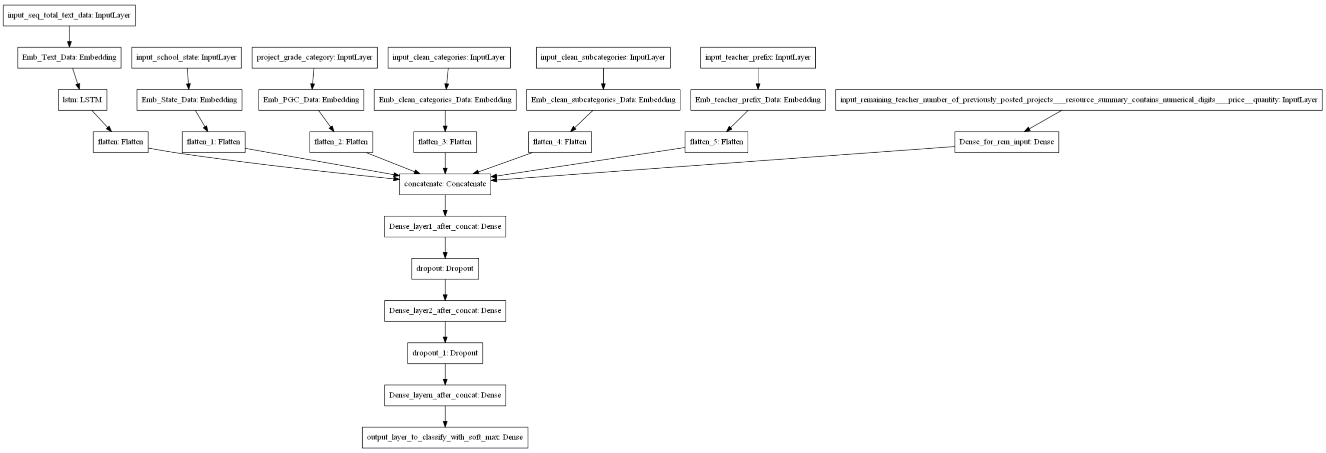
Assignment : 14

```
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, resource
s: 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-1

Build and Train deep neural network as shown below


input_seq_total_text_data: InputLayer
Emb_Text_Data: Embedding
input_school_state: InputLayer
project_grade_category: InputLayer
input_clean_categories: InputLayer
input_clean_subcategories: InputLayer
input_teacher_prefix: InputLayer
lstm: LSTM
Emb_State_Data: Embedding
Emb_PGC_Data: Embedding
Emb_clean_categories_Data: Embedding
Emb_clean_subcategories_Data: Embedding
Emb_teacher_prefix_Data: Embedding
input_remaining_teacher_number_of_previously_posted_projects___resource_summary_contains_numerical_digits___price__quantity: InputLayer
flatten: Flatten
flatten_1: Flatten
flatten_2: Flatten
flatten_3: Flatten
flatten_4: Flatten
flatten_5: Flatten
Dense_for_rem_input: Dense
concatenate: Concatenate
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/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._pric** ---concatenate remaining columns and add a Dense layer after that.

- For LSTM, you can choose your sequence padding methods on your own or you can train your LSTM without padding, there is no restriction on that.

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

https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work

input_layer = Input(shape=(n,)) embedding = Embedding(no_1, no_2, input_length=n)(input_layer) flatten = Flatten()(embedding)

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

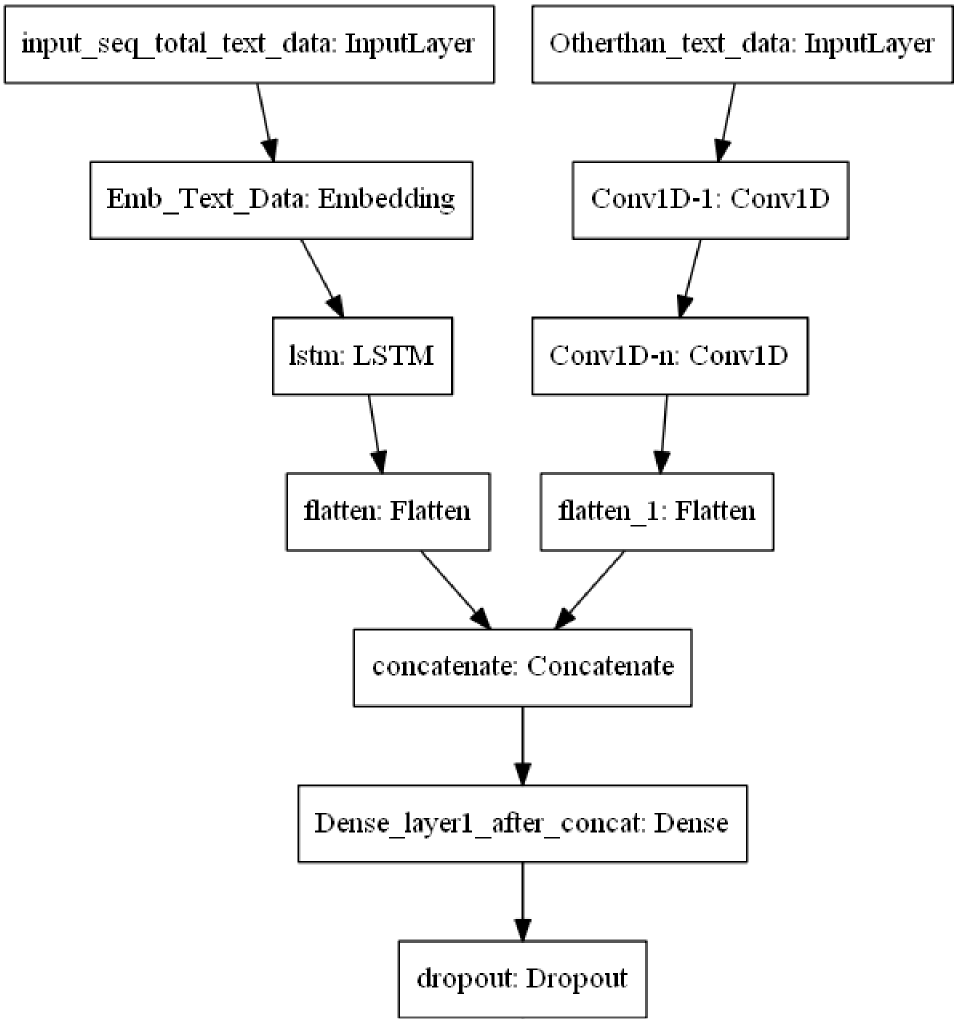
2. Please go through this link https://keras.io/getting-started/functional-api-guide/ and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.

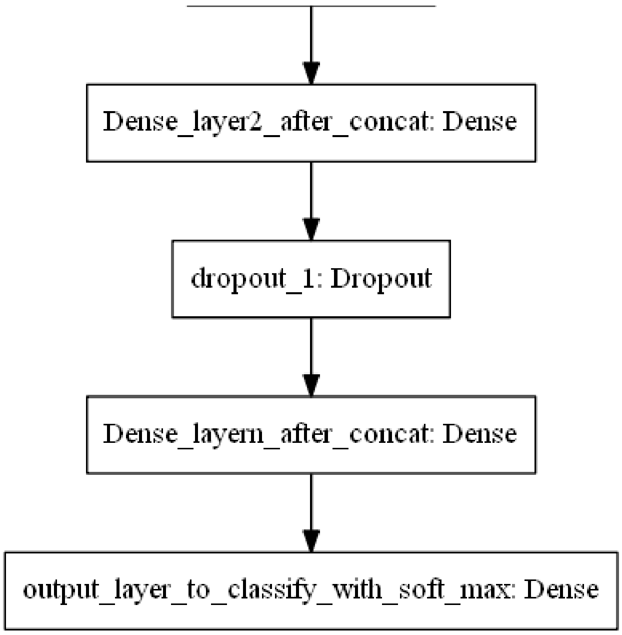
Model-2

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. Train the TF-IDF on the Train data

2. Get the idf value for each word we have in the train data.

3. Remove the low idf value and high idf value words from our data. 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. (you can plot a box pl
ots and take only the idf scores within IQR range and corresponding words)

4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train on tot
al data but in Model-2 train on data after removing some words based on IDF values)
```

