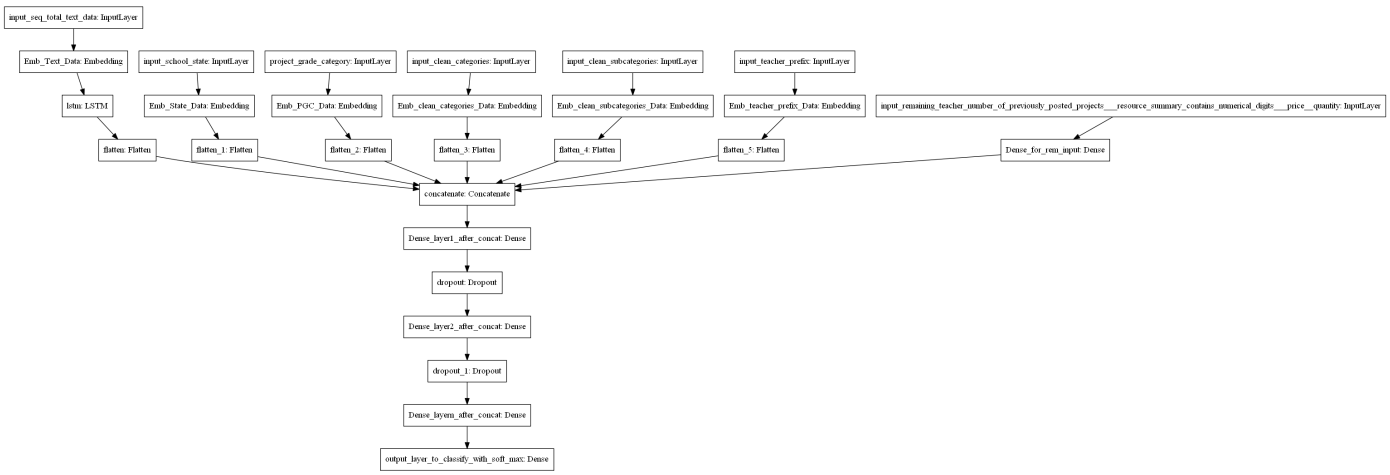
Assignment : 14

1. Preprocess all the Data we have in DonorsChoose Dataset (https://drive.google.com/drive/folders/1MIwK7BQMev8f5CbDDVNLPaFGB32pFN60) 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' (https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics) as a metric. check this (https://datascience.stackexchange.com/a/20192) for using auc as a metric
5. You are free to choose any number of layers/hiddden units but you have to use same type of architectures shown below.
6. You can use any one of the optimizers and choice of Learning rate and momentum, resources: cs231n class notes (http://cs231n.github.io/neural-networks-3/), cs231n class video (https://www.youtube.com/watch?v=hd_KFJ5ktUc).
7. For all the model's use TensorBoard (https://www.youtube.com/watch?v=2U6Jl7oqRkM) 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 (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_nu** ---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.

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

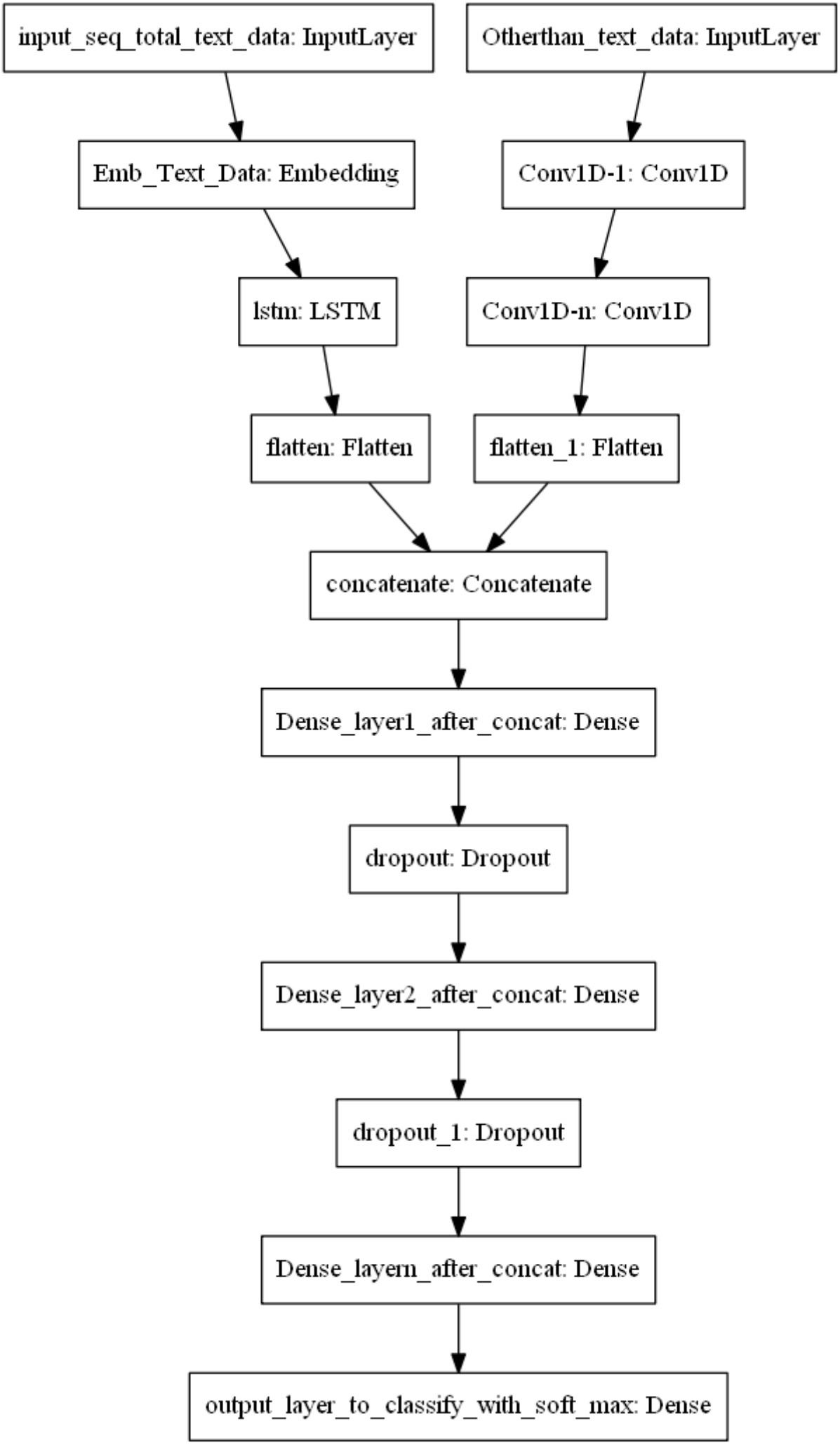
2. Please go through this link https://keras.io/getting-started/functional-api-guide/ (https://keras.io/getting-started/functional-api-guide/) and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.

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 analys
is on the Idf values and based on those values choose the low and high threshold v
alue. Because very frequent words and very very rare words don't give much informa
tion. (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 Tra
in on total data but in Model-2 train on data after removing some words based on I
DF values)
```

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 (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 al
l these onehot vectors

  . Neumerical values and use CNN1D (https://keras.io/getting-started/sequentia
l-model-guide/#sequence-classification-with-1d-convolutions) as shown in above
 figure.

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

In [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
from sklearn.preprocessing import LabelEncoder
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os



from collections import Counter
from keras.utils import to_categorical

from tensorflow.keras.callbacks import TensorBoard

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from scipy.sparse import hstack
from numpy import zeros
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Input
from keras.layers import Flatten
from keras.layers import Embedding
from keras.layers import LSTM,Bidirectional
from keras.layers.core import Dense, Dropout
from keras.models import Model, load_model
from keras.layers.normalization import BatchNormalization
from keras.callbacks import ReduceLROnPlateau
```

In [0]:

```
```

In [0]:

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

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

In [0]:

```
data = "/content/drive/My Drive/preprocessed_data.csv"
```

In [0]:

```
file_resource = "/content/drive/My Drive/resources.csv"
tr="/content/drive/My Drive/train_data.csv"
```

In [0]:

```
proj=pd.read_csv(tr)
price_da=pd.read_csv(file_resource)
```

In [0]:

```
project_data = pd.merge(proj, price_da, on='id', how='left')
```

In [0]:

```
project_data.head(2)
```

Out[70]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2
1	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2

In [0]:

```
quanlity=project_data['quantity']
```

In [0]:

```
x=pd.read_csv(data)
y=x['project_is_approved']
```

In [0]:

```
from keras.utils import to_categorical
y = to_categorical(y)
```

In [0]:

```
x['quantity']=quanlity
```

In [0]:

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y)
```

In [0]:

```
rem_input_train = np.concatenate((x_train['quantity'].values.reshape(-1,1),x_train['price']
#rem_input_cv = np.concatenate((x_cross['quantity'].values.reshape(-1,1),x_cross['price'].v
rem_input_test = np.concatenate((x_test['quantity'].values.reshape(-1,1),x_test['price'].va
```

In [0]:

```
rem_input_train_norm  = np.hstack((x_train_price_norm,x_train_tpp_norm,x_train_qty_norm))
rem_input_test_norm = np.hstack((x_test_price_norm,x_test_tpp_norm,x_test_qty_norm))
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler().fit(rem_input_train)
rem_input_train_norm = scale.transform(rem_input_train)

rem_input_test_norm = scale.transform(rem_input_test)
```

In [0]:

```
rem_input_test_norm .shape
```

Out[160]:

```
(21850, 3)
```

In [0]:

```
### text
```

In [0]:

```
#https://subscription.packtpub.com/book/application_development/9781782167853/1/ch01lvl1sec
t = Tokenizer()
t.fit_on_texts(x_train['essay'])
vocab_size = len(t.word_index) + 1
print('Total unique words in the x_train',vocab_size)
encoded_train = t.texts_to_sequences(x_train['essay'])
encoded_test = t.texts_to_sequences(x_test['essay'])
```

```
Total unique words in the x_train 51763
```

In [0]:

```
max_length = 300
padded_train = pad_sequences(encoded_train, maxlen=max_length, padding='post')
padded_test = pad_sequences(encoded_test, maxlen=max_length, padding='post')
print("length of padded_train data",len(padded_train))
print("length of padded_test data",len(padded_test))
```

```
length of padded_train data 87398
length of padded_test data 21850
```

In [0]:

```
fil = '/content/drive/My Drive/glove_vectors'
```

In [0]:

```
with open('/content/drive/My Drive/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

# for train
embedding_matrix_train = np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
    if word in glove_words:
        embedding_vector = model[word]
        embedding_matrix_train[i] = embedding_vector
```

In [0]:

```
###"#" cate
```

In [0]:

```
def self_token(column):
    unique = list(set(column))
    total = list(column)
    size = len(unique)
    count = []
    for category in unique:
        count.append([total.count(category),category])
    count.sort()
    rank = {}
    for i in range(1,len(count)+1):
        rank.update({count[i-1][1] : i})
    return (rank , unique,size)
```

In [0]:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

le.fit(x_train['clean_categories'])

x_test["clean_categories"] = x_test["clean_categories"].map(lambda a: '<unknown>' if a not
#x_cv["project_subject_categories"] = x_cv["project_subject_categories"].map(lambda a: '<un

le.classes_ = np.append(le.classes_, '<unknown>')

tr_pro_encode = le.transform(x_train['clean_categories'])

test_pro_encode = le.transform(x_test['clean_categories'])
```

In [0]:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

le.fit(x_train['clean_subcategories'])

x_test["clean_subcategories"] = x_test["clean_subcategories"].map(lambda a: '<unknown>' if

le.classes_ = np.append(le.classes_, '<unknown>')

tr_pro_sub_encode = le.transform(x_train['clean_subcategories'])
#cv_pro_sub_encode = le.transform(x_cv['clean_subcategories'])
test_pro_sub_encode = le.transform(x_test['clean_subcategories'])
```

In [0]:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

tr_sch_encode = le.fit_transform(x_train['school_state'])
#cv_sch_encode = le.transform(x_cv['school_state'])
test_sch_encode = le.transform(x_test['school_state'])
```

In [0]:

```

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

tr_tea_pre_encode = le.fit_transform(x_train['teacher_prefix'])
#cv_tea_pre_encode = le.transform(x_cv['teacher_prefix'])
test_tea_pre_encode = le.transform(x_test['teacher_prefix'])
```

In [0]:

```
t_prefix_rank, unique,size = self_token(x_train['teacher_prefix'])
print(t_prefix_rank)
teacher_prefix_size =size
encoded_t_prefix_train = []
encoded_t_prefix_test = []
for prefix in x_train['teacher_prefix']:
    encoded_t_prefix_train.append(t_prefix_rank[prefix])

for prefix in x_test['teacher_prefix']:
    if prefix in unique:
        encoded_t_prefix_test.append(t_prefix_rank[prefix])
    else:
        encoded_t_prefix_test.append(0)

encoded_t_prefix_train = np.asarray(encoded_t_prefix_train)
encoded_t_prefix_test = np.asarray(encoded_t_prefix_test)
```

```
{'dr': 1, 'teacher': 2, 'mr': 3, 'ms': 4, 'mrs': 5}
```

In [0]:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

tr_pro_gra_encode = le.fit_transform(x_train['project_grade_category'])
#cv_pro_gra_encode = le.transform(x_cv['project_grade_category'])
test_pro_gra_encode = le.transform(x_test['project_grade_category'])
```

In [0]:

```
x_train.columns
```

Out[83]:

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

In [0]:

```
ins = []
concat = []
```

In [0]:

```
from keras.layers import Reshape,Concatenate
```

creating embedding layer for categorical data

In [0]:

```
# Unique values
# Unique values
tea_pre_uni = x_train['teacher_prefix'].nunique()
emb_tea_pre_size = int(np.ceil((tea_pre_uni) / 2))

# Creating an input layer
inp_tea_pre = Input(shape = (1,), name = "teacher_prefix")

# Creating an embedding layer
emb_tea_pre = Embedding(input_dim = tea_pre_uni, output_dim = emb_tea_pre_size,
                        trainable = True, name = "teacher_prefix_emb")(inp_tea_pre)

flatten_tea_pre = Flatten()(emb_tea_pre)
```

In [0]:

```
# Unique values
sch_uni = x_train['school_state'].nunique()
emb_sch_size = int(np.ceil((sch_uni) / 2))

# Creating an input layer
inp_sch = Input(shape = (1,), name = "school_state")

# Creating an embedding layer
emb_sch = Embedding(input_dim = sch_uni, output_dim = emb_sch_size,
                        trainable = True, name = "school_state_emb")(inp_sch)

flatten_sch = Flatten()(emb_sch)
```

In [0]:

```

# Unique values
pro_gra_uni = x_train['project_grade_category'].nunique()
emb_pro_gra_size = int(np.ceil((pro_gra_uni) / 2))

# Creating an input layer
inp_pro_gra = Input(shape = (1,), name = "project_grade_category")

# Creating an embedding layer
emb_pro_gra = Embedding(input_dim = pro_gra_uni, output_dim = emb_pro_gra_size,
                        trainable = True, name = "project_grade_category_emb")(inp_pro_gra

flatten_pro_gra = Flatten()(emb_pro_gra)
```

In [0]:

```
# Unique values
pro_sub_uni = x_train['clean_categories'].nunique()
emb_pro_sub_size = int(np.ceil((pro_sub_uni) / 2))

# Creating an input layer
inp_pro_sub = Input(shape = (1,), name = "clean_categories")

# Creating an embedding layer
emb_pro_sub = Embedding(input_dim = pro_sub_uni, output_dim = emb_pro_sub_size,
                        trainable = True, name = "project_subject_categories_emb")(inp_pro

flatten_pro_sub = Flatten()(emb_pro_sub)
```

In [0]:

```


# Unique values
pro_sub_1_uni = x_train['clean_subcategories'].nunique()


emb_pro_sub_1_size = int(min(np.ceil((pro_sub_1_uni) / 2), 50))

# Creating an input layer
inp_pro_sub_1 = Input(shape = (1,), name = "project_subject_subcategories")

# Creating an embedding layer
emb_pro_sub_1 = Embedding(input_dim = pro_sub_1_uni, output_dim = emb_pro_sub_1_size,
                        trainable = True, name = "project_subject_subcategories_emb")(inp_

flatten_pro_sub_1 = Flatten()(emb_pro_sub_1)
```

In [0]:

```

```

In [0]:

```
x = concatenate([flatten_1,flatten_2,flatten_3,flatten_4,flatten_5,flatten_6,dense_1])
```

Type *Markdown* and LaTeX: α^2

In [0]:

```
import keras.backend as K
K.clear_session()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:107: The name tf.reset_default_graph is deprecated.
Please use tf.compat.v1.reset_default_graph instead.
```

In [0]:

```
from keras.layers import Flatten
from keras.regularizers import l2
from keras.layers import concatenate
```

In [0]:

```
essay_input = Input(shape=(300,), name='essay_input')

x = Embedding(vocab_size, 300, weights=[embedding_matrix_train],trainable = False, input_le
lstm_out = LSTM(100,recurrent_dropout=0.5,return_sequences=True)(x)
flatten_1 = Flatten()(lstm_out)
```

In [0]:

```
# Creating an input layer
input_lay = Input(shape = (300, ), name = "Input_Text_Data")

# Creating an embedding layer
emb_lay = Embedding(input_dim = vocab_size, output_dim = 300,
                        input_length = 300, weights = [embedding_matrix_train],
                        trainable = False, name = "lstm_text_layer")(input_lay)

# Creating LSTM layer
emb_lay_text = LSTM(128, return_sequences = True, dropout = 0.3)(emb_lay)

flatten_1 = Flatten()(emb_lay_text)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:190: The name tf.get_default_session is deprecated.
Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please u
se tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:203: The name tf.Session is deprecated. Please use t
f.compat.v1.Session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:207: The name tf.global_variables is deprecated. Ple
ase use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:216: The name tf.is_variable_initialized is deprecat
ed. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:223: The name tf.variables_initializer is deprecate
d. Please use tf.compat.v1.variables_initializer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:3733: calling dropout (from tensorflow.python.ops.nn
_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

In [0]:

```
import keras
from tensorboardcolab import *
from keras.regularizers import l2
from keras.layers import LeakyReLU
```

In [0]:

```
rem_input_layer =  Input(shape=(3,), name="rem_input_layer")

rem_input_dense = Dense(64, activation='relu',kernel_initializer='glorot_normal',kernel_reg
```

In [0]:

```
remaining_input = Input(shape=(3,), name='remaining_input')
dense_1 = Dense(100, activation='relu',kernel_initializer="he_normal",kernel_regularizer=l2
```

In [0]:

```
from keras.layers import concatenate

con_lay = concatenate([flatten_1, flatten_tea_pre, flatten_sch, flatten_pro_gra, flatten_pr
```

In [0]:

```
#con_lay = concatenate([flatten_1, flatten_tea_pre, flatten_sch, flatten_pro_gra, flatten_p

'''state = Input(shape=(1,), name='school_state')
x = Embedding(state_size, 10, input_length=1)(state)
flatten_2 = Flatten()(x)


project_grade_category = Input(shape=(1,), name='project_grade_category')
x = Embedding(project_grade_categories_size, 10, input_length=1)(project_grade_category)
flatten_3 = Flatten()(x)


clean_categories = Input(shape=(1,), name='clean_categories')
x = Embedding(categories_size, 10, input_length=1)(clean_categories)
flatten_4 = Flatten()(x)


clean_sub_categories = Input(shape=(1,), name='clean_sub_categories')
x = Embedding(subcategories_size, 10, input_length=1)(clean_sub_categories)
flatten_5 = Flatten()(x)


teacher_prefix = Input(shape=(1,), name='teacher_prefix')
x = Embedding(teacher_prefix_size, 10, input_length=1)(teacher_prefix)
flatten_6 = Flatten()(x)'''


'''remaining_input = Input(shape=(3,), name='remaining_input')
dense_1 = Dense(1, activation='relu',kernel_initializer="he_normal",kernel_regularizer=l2(0

con_lay = concatenate([flatten_1, flatten_tea_pre, flatten_sch, flatten_pro_gra, flatten_pr

# Layer 1
m = Dense(256, activation = 'relu', kernel_regularizer = l2(0.01))(con_lay)
m = Dropout(0.3)(m)

# Layer 2
m = Dense(128, activation = 'relu', kernel_regularizer = l2(0.01))(m)
m = Dropout(0.3)(m)

# Layer 3
m = Dense(64, activation = 'relu', kernel_regularizer = l2(0.01))(m)
m = Dropout(0.3)(m)

# Layer 4
m = Dense(32, activation = 'relu', kernel_regularizer = l2(0.01))(m)
m = Dropout(0.3)(m)

# Output Layer
final_output = Dense(2, activation = 'softmax', name= 'model_1_output')(m)

model_1 = Model(inputs = [input_lay, inp_tea_pre, inp_sch, inp_pro_gra,
                          inp_pro_sub, inp_pro_sub_1, rem_input_layer],outputs=[final_output]
print(model_1.summary())
```

```
Model: "model_1"
__________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==========================================================================================
Input_Text_Data (InputLayer)    (None, 300)               0
__________________________________________________________________________________________
lstm_text_layer (Embedding)     (None, 300, 300)     15528900    Input_Text_Data[0][0]
__________________________________________________________________________________________
teacher_prefix (InputLayer)     (None, 1)                 0
__________________________________________________________________________________________
school_state (InputLayer)       (None, 1)                 0
__________________________________________________________________________________________
project_grade_category (InputLa (None, 1)                 0
__________________________________________________________________________________________
clean_categories (InputLayer)   (None, 1)                 0
__________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 300, 128)     219648      lstm_text_layer[0][0]
__________________________________________________________________________________________
teacher_prefix_emb (Embedding)  (None, 1, 3)         15          teacher_prefix[0][0]
__________________________________________________________________________________________
school_state_emb (Embedding)    (None, 1, 26)        1326        school_state[0][0]
__________________________________________________________________________________________
project_grade_category_emb (Emb (None, 1, 2)         8           project_grade_category[0][0]
__________________________________________________________________________________________
project_subject_categories_emb  (None, 1, 25)        1250        clean_categories[0][0]
__________________________________________________________________________________________
project_subject_subcategories_e (None, 1, 50)        19750       clean_categories[0][0]
__________________________________________________________________________________________
rem_input_layer (InputLayer)    (None, 3)                 0
__________________________________________________________________________________________
flatten_6 (Flatten)             (None, 38400)             0           lstm_1[0][0]
```

```
__________________________________________________________________________________________________
__________________________
flatten_1 (Flatten)             (None, 3)            0           teacher_pre
fix_emb[0][0]
__________________________________________________________________________________________________
__________________________
flatten_2 (Flatten)             (None, 26)           0           school_stat
e_emb[0][0]
__________________________________________________________________________________________________
__________________________
flatten_3 (Flatten)             (None, 2)            0           project_gra
de_category_emb[0][0]
__________________________________________________________________________________________________
__________________________
flatten_4 (Flatten)             (None, 25)           0           project_sub
ject_categories_emb[0]
__________________________________________________________________________________________________
__________________________
flatten_5 (Flatten)             (None, 50)           0           project_sub
ject_subcategories_emb
__________________________________________________________________________________________________
__________________________
dense_1 (Dense)                 (None, 64)           256         rem_input_l
ayer[0][0]
__________________________________________________________________________________________________
__________________________
concatenate_1 (Concatenate)     (None, 38570)        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_1[0]
[0]
__________________________________________________________________________________________________
__________________________
dense_2 (Dense)                 (None, 256)          9874176     concatenate
_1[0][0]
__________________________________________________________________________________________________
__________________________
dropout_1 (Dropout)             (None, 256)          0           dense_2[0]
[0]
__________________________________________________________________________________________________
__________________________
dense_3 (Dense)                 (None, 128)          32896       dropout_1
[0][0]
__________________________________________________________________________________________________
__________________________
dropout_2 (Dropout)             (None, 128)          0           dense_3[0]
[0]
__________________________________________________________________________________________________
__________________________
dense_4 (Dense)                 (None, 64)           8256        dropout_2
[0][0]
__________________________________________________________________________________________________
```

```
__________________________
dropout_3 (Dropout)             (None, 64)           0           dense_4[0]
[0]
__________________________________________________________________________
__________________________
dense_5 (Dense)                 (None, 32)           2080        dropout_3
[0][0]
__________________________________________________________________________
__________________________
dropout_4 (Dropout)             (None, 32)           0           dense_5[0]
[0]
__________________________________________________________________________
__________________________
model_1_output (Dense)          (None, 2)            66          dropout_4
[0][0]
==========================================================================
======================
Total params: 25,688,627
Trainable params: 10,159,727
Non-trainable params: 15,528,900
__________________________________________________________________________
__________________________
None
```

In [0]:

```
from keras import backend as K
from keras.callbacks import ModelCheckpoint,TensorBoard,ReduceLROnPlateau, EarlyStopping
import tensorflow as tf
#https://stackoverflow.com/posts/51734992/revisions
import tensorflow as tf
from sklearn.metrics import roc_auc_score

def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)
```

In [0]:

```

```

In [0]:

```
train_2 = [padded_train,tr_tea_pre_encode, tr_sch_encode, tr_pro_sub_encode, tr_pro_encode,
test_2 = [padded_test,test_tea_pre_encode, test_sch_encode, test_pro_sub_encode, test_pro_e
```

In [0]:

```

batch_size=512
```

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [0]:

```

```

In [0]:

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy',auro
h2 = model_1.fit(train_2, y_train, batch_size=512,epochs=10, verbose=1, validation_data=(te
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimiz
ers.py:793: The name tf.train.Optimizer is deprecated. Please use tf.compat.
v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:3576: The name tf.log is deprecated. Please use tf.m
ath.log instead.

WARNING:tensorflow:From <ipython-input-37-9acb5a10c946>:9: py_func (from ten
sorflow.python.ops.script_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor
to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_co
re/python/ops/math_grad.py:1424: where (from tensorflow.python.ops.array_op
s) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please u
se tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use t
f.compat.v1.assign instead.

Train on 87398 samples, validate on 21850 samples
Epoch 1/10
87398/87398 [==============================] - 109s 1ms/step - loss: 1.9295
- acc: 0.8387 - auroc: 0.6032 - val_loss: 0.6573 - val_acc: 0.8486 - val_aur
oc: 0.7041
Epoch 2/10
87398/87398 [==============================] - 106s 1ms/step - loss: 0.5169
- acc: 0.8486 - auroc: 0.6841 - val_loss: 0.4332 - val_acc: 0.8486 - val_aur
oc: 0.7214
Epoch 3/10
87398/87398 [==============================] - 105s 1ms/step - loss: 0.4235
- acc: 0.8486 - auroc: 0.7091 - val_loss: 0.4065 - val_acc: 0.8486 - val_aur
oc: 0.7380
Epoch 4/10
87398/87398 [==============================] - 105s 1ms/step - loss: 0.4061
- acc: 0.8486 - auroc: 0.7198 - val_loss: 0.4073 - val_acc: 0.8486 - val_aur
oc: 0.7414
Epoch 5/10
87398/87398 [==============================] - 104s 1ms/step - loss: 0.3996
```

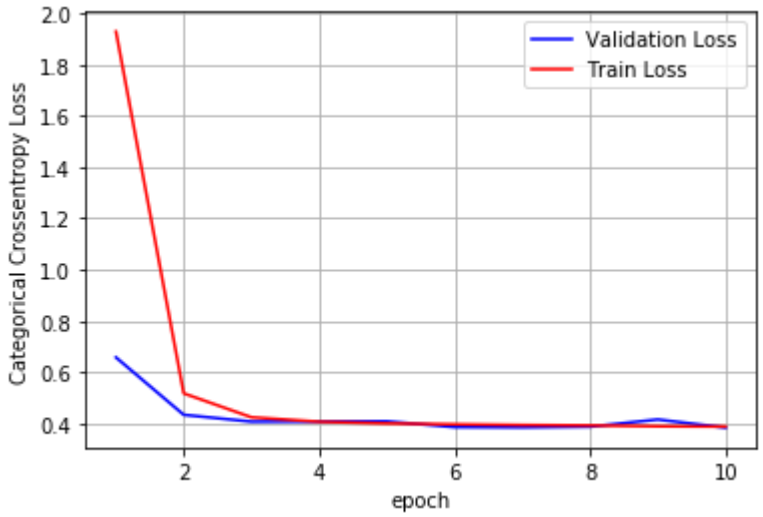
```
- acc: 0.8486 - auroc: 0.7279 - val_loss: 0.4075 - val_acc: 0.8486 - val_aur
oc: 0.7392
Epoch 6/10
87398/87398 [==============================] - 105s 1ms/step - loss: 0.3961
- acc: 0.8486 - auroc: 0.7346 - val_loss: 0.3852 - val_acc: 0.8486 - val_aur
oc: 0.7509
Epoch 7/10
87398/87398 [==============================] - 104s 1ms/step - loss: 0.3934
- acc: 0.8486 - auroc: 0.7402 - val_loss: 0.3841 - val_acc: 0.8486 - val_aur
oc: 0.7493
Epoch 8/10
87398/87398 [==============================] - 104s 1ms/step - loss: 0.3916
- acc: 0.8486 - auroc: 0.7482 - val_loss: 0.3868 - val_acc: 0.8486 - val_aur
oc: 0.7543
Epoch 9/10
87398/87398 [==============================] - 105s 1ms/step - loss: 0.3885
- acc: 0.8486 - auroc: 0.7517 - val_loss: 0.4148 - val_acc: 0.8486 - val_aur
oc: 0.7537
Epoch 10/10
87398/87398 [==============================] - 104s 1ms/step - loss: 0.3874
- acc: 0.8486 - auroc: 0.7562 - val_loss: 0.3831 - val_acc: 0.8486 - val_aur
oc: 0.7566
```

In [0]:

```
epochs=10
```

In [0]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs

vy = h2.history['val_loss']
ty = h2.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [0]:

```
result = model_1.evaluate( test_2,
          y_test,batch_size=512)
```

```
21850/21850 [==============================] - 11s 507us/step
```

In [0]:

```
#https://github.com/mrunal46/Donors-Choose-using-LSTM/blob/master/LSTM%20-%20model%202-Copy
print("Auc for test data: %0.3f"%roc_auc_score(y_test,model_1.predict(test_2)))
#print("Auc for CV data: %0.3f"%roc_auc_score(y_cv,model.predict([X_cv,X_cv_school_state,X_
          # X_cv_teacher_prefix,cv['remaining_input']])))
print("Auc for train data: %0.3f"%roc_auc_score(y_train,model_1.predict(train_2)))
```

```
Auc for test data: 0.757
Auc for train data: 0.787
```

In [0]:

```
result
```

Out[46]:

```
[0.3831118217240209, 0.8486041191622649, 0.7565998744059532]
```

In [0]:

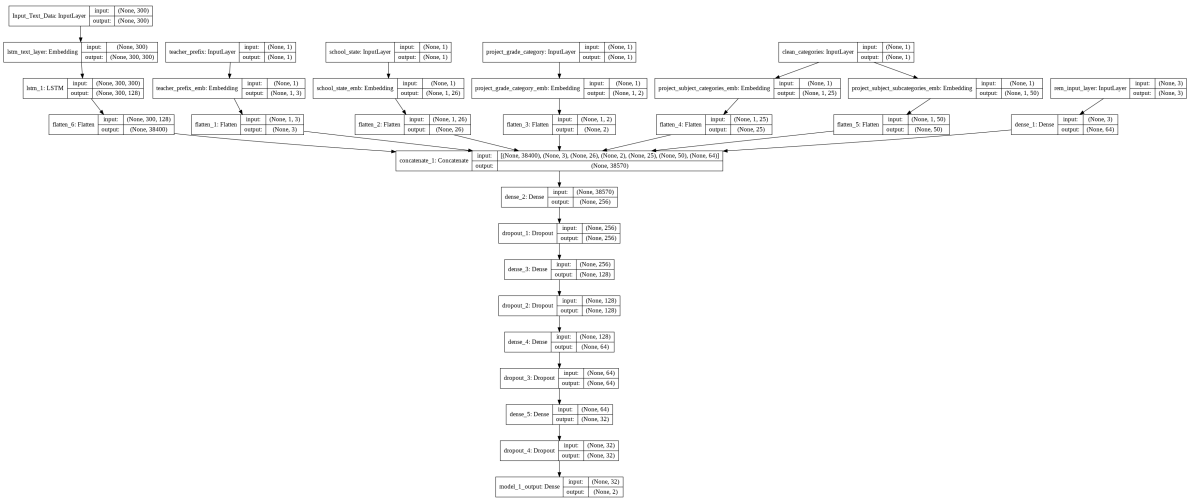
```
print("{} of test data {}". format(model.metrics_names[0],result[0]))
print("{} of test data {}". format(model.metrics_names[1],result[]))
```

```
loss of test data 0.18387332465301437
acc of test data 0.9689352181075697
```

In [0]:

```
#https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
from keras.utils.vis_utils import plot_model
plot_model(model_1, to_file='/content/drive/My Drive/1.png', show_shapes=True, show_layer_r
```

Out[49]:



In [0]:

```
x_test.shape
```

Out[95]:

```
(21850, 10)
```

##model 2

text essay

In [0]:

```
#https://stackoverflow.com/questions/45805493/sorting-tfidfvectorizer-output-by-tf-idf-lowe
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(analyzer='word', stop_words = 'english')

# fit_transform on training data
X_traintfidf = tfidf.fit_transform(x_train['essay'])
X_testtfidf=tfidf.transform(x_test['essay'])
```

In [0]:

```
# Zipping feature names corresponding to idf_ values

feat_idf = sorted(zip(tfidf.idf_, tfidf.get_feature_names()))
```

In [0]:

```
sort_idf = sorted(tfidf.idf_)

print("Mean of idf values:", np.mean(sort_idf))
print("Median of idf values:", np.median(sort_idf))
print("Maximum of idf values:", max(sort_idf))
print("Minimum of idf values:", min(sort_idf))
```

```
Mean of idf values: 10.364929043573493
Median of idf values: 11.279626831262464
Maximum of idf values: 11.685091939370627
Minimum of idf values: 1.007545645394076
```

In [0]:

```
q1 = np.percentile(sort_idf, 25)
q3 = np.percentile(sort_idf, 75)
```

In [0]:

```
print("25 percentile",q1)
print("75 percentile",q3)
```

```
25 percentile 9.670188918828364
75 percentile 11.685091939370627
```

In [0]:

```
iqr=q3-q1
```

In [0]:

```
list_words = []



for i in range(len(feat_idf)):

    if feat_idf[i][0] > iqr and feat_idf[i][0] < q3:
        words = feat_idf[i][1]
        list_words.append(words)
```

In [0]:

```
print("Number of words before taking IQR:", len(feat_idf))
print("Number of words after taking IQR:", len(list_words))
```

```
Number of words before taking IQR: 51350
Number of words after taking IQR: 31099
```

In [0]:

In [0]:

note

- all embedding layer of categorical data remail same

In [0]:

```
text=list_words
```

In [0]:

```
t = Tokenizer()
t.fit_on_texts(list(text))
vocab_size = len(t.word_index) + 1
print('Total unique words in the x_train',vocab_size)
encoded_train = t.texts_to_sequences(x_train['essay'])
encoded_test = t.texts_to_sequences(x_test['essay'])
```

```
Total unique words in the x_train 31100
```

In [0]:

```
max_length = 300
padded_train = pad_sequences(encoded_train, maxlen=max_length, padding='post')
padded_test = pad_sequences(encoded_test, maxlen=max_length, padding='post')
print("length of padded_train data",len(padded_train))
print("length of padded_test data",len(padded_test))
```

```
length of padded_train data 87398
length of padded_test data 21850
```

In [0]:

```
with open('/content/drive/My Drive/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

# for train
embedding_matrix_train = np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
    if word in glove_words:
        embedding_vector = model[word]
        embedding_matrix_train[i] = embedding_vector
```

In [0]:

```
essay_input = Input(shape=(300,), name='essay_input')

x = Embedding(vocab_size, 300, weights=[embedding_matrix_train], input_length=300,trainable
lstm_out = LSTM(100,recurrent_dropout=0.3,return_sequences=True)(x)
flatten_1 = Flatten()(lstm_out)
```

In [0]:

```
con_lay = concatenate([flatten_1, flatten_tea_pre, flatten_sch, flatten_pro_gra, flatten_pr
```

In [0]:

```
from keras.models import Model

# Layer 1
m_2 = Dense(256, activation = 'relu', kernel_regularizer = l2(0.01))(con_lay)
m_2 = Dropout(0.3)(m_2)

# Layer 2
m_2 = Dense(128, activation = 'relu', kernel_regularizer = l2(0.01))(m_2)
m_2 = Dropout(0.3)(m_2)

# Layer 3
m_2 = Dense(64, activation = 'relu', kernel_regularizer = l2(0.01))(m_2)
m_2 = Dropout(0.3)(m_2)

# Layer 3
m_2 = Dense(32, activation = 'relu', kernel_regularizer = l2(0.01))(m_2)
m_2 = Dropout(0.3)(m_2)

# Output Layer
final_output = Dense(2, activation = 'softmax', name= 'model_1_output')(m_2)

model_2 = Model(inputs = [essay_input, inp_tea_pre, inp_sch, inp_pro_gra,
                          inp_pro_sub, inp_pro_sub_1, rem_input_layer],outputs=[final_output]
print(model_2.summary())
```

```
Model: "model_2"
__________________________________________________________________________________
__________________________
Layer (type)                    Output Shape         Param #     Connected t
o
==================================================================================
======================
essay_input (InputLayer)        (None, 300)          0

__________________________________________________________________________________
__________________________
embedding_2 (Embedding)         (None, 300, 300)     9330000     essay_input
[0][0]
__________________________________________________________________________________
__________________________
teacher_prefix (InputLayer)     (None, 1)            0

__________________________________________________________________________________
__________________________
school_state (InputLayer)       (None, 1)            0

__________________________________________________________________________________
__________________________
project_grade_category (InputLa (None, 1)            0

__________________________________________________________________________________
__________________________
clean_categories (InputLayer)   (None, 1)            0

__________________________________________________________________________________
__________________________
```

```
lstm_2 (LSTM)                   (None, 300, 100)     160400      embedding_2
[0][0]
__________________________________________________________________________
_____________________
teacher_prefix_emb (Embedding)  (None, 1, 3)         15          teacher_pre
fix[0][0]
__________________________________________________________________________
_____________________
school_state_emb (Embedding)    (None, 1, 26)        1326        school_stat
e[0][0]
__________________________________________________________________________
_____________________
project_grade_category_emb (Emb (None, 1, 2)         8           project_gra
de_category[0][0]
__________________________________________________________________________
_____________________
project_subject_categories_emb  (None, 1, 26)        1326        clean_categ
ories[0][0]
__________________________________________________________________________
_____________________
project_subject_subcategories_e (None, 1, 50)        19900       clean_categ
ories[0][0]
__________________________________________________________________________
_____________________
rem_input_layer (InputLayer)    (None, 3)            0

__________________________________________________________________________
_____________________
flatten_12 (Flatten)            (None, 30000)        0           lstm_2[0]
[0]
__________________________________________________________________________
_____________________
flatten_7 (Flatten)             (None, 3)            0           teacher_pre
fix_emb[0][0]
__________________________________________________________________________
_____________________
flatten_8 (Flatten)             (None, 26)           0           school_stat
e_emb[0][0]
__________________________________________________________________________
_____________________
flatten_9 (Flatten)             (None, 2)            0           project_gra
de_category_emb[0][0]
__________________________________________________________________________
_____________________
flatten_10 (Flatten)            (None, 26)           0           project_sub
ject_categories_emb[0]
__________________________________________________________________________
_____________________
flatten_11 (Flatten)            (None, 50)           0           project_sub
ject_subcategories_emb
__________________________________________________________________________
_____________________
dense_1 (Dense)                 (None, 64)           256         rem_input_l
ayer[0][0]
__________________________________________________________________________
_____________________
concatenate_2 (Concatenate)     (None, 30171)        0           flatten_12
[0][0]
                                                                 flatten_7
[0][0]
                                                                 flatten_8
```

```
[0][0]
                                                                 flatten_9
[0][0]
                                                                 flatten_10
[0][0]
                                                                 flatten_11
[0][0]
                                                                 dense_1[0]
[0]
__________________________________________________________________________
_____________________
dense_10 (Dense)                (None, 256)          7724032     concatenate
_2[0][0]
__________________________________________________________________________
_____________________
dropout_9 (Dropout)             (None, 256)          0           dense_10[0]
[0]
__________________________________________________________________________
_____________________
dense_11 (Dense)                (None, 128)          32896       dropout_9
[0][0]
__________________________________________________________________________
_____________________
dropout_10 (Dropout)            (None, 128)          0           dense_11[0]
[0]
__________________________________________________________________________
_____________________
dense_12 (Dense)                (None, 64)           8256        dropout_10
[0][0]
__________________________________________________________________________
_____________________
dropout_11 (Dropout)            (None, 64)           0           dense_12[0]
[0]
__________________________________________________________________________
_____________________
dense_13 (Dense)                (None, 32)           2080        dropout_11
[0][0]
__________________________________________________________________________
_____________________
dropout_12 (Dropout)            (None, 32)           0           dense_13[0]
[0]
__________________________________________________________________________
_____________________
model_1_output (Dense)          (None, 2)            66          dropout_12
[0][0]
==========================================================================
======================
Total params: 17,280,561
Trainable params: 7,950,561
Non-trainable params: 9,330,000
__________________________________________________________________________
_____________________
None
```

In [0]:

```
import keras.backend as K
K.clear_session()
```

In [0]:

```
from keras import backend as K
from keras.callbacks import ModelCheckpoint,TensorBoard,ReduceLROnPlateau, EarlyStopping
import tensorflow as tf
#https://stackoverflow.com/posts/51734992/revisions
import tensorflow as tf
from sklearn.metrics import roc_auc_score

def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)
```

In [0]:

```
train_2 = [padded_train,tr_tea_pre_encode, tr_sch_encode, tr_pro_sub_encode, tr_pro_encode,
test_2 = [padded_test,test_tea_pre_encode, test_sch_encode, test_pro_sub_encode, test_pro_e
```

In [0]:

```
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import *
filepath="epochs:{epoch:03d}-val_acc:{val_auroc:.3f}.hdf5"
checkpoint_1 = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, mode='max')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5,
                        verbose=1, mode='auto', restore_best_weights=True)
```

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [0]:

```
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auroc])
h2 = model_2.fit(train_2, y_train, batch_size=512,epochs=10, verbose=1, validation_data=(te
```

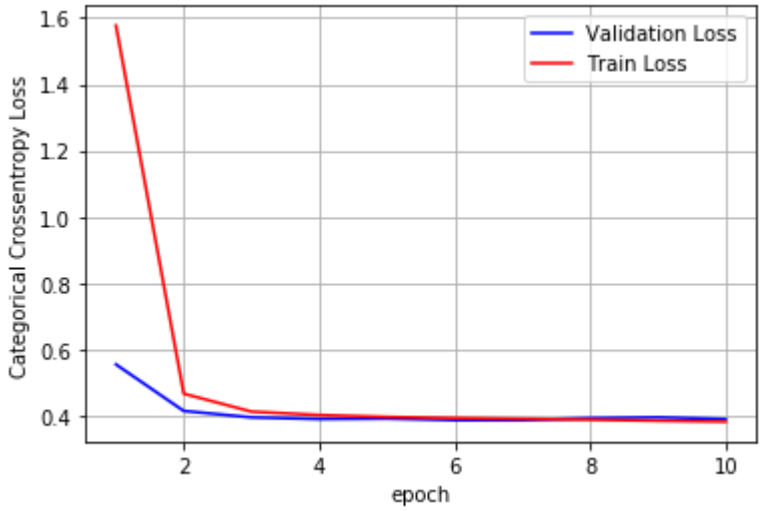
```
Train on 87398 samples, validate on 21850 samples
Epoch 1/10
87398/87398 [==============================] - 116s 1ms/step - loss: 1.5772
- auroc: 0.6478 - val_loss: 0.5544 - val_auroc: 0.7233
Epoch 2/10
87398/87398 [==============================] - 114s 1ms/step - loss: 0.4661
- auroc: 0.7081 - val_loss: 0.4140 - val_auroc: 0.7349
Epoch 3/10
87398/87398 [==============================] - 115s 1ms/step - loss: 0.4119
- auroc: 0.7207 - val_loss: 0.3945 - val_auroc: 0.7409
Epoch 4/10
87398/87398 [==============================] - 114s 1ms/step - loss: 0.4016
- auroc: 0.7275 - val_loss: 0.3896 - val_auroc: 0.7422
Epoch 5/10
87398/87398 [==============================] - 114s 1ms/step - loss: 0.3958
- auroc: 0.7355 - val_loss: 0.3908 - val_auroc: 0.7459
Epoch 6/10
87398/87398 [==============================] - 114s 1ms/step - loss: 0.3922
- auroc: 0.7440 - val_loss: 0.3861 - val_auroc: 0.7460
Epoch 7/10
87398/87398 [==============================] - 113s 1ms/step - loss: 0.3897
- auroc: 0.7485 - val_loss: 0.3860 - val_auroc: 0.7488
Epoch 8/10
87398/87398 [==============================] - 113s 1ms/step - loss: 0.3875
- auroc: 0.7530 - val_loss: 0.3918 - val_auroc: 0.7505
Epoch 9/10
87398/87398 [==============================] - 111s 1ms/step - loss: 0.3844
- auroc: 0.7587 - val_loss: 0.3946 - val_auroc: 0.7474
Epoch 10/10
87398/87398 [==============================] - 111s 1ms/step - loss: 0.3817
- auroc: 0.7645 - val_loss: 0.3891 - val_auroc: 0.7477
```

In [0]:

```
epochs=10
```

In [0]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs

vy = h2.history['val_loss']
ty = h2.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [0]:

```
result_1 = model_2.evaluate( test_2,
          y_test,batch_size=300)
result_2 = model_2.evaluate( train_2,
          y_train,batch_size=300)
```

```
21850/21850 [==============================] - 18s 824us/step
87398/87398 [==============================] - 71s 817us/step
```

In [0]:

```
print(result_1)
print(result_2)
```

```
[0.38907690329191613, 0.7471514321091453]
[0.36438259380835275, 0.7953995821825852]
```

model_3

In [0]:

```
```

text

In [0]:

```
#https://subscription.packtpub.com/book/application_development/9781782167853/1/ch01lvl1sec
t = Tokenizer()
t.fit_on_texts(x_train['essay'])
vocab_size = len(t.word_index) + 1
print('Total unique words in the x_train',vocab_size)
encoded_train = t.texts_to_sequences(x_train['essay'])
encoded_test = t.texts_to_sequences(x_test['essay'])
```

```
Total unique words in the x_train 51508
```

In [0]:

```
max_length = 300
padded_train = pad_sequences(encoded_train, maxlen=max_length, padding='post')
padded_test = pad_sequences(encoded_test, maxlen=max_length, padding='post')
print("length of padded_train data",len(padded_train))
print("length of padded_test data",len(padded_test))
```

```
length of padded_train data 87398
length of padded_test data 21850
```

categorical data

In [0]:

```
```

In [0]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in x_train['clean_categories'].values:
    my_counter.update(word.split())


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))

# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, bina
vectorizer.fit(x_train['clean_categories'].values)
#print(vectorizer.get_feature_names())


categories_one_hot_train = vectorizer.transform(x_train['clean_categories'].values)
categories_one_hot_test = vectorizer.transform(x_test['clean_categories'].values)
print("Shape of matrix after one hot encodig ",categories_one_hot_train.shape)
print("Shape of matrix after one hot encodig ",categories_one_hot_test.shape)
```

```
Shape of matrix after one hot encodig  (87398, 9)
Shape of matrix after one hot encodig  (21850, 9)
```

In [0]:

```

```

In [0]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in x_train['clean_subcategories'].values:
    my_counter.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False,
vectorizer.fit(x_train['clean_categories'].values)
#print(vectorizer.get_feature_names())


categories_one_hot_train2 = vectorizer.transform(x_train['clean_subcategories'].values)
categories_one_hot_test2 = vectorizer.transform(x_test['clean_subcategories'].values)
print("Shape of matrix after one hot encodig ",categories_one_hot_train2.shape)
print("Shape of matrix after one hot encodig ",categories_one_hot_test2.shape)
```

```
Shape of matrix after one hot encodig  (87398, 30)
Shape of matrix after one hot encodig  (21850, 30)
```

In [0]:

```
'''encode categorical feature school_state'''


from collections import Counter
my_counter = Counter()
for word in x_train['school_state'].values:
    my_counter.update(word.split(" "))

state_dict = dict(my_counter)
state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))

vectorizer = CountVectorizer(vocabulary=list(state_dict.keys()), lowercase=False, binary=Tr
vectorizer.fit(x_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(x_train['school_state'].values)
#X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = 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(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(87398, 51) (87398, 2)
(21850, 51) (21850, 2)
['vt', 'wy', 'nd', 'mt', 'ri', 'sd', 'ne', 'de', 'ak', 'nh', 'wv', 'me', 'h
i', 'dc', 'nm', 'ks', 'ia', 'id', 'ar', 'co', 'mn', 'or', 'ms', 'ky', 'nv',
'md', 'ct', 'tn', 'al', 'ut', 'wi', 'va', 'az', 'nj', 'ok', 'wa', 'la', 'm
a', 'oh', 'in', 'mo', 'pa', 'mi', 'sc', 'ga', 'il', 'nc', 'fl', 'ny', 'tx',
'ca']
====================================================================================
========================
```

In [0]:

```
'''encode categorical feature project_grade_category'''

from collections import Counter
my_counter = Counter()
for word in x_train['project_grade_category'].values:
    my_counter.update(word.split(","))

grade_dict = dict(my_counter)
grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
vectorizer = CountVectorizer(vocabulary=list(grade_dict.keys()), lowercase=False, binary=Tr



vectorizer.fit(x_train['project_grade_category'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(x_train['project_grade_category'].values)
#X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(x_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
#print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(87398, 4) (87398, 2)
(21850, 4) (21850, 2)
['grades_9_12', 'grades_6_8', 'grades_3_5', 'grades_prek_2']
====================================================================================
========================
```

In [0]:

```
from collections import Counter
my_counter = Counter()
for word in x_train['teacher_prefix'].values:
    my_counter.update(word.split())

prefix = dict(my_counter)
prefix = dict(sorted(prefix.items(), key=lambda kv: kv[1]))

# code is taken from this notebook

'''encode categorical feature teacher_prefix'''
vectorizer = CountVectorizer(vocabulary=list(prefix.keys()), lowercase=False, binary=True)
vectorizer.fit(x_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(x_train['teacher_prefix'].values.astype('U'))
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))#https:/
X_test_teacher_ohe = vectorizer.transform(x_test['teacher_prefix'].values.astype('U'))

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)

print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(87398, 5) (87398, 2)
(21850, 5) (21850, 2)
['dr', 'teacher', 'mr', 'ms', 'mrs']
====================================================================================================
========================
```

numerical

In [0]:

In [0]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

'''encode numerical feature price'''
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(x_train['price'].values.reshape(1,-1)) # use code from sample

X_train_price_norm = normalizer.transform(x_train['price'].values.reshape(-1,1))
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.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
(87398, 1) (87398, 2)
(21850, 1) (21850, 2)
====================================================================================================
```

In [0]:

```
'''encode numerical feature teacher_number_of_previously_posted_projects'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)

X_train_posted_norm= normalizer.transform(x_train['teacher_number_of_previously_posted_proj
#X_cv_posted_norm= normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'
X_test_posted_norm = normalizer.transform(x_test['teacher_number_of_previously_posted_proje

print("After vectorizations")
print(X_train_posted_norm.shape, y_train.shape)

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

```
After vectorizations
(87398, 1) (87398, 2)
(21850, 1) (21850, 2)
====================================================================================================
```

In [0]:

```
'''encode numerical feature teacher_number_of_previously_posted_projects'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(x_train['quantity'].values.reshape(1,-1))

X_train_quan_norm= normalizer.transform(x_train['quantity'].values.reshape(-1,1))
#X_cv_quan_norm= normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quan_norm = normalizer.transform(x_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_posted_norm.shape, y_train.shape)

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

```
After vectorizations
(87398, 1) (87398, 2)
(21850, 1) (21850, 2)
====================================================================================
========================
```

In [0]:

```
#merge all features
from scipy.sparse import hstack
X_tr = hstack(( X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_nc
#X_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_pric
X_te = hstack(( X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm,c

print("Final Data matrix")
print("train matrix=>",X_tr.shape, y_train.shape)
#print("cv matrix=>",X_cr.shape, y_cv.shape)
print("test matrix=>",X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
train matrix=> (87398, 102) (87398, 2)
test matrix=> (21850, 102) (21850, 2)
====================================================================================
========================
```

In [0]:

```
rest_features_train = np.expand_dims(X_tr,2)
rest_features_test = np.expand_dims(X_te,2)
```

In [0]:

```
print(rest_features_test.shape)
print(rest_features_train.shape)
```

```
(21850, 102, 1)
(87398, 102, 1)
```

In [0]:

```
X_tr=rest_features_train
X_te=rest_features_test
```

In [0]:

```
from keras.layers import Conv1D, MaxPooling2D
```

In [0]:

```
essay_input = Input(shape=(300,), name='essay_input')

x = Embedding(vocab_size, 300, weights=[embedding_matrix_train], input_length=300,trainable
lstm_out = LSTM(100,recurrent_dropout=0.5,return_sequences=True)(x)
flatten_1 = Flatten()(lstm_out)

#other=Input(shape=(102,1), name='other')

other = Input(shape=(X_tr.shape[1],1), name="other")
from keras.layers import Dense, Dropout, Flatten, Conv1D, MaxPooling1D, Activation



# Block 1
con1 = Conv1D(64, kernel_size = 3, activation = 'relu', name = 'block_1')(other)

# Block 2
con2 = Conv1D(64, 3, activation='relu', padding = 'same', name = 'block_2')(con1)

# Block 3
con3 = Conv1D(32, 3, activation='softmax', padding = 'same',  name = 'block_3')(con2)

# Block 4
con4 = Conv1D(32, 3, activation='softmax', padding = 'same',  name = 'block_4')(con3)

# Flattening
flatten_2 = Flatten()(con4)
#flatten_2 = Flatten()(rem_conv2)


x = concatenate([flatten_1,flatten_2])




# Layer 1
m_3 = Dense(256, activation = 'relu', kernel_regularizer = l2(0.01))(x)
m_3 = Dropout(0.3)(m_3)

# Layer 2
m_3 = Dense(128, activation = 'relu', kernel_regularizer = l2(0.01))(m_3)
m_3 = Dropout(0.3)(m_3)

# Layer 3
m_3 = Dense(64, activation = 'relu', kernel_regularizer = l2(0.01))(m_3)
m_3 = Dropout(0.3)(m_3)

# Layer 4
m_3 = Dense(32, activation = 'relu', kernel_regularizer = l2(0.01))(m_3)
m_3 = Dropout(0.3)(m_3)

x = Dense(64, activation='relu',kernel_initializer="he_normal",kernel_regularizer=l2(0.001)
final_output = Dense(2, activation='softmax',kernel_initializer="he_normal")(x)

model = Model(inputs=[essay_input,other], outputs=[final_output])
print(model.summary())
```

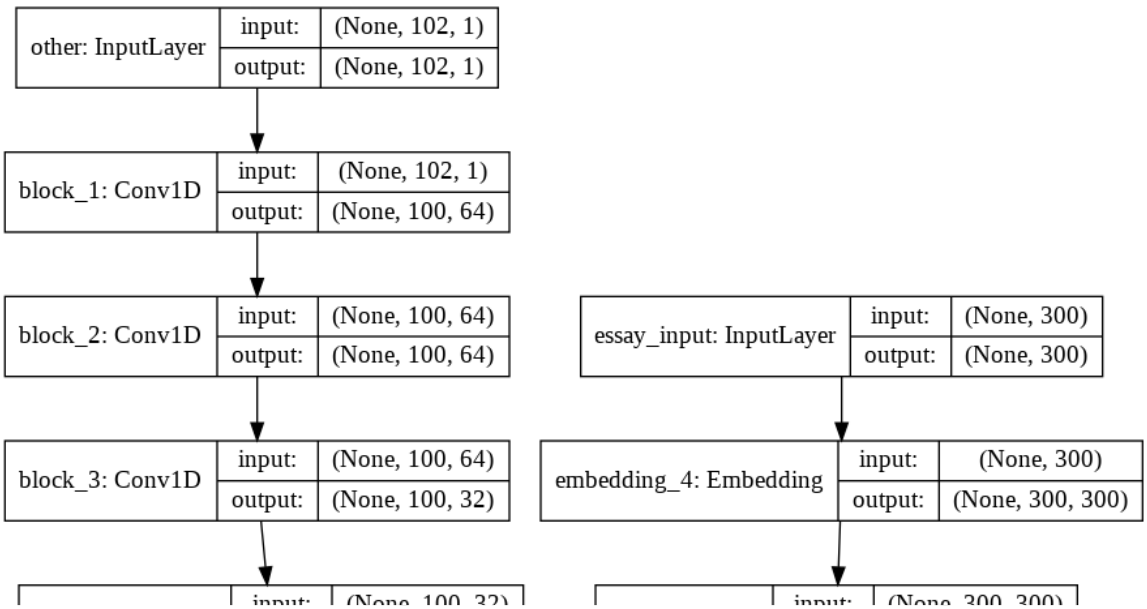
```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
other (InputLayer)              (None, 102, 1)       0
__________________________________________________________________________________________________
block_1 (Conv1D)                (None, 100, 64)      256         other[0][0]
__________________________________________________________________________________________________
essay_input (InputLayer)        (None, 300)          0
__________________________________________________________________________________________________
block_2 (Conv1D)                (None, 100, 64)      12352       block_1[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 300, 300)     15452400    essay_input[0][0]
__________________________________________________________________________________________________
block_3 (Conv1D)                (None, 100, 32)      6176        block_2[0][0]
__________________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 300, 100)     160400      embedding_4[0][0]
__________________________________________________________________________________________________
block_4 (Conv1D)                (None, 100, 32)      3104        block_3[0][0]
__________________________________________________________________________________________________
flatten_7 (Flatten)             (None, 30000)        0           lstm_4[0][0]
__________________________________________________________________________________________________
flatten_8 (Flatten)             (None, 3200)         0           block_4[0][0]
__________________________________________________________________________________________________
concatenate_4 (Concatenate)     (None, 33200)        0           flatten_7[0][0]
                                                                 flatten_8[0][0]
__________________________________________________________________________________________________
dense_13 (Dense)                (None, 256)          8499456     concatenate_4[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 256)          0           dense_13[0]
```

```
[0]
__________________________________________________________________________________
_____________________
dense_14 (Dense)                (None, 128)          32896       dropout_9
[0][0]
__________________________________________________________________________________
_____________________
dropout_10 (Dropout)            (None, 128)          0           dense_14[0]
[0]
__________________________________________________________________________________
_____________________
dense_15 (Dense)                (None, 64)           8256        dropout_10
[0][0]
__________________________________________________________________________________
_____________________
dropout_11 (Dropout)            (None, 64)           0           dense_15[0]
[0]
__________________________________________________________________________________
_____________________
dense_16 (Dense)                (None, 32)           2080        dropout_11
[0][0]
__________________________________________________________________________________
_____________________
dropout_12 (Dropout)            (None, 32)           0           dense_16[0]
[0]
__________________________________________________________________________________
_____________________
dense_17 (Dense)                (None, 64)           2112        dropout_12
[0][0]
__________________________________________________________________________________
_____________________
dense_18 (Dense)                (None, 2)            130         dense_17[0]
[0]
==================================================================================
======================
Total params: 24,179,618
Trainable params: 8,727,218
Non-trainable params: 15,452,400
__________________________________________________________________________________
_____________________
None
```

In [0]:

```
#https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/My Drive/model3.png', show_shapes=True, show_laye
```

Out[104]:



In [0]:

```
from keras import backend as K
from keras.callbacks import ModelCheckpoint,TensorBoard,ReduceLROnPlateau, EarlyStopping
import tensorflow as tf
#https://stackoverflow.com/posts/51734992/revisions
import tensorflow as tf
from sklearn.metrics import roc_auc_score

def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)
```

In [0]:

```
train_2 = [padded_train,X_tr]
test_2 = [padded_test,X_te]
```

In [0]:

```

```

In [0]:

```
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import *
filepath="epochs:{epoch:03d}-val_acc:{val_auroc:.3f}.hdf5"
checkpoint_1 = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, mode='max')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5,
                        verbose=1, mode='auto', restore_best_weights=True)
```

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [0]:

```
X_te.shape
```

Out[74]:

```
(21850, 102)
```

In [0]:

```
X_trn = np.expand_dims(X_tr, 2)
```

In [0]:

```
X_tr.shape
```

Out[57]:

```
(87398, 102, 1)
```

In [0]:

```
from numpy import zeros, newaxis
X_te=X_te.toarray()
X_te= X_te[:, :, newaxis]
#x_cv_rem_reshape = np.array(x_cv_rem).reshape(17480, 102,1)
```

In [0]:

```
X_te.shape
```

Out[43]:

```
(21850, 102, 1)
```

In [0]:

```
X_tr.shape
```

Out[99]:

```
(87398, 102, 1)
```

In [0]:

```
from tensorflow.keras.callbacks import TensorBoard
```

In [0]:

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auroc])
h2 = model.fit(train_2, y_train, batch_size=300,epochs=10, verbose=1, validation_data=(test
```

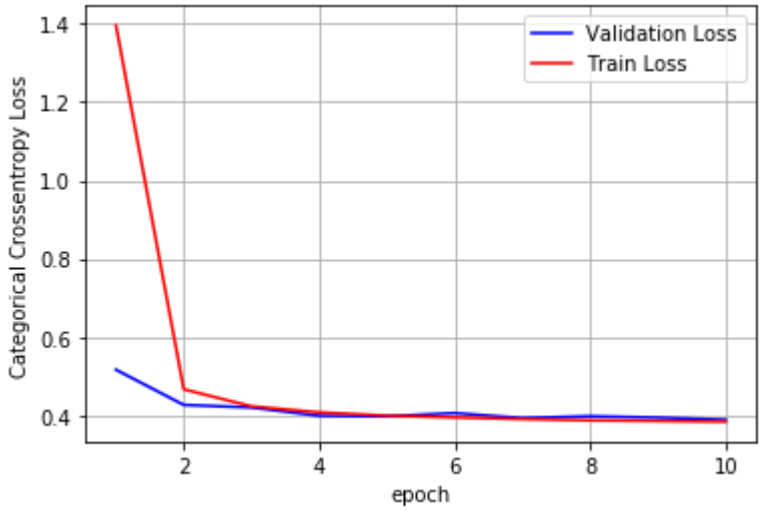
```
Train on 87398 samples, validate on 21850 samples
Epoch 1/10
87398/87398 [==============================] - 207s 2ms/step - loss: 1.3950
- auroc: 0.6431 - val_loss: 0.5170 - val_auroc: 0.7216
Epoch 2/10
87398/87398 [==============================] - 204s 2ms/step - loss: 0.4670
- auroc: 0.7139 - val_loss: 0.4271 - val_auroc: 0.7401
Epoch 3/10
87398/87398 [==============================] - 204s 2ms/step - loss: 0.4235
- auroc: 0.7269 - val_loss: 0.4205 - val_auroc: 0.7442
Epoch 4/10
87398/87398 [==============================] - 204s 2ms/step - loss: 0.4081
- auroc: 0.7378 - val_loss: 0.3995 - val_auroc: 0.7455
Epoch 5/10
87398/87398 [==============================] - 202s 2ms/step - loss: 0.3996
- auroc: 0.7453 - val_loss: 0.3980 - val_auroc: 0.7525
Epoch 6/10
87398/87398 [==============================] - 202s 2ms/step - loss: 0.3953
- auroc: 0.7490 - val_loss: 0.4062 - val_auroc: 0.7540
Epoch 7/10
87398/87398 [==============================] - 203s 2ms/step - loss: 0.3909
- auroc: 0.7563 - val_loss: 0.3935 - val_auroc: 0.7614
Epoch 8/10
87398/87398 [==============================] - 202s 2ms/step - loss: 0.3878
- auroc: 0.7590 - val_loss: 0.3982 - val_auroc: 0.7588
Epoch 9/10
87398/87398 [==============================] - 202s 2ms/step - loss: 0.3862
- auroc: 0.7619 - val_loss: 0.3947 - val_auroc: 0.7610
Epoch 10/10
87398/87398 [==============================] - 201s 2ms/step - loss: 0.3844
- auroc: 0.7630 - val_loss: 0.3890 - val_auroc: 0.7557
```

In [0]:

```
epochs=10
```

In [0]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs

vy = h2.history['val_loss']
ty = h2.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [0]:

```
result_1 = model.evaluate( test_2,
          y_test,batch_size=300)
result_2 = model.evaluate( train_2,
          y_train,batch_size=300)
```

```
21850/21850 [==============================] - 20s 900us/step
87398/87398 [==============================] - 78s 889us/step
```

In [0]:

```
print(result_1)
print(result_2)
```

```
[0.38902702382009147, 0.7557097569848487]
[0.37026190312547774, 0.7861486324860277]
```

In [0]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["model", "train_auc","test_auc"]
```

In [0]:

```
x.add_row(['model_1',0.787,0.757])
x.add_row(['model_2',0.795,0.747])
x.add_row(['model_3',0.786,0.755])
```

In [3]:

```
print(x)
```

```
+---------+-----------+----------+
|  model  | train_auc | test_auc |
+---------+-----------+----------+
| model_1 |   0.787   |  0.757   |
| model_2 |   0.795   |  0.747   |
| model_3 |   0.786   |  0.755   |
+---------+-----------+----------+
```

Type *Markdown* and LaTeX: α^2

note

- refercence used
- https://github.com/ravi-1654003/LSTM-DonorsChoose/blob/master/LSTM_DonorsChoose.ipynb (https://github.com/ravi-1654003/LSTM-DonorsChoose/blob/master/LSTM_DonorsChoose.ipynb)
- https://github.com/sandeepburra/LSTM-Donors-Choose/blob/master/LSTM.ipynb (https://github.com/sandeepburra/LSTM-Donors-Choose/blob/master/LSTM.ipynb)
- https://github.com/mrunal46/Donors-Choose-using-LSTM/blob/masterLSTM%20on%20Donor's%20Choose%20-%20Model%201-Copy1.ipynb (https://github.com/mrunal46/Donors-Choose-using-LSTM/blob/masterLSTM%20on%20Donor's%20Choose%20-%20Model%201-Copy1.ipynb)
- https://github.com/richardxing/DonorsChoose/blob/master/DonorsChoose-RNN.ipynb (https://github.com/richardxing/DonorsChoose/blob/master/DonorsChoose-RNN.ipynb)
- https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526 (https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526) - https://medium.com/@satnalikamayank12/on-learning-embeddings-for-categorical-data-using-keras-165ff2773fc9 (https://medium.com/@satnalikamayank12/on-learning-embeddings-for-categorical-data-using-keras-165ff2773fc9)
- https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ (https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/)