y_pred)
    return accuracy.result().numpy()



# defining learning rate function with value of 0.0001
def step_decay(epoch):
```

```
    initial_lr = 0.0001
    lr_drop = 1e-6
    epochs_drop = 1
    lr = initial_lr * math.pow(lr_drop, math.floor((1 + epoch) / epochs_drop))
    return lr



#input 1
input1 = Input(shape=(250,))
x1 = Embedding(input_dim=45001,output_dim= 300,weights=[embedding_mat(feature_names[5])],trainable=False)(input1)
x1 = SpatialDropout1D(0.3)(x1)
x1 = LSTM(128,return_sequences=True)(x1)
x1 = Flatten()(x1)

#input 2
input2 = Input(shape=(1,))
x2 = Embedding(input_dim= 55,output_dim= 2)(input2)
x2 = SpatialDropout1D(0.3)(x2)
x2 = Flatten()(x2)

#input 3
input3 = Input(shape=(1,))
x3 = Embedding(input_dim= 5,output_dim= 2)(input3)
x3 = SpatialDropout1D(0.3)(x3)
x3 = Flatten()(x3)

#input 4
input4 = Input(shape=(1,))
x4 = Embedding(input_dim=50,output_dim= 2)(input4)
x4 = SpatialDropout1D(0.3)(x4)
x4 = Flatten()(x4)

#input 5
input5 = Input(shape=(1,))
x5 = Embedding(input_dim= 385,output_dim= 50)(input5)
x5 = SpatialDropout1D(0.3)(x5)
x5 = Flatten()(x5)

#input 6
input6 = Input(shape=(1,))
x6 = Embedding(input_dim= 6,output_dim= 5)(input6)
x6 = SpatialDropout1D(0.3)(x6)
x6 = Flatten()(x6)

#input 7
input7 = Input(shape=(1,))
x7 = Dense(16,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(input7)
x7 = Flatten()(x7)
# now will merge the all the input
concatated_layer = concatenate([x1,x2,x3,x4,x5,x6,x7])

x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(concatated_layer)
x = Dropout(0.5)(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.5)(x)
output = Dense(2, activation = 'softmax')(x)

# create model with seven inputs
model = Model([input1,input2,input3,input4,input5,input6,input7], output)
tensorboard = TensorBoard(log_dir='logs/{}'.format(time()))
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0006,decay = 1e-4),metrics=['accuracy',auc])
print(model.summary())
```

```
flatten_2 (Flatten)            (None, 2)            0           ['spatial_dropout1d_2[0][0]']

flatten_3 (Flatten)            (None, 2)            0           ['spatial_dropout1d_3[0][0]']

flatten_4 (Flatten)            (None, 50)           0           ['spatial_dropout1d_4[0][0]']

flatten_5 (Flatten)            (None, 5)            0           ['spatial_dropout1d_5[0][0]']

flatten_6 (Flatten)            (None, 16)           0           ['dense[0][0]']

concatenate (Concatenate)      (None, 32077)        0           ['flatten[0][0]',
                                                                 'flatten_1[0][0]',
                                                                 'flatten_2[0][0]',
                                                                 'flatten_3[0][0]',
                                                                 'flatten_4[0][0]',
                                                                 'flatten_5[0][0]',
                                                                 'flatten_6[0][0]']

dense_1 (Dense)                (None, 128)          4105984     ['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, 2)            66          ['dropout_2[0][0]']

==================================================================================================
Total params: 17,856,122
Trainable params: 4,355,694
Non-trainable params: 13,500,428
__________________________________________________________________________________________________
None
```

```
#model fitting
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
filepath="weights_2.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_auc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]

# Remove TensorBoard from the callbacks_list
model.fit([X_train, X_train_school_state, X_train_project_grade, X_train_clean_categories, X_train_clean_subcategories,
           X_train_teacher_prefix, train['price']], y_train, epochs=50, verbose=1, batch_size=256,
          validation_data=([X_cv, X_cv_school_state, X_cv_project_grade, X_cv_clean_categories, X_cv_clean_subcategories,
                            X_cv_teacher_prefix, cv['price']], y_cv), callbacks=callbacks_list)
```

```
Epoch 00045: val_auc did not improve from 0.76925
239/239 [==============================] - 16s 69ms/step - loss: 0.3201 - accuracy: 0.8691 - auc: 0.8523 - val_loss: 0.3878 - va
Epoch 46/50
239/239 [==============================] - ETA: 0s - loss: 0.3163 - accuracy: 0.8707 - auc: 0.8560
Epoch 00046: val_auc did not improve from 0.76925
239/239 [==============================] - 18s 77ms/step - loss: 0.3163 - accuracy: 0.8707 - auc: 0.8560 - val_loss: 0.3899 - va
Epoch 47/50
239/239 [==============================] - ETA: 0s - loss: 0.3131 - accuracy: 0.8719 - auc: 0.8605
Epoch 00047: val_auc did not improve from 0.76925
239/239 [==============================] - 15s 65ms/step - loss: 0.3131 - accuracy: 0.8719 - auc: 0.8605 - val_loss: 0.3939 - va
Epoch 48/50
239/239 [==============================] - ETA: 0s - loss: 0.3124 - accuracy: 0.8719 - auc: 0.8633
Epoch 00048: val_auc did not improve from 0.76925
239/239 [==============================] - 17s 69ms/step - loss: 0.3124 - accuracy: 0.8719 - auc: 0.8633 - val_loss: 0.4038 - va
Epoch 49/50
239/239 [==============================] - ETA: 0s - loss: 0.3076 - accuracy: 0.8753 - auc: 0.8689
Epoch 00049: val_auc did not improve from 0.76925
239/239 [==============================] - 14s 60ms/step - loss: 0.3076 - accuracy: 0.8753 - auc: 0.8689 - val_loss: 0.3953 - va
Epoch 50/50
238/239 [============================>.] - ETA: 0s - loss: 0.3062 - accuracy: 0.8758 - auc: 0.8714
Epoch 00050: val_auc did not improve from 0.76925
239/239 [==============================] - 14s 57ms/step - loss: 0.3060 - accuracy: 0.8760 - auc: 0.8715 - val_loss: 0.4061 - va
```

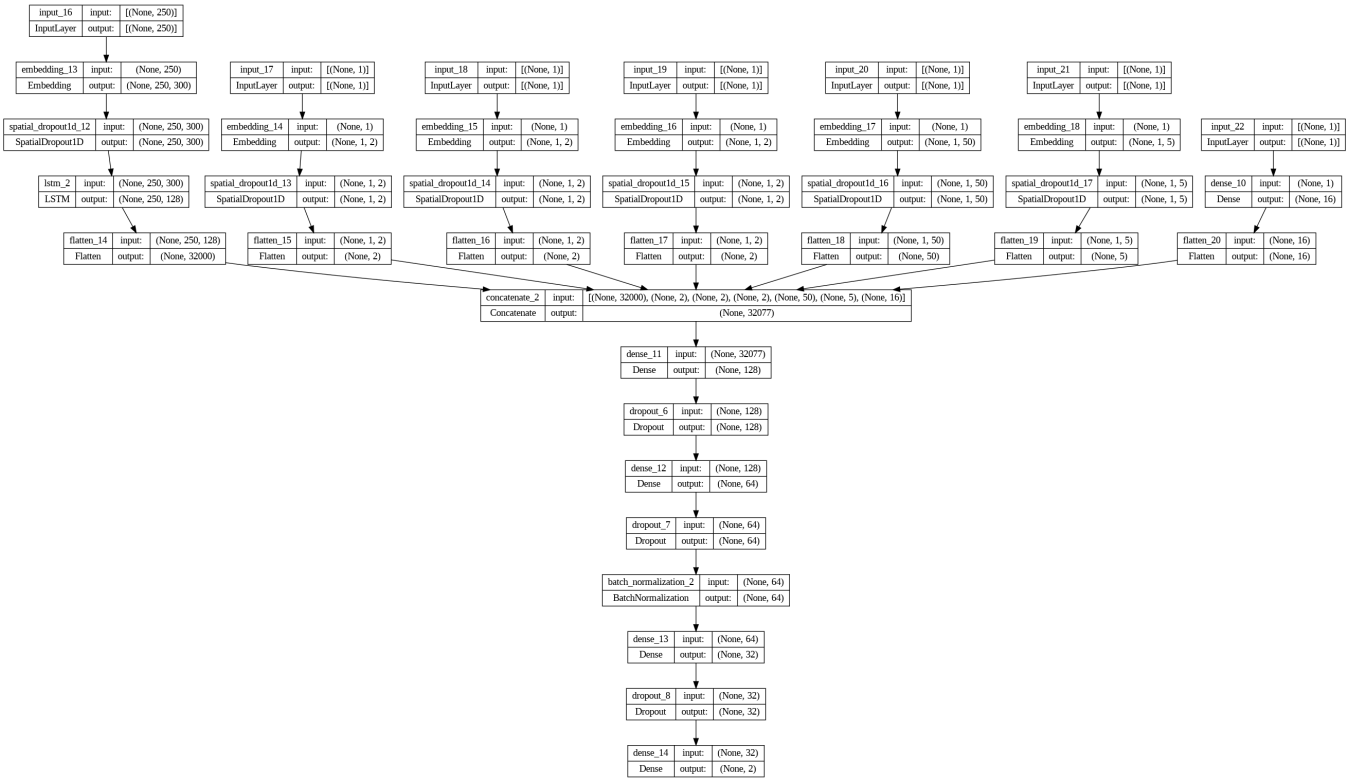
We have got the auc score 87.15% and accuracy 87.60%

```
#input 1
input1 = Input(shape=(250,))
x1 = Embedding(input_dim=45001,output_dim= 300,weights=[embedding_mat(feature_names[5])],trainable=False)(input1)
x1 = SpatialDropout1D(0.3)(x1)
x1 = LSTM(128,return_sequences=True)(x1)
x1 = Flatten()(x1)

#input 2
input2 = Input(shape=(1,))
x2 = Embedding(input_dim= 55,output_dim= 2)(input2)
x2 = SpatialDropout1D(0.3)(x2)
x2 = Flatten()(x2)

#input 3
input3 = Input(shape=(1,))
x3 = Embedding(input_dim= 5,output_dim= 2)(input3)
x3 = SpatialDropout1D(0.3)(x3)
x3 = Flatten()(x3)

#input 4
input4 = Input(shape=(1,))
x4 = Embedding(input_dim=50,output_dim= 2)(input4)
x4 = SpatialDropout1D(0.3)(x4)
x4 = Flatten()(x4)

#input 5
input5 = Input(shape=(1,))
x5 = Embedding(input_dim= 385,output_dim= 50)(input5)
x5 = SpatialDropout1D(0.3)(x5)
x5 = Flatten()(x5)

#input 6
input6 = Input(shape=(1,))
x6 = Embedding(input_dim= 6,output_dim= 5)(input6)
x6 = SpatialDropout1D(0.3)(x6)
x6 = Flatten()(x6)

#input 7
input7 = Input(shape=(1,))
x7 = Dense(16,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(input7)
x7 = Flatten()(x7)
# now will merge the all the input
concatated_layer = concatenate([x1,x2,x3,x4,x5,x6,x7])

x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(concatated_layer)
x = Dropout(0.5)(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.5)(x)
output = Dense(2, activation = 'softmax')(x)

# create model with seven inputs
model = Model([input1,input2,input3,input4,input5,input6,input7], output)
tensorboard = TensorBoard(log_dir='logs/{}'.format(time()))
```

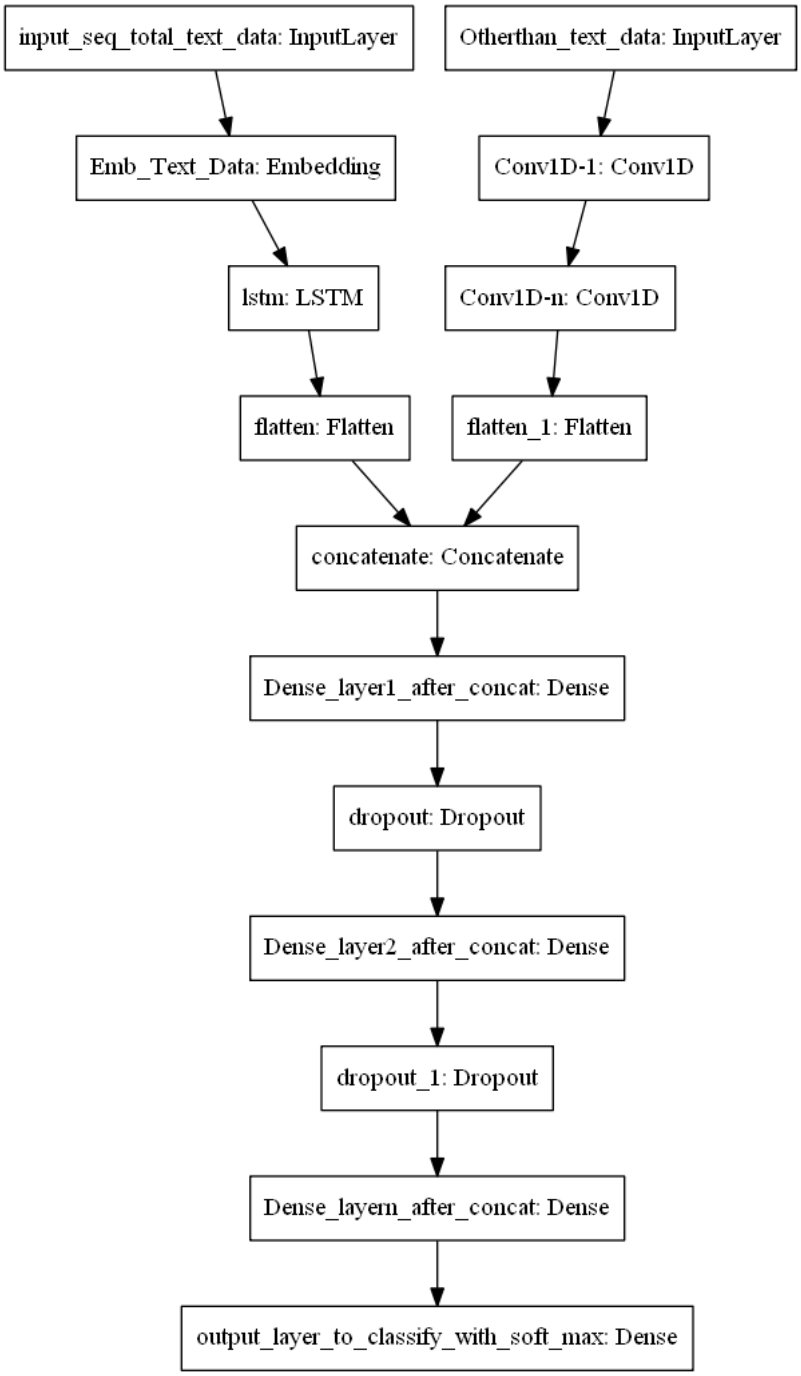
```
model.load_weights("weights_2.best.hdf5")

model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0006,decay = 1e-4),metrics=['accuracy',auc])


print("Auc for test data: %0.3f"%roc_auc_score(y_test,model.predict([X_test,X_test_school_state,X_test_project_grade,X_test_clean_categor
        X_test_teacher_prefix,test['price']])))
print("Auc for CV data: %0.3f"%roc_auc_score(y_cv,model.predict([X_cv,X_cv_school_state,X_cv_project_grade,X_cv_clean_categories,X_cv_cle
          X_cv_teacher_prefix,cv['price']])))
print("Auc for train data: %0.3f"%roc_auc_score(y_train,model.predict([X_train,X_train_school_state,X_train_project_grade,X_train_clean_c
          X_train_teacher_prefix,train['price']])))
```

```
1025/1025 [==============================] - 9s 9ms/step
Auc for test data: 0.763
478/478 [==============================] - 3s 6ms/step
Auc for CV data: 0.770
1912/1912 [==============================] - 13s 7ms/step
Auc for train data: 0.823
```

```
from tensorflow.keras.utils import plot_model



from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```



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.


#Reading the dataset
project_data = pd.read_csv('/content/drive/MyDrive/preprocessed_data.csv')
project_data.head(3)
```

school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_c

```
class_label = project_data['project_is_approved']



project_data.replace(to_replace=np.NaN, value= str('nan'),inplace=True)



col = ['teacher_prefix', 'school_state', 'project_grade_category',
       'clean_categories', 'clean_subcategories','essay',
       'price']
project_data = project_data[col]
```

```
 #ref : https://www.geeksforgeeks.org/frequent-word-array-strings/
def word_ranking(df):
    col_names = df.columns
    features = []
    #performing train test split
    train,test,y_train,y_test = train_test_split(df, class_label , stratify = class_label, train_size = 0.7)

    train,cv,y_train,y_cv = train_test_split(train,y_train,stratify = y_train,train_size = 0.8)
    for coln in col_names[5:6]:
        print(coln)
        bag_of_words = CountVectorizer(lowercase= False)
        bow_words = bag_of_words.fit_transform(train[coln])
        print(bow_words.shape)

        #Lets now store the document term matrix in a dictionary.
        freqs = bow_words.sum(axis=0).A1
        index = freqs.argsort()
        words = bag_of_words.get_feature_names()



        # Assigning Rank to each word based on its freq of occurance. Word with highest freq is assigned rank 1
        word_rank = dict()
        rank = 1
        for i in index[::-1]:
            k = words[i]
            word_rank[k] = rank
            rank+=1
        features.append(word_rank)

        #Every word in each review is replaced by its rank
        rank = [] # list of all the review with words replaced with rank
        for sent in train[coln].values:
            txt_row = []
            for word in sent.split():
                if word in word_rank.keys():
                    txt_row.append(word_rank[word])
                else:
                    pass
            rank.append(txt_row)

        train[coln] = rank

        rank = [] # list of all the review with words replaced with rank
        for sent in test[coln].values:
            txt_row = []
            for word in sent.split():
                if word in word_rank.keys():
                    txt_row.append(word_rank[word])
                else:
                    pass
            rank.append(txt_row)

        test[coln] = rank

        rank = [] # list of all the review with words replaced with rank
        for sent in cv[coln].values:
            txt_row = []
            for word in sent.split():
                if word in word_rank.keys():
                    txt_row.append(word_rank[word])
                else:
                    pass
            rank.append(txt_row)

        cv[coln] = rank
```

```
    return train,test,cv,y_train,y_test,y_cv,features
```

```
train,test,cv,y_train,y_test,y_cv,feature_names = word_ranking(project_data)
```

```
essay
(61178, 44986)
```

```
print("Shape of the Train dataset: ", train.shape[0])
print("Shape of the Test dataset: ", test.shape[0])
print("Shape of the CV dataset: ", cv.shape[0])
```

```
Shape of the Train dataset:  61178
Shape of the Test dataset:  32775
Shape of the CV dataset:  15295
```

```
y_train.shape
```

```
(61178,)
```

```
#converting class labels to categorical variables
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_cv = to_categorical(y_cv)
```

Tokenizing the Text part

```
max_review_length = 250
X_train = pad_sequences(train['essay'], maxlen=max_review_length)  #padding zeros at the begining of each review to make max len as 200
X_test = pad_sequences(test['essay'], maxlen=max_review_length)
X_cv = pad_sequences(cv['essay'], maxlen=max_review_length)
print(X_train.shape)
print(X_train[256])
```

```
(61178, 250)
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0    14  1433   116   294   142   197    13
   186     9   477   436     2    90   135   411     3   104    58     1
    18   116   205     1   103    59  6051    11   703    15   131   277
   116   616   116    43    88   206    15   366  2767    65  1189    13
    14  1433    85    55   116    13    90    41   294    85    37   223
   687     1  1111   116     1    92 17135  2135   465  3114    28  1915
     7 12157  4365  2244   465  2419   197  2135   149   255  7100  1451
   256  3134  8351   149     6     1    55 12035  1451  6096  1451   253
  2041  1682    13    32  2837  3222  6946   256   520  3155   413  4091
  1682    13    32  3411  1316 17619    27    53   128     1  1433   294
   116   255     6  1411    89  4738  1316  1972  1375    12]
```

Tokenizing the school state

```
token_school_state = CountVectorizer()

# integer encode the documents
school_state_train = token_school_state.fit_transform(train['school_state'])
school_state_test = token_school_state.transform(test['school_state'])
school_state_cv = token_school_state.transform(cv['school_state'])

print(school_state_train.shape)
```

```
(61178, 51)
```

Tokenizing the project_grade_category

```
token_project_grade_category = CountVectorizer()

# integer encode the documents
project_grade_train = token_project_grade_category.fit_transform(train['project_grade_category'])
project_grade_test = token_project_grade_category.transform(test['project_grade_category'])
project_grade_cv = token_project_grade_category.transform(cv['project_grade_category'])

print(project_grade_train.shape)
```

```
(61178, 4)
```

Tokenizing the project categories

```
token_clean_categories = CountVectorizer()

# integer encode the documents
train_clean_categories= token_clean_categories.fit_transform(train['clean_categories'])
test_clean_categories = token_clean_categories.transform(test['clean_categories'])
cv_clean_categories = token_clean_categories.transform(cv['clean_categories'])


print(train_clean_categories.shape)
```

```
(61178, 9)
```

Tokenizing the project subcategories

```
token_clean_subcategories = CountVectorizer()

# integer encode the documents
train_clean_subcategories = token_clean_subcategories.fit_transform(train['clean_subcategories'])
test_clean_subcategories = token_clean_subcategories.transform(test['clean_subcategories'])
cv_clean_subcategories = token_clean_subcategories.transform(cv['clean_subcategories'])

print(train_clean_subcategories.shape)
```

```
(61178, 30)
```

Tokenizing the teacher prefix

```
token_teacher_prefix = CountVectorizer()

# integer encode the documents
teacher_prefix_train = token_teacher_prefix.fit_transform(train['teacher_prefix'])
teacher_prefix_test = token_teacher_prefix.transform(test['teacher_prefix'])
teacher_prefix_cv = token_teacher_prefix.transform(cv['teacher_prefix'])
print(teacher_prefix_train.shape)
```

```
(61178, 5)
```

```
# now will stack the data set

from scipy.sparse import  hstack

input2_train = hstack((school_state_train,project_grade_train,train_clean_categories,train_clean_subcategories,teacher_prefix_train,trair

input2_cv = hstack((school_state_cv,project_grade_cv,cv_clean_categories,cv_clean_subcategories,teacher_prefix_cv,cv['price'][:,None]))
input2_test = hstack((school_state_test,project_grade_test,test_clean_categories,test_clean_subcategories,teacher_prefix_test,test['price


input2_train.shape
```

```
(61178, 100)
```

```
 # ref : https://www.geeksforgeeks.org/python-pandas-series-to-dense/
```

```
train=input2_train.todense()
test = input2_test.todense()
cv = input2_cv.todense()
```

```
 # now will do the reshaping

train = np.resize(train,new_shape=(61178,495,1))
test =np.resize(test,new_shape=(32775,495,1))
cv = np.resize(cv,new_shape=(15295,495,1))


train.shape ,test.shape ,cv.shape
```

```
     ((61178, 495, 1), (32775, 495, 1), (15295, 495, 1))
```

Deep Learning Models

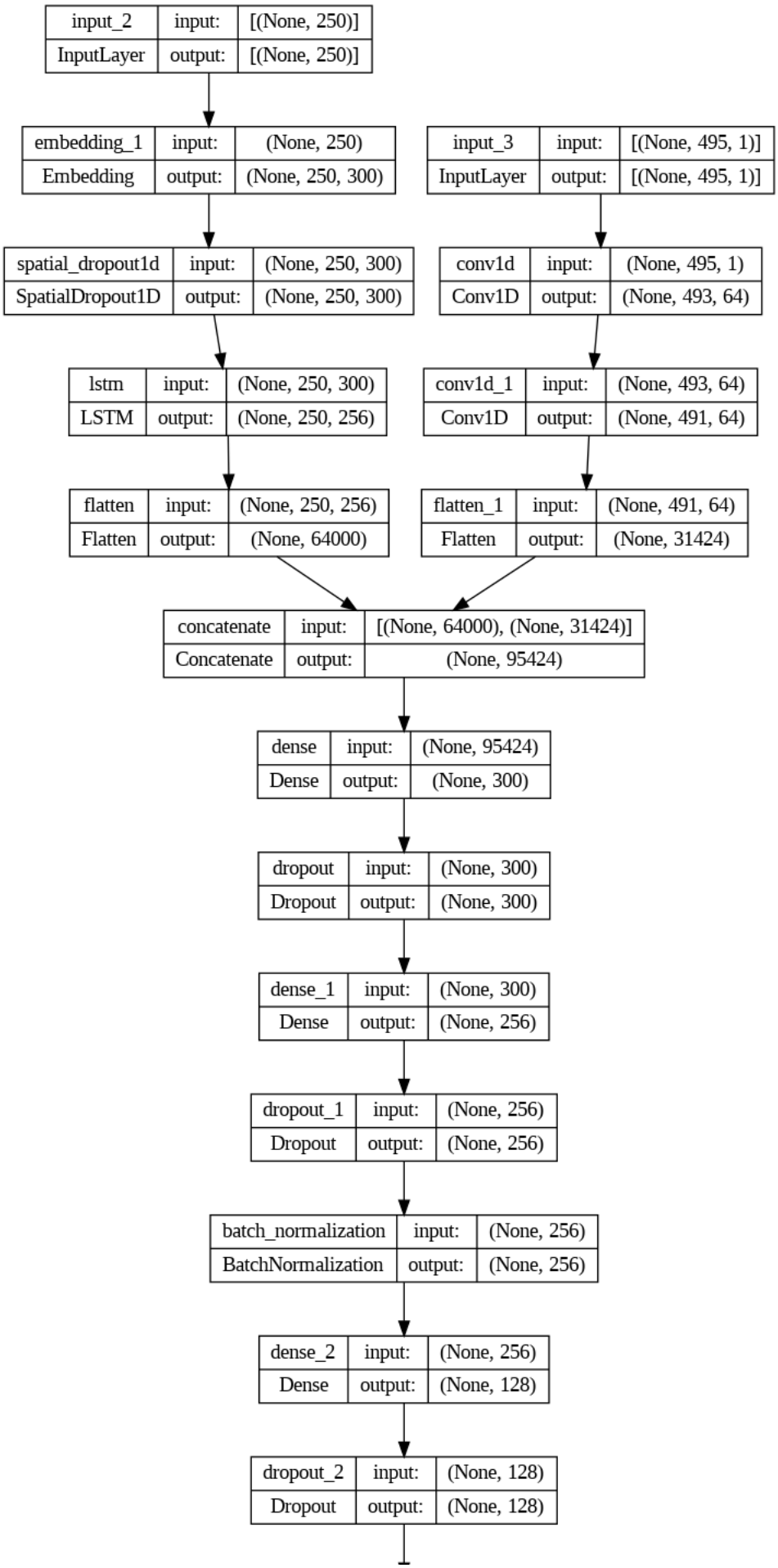
Model 3

```
dumb_file = open('/content/drive/MyDrive/glove_vectors', 'rb')
dumb_file = pickle.load(dumb_file)


#Getting word vectors with 50 dim
def embedding_mat(word_index,embedding_dim = 300):
    embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
    for word, i in word_index.items():
        embedding_vector = dumb_file.get(word)
        if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix


# defining the auc score for model
def auc( y_true, y_pred ) :
    score = tf.numpy_function(lambda y_true, y_pred : roc_auc_score( y_true, y_pred ,average='weighted').astype('float32'),
                        [y_true, y_pred],
                        'float32',
                        name='sklearnAUC')
    return score


# defining the accuracy score for model

def accuracy(y_true, y_pred):
    y_pred = tf.argmax(y_pred, axis=1)
    accuracy = tf.keras.metrics.Accuracy()
    accuracy.update_state(y_true, y_pred)
    return accuracy.result().numpy()



# defining learning rate function with value of 0.0001
def step_decay(epoch):
    initial_lr = 0.0001
    lr_drop = 1e-6
    epochs_drop = 1
    lr = initial_lr * math.pow(lr_drop, math.floor((1 + epoch) / epochs_drop))
    return lr


# refer : https://keras.io/api/layers/recurrent_layers/lstm/
# input 1
input1 = Input(batch_shape=(None,250))
x1 = Embedding(input_dim=44987, output_dim=300, weights=[embedding_mat(feature_names[0])], trainable=False)(input1)
x1 = SpatialDropout1D(0.3)(x1)
x1 = LSTM(256, return_sequences=True)(x1)
x1 = Flatten()(x1)

# input 2
input2 = Input(shape=(495,1))
x2 = Conv1D(filters=64, kernel_size=3, strides=1)(input2)
x2 = Conv1D(filters=64, kernel_size=3, strides=1)(x2)
x2 = Flatten()(x2)

# merging both the inputs
concat = concatenate([x1,x2])
x = Dense(300, activation='relu', kernel_initializer=he_normal(), kernel_regularizer=l2(0.0001))(concat)
x = Dropout(0.4)(x)
```

```
x = Dense(256, activation='relu', kernel_initializer=he_normal(), kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(128, activation='relu', kernel_initializer=he_normal(), kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.6)(x)
output = Dense(2, activation='softmax')(x)

# create model with two inputs
model = Model([input1, input2], output)
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0006, decay=1e-4), metrics=['accuracy' ,auc])

print(model.summary())

# model fitting
filepath = "weights_3.best_copy.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_auc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
class_weights = {0: 1.0, 1: 2.0}  # example class weights
history = model.fit(x=[X_train, train], y=y_train, epochs=25, verbose=1, batch_size=256,
                    validation_data=([X_cv, cv], y_cv), callbacks=callbacks_list, class_weight=class_weights)
```

