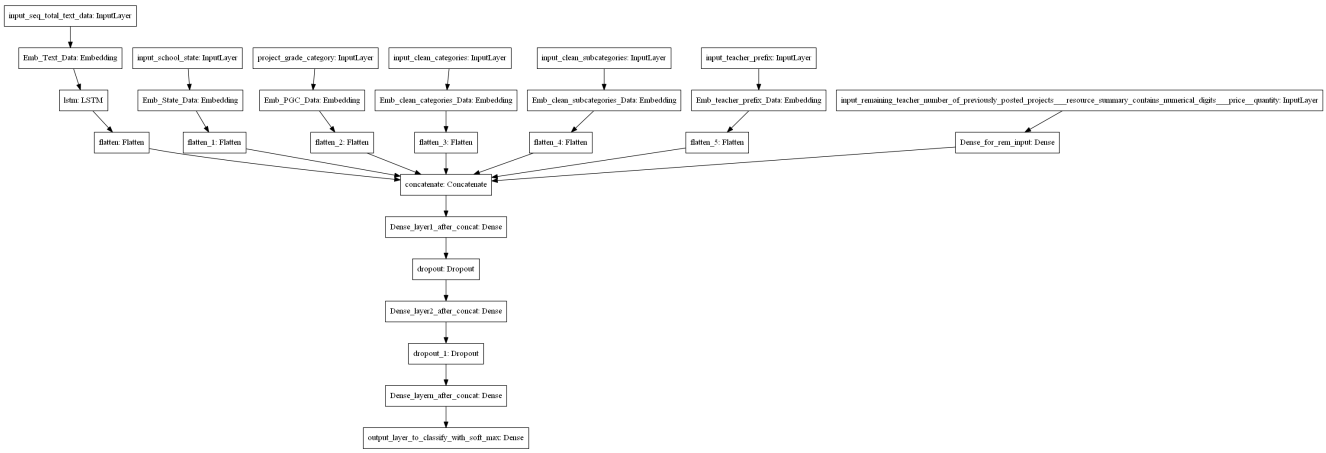
Assignment : 14

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-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** ---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/

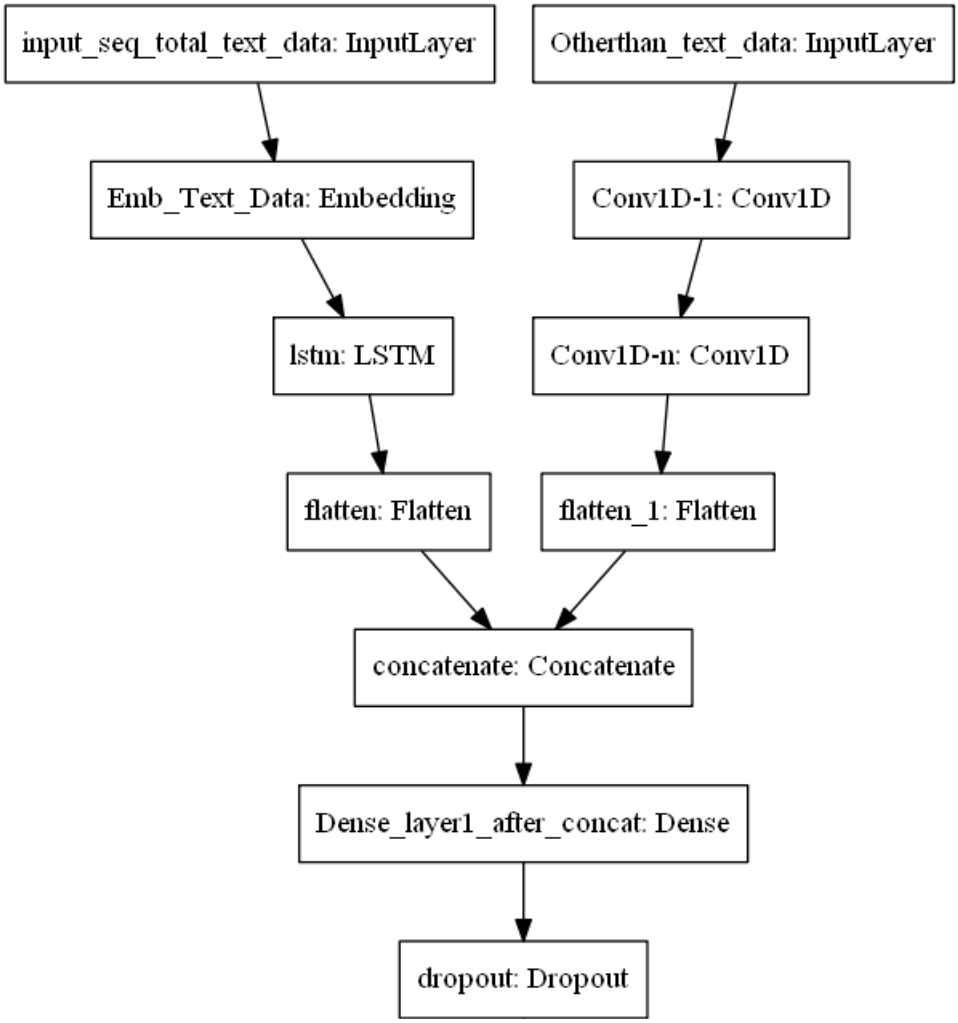
2. Please go through this link https://keras.io/getting-started/functional-api-guide/ and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.

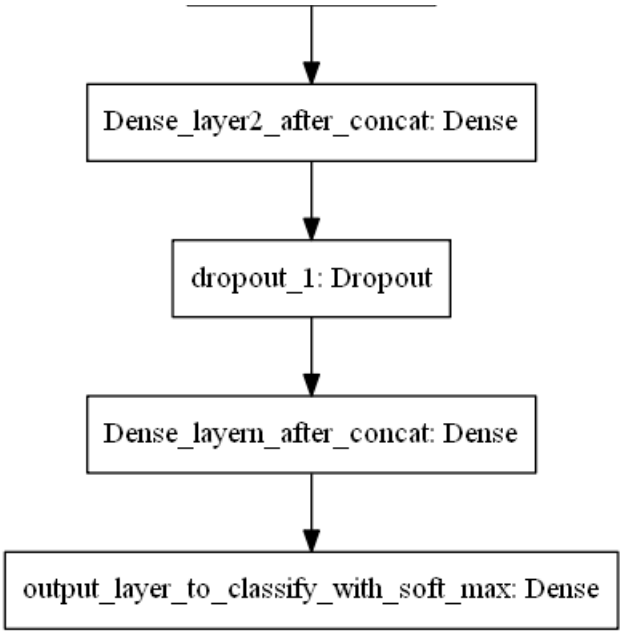
Model-2

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





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 [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,Conv1D
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
```

```
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]:

```
x_train.head()
```

Out[6]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_su
18589	mi	ms	grades_3_5	3	appliedlearning	charac
102121	ky	ms	grades_6_8	0	math_science	
40835	ca	ms	grades_3_5	0	math_science music_arts	environm
10349	ma	ms	grades_prek_2	17	literacy_language	
80988	ca	mrs	grades_prek_2	22	appliedlearning math_science	early healt

school_state teacher_prefix project_grade_category teacher_number_of_previously_posted_projects clean_categories clean_su

In [7]:

```
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 [8]:

```
from sklearn.preprocessing import OneHotEncoder
feature = np.array(['ram','shayam','meera','ram']).reshape(-1,1)
ohe = OneHotEncoder()
ohe.fit(feature)
x = ohe.transform(feature)
```

In [9]:

```
x.shape
```

Out[9]:

```
(4, 3)
```

In [10]:

```
def ohe_cat_data(x_train,x_cv,x_test,category):

    ohe = CountVectorizer()
    ohe.fit(x_train[category])
    x_train_cat = ohe.transform(x_train[category])
    x_cv_cat = ohe.transform(x_cv[category])
    x_test_cat = ohe.transform(x_test[category])

    print("size of training dataset {}".format(x_train_cat.shape))
    print("size of cv dataset {}".format(x_cv_cat.shape))
    print("size of test dataset {}".format(x_test_cat.shape))
    return x_train_cat,x_cv_cat,x_test_cat
```

In [11]:

```
ohe_ss_train, ohe_ss_cv, ohe_ss_test = ohe_cat_data(x_train,x_cv,x_test,category='school_state')
```

```
size of training dataset (69918, 51)
size of cv dataset (17480, 51)
size of test dataset (21850, 51)
```

In [12]:

```
ohe_tpr_train, ohe_tpr_cv, ohe_tpr_test =
ohe_cat_data(x_train,x_cv,x_test,category='teacher_prefix')
```

```
size of training dataset (69918, 5)
size of cv dataset (17480, 5)
size of test dataset (21850, 5)
```

In [13]:

```
ohe_pgc_train, ohe_pgc_cv, ohe_pgc_test =
ohe_cat_data(x_train,x_cv,x_test,category='project_grade_category')
```

```
size of training dataset (69918, 4)
size of cv dataset (17480, 4)
size of test dataset (21850, 4)
```

In [14]:

```
ohe_cc_train, ohe_cc_cv, ohe_cc_test =
ohe_cat_data(x_train,x_cv,x_test,category='clean_categories')
```

```
size of training dataset (69918, 9)
size of cv dataset (17480, 9)
size of test dataset (21850, 9)
```

In [15]:

```
ohe_csc_train, ohe_csc_cv, ohe_csc_test =
ohe_cat_data(x_train,x_cv,x_test,category='clean_subcategories')
```

```
size of training dataset (69918, 30)
size of cv dataset (17480, 30)
size of test dataset (21850, 30)
```

In [16]:

```
#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 [17]:

```
## class weightage -- pass in our model fit parameter.
class_wght = compute_class_weight("balanced", classes= np.unique(y_data),y=y_data)
class_wght = {i:wght for i,wght in enumerate(class_wght)}
class_wght
```

Out[17]:

```
{0: 3.3021400072542617, 1: 0.5892175263736975}
```

In [18]:

```
### 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 [19]:

```
padded_text_train = pad_sequences(text_seq_train,maxlen=500,padding='post', truncating='post')
padded_text_test = pad_sequences(text_seq_test, maxlen=500,padding='post', truncating='post')
padded_text_cv = pad_sequences(text_seq_cv, maxlen=500,padding='post', truncating='post')
```

In [20]:

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

Out[20]:

```
47376
```

In [21]:

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

In [22]:

```
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 [23]:

```
### 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))
    X_cv_norm = normalizer.transform(X_cv[col_name].values.reshape(-1,1))
    X_test_norm = normalizer.transform(X_test[col_name].values.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 [24]:

```
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 [25]:

```
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 [26]:

```
## prepare your data set to train your model
from scipy.sparse import hstack
X_tr_ohe_csr = hstack((ohe_ss_train,ohe_tpr_train, ohe_pgc_train, ohe_cc_train,
ohe_csc_train,X_train_nopp_norm,X_train_price_norm)).todense()
X_cv_ohe_csr = hstack((ohe_ss_cv,ohe_tpr_cv, ohe_pgc_cv, ohe_cc_cv, ohe_csc_cv,
X_cv_nopp_norm,X_cv_price_norm)).todense()
X_test_ohe_csr = hstack((ohe_ss_test,ohe_tpr_test, ohe_pgc_test, ohe_cc_test, ohe_csc_test, X_test_
nopp_norm,X_test_price_norm)).todense()

X_tr_ohe_csr = np.resize(X_tr_ohe_csr,new_shape=(y_train.shape[0],101,1))
X_test_ohe_csr = np.resize(X_test_ohe_csr,new_shape=(y_test.shape[0],101,1))
X_cv_ohe_csr = np.resize(X_cv_ohe_csr,new_shape=(y_cv.shape[0],101,1))
```

```
print("Final Data matrix")
print(X_tr_ohe_csr.shape, y_train.shape)
print(X_cv_ohe_csr.shape, y_cv.shape)
print(X_test_ohe_csr.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(69918, 101, 1) (69918, 2)
(17480, 101, 1) (17480, 2)
(21850, 101, 1) (21850, 2)
====================================================================================================
```

In [27]:

```
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.py_function(auc_roc, (y_true, y_pred), tf.double)
```

In [28]:

```
input_text = Input(shape=(500,),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)
#embed_text_data = SpatialDropout1D(0.3)(embed_text_data)
lstm_out = LSTM(128,kernel_regularizer=regularizers.l2(0.001),return_sequences=True)
(embed_text_data)
flatted_text_out = Flatten()(lstm_out)
```

In [33]:

```
# input 2
num_input = Input(shape=(101,1))
x = Conv1D(filters=64,kernel_size=3,strides=1)(num_input)
x = Conv1D(filters=64,kernel_size=3,strides=1)(x)
flatted_num_out = Flatten()(x)
```

In [34]:

```
concat_out = concatenate([flatted_text_out,flatted_num_out])
x = Dense(512,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(concat
_out)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(256,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.5)(x)
x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.5)(x)
output = Dense(2, activation = 'softmax')(x)
```

In [35]:

```
model = Model([input_text,num_input], output)
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001,decay = 1e-
4),metrics=[auc_roc_score])
print(model.summary())
```

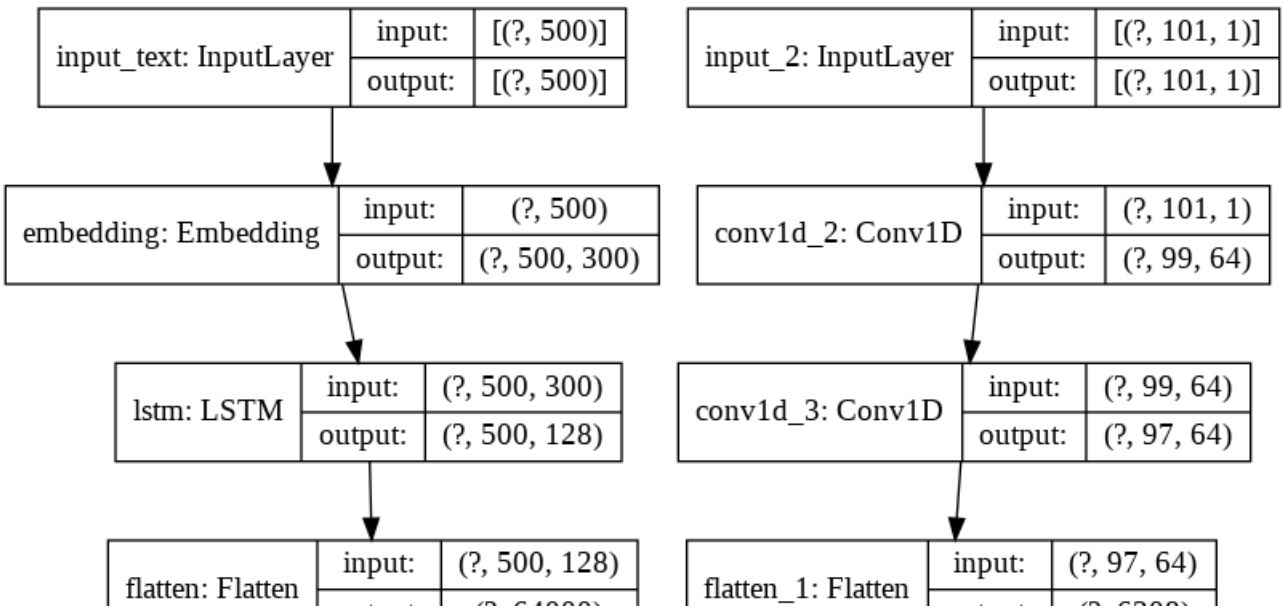
```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_text (InputLayer)         [(None, 500)]        0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 101, 1)]     0
```

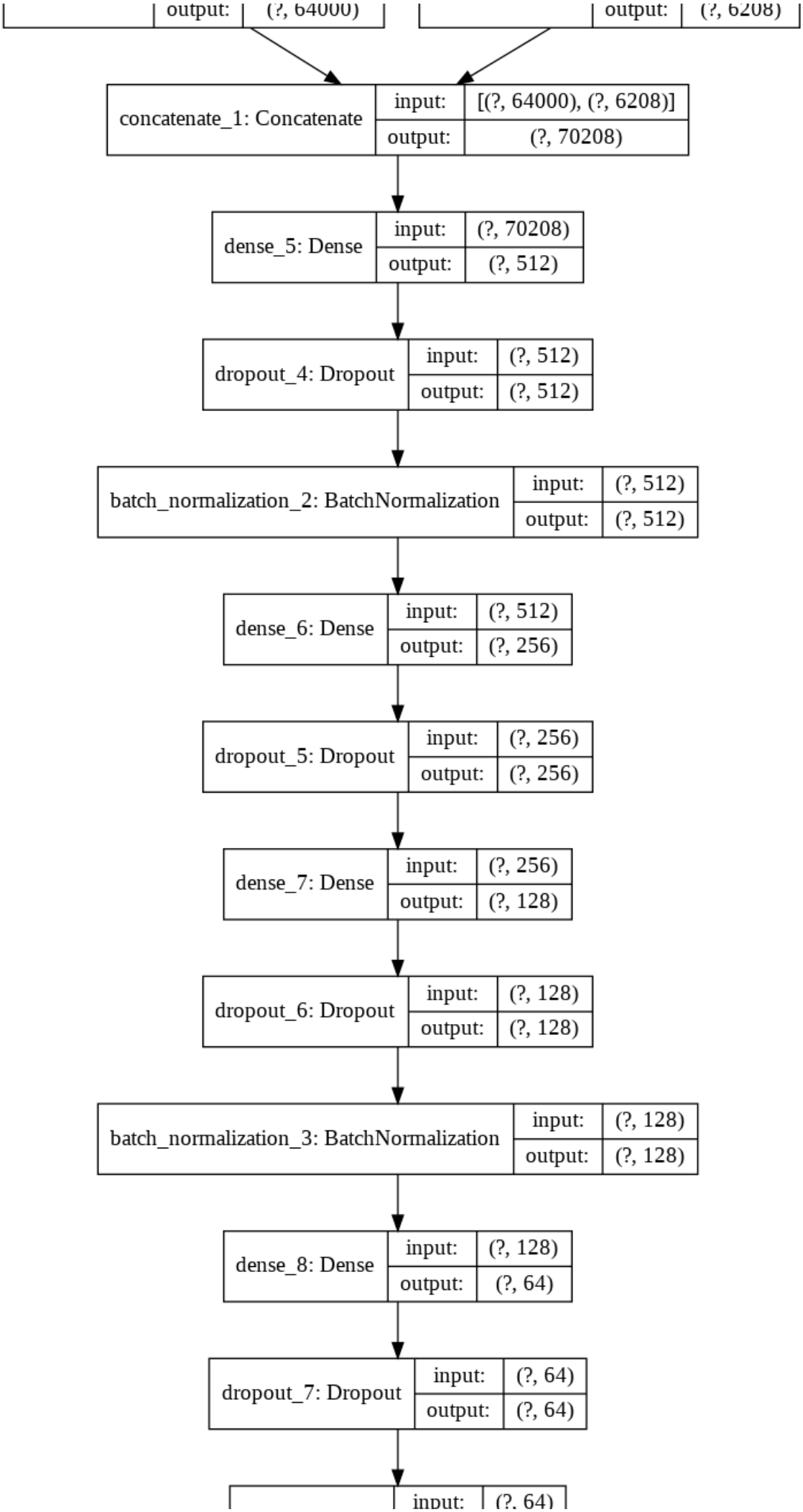
```
input_2 (InputLayer)            [(None, 101, 1)]     0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 500, 300)     14212800    input_text[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 99, 64)       256         input_2[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 500, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 97, 64)       12352       conv1d_2[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 64000)        0           lstm[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 6208)         0           conv1d_3[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 70208)        0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 512)          35947008    concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 512)          0           dense_5[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 512)          2048        dropout_4[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 256)          131328      batch_normalization_2[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 256)          0           dense_6[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 128)          32896       dropout_5[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 128)          0           dense_7[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 128)          512         dropout_6[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 64)           8256        batch_normalization_3[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 64)           0           dense_8[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 2)            130         dropout_7[0][0]
==================================================================================================
Total params: 50,567,234
Trainable params: 36,353,154
Non-trainable params: 14,214,080
__________________________________________________________________________________________________
None
```

In [36]:

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

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

Out[36]:




output: (?, 64000)
output: (?, 6208)
concatenate_1: Concatenate
input: [(?, 64000), (?, 6208)]
output: (?, 70208)
dense_5: Dense
input: (?, 70208)
output: (?, 512)
dropout_4: Dropout
input: (?, 512)
output: (?, 512)
batch_normalization_2: BatchNormalization
input: (?, 512)
output: (?, 512)
dense_6: Dense
input: (?, 512)
output: (?, 256)
dropout_5: Dropout
input: (?, 256)
output: (?, 256)
dense_7: Dense
input: (?, 256)
output: (?, 128)
dropout_6: Dropout
input: (?, 128)
output: (?, 128)
batch_normalization_3: BatchNormalization
input: (?, 128)
output: (?, 128)
dense_8: Dense
input: (?, 128)
output: (?, 64)
dropout_7: Dropout
input: (?, 64)
output: (?, 64)
input: (?, 64)

dense_9: Dense	output:	(?, 2)

In [37]:

```
train_data = [padded_text_train,X_tr_ohe_csr]
cv_data = [padded_text_cv,X_cv_ohe_csr]
test_data = [padded_text_test,X_test_ohe_csr]
```

In [38]:

```
# 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/20200718-092631
```

In [39]:

```
%load_ext tensorboard
%tensorboard --logdir $logdir
```

In [40]:

```
class LearningRateScheduler(tf.keras.callbacks.Callback):

    def __init__(self):

        self.lr = None
        self.val_auc = None
        self.prev_val_auc = 1e10
        self.epoch_cnt = 0

    def on_epoch_end(self, epoch, logs={}):

        # Get the current learning rate from model's optimizer.
        self.lr = float(tf.keras.backend.get_value(self.model.optimizer.lr))
        self.val_auc = float(logs.get('val_auc_roc_score'))
        self.epoch_cnt += 1

        print('Validation Accuracy is {}'.format(self.val_auc))
        scheduled_lr = self.lr
        if self.val_auc < self.prev_val_auc:
            # Set the value back to the optimizer before this epoch starts
            scheduled_lr = 0.1 * scheduled_lr
            tf.keras.backend.set_value(self.model.optimizer.lr, scheduled_lr)

        print('Optimized Learning Rate is {}'.format(scheduled_lr))
        self.prev_val_auc = self.val_auc
```

In [ ]:

```
#learning_rate = LearningRateScheduler()
```

In [41]:

```
#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 [42]:

```
model.fit(train_data, y_train,epochs=30,verbose=1,batch_size=256,
          validation_data=(cv_data,y_cv),callbacks =callbacks_list,class_weight = class_wght )
```

```
Epoch 1/30
274/274 [==============================] - ETA: 0s - loss: 2.8825 - auc_roc_score: 0.5162
Epoch 00001: val_auc_roc_score improved from -inf to 0.57011, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 436ms/step - loss: 2.8825 - auc_roc_score: 0.5162
- val_loss: 2.4297 - val_auc_roc_score: 0.5701
Epoch 2/30
274/274 [==============================] - ETA: 0s - loss: 2.4430 - auc_roc_score: 0.5296
Epoch 00002: val_auc_roc_score improved from 0.57011 to 0.65100, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 118s 431ms/step - loss: 2.4430 - auc_roc_score: 0.5296
- val_loss: 2.2593 - val_auc_roc_score: 0.6510
Epoch 3/30
274/274 [==============================] - ETA: 0s - loss: 2.1758 - auc_roc_score: 0.5592
Epoch 00003: val_auc_roc_score did not improve from 0.65100
274/274 [==============================] - 117s 427ms/step - loss: 2.1758 - auc_roc_score: 0.5592
- val_loss: 2.0615 - val_auc_roc_score: 0.6410
Epoch 4/30
274/274 [==============================] - ETA: 0s - loss: 2.0047 - auc_roc_score: 0.5811
Epoch 00004: val_auc_roc_score improved from 0.65100 to 0.67663, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 433ms/step - loss: 2.0047 - auc_roc_score: 0.5811
- val_loss: 1.9236 - val_auc_roc_score: 0.6766
Epoch 5/30
274/274 [==============================] - ETA: 0s - loss: 1.8730 - auc_roc_score: 0.5945
Epoch 00005: val_auc_roc_score improved from 0.67663 to 0.69209, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 118s 432ms/step - loss: 1.8730 - auc_roc_score: 0.5945
- val_loss: 1.8109 - val_auc_roc_score: 0.6921
Epoch 6/30
274/274 [==============================] - ETA: 0s - loss: 1.7670 - auc_roc_score: 0.5968
Epoch 00006: val_auc_roc_score improved from 0.69209 to 0.69925, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 433ms/step - loss: 1.7670 - auc_roc_score: 0.5968
- val_loss: 1.6841 - val_auc_roc_score: 0.6993
Epoch 7/30
274/274 [==============================] - ETA: 0s - loss: 1.6772 - auc_roc_score: 0.6237
Epoch 00007: val_auc_roc_score improved from 0.69925 to 0.70340, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 433ms/step - loss: 1.6772 - auc_roc_score: 0.6237
- val_loss: 1.6626 - val_auc_roc_score: 0.7034
Epoch 8/30
274/274 [==============================] - ETA: 0s - loss: 1.6145 - auc_roc_score: 0.6273
Epoch 00008: val_auc_roc_score improved from 0.70340 to 0.70986, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 433ms/step - loss: 1.6145 - auc_roc_score: 0.6273
- val_loss: 1.6355 - val_auc_roc_score: 0.7099
Epoch 9/30
274/274 [==============================] - ETA: 0s - loss: 1.5378 - auc_roc_score: 0.6530
Epoch 00009: val_auc_roc_score improved from 0.70986 to 0.71051, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 118s 432ms/step - loss: 1.5378 - auc_roc_score: 0.6530
- val_loss: 1.5875 - val_auc_roc_score: 0.7105
Epoch 10/30
274/274 [==============================] - ETA: 0s - loss: 1.4823 - auc_roc_score: 0.6623
Epoch 00010: val_auc_roc_score improved from 0.71051 to 0.71608, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 433ms/step - loss: 1.4823 - auc_roc_score: 0.6623
- val_loss: 1.4694 - val_auc_roc_score: 0.7161
Epoch 11/30
274/274 [==============================] - ETA: 0s - loss: 1.4278 - auc_roc_score: 0.6767
Epoch 00011: val_auc_roc_score improved from 0.71608 to 0.71840, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 434ms/step - loss: 1.4278 - auc_roc_score: 0.6767
- val_loss: 1.4149 - val_auc_roc_score: 0.7184
Epoch 12/30
274/274 [==============================] - ETA: 0s - loss: 1.3788 - auc_roc_score: 0.6894
Epoch 00012: val_auc_roc_score improved from 0.71840 to 0.72295, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 434ms/step - loss: 1.3788 - auc_roc_score: 0.6894
```

```
- val_loss: 1.3435 - val_auc_roc_score: 0.7230
Epoch 13/30
274/274 [==============================] - ETA: 0s - loss: 1.3407 - auc_roc_score: 0.6936
Epoch 00013: val_auc_roc_score improved from 0.72295 to 0.72704, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 434ms/step - loss: 1.3407 - auc_roc_score: 0.6936
- val_loss: 1.4450 - val_auc_roc_score: 0.7270
Epoch 14/30
274/274 [==============================] - ETA: 0s - loss: 1.3042 - auc_roc_score: 0.6988
Epoch 00014: val_auc_roc_score improved from 0.72704 to 0.72912, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 436ms/step - loss: 1.3042 - auc_roc_score: 0.6988
- val_loss: 1.3099 - val_auc_roc_score: 0.7291
Epoch 15/30
274/274 [==============================] - ETA: 0s - loss: 1.2606 - auc_roc_score: 0.7129
Epoch 00015: val_auc_roc_score improved from 0.72912 to 0.73073, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 120s 436ms/step - loss: 1.2606 - auc_roc_score: 0.7129
- val_loss: 1.2616 - val_auc_roc_score: 0.7307
Epoch 16/30
274/274 [==============================] - ETA: 0s - loss: 1.2283 - auc_roc_score: 0.7190
Epoch 00016: val_auc_roc_score did not improve from 0.73073
274/274 [==============================] - 118s 429ms/step - loss: 1.2283 - auc_roc_score: 0.7190
- val_loss: 1.2143 - val_auc_roc_score: 0.7302
Epoch 17/30
274/274 [==============================] - ETA: 0s - loss: 1.1978 - auc_roc_score: 0.7242
Epoch 00017: val_auc_roc_score improved from 0.73073 to 0.73410, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 433ms/step - loss: 1.1978 - auc_roc_score: 0.7242
- val_loss: 1.2558 - val_auc_roc_score: 0.7341
Epoch 18/30
274/274 [==============================] - ETA: 0s - loss: 1.1664 - auc_roc_score: 0.7325
Epoch 00018: val_auc_roc_score improved from 0.73410 to 0.73560, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 433ms/step - loss: 1.1664 - auc_roc_score: 0.7325
- val_loss: 1.2271 - val_auc_roc_score: 0.7356
Epoch 19/30
274/274 [==============================] - ETA: 0s - loss: 1.1355 - auc_roc_score: 0.7403
Epoch 00019: val_auc_roc_score improved from 0.73560 to 0.73875, saving model to
weights_copy.best.hdf5
274/274 [==============================] - 119s 434ms/step - loss: 1.1355 - auc_roc_score: 0.7403
- val_loss: 1.1650 - val_auc_roc_score: 0.7387
Epoch 20/30
274/274 [==============================] - ETA: 0s - loss: 1.1070 - auc_roc_score: 0.7501
Epoch 00020: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 117s 428ms/step - loss: 1.1070 - auc_roc_score: 0.7501
- val_loss: 1.2207 - val_auc_roc_score: 0.7318
Epoch 21/30
274/274 [==============================] - ETA: 0s - loss: 1.0837 - auc_roc_score: 0.7549
Epoch 00021: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 117s 428ms/step - loss: 1.0837 - auc_roc_score: 0.7549
- val_loss: 1.3183 - val_auc_roc_score: 0.7317
Epoch 22/30
274/274 [==============================] - ETA: 0s - loss: 1.0595 - auc_roc_score: 0.7614
Epoch 00022: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 117s 429ms/step - loss: 1.0595 - auc_roc_score: 0.7614
- val_loss: 1.1968 - val_auc_roc_score: 0.7315
Epoch 23/30
274/274 [==============================] - ETA: 0s - loss: 1.0348 - auc_roc_score: 0.7714
Epoch 00023: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 117s 427ms/step - loss: 1.0348 - auc_roc_score: 0.7714
- val_loss: 1.0735 - val_auc_roc_score: 0.7347
Epoch 24/30
274/274 [==============================] - ETA: 0s - loss: 1.0071 - auc_roc_score: 0.7860
Epoch 00024: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 118s 430ms/step - loss: 1.0071 - auc_roc_score: 0.7860
- val_loss: 1.2051 - val_auc_roc_score: 0.7262
Epoch 25/30
274/274 [==============================] - ETA: 0s - loss: 0.9788 - auc_roc_score: 0.8022
Epoch 00025: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 118s 429ms/step - loss: 0.9788 - auc_roc_score: 0.8022
- val_loss: 1.0536 - val_auc_roc_score: 0.7240
Epoch 26/30
274/274 [==============================] - ETA: 0s - loss: 0.9481 - auc_roc_score: 0.8177
Epoch 00026: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 118s 430ms/step - loss: 0.9481 - auc_roc_score: 0.8177
- val_loss: 1.2340 - val_auc_roc_score: 0.7208
```

```
Epoch 27/30
274/274 [==============================] - ETA: 0s - loss: 0.9154 - auc_roc_score: 0.8381
Epoch 00027: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 118s 432ms/step - loss: 0.9154 - auc_roc_score: 0.8381
- val_loss: 1.2068 - val_auc_roc_score: 0.7120
Epoch 28/30
274/274 [==============================] - ETA: 0s - loss: 0.8713 - auc_roc_score: 0.8632
Epoch 00028: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 118s 432ms/step - loss: 0.8713 - auc_roc_score: 0.8632
- val_loss: 1.0804 - val_auc_roc_score: 0.6978
Epoch 29/30
274/274 [==============================] - ETA: 0s - loss: 0.8320 - auc_roc_score: 0.8839
Epoch 00029: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 118s 430ms/step - loss: 0.8320 - auc_roc_score: 0.8839
- val_loss: 1.0555 - val_auc_roc_score: 0.6933
Epoch 30/30
274/274 [==============================] - ETA: 0s - loss: 0.7784 - auc_roc_score: 0.9107
Epoch 00030: val_auc_roc_score did not improve from 0.73875
274/274 [==============================] - 118s 430ms/step - loss: 0.7784 - auc_roc_score: 0.9107
- val_loss: 1.1530 - val_auc_roc_score: 0.6748
```

Out[42]:

```
<tensorflow.python.keras.callbacks.History at 0x7fde4314cda0>
```

In [43]:

```
### load the weight from the saved file
model.load_weights("weights_copy.best.hdf5")
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001,decay = 1e-
4),metrics=[auc_roc_score])
```

In [44]:

```
### 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 [45]:

```
print("ROC-AUC for test data: %0.3f"%roc_auc_score(y_test,y_predict_test))
print("ROC-AUC for CV data: %0.3f"%roc_auc_score(y_cv,y_predict_cv))
print("ROC-AUC for train data: %0.3f"%roc_auc_score(y_train,y_predict_train))
```

```
ROC-AUC for test data: 0.734
ROC-AUC for CV data: 0.738
ROC-AUC for train data: 0.776
```

In [46]:

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

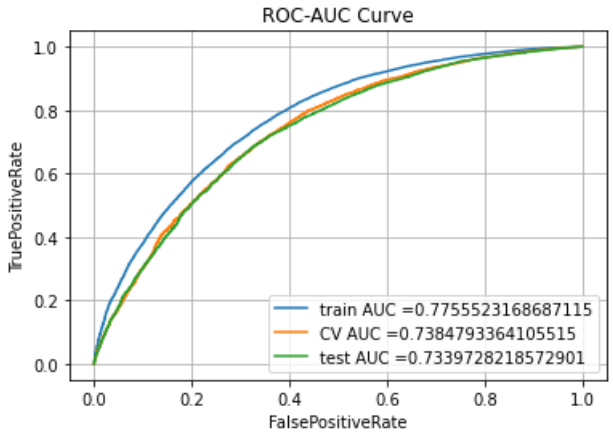
In [47]:

```
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 [48]:

```
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 [ ]: