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.
```

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 [ ]:

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

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

In [ ]:

```python
import numpy as np
import pandas as pd
```

In [ ]:

```python
import tensorflow as tf
```

```python
from tensorflow import keras
from tensorflow.keras import layers
```

In [ ]:

```python
processed_data=pd.read_csv('/content/drive/MyDrive/Applied ai/for_colab/preprocessed_data.csv')
```

In [ ]:

```python
processed_data.head()
```

Out[ ]:

	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
4	wa	mrs	grades_3_5	2	1	literacy_language	

In [ ]:

```
processed_data.shape
```

Out[ ]:

```
(109248, 9)
```

In [ ]:

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

In [ ]:

```
print(X.shape, y.shape)
```

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

In [ ]:

```
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 [ ]:

```
print(X_train.shape, X_test.shape, X_cv.shape)
```

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

In [ ]:

```
X_train.head(3)
```

Out[ ]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories
14367	pa	mrs	grades_prek_2	13	appliedlearning health_sports	earlydevelopment health_wellness
108613	or	ms	grades_9_12	1	literacy_language	literacy

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories
104542	dc	mrs	grades_3_5	5	music_arts	music

Essay encoding:

In [ ]:
```
essay_encode_train=X_train['essay'].tolist()
essay_encode_test=X_test['essay'].tolist()
essay_encode_cv=X_cv['essay'].tolist()
```

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

Out[ ]:
```
18747
```

In [ ]:
```
essay_encode_train[0]
```

Out[ ]:
```
'children walk classroom hopes learn instead find cracked chairs small i donor choose project chairs years ago worked
great however years several chairs broken chairs decorated student art work i would like replace get set larger chair
s growing students the kindergarteners mckees rocks pa come school challenged beyond 5 year olds experience here chil
dren come kindergarten unprepared hungry tired cold nervous our school struggled meet children basic necessities prov
iding free breakfasts lunches winter coats book bags school supplies help complete homework please help make children
feel safer comfortable help provide students solid chair kindergarteners love learn new things however school afford
limited amount paper supplies forces teachers become resourceful i asked something simple chairs years ago i gracious
ly received donation classroom chairs however years several broken scribbled bottoms scratch floors unfortunately cha
irs small please help provide safe chairs every child classroom coming school pleasant experience while parent worry
bullies learning security not worry child chair help make children feel safe secure learn donating donor choose thank
generous consideration support school teachers importantly our children nannan'
```

In [ ]:
```
token=tf.keras.preprocessing.text.Tokenizer()
```

```
In [ ]: token.fit_on_texts(essay_encode_train)
```

```
In [ ]: type(token.word_index)
```

```
Out[ ]: dict
```

```
In [ ]: #token.word_index
```

```
In [ ]: #token.index_word
```

```
In [ ]: encoded_essay_train=token.texts_to_sequences(essay_encode_train)
        encoded_essay_test=token.texts_to_sequences(essay_encode_test)
        encoded_essay_cv=token.texts_to_sequences(essay_encode_cv)
```

```
In [ ]: vocab_size=len(token.word_index)+1
```

```
In [ ]: vocab_size
```

```
Out[ ]: 46331
```

```
In [ ]: len(token.index_word), len(token.word_index)
```

```
Out[ ]: (46330, 46330)
```

```
In [ ]: glove_mat=np.zeros((vocab_size, 300))
```

```
In [ ]: glove_mat
```

```
Out[ ]: 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.]])
```

In [ ]:
```
glove_mat.shape
```

Out[ ]:
```
(46331, 300)
```

In [ ]:
```
#encoded_essay_train[:5]
```

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

Out[ ]:
```
66466
```

In [ ]:
```
mx_len=0
for ele in encoded_essay_train:
    if len(ele)>mx_len:
        mx_len=len(ele)
print(mx_len)
```

```
331
```

In [ ]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
essay_padded_train=pad_sequences(encoded_essay_train, maxlen=mx_len, padding='pre')
essay_padded_test=pad_sequences(encoded_essay_test, maxlen=mx_len, padding='pre')
essay_padded_cv=pad_sequences(encoded_essay_cv, maxlen=mx_len, padding='pre')
```

In [ ]:
```
len(essay_padded_train), len(essay_padded_test), len(essay_padded_cv)
```

Out[ ]:
```
(66466, 24035, 18747)
```

```
In [ ]: essay_padded_train[:1]
```

```
Out[ ]: 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,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,    47,   395,     6,  1961,    10,   602,
              185,  6565,   174,    82,     2,  2638,   295,    52,   174,
              157,  1480,  1221,    91,   196,   157,   291,   174,  1163,
              174,  7172,    36,   126,    16,     2,    24,    53,  1554,
               40,   189,  1296,   174,   446,     1,     7,  1020, 29750,
             3109,  5049,    22,     3,   886,   458,   315,    30,   836,
               99,  2558,    47,    22,   142,  5008,   863,  2418,  1873,
             3330,    25,     3,  2916,   204,    47,   271,  1816,   274,
               63,  5557,   946,  2024,  3654,   117,  1002,     3,    86,
               11,   273,  1022,   372,    11,    33,    47,   137,  3898,
              225,    11,    46,     1,  2115,   417,  1020,    19,    10,
               31,   111,   196,     3,   715,   223,   643,   261,    86,
             3124,   168,   101,  3914,     2,   543,   257,   589,   174,
              157,  1480,     2,  8030,  1260,   414,     6,   174,   196,
              157,   291,  1163, 21213,  7516,  2971,  4960,   520,   174,
               82,   372,    11,    46,   200,   174,    44,   188,     6,
              359,     3,  5818,    99,   457,   499,  1424,  9988,     5,
             3667,     8,  1424,   188,   417,    11,    33,    47,   137,
              200,  2082,    10,   826,  2638,   295,   389,   870,  2068,
              109,     3,   168,  1197,    25,    47,    13]], dtype=int32)
```

```
In [ ]: len(essay_padded_train[5])
```

```
Out[ ]: 331
```

```
In [ ]: len(essay_padded_test[5])
```

```
Out[ ]: 331
```

```
In [ ]: #token.index_word.items()
```

```
In [ ]: try:
            import dill as pickle
        except ImportError:
            import pickle
```

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

```
In [ ]: #glove_words
```

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

            else:
                non_glove_words.append(word)
        print('Totally ',len(non_glove_words),' words are not there in glove')
```

```
Totally  4433  words are not there in glove
```

```
In [ ]: #non_glove_words
```

In [ ]:
```
glove_mat
```

Out[ ]:
```
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,  0.        ,
         0.        ],
       [ 0.15243   , -0.16945   , -0.022748  , ...,  0.61801   ,  0.41281   ,
         0.0010077],
       [-0.043504  , -0.18484   , -0.14613   , ...,  0.1008    ,  0.1068    ,
         0.089065 ],
       ...,
       [ 0.02275   , -0.027457  , -0.17443   , ..., -0.16559   , -0.010907  ,
         0.25632   ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,  0.        ,
         0.        ],
       [ 0.51299   , -0.24741   , -0.0097454, ..., -0.18383   , -0.54002   ,
         0.12206   ]])
```

In [ ]:
```
glove_mat.shape
```

Out[ ]:
```
(46331, 300)
```

School_state encoding:

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
school_state_train=enc.fit_transform(np.array(X_train['school_state']).reshape(-1,1))
school_state_test=enc.transform(np.array(X_test['school_state']).reshape(-1,1))
school_state_cv=enc.transform(np.array(X_cv['school_state']).reshape(-1,1))
```

In [ ]:
```
school_state_train.shape, school_state_test.shape, school_state_cv.shape
```

Out[ ]:
```
((66466, 51), (24035, 51), (18747, 51))
```

In [ ]:
```
school_state_train[5]
```

Out[ ]:
```
<1x51 sparse matrix of type '<class 'numpy.float64'>'
	with 1 stored elements in Compressed Sparse Row format>
```

Project_grade:

In [ ]:
```
X_train['project_grade_category'].value_counts()
```

Out[ ]:
```
grades_prek_2    26866
grades_3_5       22601
grades_6_8       10372
grades_9_12       6627
Name: project_grade_category, dtype: int64
```

In [ ]:
```
enc = OneHotEncoder(handle_unknown='ignore')
project_grad_train=enc.fit_transform(np.array(X_train['project_grade_category']).reshape(-1,1))
project_grad_test=enc.transform(np.array(X_test['project_grade_category']).reshape(-1,1))
project_grad_cv=enc.transform(np.array(X_cv['project_grade_category']).reshape(-1,1))
```

In [ ]:
```
project_grad_train.shape, project_grad_test.shape, project_grad_cv.shape
```

Out[ ]:
```
((66466, 4), (24035, 4), (18747, 4))
```

Clean_categories:

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer()
clean_cate_train=vec.fit_transform(X_train['clean_categories'])
clean_cate_test=vec.transform(X_test['clean_categories'])
clean_cate_cv=vec.transform(X_cv['clean_categories'])
```