```
239/239 [==============================] - 32s 132ms/step - loss: 0.6280 - accuracy: 0.8486 - auc: 0.5173 - val_loss: 0.5234 - v
Epoch 12/25
239/239 [==============================] - ETA: 0s - loss: 0.6097 - accuracy: 0.8486 - auc: 0.5132
Epoch 00012: val_auc did not improve from 0.68262
239/239 [==============================] - 32s 132ms/step - loss: 0.6097 - accuracy: 0.8486 - auc: 0.5132 - val_loss: 0.4943 - v
Epoch 13/25
239/239 [==============================] - ETA: 0s - loss: 0.5909 - accuracy: 0.8486 - auc: 0.6204
Epoch 00013: val_auc improved from 0.68262 to 0.71322, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 33s 140ms/step - loss: 0.5909 - accuracy: 0.8486 - auc: 0.6204 - val_loss: 0.5280 - v
Epoch 14/25
239/239 [==============================] - ETA: 0s - loss: 0.5773 - accuracy: 0.8486 - auc: 0.6889
Epoch 00014: val_auc improved from 0.71322 to 0.71985, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 34s 140ms/step - loss: 0.5773 - accuracy: 0.8486 - auc: 0.6889 - val_loss: 0.4934 - v
Epoch 15/25
239/239 [==============================] - ETA: 0s - loss: 0.5705 - accuracy: 0.8486 - auc: 0.6994
Epoch 00015: val_auc improved from 0.71985 to 0.73109, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 33s 139ms/step - loss: 0.5705 - accuracy: 0.8486 - auc: 0.6994 - val_loss: 0.4806 - v
Epoch 16/25
239/239 [==============================] - ETA: 0s - loss: 0.5577 - accuracy: 0.8486 - auc: 0.7111
Epoch 00016: val_auc improved from 0.73109 to 0.74131, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 34s 141ms/step - loss: 0.5577 - accuracy: 0.8486 - auc: 0.7111 - val_loss: 0.4796 - v
Epoch 17/25
239/239 [==============================] - ETA: 0s - loss: 0.5537 - accuracy: 0.8487 - auc: 0.7155
Epoch 00017: val_auc improved from 0.74131 to 0.74482, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 33s 139ms/step - loss: 0.5537 - accuracy: 0.8487 - auc: 0.7155 - val_loss: 0.4768 - v
Epoch 18/25
239/239 [==============================] - ETA: 0s - loss: 0.5430 - accuracy: 0.8489 - auc: 0.7241
Epoch 00018: val_auc improved from 0.74482 to 0.74643, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 33s 140ms/step - loss: 0.5430 - accuracy: 0.8489 - auc: 0.7241 - val_loss: 0.4747 - v
Epoch 19/25
239/239 [==============================] - ETA: 0s - loss: 0.5396 - accuracy: 0.8491 - auc: 0.7289
Epoch 00019: val_auc did not improve from 0.74643
239/239 [==============================] - 32s 134ms/step - loss: 0.5396 - accuracy: 0.8491 - auc: 0.7289 - val_loss: 0.4654 - v
Epoch 20/25
239/239 [==============================] - ETA: 0s - loss: 0.5299 - accuracy: 0.8492 - auc: 0.7336
Epoch 00020: val_auc improved from 0.74643 to 0.75794, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 34s 140ms/step - loss: 0.5299 - accuracy: 0.8492 - auc: 0.7336 - val_loss: 0.4368 - v
Epoch 21/25
239/239 [==============================] - ETA: 0s - loss: 0.5226 - accuracy: 0.8489 - auc: 0.7402
Epoch 00021: val_auc did not improve from 0.75794
239/239 [==============================] - 32s 134ms/step - loss: 0.5226 - accuracy: 0.8489 - auc: 0.7402 - val_loss: 0.4532 - v
Epoch 22/25
239/239 [==============================] - ETA: 0s - loss: 0.5172 - accuracy: 0.8490 - auc: 0.7459
Epoch 00022: val_auc improved from 0.75794 to 0.75799, saving model to weights_3.best_copy.hdf5
239/239 [==============================] - 34s 141ms/step - loss: 0.5172 - accuracy: 0.8490 - auc: 0.7459 - val_loss: 0.4591 - v
Epoch 23/25
239/239 [==============================] - ETA: 0s - loss: 0.5109 - accuracy: 0.8489 - auc: 0.7491
Epoch 00023: val_auc did not improve from 0.75799
239/239 [==============================] - 32s 134ms/step - loss: 0.5109 - accuracy: 0.8489 - auc: 0.7491 - val_loss: 0.4628 - v
Epoch 24/25
239/239 [==============================] - ETA: 0s - loss: 0.5104 - accuracy: 0.8496 - auc: 0.7516
Epoch 00024: val_auc did not improve from 0.75799
239/239 [==============================] - 32s 134ms/step - loss: 0.5104 - accuracy: 0.8496 - auc: 0.7516 - val_loss: 0.4416 - v
Epoch 25/25
239/239 [==============================] - ETA: 0s - loss: 0.5035 - accuracy: 0.8494 - auc: 0.7574
Epoch 00025: val_auc did not improve from 0.75799
239/239 [==============================] - 32s 135ms/step - loss: 0.5035 - accuracy: 0.8494 - auc: 0.7574 - val_loss: 0.4507 - v
```

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```




```
# input 1
input1 = Input(batch_shape=(None,250))
x1 = Embedding(input_dim=44987,output_dim= 300,weights=[embedding_mat(feature_names[0])],trainable = False)(input1)
x1 = SpatialDropout1D(0.3)(x1)
x1 = CuDNNLSTM(256,return_sequences=True)(x1)
x1 = Flatten()(x1)
```

```
# input 2
input2 = Input(shape=(495,1))
x2 = Conv1D(filters=64,kernel_size=3,strides=1)(input2)
x2 = Conv1D(filters=64,kernel_size=3,strides=1)(x2)
x2 = Flatten()(x2)

# merging both the inputs
concat = concatenate([x1,x2])
x = Dense(300,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(concat)
x = Dropout(0.4)(x)
x = Dense(256,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0001))(x)
x = Dropout(0.6)(x)
output = Dense(2, activation = 'softmax')(x)

# create model with two inputs
model = Model([input1,input2], output)
model.load_weights("weights_3.best_copy.hdf5")
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0006,decay = 1e-4), metrics=[auc])


print("Auc for test data: %0.3f"%roc_auc_score(y_test,model.predict([X_test,test])))
print("Auc for CV data: %0.3f"%roc_auc_score(y_cv,model.predict([X_cv,cv])))
print("Auc for train data: %0.3f"%roc_auc_score(y_train,model.predict([X_train,train])))
```

```
1025/1025 [==============================] - 8s 7ms/step
Auc for test data: 0.747
478/478 [==============================] - 4s 9ms/step
Auc for CV data: 0.759
1912/1912 [==============================] - 15s 8ms/step
Auc for train data: 0.776
```

✓ 1s completed at 10:08 AM