Model-3




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 o
nehot 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 [1]:

```
### Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from tensorflow.keras.utils import to_categorical
from sklearn.utils import compute_class_weight
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Embedding, Flatten, LSTM, Dense, concatenate, Dropout,Ba
tchNormalization,SpatialDropout1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import he_normal
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Model, load_model
from sklearn.metrics import roc_auc_score
from tensorflow.keras import regularizers
from tensorflow.python.keras.callbacks import TensorBoard,ModelCheckpoint
from sklearn.preprocessing import Normalizer
import pickle
import warnings
import warnings
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

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

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

In [3]:

```
### read pre-processed data
project_data = pd.read_csv('/content/drive/My Drive/LSTM Assignment/preprocessed_data.csv')
project_data.shape
```

Out[3]:

```
(109248, 9)
```

In [4]:

```
y_data = project_data['project_is_approved']
x_data = project_data.drop(['project_is_approved'],axis=1)
```

In [5]:

```
### split ur data in train, test and Cross Validation data
x_train,x_test,y_train,y_test = train_test_split(x_data, y_data , stratify = y_data, train_size = 0
.8, random_state =99)

x_train,x_cv,y_train,y_cv = train_test_split(x_train, y_train, stratify = y_train, train_size = 0.8
, random_state = 99)
```

In [6]:

```
print("Shape of the Train dataset: ", x_train.shape[0])
print("Shape of the Test dataset: ", x_test.shape[0])
print("Shape of the cv dataset:", x_cv.shape[0])
```

```
Shape of the Train dataset:  69918
Shape of the Test dataset:  21850
Shape of the cv dataset: 17480
```

In [7]:

```
def tokenize_cat_data(x_train,x_cv,x_test,category):
  from tensorflow.keras.preprocessing.text import Tokenizer
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(x_train[category].tolist())
  seq_train = tokenizer.texts_to_sequences(x_train[category])
  seq_cv = tokenizer.texts_to_sequences(x_cv[category])
  seq_test = tokenizer.texts_to_sequences(x_test[category])
  vocab_size = len(tokenizer.word_index) + 1

  x_train[category] = seq_train

  x_train[category] = seq_train
  x_cv[category] = seq_cv
  x_test[category] = seq_test
  return x_train,x_cv,x_test, vocab_size
```

In [8]:

```
x_train,x_cv,x_test,state_size = tokenize_cat_data(x_train,x_cv,x_test,category='school_state')
```

In [9]:

```
x_train,x_cv,x_test,tpr_size = tokenize_cat_data(x_train,x_cv,x_test,category='teacher_prefix')
```

In [10]:

```
x_train,x_cv,x_test,pgc_size =
tokenize_cat_data(x_train,x_cv,x_test,category='project_grade_category')
```

In [11]:

```
x_train,x_cv,x_test,cc_size = tokenize_cat_data(x_train,x_cv,x_test,category='clean_categories')
```

In [12]:

```
x_train,x_cv,x_test,csc_size =
tokenize_cat_data(x_train,x_cv,x_test,category='clean_subcategories')
```

In [13]:

```
x_train.head()
```

Out[13]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_su
18589	[9]	[2]	[1, 4, 5]	3	[8]	
102121	[29]	[2]	[1, 6, 7]	0	[3, 4]	
40835	[1]	[2]	[1, 4, 5]	0	[3, 4, 9, 10]	
10349	[14]	[2]	[1, 2, 3]	17	[1, 2]	
80988	[1]	[1]	[1, 2, 3]	22	[8, 3, 4]	

In [14]:

```
#converting class labels to two class lable categorical variables
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_cv = to_categorical(y_cv)
```

In [15]:

```
## class weightage -- pass in our model fit parameter.
class_wght = compute_class_weight("balanced", classes= np.unique(y_data),y=y_data)
class_wght = {0:class_wght[0],
              1: class_wght[1]}
```

In [16]:

In [16]:

```
##pad sequences to have equal number of features
### school state
max_length = x_train['school_state'].apply(lambda x : len(x)).max()
X_train_school_state = pad_sequences(x_train['school_state'], maxlen=max_length)
X_test_school_state = pad_sequences(x_test['school_state'], maxlen=max_length)
X_cv_school_state = pad_sequences(x_cv['school_state'], maxlen=max_length)
print(X_train_school_state[55])

### teacher prefix
max_length = x_train['teacher_prefix'].apply(lambda x : len(x)).max()
X_train_tpr = pad_sequences(x_train['teacher_prefix'], maxlen=max_length)
X_test_tpr = pad_sequences(x_test['teacher_prefix'], maxlen=max_length)
X_cv_tpr = pad_sequences(x_cv['teacher_prefix'], maxlen=max_length)
print(X_train_tpr[55])

### proejct grade category
max_length = x_train['project_grade_category'].apply(lambda x : len(x)).max()
X_train_pgc = pad_sequences(x_train['project_grade_category'], maxlen=max_length)
X_test_pgc = pad_sequences(x_test['project_grade_category'], maxlen=max_length)
X_cv_pgc = pad_sequences(x_cv['project_grade_category'], maxlen=max_length)
print(X_train_pgc[55])

