Assignment : 14

1. Preprocess all the Data we have in DonorsChoose Dataset use train.csv
2. Combine 4 essay's into one column named - 'preprocessed_essays'.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check this for using auc as a metric
5. You are free to choose any number of layers/hiddden units but you have to use same ty
6. You can use any one of the optimizers and choice of Learning rate and momentum, resou
7. For all the model's use TensorBoard and plot the Metric value and Loss with epoch. Wh
8. Use Categorical Cross Entropy as Loss to minimize.

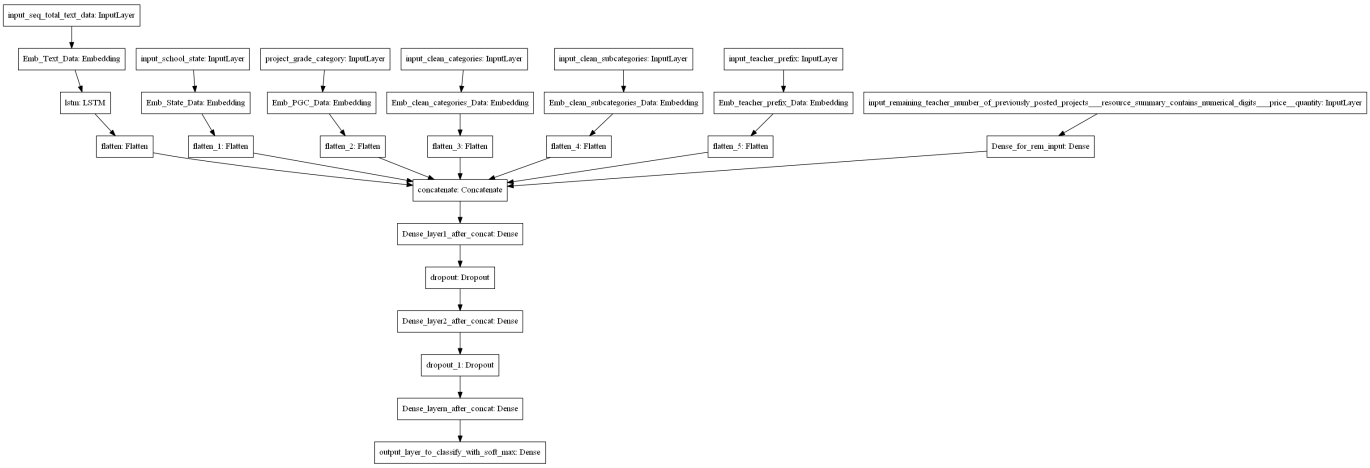
```
#you can use gdown modules to import dataset for the assignment
#for importing any file from drive to Colab you can write the syntax as !gdown --id file_i
#you can run the below cell to import the required preprocessed data.csv file and glove ve
```

```
!gdown --id 1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
!gdown --id 1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_
```

```
Downloading...
From: https://drive.google.com/uc?id=1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
To: /content/preprocessed_data.csv
100% 124M/124M [00:02<00:00, 51.4MB/s]
Downloading...
From: https://drive.google.com/uc?id=1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_
To: /content/glove_vectors
100% 128M/128M [00:03<00:00, 35.1MB/s]
```

Model-1

Build and Train deep neural network as shown below



ref: https://i.imgur.com/w395Yk9.png

- **Input_seq_total_text_data** --- You have to give Total text data columns. After this use the Embedding layer to get word vectors. Use given predefined glove word vectors, don't train any word vectors. After this use LSTM and get the LSTM output and Flatten that output.
- **Input_school_state** --- Give 'school_state' column as input to embedding layer and Train the Keras Embedding layer.
- **Project_grade_category** --- Give 'project_grade_category' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_categories** --- Give 'input_clean_categories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_clean_subcategories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_teacher_prefix' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_remaining_teacher_number_of_previously_posted_projects._resource_summary_contains_numerical_digits._price._quantity** ---concatenate remaining columns and add a Dense layer after that.

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

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

```
# https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
input_layer = Input(shape=(n,))
embedding = Embedding(no_1, no_2, input_length=n)(input_layer)
flatten = Flatten()(embedding)
```

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

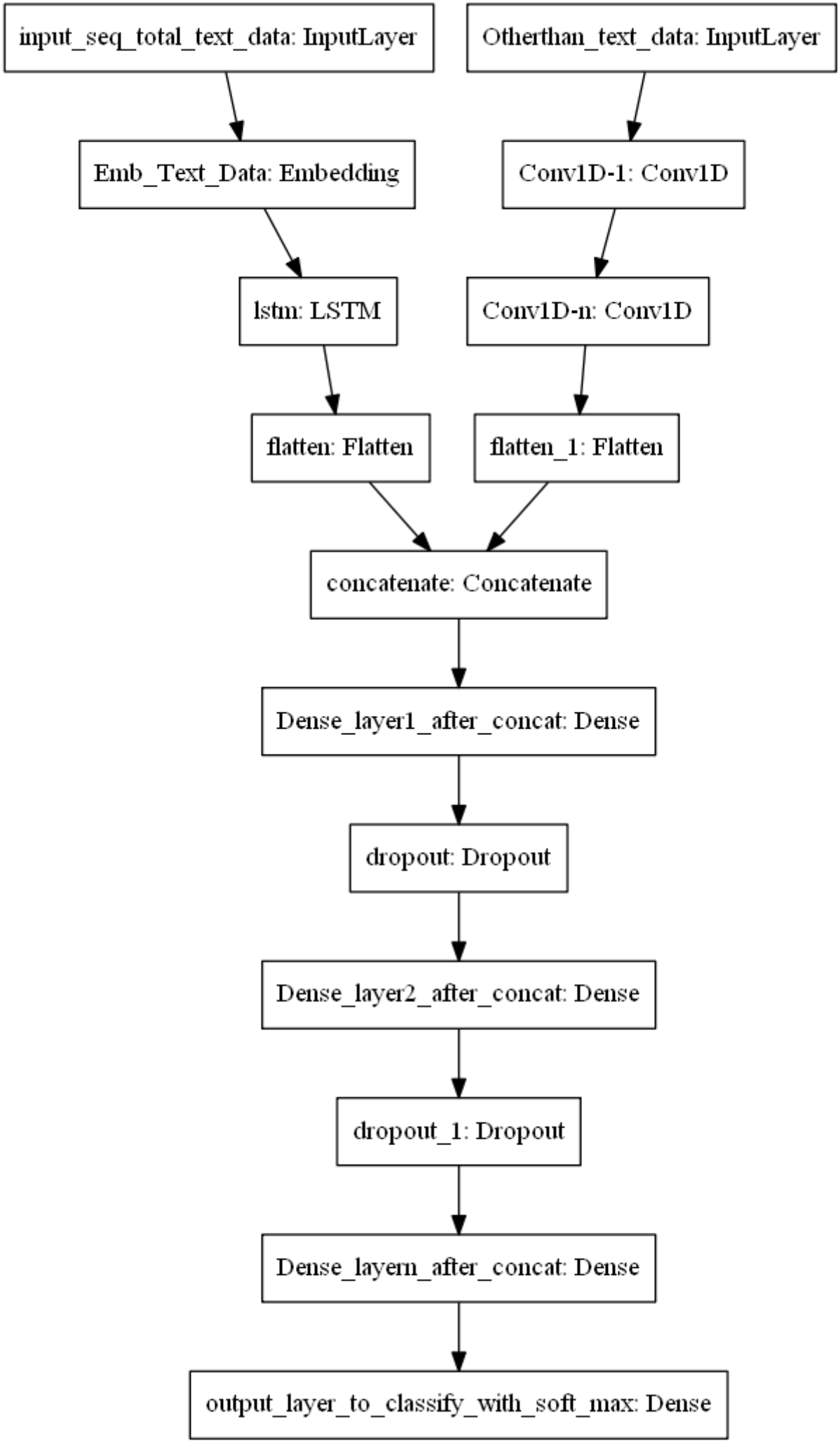
2. Please go through this link https://keras.io/getting-started/functional-api-guide/ and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.

Model-2

Use the same model as above but for 'input_seq_total_text_data' give only some words in the sentance not all the words. Filter the words as below.

```
1. Train the TF-IDF on the Train data

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

3. Remove the low idf value and high idf value words from our data. Do some analysis on

4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train on
```

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

- **input_seq_total_text_data**:

  . Use text column('essay'), and use the Embedding layer to get word ve

  . Use given predefined glove word vectors, don't train any word vector

  . Use LSTM that is given above, get the LSTM output and Flatten that o

  . 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 concate

  . Neumerical values and use CNN1D as shown in above figure.

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

Loading Preprocessed data

```
# import all the libraries
#make sure that you import your libraries from tf.keras and not just keras
import tensorflow
from tensorflow.keras.layers import Input,Dense,LSTM


#read the csv file
import pandas as pd
df = pd.read_csv('preprocessed_data.csv',nrows=100000)


df.columns
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
```

```
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

```
X = df.drop(['project_is_approved'],axis=1)
Y = df['project_is_approved']
X.head(2)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previousl
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, stratify =Y)
#X_train, X_cv , y_train, y_cv  = train_test_split(X_train,y_train, test_size =0.25, strat
```

```
from numpy import zeros
import os
import sys
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Input, Flatten, LSTM, GlobalMaxPool1D
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.initializers import Constant
import pickle
from tqdm import tqdm
import tensorflow as tf
from tensorflow.keras import layers
```

```
from google.colab import files
files=files.upload()
```

Choose Files glove_vectors

- **glove_vectors**(n/a) - 127506004 bytes, last modified: 5/29/2019 - 100% done

```
Saving glove_vectors to glove_vectors (1)
```

Model-1

Essay Feature

```
# Apply tokenizer
t = Tokenizer()
t.fit_on_texts(X_train['essay'].values)
vocab_size = len(t.word_index) + 1

print(vocab_size)

X_train_essay_pre_seq = t.texts_to_sequences(X_train['essay'].values)
X_test_essay_pre_seq = t.texts_to_sequences(X_test['essay'].values)

# padd sequence

max_length = 600
X_train_essay_pad_seq = pad_sequences(X_train_essay_pre_seq,maxlen=max_length)
X_test_essay_pad_seq = pad_sequences(X_test_essay_pre_seq,maxlen=max_length)

embeddings_index = dict()

f = open("glove_vectors", "rb")
text = f.read().decode(errors='replace')


for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

# create a weight matrix for words in training docs
embedding_matrix = zeros((vocab_size, 300))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector



essay_input = Input(shape=(600,), name="essay")
essay_feature = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=4, tra
#essay_feature = Embedding(input_dim=vocab_size + 1, output_dim= 300, weights=[embedding_m
essay_feature = LSTM(100)(essay_feature)
essay_feature = Flatten()(essay_feature)
```

```
49918
```

```
embedding_matrix
```

```
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.15243   , -0.16945   , -0.022748  , ...,  0.61800998,
         0.41281   ,  0.0010077 ],
       [-0.043504  , -0.18483999, -0.14613   , ...,  0.1008    ,
         0.1068    ,  0.089065  ],
       ...,
       [ 0.053373  ,  0.63700998, -0.13455001, ..., -0.32295001,
         0.34681001,  0.20094   ],
       [ 0.13623001,  0.73717999,  0.68703002, ..., -0.11312   ,
         0.18147001,  0.21788999],
       [ 0.11642   , -0.10357   , -0.16644   , ...,  0.24579   ,
         0.20392001, -0.27200001]])
```

Input_school_state

```
# Apply tokenizer
t2 = Tokenizer()
t2.fit_on_texts(X_train['school_state'].values)
vocab_size = len(t2.word_index) + 1

X_train_school_state_pre_seq = t2.texts_to_sequences(X_train['school_state'])
X_test_school_state_pre_seq = t2.texts_to_sequences(X_test['school_state'])

max_length = 1
X_train_school_state_pad_seq = pad_sequences(X_train_school_state_pre_seq,maxlen=max_lengt
X_test_school_state_pad_seq = pad_sequences(X_test_school_state_pre_seq,maxlen=max_length)

print(vocab_size)

embedding_size = min(vocab_size ,50)
school_state_input = Input(shape=(1,), name="school_state")
school_state_feature = Embedding(input_dim=vocab_size +1,output_dim=embedding_size)(school
#school_state_feature = Embedding(input_dim = vocab_size+1, output_dim = 2, input_length =
school_state_feature = Flatten()(school_state_feature)
```

```
52
```

```
X_train_school_state_pad_seq.shape
```

```
(80000, 1)
```

Project_grade_category

```
# Apply tokenizer
t3 = Tokenizer()
t3.fit_on_texts(X_train['project_grade_category'].values)
```

```
vocab_size = len(t3.word_index) + 1

print(vocab_size)

X_train_project_grade_category_pre_seq = t3.texts_to_sequences(X_train['project_grade_cate
X_test_project_grade_category_pre_seq = t3.texts_to_sequences(X_test['project_grade_catego

max_length = 5
X_train_project_grade_category_pad_seq = pad_sequences(X_train_project_grade_category_pre_
X_test_project_grade_category_pad_seq = pad_sequences(X_test_project_grade_category_pre_se

embedding_size = min(vocab_size ,50)

project_grade_category_input = Input(shape=(5,), name="project_grade_category")
project_grade_category_feature = Embedding(input_dim=vocab_size +1,output_dim=embedding_si
project_grade_category_feature = Flatten()(project_grade_category_feature)
```

```
10
```

```
X_train_project_grade_category_pad_seq
```

```
array([[0, 0, 1, 6, 7],
       [0, 0, 1, 2, 3],
       [0, 0, 1, 4, 5],
       ...,
       [0, 0, 1, 4, 5],
       [0, 0, 1, 4, 5],
       [0, 0, 1, 4, 5]], dtype=int32)
```

project_subject_categories

```
# Apply tokenizer
t4 = Tokenizer()
t4.fit_on_texts(X_train['clean_categories'].values)
vocab_size = len(t4.word_index) + 1

print(vocab_size)

X_train_clean_categories_pre_seq = t4.texts_to_sequences(X_train['clean_categories'])
X_test_clean_categories_pre_seq = t4.texts_to_sequences(X_test['clean_categories'])

max_length = 5
X_train_clean_categories_pad_seq = pad_sequences(X_train_clean_categories_pre_seq,maxlen=m
X_test_clean_categories_pad_seq = pad_sequences(X_test_clean_categories_pre_seq,maxlen=max

embedding_size = min(vocab_size ,50)

project_subject_categories_input = Input(shape=(5,), name="clean_categories")
project_subject_categories_feature = Embedding(input_dim=vocab_size +1,output_dim=embeddin
project_subject_categories_feature = Flatten()(project_subject_categories_feature)
```

```
16
```

```
X_train_clean_categories_pad_seq[9]
```

```
array([0, 0, 0, 1, 2], dtype=int32)
```

project_subject_subcategories

```
# Apply tokenizer
t5 = Tokenizer()
t5.fit_on_texts(X_train['clean_subcategories'].values)
vocab_size = len(t5.word_index) + 1

print(vocab_size)

X_train_clean_subcategories_pre_seq = t5.texts_to_sequences(X_train['clean_subcategories']
X_test_clean_subcategories_pre_seq = t5.texts_to_sequences(X_test['clean_subcategories'])

max_length = 5
X_train_clean_subcategories_pad_seq = pad_sequences(X_train_clean_subcategories_pre_seq,ma
X_test_clean_subcategories_pad_seq = pad_sequences(X_test_clean_subcategories_pre_seq,maxl

embedding_size = min(vocab_size ,50)

project_subject_subcategories_input = Input(shape=(5,), name="clean_subcategories")
project_subject_subcategories_feature = Embedding(input_dim=vocab_size +1,output_dim=embed
project_subject_subcategories_feature = Flatten()(project_subject_subcategories_feature)
```

```
38
```

```
X_train_clean_subcategories_pad_seq
```

```
array([[0, 0, 1, 3, 4],
       [0, 0, 0, 0, 1],
       [0, 0, 1, 3, 4],
       ...,
       [0, 0, 1, 3, 4],
       [0, 0, 0, 0, 2],
       [0, 0, 3, 4, 2]], dtype=int32)
```

teacher_prefix

```
# Apply tokenizer
t6 = Tokenizer()
t6.fit_on_texts(X_train['teacher_prefix'].values)

vocab_size = len(t6.word_index) + 1
```

```
print(vocab_size)

X_train_teacher_prefix_pre_seq = t6.texts_to_sequences(X_train['teacher_prefix'])
X_test_teacher_prefix_pre_seq = t6.texts_to_sequences(X_test['teacher_prefix'])

max_length = 1
X_train_teacher_prefix_pad_seq = pad_sequences(X_train_teacher_prefix_pre_seq,maxlen=max_l
X_test_teacher_prefix_pad_seq = pad_sequences(X_test_teacher_prefix_pre_seq,maxlen=max_len

embedding_size = min(vocab_size ,50)

teacher_prefix_input = Input(shape=(1,), name="teacher_prefix")
teacher_prefix_feature = Embedding(input_dim=vocab_size +1,output_dim=embedding_size)(teac
teacher_prefix_feature = Flatten()(teacher_prefix_feature)
```

```
6
```

```
X_train_teacher_prefix_pad_seq.shape
```

```
(80000, 1)
```

Numerical feature -teacher_number_of_previously_posted_projects and price

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer

teacher_number_scalar = StandardScaler()
teacher_number_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.r
X_train_teacher_number_of_previously_posted_project = teacher_number_scalar.transform(X_tr
X_test_teacher_number_of_previously_posted_project = teacher_number_scalar.transform(X_tes


normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

# to cancatenate numeric feature reshaping array
X_train_price_norm =X_train_price_norm.reshape(-1,1)
X_test_price_norm =X_test_price_norm.reshape(-1,1)

X_train_num_teacher_number_of_previously_posted_projects_and_price = np.hstack((X_train_te
X_test_num_teacher_number_of_previously_posted_projects_and_price = np.hstack((X_test_teac

numerical_input = Input(shape=(2,))
numeric_dense = Dense(10)(numerical_input )
```

Merge All layers

```
from keras.layers.merge import concatenate
from keras.layers import Dropout

# Add all embeddings
merge = concatenate([essay_feature,school_state_feature,project_grade_category_feature,pro
                     teacher_prefix_feature,numeric_dense])
dense1 = Dense(100, activation='relu', kernel_initializer='he_normal')(merge)
drop_out_layer1 = Dropout(0.6)(dense1)
dense2 = Dense(100, activation='relu', kernel_initializer='he_normal')(drop_out_layer1)
drop_out_layer2 = Dropout(0.6)(dense2)
dense3 = Dense(50, activation='relu', kernel_initializer='he_normal')(drop_out_layer2)
```

```
output =Dense(2, activation='softmax')(dense3)
model = Model(inputs=[essay_input,school_state_input,project_grade_category_input,project_
                      project_subject_subcategories_input,teacher_prefix_input,numerical_i
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001), loss='categorical_crossentrop
model.summary()
```

```
clean_categories (InputLayer)   [(None, 5)]         0
_________________________________________________________________
clean_subcategories (InputLayer [(None, 5)]         0
_________________________________________________________________
teacher_prefix (InputLayer)     [(None, 1)]         0
_________________________________________________________________
lstm (LSTM)                     (None, 100)         160400      embedding[0][0]
_________________________________________________________________
embedding_1 (Embedding)         (None, 1, 50)       2650        school_state[0][0
_________________________________________________________________
embedding_2 (Embedding)         (None, 5, 10)       110         project_grade_cat
_________________________________________________________________
embedding_3 (Embedding)         (None, 5, 16)       272         clean_categories[
_________________________________________________________________
embedding_4 (Embedding)         (None, 5, 38)       1482        clean_subcategori
_________________________________________________________________
embedding_5 (Embedding)         (None, 1, 6)        42          teacher_prefix[0]
_________________________________________________________________
input_2 (InputLayer)            [(None, 2)]         0
_________________________________________________________________
flatten (Flatten)               (None, 100)         0           lstm[0][0]
_________________________________________________________________
flatten_1 (Flatten)             (None, 50)          0           embedding_1[0][0]
_________________________________________________________________
flatten_2 (Flatten)             (None, 50)          0           embedding_2[0][0]
_________________________________________________________________
flatten_3 (Flatten)             (None, 80)          0           embedding_3[0][0]
_________________________________________________________________
flatten_4 (Flatten)             (None, 190)         0           embedding_4[0][0]
_________________________________________________________________
flatten_5 (Flatten)             (None, 6)           0           embedding_5[0][0]
```

```
dense (Dense)                   (None, 10)           30          input_2[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 486)          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, 100)          48700       concatenate[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 100)          0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 100)          10100       dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 100)          0           dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 50)           5050        dropout_1[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 2)            102         dense_3[0][0]
==================================================================================================
Total params: 15,204,338
Trainable params: 228,938
Non-trainable params: 14,975,400
```

```
from tensorflow.keras.utils import plot_model
plot_model(model, "multi_input_and_output_model.png", show_shapes=True)
```


```
%load_ext tensorboard

# Clear any logs from previous runs
!rm -rf ./logs/

import tensorflow as tf
import datetime

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, wr
```

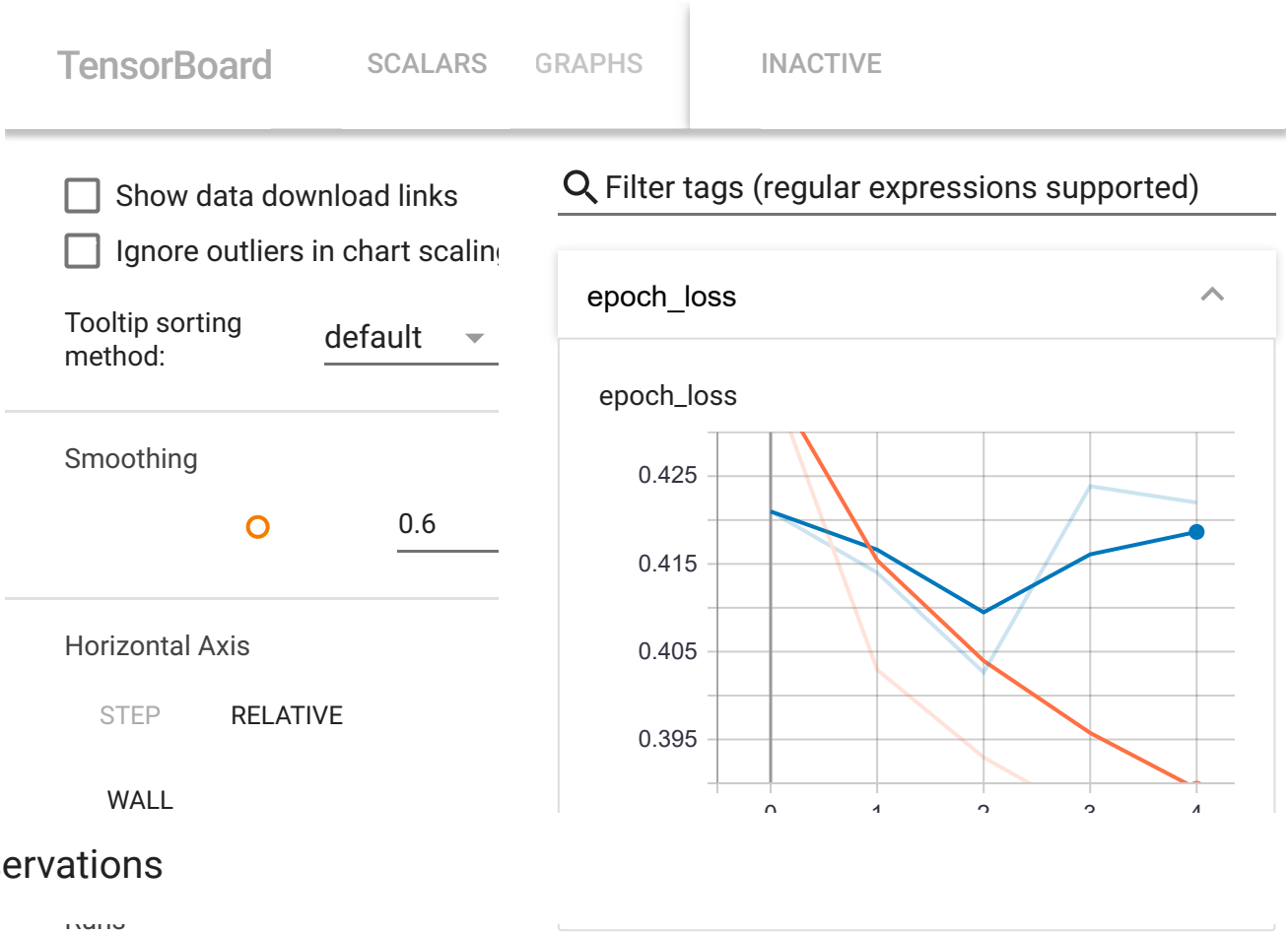
```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
```

```
y_tr_ohe =pd.get_dummies(y_train)
y_test_ohe = pd.get_dummies(y_test)


import keras
import numpy as np
import sklearn.metrics as sklm


class Metrics(keras.callbacks.Callback):

    def __init__(self,training_data,validation_data):
        self.x = training_data[0]
        self.y = training_data[1]
        self.x_val = validation_data[0]
        self.y_val = validation_data[1]

    def on_train_begin(self, logs={}):
        self.auc_tr = []
        self.auc_te = []

    def on_epoch_end(self, epoch, logs={}):
        score_train = np.asarray(self.model.predict(self.x))
        score_test = np.asarray(self.model.predict(self.x_val))
        #predict = np.round(np.asarray(self.model.predict(self.validation_data[0])))
        #targ_train = validation_data[1]
        auc_train =sklm.roc_auc_score(self.y, score_train)
        auc_test =sklm.roc_auc_score(self.y_val, score_test)
        self.auc_tr.append(auc_train)
        self.auc_te.append(auc_test)
        #print("  auc: "+"{:.4f}".format(auc));
        print(' roc-auc_train: %s - roc-auc_test: %s' % (str(round(auc_train,4)),str(round
        return



validation_data=([X_test_essay_pad_seq,X_test_school_state_pad_seq,X_test_project_grade_ca
```

```
                              X_test_clean_subcategories_pad_seq,X_test_teacher_prefix_pad_s
training_data =([X_train_essay_pad_seq,X_train_school_state_pad_seq,X_train_project_grade_
           X_train_teacher_prefix_pad_seq,X_train_num_teacher_number_of_previously_posted_

metrics = Metrics(training_data,validation_data)

model.fit([X_train_essay_pad_seq,X_train_school_state_pad_seq,X_train_project_grade_catego
           X_train_teacher_prefix_pad_seq,X_train_num_teacher_number_of_previously_posted_
          validation_data=([X_test_essay_pad_seq,X_test_school_state_pad_seq,X_test_projec
                              X_test_clean_subcategories_pad_seq,X_test_teacher_prefix_pad_s
          callbacks=[tensorboard_callback, metrics])
```

```
Epoch 1/5
   1/2500 [..............................] - ETA: 0s - loss: 0.6791WARNING:tensorflow
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
2500/2500 [==============================] - ETA: 0s - loss: 0.4362 roc-auc_train: 0
2500/2500 [==============================] - 2561s 1s/step - loss: 0.4362 - val_loss
Epoch 2/5
2500/2500 [==============================] - ETA: 0s - loss: 0.4030 roc-auc_train: 0
2500/2500 [==============================] - 2517s 1s/step - loss: 0.4030 - val_loss
Epoch 3/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3930 roc-auc_train: 0
2500/2500 [==============================] - 2556s 1s/step - loss: 0.3930 - val_loss
Epoch 4/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3861 roc-auc_train: 0
2500/2500 [==============================] - 2580s 1s/step - loss: 0.3861 - val_loss
Epoch 5/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3811 roc-auc_train: 0
2500/2500 [==============================] - 2570s 1s/step - loss: 0.3811 - val_loss
<tensorflow.python.keras.callbacks.History at 0x7f40241cd1d0>
```

```
tensorboard --logdir logs/fit
```



Observations

1. As every epoch number increases, loss is decreasing and AUC score is increasing

2. From epoch number 4 to 5 roc_auc_train is increased but roc_auc_test slightly decreased.

3. Highest AUC_train is: 0.7526 and AUC_test is: 0.7334

Model-2

```
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt


vectorizer = TfidfVectorizer()

# encoding eassay
train_tf=vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
idf_scores = train_tf.idf_

# generate boxplot on idf_ values

#plt.boxplot([vectorizer.idf_])
```

```
#q75, q25 = np.percentile(idf_scores, [75 ,25])
#print(q75, q25)

#listofNum = list(filter(lambda x : x >= q25 and x <= q75, vectorizer.idf_))

filtered_indices = np.argwhere(((idf_scores> 2) & (idf_scores < 11) ))
filtered_indices = [idx[0] for idx in filtered_indices]

vocabulary = train_tf.get_feature_names()

#preparing a set with filtered vocabulary
filtered_voc = {vocabulary[i] for i in filtered_indices}
#removing the words (which are not in filtered voc) from essays

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



# prepare tokenizer
t_1 = Tokenizer()
t_1.fit_on_texts(filtered_text_list)
vocab_size = len(t_1.word_index) + 1

print(vocab_size)

X_train_essay_pre_seq2 = t_1.texts_to_sequences(X_train['essay'].values)
X_test_essay_pre_seq2 = t_1.texts_to_sequences(X_test['essay'].values)

# padd sequence

max_length = 600
X_train_essay_pad_seq2 = pad_sequences(X_train_essay_pre_seq2,maxlen=max_length)
X_test_essay_pad_seq2 = pad_sequences(X_test_essay_pre_seq2,maxlen=max_length)

embeddings_index = dict()

f = open('glove.42B.300d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

# create a weight matrix for words in training docs
embedding_matrix = zeros((vocab_size, 300))
for word, i in t_1.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
# define feature

essay_input = Input(shape=(600,), name="essay")
essay_feature = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=4, tra
#essay_feature = Embedding(input_dim=vocab_size + 1, output_dim= 300, weights=[embedding_m
essay_feature = LSTM(100)(essay_feature)
essay_feature = Flatten()(essay_feature)
```

```
24741
```

```
y_tr_ohe =pd.get_dummies(y_train)
y_test_ohe = pd.get_dummies(y_test)


validation_data=([X_test_essay_pad_seq2,X_test_school_state_pad_seq,X_test_project_grade_c
                            X_test_clean_subcategories_pad_seq,X_test_teacher_prefix_pad_s


training_data =([X_train_essay_pad_seq2,X_train_school_state_pad_seq,X_train_project_grade
           X_train_teacher_prefix_pad_seq,X_train_num_teacher_number_of_previously_posted_

metrics = Metrics(training_data,validation_data)



from keras.layers.merge import concatenate
from keras.layers import Dropout

# concat all embeddings
merge = concatenate([essay_feature,school_state_feature,project_grade_category_feature,pro
                     teacher_prefix_feature,numeric_dense])
dense1 = Dense(100, activation='relu', kernel_initializer='he_normal')(merge)
drop_out_layer1 = Dropout(0.5)(dense1)
dense2 = Dense(100, activation='relu', kernel_initializer='he_normal')(drop_out_layer1)
drop_out_layer2 = Dropout(0.5)(dense2)
dense3 = Dense(50, activation='relu', kernel_initializer='he_normal')(drop_out_layer2)

output =Dense(2, activation='softmax')(dense3)
model = Model(inputs=[essay_input,school_state_input,project_grade_category_input,project_
                      project_subject_subcategories_input,teacher_prefix_input,numerical_i
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001), loss='categorical_crossentrop
model.summary()
```

```
__________________________________________________________________________________________
clean_categories (InputLayer)   [(None, 5)]          0
__________________________________________________________________________________________
clean_subcategories (InputLayer [(None, 5)]          0
__________________________________________________________________________________________
teacher_prefix (InputLayer)     [(None, 1)]          0
__________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 100)          160400      embedding_6[0][0]
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 50)        2650        school_state[0][0
__________________________________________________________________________________________
embedding_2 (Embedding)         (None, 5, 10)        110         project_grade_cat
```

```
embedding_2 (Embedding)         (None, 5, 10)        110         project_grade_cat[
__________________________________________________________________________________
embedding_3 (Embedding)         (None, 5, 16)        272         clean_categories[0
__________________________________________________________________________________
embedding_4 (Embedding)         (None, 5, 38)        1482        clean_subcategori
__________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 6)         42          teacher_prefix[0]
__________________________________________________________________________________
input_2 (InputLayer)            [(None, 2)]          0
__________________________________________________________________________________
flatten_6 (Flatten)             (None, 100)          0           lstm_1[0][0]
__________________________________________________________________________________
flatten_1 (Flatten)             (None, 50)           0           embedding_1[0][0]
__________________________________________________________________________________
flatten_2 (Flatten)             (None, 50)           0           embedding_2[0][0]
__________________________________________________________________________________
flatten_3 (Flatten)             (None, 80)           0           embedding_3[0][0]
__________________________________________________________________________________
flatten_4 (Flatten)             (None, 190)          0           embedding_4[0][0]
__________________________________________________________________________________
flatten_5 (Flatten)             (None, 6)            0           embedding_5[0][0]
__________________________________________________________________________________
dense (Dense)                   (None, 10)           30          input_2[0][0]
__________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 486)          0           flatten_6[0][0]
                                                                 flatten_1[0][0]
                                                                 flatten_2[0][0]
                                                                 flatten_3[0][0]
                                                                 flatten_4[0][0]
                                                                 flatten_5[0][0]
                                                                 dense[0][0]
__________________________________________________________________________________
dense_5 (Dense)                 (None, 100)          48700       concatenate_1[0][0
__________________________________________________________________________________
dropout_2 (Dropout)             (None, 100)          0           dense_5[0][0]
__________________________________________________________________________________
dense_6 (Dense)                 (None, 100)          10100       dropout_2[0][0]
__________________________________________________________________________________
dropout_3 (Dropout)             (None, 100)          0           dense_6[0][0]
__________________________________________________________________________________
dense_7 (Dense)                 (None, 50)           5050        dropout_3[0][0]
__________________________________________________________________________________
dense_8 (Dense)                 (None, 2)            102         dense_7[0][0]
==================================================================================
Total params: 7,651,238
Trainable params: 228,938
Non-trainable params: 7,422,300
```

```
%load_ext tensorboard

# Clear logs from previous runs:
!rm -rf ./logs/

import tensorflow as tf
import datetime

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, wr
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
```

```
model.fit([X_train_essay_pad_seq2,X_train_school_state_pad_seq,X_train_project_grade_categ
           X_train_teacher_prefix_pad_seq,X_train_num_teacher_number_of_previously_posted_
          validation_data=([X_test_essay_pad_seq2,X_test_school_state_pad_seq,X_test_proje
                            X_test_clean_subcategories_pad_seq,X_test_teacher_prefix_pad_s
          callbacks=[tensorboard_callback, metrics])
```

```
Epoch 1/5
2500/2500 [==============================] - ETA: 0s - loss: 0.4305 roc-auc_train: 0
2500/2500 [==============================] - 2534s 1s/step - loss: 0.4305 - val_loss
Epoch 2/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3964 roc-auc_train: 0
2500/2500 [==============================] - 2536s 1s/step - loss: 0.3964 - val_loss
Epoch 3/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3849 roc-auc_train: 0
2500/2500 [==============================] - 2571s 1s/step - loss: 0.3849 - val_loss
Epoch 4/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3771 roc-auc_train: 0
2500/2500 [==============================] - 2616s 1s/step - loss: 0.3771 - val_loss
Epoch 5/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3728 roc-auc_train: 0
2500/2500 [==============================] - 2629s 1s/step - loss: 0.3728 - val_loss
<tensorflow.python.keras.callbacks.History at 0x7f401d4bf898>
```

```
tensorboard --logdir logs/fit
```

Observations

1. As every epoch number increases, loss decreases.
2. roc_auc_train and roc_auc_test is slightly better than model-1
3. Highest auc_train: 0.7661 and auc_test: 0.739

Model-3

School state

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizer = CountVectorizer()
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_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
(80000, 51) (80000,)
(20000, 51) (20000,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id',
====================================================================================
```

encoding categorical features: project_grade_category

```
vectorizer = CountVectorizer()
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_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_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(80000, 4) (80000,)
(20000, 4) (20000,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================
```

encoding categorical features: clean_categories

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)
```

```
print("After vectorizations")
print(X_train_clean_categories_ohe.shape, y_train.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(80000, 9) (80000,)
(20000, 9) (20000,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_langu
====================================================================================
```

encoding categorical features: clean_subcategories

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train da

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].valu
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values

print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(80000, 30) (80000,)
(20000, 30) (20000,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'colleg
====================================================================================
```

encoding categorical features: teacher_prefix

```
vectorizer = CountVectorizer()
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)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

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
(80000, 5) (80000,)
(20000, 5) (20000,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==================================================================================
```

Numeric field - price and teacher_number_of_previously_posted_projects

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer

teacher_number_scalar = StandardScaler()
teacher_number_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.r
X_train_teacher_number_of_previously_posted_project = teacher_number_scalar.transform(X_tr
X_test_teacher_number_of_previously_posted_project = teacher_number_scalar.transform(X_tes


normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

# to cancatenate numeric feature reshaping array
X_train_price_norm =X_train_price_norm.reshape(-1,1)
X_test_price_norm =X_test_price_norm.reshape(-1,1)
```

Concatenate all except essay feature

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,X_train_clean_cat
X_te = hstack((X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,X_test_clean_categor

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(80000, 101) (80000,)
(20000, 101) (20000,)
==================================================================================
```

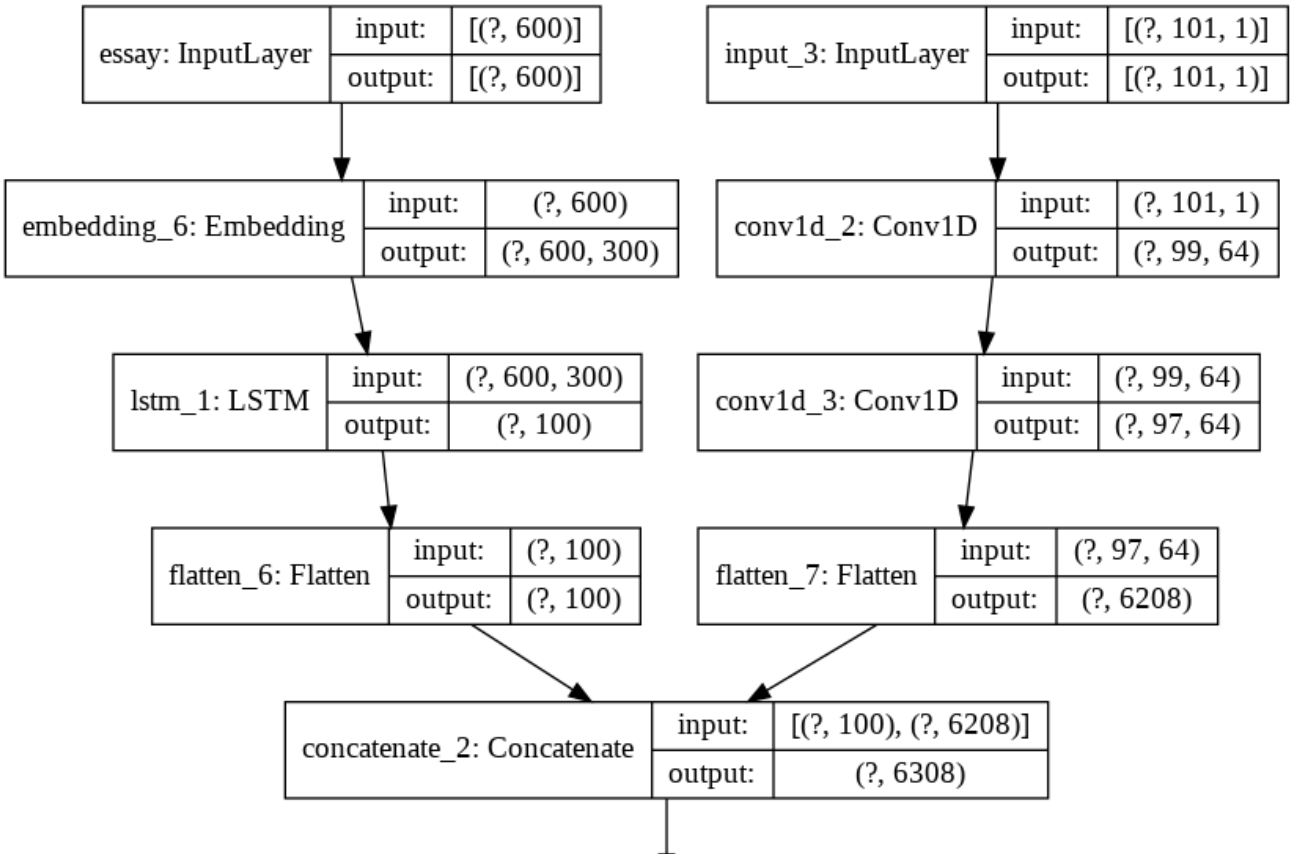
```
import tensorflow as tf
from tensorflow import keras
```

```
from tensorflow.keras import layers


input_cat_data = keras.Input(shape=(101,1))

cov1 = layers.Conv1D(64,3,activation='relu')(input_cat_data)
cov2 = layers.Conv1D(64,3,activation='relu')(cov1)
flatten2 = Flatten()(cov2)



from keras.layers.merge import concatenate
from keras.layers import Dropout

# concat all embeddings
merge = concatenate([essay_feature,flatten2])
dense1 = Dense(100, activation='relu', kernel_initializer='he_normal')(merge)
drop_out_layer1 = Dropout(0.6)(dense1)
dense2 = Dense(100, activation='relu', kernel_initializer='he_normal')(drop_out_layer1)
drop_out_layer2 = Dropout(0.6)(dense2)
dense3 = Dense(50, activation='relu', kernel_initializer='he_normal')(drop_out_layer2)

output =Dense(2, activation='softmax')(dense3)
model = Model(inputs=[essay_input,input_cat_data], outputs=output)
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001), loss='categorical_crossentrop
model.summary()
```

```
Model: "functional_7"
__________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==========================================================================================
essay (InputLayer)           [(None, 600)]             0
__________________________________________________________________________________________
input_3 (InputLayer)         [(None, 101, 1)]          0
__________________________________________________________________________________________
embedding_7 (Embedding)      (None, 600, 300)          14975400    essay[0][0]
__________________________________________________________________________________________
conv1d_2 (Conv1D)            (None, 99, 64)            256         input_3[0][0]
__________________________________________________________________________________________
lstm_2 (LSTM)                (None, 100)               160400      embedding_7[0][0]
__________________________________________________________________________________________
conv1d_3 (Conv1D)            (None, 97, 64)            12352       conv1d_2[0][0]
__________________________________________________________________________________________
flatten_8 (Flatten)          (None, 100)               0           lstm_2[0][0]
__________________________________________________________________________________________
flatten_7 (Flatten)          (None, 6208)              0           conv1d_3[0][0]
__________________________________________________________________________________________
concatenate_3 (Concatenate)  (None, 6308)              0           flatten_8[0][0]
                                                                   flatten_7[0][0]
__________________________________________________________________________________________
dense_13 (Dense)             (None, 100)               630900      concatenate_3[0][0]
__________________________________________________________________________________________
dropout_6 (Dropout)          (None, 100)               0           dense_13[0][0]
__________________________________________________________________________________________
dense_14 (Dense)             (None, 100)               10100       dropout_6[0][0]
__________________________________________________________________________________________
```

```
dropout_7 (Dropout)             (None, 100)          0           dense_14[0][0]
__________________________________________________________________________________________________
dense_15 (Dense)                (None, 50)           5050        dropout_7[0][0]
__________________________________________________________________________________________________
dense_16 (Dense)                (None, 2)            102         dense_15[0][0]
==================================================================================================
Total params: 15,794,560
Trainable params: 819,160
Non-trainable params: 14,975,400
__________________________________________________________________________________________________
```

```
from tensorflow.keras.utils import plot_model
plot_model(model, "multi_input_and_output_model.png", show_shapes=True)
```




```
y_tr_ohe =pd.get_dummies(y_train)
y_test_ohe = pd.get_dummies(y_test)
```

```
validation_data=([X_test_essay_pad_seq,X_te],y_test_ohe)
training_data =([X_train_essay_pad_seq,X_tr],y_tr_ohe)

metrics = Metrics(training_data,validation_data)
```

```
%load_ext tensorboard

# Clear any logs from previous runs
!rm -rf ./logs/

import tensorflow as tf
import datetime

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, wr
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
```

```
|                                  | output: | (?, 2)  |
```

```
model.fit([X_train_essay_pad_seq,X_tr],y_tr_ohe,epochs=5,
          validation_data=([X_test_essay_pad_seq,X_te],y_test_ohe),
          callbacks=[tensorboard_callback, metrics])
```

```
Epoch 1/5
2500/2500 [==============================] - ETA: 0s - loss: 0.4281 roc-auc_train: 0
```

```
2500/2500 [==============================] - 2645s 1s/step - loss: 0.4281 - val_loss
Epoch 2/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3953 roc-auc_train: 0
2500/2500 [==============================] - 2631s 1s/step - loss: 0.3953 - val_loss
Epoch 3/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3861 roc-auc_train: 0
2500/2500 [==============================] - 2611s 1s/step - loss: 0.3861 - val_loss
Epoch 4/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3804 roc-auc_train: 0
2500/2500 [==============================] - 2611s 1s/step - loss: 0.3804 - val_loss
Epoch 5/5
2500/2500 [==============================] - ETA: 0s - loss: 0.3760 roc-auc_train: 0
2500/2500 [==============================] - 2692s 1s/step - loss: 0.3760 - val_loss
<tensorflow.python.keras.callbacks.History at 0x7f4027c06278>
```

```
tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 1132), started 7:48:53 ago. (Use '!kill 1132' t
```

TensorBoard SCALARS GRAPHS INACTIVE

Show data download links

Filter tags (regular expressions supported)

Observations

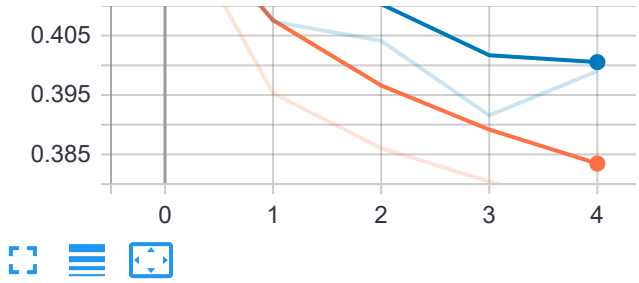
method:

1. AS every epoch number increases, loss is decreasing and AUC score is increasing
2. In Model 3, roc_auc_train & roc_auc_test is slightly better than model 1 and model 2
3. Highest value of Test_auc score is 74.54% .

Horizontal Axis

STEP RELATIVE

WALL



Runs

Write a regex to filter runs

20200909-005430/train

20200909-005430/validation

TOGGLE ALL RUNS

logs/fit

✓ 31m 2s completed at 1:06 AM