Assignment : 14

1. Preprocess all the Data we have in DonorsChoose Dataset use train.csv
2. Combine 4 essay's into one column named - 'preprocessed_essays'.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check 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, resources: cs231n class notes, cs231n class video.
7. For all the model's use TensorBoard and plot the Metric value and Loss with epoch. While submitting, take a screenshot of plots and include those images in .ipynb notebook and PDF.
8. Use Categorical Cross Entropy as Loss to minimize.

Model-2

Build and Train deep neural network as shown below

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.

In [ ]:

In [12]:
```
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_rem
ount=True).
```

In [13]:
```
import numpy as np
import pandas as pd
```

In [14]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [15]:
```
processed_data=pd.read_csv('/content/drive/MyDrive/Applied ai/for_colab/preprocessed_data.csv')
```

In [16]:
```
processed_data.head()
```

Out[16]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_
0	ca	mrs	grades_prek_2	53	1	math_science	a hea
1	ut	ms	grades_3_5	4	1	specialneeds	
2	ca	mrs	grades_prek_2	10	1	literacy_language	
3	ga	mrs	grades_prek_2	2	1	appliedlearning	ea

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_
4	wa	mrs	grades_3_5	2	1	literacy_language	

In [17]:
```
processed_data.shape
```

Out[17]:
```
(109248, 9)
```

In [18]:
```
y = processed_data['project_is_approved']
X = processed_data.drop(['project_is_approved'],axis=1)
```

In [19]:
```
print(X.shape, y.shape)
```

```
(109248, 8) (109248,)
```

In [20]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.22, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.22, stratify=y_train)
```

In [21]:
```
print(X_train.shape, X_test.shape, X_cv.shape)
```

```
(66466, 8) (24035, 8) (18747, 8)
```

In [22]:
```
X_train.head(3)
```

Out[22]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	
40657	or	mr	grades_3_5	0	literacy_language	literature_writing	
86685	ca	mrs	grades_prek_2	4	literacy_language	literature_writing	pa
16583	mi	ms	grades_prek_2	0	literacy_language specialneeds	literature_writing specialneeds	

Essay encoding:

In [23]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vec = TfidfVectorizer()
tfidf_vec.fit(X_train['essay'])
```

Out[23]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=None,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

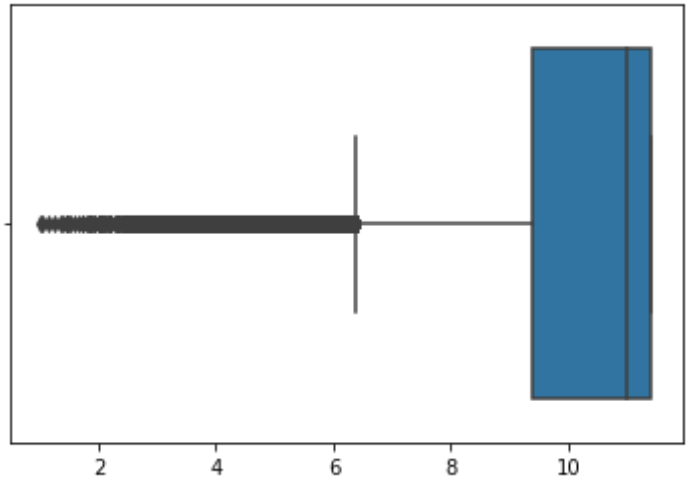
In [24]:
```
tfidf_vec.idf_
```

Out[24]:
```
array([ 7.13464756,  5.94748188, 11.41131368, ..., 11.00584857,
       11.00584857, 11.41131368])
```

In [25]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.boxplot(tfidf_vec.idf_)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a key
word arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6e7be9dd0>
```



In [26]:
```
np.percentile([20, 2, 7, 1, 34], [75,100])
```

Out[26]:
```
array([20., 34.])
```

In [27]:
```
np.percentile(tfidf_vec.idf_,[2,5,10,20,30,40,50,60,70,80,90,95,98,99,100])
```

Out[27]:
```
array([ 4.84886959,  6.13319902,  7.39493066,  8.84636432,  9.80187577,
       10.49502295, 11.00584857, 11.00584857, 11.41131368, 11.41131368,
       11.41131368, 11.41131368, 11.41131368, 11.41131368, 11.41131368])
```

In [28]:
```
feature_names = tfidf_vec.get_feature_names()
```

In [29]:
```
idf_vals = tfidf_vec.idf_
```

In [30]:
```
len(feature_names), len(idf_vals)
```

Out[30]:
```
(46311, 46311)
```

In [31]:
```
cnt=0
for ele in tfidf_vec.idf_:
    if ele<4.8:
        cnt+=1
print(cnt, ' words have less than 4.8 idf value')
```

```
892  words have less than 4.8 idf value
```

In [32]:
```
cnt=0
for ele in tfidf_vec.idf_:
    if ele>11.2:
        cnt+=1
print(cnt,  ' words have greater than 11.2 idf value')
```

```
18035  words have greater than 11.2 idf value
```

In [33]:
```
if 2>=4 or 2<=11:
    print(True)
```

```
True
```

In [34]:
```
final_words = []
for item in dict(zip(feature_names,idf_vals)).items():
    if(item[1]>=4.8 and item[1]<=11.2):
        final_words.append(item[0])
```

In [35]:
```
```

```
len(final_words)
```

Out[35]:
```
27384
```

In [36]:
```
essay_new = []
from tqdm import tqdm
for ele in tqdm(X_train['essay']):
    lst = ele.split(" ")
    string = ""
    for word in lst:
        if word in final_words:
            string = string+" "+word
    essay_new.append(string)
```

```
100%|██████████| 66466/66466 [1:38:56<00:00, 11.20it/s]
```

In [37]:
```
essay_new[:3]
```

Out[37]:
```
[' lost relevant nearly aspects computing informational narratives poetry slideshows movies conferencing presentation
s links webpages documents then reports papers presentations submit savvy keyboarding editing revising papers process
ing',
 ' dual immersion norton aeronautics academy rigors encourages promotes multicultural settings manner implementing pe
rformance negatively impacting alternate yoga yoga mats cushions nurture kidney',
 ' limit implant modalities tactical auditory chart document cameras enormously timer kiddos transition erasers folde
rs welcomed unprepared knowing anxious']
```

In [38]:
```
essay_test = []
from tqdm import tqdm
for ele in tqdm(X_test['essay']):
    lst = ele.split(" ")
    string = ""
    for word in lst:
        if word in final_words:
            string = string+" "+word
    essay_test.append(string)
```

```
100%|██████████| 24035/24035 [35:47<00:00, 11.19it/s]
```

In [39]:
```
essay_cv = []
from tqdm import tqdm
for ele in tqdm(X_cv['essay']):
    lst = ele.split(" ")
    string = ""
    for word in lst:
        if word in final_words:
            string = string+" "+word
    essay_cv.append(string)
```

```
100%|██████████| 18747/18747 [27:51<00:00, 11.22it/s]
```

In [40]:
```
token = tf.keras.preprocessing.text.Tokenizer()
token.fit_on_texts(essay_new)
essay_idf_train =token.texts_to_sequences(essay_new )
essay_idf_test = token.texts_to_sequences(essay_test)
essay_idf_cv = token.texts_to_sequences(essay_cv)
```

In [41]:
```
mx_len=0
for ele in essay_new:
    if len(ele.split())>mx_len:
        mx_len=len(ele.split())
print(mx_len)
```

```
159
```

In [43]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
pad_sequences([[1,2,4],[3,5,4],[1,6],[1,8,9,4,8],[4]],4)
```

Out[43]:
```
array([[0, 1, 2, 4],
       [0, 3, 5, 4],
       [0, 0, 1, 6],
       [8, 9, 4, 8],
       [0, 0, 0, 4]], dtype=int32)
```

In [44]:
```
vocab_size2=len(token.index_word)+1
```

In [45]:
```
padded_idf_essay_train = pad_sequences(essay_idf_train,maxlen=mx_len)
padded_idf_essay_test = pad_sequences(essay_idf_test,maxlen=mx_len)
padded_idf_essay_cv = pad_sequences(essay_idf_cv,maxlen=mx_len)
padded_idf_essay_train[:1]
```

Out[45]:
```
array([[    0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,    51,   189,
          469,   537,  4073,   819,  3326,  1287,  5847,  1654,  6863,
           52,  3945, 11461,  1338,   789,  1051,   574,    52,  2793,
         1353,  2143,  1359,  4428,   574,  1001]], dtype=int32)
```

In [46]:
```
glove_mat2=np.zeros((vocab_size2, 300))
```

In [47]:
```
glove_mat2.shape
```

Out[47]:
```
(27385, 300)
```

In [49]:
```
try:
    import dill as pickle
except ImportError:
    import pickle
with open('/content/drive/MyDrive/Applied ai/for_colab/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [50]:
```
non_glove_words2=[]
for index, word in token.index_word.items():
    if word in glove_words:
        glove_mat2[index]=model[word]


#     vec=model[word]
#     if vec is not None:
#         glove_mat[index]=vec
    else:
        non_glove_words2.append(word)
print('Totally ',len(non_glove_words2),' words are not there in glove')
print(glove_mat2.shape)
```

```
Totally  636  words are not there in glove
(27385, 300)
```

In [51]:
```
tf.keras.backend.clear_session()
```

School_state encoding:

In [52]:
```
schoolstate_train=X_train['school_state'].tolist()
```

In [53]:
```
len(schoolstate_train)
```

Out[53]:
```
66466
```

In [54]:
```
len(set(schoolstate_train))
```

Out[54]:
```
51
```

In [55]:
```
schoolstate_len=len(set(schoolstate_train))
```

In [56]:

```
#X_train['school_state'].value_counts()
```

In [57]:
```
from tensorflow.keras.preprocessing.text import one_hot
ohe_school_state_train=[one_hot(state, schoolstate_len) for state in X_train['school_state'].tolist()]
ohe_school_state_test=[one_hot(state, schoolstate_len) for state in X_test['school_state'].tolist()]
ohe_school_state_cv=[one_hot(state, schoolstate_len) for state in X_cv['school_state'].tolist()]
```

In [58]:
```
ohe_school_state_train[5]
```

Out[58]:
```
[38]
```

In [59]:
```
len(ohe_school_state_train[5])
```

Out[59]:
```
1
```

Project_grade:

In [60]:
```
prjectgrade_len=len(set(X_train['project_grade_category'].tolist()))
ohe_project_grade_train=[one_hot(project_grade, prjectgrade_len, filters = '') for project_grade in X_train['project_
ohe_project_grade_test=[one_hot(project_grade, prjectgrade_len, filters = '') for project_grade in X_test['project_gr
ohe_project_grade_cv=[one_hot(project_grade, prjectgrade_len, filters = '') for project_grade in X_cv['project_grade_
```

In [61]:
```
ohe_project_grade_train[:5]
```

Out[61]:
```
[[3], [3], [3], [1], [3]]
```

Clean_categories:

In [62]:
```
cleancate_lst=X_train['clean_categories'].tolist()
cleancate_join=' '.join(cleancate_lst)
cleancate_len=len(set(cleancate_join.split()))
```

```
clean_cate_train=[one_hot(clean_cate, cleancate_len, filters = ' ') for clean_cate in X_train['clean_categories'].tol
clean_cate_test=[one_hot(clean_cate, cleancate_len, filters = ' ') for clean_cate in X_test['clean_categories'].tolis
clean_cate_cv=[one_hot(clean_cate, cleancate_len, filters = ' ') for clean_cate in X_cv['clean_categories'].tolist()]
```

In [63]:
```
cleancate_len
```

Out[63]:
```
9
```

In [64]:
```
len(clean_cate_train)
```

Out[64]:
```
66466
```

In [65]:
```
len(clean_cate_train[200])
```

Out[65]:
```
2
```

In [66]:
```
cleancate_mxln=0
val=''
for ele in clean_cate_train:
    if len(ele)>cleancate_mxln:
        val=ele
        cleancate_mxln=len(ele)
```

In [67]:
```
cleancate_mxln
```

Out[67]:
```
3
```

In [68]:
```
val
```

Out[68]:
```
[5, 4, 7]
```

In [69]:
```
padded_cleancate_train = pad_sequences(clean_cate_train, maxlen=cleancate_mxln)
```

```
padded_cleancate_test = pad_sequences(clean_cate_test, maxlen=cleancate_mxln)
padded_cleancate_cv = pad_sequences(clean_cate_cv, maxlen=cleancate_mxln)
```

In [70]:
```
padded_cleancate_train[:5]
```

Out[70]:
```
array([[0, 0, 2],
       [0, 0, 2],
       [0, 2, 7],
       [0, 0, 2],
       [0, 0, 7]], dtype=int32)
```

Clean_subcategory:

In [71]:
```
# cleansubcate_len=len(set(X_train['clean_subcategories'].tolist()))
cleansubcate_lst=X_train['clean_subcategories'].tolist()
cleansubcate_join=' '.join(cleansubcate_lst)
cleansubcate_len=len(set(cleansubcate_join.split()))
clean_subcate_train=[one_hot(clean_subcate, cleansubcate_len) for clean_subcate in X_train['clean_subcategories'].tol
clean_subcate_test=[one_hot(clean_subcate, cleansubcate_len) for clean_subcate in X_test['clean_subcategories'].tolis
clean_subcate_cv=[one_hot(clean_subcate, cleansubcate_len) for clean_subcate in X_cv['clean_subcategories'].tolist()]
```

In [72]:
```
clean_subcate_train[:5]
```

Out[72]:
```
[[28, 8], [28, 8], [28, 8, 3], [11], [3]]
```

In [73]:
```
cleansubcate_len
```

Out[73]:
```
30
```

In [74]:
```
cleansubcate_mxln=0
val2=''
for ele in clean_subcate_train:
    if len(ele)>cleansubcate_mxln:
        val2=ele
        cleansubcate_mxln=len(ele)
```

In [75]:
```
cleansubcate_mxln
```

Out[75]:
```
5
```

In [76]:
```
val2
```

Out[76]:
```
[2, 29, 12, 10, 25]
```

In [77]:
```
padded_cleansubcate_train = pad_sequences(clean_subcate_train, maxlen=cleansubcate_mxln)
padded_cleansubcate_test = pad_sequences(clean_subcate_test, maxlen=cleansubcate_mxln)
padded_cleansubcate_cv = pad_sequences(clean_subcate_cv, maxlen=cleansubcate_mxln)
```

In [78]:
```
padded_cleansubcate_train[:5]
```

Out[78]:
```
array([[ 0,  0,  0, 28,  8],
       [ 0,  0,  0, 28,  8],
       [ 0,  0, 28,  8,  3],
       [ 0,  0,  0,  0, 11],
       [ 0,  0,  0,  0,  3]], dtype=int32)
```

Teacher_prefix:

In [79]:
```
X_train['teacher_prefix'].value_counts()
```

Out[79]:
```
mrs        34773
ms         23765
mr          6514
teacher     1403
dr            11
Name: teacher_prefix, dtype: int64
```

In [80]:
```
#X_train['teacher_prefix'].tolist()
```

In [81]:
```
X_train['teacher_prefix'].values
```

Out[81]:
```
array(['mr', 'mrs', 'ms', ..., 'mrs', 'mrs', 'mrs'], dtype=object)
```

In [82]:
```
teacher_prefix_len=len(set(X_train['teacher_prefix'].tolist()))
teacher_prefix_train=[one_hot(prefix, teacher_prefix_len) for prefix in X_train['teacher_prefix'].tolist()]
teacher_prefix_test=[one_hot(prefix, teacher_prefix_len) for prefix in X_test['teacher_prefix'].tolist()]
teacher_prefix_cv=[one_hot(prefix, teacher_prefix_len) for prefix in X_cv['teacher_prefix'].tolist()]
```

In [83]:
```
#teacher_prefix_len
```

Numerical_features:

In [84]:
```
remain_numeric_train = X_train[['teacher_number_of_previously_posted_projects', 'price']].values
remain_numeric_test = X_test[['teacher_number_of_previously_posted_projects', 'price']].values
remain_numeric_cv = X_cv[['teacher_number_of_previously_posted_projects', 'price']].values
```

In [84]:
```

```

In [85]:
```
from tensorflow.keras.layers import Input, Embedding, LSTM, Concatenate, Dense,Dropout, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

In [ ]:
```

```

Model2

In [86]:
```
def lr_update(epoch,lr):
    if epoch%5 == 0 and lr>1e-4:
        return lr - (0.1*lr)
```

```
    else:
        return lr

from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import TerminateOnNaN
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.callbacks import TensorBoard
import datetime

early_callback = EarlyStopping(monitor="val_auc",patience=10,mode='auto')
path="model2.hdf5"
log_dir="logs2/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True,write_grads=True)
model_callback = ModelCheckpoint(filepath=path, monitor='val_auc',  verbose=1, save_best_only=True, mode='max')
terminate_callback = TerminateOnNaN()
learning_rate_callback = LearningRateScheduler(lr_update,verbose=1)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

In [87]:

```
from sklearn.metrics import roc_auc_score
def auc(y_true,y_pred):
    return tf.py_function(roc_auc_score,(y_true,y_pred),tf.double)
```

In [88]:

```
essay_input=Input(shape=(mx_len,))
essay_embed = Embedding(vocab_size2, 300, input_length=mx_len,weights=[glove_mat2],trainable = False)(essay_input)
essay_lstm=LSTM(15)(essay_embed)
flatten_lstm = Flatten(data_format='channels_last',name='flatten')(essay_lstm)

input_school_state = Input(shape=(1,))
emb_school_state = Embedding(schoolstate_len, 3, input_length=1)(input_school_state)
flatten_school_state = Flatten(data_format='channels_last',name='flatten_school_state')(emb_school_state)

input_proj_grade = Input(shape=(1,))
emb_proj_grade = Embedding(prjectgrade_len, 3, input_length=1)(input_proj_grade)
flatten_proj_grade = Flatten(data_format='channels_last',name='flatten_proj_grade')(emb_proj_grade)

input_clean_cate = Input(shape=(cleancate_mxln,))
emb_clean_cate = Embedding(cleancate_len, 5, input_length=cleancate_mxln)(input_clean_cate)
```

```
flatten_clean_cate = Flatten(data_format='channels_last',name='flatten_clean_cate')(emb_clean_cate)

input_clean_subcate = Input(shape=(cleansubcate_mxln,))
emb_clean_subcate = Embedding(cleansubcate_len, 5, input_length=cleansubcate_mxln)(input_clean_subcate)
flatten_clean_subcate = Flatten(data_format='channels_last',name='flatten_clean_subcate')(emb_clean_subcate)

input_teacher_prefix = Input(shape=(1,))
emb_teacher_prefix = Embedding(teacher_prefix_len, 3, input_length=1)(input_teacher_prefix)
flatten_teacher_prefix = Flatten(data_format='channels_last',name='flatten_teacher_prefix')(emb_teacher_prefix)

input_numeric = Input(shape=(2,))
numeric_layer = Dense(16, activation='relu')(input_numeric)


conc = Concatenate(axis=1)([flatten_lstm,flatten_school_state,flatten_proj_grade,flatten_clean_cate,flatten_clean_sub

#dense1
FC1 = Dense(512,activation='relu',kernel_initializer='he_normal')(conc)

#dropout1
drop1 = Dropout(0.4)(FC1)

#dense2
FC2 = Dense(256,activation='relu',kernel_initializer='he_normal')(drop1)

#dropout2
drop2 = Dropout(0.35)(FC2)

#dense3
FC3 =Dense(64,activation='relu',kernel_initializer='he_normal')(drop2)

#output
output = Dense(1,activation='sigmoid',kernel_initializer='glorot_normal')(FC3)
```

In [89]:

```
model2= Model(inputs=[essay_input, input_school_state,input_proj_grade,input_clean_cate,input_clean_subcate,input_tea
```

In [90]:

```
model2.summary()
```

```
Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 159)]	0	
embedding (Embedding)	(None, 159, 300)	8215500	input_1[0][0]
input_2 (InputLayer)	[(None, 1)]	0	
input_3 (InputLayer)	[(None, 1)]	0	
input_4 (InputLayer)	[(None, 3)]	0	
input_5 (InputLayer)	[(None, 5)]	0	
input_6 (InputLayer)	[(None, 1)]	0	
lstm (LSTM)	(None, 15)	18960	embedding[0][0]
embedding_1 (Embedding)	(None, 1, 3)	153	input_2[0][0]
embedding_2 (Embedding)	(None, 1, 3)	12	input_3[0][0]
embedding_3 (Embedding)	(None, 3, 5)	45	input_4[0][0]
embedding_4 (Embedding)	(None, 5, 5)	150	input_5[0][0]
embedding_5 (Embedding)	(None, 1, 3)	15	input_6[0][0]
input_7 (InputLayer)	[(None, 2)]	0	
flatten (Flatten)	(None, 15)	0	lstm[0][0]
flatten_school_state (Flatten)	(None, 3)	0	embedding_1[0][0]
flatten_proj_grade (Flatten)	(None, 3)	0	embedding_2[0][0]
flatten_clean_cate (Flatten)	(None, 15)	0	embedding_3[0][0]
flatten_clean_subcate (Flatten)	(None, 25)	0	embedding_4[0][0]
flatten_teacher_prefix (Flatten	(None, 3)	0	embedding_5[0][0]

```
dense (Dense)                   (None, 16)           48          input_7[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 80)           0           flatten[0][0]
                                                                 flatten_school_state[0][0]
                                                                 flatten_proj_grade[0][0]
                                                                 flatten_clean_cate[0][0]
                                                                 flatten_clean_subcate[0][0]
                                                                 flatten_teacher_prefix[0][0]
                                                                 dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 512)          41472       concatenate[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 512)          0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          131328      dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 64)           16448       dropout_1[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1)            65          dense_3[0][0]
==================================================================================================
Total params: 8,424,196
Trainable params: 208,696
Non-trainable params: 8,215,500
__________________________________________________________________________________________________
```

In [92]:

```
from tensorflow.keras.utils import plot_model
#https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
plot_model(model2, to_file='model2_plot.png', show_shapes=True, show_layer_names=True)
```

Out[92]:

In [91]:
```
model2.compile(optimizer=Adam(learning_rate=0.001),loss='binary_crossentropy',metrics=['accuracy',auc])
```

In [93]:
```
model2.fit([padded_idf_essay_train,np.array(ohe_school_state_train),np.array(ohe_project_grade_train),padded_cleancat
```

```
Epoch 1/20

Epoch 00001: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 102s 95ms/step - loss: 1.0390 - accuracy: 0.8172 - auc: 0.5094 - val_los
s: 0.4307 - val_accuracy: 0.8486 - val_auc: 0.6135

Epoch 00001: val_auc improved from -inf to 0.61352, saving model to model2.hdf5
Epoch 2/20

Epoch 00002: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 106s 102ms/step - loss: 0.4251 - accuracy: 0.8478 - auc: 0.6180 - val_lo
ss: 0.4025 - val_accuracy: 0.8486 - val_auc: 0.6838
```

```
Epoch 00002: val_auc improved from 0.61352 to 0.68380, saving model to model2.hdf5
Epoch 3/20

Epoch 00003: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 106s 102ms/step - loss: 0.4052 - accuracy: 0.8484 - auc: 0.6734 - val_lo
ss: 0.3946 - val_accuracy: 0.8486 - val_auc: 0.6983

Epoch 00003: val_auc improved from 0.68380 to 0.69831, saving model to model2.hdf5
Epoch 4/20

Epoch 00004: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 107s 103ms/step - loss: 0.3984 - accuracy: 0.8484 - auc: 0.6949 - val_lo
ss: 0.3941 - val_accuracy: 0.8486 - val_auc: 0.6992

Epoch 00004: val_auc improved from 0.69831 to 0.69920, saving model to model2.hdf5
Epoch 5/20

Epoch 00005: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 110s 105ms/step - loss: 0.3917 - accuracy: 0.8485 - auc: 0.7109 - val_lo
ss: 0.3914 - val_accuracy: 0.8486 - val_auc: 0.7041

Epoch 00005: val_auc improved from 0.69920 to 0.70412, saving model to model2.hdf5
Epoch 6/20

Epoch 00006: LearningRateScheduler setting learning rate to 0.0008100000384729355.
1039/1039 [==============================] - 118s 114ms/step - loss: 0.3875 - accuracy: 0.8485 - auc: 0.7224 - val_lo
ss: 0.3924 - val_accuracy: 0.8486 - val_auc: 0.7063

Epoch 00006: val_auc improved from 0.70412 to 0.70634, saving model to model2.hdf5
Epoch 7/20

Epoch 00007: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 110s 106ms/step - loss: 0.3809 - accuracy: 0.8485 - auc: 0.7380 - val_lo
ss: 0.3912 - val_accuracy: 0.8486 - val_auc: 0.7050

Epoch 00007: val_auc did not improve from 0.70634
Epoch 8/20

Epoch 00008: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 125s 120ms/step - loss: 0.3736 - accuracy: 0.8485 - auc: 0.7507 - val_lo
ss: 0.3934 - val_accuracy: 0.8486 - val_auc: 0.7028
```

```
Epoch 00008: val_auc did not improve from 0.70634
Epoch 9/20

Epoch 00009: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 125s 120ms/step - loss: 0.3690 - accuracy: 0.8486 - auc: 0.7612 - val_lo
ss: 0.3951 - val_accuracy: 0.8486 - val_auc: 0.7019

Epoch 00009: val_auc did not improve from 0.70634
Epoch 10/20

Epoch 00010: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 120s 116ms/step - loss: 0.3627 - accuracy: 0.8485 - auc: 0.7705 - val_lo
ss: 0.4023 - val_accuracy: 0.8486 - val_auc: 0.7016

Epoch 00010: val_auc did not improve from 0.70634
Epoch 11/20

Epoch 00011: LearningRateScheduler setting learning rate to 0.0007290000503417104.
1039/1039 [==============================] - 127s 122ms/step - loss: 0.3554 - accuracy: 0.8484 - auc: 0.7856 - val_lo
ss: 0.3980 - val_accuracy: 0.8486 - val_auc: 0.6940

Epoch 00011: val_auc did not improve from 0.70634
```

Out[93]:
```
<keras.callbacks.History at 0x7fa6e1f341d0>
```

In [94]:
```
%load_ext tensorboard
```

In [95]:
```
%tensorboard --logdir logs2/fit/
```

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

Model started performing better on validation data, in the initial epochs, but later it started overfitting after 7th epoch

But even without low and high idf words from essay data, the model started predicting well

In [ ]:
```

```