### clean_categories
max_length = x_train['clean_categories'].apply(lambda x : len(x)).max()
X_train_cc = pad_sequences(x_train['clean_categories'], maxlen=max_length)
X_test_cc = pad_sequences(x_test['clean_categories'], maxlen=max_length)
X_cv_cc = pad_sequences(x_cv['clean_categories'], maxlen=max_length)
print(X_train_cc[55])

### clean_subcategories
max_length = x_train['clean_subcategories'].apply(lambda x : len(x)).max()
print(max_length)
X_train_csc = pad_sequences(x_train['clean_subcategories'], maxlen=max_length)
X_test_csc = pad_sequences(x_test['clean_subcategories'], maxlen=max_length)
X_cv_csc = pad_sequences(x_cv['clean_subcategories'], maxlen=max_length)
print(X_train_csc[55])
```

```
[9]
[1]
[1 8 9]
[0 0 0 3 4]
5
[ 0  0  7  5 14]
```

In [17]:

```
### https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train["essay"].tolist())
text_seq_train = tokenizer.texts_to_sequences(x_train["essay"])
text_seq_cv = tokenizer.texts_to_sequences(x_cv["essay"])
text_seq_test = tokenizer.texts_to_sequences(x_test["essay"])
```

In [18]:

```
padded_text_train = pad_sequences(text_seq_train,maxlen=200,padding='post', truncating='post')
padded_text_test = pad_sequences(text_seq_test, maxlen=200,padding='post', truncating='post')
padded_text_cv = pad_sequences(text_seq_cv, maxlen=200,padding='post', truncating='post')
```

In [19]:

```
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out[19]:

```
47376
```

In [20]:

```
glove_vector_saved = open("/content/drive/My Drive/LSTM Assignment/glove_vectors","rb")
glove_words = pickle.load(glove_vector_saved)
```

In [21]:

```
embedding_mat = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vec = glove_words.get(word)
    if embedding_vec is not None:
        embedding_mat[i] = embedding_vec

print(embedding_mat.shape)
```

```
(47376, 300)
```

In [22]:

```
input_text = Input(shape=(200,),name="input_text") ## dim of text input
embed_text_data = Embedding(input_dim = vocab_size,output_dim =
300,weights=[embedding_mat],trainable=False)(input_text)
##dropped_out = SpatialDropout1D(0.3)(embed_text_data)
lstm_out = LSTM(128,return_sequences=True,recurrent_dropout=0.3,recurrent_regularizer=l2(0.001))
(embed_text_data)
flatted_text_out = Flatten()(lstm_out)
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't meet the cuDNN kernel cri
teria. It will use generic GPU kernel as fallback when running on GPU
```

In [23]:

```
input_state = Input(shape=(1),name="input_state") ## dim of state input
embed_state_data = Embedding(state_size,2)(input_state)
flatted_state_out = Flatten()(embed_state_data)
```

In [24]:

```
input_pgc = Input(shape=(3),name="input_pgc") ## dim of project grade category input
embed_pgc_data = Embedding(pgc_size,2)(input_pgc)
flatted_pgc_out = Flatten()(embed_pgc_data)
```

In [25]:

```
input_tchr_pre = Input(shape=(1),name="input_tchr_pre")
embed_tpr_data = Embedding(tpr_size,2)(input_tchr_pre)
flatted_tpr_out = Flatten()(embed_tpr_data)
```

In [26]:

```
sizeof_clean_cat = len(X_train_cc[0])
input_clean_cat = Input(shape=(sizeof_clean_cat),name="input_clean_cat")
embed_cc_data = Embedding(cc_size,2)(input_clean_cat)
flatted_cc_out = Flatten()(embed_cc_data)
```

In [27]:

```
sizeof_sub_clean_cat = len(X_train_csc[0])
input_clean_sub_cat = Input(shape=(sizeof_sub_clean_cat),name="input_clean_sub_cat")
embed_csc_data = Embedding(csc_size,2)(input_clean_sub_cat)
flatted_csc_out = Flatten()(embed_csc_data)
```

In [28]:

```
### Normalize your data
def norm_data(X_tr, X_cv, X_test, col_name = 'price'):

    normalizer = Normalizer()
    normalizer.fit(X_tr[col_name].values.reshape(1,-1))
    X_tr_norm = normalizer.transform(X_tr[col_name].values.reshape(1,-1)).reshape(-1,1)
    X_cv_norm = normalizer.transform(X_cv[col_name].values.reshape(1,-1)).reshape(-1,1)
    X_test_norm = normalizer.transform(X_test[col_name].values.reshape(1,-1)).reshape(-1,1)


    print("After vectorizations")
    print("Shape of training data {}" .format(X_tr_norm.shape))
    print("Shape of cross validation data {}".format(X_cv_norm.shape))
    print("Shape of test data {}".format(X_test_norm.shape))
    print("="*100)
    return X_tr_norm,X_cv_norm,X_test_norm
```

In [29]:

```
X_train_price_norm,X_cv_price_norm,X_test_price_norm = norm_data(x_train, x_cv, x_test, "price")
```

```
After vectorizations
Shape of training data (69918, 1)
Shape of cross validation data (17480, 1)
Shape of test data (21850, 1)
===================================================================================================
```

In [30]:

```
X_train_nopp_norm,X_cv_nopp_norm,X_test_nopp_norm = norm_data(x_train, x_cv, x_test, "teacher_numbe
r_of_previously_posted_projects")
```

```
After vectorizations
Shape of training data (69918, 1)
Shape of cross validation data (17480, 1)
Shape of test data (21850, 1)
===================================================================================================
```

In [31]:

```
numerical_in = Input(shape=(2,),name="numerical_features")
numerical_dense_out = Dense(100,activation="relu",kernel_initializer="he_normal",kernel_regularizer
=regularizers.l2(0.001))(numerical_in)
```

In [32]:

```
concat_out =
concatenate([flatted_text_out,flatted_state_out,flatted_pgc_out,flatted_tpr_out,flatted_cc_out,fla
tted_csc_out,numerical_dense_out])
x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(concat
_out)
x = Dropout(0.4)(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.4)(x)
x = BatchNormalization()(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.4)(x)
output = Dense(2, activation = 'softmax')(x)
```

In [33]:

```
def auc_roc(y_true, y_pred):
    if len(np.unique(y_true[:,1])) == 1:
        return 0.5
    else:
        return roc_auc_score(y_true, y_pred)
def auc_roc_score(y_true, y_pred):
    return tf.compat.v1.py_func(auc_roc, (y_true, y_pred), tf.double)
```

In [34]:

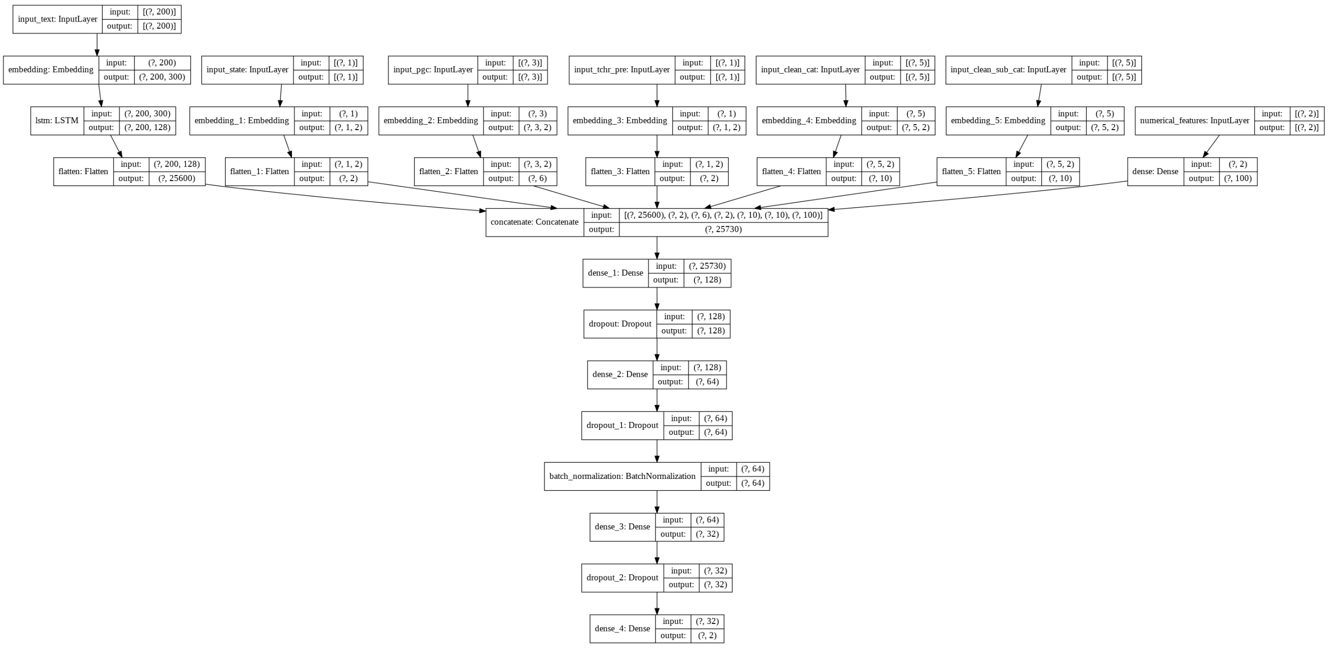
```
model = Model([input_text,input_state,input_pgc,input_tchr_pre,input_clean_cat,input_clean_sub_cat
,numerical_in], output)
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001,decay = 1e-
4),metrics=[auc_roc_score])
print(model.summary())
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_text (InputLayer)         [(None, 200)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 200, 300)     14212800    input_text[0][0]
__________________________________________________________________________________________________
input_state (InputLayer)        [(None, 1)]          0
__________________________________________________________________________________________________
input_pgc (InputLayer)          [(None, 3)]          0
__________________________________________________________________________________________________
input_tchr_pre (InputLayer)     [(None, 1)]          0
__________________________________________________________________________________________________
input_clean_cat (InputLayer)    [(None, 5)]          0
__________________________________________________________________________________________________
input_clean_sub_cat (InputLayer [(None, 5)]          0
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 200, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 2)         104         input_state[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 3, 2)         20          input_pgc[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 2)         12          input_tchr_pre[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 5, 2)         32          input_clean_cat[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 5, 2)         76          input_clean_sub_cat[0][0]
__________________________________________________________________________________________________
numerical_features (InputLayer) [(None, 2)]          0
__________________________________________________________________________________________________
flatten (Flatten)               (None, 25600)        0           lstm[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 2)            0           embedding_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 6)            0           embedding_2[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 2)            0           embedding_3[0][0]
__________________________________________________________________________________________________
flatten_4 (Flatten)             (None, 10)           0           embedding_4[0][0]
__________________________________________________________________________________________________
flatten_5 (Flatten)             (None, 10)           0           embedding_5[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 100)          300         numerical_features[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 25730)        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)          3293568     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 (BatchNorma (None, 64)           256         dropout_1[0][0]
__________________________________________________________________________________________________
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: 17,737,218
Trainable params: 3,524,290
Trainable params: 5,924,290
Non-trainable params: 14,212,928
_________________________________________________________________________________________________
None
```

In [35]:

```python
# summarize the model
from tensorflow.keras.utils import plot_model

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

Out[35]:


input_text: InputLayer | input: [(?, 200)] | output: [(?, 200)]
embedding: Embedding | input: (?, 200) | output: (?, 200, 300)
input_state: InputLayer | input: [(?, 1)] | output: [(?, 1)]
input_pgc: InputLayer | input: [(?, 3)] | output: [(?, 3)]
input_tchr_pre: InputLayer | input: [(?, 1)] | output: [(?, 1)]
input_clean_cat: InputLayer | input: [(?, 5)] | output: [(?, 5)]
input_clean_sub_cat: InputLayer | input: [(?, 5)] | output: [(?, 5)]
lstm: LSTM | input: (?, 200, 300) | output: (?, 200, 128)
embedding_1: Embedding | input: (?, 1) | output: (?, 1, 2)
embedding_2: Embedding | input: (?, 3) | output: (?, 3, 2)
embedding_3: Embedding | input: (?, 1) | output: (?, 1, 2)
embedding_4: Embedding | input: (?, 5) | output: (?, 5, 2)
embedding_5: Embedding | input: (?, 5) | output: (?, 5, 2)
numerical_features: InputLayer | input: [(?, 2)] | output: [(?, 2)]
flatten: Flatten | input: (?, 200, 128) | output: (?, 25600)
flatten_1: Flatten | input: (?, 1, 2) | output: (?, 2)
flatten_2: Flatten | input: (?, 3, 2) | output: (?, 6)
flatten_3: Flatten | input: (?, 1, 2) | output: (?, 2)
flatten_4: Flatten | input: (?, 5, 2) | output: (?, 10)
flatten_5: Flatten | input: (?, 5, 2) | output: (?, 10)
dense: Dense | input: (?, 2) | output: (?, 100)
concatenate: Concatenate | input: [(?, 25600), (?, 2), (?, 6), (?, 2), (?, 10), (?, 10), (?, 100)] | output: (?, 25730)
dense_1: Dense | input: (?, 25730) | output: (?, 128)
dropout: Dropout | input: (?, 128) | output: (?, 128)
dense_2: Dense | input: (?, 128) | output: (?, 64)
dropout_1: Dropout | input: (?, 64) | output: (?, 64)
batch_normalization: BatchNormalization | input: (?, 64) | output: (?, 64)
dense_3: Dense | input: (?, 64) | output: (?, 32)
dropout_2: Dropout | input: (?, 32) | output: (?, 32)
dense_4: Dense | input: (?, 32) | output: (?, 2)

In [36]:

```python
train_numeric_feature = np.array([X_train_price_norm,X_train_nopp_norm]).reshape(-1,2)
cv_numeric_feature = np.array([X_cv_price_norm,X_cv_nopp_norm]).reshape(-1,2)
test_numeric_feature = np.array([X_test_price_norm,X_test_nopp_norm]).reshape(-1,2)
```

In [37]:

```python
train_data =
[padded_text_train,X_train_school_state,X_train_pgc,X_train_tpr,X_train_cc,X_train_csc,train_numeri
c_feature]
cv_data = [padded_text_cv,X_cv_school_state,X_cv_pgc,X_cv_tpr,X_cv_cc,X_cv_csc,cv_numeric_feature]
test_data =
[padded_text_test,X_test_school_state,X_test_pgc,X_test_tpr,X_test_cc,X_test_csc,test_numeric_featu
re]
```

In [38]:

```python
# tensor-board in colab
# Refer: https://www.tensorflow.org/tensorboard/get_started
import os
import datetime

! rm -rf ./logs/
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
print(logdir)
```

```
logs/20200710-124642
```

In [39]:

```python
%reload_ext tensorboard
%tensorboard --logdir $logdir
```

In [40]:

```
#model fitting
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
tensorboard_callback = TensorBoard(logdir, histogram_freq=1)
filepath="weights_copy.best.hdf5"
checkpoint_callback = ModelCheckpoint(filepath, monitor='val_auc_roc_score', verbose=1, save_best_o
nly=True, mode='max')
callbacks_list = [checkpoint_callback,tensorboard_callback]
```

In [41]:

```
model.fit(train_data, y_train,epochs=10,verbose=1,batch_size=256,
          validation_data=(cv_data,y_cv),callbacks =callbacks_list,class_weight = class_wght )
```

```
Epoch 1/10
WARNING:tensorflow:From <ipython-input-33-abbf107bdcb5>:7: py_func (from
tensorflow.python.ops.script_ops) is deprecated and will be removed in a future version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

274/274 [==============================] - ETA: 0s - loss: 1.4453 - auc_roc_score: 0.5120
Epoch 00001: val_auc_roc_score improved from -inf to 0.48597, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 286s 1s/step - loss: 1.4453 - auc_roc_score: 0.5120 - v
al_loss: 1.2028 - val_auc_roc_score: 0.4860
Epoch 2/10
274/274 [==============================] - ETA: 0s - loss: 1.0229 - auc_roc_score: 0.5482
Epoch 00002: val_auc_roc_score improved from 0.48597 to 0.61857, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 279s 1s/step - loss: 1.0229 - auc_roc_score: 0.5482 - v
al_loss: 0.9431 - val_auc_roc_score: 0.6186
Epoch 3/10
274/274 [==============================] - ETA: 0s - loss: 0.8666 - auc_roc_score: 0.6685
Epoch 00003: val_auc_roc_score improved from 0.61857 to 0.72978, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 283s 1s/step - loss: 0.8666 - auc_roc_score: 0.6685 - v
al_loss: 0.9062 - val_auc_roc_score: 0.7298
Epoch 4/10
274/274 [==============================] - ETA: 0s - loss: 0.7732 - auc_roc_score: 0.7246
Epoch 00004: val_auc_roc_score improved from 0.72978 to 0.73985, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 279s 1s/step - loss: 0.7732 - auc_roc_score: 0.7246 - v
al_loss: 0.7065 - val_auc_roc_score: 0.7399
Epoch 5/10
274/274 [==============================] - ETA: 0s - loss: 0.7344 - auc_roc_score: 0.7399
Epoch 00005: val_auc_roc_score improved from 0.73985 to 0.74684, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 279s 1s/step - loss: 0.7344 - auc_roc_score: 0.7399 - v
al_loss: 0.8204 - val_auc_roc_score: 0.7468
Epoch 6/10
274/274 [==============================] - ETA: 0s - loss: 0.7067 - auc_roc_score: 0.7508
Epoch 00006: val_auc_roc_score improved from 0.74684 to 0.74734, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 279s 1s/step - loss: 0.7067 - auc_roc_score: 0.7508 - v
al_loss: 0.7508 - val_auc_roc_score: 0.7473
Epoch 7/10
274/274 [==============================] - ETA: 0s - loss: 0.6921 - auc_roc_score: 0.7580
Epoch 00007: val_auc_roc_score improved from 0.74734 to 0.75127, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 279s 1s/step - loss: 0.6921 - auc_roc_score: 0.7580 - v
al_loss: 0.7232 - val_auc_roc_score: 0.7513
Epoch 8/10
274/274 [==============================] - ETA: 0s - loss: 0.6793 - auc_roc_score: 0.7634
```