In [ ]:
```
clean_cate_train.shape, clean_cate_test.shape, clean_cate_cv.shape
```

Out[ ]:
```
((66466, 9), (24035, 9), (18747, 9))
```

Clean_subcategory:

In [ ]:
```
vec = CountVectorizer()
clean_subcate_train=vec.fit_transform(X_train['clean_subcategories'])
clean_subcate_test=vec.transform(X_test['clean_subcategories'])
clean_subcate_cv=vec.transform(X_cv['clean_subcategories'])
```

In [ ]:
```
clean_subcate_train.shape, clean_subcate_test.shape, clean_subcate_cv.shape
```

Out[ ]:
```
((66466, 30), (24035, 30), (18747, 30))
```

Teacher_prefix:

In [ ]:
```
vec = CountVectorizer()
teacher_prefix_train=vec.fit_transform(X_train['teacher_prefix'])
teacher_prefix_test=vec.transform(X_test['teacher_prefix'])
teacher_prefix_cv=vec.transform(X_cv['teacher_prefix'])
```

In [ ]:
```
teacher_prefix_train.shape, teacher_prefix_test.shape, teacher_prefix_cv.shape
```

Out[ ]:
```
((66466, 5), (24035, 5), (18747, 5))
```

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

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

Out[ ]:
```
mrs         34753
ms          23839
mr           6400
teacher      1468
dr              6
Name: teacher_prefix, dtype: int64
```

Numerical_features:

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
numerical_train=scaler.fit_transform(X_train[['teacher_number_of_previously_posted_projects', 'price']])
numerical_test=scaler.transform(X_test[['teacher_number_of_previously_posted_projects', 'price']])
numerical_cv=scaler.transform(X_cv[['teacher_number_of_previously_posted_projects', 'price']])
```

In [ ]:
```
numerical_train.shape, numerical_test.shape, numerical_cv.shape
```

Out[ ]:
```
((66466, 2), (24035, 2), (18747, 2))
```

In [ ]:
```
numerical_train.mean()
```

Out[ ]:
```
5.2863626942101154e-17
```

In [ ]:
```
numerical_train[5]
```

Out[ ]:
```
array([-0.40346562, -0.26479224])
```

In [ ]:
```
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 [ ]:
```
essay_padded_train.shape
```

Out[ ]:
```
(66466, 331)
```

In [ ]:
```
vocab_size
```

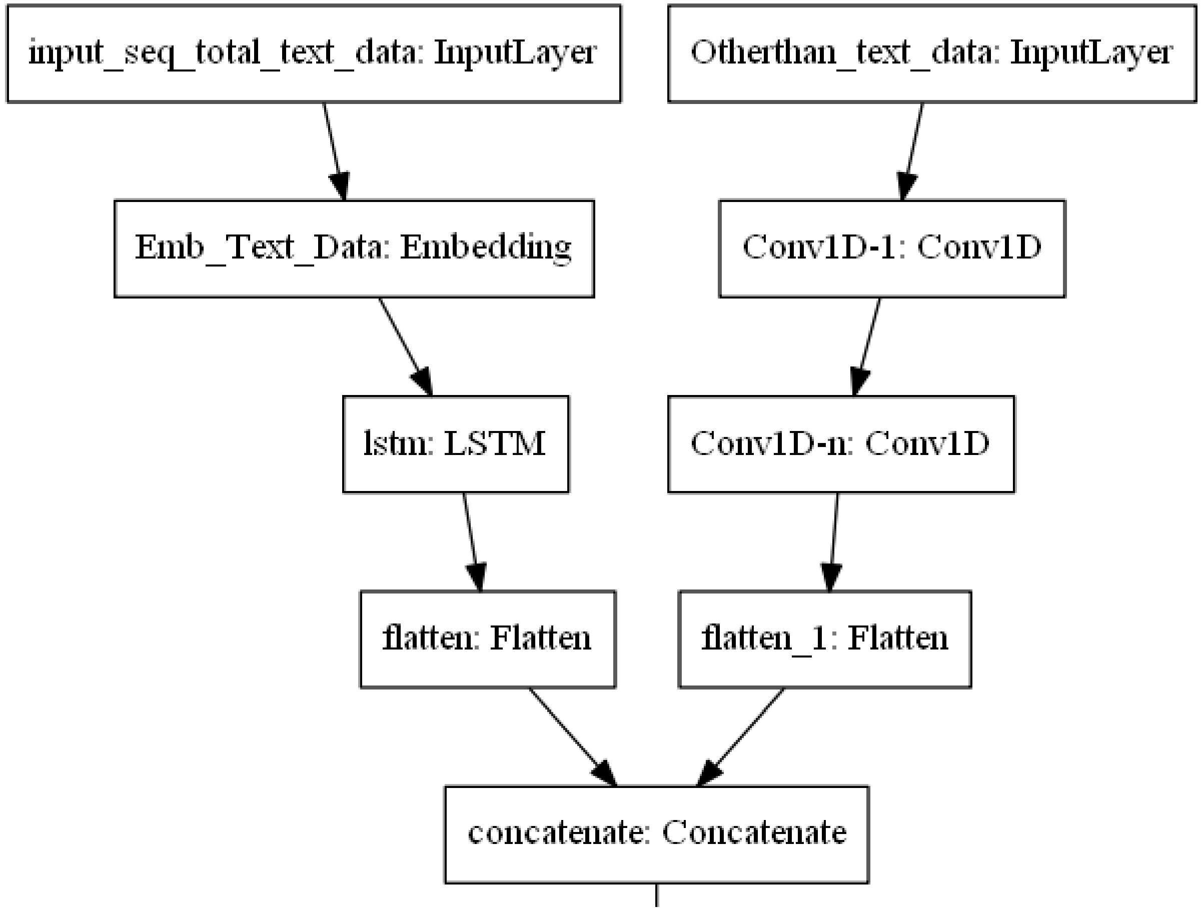
Out[ ]:
```
46331
```

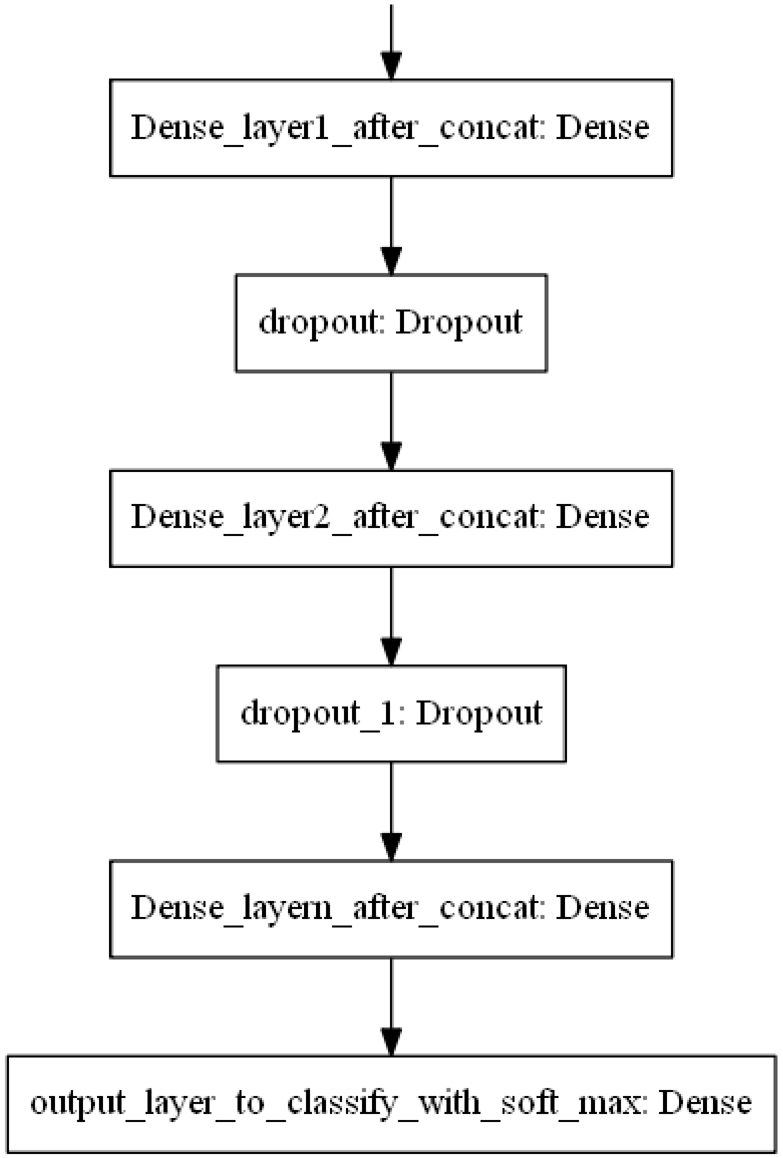
In [ ]:

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

In [ ]:
```
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)
```

Model-3




Dense_layer1_after_concat: Dense
dropout: Dropout
Dense_layer2_after_concat: Dense
dropout_1: Dropout
Dense_layern_after_concat: Dense
output_layer_to_classify_with_soft_max: Dense

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

- **input_seq_total_text_data**:

```
. Use text column('essay'), and use the Embedding layer to get word vectors.
. Use given predefined glove word vectors, don't train any word vectors.

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

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

- **Other_than_text_data**:

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

. Neumerical values and use CNN1D as shown in above figure.

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

</pre>

In [ ]:

```
#padded_idf_essay_train.shape, essay_padded_train.shape
```

In [ ]:

```
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="model3.hdf5"
log_dir="logs3/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 [ ]:

```
otherthan_text_data_train=np.hstack((school_state_train.todense(),project_grad_train.todense(),clean_cate_train.toder
otherthan_text_data_test=np.hstack((school_state_test.todense(),project_grad_test.todense(),clean_cate_test.todense()
otherthan_text_data_cv=np.hstack((school_state_cv.todense(),project_grad_cv.todense(),clean_cate_cv.todense(),clean_s
```

In [ ]:

```
otherthan_text_data_train.shape, otherthan_text_data_test.shape, otherthan_text_data_cv.shape #1+1+3+5+1+2
```

Out[ ]:

```
((66466, 101), (24035, 101), (18747, 101))
```

In [ ]:

```
from tensorflow.keras.layers import Conv1D,MaxPool1D,Activation,Flatten

#essay text
essay_input = Input(shape=(mx_len,))
essay_embed = Embedding(vocab_size, 300, input_length=mx_len,weights=[glove_mat],trainable = False)(essay_input)
essay_lstm=LSTM(25)(essay_embed)
flatten = Flatten(data_format='channels_last',name='flatten')(essay_lstm)

#other inputs
input_others = Input(shape=(101,1))
conv = Conv1D(64, 3,data_format="channels_last",padding="same")(input_others)
conv_1 = Conv1D(64, 3,data_format="channels_last",padding="same")(conv)
flatten_1 = Flatten(data_format='channels_last',name='flatten_1')(conv_1)

#concatenate
conc = Concatenate(axis=1)([flatten,flatten_1])

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

#dropout1
drop = Dropout(0.35)(FC1)

#Dense2
FC2 = Dense(256,activation='relu',kernel_initializer='he_normal')(drop)

#Dropout2
drop_1 = Dropout(0.3)(FC2)

#Dense3
FC3 = Dense(32,activation='relu',kernel_initializer='he_normal')(drop_1)
```

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

In [ ]:

```
model3 = Model(inputs=[essay_input,input_others],outputs=output)
model3.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            [(None, 331)]        0

__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 101, 1)]     0

__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 331, 300)     13899300    input_5[0][0]

__________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, 101, 64)      256         input_6[0][0]

__________________________________________________________________________________________________
lstm_2 (LSTM)                   (None, 25)           32600       embedding_2[0][0]

__________________________________________________________________________________________________
conv1d_5 (Conv1D)               (None, 101, 64)      12352       conv1d_4[0][0]

__________________________________________________________________________________________________
flatten (Flatten)               (None, 25)           0           lstm_2[0][0]

__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 6464)         0           conv1d_5[0][0]

__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 6489)         0           flatten[0][0]
                                                                 flatten_1[0][0]

__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 512)          3322880     concatenate_2[0][0]

__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 512)          0           dense_8[0][0]

__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 256)          131328      dropout_4[0][0]

__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 256)          0           dense_9[0][0]

__________________________________________________________________________________________________
dense_10 (Dense)                (None, 32)           8224        dropout_5[0][0]

__________________________________________________________________________________________________
dense_11 (Dense)                (None, 1)            33          dense_10[0][0]
```

```
==================================================================================================
Total params: 17,406,973
Trainable params: 3,507,673
Non-trainable params: 13,899,300
__________________________________________________________________________________________________
```

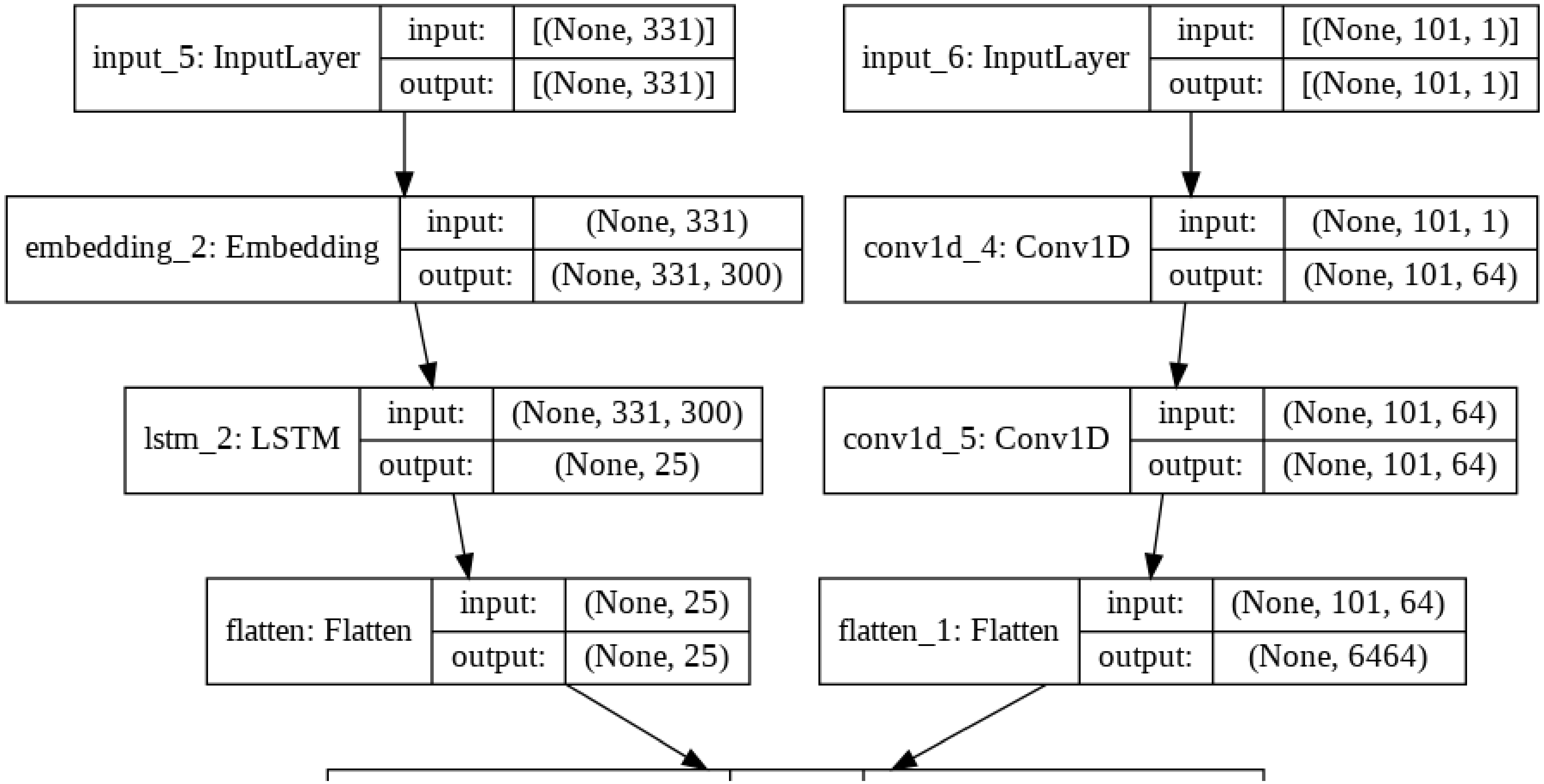
In [ ]:

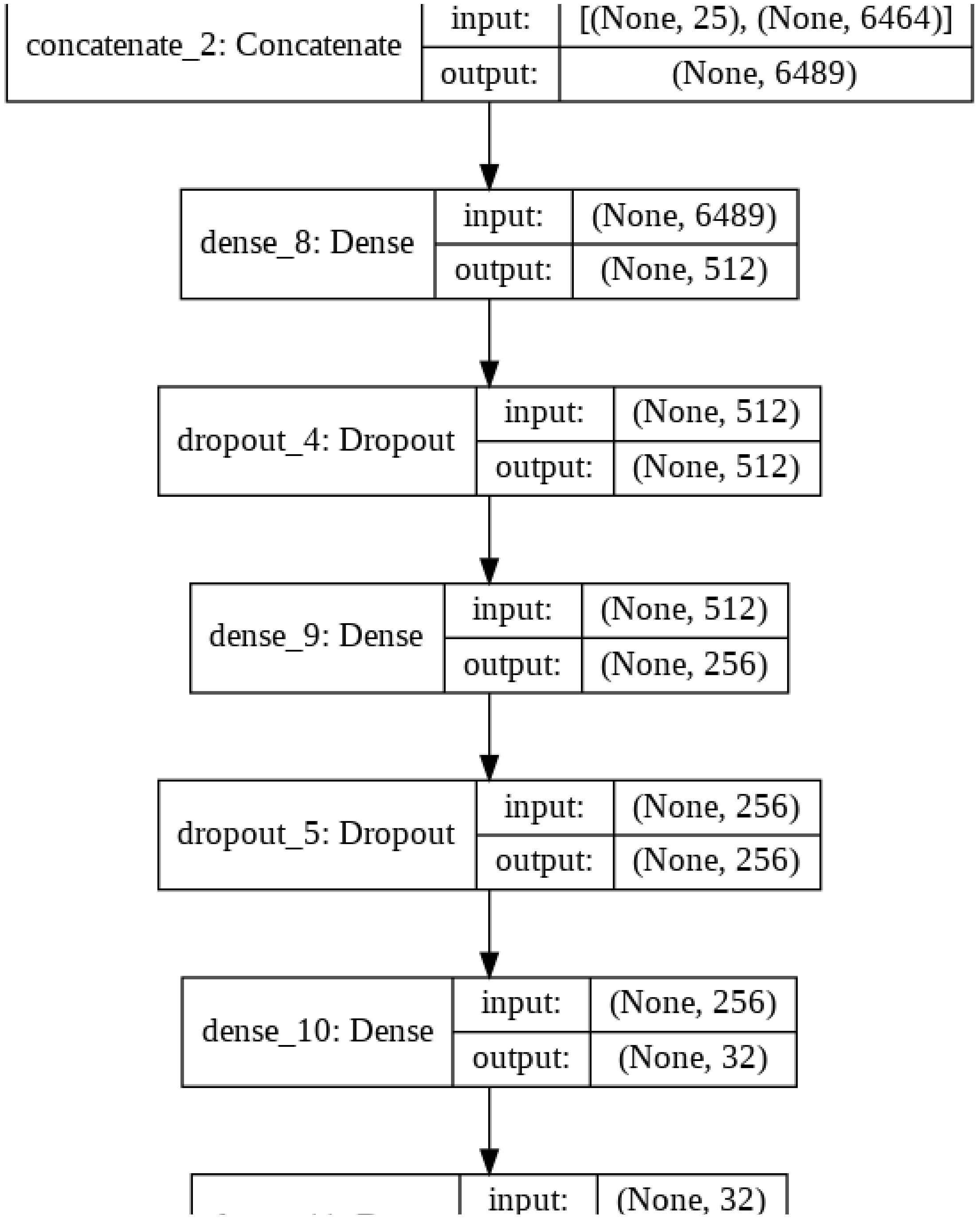
```
model3.compile(optimizer=Adam(learning_rate=0.001),loss='binary_crossentropy',metrics=['accuracy',auc])
```

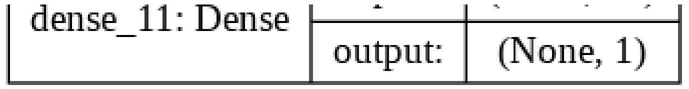
In [ ]:

```
from tensorflow.keras.utils import plot_model
#https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
plot_model(model3, to_file='model3_plot.png', show_shapes=True, show_layer_names=True)
```

Out[ ]:




concatenate_2: Concatenate
input: [(None, 25), (None, 6464)]
output: (None, 6489)
dense_8: Dense
input: (None, 6489)
output: (None, 512)
dropout_4: Dropout
input: (None, 512)
output: (None, 512)
dense_9: Dense
input: (None, 512)
output: (None, 256)
dropout_5: Dropout
input: (None, 256)
output: (None, 256)
dense_10: Dense
input: (None, 256)
output: (None, 32)
input: (None, 32)



In [ ]:
```
model3.fit([essay_padded_train, otherthan_text_data_train],y_train,validation_data=([essay_padded_cv, otherthan_text_
```

```
Epoch 1/20

Epoch 00001: LearningRateScheduler setting learning rate to 0.0009000000427477062.
520/520 [==============================] - 65s 65ms/step - loss: 0.4047 - accuracy: 0.8485 - auc: 0.6643 - val_loss:
0.3825 - val_accuracy: 0.8509 - val_auc: 0.7208

Epoch 00001: val_auc improved from -inf to 0.72083, saving model to model3.hdf5
Epoch 2/20

Epoch 00002: LearningRateScheduler setting learning rate to 0.0009000000427477062.
520/520 [==============================] - 30s 59ms/step - loss: 0.3761 - accuracy: 0.8519 - auc: 0.7374 - val_loss:
0.3704 - val_accuracy: 0.8549 - val_auc: 0.7433

Epoch 00002: val_auc improved from 0.72083 to 0.74334, saving model to model3.hdf5
Epoch 3/20

Epoch 00003: LearningRateScheduler setting learning rate to 0.0009000000427477062.
520/520 [==============================] - 32s 62ms/step - loss: 0.3661 - accuracy: 0.8548 - auc: 0.7589 - val_loss:
0.3672 - val_accuracy: 0.8510 - val_auc: 0.7546

Epoch 00003: val_auc improved from 0.74334 to 0.75459, saving model to model3.hdf5
Epoch 4/20

Epoch 00004: LearningRateScheduler setting learning rate to 0.0009000000427477062.
520/520 [==============================] - 32s 62ms/step - loss: 0.3592 - accuracy: 0.8570 - auc: 0.7711 - val_loss:
0.3662 - val_accuracy: 0.8559 - val_auc: 0.7546

Epoch 00004: val_auc improved from 0.75459 to 0.75460, saving model to model3.hdf5
Epoch 5/20

Epoch 00005: LearningRateScheduler setting learning rate to 0.0009000000427477062.
520/520 [==============================] - 32s 62ms/step - loss: 0.3518 - accuracy: 0.8604 - auc: 0.7840 - val_loss:
0.3681 - val_accuracy: 0.8557 - val_auc: 0.7535

Epoch 00005: val_auc did not improve from 0.75460
Epoch 6/20

Epoch 00006: LearningRateScheduler setting learning rate to 0.0008100000384729355.
520/520 [==============================] - 32s 62ms/step - loss: 0.3436 - accuracy: 0.8633 - auc: 0.7968 - val_loss:
0.3654 - val_accuracy: 0.8552 - val_auc: 0.7545

Epoch 00006: val_auc did not improve from 0.75460
Epoch 7/20

Epoch 00007: LearningRateScheduler setting learning rate to 0.0008100000559352338.
520/520 [==============================] - 32s 61ms/step - loss: 0.3377 - accuracy: 0.8653 - auc: 0.8052 - val_loss:
0.3715 - val_accuracy: 0.8565 - val_auc: 0.7523

Epoch 00007: val_auc did not improve from 0.75460
Epoch 8/20

Epoch 00008: LearningRateScheduler setting learning rate to 0.0008100000559352338.
520/520 [==============================] - 32s 62ms/step - loss: 0.3308 - accuracy: 0.8683 - auc: 0.8168 - val_loss:
0.3694 - val_accuracy: 0.8543 - val_auc: 0.7506

Epoch 00008: val_auc did not improve from 0.75460
Epoch 9/20

Epoch 00009: LearningRateScheduler setting learning rate to 0.0008100000559352338.
520/520 [==============================] - 30s 58ms/step - loss: 0.3245 - accuracy: 0.8710 - auc: 0.8244 - val_loss:
0.3719 - val_accuracy: 0.8547 - val_auc: 0.7456

Epoch 00009: val_auc did not improve from 0.75460
Epoch 10/20

Epoch 00010: LearningRateScheduler setting learning rate to 0.0008100000559352338.
520/520 [==============================] - 30s 57ms/step - loss: 0.3187 - accuracy: 0.8730 - auc: 0.8335 - val_loss:
0.3765 - val_accuracy: 0.8525 - val_auc: 0.7378

Epoch 00010: val_auc did not improve from 0.75460
Epoch 11/20

Epoch 00011: LearningRateScheduler setting learning rate to 0.0007290000503417104.
520/520 [==============================] - 32s 61ms/step - loss: 0.3092 - accuracy: 0.8771 - auc: 0.8438 - val_loss:
0.3869 - val_accuracy: 0.8515 - val_auc: 0.7373

Epoch 00011: val_auc did not improve from 0.75460
```

```
Out[ ]: <keras.callbacks.History at 0x7f70176ed6d0>
```

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

```
In [ ]: %tensorboard --logdir logs3/fit/
```

```
UsageError: Line magic function `%tensorboard` not found.
```

```
In [ ]: y_test_pred = model3.predict([essay_padded_test, otherthan_text_data_test])
        test_fpr, test_tpr, thres = roc_curve(y_test, y_test_pred)
        auc(test_fpr, test_tpr)
```

```
Out[ ]: 0.7534767093817758
```

Compared to all other models, model3 performing well and even coverged faster, for the higher AUC values.