Assignment : 14

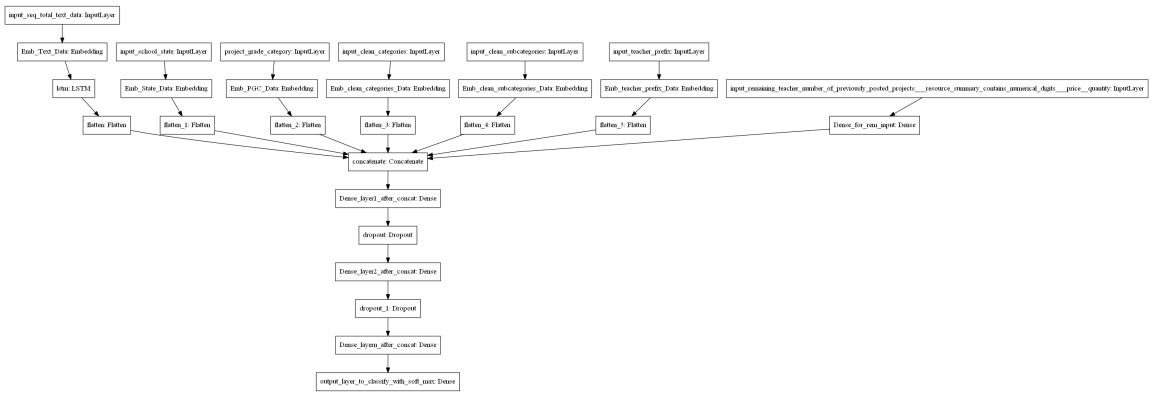
1. You can work with preprocessed_data.csv for the assignment. You can get the data from - Data folder
2. Load the data in your notebook.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check this and this for using auc as a metric
5. 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.
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 a separate pad and write your observations about them.
8. Make sure that you are using GPU to train the given models.

In [1]:
```
#you can use gdown modules to import dataset for the assignment
#for importing any file from drive to Colab you can write the syntax as !gdown --id
#you can run the below cell to import the required preprocessed data.csv file and g
```

In [2]:
```
#!gdown --id 1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
#!gdown --id 1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_
```

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_c** ---concatenate remaining columns and add a Dense layer after 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 [3]:
```
# https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-w
#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-1

In [1]:
```
# import all the libraries
#make sure that you import your libraries from tf.keras and not just keras
import tensorflow as tf
from tensorflow.keras.layers import Input,Dense,LSTM, Dense,Activation, Dropout, Em
from tensorflow.keras.layers import BatchNormalization, SpatialDropout1D

from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
import numpy as np
```

```
from sklearn.preprocessing import OrdinalEncoder
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPl
from tensorflow.keras.utils import plot_model
import datetime
from sklearn.metrics import roc_auc_score
from tensorflow.keras.utils import to_categorical
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
import seaborn as sns
import os
from scipy.sparse import hstack
%load_ext tensorboard
```

In [2]:
```
#read the csv file
import pandas as pd
df = pd.read_csv('preprocessed_data.csv')
```

In [3]:
```
# perform stratified train test split on the dataset
df.head()
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projec
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	
3	ga	mrs	grades_prek_2	
4	wa	mrs	grades_3_5	

In [4]:
```
y = df['project_is_approved'].values
X = df.drop(['project_is_approved'], axis = 1)
```

In [5]:
```
x_train, x_test, y_train, y_test = train_test_split(X , y, test_size = 0.25 , strat
```

In [6]:
```
y_train = y_train.reshape(-1,1)
y_test = y_test.reshape(-1,1)
```

In [7]:
```
x_train.head()
```

Out[7]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_
50356	wi	ms	grades_9_12	
108124	ca	ms	grades_3_5	
107343	mi	mrs	grades_prek_2	
19811	wa	mrs	grades_3_5	
68349	ct	ms	grades_prek_2	

In [8]:
```
#Using predefined glove_vectors where each word is represented using 300 dimensions
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
```

General Functions for all 3 models:

In [9]:
```
#Code is copied from: https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenizati
def encoding (category):

    ordinal_enc = OrdinalEncoder(categories='auto', handle_unknown='use_encoded_val
    ordinal_enc.fit(x_train[category].values.reshape(-1,1))
    train_category = ordinal_enc.transform(x_train[category].values.reshape(-1,1))
    test_category = ordinal_enc.transform(x_test[category].values.reshape(-1,1))

    return train_category, test_category
```

In [10]:
```
# https://towardsdatascience.com/deep-embeddings-for-categorical-variables-cat2vec-
def embedding_input_dim_output_dim(encoded_category):
    input_dimensions = int(len(np.unique(encoded_category)))
    output_dimensions = int(min(50, (input_dimensions// 2)+1))
    return input_dimensions,output_dimensions
```

In [11]:
```
# https://stackoverflow.com/questions/57574501/how-to-use-sklearn-auc-in-tensorflow
def auc_2(y_true, y_pred):
    return tf.numpy_function(auc_score, (y_true, y_pred), tf.double)

def auc_score(y_true, y_pred):
    if len(np.unique(y_true[:,0])) == 1:
        return 0.5
    else:
        return roc_auc_score(y_true, y_pred)
```

1.1 Text Vectorization

In [40]:
```
#since the data is already preprocessed, we can directly move to vectorization part
#first we will vectorize the text data
#for vectorization of text data in deep learning we use tokenizer, you can go throu
# https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenization-text-data-prep.ht
#https://stackoverflow.com/questions/51956000/what-does-keras-tokenizer-method-exac
# after text vectorization you should get train_padded_docs and test_padded_docs
```

1.1.1 Vectorizing essay

In [41]:
```
#Code is copied from: https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenizati

#oov_token = '<UNK>'
pad_type = 'post'
trunc_type = 'post'

tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')  # Removed '_'
tokenizer.fit_on_texts(x_train['essay'])


# Encode training data sentences into sequences
train_sequences = tokenizer.texts_to_sequences(x_train['essay'])

# Get max training sequence length
maxlen = max([len(x) for x in train_sequences])
```

```
# Pad the training sequences
train_padded_essay = pad_sequences(train_sequences, padding=pad_type, truncating=tr

test_sequences = tokenizer.texts_to_sequences(x_test['essay'])
test_padded_essay = pad_sequences(test_sequences, padding=pad_type, truncating=trun
```

In [42]:
```
vocab = tokenizer.word_index
#https://datascience.stackexchange.com/questions/93651/reason-for-adding-1-to-word-
vocab_size = len(vocab)+1 # word_index starts with index 1. Last word in word_index
                          # Normally index always starts at zero. So, to access the
                          # we need to specify vocab_size = len(vocab) + 1
```

In [43]:
```
vocab_size
```

Out[43]:
```
50344
```

In [44]:
```
print(train_padded_essay.shape)
print(test_padded_essay.shape)
```

```
(81936, 339)
(27312, 339)
```

In [45]:
```
#after getting the padded_docs you have to use predefined glove vectors to get 300
# we will be storing this data in form of an embedding matrix and will use it while
# Please go through following blog's 'Example of Using Pre-Trained GloVe Embedding'
# https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
```

In [46]:
```
embedding_matrix = np.zeros((vocab_size, 300))  # We used 300 dimensional pre-defin
for word, i in vocab.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

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

Out[47]:
```
(50344, 300)
```

1.2 Categorical feature Vectorization

In [27]:
```
# for model 1 and model 2, we have to assign a unique number to each feature in a p
# you can either use tokenizer,label encoder or ordinal encoder to perform the task
# label encoder gives an error for 'unseen values' (values present in test but not
# handle unseen values with label encoder - https://stackoverflow.com/a/56876351
# ordinal encoder also gives error with unseen values but you can use modify handle
# documentation of ordianl encoder https://scikit-learn.org/stable/modules/generate
# after categorical feature vectorization you will have column_train_data and colum
```

In [28]:
```
train_school_state_encoded , test_school_state_encoded = encoding('school_state')
train_teacher_prefix_encoded , test_teacher_prefix_encoded = encoding('teacher_pref
train_project_grade_category_encoded , test_project_grade_category_encoded = encodi
train_clean_categories_encoded , test_clean_categories_encoded = encoding('clean_ca
train_clean_subcategories_encoded , test_clean_subcategories_encoded = encoding('cl
```

In [29]:
```
print(train_school_state_encoded.shape)
print(train_teacher_prefix_encoded.shape)
print(train_project_grade_category_encoded.shape)
print(train_clean_categories_encoded.shape)
print(train_clean_subcategories_encoded.shape)
print(test_school_state_encoded.shape)
print(test_teacher_prefix_encoded.shape)
print(test_project_grade_category_encoded.shape)
print(test_clean_categories_encoded.shape)
print(test_clean_subcategories_encoded.shape)
```

```
(81936, 1)
(81936, 1)
(81936, 1)
(81936, 1)
(81936, 1)
(27312, 1)
(27312, 1)
(27312, 1)
(27312, 1)
(27312, 1)
```

1.3 Numerical feature Vectorization

In [30]:
```
# you have to standardise the numerical columns
# stack both the numerical features
#after numerical feature vectorization you will have numerical_data_train and numer
```

In [31]:
```
scaler = StandardScaler()
scaler.fit(x_train['price'].values.reshape(-1,1))

x_train_price_norm = scaler.transform(x_train['price'].values.reshape(-1,1))
x_test_price_norm = scaler.transform(x_test['price'].values.reshape(-1,1))


print("After vectorizations")
print(x_train_price_norm.shape, y_train.shape)
print(x_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(81936, 1) (81936, 1)
(27312, 1) (27312, 1)
====================================================================================
==================
```

In [32]:
```
scaler = StandardScaler()
scaler.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(-

x_train_teacher_num_previous_projects_norm =scaler.transform(x_train['teacher_numbe
x_test_teacher_num_previous_projects_norm = scaler.transform(x_test['teacher_number

print("After vectorizations")
print(x_train_teacher_num_previous_projects_norm.shape, y_train.shape)
```

```
print(x_test_teacher_num_previous_projects_norm.shape, y_test.shape)
print("="*100)
```

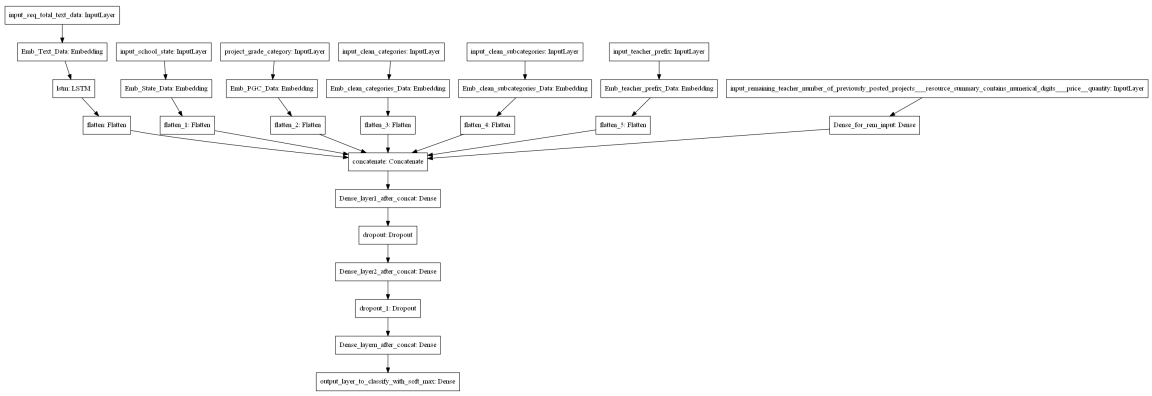
```
After vectorizations
(81936, 1) (81936, 1)
(27312, 1) (27312, 1)
====================================================================================
==================
```

In [33]:
```
train_numerical_features = np.concatenate((x_train_price_norm, x_train_teacher_num_
test_numerical_features = np.concatenate((x_test_price_norm, x_test_teacher_num_pre
```

In [34]:
```
print(train_numerical_features.shape)
print(test_numerical_features.shape)
```

```
(81936, 2)
(27312, 2)
```

1.4 Defining the model



In [28]:
```
# as of now we have vectorized all our features now we will define our model.
# as it is clear from above image that the given model has multiple input layers an
# Please go through - https://keras.io/guides/functional_api/
# it is a good programming practise to define your complete model i.e all inputs ,
# while defining your model make sure that you use variable names while defining an
#for ex.- you should write the code as 'input_text = Input(shape=(pad_length,))' an
# the embedding layer for text data should be non trainable
# the embedding layer for categorical data should be trainable
# https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-w
# https://towardsdatascience.com/deep-embeddings-for-categorical-variables-cat2vec-
#print model.summary() after you have defined the model
#plot the model using utils.plot_model module and make sure that it is similar to t
```

In [48]:
```
# input text layer and text embedding layer
tf.keras.backend.clear_session()
input_layer_text = Input(shape=(maxlen),)
embedding_layer_text = Embedding(input_dim = vocab_size, output_dim = 300, weights
                                 input_length=maxlen, trainable=False) (input_layer_text


lstm = LSTM(128, return_sequences = True) (embedding_layer_text)    # return_sequen
                                                                    # LSTM units in
```

```
flat_text = Flatten() (lstm)

# input school state layer and school state embedding layer
input_school_state = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_school_state_encoded)

embedding_layer_school_state = Embedding(input_dim = inp_dim, output_dim = out_dim,
                            input_length=1, trainable=True) (input_school_state)

flat_school_state = Flatten() (embedding_layer_school_state)


# input project grade category layer and project grade category embedding layer
input_project_grade_category = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_project_grade_category_enco

embedding_layer_project_grade_category = Embedding(input_dim = inp_dim, output_dim
                                        input_length=1, trainable=True) (input_pro

flat_project_grade_category = Flatten() (embedding_layer_project_grade_category)

# input clean categories layer and clean categories embedding layer
input_clean_categories = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_clean_categories_encoded)

embedding_layer_clean_categories = Embedding(input_dim = inp_dim, output_dim = out_
                                        input_length=1, trainable=True) (input_cle

flat_clean_categories = Flatten() (embedding_layer_clean_categories)

# input clean subcategories layer and clean subcategories embedding layer
input_clean_subcategories = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_clean_subcategories_encoded

embedding_layer_clean_subcategories = Embedding(input_dim = inp_dim, output_dim = o
                                       input_length=1, trainable=True) (input_clea

flat_clean_subcategories = Flatten() (embedding_layer_clean_subcategories)

# input teacher prefix layer and teacher prefix embedding layer
input_teacher_prefix = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_teacher_prefix_encoded)

embedding_layer_teacher_prefix = Embedding(input_dim = inp_dim, output_dim = out_di
                                        input_length=1, trainable=True) (input_tea

flat_teacher_prefix = Flatten() (embedding_layer_teacher_prefix)


# input numerical features layer
```

```
input_numerical_features = Input(shape=(2,))
dense_numerical = Dense(64,activation='relu', kernel_initializer='HeUniform')(input

concatenate_1 = Concatenate()([flat_text, flat_school_state, flat_project_grade_cat
                               flat_clean_subcategories, flat_teacher_prefix, dense_n
dense_after_concat_1  = Dense(256,activation='relu', kernel_initializer='HeUniform'



dropout_1 = Dropout(0.5) (dense_after_concat_1)


dense_after_concat_2  = Dense(512,activation='relu', kernel_initializer='HeUniform'

batch_norm = BatchNormalization() (dense_after_concat_2)
dropout_2 = Dropout(0.5) (batch_norm)

dense_after_concat_3 = Dense(512,activation='relu', kernel_initializer='HeUniform')
output_one = Dense(1,activation='sigmoid')(dense_after_concat_3)

model_one = Model(inputs=[input_layer_text, input_school_state, input_project_grade
                          input_clean_subcategories, input_teacher_prefix, input_nu
model_one.summary()
```

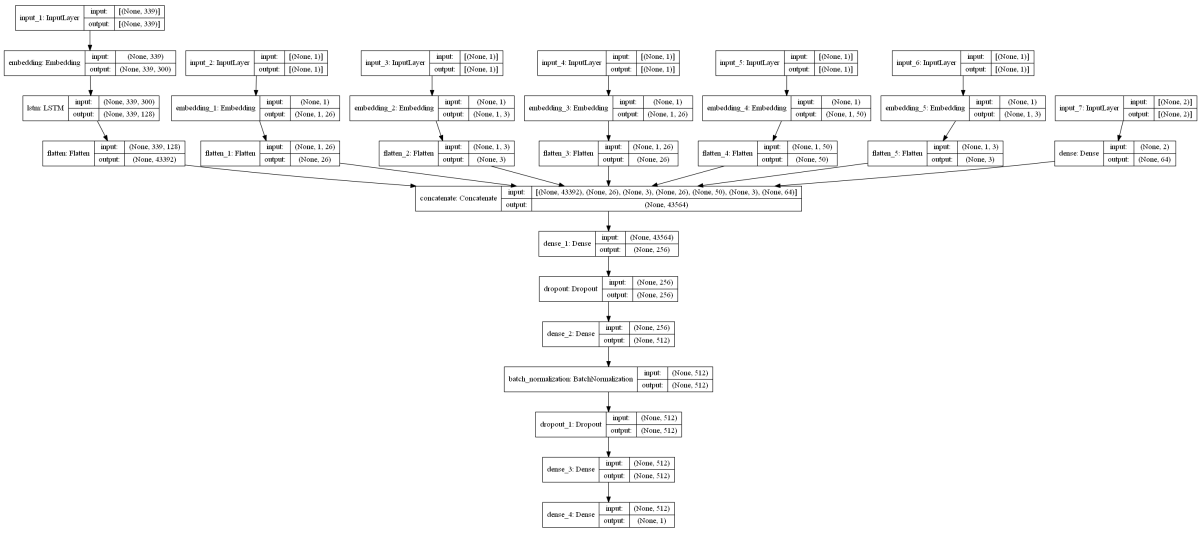
```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 339)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 339, 300)     15103200    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_6 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 339, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 26)        1326        input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 3)         12          input_3[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 26)        1300        input_4[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 50)        19650       input_5[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 3)         15          input_6[0][0]
```

input_7 (InputLayer)	[(None, 2)]	0	
flatten (Flatten)	(None, 43392)	0	lstm[0][0]
flatten_1 (Flatten)	(None, 26)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 3)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 26)	0	embedding_3[0][0]
flatten_4 (Flatten)	(None, 50)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 3)	0	embedding_5[0][0]
dense (Dense)	(None, 64)	192	input_7[0][0]
concatenate (Concatenate)	(None, 43564)	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, 256)	11152640	concatenate[0][0]
dropout (Dropout)	(None, 256)	0	dense_1[0][0]

```
dense_2 (Dense)                 (None, 512)          131584      dropout[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 512)          2048        dense_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 512)          0           batch_normalizati
on[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 512)          262656      dropout_1[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1)            513         dense_3[0][0]
==================================================================================================
Total params: 26,894,784
Trainable params: 11,790,560
Non-trainable params: 15,104,224
__________________________________________________________________________________________________
```

In [49]:
```
plot_model(model_one, to_file='model_one.png', show_shapes=True)
```

Out[49]:



1.5 Compiling and fitting model one

In [50]:
```
#define custom auc as metric , do not use tf.keras.metrics
# https://stackoverflow.com/a/46844409 - custom AUC reference 1
# https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/808
# compile and fit your model
```

In [51]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       ,histogram_freq=1,write_graph=True)


filepath="model_save/best_model_one.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc_2',  verbose=1, sa
# defining mode as 'max' as val_auc_2 is a customm metric and we need to specify ho

call_back_list = [tensorboard_callback, checkpoint]
```

In [52]:
```
model_one.compile(loss='binary_crossentropy', optimizer='adam',metrics=[auc_2])

train_inp = [train_padded_essay, train_school_state_encoded, train_teacher_prefix_e
            train_clean_categories_encoded, train_clean_subcategories_encoded, trai

test_inp = [test_padded_essay, test_school_state_encoded, test_teacher_prefix_encod
           test_clean_categories_encoded, test_clean_subcategories_encoded, test_nu

model_one.fit(train_inp, y_train ,epochs=5, validation_data=(test_inp, y_test),
              batch_size=512, callbacks=call_back_list)
```

```
Epoch 1/5
161/161 [==============================] - 39s 225ms/step - loss: 0.4629 - auc_2:
0.6097 - val_loss: 0.4908 - val_auc_2: 0.7010

Epoch 00001: val_auc_2 improved from -inf to 0.70103, saving model to model_save\b
est_model_one.hdf5
Epoch 2/5
161/161 [==============================] - 33s 204ms/step - loss: 0.4194 - auc_2:
0.6757 - val_loss: 0.4130 - val_auc_2: 0.7244

Epoch 00002: val_auc_2 improved from 0.70103 to 0.72436, saving model to model_sav
e\best_model_one.hdf5
Epoch 3/5
161/161 [==============================] - 33s 205ms/step - loss: 0.3922 - auc_2:
0.7153 - val_loss: 0.4106 - val_auc_2: 0.7267

Epoch 00003: val_auc_2 improved from 0.72436 to 0.72665, saving model to model_sav
e\best_model_one.hdf5
Epoch 4/5
161/161 [==============================] - 33s 202ms/step - loss: 0.3782 - auc_2:
0.7379 - val_loss: 0.5474 - val_auc_2: 0.7551

Epoch 00004: val_auc_2 improved from 0.72665 to 0.75509, saving model to model_sav
e\best_model_one.hdf5
Epoch 5/5
161/161 [==============================] - 33s 205ms/step - loss: 0.3691 - auc_2:
0.7553 - val_loss: 0.4680 - val_auc_2: 0.7619

Epoch 00005: val_auc_2 improved from 0.75509 to 0.76189, saving model to model_sav
e\best_model_one.hdf5
```

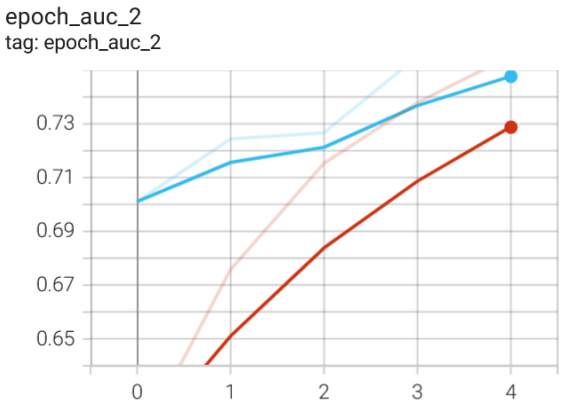
Out[52]:
```
<keras.callbacks.History at 0x1ad35bd2c10>
```

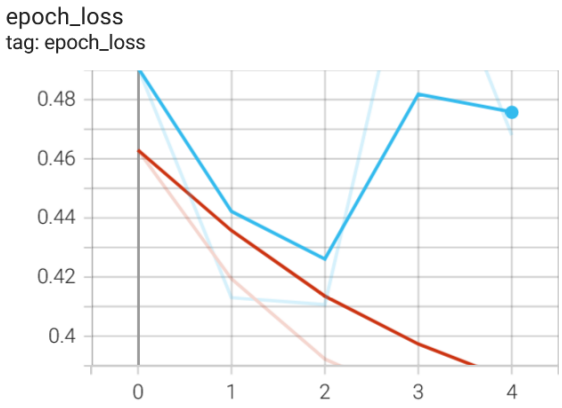
Our final validation AUC for model one is 0.7618 after 5 epochs

In [37]:
```
%tensorboard --logdir logs/fits
```

```
Reusing TensorBoard on port 6006 (pid 12492), started 17:21:43 ago. (Use '!kill 12
492' to kill it.)
```

1.6 Tensorboard image for model one





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. Fit TF-IDF vectorizer on the Train data

2. Get the idf value for each word we have in the train data. Please go through this

3. 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.
Hint - A preferable IDF range is 2-11 for model 2.

4.Remove the low idf value and high idf value words from the train and test data. You can go through each of the sentence of train and test data and include only those features(words) which are present in the defined IDF range.
5. Perform tokenization on the modified text data same as you have done for previous model.
6. Create embedding matrix for model 2 and then use the rest of the features similar to previous model.
7. Define the model, compile and fit the model.

2.1 Text Vectorization using top idf words

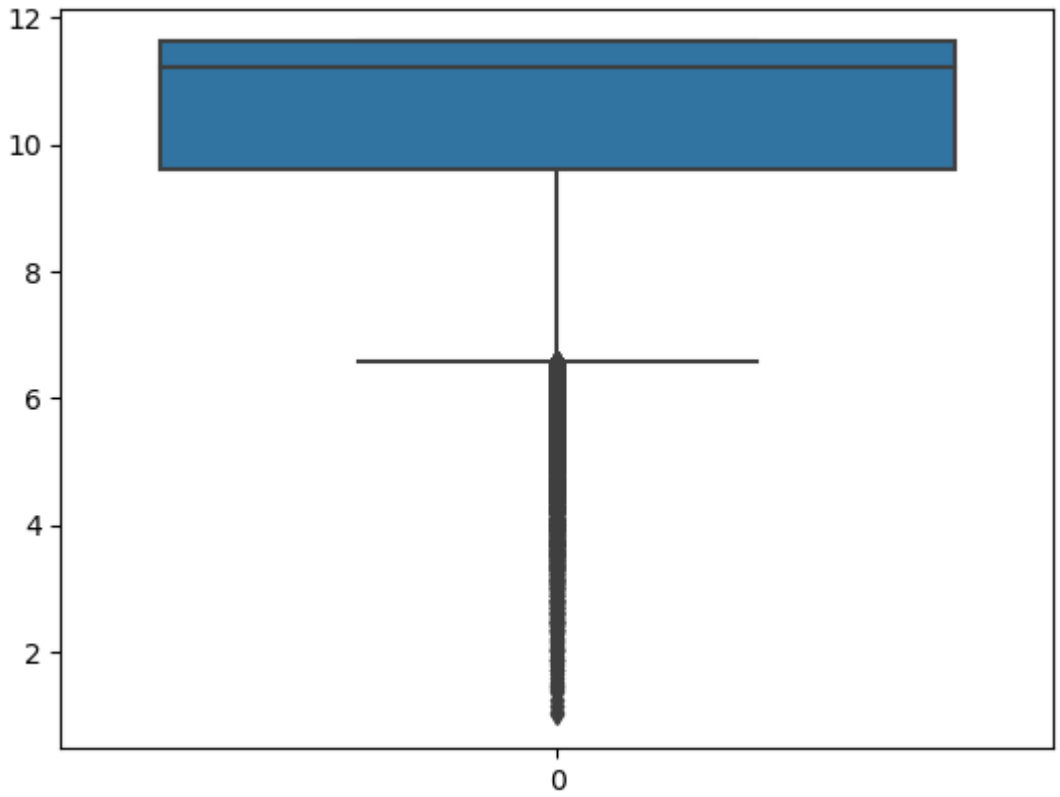
In [12]:
```
tfidf_vectorizer = TfidfVectorizer()
tfidf_text= tfidf_vectorizer.fit_transform(x_train['essay'])
```

In [13]:
```
idf_text = tfidf_vectorizer.idf_
print(len(idf_text))
```

```
50307
```

In [14]:
```
sns.boxplot(idf_text)
```

Out[14]: <AxesSubplot: >



In [15]:
```
for i in range(0,101,10):
    print(" {} percentile idf is {}".format(i, np.percentile(idf_text,i)))
```

```
 0 percentile idf is 1.0076693505388723
 10 percentile idf is 7.560115747169021
 20 percentile idf is 9.055609400253903
 30 percentile idf is 10.01112084528134
 40 percentile idf is 10.704268025841285
 50 percentile idf is 11.215093649607276
 60 percentile idf is 11.215093649607276
 70 percentile idf is 11.62055875771544
 80 percentile idf is 11.62055875771544
 90 percentile idf is 11.62055875771544
 100 percentile idf is 11.62055875771544
```

We can pick the words with idf value between 2 and 11.62

In [17]:
```
tfidf_words_final = []
for idf_value , word in tqdm(zip(tfidf_vectorizer.idf_ , tfidf_vectorizer.get_featu
    if idf_value >= 2 and idf_value <= 11.62:
        tfidf_words_final.append(word)
print(len(tfidf_words_final))
```

```
50307it [00:00, 1733709.52it/s]
30624
```

In [18]:
```
def get_final_text(text):
    final_text = " "
    for word in text.split():
        if word in tfidf_words_final:
            final_text = final_text + " " + word
    final_text = final_text.strip()
    return final_text
```

In [19]:
```
if not (os.path.isfile('model_two_train_inp.csv') and os.path.isfile('model_two_tes
    tqdm.pandas()
    x_train['idf_preprocessed_essay'] = x_train['essay'].progress_map(get_final_tex
    x_test['idf_preprocessed_essay'] = x_test['essay'].progress_map(get_final_text)
    x_train.to_csv('model_two_train_inp.csv',index= False)
    x_test.to_csv('model_two_test_inp.csv',index= False)
else:
    x_train = pd.read_csv('model_two_train_inp.csv')
    x_test = pd.read_csv('model_two_test_inp.csv')
    print('idf preprocessed input file is already present')
```

```
100%|████████████████████████████████████████████████████████████████| 8
1936/81936 [1:55:51<00:00, 11.79it/s]
100%|████████████████████████████████████████████████████████████████|
27312/27312 [38:33<00:00, 11.81it/s]
```

In [53]:
```
x_train.head()
```

Out[53]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_
50356	wi	ms	grades_9_12	
108124	ca	ms	grades_3_5	
107343	mi	mrs	grades_prek_2	
19811	wa	mrs	grades_3_5	
68349	ct	ms	grades_prek_2	

In [21]:

```
#Code is copied from: https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenizati

oov_token = '<UNK>'
pad_type = 'post'
trunc_type = 'post'

tokenizer = Tokenizer(oov_token = oov_token, filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|
tokenizer.fit_on_texts(x_train['idf_preprocessed_essay'])


# Encode training data sentences into sequences
train_sequences_idf = tokenizer.texts_to_sequences(x_train['idf_preprocessed_essay'

# Get max training sequence length
maxlen = max([len(x) for x in train_sequences_idf])

# Pad the training sequences
train_padded_essay_idf = pad_sequences(train_sequences_idf, padding=pad_type, trunc
```

```
test_sequences_idf = tokenizer.texts_to_sequences(x_test['idf_preprocessed_essay'])
test_padded_essay_idf = pad_sequences(test_sequences_idf, padding=pad_type, truncat
```

In [22]:
```
vocab = tokenizer.word_index
#https://datascience.stackexchange.com/questions/93651/reason-for-adding-1-to-word-
vocab_size = len(vocab)+1 # word_index starts with index 1. Last word in word_index
                          # Normally index always starts at zero. So, to access the
                          # we need to specify vocab_size = len(vocab) + 1
```

In [23]:
```
vocab_size
```

Out[23]:
```
30626
```

In [24]:
```
print(train_padded_essay_idf.shape)
print(test_padded_essay_idf.shape)
```

```
(81936, 290)
(27312, 290)
```

In [25]:
```
embedding_matrix = np.zeros((vocab_size, 300))  # We used 300 dimensional pre-defin
for word, i in vocab.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

In [26]:
```
embedding_matrix.shape
```

Out[26]:
```
(30626, 300)
```

2.2 Defining model two

In [35]:
```
# input text layer and text embedding layer
tf.keras.backend.clear_session()
input_layer_idf_preprocessed_text = Input(shape=(maxlen,))
embedding_layer_text = Embedding(input_dim = vocab_size, output_dim = 300, weights
                           input_length=maxlen, trainable = False)(input_layer_idf


lstm = LSTM(128, return_sequences = True) (embedding_layer_text)        # return_se
                                                                        # multiple
flat_text = Flatten() (lstm)

# input school state layer and school state embedding layer
input_school_state = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_school_state_encoded)

embedding_layer_school_state = Embedding(input_dim = inp_dim, output_dim = out_dim,
                           input_length=1, trainable=True) (input_school_state)

flat_school_state = Flatten() (embedding_layer_school_state)
```

```
# input project grade category layer and project grade category embedding layer
input_project_grade_category = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_project_grade_category_enco

embedding_layer_project_grade_category = Embedding(input_dim = inp_dim, output_dim
                                         input_length=1, trainable=True) (input_pro

flat_project_grade_category = Flatten() (embedding_layer_project_grade_category)

# input clean categories layer and clean categories embedding layer
input_clean_categories = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_clean_categories_encoded)

embedding_layer_clean_categories = Embedding(input_dim = inp_dim, output_dim = out_
                                         input_length=1, trainable=True) (input_cle

flat_clean_categories = Flatten() (embedding_layer_clean_categories)

# input clean subcategories layer and clean subcategories embedding layer
input_clean_subcategories = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_clean_subcategories_encoded

embedding_layer_clean_subcategories = Embedding(input_dim = inp_dim, output_dim = o
                                       input_length=1, trainable=True) (input_clea

flat_clean_subcategories = Flatten() (embedding_layer_clean_subcategories)

# input teacher prefix layer and teacher prefix embedding layer
input_teacher_prefix = Input(shape=(1,))

inp_dim, out_dim = embedding_input_dim_output_dim(train_teacher_prefix_encoded)

embedding_layer_teacher_prefix = Embedding(input_dim = inp_dim, output_dim = out_di
                                         input_length=1, trainable=True) (input_tea

flat_teacher_prefix = Flatten() (embedding_layer_teacher_prefix)


# input numerical features layer
input_numerical_features = Input(shape=(2,))
dense_numerical = Dense(16,activation='relu', kernel_initializer='HeUniform')(input

concatenate_1 = Concatenate()([flat_text, flat_school_state, flat_project_grade_cat
                               flat_clean_subcategories, flat_teacher_prefix, dense_n


dense_after_concat_1 = Dense(256,activation='relu', kernel_initializer='HeUniform')




dropout_1 = Dropout(0.5) (dense_after_concat_1)
```

```
dense_after_concat_2 = Dense(512,activation= 'relu', kernel_initializer='HeUniform'


dropout_2 = Dropout(0.5) (dense_after_concat_2)


dense_after_concat_3 = Dense(512,activation= 'relu', kernel_initializer='HeUniform'


output_two = Dense(1,activation='sigmoid')(dense_after_concat_3)

model_two = Model(inputs=[input_layer_idf_preprocessed_text, input_school_state, in
                          input_clean_categories, input_clean_subcategories, input_
                          input_numerical_features], outputs=output_two)
model_two.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 290)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 290, 300)     9187800     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_6 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 290, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 26)        1326        input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 3)         12          input_3[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 26)        1300        input_4[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 50)        19650       input_5[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 3)         15          input_6[0][0]
```

input_7 (InputLayer)	[(None, 2)]	0	
flatten (Flatten)	(None, 37120)	0	lstm[0][0]
flatten_1 (Flatten)	(None, 26)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 3)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 26)	0	embedding_3[0][0]
flatten_4 (Flatten)	(None, 50)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 3)	0	embedding_5[0][0]
dense (Dense)	(None, 16)	48	input_7[0][0]
concatenate (Concatenate)	(None, 37244)	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, 256)	9534720	concatenate[0][0]
dropout (Dropout)	(None, 256)	0	dense_1[0][0]

```
_____
dense_2 (Dense)                 (None, 512)          131584      dropout[0][0]
_____
_____
dropout_1 (Dropout)             (None, 512)          0           dense_2[0][0]
_____
_____
dense_3 (Dense)                 (None, 512)          262656      dropout_1[0][0]
_____
_____
dense_4 (Dense)                 (None, 1)            513         dense_3[0][0]
==========================================================================
================
Total params: 19,359,272
Trainable params: 10,171,472
Non-trainable params: 9,187,800
_____
_____
```

In [36]:
```
plot_model(model_two, to_file='model_two.png', show_shapes=True)
```

Out[36]:

2.3 Compiling and fitting model two

In [37]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       ,histogram_freq=1,write_graph=True)


filepath="model_save/best_model_two.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc_2',  verbose=1, sa
# defining mode as 'max' as val_auc_2 is a customm metric and we need to specify ho

call_back_list = [checkpoint, tensorboard_callback]
```

In [38]:
```
model_two.compile(loss='binary_crossentropy', optimizer='adam',metrics=[auc_2])

train_inp = [train_padded_essay_idf, train_school_state_encoded, train_teacher_pref
             train_clean_categories_encoded, train_clean_subcategories_encoded, trai

test_inp = [test_padded_essay_idf, test_school_state_encoded, test_teacher_prefix_e
           test_clean_categories_encoded, test_clean_subcategories_encoded, test_nu

model_two.fit(train_inp, y_train ,epochs=5, validation_data=(test_inp, y_test),
              batch_size=512, callbacks=call_back_list)
```

```
Epoch 1/5
161/161 [==============================] - 35s 191ms/step - loss: 0.4372 - auc_2:
0.6680 - val_loss: 0.4523 - val_auc_2: 0.7233

Epoch 00001: val_auc_2 improved from -inf to 0.72331, saving model to model_save\b
est_model_two.hdf5
Epoch 2/5
161/161 [==============================] - 30s 183ms/step - loss: 0.3804 - auc_2:
0.7379 - val_loss: 0.4723 - val_auc_2: 0.7335

Epoch 00002: val_auc_2 improved from 0.72331 to 0.73346, saving model to model_sav
e\best_model_two.hdf5
Epoch 3/5
161/161 [==============================] - 30s 184ms/step - loss: 0.3648 - auc_2:
0.7680 - val_loss: 0.4857 - val_auc_2: 0.7244

Epoch 00003: val_auc_2 did not improve from 0.73346
Epoch 4/5
161/161 [==============================] - 30s 184ms/step - loss: 0.3550 - auc_2:
0.7861 - val_loss: 0.4039 - val_auc_2: 0.7401

Epoch 00004: val_auc_2 improved from 0.73346 to 0.74008, saving model to model_sav
e\best_model_two.hdf5
Epoch 5/5
161/161 [==============================] - 30s 184ms/step - loss: 0.3303 - auc_2:
0.8218 - val_loss: 0.4091 - val_auc_2: 0.7231

Epoch 00005: val_auc_2 did not improve from 0.74008
```

Out[38]:
```
<keras.callbacks.History at 0x1ad59a84e80>
```

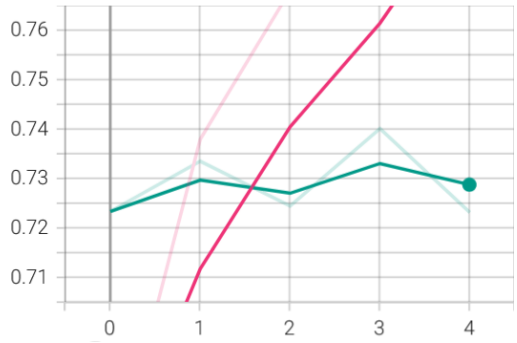
Our final validation AUC for model one is 0.74 after 5 epochs

In [39]:
```
%tensorboard --logdir logs/fits
```

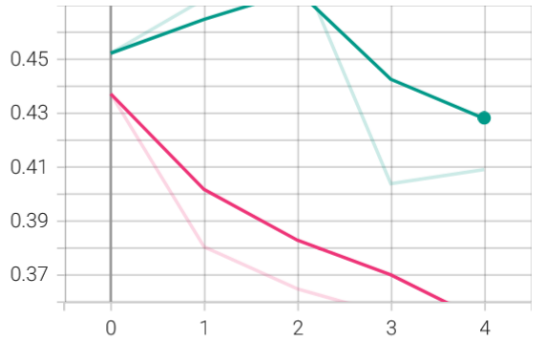
```
Reusing TensorBoard on port 6006 (pid 12492), started 2 days, 7:40:41 ago. (Use '!
kill 12492' to kill it.)
```

2.4 Tensorboard image for model two

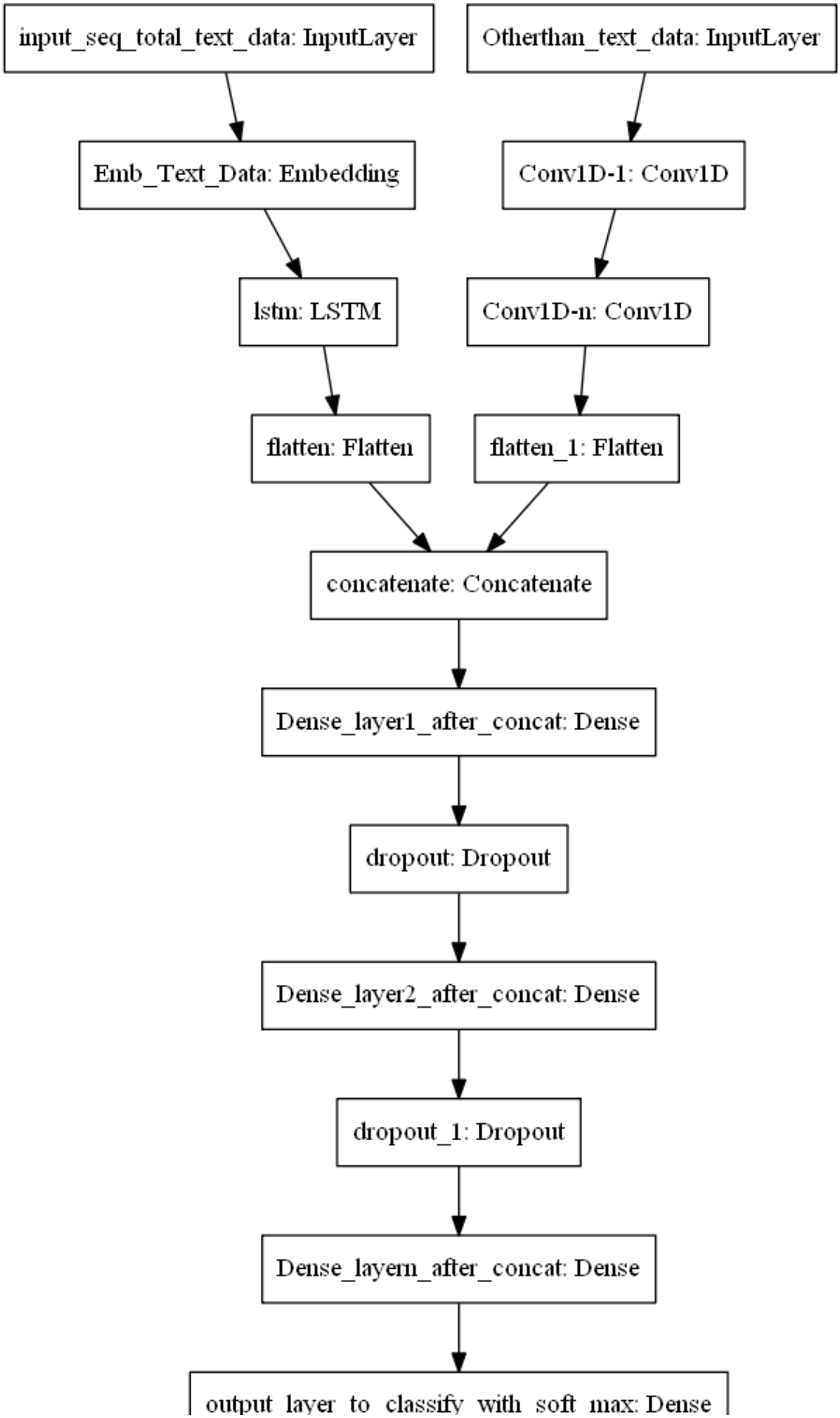
epoch_auc_2
tag: epoch_auc_2

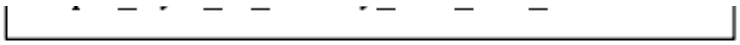


epoch_loss
tag: epoch_loss



Model-3


input_seq_total_text_data: InputLayer
Otherthan_text_data: InputLayer
Emb_Text_Data: Embedding
Conv1D-1: Conv1D
lstm: LSTM
Conv1D-n: Conv1D
flatten: Flatten
flatten_1: Flatten
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/fkQ8nGo.png

3.1 Vectorizing features other than text data

In [19]:
```
#in this model you can use the text vectorized data from model1
#for other than text data consider the following steps
# you have to perform one hot encoding of categorical features. You can use onehote
# Stack up standardised numerical features and all the one hot encoded categorical
#the input to conv1d layer is 3d, you can convert your 2d data to 3d using np.newax
# Note - deep learning models won't work with sparse features, you have to convert
```

In [20]:
```
school_state_vectorizer = CountVectorizer()
school_state_vectorizer.fit(x_train['school_state'].values) # fit has to happen onl

# we use the fitted CountVectorizer to convert the text to vector
x_train_state_ohe = school_state_vectorizer.transform(x_train['school_state'].value
x_test_state_ohe = school_state_vectorizer.transform(x_test['school_state'].values)

print("After vectorizations")
print(x_train_state_ohe.shape, y_train.shape)
print(x_test_state_ohe.shape, y_test.shape)
print(school_state_vectorizer.get_feature_names_out())
print("="*100)
```

```
After vectorizations
(81936, 51) (81936,)
(27312, 51) (27312,)
['ak' 'al' 'ar' 'az' 'ca' 'co' 'ct' 'dc' 'de' 'fl' 'ga' 'hi' 'ia' 'id'
 'il' 'in' 'ks' 'ky' 'la' 'ma' 'md' 'me' 'mi' 'mn' 'mo' 'ms' 'mt' 'nc'
 'nd' 'ne' 'nh' 'nj' 'nm' 'nv' 'ny' 'oh' 'ok' 'or' 'pa' 'ri' 'sc' 'sd'
 'tn' 'tx' 'ut' 'va' 'vt' 'wa' 'wi' 'wv' 'wy']
====================================================================================
==================
```

In [21]:
```
teacher_prefix_vectorizer = CountVectorizer()
teacher_prefix_vectorizer.fit(x_train['teacher_prefix'].values) # fit has to happen

# we use the fitted CountVectorizer to convert the text to vector
x_train_teacher_ohe = teacher_prefix_vectorizer.transform(x_train['teacher_prefix']
x_test_teacher_ohe = teacher_prefix_vectorizer.transform(x_test['teacher_prefix'].v

print("After vectorizations")
print(x_train_teacher_ohe.shape, y_train.shape)
print(x_test_teacher_ohe.shape, y_test.shape)
print(teacher_prefix_vectorizer.get_feature_names_out())
print("="*100)
```

```
After vectorizations
(81936, 5) (81936,)
(27312, 5) (27312,)
['dr' 'mr' 'mrs' 'ms' 'teacher']
====================================================================================
==================
```

In [22]:
```
project_grade_category_vectorizer = CountVectorizer()
project_grade_category_vectorizer.fit(x_train['project_grade_category'].values) # f

# we use the fitted CountVectorizer to convert the text to vector
x_train_grade_ohe = project_grade_category_vectorizer.transform(x_train['project_gr
x_test_grade_ohe = project_grade_category_vectorizer.transform(x_test['project_grad

print("After vectorizations")
print(x_train_grade_ohe.shape, y_train.shape)
print(x_test_grade_ohe.shape, y_test.shape)
print(project_grade_category_vectorizer.get_feature_names_out())
print("="*100)
```

```
After vectorizations
(81936, 4) (81936,)
(27312, 4) (27312,)
['grades_3_5' 'grades_6_8' 'grades_9_12' 'grades_prek_2']
====================================================================================
==================
```

In [23]:
```
clean_categories_vectorizer = CountVectorizer()
clean_categories_vectorizer.fit(x_train['clean_categories'].values) # fit has to ha

# we use the fitted CountVectorizer to convert the text to vector
x_train_categories_ohe = clean_categories_vectorizer.transform(x_train['clean_categ
x_test_categories_ohe = clean_categories_vectorizer.transform(x_test['clean_categor

print("After vectorizations")
print(x_train_categories_ohe.shape, y_train.shape)
print(x_test_categories_ohe.shape, y_test.shape)
print(clean_categories_vectorizer.get_feature_names_out())
print("="*100)
```

```
After vectorizations
(81936, 9) (81936,)
(27312, 9) (27312,)
['appliedlearning' 'care_hunger' 'health_sports' 'history_civics'
 'literacy_language' 'math_science' 'music_arts' 'specialneeds' 'warmth']
====================================================================================
==================
```

In [24]:
```
clean_subcategories_vectorizer = CountVectorizer()
clean_subcategories_vectorizer.fit(x_train['clean_subcategories'].values) # fit has

# we use the fitted CountVectorizer to convert the text to vector
x_train_subcategories_ohe = clean_subcategories_vectorizer.transform(x_train['clean
x_test_subcategories_ohe = clean_subcategories_vectorizer.transform(x_test['clean_s

print("After vectorizations")
print(x_train_subcategories_ohe.shape, y_train.shape)
print(x_test_subcategories_ohe.shape, y_test.shape)
print(clean_subcategories_vectorizer.get_feature_names_out())
print("="*100)
```

```
After vectorizations
(81936, 30) (81936,)
(27312, 30) (27312,)
['appliedsciences' 'care_hunger' 'charactereducation' 'civics_government'
 'college_careerprep' 'communityservice' 'earlydevelopment' 'economics'
 'environmentalscience' 'esl' 'extracurricular' 'financialliteracy'
 'foreignlanguages' 'gym_fitness' 'health_lifescience' 'health_wellness'
 'history_geography' 'literacy' 'literature_writing' 'mathematics' 'music'
 'nutritioneducation' 'other' 'parentinvolvement' 'performingarts'
 'socialsciences' 'specialneeds' 'teamsports' 'visualarts' 'warmth']
==================================================================================
==================
```

In [25]:
```
scaler = StandardScaler()
scaler.fit(x_train['price'].values.reshape(-1,1))

x_train_price_norm = scaler.transform(x_train['price'].values.reshape(-1,1))
x_test_price_norm = scaler.transform(x_test['price'].values.reshape(-1,1))


print("After vectorizations")
print(x_train_price_norm.shape, y_train.shape)
print(x_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(81936, 1) (81936,)
(27312, 1) (27312,)
==================================================================================
==================
```

In [26]:
```
scaler = StandardScaler()
scaler.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(-

x_train_teacher_num_previous_projects_norm =scaler.transform(x_train['teacher_numbe
x_test_teacher_num_previous_projects_norm = scaler.transform(x_test['teacher_number

print("After vectorizations")
print(x_train_teacher_num_previous_projects_norm.shape, y_train.shape)
print(x_test_teacher_num_previous_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(81936, 1) (81936,)
(27312, 1) (27312,)
==================================================================================
==================
```

In [27]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
x_train_other = hstack((x_train_state_ohe, x_train_teacher_ohe, x_train_grade_ohe,
x_test_other = hstack((x_test_state_ohe, x_test_teacher_ohe, x_test_grade_ohe, x_te
x_train_other = x_train_other.todense()
x_test_other = x_test_other.todense()
print(x_train_other.shape, x_test_other.shape)
```

```
(81936, 101) (27312, 101)
```

In [28]:
```
x_train_other = np.resize(x_train_other, (81936, 101,1))
x_test_other = np.resize(x_test_other, (27312, 101,1))
```

In [29]:
```
print(x_train_other.shape, x_test_other.shape)
```

```
(81936, 101, 1) (27312, 101, 1)
```

3.2 Define model three

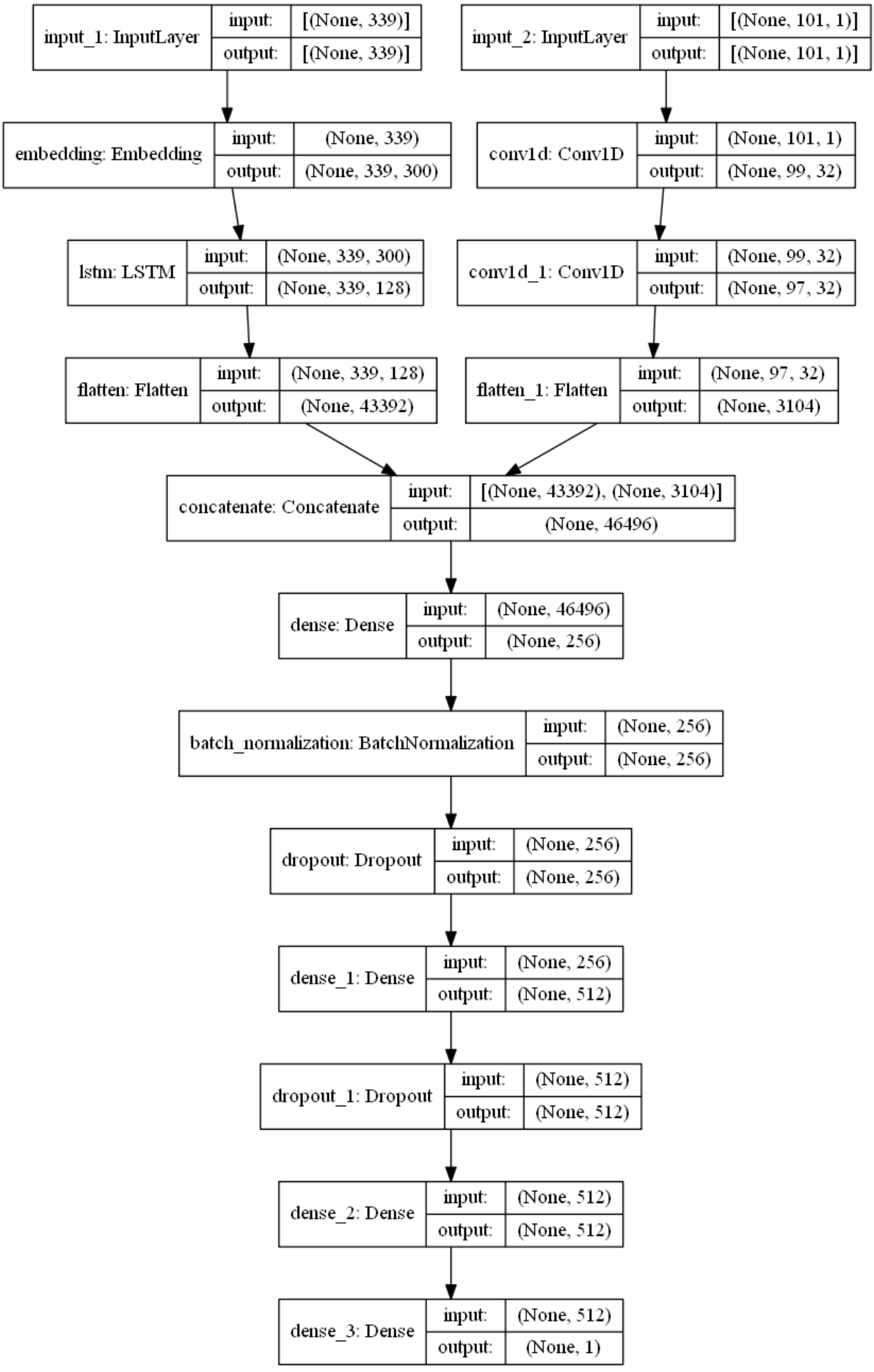
In [30]:
```
# input text layer and text embedding layer
tf.keras.backend.clear_session()
input_layer_text = Input(shape=(maxlen),)
embedding_layer_text = Embedding(input_dim = vocab_size, output_dim = 300, weights
                           input_length=maxlen, trainable=False) (input_layer_text


lstm = LSTM(128, return_sequences = True) (embedding_layer_text)    # return_sequen
                                                                    # LSTM units in

flat_text = Flatten() (lstm)

# input other than text data

other_than_text_input = Input(shape=(101,1,))
conv_1 = Conv1D(32, 3, activation = 'relu', kernel_initializer='HeUniform') (other_
conv_2 = Conv1D(32, 3, activation = 'relu', kernel_initializer='HeUniform') (conv_1

flat_other_than_text_input = Flatten() (conv_2)



concatenate_1 = Concatenate()([flat_text, flat_other_than_text_input])
dense_after_concat_1  = Dense(256,activation='relu', kernel_initializer='HeUniform'


batch_norm = BatchNormalization() (dense_after_concat_1)
dropout_1 = Dropout(0.5) (batch_norm)


dense_after_concat_2 = Dense(512,activation='relu', kernel_initializer='HeUniform')

dropout_2 = Dropout(0.5) (dense_after_concat_2)

dense_after_concat_3 = Dense(512,activation='relu', kernel_initializer='HeUniform')

output_three = Dense(1,activation='sigmoid')(dense_after_concat_3)

model_three = Model(inputs=[input_layer_text, other_than_text_input], outputs=outpu
model_three.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 339)]        0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 101, 1)]     0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 339, 300)     15075900    input_1[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 99, 32)       128         input_2[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 339, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 97, 32)       3104        conv1d[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 43392)        0           lstm[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 3104)         0           conv1d_1[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 46496)        0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          11903232    concatenate[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 256)          1024        dense[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 256)          0           batch_normalizati
on[0][0]
__________________________________________________________________________________________________
```

```
_________________
dense_1 (Dense)                 (None, 512)          131584      dropout[0][0]

_____________________________________________________________________________
_________________
dropout_1 (Dropout)             (None, 512)          0           dense_1[0][0]

_____________________________________________________________________________
_________________
dense_2 (Dense)                 (None, 512)          262656      dropout_1[0][0]

_____________________________________________________________________________
_________________
dense_3 (Dense)                 (None, 1)            513         dense_2[0][0]

=============================================================================
================
Total params: 27,597,789
Trainable params: 12,521,377
Non-trainable params: 15,076,412
_____________________________________________________________________________
_________________
```

In [31]:
```
plot_model(model_three, to_file='model_three.png', show_shapes=True)
```

Out[31]:


input_1: InputLayer
input: [(None, 339)]
output: [(None, 339)]
input_2: InputLayer
input: [(None, 101, 1)]
output: [(None, 101, 1)]
embedding: Embedding
input: (None, 339)
output: (None, 339, 300)
conv1d: Conv1D
input: (None, 101, 1)
output: (None, 99, 32)
lstm: LSTM
input: (None, 339, 300)
output: (None, 339, 128)
conv1d_1: Conv1D
input: (None, 99, 32)
output: (None, 97, 32)
flatten: Flatten
input: (None, 339, 128)
output: (None, 43392)
flatten_1: Flatten
input: (None, 97, 32)
output: (None, 3104)
concatenate: Concatenate
input: [(None, 43392), (None, 3104)]
output: (None, 46496)
dense: Dense
input: (None, 46496)
output: (None, 256)
batch_normalization: BatchNormalization
input: (None, 256)
output: (None, 256)
dropout: Dropout
input: (None, 256)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 512)
dropout_1: Dropout
input: (None, 512)
output: (None, 512)
dense_2: Dense
input: (None, 512)
output: (None, 512)
dense_3: Dense
input: (None, 512)
output: (None, 1)

3.3 Compiling and fitting model three

In [32]:
```
#define custom auc as metric , do not use tf.keras.metrics
# https://stackoverflow.com/a/46844409 - custom AUC reference 1
# https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/808
# compile and fit your model
```

In [33]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       ,histogram_freq=1,write_graph=True)


filepath="model_save/best_model_three.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc_2',  verbose=1, sa
# defining mode as 'max' as val_auc_2 is a customm metric and we need to specify ho

call_back_list = [checkpoint, tensorboard_callback]
```

In [34]:
```
model_three.compile(loss='binary_crossentropy', optimizer='adam',metrics=[auc_2])

train_inp = [train_padded_essay, x_train_other]

test_inp = [test_padded_essay, x_test_other]

model_three.fit(train_inp, y_train ,epochs=5, validation_data=(test_inp, y_test),
              batch_size=512, callbacks=call_back_list)
```

```
Epoch 1/5
161/161 [==============================] - 39s 213ms/step - loss: 0.4466 - auc_2:
0.6370 - val_loss: 0.3938 - val_auc_2: 0.7190

Epoch 00001: val_auc_2 improved from -inf to 0.71901, saving model to model_save\b
est_model_three.hdf5
Epoch 2/5
161/161 [==============================] - 33s 206ms/step - loss: 0.3910 - auc_2:
0.7211 - val_loss: 0.3833 - val_auc_2: 0.7438

Epoch 00002: val_auc_2 improved from 0.71901 to 0.74384, saving model to model_sav
e\best_model_three.hdf5
Epoch 3/5
161/161 [==============================] - 33s 204ms/step - loss: 0.3765 - auc_2:
0.7441 - val_loss: 0.3771 - val_auc_2: 0.7486

Epoch 00003: val_auc_2 improved from 0.74384 to 0.74858, saving model to model_sav
e\best_model_three.hdf5
Epoch 4/5
161/161 [==============================] - 33s 205ms/step - loss: 0.3610 - auc_2:
0.7718 - val_loss: 0.3889 - val_auc_2: 0.7536

Epoch 00004: val_auc_2 improved from 0.74858 to 0.75358, saving model to model_sav
e\best_model_three.hdf5
Epoch 5/5
161/161 [==============================] - 33s 205ms/step - loss: 0.3471 - auc_2:
0.7889 - val_loss: 0.4075 - val_auc_2: 0.7475

Epoch 00005: val_auc_2 did not improve from 0.75358
```

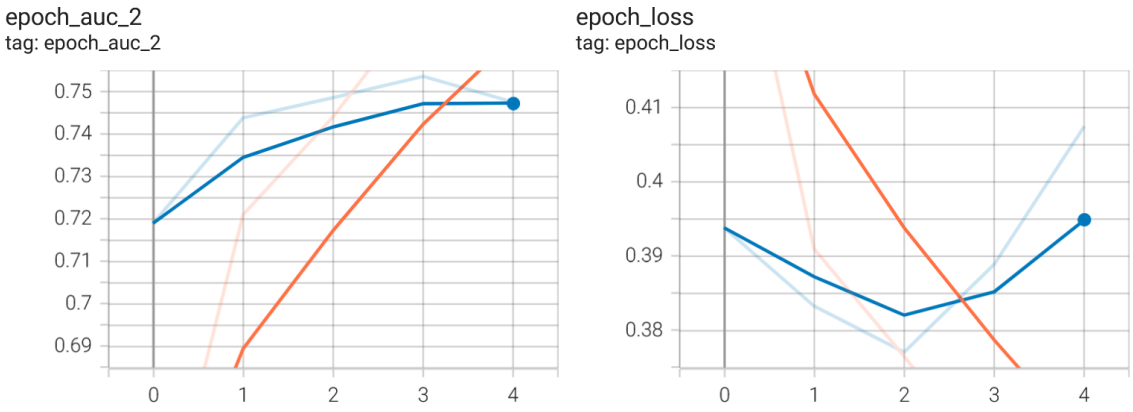
Out[34]: <keras.callbacks.History at 0x19f5008b7f0>

Our final validation AUC for model three is 0.7536 after 5 epochs

In [35]:
```
%tensorboard --logdir logs/fits
```

```
Reusing TensorBoard on port 6006 (pid 12492), started 1 day, 10:03:37 ago. (Use '!
kill 12492' to kill it.)
```

3.4 Tensorboard image for model three



In [54]:
```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", "Epochs", "val_accuracy"]

x.add_row(["model_one", "5","0.7618"])
x.add_row(["model_two", "5","0.74"])
x.add_row(["model_three", "5","0.7536"])

print(x)
```

```
+-------------+--------+--------------+
|    Model    | Epochs | val_accuracy |
+-------------+--------+--------------+
|  model_one  |   5    |    0.7618    |
|  model_two  |   5    |     0.74     |
| model_three |   5    |    0.7536    |
+-------------+--------+--------------+
```