```
Epoch 00008: val_auc_roc_score improved from 0.75127 to 0.75411, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 279s 1s/step - loss: 0.6793 - auc_roc_score: 0.7634 - v
al_loss: 0.7397 - val_auc_roc_score: 0.7541
Epoch 9/10
274/274 [==============================] - ETA: 0s - loss: 0.6782 - auc_roc_score: 0.7751
Epoch 00009: val_auc_roc_score did not improve from 0.75411
274/274 [==============================] - 280s 1s/step - loss: 0.6782 - auc_roc_score: 0.7751 - v
al_loss: 0.6025 - val_auc_roc_score: 0.7517
Epoch 10/10
274/274 [==============================] - ETA: 0s - loss: 0.6787 - auc_roc_score: 0.7841
Epoch 00010: val_auc_roc_score did not improve from 0.75411
274/274 [==============================] - 279s 1s/step - loss: 0.6787 - auc_roc_score: 0.7841 - v
al_loss: 0.6577 - val_auc_roc_score: 0.7450
```

Out[41]:

```
<tensorflow.python.keras.callbacks.History at 0x7f777306e3c8>
```

Check the output of the test from the best fit model

In [43]:

```python
### load the weight from the saved file
model.load_weights("weights_copy.best.hdf5")
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001,decay = 1e-
4),metrics=[auc_roc_score])
```

In [44]:

```python
print("ROC-AUC for test data: %0.3f"%roc_auc_score(y_test,model.predict(test_data)))
print("ROC-AUC for CV data: %0.3f"%roc_auc_score(y_cv,model.predict(cv_data)))
print("ROC-AUC for train data: %0.3f"%roc_auc_score(y_train,model.predict(train_data)))
```

```
ROC-AUC for test data: 0.750
ROC-AUC for CV data: 0.755
ROC-AUC for train data: 0.798
```

In [45]:

```python
### plot AUC for test train-data
y_predict_test = model.predict(test_data)
y_predict_cv = model.predict(cv_data)
y_predict_train = model.predict(train_data)
```

In [66]:

```python
y_pred_tr = y_predict_train[:,1]
y_pred_cv = y_predict_cv[:,1]
y_pred_test = y_predict_test[:,1]
```

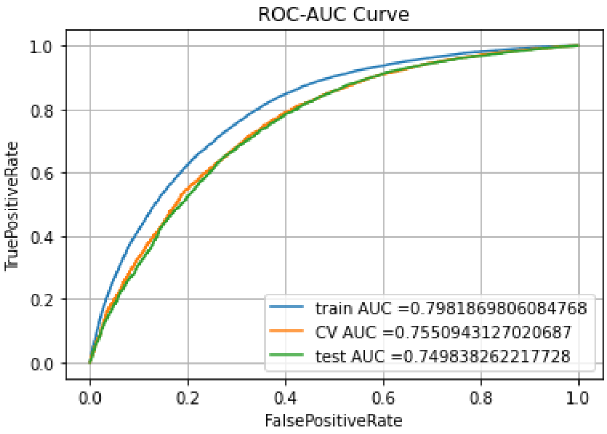
In [77]:

```python
y_tr = np.where(y_train == 1)[1]
y_tst = np.where(y_test == 1)[1]
y_crs_val = np.where(y_cv == 1)[1]
```

In [78]:

```python
from sklearn.metrics import roc_curve, auc
train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_pred_tr)
cv_fpr, cv_tpr, cv_thresholds = roc_curve(y_crs_val, y_pred_cv)
test_fpr, test_tpr, test_thresholds = roc_curve(y_tst, y_pred_test)
auc_tfidf_train = auc(train_fpr, train_tpr)
auc_tfidf_cv = auc(cv_fpr, cv_tpr)
auc_tfidf_test = auc(test_fpr, test_tpr)
### feature importance
```

```
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_tfidf_train))
plt.plot(cv_fpr, cv_tpr, label="CV AUC ="+str(auc_tfidf_cv))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_tfidf_test))
plt.legend()
plt.xlabel("FalsePositiveRate")
plt.ylabel("TruePositiveRate")
plt.title("ROC-AUC Curve")
plt.grid()
plt.show()
```



In [ ]: