Assignment : 14

Referrence:

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

https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values

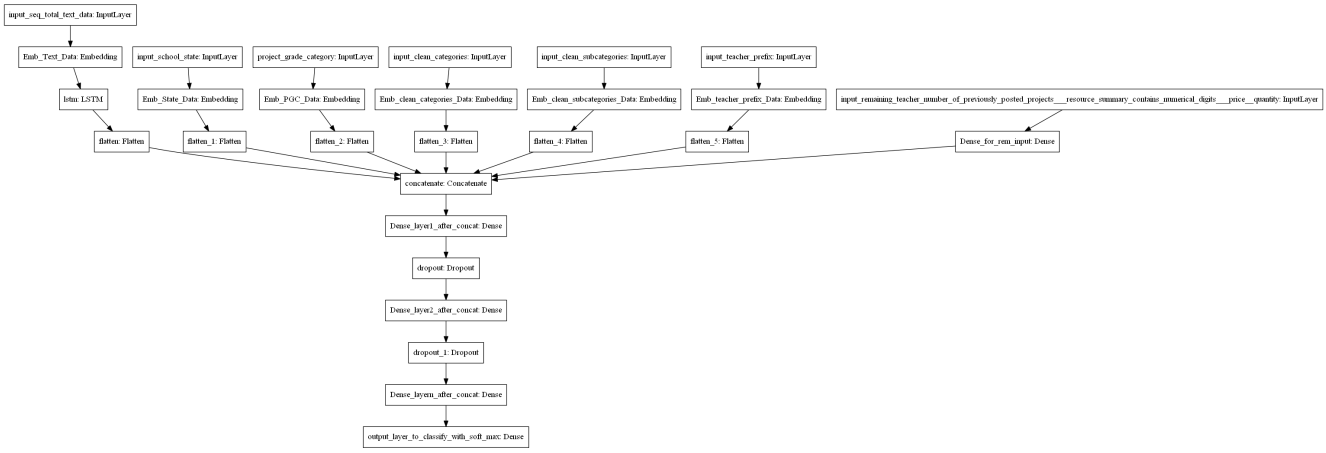
https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/80807
https://www.tensorflow.org/api_docs/python/tf/py_function

https://stackoverflow.com/questions/56156260/how-to-find-and-remove-words-which-have-low-and-high-idf-values

```
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



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.

In [ ]:

```
# 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.

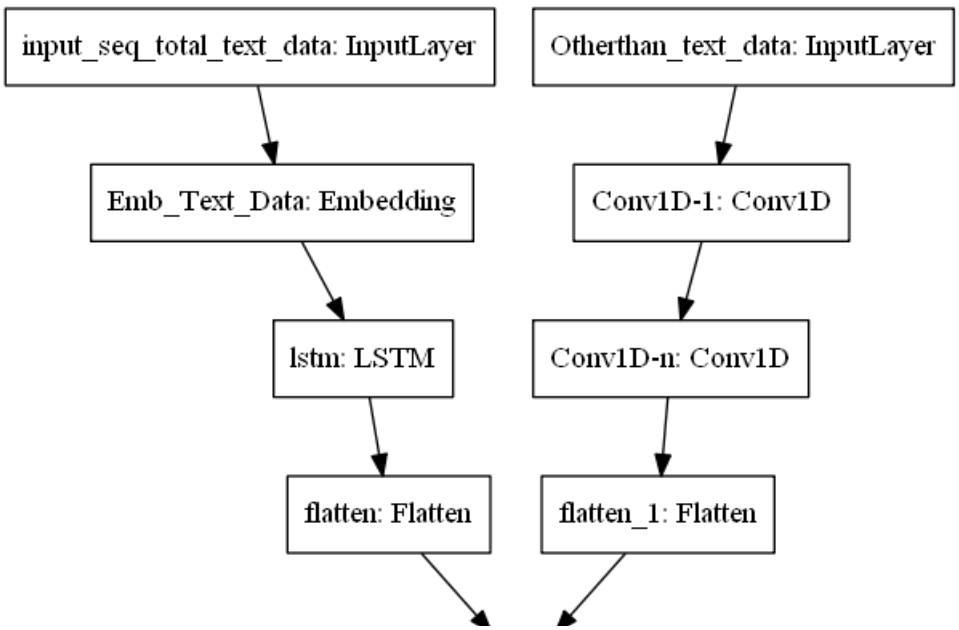
1. Train the TF-IDF on the Train data

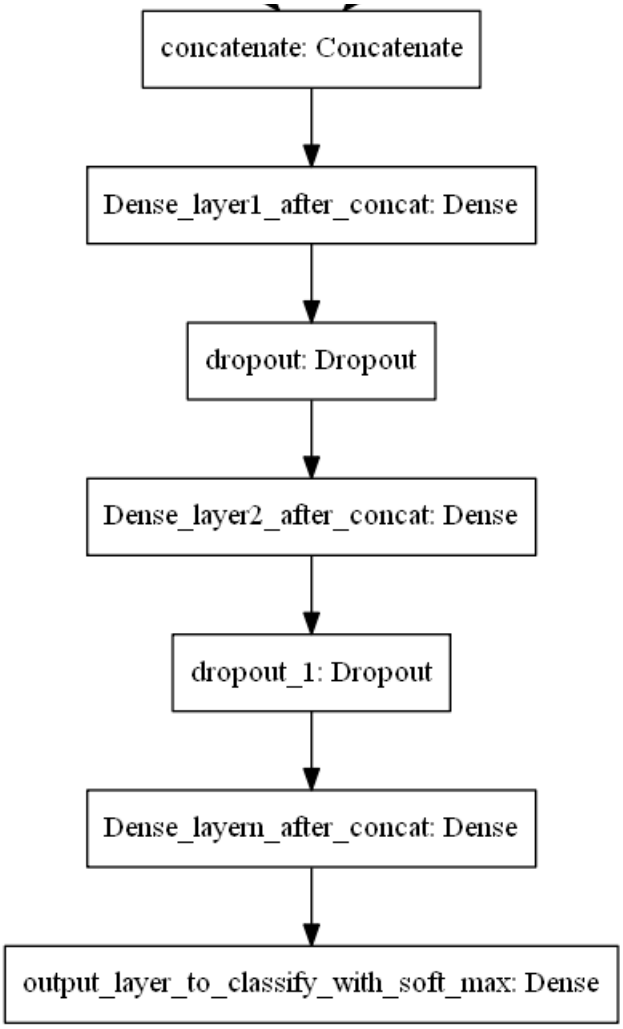
2. Get the idf value for each word we have in the train data.

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 plots 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 total 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 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>

Importing libraries

In [3]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences
```

```
import pickle
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Flatten, Concatenate, Dense, Dropout, C
onv1D
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from datetime import datetime
import matplotlib.pyplot as plt
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
time: 9.14 ms
```

Connecting to Google Drive to import Data files

In [4]:

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

```
Mounted at /content/drive
time: 24.1 s
```

In [5]:

```
%cd /content/drive/My Drive/Applied AI Course/Assignments/25. LSTM on Donors Choose
```

```
/content/drive/My Drive/Applied AI Course/Assignments/25. LSTM on Donors Choose
time: 404 ms
```

In [6]:

```
data = pd.read_csv('preprocessed_data.csv')
```

```
time: 4.63 s
```

Model-1

In [7]:

```
data.columns
```

Out[7]:

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


time: 10 ms
```

In [8]:

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.2, stratify=y_train)

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
```

```
print(X_test.shape, y_test.shape)
```

```
(69918, 8) (69918,)
(17480, 8) (17480,)
(21850, 8) (21850,)
time: 151 ms
```

Essay Preparation

In [9]:

```
len_list = []
for each_word in X_train['essay'].astype('str'):
  spl = each_word.split()
  len_list.append(len(spl))

print("The length of the list", len(len_list))
max_len = max(len_list)
print("Maximum Length", max_len)
```

```
The length of the list 69918
Maximum Length 339
time: 632 ms
```

In [10]:

```
t = tf.keras.preprocessing.text.Tokenizer()
t.fit_on_texts(X_train['essay'])

vocab_size = len(t.word_index) + 1
print('vocab size: ', vocab_size)

encoded_train = t.texts_to_sequences(X_train['essay'])
encoded_cv = t.texts_to_sequences(X_cv['essay'])
encoded_test = t.texts_to_sequences(X_test['essay'])

padded_train = pad_sequences(encoded_train, maxlen=max_len, padding='post')
padded_cv = pad_sequences(encoded_cv, maxlen=max_len, padding='post')
padded_test = pad_sequences(encoded_test, maxlen=max_len, padding='post')
```

```
vocab size:  47394
time: 17.5 s
```

In [11]:

```
with open('glove_vectors', 'rb') as f:
  loader = pickle.load(f)
  embedding_matrix = np.zeros((vocab_size, 300)) # 47394 * 300
  for word, i in t.word_index.items():
    embedding_vector = loader.get(word)
    if embedding_vector is not None:
      embedding_matrix[i] = embedding_vector

print(embedding_matrix.shape)
```

```
(47394, 300)
time: 5.83 s
```

Categorical Features

teacher_prefix

In [12]:

```
teacher_prefix_vocab = X_train['teacher_prefix'].nunique()

le = LabelEncoder()
train_teacher_prefix = le.fit_transform(X_train['teacher_prefix'])
```

```
cv_teacher_prefix = le.transform(X_cv['teacher_prefix'])
test_teacher_prefix = le.transform(X_test['teacher_prefix'])
print(train_teacher_prefix.shape)
print(cv_teacher_prefix.shape)
print(test_teacher_prefix.shape)
```

```
(69918,)
(17480,)
(21850,)
time: 30.3 ms
```

school_state

In [13]:

```
school_state_vocab = X_train['school_state'].nunique()

train_school_state_prefix = le.fit_transform(X_train['school_state'])
cv_school_state_prefix = le.transform(X_cv['school_state'])
test_school_state_prefix = le.transform(X_test['school_state'])
```

```
time: 30.1 ms
```

project_grade_category

In [14]:

```
grade_vocab = X_train['project_grade_category'].nunique()

train_grade = le.fit_transform(X_train['project_grade_category'])
cv_grade = le.transform(X_cv['project_grade_category'])
test_grade = le.transform(X_test['project_grade_category'])
```

```
time: 29.2 ms
```

clean_categories

In [15]:

```
# reference: https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-bef
ore-values

subject_categories_vocab = X_train['clean_categories'].nunique()

train_subject_categories = le.fit_transform(X_train['clean_categories'])
X_cv['clean_categories'] = X_cv['clean_categories'].map(lambda s: '<unknown>' if s not in le.classe
s_ else s)
X_test['clean_categories'] = X_test['clean_categories'].map(lambda s: '<unknown>' if s not in le.cl
asses_ else s)
le.classes_ = np.append(le.classes_, '<unknown>')
cv_subject_categories = le.transform(X_cv['clean_categories'])
test_subject_categories = le.transform(X_test['clean_categories'])
```

```
time: 313 ms
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

clean_subcategories

In [16]:

```
# reference: https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values

subject_subcategories_vocab = X_train['clean_subcategories'].nunique()

train_subject_subcategories = le.fit_transform(X_train['clean_subcategories'])
X_cv['clean_subcategories'] =X_cv['clean_subcategories'].map(lambda s: '<unknown>' if s not in le.classes_ else s)
X_test['clean_subcategories'] = X_test['clean_subcategories'].map(lambda s: '<unknown>' if s not in le.classes_ else s)
le.classes_ = np.append(le.classes_, '<unknown>')
cv_subject_subcategories = le.transform(X_cv['clean_subcategories'])
test_subject_subcategories = le.transform(X_test['clean_subcategories'])
```

time: 571 ms

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

all numerical features

In [17]:

```
train_price = X_train['price'].values.reshape(-1,1)
train_previously_posted = X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

cv_price = X_cv['price'].values.reshape(-1,1)
cv_previously_posted = X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

test_price = X_test['price'].values.reshape(-1,1)
test_previously_posted = X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

all_numeric_train = np.concatenate((train_price, train_previously_posted), axis=1)
all_numeric_cv = np.concatenate((cv_price, cv_previously_posted), axis=1)
all_numeric_test = np.concatenate((test_price, test_previously_posted), axis=1)

ss = StandardScaler()
train_numeric_norm = ss.fit_transform(all_numeric_train)
cv_numeric_norm = ss.transform(all_numeric_cv)
test_numeric_norm = ss.transform(all_numeric_test)
```

time: 29.2 ms

Model Architecture

In [18]:

```
init = tf.keras.initializers.he_normal()
regular = tf.keras.regularizers.l2(l2=0.01)

# layer architecture for essay
inp_essay = Input((max_len,))
essay_emb = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_len, trainable=False)(inp_essay)
lstm1 = LSTM(units=128, return_sequences=True)(essay_emb)
flat1 = Flatten()(lstm1)

# layer architecture for teacher_prefix
inp_teacher_prefix = Input((1,))
teacher_prefix_emb = Embedding(teacher_prefix_vocab, min(teacher_prefix_vocab,50), trainable=True)(inp_teacher_prefix)
flat2 = Flatten()(teacher_prefix_emb)
```

```
# layer architecture for school_state
inp_school_state = Input((1,))
school_state_emb = Embedding(school_state_vocab, min(school_state_vocab,50), trainable=True)
(inp_school_state)
flat3 = Flatten()(school_state_emb)

# layer architecture for project_grade_category
inp_grade = Input((1,))
grade_emb = Embedding(grade_vocab, min(grade_vocab,50), trainable=True)(inp_grade)
flat4 = Flatten()(grade_emb)

# layer architecture for clean_categories
inp_subject_categories = Input((1,))
subject_categories_emb = Embedding(subject_categories_vocab+1, min(subject_categories_vocab,50),
trainable=True)(inp_subject_categories)
flat5 = Flatten()(subject_categories_emb)

# layer architecture for clean_subcategories
inp_subject_subcategories = Input((1,))
subject_subcategories_emb = Embedding(subject_subcategories_vocab+1,
min(subject_subcategories_vocab,50), trainable=True)(inp_subject_subcategories)
flat6 = Flatten()(subject_subcategories_emb)

# layer architecture for numeric values
inp_numeric = Input((2,))
dense = Dense(units=96, activation='relu', kernel_regularizer=regular, kernel_initializer=init)(inp
_numeric)



# concatenate them
concat1 = Concatenate()([flat1, flat2, flat3, flat4, flat5, flat6, dense])

# dense layer
dense1 = Dense(units=128, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(c
oncat1)

# dropout layer
drop1 = Dropout(rate=0.5)(dense1)

# dense layer
dense2 = Dense(units=256, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(d
rop1)

# dropout layer
drop2 = Dropout(rate=0.5)(dense2)

# dense layer
dense3 = Dense(units=64, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(dr
op2)

# output layer
out_layer = Dense(units=2, activation='softmax')(dense3)

# determining input and output
model1 = Model(
    inputs=[inp_essay, inp_teacher_prefix, inp_school_state, inp_grade, inp_subject_categories, inp
_subject_subcategories, inp_numeric],
    outputs=out_layer)

model1.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 339)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 339, 300)     14218200    input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 1)]          0
```

```
input_5 (InputLayer)            [(None, 1)]          0
_____________________________________________________________________________________________
input_6 (InputLayer)            [(None, 1)]          0
_____________________________________________________________________________________________
lstm (LSTM)                     (None, 339, 128)     219648      embedding[0][0]
_____________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 5)         25          input_2[0][0]
_____________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 50)        2550        input_3[0][0]
_____________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 4)         16          input_4[0][0]
_____________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 50)        2550        input_5[0][0]
_____________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 50)        19400       input_6[0][0]
_____________________________________________________________________________________________
input_7 (InputLayer)            [(None, 2)]          0
_____________________________________________________________________________________________
flatten (Flatten)               (None, 43392)        0           lstm[0][0]
_____________________________________________________________________________________________
flatten_1 (Flatten)             (None, 5)            0           embedding_1[0][0]
_____________________________________________________________________________________________
flatten_2 (Flatten)             (None, 50)           0           embedding_2[0][0]
_____________________________________________________________________________________________
flatten_3 (Flatten)             (None, 4)            0           embedding_3[0][0]
_____________________________________________________________________________________________
flatten_4 (Flatten)             (None, 50)           0           embedding_4[0][0]
_____________________________________________________________________________________________
flatten_5 (Flatten)             (None, 50)           0           embedding_5[0][0]
_____________________________________________________________________________________________
dense (Dense)                   (None, 96)           288         input_7[0][0]
_____________________________________________________________________________________________
concatenate (Concatenate)       (None, 43647)        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)          5586944     concatenate[0][0]
_____________________________________________________________________________________________
dropout (Dropout)               (None, 128)          0           dense_1[0][0]
_____________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          33024       dropout[0][0]
_____________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           dense_2[0][0]
_____________________________________________________________________________________________
dense_3 (Dense)                 (None, 64)           16448       dropout_1[0][0]
_____________________________________________________________________________________________
dense_4 (Dense)                 (None, 2)            130         dense_3[0][0]
=============================================================================================
Total params: 20,099,223
Trainable params: 5,881,023
Non-trainable params: 14,218,200
_____________________________________________________________________________________________
time: 6.4 s
```

In [19]:

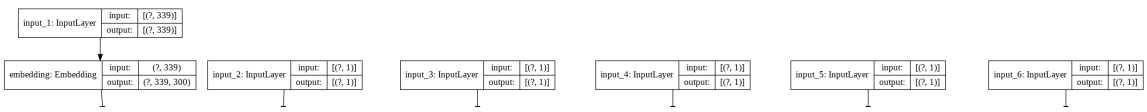
```
!mkdir plot
```

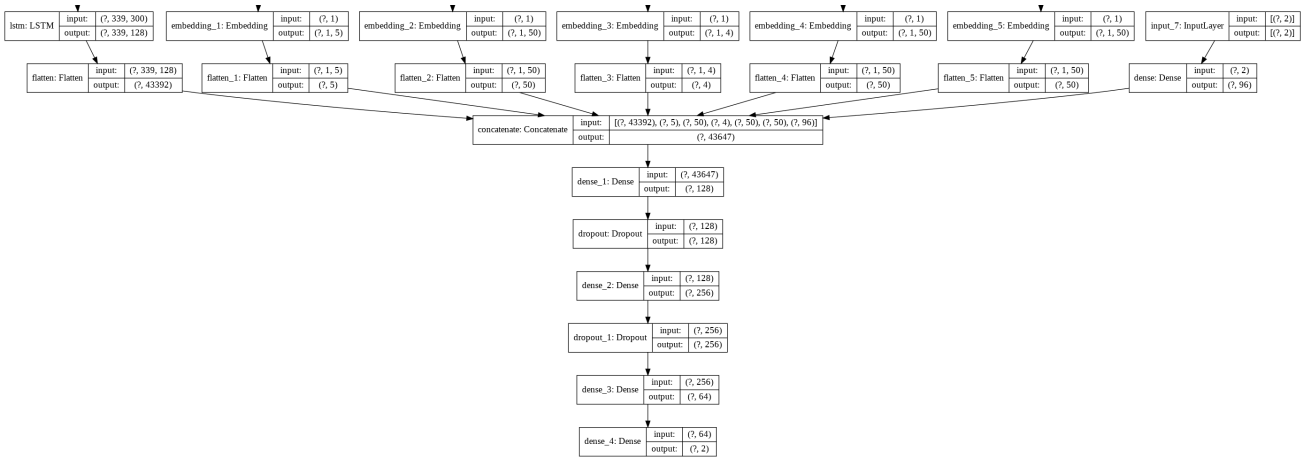
```
time: 144 ms
```

In [20]:

```
plot = 'plot/model_1.png'
tf.keras.utils.plot_model(model1, to_file=plot, show_shapes=True)
```

Out[20]:





time: 510 ms

In [21]:

```
train_data = [padded_train, train_teacher_prefix, train_school_state_prefix, train_grade,
train_subject_categories,
              train_subject_subcategories, train_numeric_norm]
cv_data = [padded_cv, cv_teacher_prefix, cv_school_state_prefix, cv_grade, cv_subject_categories,
              cv_subject_subcategories, cv_numeric_norm]
test_data = [padded_test, test_teacher_prefix, test_school_state_prefix, test_grade, test_subject_c
ategories,
              test_subject_subcategories, test_numeric_norm]

train_label = tf.keras.utils.to_categorical(y_train, 2)
cv_label = tf.keras.utils.to_categorical(y_cv, 2)
test_label = tf.keras.utils.to_categorical(y_test, 2)
```

time: 10.5 ms

In [22]:

```
# reference: https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/80807
# reference: https://www.tensorflow.org/api_docs/python/tf/py_function

def auc(y_true, y_pred):
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)
```

time: 1.46 ms

In [23]:

```
!mkdir model
```

time: 142 ms

In [24]:

```
filepath = "model/best_model_1.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc', verbose=1, save_best_only=True,
mode='max')

es = EarlyStopping(monitor='val_auc', patience=2, mode='max', verbose=1, restore_best_weights=True)

log_dir = "model_1" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_grap
h=True, write_grads=True)

cb = [es, checkpoint, tensorboard_callback]
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```
time: 202 ms

In [25]:

```
model1.compile(optimizer=tf.keras.optimizers.Adam(), loss='categorical_crossentropy', metrics=[auc
])
```

time: 22.1 ms

In [26]:

```
history1 = model1.fit(train_data, train_label, batch_size=512, epochs=10, verbose=1,
validation_data=(cv_data, cv_label), callbacks=cb)
```

```
Epoch 1/10
  1/137 [..............................] - ETA: 0s - loss: 11.4209 - auc:
0.5383WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after 2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
137/137 [==============================] - ETA: 0s - loss: 5.9997 - auc: 0.6394
Epoch 00001: val_auc improved from -inf to 0.72328, saving model to model/best_model_1.h5
137/137 [==============================] - 58s 425ms/step - loss: 5.9997 - auc: 0.6394 - val_loss:
3.5117 - val_auc: 0.7233
Epoch 2/10
137/137 [==============================] - ETA: 0s - loss: 2.6220 - auc: 0.7249
Epoch 00002: val_auc improved from 0.72328 to 0.74449, saving model to model/best_model_1.h5
137/137 [==============================] - 59s 430ms/step - loss: 2.6220 - auc: 0.7249 - val_loss:
2.0311 - val_auc: 0.7445
Epoch 3/10
137/137 [==============================] - ETA: 0s - loss: 1.7171 - auc: 0.7424
Epoch 00003: val_auc improved from 0.74449 to 0.74993, saving model to model/best_model_1.h5
137/137 [==============================] - 58s 426ms/step - loss: 1.7171 - auc: 0.7424 - val_loss:
1.4813 - val_auc: 0.7499
Epoch 4/10
137/137 [==============================] - ETA: 0s - loss: 1.3368 - auc: 0.7524
Epoch 00004: val_auc improved from 0.74993 to 0.75329, saving model to model/best_model_1.h5
137/137 [==============================] - 58s 425ms/step - loss: 1.3368 - auc: 0.7524 - val_loss:
1.2072 - val_auc: 0.7533
Epoch 5/10
137/137 [==============================] - ETA: 0s - loss: 1.1075 - auc: 0.7594
Epoch 00005: val_auc improved from 0.75329 to 0.75583, saving model to model/best_model_1.h5
137/137 [==============================] - 58s 426ms/step - loss: 1.1075 - auc: 0.7594 - val_loss:
1.0237 - val_auc: 0.7558
Epoch 6/10
137/137 [==============================] - ETA: 0s - loss: 0.9407 - auc: 0.7674
Epoch 00006: val_auc did not improve from 0.75583
137/137 [==============================] - 56s 412ms/step - loss: 0.9407 - auc: 0.7674 - val_loss:
0.8743 - val_auc: 0.7530
Epoch 7/10
137/137 [==============================] - ETA: 0s - loss: 0.8110 - auc: 0.7705
Epoch 00007: val_auc improved from 0.75583 to 0.75804, saving model to model/best_model_1.h5
137/137 [==============================] - 58s 426ms/step - loss: 0.8110 - auc: 0.7705 - val_loss:
0.7626 - val_auc: 0.7580
Epoch 8/10
137/137 [==============================] - ETA: 0s - loss: 0.7097 - auc: 0.7782
Epoch 00008: val_auc did not improve from 0.75804
137/137 [==============================] - 57s 413ms/step - loss: 0.7097 - auc: 0.7782 - val_loss:
0.6793 - val_auc: 0.7580
Epoch 9/10
137/137 [==============================] - ETA: 0s - loss: 0.6315 - auc: 0.7833Restoring model wei
ghts from the end of the best epoch.

Epoch 00009: val_auc did not improve from 0.75804
137/137 [==============================] - 57s 414ms/step - loss: 0.6315 - auc: 0.7833 - val_loss:
0.6123 - val_auc: 0.7544
Epoch 00009: early stopping
time: 8min 52s
```

In [27]:

```
%tensorboard --logdir model_120201013-024112
```

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

Model - 2

In [28]:

```
# reference: https://stackoverflow.com/questions/56156260/how-to-find-and-remove-words-which-have-low-and-high-idf-values

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
train_tf = vectorizer.fit(data['essay'].values)
idf_scores = train_tf.idf_
print("Length if idf_scores ",len(idf_scores))
```

```
Length if idf_scores  56345
time: 12.2 s
```

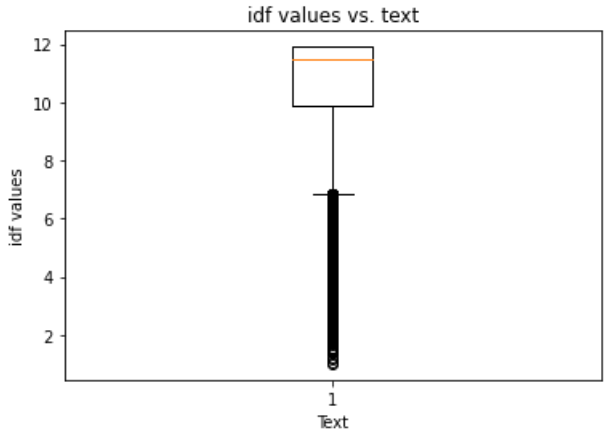
Analyzing idf values

In [29]:

```
plt.boxplot(idf_scores)
plt.title('idf values vs. text')
plt.xlabel('Text')
plt.ylabel('idf values')
```

Out[29]:

```
Text(0, 0.5, 'idf values')
```




```
time: 192 ms
```

In [45]:

```
print("0 to 10 percentile values")
for i in range(0,11,1):
  print(i, ' percentile value is: ', np.percentile(idf_scores, i))
if i == 10:
  print("\n20 to 100 percentile values")
  for i in range(20,110,10):
    print(i, ' percentile value is: ', np.percentile(idf_scores, i))
```

```
0 to 10 percentile values
0  percentile value is:  1.0077093449425296
1  percentile value is:  4.268197117636996
2  percentile value is:  5.106888022296025
3  percentile value is:  5.629379083706747
4  percentile value is:  6.082237671657471
5  percentile value is:  6.478892150083481
6  percentile value is:  6.802292305137341
```

```
7  percentile value is:  7.087956213432885
8  percentile value is:  7.3491115315512365
9  percentile value is:  7.577504438751591
10  percentile value is:  7.773071222295567

20 to 100 percentile values
20  percentile value is:  9.343288421576386
30  percentile value is:  10.298799866603822
40  percentile value is:  10.991947047163766
50  percentile value is:  11.502772670929756
60  percentile value is:  11.502772670929756
70  percentile value is:  11.908237779037922
80  percentile value is:  11.908237779037922
90  percentile value is:  11.908237779037922
100  percentile value is:  11.908237779037922
time: 40.2 ms
```

Filtering low and high tfidf values indices { removing lower thrshold value = 6 and higher thrshold value = 11.5 }

In [30]:

```
filtered_indices = np.argwhere(((idf_scores>=4) & (idf_scores<=11.5)))
filtered_indices = [idx[0] for idx in filtered_indices]
# list of vocabulary from the vectorizer
vocabulary = train_tf.get_feature_names()
# preparing a set with filtered vocabulary
filtered_voc = {vocabulary[i] for i in filtered_indices}
len(filtered_voc)
```

Out[30]:

```
27345
```

```
time: 70.7 ms
```

In [31]:

```
filtered_text_list = []
for text in data['essay'].values:
  text_word_list = [word for word in text.split() if word in filtered_voc]
  filtered_text_list.append(' '.join(text_word_list))
len(filtered_text_list)
```

Out[31]:

```
109248
```

```
time: 3.04 s
```

Removing the essay feature and adding filtered essay feature with medium tfidf value words

In [32]:

```
data['top_essay'] = filtered_text_list
data = data.drop('essay', axis=1)
data.columns
```

Out[32]:

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

```
time: 54.5 ms
```

In [33]:

```
y = data['project_is_approved']
```

```
X = data.drop('project_is_approved', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.2, stratify=y_train)

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(69918, 8) (69918,)
(17480, 8) (17480,)
(21850, 8) (21850,)
time: 165 ms
```

Essay Preparation

In [34]:

```
len_list = []
for each_word in X_train['top_essay'].astype('str'):
  spl = each_word.split()
  len_list.append(len(spl))

print("The length of the list", len(len_list))
max_len = max(len_list)
print("Maximum Length", max_len)
```

```
The length of the list 69918
Maximum Length 184
time: 288 ms
```

In [35]:

```
t = tf.keras.preprocessing.text.Tokenizer()
t.fit_on_texts(X_train['top_essay'])

vocab_size = len(t.word_index) + 1
print('vocab size: ', vocab_size)

encoded_train = t.texts_to_sequences(X_train['top_essay'])
encoded_cv = t.texts_to_sequences(X_cv['top_essay'])
encoded_test = t.texts_to_sequences(X_test['top_essay'])

padded_train = pad_sequences(encoded_train, maxlen=max_len, padding='post')
padded_cv = pad_sequences(encoded_cv, maxlen=max_len, padding='post')
padded_test = pad_sequences(encoded_test, maxlen=max_len, padding='post')
```

```
vocab size:  27151
time: 8.36 s
```

In [36]:

```
with open('glove_vectors', 'rb') as f:
  loader = pickle.load(f)
  embedding_matrix = np.zeros((vocab_size, 300)) # 27151 * 300
  for word, i in t.word_index.items():
    embedding_vector = loader.get(word)
    if embedding_vector is not None:
      embedding_matrix[i] = embedding_vector

print(embedding_matrix.shape)
```

```
(27151, 300)
time: 573 ms
```

Architecture

In [37]:

```
# layer architecture for essay
init = tf.keras.initializers.he_normal()
init1 = tf.keras.initializers.glorot_normal()
regular = tf.keras.regularizers.l2(l2=0.01)

inp_topessay = Input((max_len,))
essay_emb = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_len, trainable
=False)(inp_topessay)
lstm1 = LSTM(units=100, return_sequences=True)(essay_emb)
flat = Flatten()(lstm1)

# concatenate them
concat1 = Concatenate()([flat, flat2, flat3, flat4, flat5, flat6, dense])

# dense layer
dense1 = Dense(units=128, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(c
oncat1)

# dropout layer
drop1 = Dropout(rate=0.5)(dense1)

# dense layer
dense2 = Dense(units=256, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(d
rop1)

# dropout layer
drop2 = Dropout(rate=0.5)(dense2)

# dense layer
dense3 = Dense(units=64, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(dr
op2)

# output layer
out_layer = Dense(units=2, activation='softmax')(dense3)

# determining input and output
model2 = Model(
    inputs=[inp_topessay, inp_teacher_prefix, inp_school_state, inp_grade, inp_subject_categories,
inp_subject_subcategories, inp_numeric],
    outputs=out_layer)

model2.summary()
```

```
Model: "functional_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_8 (InputLayer)            [(None, 184)]        0
__________________________________________________________________________________________________
embedding_6 (Embedding)         (None, 184, 300)     8145300     input_8[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 184, 100)     160400      embedding_6[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 5)         25          input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 50)        2550        input_3[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 4)         16          input_4[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 50)        2550        input_5[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 50)        19400       input_6[0][0]
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, 2)]          0
__________________________________________________________________________________________________
```

flatten_6 (Flatten)	(None, 18400)	0	lstm_1[0][0]
flatten_1 (Flatten)	(None, 5)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 50)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 4)	0	embedding_3[0][0]
flatten_4 (Flatten)	(None, 50)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 50)	0	embedding_5[0][0]
dense (Dense)	(None, 96)	288	input_7[0][0]
concatenate_1 (Concatenate)	(None, 18655)	0	flatten_6[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_5 (Dense)	(None, 128)	2387968	concatenate_1[0][0]
dropout_2 (Dropout)	(None, 128)	0	dense_5[0][0]
dense_6 (Dense)	(None, 256)	33024	dropout_2[0][0]
dropout_3 (Dropout)	(None, 256)	0	dense_6[0][0]
dense_7 (Dense)	(None, 64)	16448	dropout_3[0][0]
dense_8 (Dense)	(None, 2)	130	dense_7[0][0]

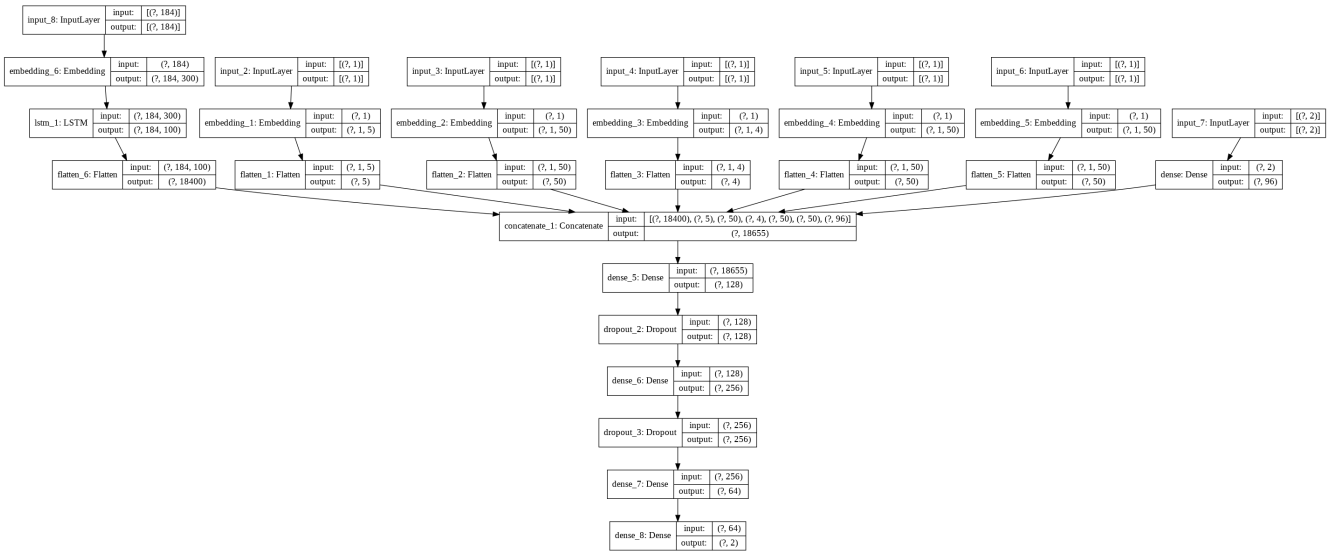
Total params: 10,768,099
Trainable params: 2,622,799
Non-trainable params: 8,145,300

time: 489 ms

In [38]:

```
plot = 'plot/model_2.png'
tf.keras.utils.plot_model(model2, to_file=plot, show_shapes=True)
```

Out[38]:



time: 332 ms

In [39]:

```
train_data = [padded_train, train_teacher_prefix, train_school_state_prefix, train_grade,
train_subject_categories,
```

```
              train_subject_subcategories, train_numeric_norm]
cv_data = [padded_cv, cv_teacher_prefix, cv_school_state_prefix, cv_grade, cv_subject_categories,
              cv_subject_subcategories, cv_numeric_norm]
test_data = [padded_test, test_teacher_prefix, test_school_state_prefix, test_grade, test_subject_c
ategories,
              test_subject_subcategories, test_numeric_norm]

train_label = tf.keras.utils.to_categorical(y_train, 2)
cv_label = tf.keras.utils.to_categorical(y_cv, 2)
test_label = tf.keras.utils.to_categorical(y_test, 2)
```

time: 7.98 ms

In [40]:

```
filepath = "model/best_model_2.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc', verbose=1, save_best_only=True,
mode='max')

es = EarlyStopping(monitor='val_auc', patience=2, mode='max', verbose=1)

log_dir = "model_2" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_grap
h=True, write_grads=True)

cb = [es, checkpoint, tensorboard_callback]
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
time: 10.5 ms
```

In [41]:

```
model2.compile(optimizer=tf.keras.optimizers.Adam(), loss='categorical_crossentropy', metrics=[auc
])
```

time: 15.5 ms

In [42]:

```
model2.fit(train_data, train_label, batch_size=512, epochs=10, verbose=1, validation_data=(cv_data,
cv_label), callbacks=cb)
```

```
Epoch 1/10
137/137 [==============================] - ETA: 0s - loss: 3.8818 - auc: 0.6405
Epoch 00001: val_auc improved from -inf to 0.70597, saving model to model/best_model_2.h5
137/137 [==============================] - 31s 226ms/step - loss: 3.8818 - auc: 0.6405 - val_loss:
1.5959 - val_auc: 0.7060
Epoch 2/10
137/137 [==============================] - ETA: 0s - loss: 1.0167 - auc: 0.6905
Epoch 00002: val_auc improved from 0.70597 to 0.71615, saving model to model/best_model_2.h5
137/137 [==============================] - 31s 225ms/step - loss: 1.0167 - auc: 0.6905 - val_loss:
0.6866 - val_auc: 0.7161
Epoch 3/10
137/137 [==============================] - ETA: 0s - loss: 0.5857 - auc: 0.7073
Epoch 00003: val_auc improved from 0.71615 to 0.71993, saving model to model/best_model_2.h5
137/137 [==============================] - 31s 224ms/step - loss: 0.5857 - auc: 0.7073 - val_loss:
0.5191 - val_auc: 0.7199
Epoch 4/10
137/137 [==============================] - ETA: 0s - loss: 0.4944 - auc: 0.7176
Epoch 00004: val_auc did not improve from 0.71993
137/137 [==============================] - 30s 217ms/step - loss: 0.4944 - auc: 0.7176 - val_loss:
0.4716 - val_auc: 0.7192
Epoch 5/10
137/137 [==============================] - ETA: 0s - loss: 0.4547 - auc: 0.7292
Epoch 00005: val_auc improved from 0.71993 to 0.72221, saving model to model/best_model_2.h5
137/137 [==============================] - 31s 226ms/step - loss: 0.4547 - auc: 0.7292 - val_loss:
0.4474 - val_auc: 0.7222
Epoch 6/10
137/137 [==============================] - ETA: 0s - loss: 0.4314 - auc: 0.7408
Epoch 00006: val_auc did not improve from 0.72221
137/137 [==============================] - 30s 217ms/step - loss: 0.4314 - auc: 0.7408 - val_loss:
```

```
0.4340 - val_auc: 0.7143
Epoch 7/10
137/137 [==============================] - ETA: 0s - loss: 0.4124 - auc: 0.7534
Epoch 00007: val_auc did not improve from 0.72221
137/137 [==============================] - 30s 217ms/step - loss: 0.4124 - auc: 0.7534 - val_loss:
0.4233 - val_auc: 0.7076
Epoch 00007: early stopping
```

Out[42]:

```
<tensorflow.python.keras.callbacks.History at 0x7f4ee49efdd8>
```

time: 3min 36s

In [43]:

```
%tensorboard --logdir model_220201013-030648
```

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

Model - 3

teacher_prefix

In [44]:

```
from sklearn.feature_extraction.text import CountVectorizer

countvect = CountVectorizer()
train_teacher_prefix_ohe = countvect.fit_transform(X_train['teacher_prefix'].values)
cv_teacher_prefix_ohe = countvect.transform(X_cv['teacher_prefix'].values)
test_teacher_prefix_ohe = countvect.transform(X_test['teacher_prefix'].values)
```

time: 350 ms

school_state

In [46]:

```
train_school_state_ohe = countvect.fit_transform(X_train['school_state'].values)
cv_school_state_ohe = countvect.transform(X_cv['school_state'].values)
test_school_state_ohe = countvect.transform(X_test['school_state'].values)
```

time: 326 ms

project_grade_category

In [47]:

```
train_grade_ohe = countvect.fit_transform(X_train['project_grade_category'].values)
cv_grade_ohe = countvect.transform(X_cv['project_grade_category'].values)
test_grade_ohe = countvect.transform(X_test['project_grade_category'].values)
```

time: 365 ms

clean_categories

In [48]:

```
train_clean_categories_ohe = countvect.fit_transform(X_train['clean_categories'].values)
cv_clean_categories_ohe = countvect.transform(X_cv['clean_categories'].values)
test_clean_categories_ohe = countvect.transform(X_test['clean_categories'].values)
```

time: 399 ms

time: 599 ms

clean_subcategories

In [49]:

```
train_clean_subcategories_ohe = countvect.fit_transform(X_train['clean_subcategories'].values)
cv_clean_subcategories_ohe = countvect.transform(X_cv['clean_subcategories'].values)
test_clean_subcategories_ohe = countvect.transform(X_test['clean_subcategories'].values)
```

time: 419 ms

price

In [50]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()
train_price_norm = normalizer.fit_transform(X_train['price'].values.reshape(1,-1))
cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

train_price_norm = train_price_norm.reshape(-1,1)
cv_price_norm = cv_price_norm.reshape(-1,1)
test_price_norm = test_price_norm.reshape(-1,1)
```

time: 7.79 ms

previously_posted_questions

In [51]:

```
train_previously_posted_norm =
normalizer.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,
-1))
cv_previously_posted_norm =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
test_previously_posted_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

train_previously_posted_norm = train_previously_posted_norm.reshape(-1,1)
cv_previously_posted_norm = cv_previously_posted_norm.reshape(-1,1)
test_previously_posted_norm = test_previously_posted_norm.reshape(-1,1)
```

time: 10.3 ms

Combining features

In [52]:

```
from scipy.sparse import hstack

non_text_train = hstack((train_teacher_prefix_ohe, train_school_state_ohe, train_grade_ohe,
train_clean_categories_ohe,
                train_clean_subcategories_ohe, train_price_norm, train_previously_posted_norm)).toc
ense() # todense returns matrix
non_text_cv = hstack((cv_teacher_prefix_ohe, cv_school_state_ohe, cv_grade_ohe, cv_clean_categories
_ohe, cv_clean_subcategories_ohe,
            cv_price_norm, cv_previously_posted_norm)).todense()
non_text_test = hstack((test_teacher_prefix_ohe, test_school_state_ohe, test_grade_ohe, test_clean_
categories_ohe,
                test_clean_subcategories_ohe, test_price_norm, test_previously_posted_norm)).todens
e()
```

time: 65.8 ms

In [53]:

```
print(non_text_train.shape)
print(non_text_cv.shape)
print(non_text_test.shape)
```

```
(69918, 101)
(17480, 101)
(21850, 101)
time: 1.49 ms
```

Layer Architecture

In [54]:

```
# layer architecture for essay
init = tf.keras.initializers.he_normal()
regular = tf.keras.regularizers.l2(l2=0.01)

inp_text = Input((max_len,))
essay_emb = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_len, trainable
=False)(inp_text)
lstm1 = LSTM(units=100, return_sequences=True)(essay_emb)
flat7 = Flatten()(lstm1)

# layer architecture for non-text
inp_nontext = Input((101,1))
conv1 = Conv1D(filters=64, kernel_size=3, activation='relu', kernel_initializer=init)(inp_nontext)
conv2 = Conv1D(filters=64, kernel_size=3, activation='relu', kernel_initializer=init)(conv1)
flat8 = Flatten()(conv2)

# concatenation of these two parts
concat2 = Concatenate()([flat7, flat8])

# dense layer
dense4 = Dense(units=128, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(c
oncat2)

# dropout layer
drop3 = Dropout(rate=0.5)(dense4)

# dense layer
dense5 = Dense(units=256, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(d
rop3)

# dropout layer
drop4 = Dropout(rate=0.5)(dense5)

# dense layer
dense6 = Dense(units=64, activation='relu', kernel_initializer=init, kernel_regularizer=regular)(dr
op4)

# output layer
out_layer2 = Dense(units=2, activation='softmax')(dense6)

# determining input and output
model3 = Model(inputs=[inp_text, inp_nontext], outputs=out_layer2)

model3.summary()
```

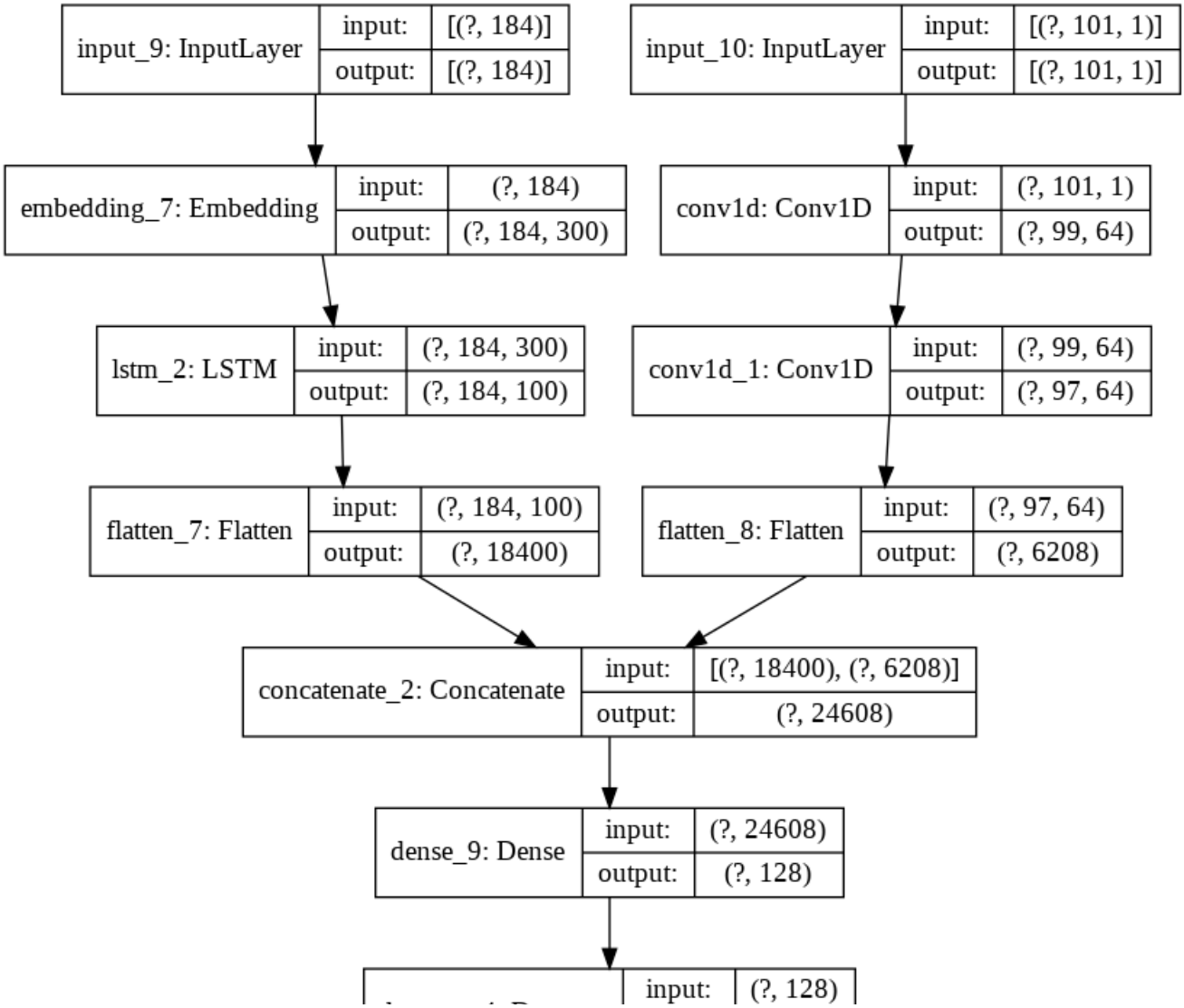
```
Model: "functional_5"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_9 (InputLayer)            [(None, 184)]        0
__________________________________________________________________________________________________
input_10 (InputLayer)           [(None, 101, 1)]     0
__________________________________________________________________________________________________
embedding_7 (Embedding)         (None, 184, 300)     8145300     input_9[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 99, 64)       256         input_10[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   (None, 184, 100)     160400      embedding_7[0][0]
```

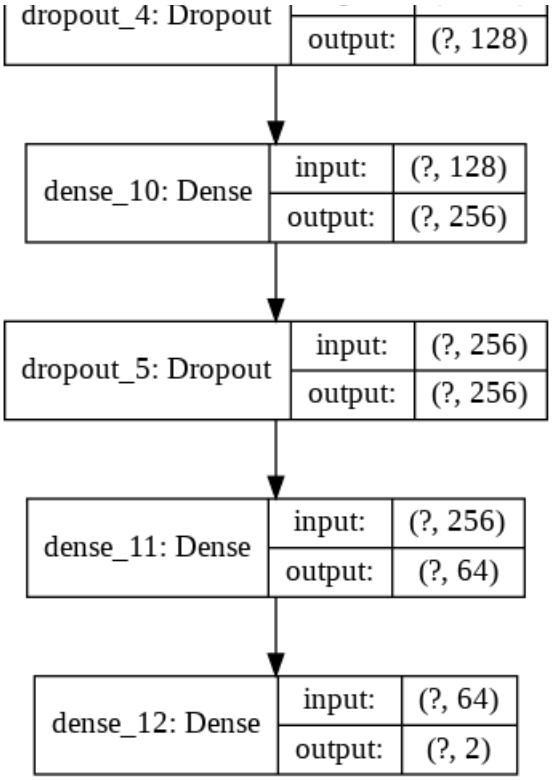
```
conv1d_1 (Conv1D)               (None, 97, 64)       12352       conv1d[0][0]
__________________________________________________________________________________________________
flatten_7 (Flatten)             (None, 18400)        0           lstm_2[0][0]
__________________________________________________________________________________________________
flatten_8 (Flatten)             (None, 6208)         0           conv1d_1[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 24608)        0           flatten_7[0][0]
                                                                 flatten_8[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 128)          3149952     concatenate_2[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 128)          0           dense_9[0][0]
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 256)          33024       dropout_4[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 256)          0           dense_10[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 64)           16448       dropout_5[0][0]
__________________________________________________________________________________________________
dense_12 (Dense)                (None, 2)            130         dense_11[0][0]
==================================================================================================
Total params: 11,517,862
Trainable params: 3,372,562
Non-trainable params: 8,145,300
__________________________________________________________________________________________________
time: 806 ms
```

In [55]:

```
plot = 'plot/model_3.png'
tf.keras.utils.plot_model(model3, to_file=plot, show_shapes=True)
```

Out[55]:





time: 249 ms

In [56]:

```
train_data = [padded_train, non_text_train]
cv_data = [padded_cv, non_text_cv]
test_data = [padded_test, non_text_test]

train_label = tf.keras.utils.to_categorical(y_train, 2)
cv_label = tf.keras.utils.to_categorical(y_cv, 2)
test_label = tf.keras.utils.to_categorical(y_test, 2)
```

time: 10.5 ms

In [57]:

```
def auc(y_true, y_pred):
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)
```

time: 1.18 ms

In [58]:

```
filepath = "model/best_model_3.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc', verbose=1, save_best_only=True,
mode='max')

es = EarlyStopping(monitor='val_auc', patience=2, mode='max', verbose=1)

log_dir = "model_3" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_grap
h=True, write_grads=True)

cb = [es, checkpoint, tensorboard_callback]
```

WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
time: 11.4 ms

In [59]:

```
model3.compile(optimizer= tf.keras.optimizers.Adam(), loss='categorical crossentropy',
```

```
metrics=[auc])
```

time: 20.4 ms

In [60]:

```
model3.fit(train_data, train_label, batch_size=512, epochs=10, verbose=1, validation_data=(cv_data,
cv_label), callbacks=cb)
```

```
Epoch 1/10
137/137 [==============================] - ETA: 0s - loss: 3.8213 - auc: 0.6264
Epoch 00001: val_auc improved from -inf to 0.69988, saving model to model/best_model_3.h5
137/137 [==============================] - 33s 240ms/step - loss: 3.8213 - auc: 0.6264 - val_loss:
1.5334 - val_auc: 0.6999
Epoch 2/10
137/137 [==============================] - ETA: 0s - loss: 0.9203 - auc: 0.6908
Epoch 00002: val_auc improved from 0.69988 to 0.71068, saving model to model/best_model_3.h5
137/137 [==============================] - 32s 235ms/step - loss: 0.9203 - auc: 0.6908 - val_loss:
0.5692 - val_auc: 0.7107
Epoch 3/10
137/137 [==============================] - ETA: 0s - loss: 0.4787 - auc: 0.7065
Epoch 00003: val_auc did not improve from 0.71068
137/137 [==============================] - 31s 226ms/step - loss: 0.4787 - auc: 0.7065 - val_loss:
0.4269 - val_auc: 0.7091
Epoch 4/10
137/137 [==============================] - ETA: 0s - loss: 0.4120 - auc: 0.7191
Epoch 00004: val_auc improved from 0.71068 to 0.71643, saving model to model/best_model_3.h5
137/137 [==============================] - 32s 232ms/step - loss: 0.4120 - auc: 0.7191 - val_loss:
0.4063 - val_auc: 0.7164
Epoch 5/10
137/137 [==============================] - ETA: 0s - loss: 0.3974 - auc: 0.7299
Epoch 00005: val_auc did not improve from 0.71643
137/137 [==============================] - 31s 226ms/step - loss: 0.3974 - auc: 0.7299 - val_loss:
0.4043 - val_auc: 0.7148
Epoch 6/10
137/137 [==============================] - ETA: 0s - loss: 0.3929 - auc: 0.7378
Epoch 00006: val_auc did not improve from 0.71643
137/137 [==============================] - 31s 226ms/step - loss: 0.3929 - auc: 0.7378 - val_loss:
0.4051 - val_auc: 0.7143
Epoch 00006: early stopping
```

Out[60]:

```
<tensorflow.python.keras.callbacks.History at 0x7f4ee47410f0>
```

time: 3min 13s

In [61]:

```
%tensorboard --logdir model_320201013-031855
```

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

Model Comparison (For best models)

In [62]:

```
from prettytable import PrettyTable

Model_Comparion = PrettyTable(['Model', 'AUC', 'val AUC', 'loss', 'val loss'])

Model_Comparion.add_row(['Model-1', 0.7833,0.7544,0.6315,0.6123])
Model_Comparion.add_row(['Model-2', 0.7534,0.7076,0.4124,0.4233])
Model_Comparion.add_row(['Model-3', 0.7378,0.7143,0.3929,0.4051])

print(Model_Comparion)
```

```
+---------+--------+---------+--------+----------+
|  Model  |  AUC   | val AUC |  loss  | val loss |
```

```
+---------+--------+---------+--------+----------+
| Model-1 | 0.7833 |  0.7544 | 0.6315 |  0.6123  |
| Model-2 | 0.7534 |  0.7076 | 0.4124 |  0.4233  |
| Model-3 | 0.7378 |  0.7143 | 0.3929 |  0.4051  |
+---------+--------+---------+--------+----------+
time: 17.1 ms
```

Models Observation

Observed that there is neither overfitting not underfitting except for Model-2 slight overfitting. val_auc dropped by 1% while accuracy is increasing.

All 3 models satisfied the requirements. i.e.; Both auc and Validation auc is more than 70% and validation auc for model-1 is 75%.

Deep Learning model, LSTM didn't much time compared with machine learning models. Here we need to work with hyper parameter tuning to avoid overfitting in model-2.