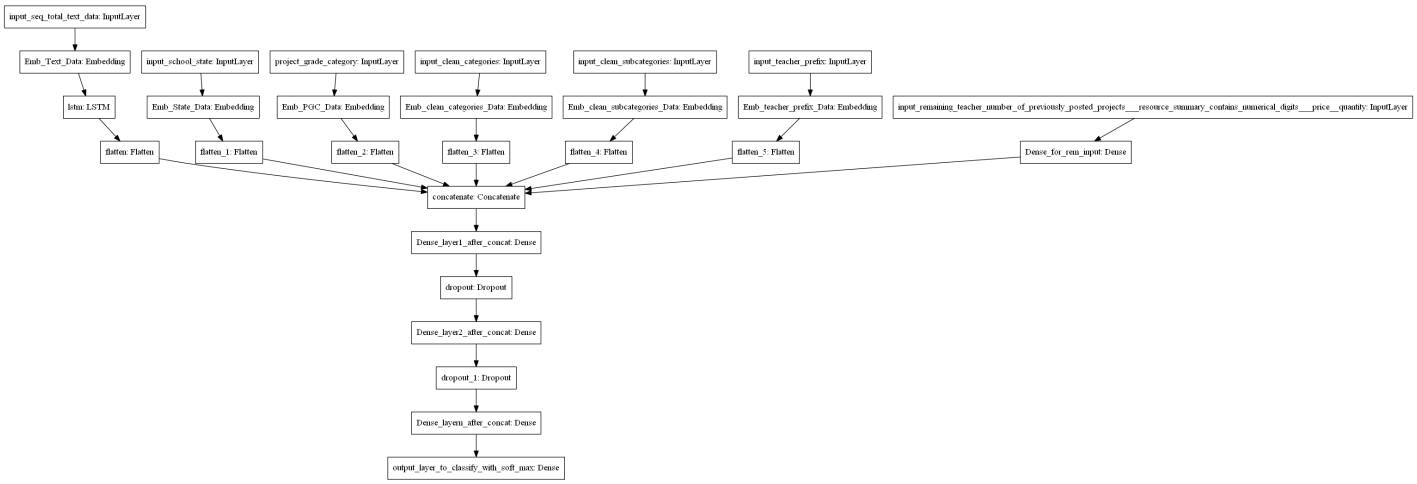
Assignment : Lstm On Donor Choose



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.

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.

```
import numpy as np
import pandas as pd
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Input , Dropout
from keras.layers import Flatten
from keras.layers import concatenate
from keras.layers.embeddings import Embedding
from keras.models import Model
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
import matplotlib.pyplot as plt
import pickle
from keras.layers import LSTM
from keras.preprocessing.text import text_to_word_sequence
import tensorflow as tf
from keras.callbacks import ModelCheckpoint,TensorBoard,ReduceLROnPlateau, EarlyStopping
from keras.layers import LayerNormalization
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns
from keras.regularizers import l2
from sklearn.metrics import roc_auc_score
from keras.models import load_model
from IPython.display import Image
from scipy.sparse import hstack
from keras.layers import Conv1D
from sklearn.feature_extraction.text import CountVectorizer
from prettytable import PrettyTable
from keras.preprocessing import sequence
from numpy import zeros
from keras import regularizers
from keras.layers import Conv1D
from keras.initializers import he_normal
from keras import optimizers
```

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

```
Mounted at /content/drive
```

```
project_data  = pd.read_csv("/content/drive/MyDrive/Lstm on donor choose/preprocessed_data.cs
project_data.head(2)
```

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

```
project_data.shape
```

```
(109248, 9)
```

```
x = project_data.drop(['project_is_approved'], axis = 1)
y = project_data['project_is_approved']
```

```
print(y.shape)
print(x.shape)
```

```
(109248,)
(109248, 8)
```

splitting data

```
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)
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size=0.2, stratify=y_t

print('Train Data Set', x_train.shape, y_train.shape)
print('Cross Validate Data Set', x_cv.shape, y_cv.shape)
print('Test Data Set', x_test.shape, y_test.shape)
```

```
Train Data Set (69918, 8) (69918,)
Cross Validate Data Set (17480, 8) (17480,)
Test Data Set (21850, 8) (21850,)
```

Chaning type of dependent variable (y) to categorical type

```
from keras.utils import np_utils

#np_utils.to_categorical is used to convert array of labeled data(from 0 to nb_classes-1) to

y_train = np_utils.to_categorical(y_train, 2)
y_test = np_utils.to_categorical(y_test, 2)
y_cv = np_utils.to_categorical(y_cv, 2)
```

```
y_cv.shape
```

```
(17480, 2)
```

```
print("Shape of x_train:", x_train.shape)
print("Shape of x_cv:", x_cv.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_cv:", y_cv.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of x_train: (69918, 8)
Shape of x_cv: (17480, 8)
Shape of x_test: (21850, 8)
Shape of y_train: (69918, 2)
Shape of y_cv: (17480, 2)
Shape of y_test: (21850, 2)
```

```
y_train.shape
```

```
(69918, 2)
```

Encoding Text Data

padding:

Padding document is to have the same input length of each document.

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
def encoding_padding(input_text, max_length):
    encoded_data = tokenizer.texts_to_sequences(input_text)
    padded_data = pad_sequences(encoded_data, maxlen=max_length, padding='post')
    return padded_data
```

Tokenize:

Input data to layer should be integer. So, using tokenize inbuilt function, we will integer encode the text data

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
train_text = x_train["essay"].values.tolist()
test_text = x_test['essay'].values.tolist()
val_text = x_cv['essay'].values.tolist()

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_text)
vocab_size = len(tokenizer.word_index) + 1

max_length = 400
train_text_encode = encoding_padding(train_text,  max_length=max_length)
test_text_encode = encoding_padding(test_text, max_length=max_length)
val_text_encode = encoding_padding(val_text, max_length=max_length)

print(train_text_encode.shape)
print(test_text_encode.shape)
print(val_text_encode.shape)
```

```
(69918, 400)
(21850, 400)
(17480, 400)
```

```
vocab_size
```

```
47363
```

Weight Matrix

```
glove_vector = open("/content/drive/MyDrive/Lstm on donor choose/glove_vectors","rb")
glove_words = pickle.load(glove_vector)


#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

print(embedding_matrix.shape)
embedding_matrix[10]
```

```
            2.3450e-01, -3.7587e+00, -4.6558e-01,  8.9927e-03,  3.1239e-02,
            4.8702e-02, -1.0355e-01, -4.6802e-02, -4.9096e-01,  2.8895e-01,
```

```
 4.1719e-03, -2.9244e-02, -1.5100e-01, -5.9874e-02, -1.0642e-01,
 1.0236e-02,  8.6611e-02, -1.2328e-01,  5.8111e-02,  4.4774e-01,
 1.2824e-01,  1.0826e-01,  4.0072e-02, -2.9260e-01, -2.2230e-01,
-5.2156e-01, -3.3644e-01,  2.5296e-01,  3.5089e-01, -1.1302e-01,
-3.6476e-01, -2.6759e-01, -2.5142e-01,  9.1741e-02,  4.1009e-01,
-1.1340e-01,  1.0475e-01, -5.5954e-01, -2.3784e-01,  4.0832e-01,
-1.8500e-03,  7.7391e-02,  1.1396e-01,  3.6684e-01, -7.2937e-01,
 1.7876e-01, -4.1155e-01, -1.0766e-01, -3.1598e-01, -3.3749e-02,
-1.7046e-01,  7.6821e-02,  5.3448e-02, -4.4717e-01, -1.8609e-01,
 3.1345e-02, -4.2470e-01,  2.8402e-01, -4.0830e-01, -4.1521e-01,
 2.2276e-02,  4.6234e-01, -1.1246e-01,  4.1678e-02, -3.0998e-01,
-3.3800e-01,  3.8019e-01, -2.3257e-01, -3.1180e-01, -4.6492e-01,
-7.7942e-02, -1.6737e-01,  7.2783e-02,  3.3261e-01, -1.4799e-01,
 9.2606e-02,  1.1133e-01,  1.4746e-01,  4.2155e-01, -9.6644e-02,
 8.7671e-02,  4.4040e-01, -4.7194e-01,  1.8347e-01, -1.8758e-01,
-1.0297e-01,  2.2551e-01, -3.4930e-01,  9.1168e-02, -3.3402e-01,
-4.4076e-01, -2.5139e+00,  4.8468e-01,  5.4613e-01,  1.3462e-01,
-2.4983e-01,  1.7436e-01,  2.6503e-01, -2.8247e-01,  1.4478e-01,
-3.0763e-01,  9.2538e-02,  8.0730e-02, -7.1329e-04, -5.1817e-02,
 1.0628e-01, -1.3635e-01, -2.8566e-01,  7.6668e-01, -5.3961e-01,
-6.0371e-02, -4.1717e-01, -1.5031e-01, -1.5060e-01, -5.0936e-02,
 1.3729e-01,  2.5398e-01,  2.8120e-01, -2.7441e-02, -2.2246e-01,
-1.8652e-01, -2.3802e-01, -1.4315e-01,  1.0879e-01,  4.1365e-02,
-1.5318e-01,  1.3241e-01,  4.5773e-02,  1.2711e-01,  5.6549e-01,
-6.1564e-01,  5.2479e-01,  1.6012e-01, -2.4330e-01,  9.1544e-01,
 9.6795e-02,  1.8513e-01,  8.1942e-02, -7.6229e-02, -4.7359e-01,
-3.7924e-01,  2.4905e-02, -2.3480e-01, -3.4498e-02, -1.8605e-01,
-9.1673e-02,  3.9880e-02, -1.7821e-02, -3.0448e-01,  6.2911e-02,
 1.5838e-01, -8.6701e-02,  1.1552e-01, -5.7740e-02, -7.9095e-02,
 3.5894e-01,  2.1118e-01, -9.0954e-02,  1.4708e-01,  1.0457e-01,
 4.1424e-01, -6.9558e-01,  4.0956e-01,  2.3965e-01,  8.3260e-02,
-2.3873e-01, -1.6275e-01,  5.2594e-02, -6.2785e-01, -5.8905e-02,
-8.3738e-02, -2.9720e-01,  3.5043e-01, -7.8798e-01, -8.3097e-03,
-2.2901e-01,  6.4798e-01, -1.8864e-01, -2.4055e-01, -1.2587e-01,
 3.2249e-01, -4.5415e-01,  2.1713e-01, -6.2449e-02, -1.8511e-01,
-3.2010e-01, -1.4711e-01,  8.7107e-02,  1.4972e-01,  2.4450e-02,
-1.0669e-01,  3.1007e-01, -3.8015e-01,  3.0550e-01,  2.4858e-01,
 2.5085e-01,  6.0111e-02, -2.3087e-01,  3.1406e-01, -3.1798e-01,
-6.4502e-02,  3.6925e-01, -1.5095e-01,  3.4986e-01,  1.7301e-01,
 2.2796e-02, -1.6879e-01, -8.5349e-03, -3.8463e-01,  3.8428e-01,
-2.2400e-01,  6.7523e-02,  1.9129e-01,  4.3048e-02, -9.4248e-02,
-4.0223e-01, -2.7964e-01, -8.0078e-01, -3.6238e-02, -1.7517e+00,
 2.7017e-02, -5.3902e-02,  4.0800e-01,  1.8626e-02,  4.2807e-01,
-8.9495e-02,  1.0047e-01, -6.9022e-03,  4.6068e-01, -8.2198e-02,
-2.0553e-01,  4.7258e-01, -2.1621e-01, -3.4478e-01,  2.7506e-01,
-1.0485e-01, -5.7402e-02, -4.3152e-01, -2.6496e-01,  7.6901e-02,
 1.4614e-01, -3.5529e-01, -2.4013e-02,  2.4993e-01,  2.4225e-01,
 5.9576e-03,  9.4487e-02,  1.7618e-01, -1.4687e-01,  8.6212e-01,
-2.0514e-01,  1.3622e-01,  1.8780e-01,  2.2014e-01, -2.9853e-01,
 2.2139e-01, -2.1277e-01,  1.2880e-02, -4.1641e-01, -1.9766e-01,
-3.0919e-01, -9.9688e-02,  1.6407e-01,  4.8313e-01,  5.1285e-01,
-2.5671e-01,  1.0048e-01, -2.8643e-01,  1.7951e-02, -2.8679e-01,
 1.9436e-02,  3.9268e-01,  1.6769e-01, -3.3534e-02, -4.6577e-01,
-3.7040e-01,  6.4633e-02, -3.6884e-01,  9.2190e-02, -8.9853e-02,
 3.4780e-01, -5.4120e-01, -1.7190e-01, -3.1266e-01,  6.8850e-02,

 4.3334e-02, -5.4285e-01,  2.7817e-01,  2.1425e-01, -3.6066e-01,
```

```
train_text_encode.shape[1]
```

```
400
```

```
embedding_matrix.shape[1]
```

```
300
```

1. Embedded Layer For Input essay

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
#creating input layer
input_essay = Input(shape=(train_text_encode.shape[1],), name='input_essay')
#creating embedding layer
# This embedding layer will encode the input sequence
# into a sequence of dense 300-dimensional vectors.
x1 =Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1], weights=[embedding_
#creating lstm layer
x1 = LSTM(128,kernel_regularizer=l2(0.001),return_sequences=True)(x1)

layer_1 = Flatten()(x1)
```

Categorical Feature:

1-Teacher prefix

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train["teacher_prefix"].values.tolist())

print('-'*50)
# This embedding layer will encode the input sequence

#embedding_layer = Embedding(input, output_dim, weights=[embedding_matrix],input_length=max_l
input_prefix = Input(shape=(1,),name="teacher_prefix")
x2 = Embedding(input_dim=52, output_dim=min(52//2,50), name="embed_prefix", trainable=True)(i
layer_2 = Flatten()(x2)

train_prefix_encode = encoding_padding(x_train["teacher_prefix"], max_length=1)
test_prefix_encode = encoding_padding(x_test["teacher_prefix"], max_length=1)
val_prefix_encode = encoding_padding(x_cv["teacher_prefix"], max_length=1)

print(train_prefix_encode.shape)
print(test_prefix_encode.shape)
print(val_prefix_encode.shape)
```

```
--------------------------------------------------
```

```
(69918, 1)
(21850, 1)
(17480, 1)
```

2-school state

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train["school_state"].values.tolist())

print('-'*50)
input_state = Input(shape=(1,),name="school_state")
x3 = Embedding(input_dim=52, output_dim=min(52//2,50), name="embed_state", trainable=True)(in
layer_3 = Flatten()(x3)

train_state_encode = encoding_padding(x_train["school_state"], max_length=1)
test_state_encode = encoding_padding(x_test["school_state"], max_length=1)
val_state_encode = encoding_padding(x_cv["school_state"], max_length=1)

print(train_state_encode.shape)
print(test_state_encode.shape)
print(val_state_encode.shape)
```

```
--------------------------------------------------
(69918, 1)
(21850, 1)
(17480, 1)
```

3- project garde category

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train["project_grade_category"].values.tolist())

print('-'*50)

input_grade = Input(shape=(1,),name="grade")
x4 = Embedding(input_dim=52, output_dim=min(52//2,50), name="embed_grade", trainable=True)(in
layer_4 = Flatten()(x4)

train_grade_encode = encoding_padding(x_train["project_grade_category"], max_length=1)
test_grade_encode = encoding_padding(x_test["project_grade_category"], max_length=1)
val_grade_encode = encoding_padding(x_cv["project_grade_category"], max_length=1)

print(train_grade_encode.shape)
print(test_grade_encode.shape)
print(val_grade_encode.shape)
```

```
--------------------------------------------------
(69918, 1)
```

```
(21850, 1)
(17480, 1)
```

4- clean categories

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train["clean_categories"].values.tolist())

print('-'*50)

input_cat= Input(shape=(1,),name="clean_categories")
x5 = Embedding(input_dim=52, output_dim=min(52//2,50), name="embed_cat", trainable=True)(inpu
layer_5 = Flatten()(x5)

train_cat_encode = encoding_padding(x_train["clean_categories"], max_length=1)
test_cat_encode = encoding_padding(x_test["clean_categories"], max_length=1)
val_cat_encode = encoding_padding(x_cv["clean_categories"], max_length=1)

print(train_cat_encode.shape)
print(test_cat_encode.shape)
print(val_cat_encode.shape)
```

```
--------------------------------------------------
(69918, 1)
(21850, 1)
(17480, 1)
```

5-clean sub categories

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train["clean_subcategories"].values.tolist())

print('-'*50)

input_subcat= Input(shape=(1,),name="subject_subcategories")
x6 = Embedding(input_dim=384 , output_dim=min(384//2,50), name="embed_subcat", trainable=True
layer_6 = Flatten()(x6)

train_subcat_encode = encoding_padding(x_train["clean_subcategories"], max_length=1)
test_subcat_encode = encoding_padding(x_test["clean_subcategories"], max_length=1)
val_subcat_encode = encoding_padding(x_cv["clean_subcategories"], max_length=1)

print(train_subcat_encode.shape)
print(test_subcat_encode.shape)
print(val_subcat_encode.shape)
```

```
--------------------------------------------------
(69918, 1)
(21850, 1)
(17480, 1)
```

Numerical Features

We will reshape the numerical features to (-1, 1). Then concatenate numerical features and standardize the final output.

- price
- teacher number of previously project

```
train_remaining_input = np.concatenate((x_train['price'].values.reshape(-1,1),
                                        x_train['teacher_number_of_previously_posted_projects'].val

test_remaining_input = np.concatenate((x_test['price'].values.reshape(-1,1),
                                       x_test['teacher_number_of_previously_posted_projects'].valu

val_remaining_input = np.concatenate((x_cv['price'].values.reshape(-1,1),
                                      x_cv['teacher_number_of_previously_posted_projects'].values

print(train_remaining_input.shape)
print(test_remaining_input.shape)
print(val_remaining_input.shape)
```

```
(69918, 2)
(21850, 2)
(17480, 2)
```

Standardizing Numerical Features

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

train_rem_input = scaler.fit_transform(train_remaining_input)
test_rem_input = scaler.transform(test_remaining_input)
val_rem_input = scaler.transform(val_remaining_input)

num_feats = Input(shape=(2,), name="numerical_features")
layer_7 = Dense(64,activation="relu",kernel_initializer="he_normal",kernel_regularizer=regula

print(train_rem_input.shape)
print(test_rem_input.shape)
print(val_rem_input.shape)
```

```
(69918, 2)
(21850, 2)
(17480, 2)
```

```
import numpy as np
import pandas as pd
```

```
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Input , Dropout
from keras.layers import Flatten
from keras.layers import concatenate
from keras.layers.embeddings import Embedding
from keras.models import Model
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
import matplotlib.pyplot as plt
import pickle
from keras.layers import LSTM
from keras.preprocessing.text import text_to_word_sequence
import tensorflow as tf
from keras.callbacks import ModelCheckpoint,TensorBoard,ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.layers import Input, Embedding, LSTM, Dropout, BatchNormalization
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns
from keras.regularizers import l2
from sklearn.metrics import roc_auc_score
from keras.models import load_model
from IPython.display import Image
from scipy.sparse import hstack
from keras.layers import Conv1D
from sklearn.feature_extraction.text import CountVectorizer
from prettytable import PrettyTable
from keras.preprocessing import sequence
from numpy import zeros
from keras import regularizers
from keras.layers import Conv1D
from keras.initializers import he_normal
from keras import optimizers
```

Concatenating all the flattened layers.

```
x_concat = concatenate([layer_1, layer_2, layer_3, layer_4, layer_5, layer_6, layer_7])
```

Keras Model-1.1

- Activation : { hidden_layer : relu , output_layer : softmax }
- optimizer : { 'Adam'}
- loss : { 'categorical_crossentropy' }
- Batch_size : {512}
- epoch : {10}

```
x = Dense(500,activation="relu", kernel_initializer="he_normal",kernel_regularizer=regularize
x = Dropout(0.5)(x)
x = Dense(250,activation="relu",kernel_initializer="glorot_normal",kernel_regularizer=regular
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(100,activation="relu", kernel_initializer="glorot_normal",kernel_regularizer=regula
x = Dropout(0.4)(x)
x = Dense(50,activation="relu", kernel_initializer="glorot_normal",kernel_regularizer=regular

output = Dense(2, activation='softmax', name='Output')(x)

model_1 = Model(inputs=[input_essay, input_state ,input_prefix,input_cat,
                        input_subcat ,input_grade ,num_feats ],outputs=[output])

print(model_1.summary())
```

```
Model: "model"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_essay (InputLayer)        [(None, 400)]        0
__________________________________________________________________________________________
embed_essay (Embedding)         (None, 400, 300)     14233500    input_essay[0][0]
__________________________________________________________________________________________
teacher_prefix (InputLayer)     [(None, 1)]          0
__________________________________________________________________________________________
school_state (InputLayer)       [(None, 1)]          0
__________________________________________________________________________________________
grade (InputLayer)              [(None, 1)]          0
__________________________________________________________________________________________
clean_categories (InputLayer)   [(None, 1)]          0
__________________________________________________________________________________________
subject_subcategories (InputLay [(None, 1)]          0
__________________________________________________________________________________________
lstm (LSTM)                     (None, 400, 128)     219648      embed_essay[0][0]
__________________________________________________________________________________________
embed_prefix (Embedding)        (None, 1, 26)        1352        teacher_prefix[0][0]
__________________________________________________________________________________________
embed_state (Embedding)         (None, 1, 26)        1352        school_state[0][0]
__________________________________________________________________________________________
embed_grade (Embedding)         (None, 1, 26)        1352        grade[0][0]
__________________________________________________________________________________________
embed_cat (Embedding)           (None, 1, 26)        1352        clean_categories[0][(
__________________________________________________________________________________________
embed_subcat (Embedding)        (None, 1, 50)        19200       subject_subcategorie
__________________________________________________________________________________________
numerical_features (InputLayer) [(None, 2)]          0
__________________________________________________________________________________________
flatten (Flatten)               (None, 51200)        0           lstm[0][0]
__________________________________________________________________________________________
flatten_1 (Flatten)             (None, 26)           0           embed_prefix[0][0]
__________________________________________________________________________________________
flatten_2 (Flatten)             (None, 26)           0           embed_state[0][0]
```

flatten_3 (Flatten)	(None, 26)	0	embed_grade[0][0]
flatten_4 (Flatten)	(None, 26)	0	embed_cat[0][0]
flatten_5 (Flatten)	(None, 50)	0	embed_subcat[0][0]
dense (Dense)	(None, 64)	192	numerical_features[0
concatenate (Concatenate)	(None, 51418)	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, 500)	25709500	concatenate[0][0]
dropout (Dropout)	(None, 500)	0	dense_1[0][0]

Getting all data into list

```
#Getting all data into list
#for train data
train_data = [train_text_encode, train_prefix_encode, train_state_encode, train_grade_encode,
              train_cat_encode, train_subcat_encode, train_rem_input]
#for test data
test_data = [test_text_encode, test_prefix_encode, test_state_encode, test_grade_encode,
             test_cat_encode, test_subcat_encode, test_rem_input]
#for val data
val_data = [val_text_encode, val_prefix_encode, val_state_encode, val_grade_encode,
            val_cat_encode, val_subcat_encode, val_rem_input]
```

```
y_train.shape
```

```
(69918, 2)
```

Network Artitecture

```
# https://github.com/mmortazavi/EntityEmbedding-Working_Example/blob/master/EntityEmbedding.i

import pydot_ng as pydot
from keras.utils.vis_utils import plot_model
from IPython.display import Image

plot_model(model_1, show_shapes = True, show_layer_names = True, to_file = 'model_1.png')

Image(retina = True, filename = 'model_1.png')
```

AUC-ROC custom function

```
from sklearn.metrics import roc_auc_score

import tensorflow as tf

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

Creating Callback with Checkpoint, EarlyStopping and Tensorboard

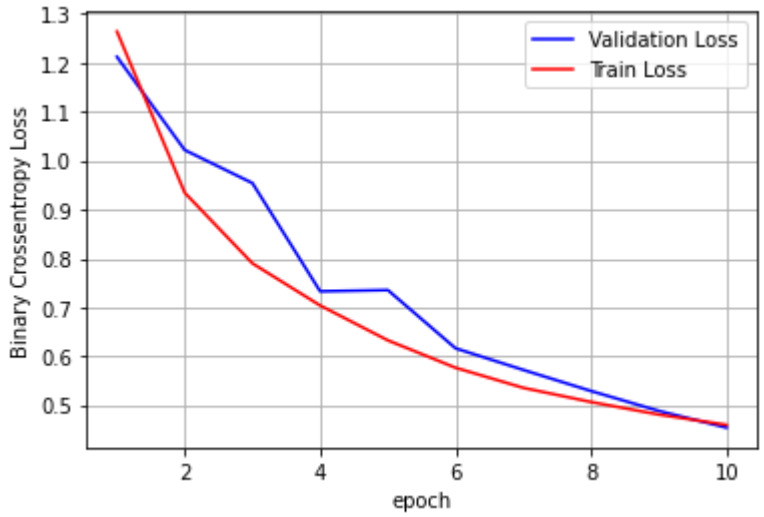
```
checkpoint1 = ModelCheckpoint("model_1.h5",
                              monitor="val_auroc",
                              mode="max",
                              save_best_only = True,
                              verbose=1)
earlystop1 = EarlyStopping(monitor = 'val_auroc',
                           mode="max",
                           min_delta = 0,
                           patience = 3,
                           verbose = 1)

tensorboard1 = TensorBoard(log_dir='Model1_visualization')

callbacks_1 = [checkpoint1,earlystop1,tensorboard1]


from tensorflow.keras.optimizers import Adam
model_1.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = [auroc])


history = model_1.fit(train_data, y_train, batch_size=512,epochs = 10, verbose=1,callbacks=ca
```

```
Epoch 1/10
137/137 [==============================] - 790s 6s/step - loss: 1.2642 - auroc: 0.6249 -

Epoch 00001: val_auroc improved from -inf to 0.70428, saving model to model_1.h5
Epoch 2/10
137/137 [==============================] - 702s 5s/step - loss: 0.9345 - auroc: 0.6939 -

Epoch 00002: val_auroc improved from 0.70428 to 0.72512, saving model to model_1.h5
Epoch 3/10
137/137 [==============================] - 722s 5s/step - loss: 0.7901 - auroc: 0.7154 -

Epoch 00003: val_auroc improved from 0.72512 to 0.73998, saving model to model_1.h5
Epoch 4/10
137/137 [==============================] - 701s 5s/step - loss: 0.7039 - auroc: 0.7288 -

Epoch 00004: val_auroc did not improve from 0.73998
Epoch 5/10
137/137 [==============================] - 722s 5s/step - loss: 0.6326 - auroc: 0.7378 -

Epoch 00005: val_auroc improved from 0.73998 to 0.74809, saving model to model_1.h5
Epoch 6/10
137/137 [==============================] - 723s 5s/step - loss: 0.5765 - auroc: 0.7426 -

Epoch 00006: val_auroc improved from 0.74809 to 0.75099, saving model to model_1.h5
Epoch 7/10
137/137 [==============================] - 720s 5s/step - loss: 0.5354 - auroc: 0.7512 -

Epoch 00007: val_auroc improved from 0.75099 to 0.75237, saving model to model_1.h5
Epoch 8/10
```

```
137/137 [==============================] - 700s 5s/step - loss: 0.5065 - auroc: 0.7543 -

Epoch 00008: val_auroc improved from 0.75237 to 0.75595, saving model to model_1.h5
Epoch 9/10
137/137 [==============================] - 720s 5s/step - loss: 0.4808 - auroc: 0.7565 -

Epoch 00009: val_auroc improved from 0.75595 to 0.76109, saving model to model_1.h5
Epoch 10/10
137/137 [==============================] - 704s 5s/step - loss: 0.4597 - auroc: 0.7604 -

Epoch 00010: val_auroc did not improve from 0.76109
```

Plot loss vs epoch

```
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()


%matplotlib inline
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
x = list(range(1,11))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```



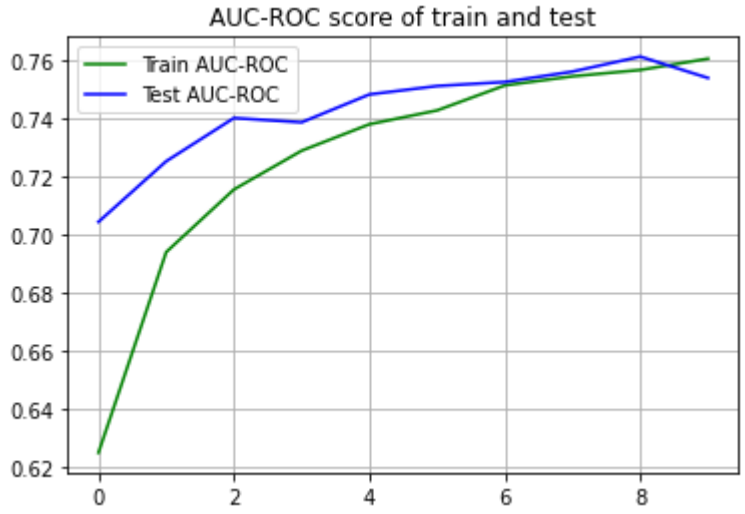
observation:

We can see that at first both the training and validation losses are decreasing but after epoch reaches to 5 validation loss remain constant for sometimes and again it's decrease but training loss keeps on decreasing without any constant

Evaluating test data

```
# Evaluating test data
score_1 = model_1.evaluate(test_data, y_test, verbose = 1, batch_size = 512)
print('Test Loss:', score_1[0])
print('Test ROC-AUC score:', score_1[1], '\n')

# Plotting train and test auc roc score
plt.plot(history.history['auroc'], 'g')
plt.plot(history.history['val_auroc'], 'b')
plt.title("AUC-ROC score of train and test")
plt.legend({'Train AUC-ROC': 'g', 'Test AUC-ROC':'b'})
plt.grid()
plt.show()
```

```
43/43 [==============================] - 78s 2s/step - loss: 0.4525 - auroc: 0.7566
Test Loss: 0.45246291160583496
Test ROC-AUC score: 0.7566255331039429
```



Observation:

- Test Loss - 0.4524
- Test AUC-ROC - 0.7566

Keras Model-1.2

- Activation : { hidden_layer : relu , output_layer : softmax }same as model_1 but slightly different hidden layer number
- optimizer : { 'Adam'}
- loss : { 'categorical_crossentropy' }
- Batch_size : {256}
- epoch : {10}

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
#creating input layer
input_essay = Input(shape=(train_text_encode.shape[1],), name='input_essay')
#creating embedding layer
# This embedding layer will encode the input sequence
# into a sequence of dense 324-dimensional vectors.
x1 =Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1], weights=[embedding_
#creating lstm layer
x1 = LSTM(100, activation = 'relu',kernel_regularizer=regularizers.l2(0.001),return_sequences

layer_1 = Flatten()(x1)
```

+ Code + Text

```
x_concat = concatenate([layer_1, layer_2, layer_3, layer_4, layer_5, layer_6, layer_7])


x = Dense(512,activation="relu", kernel_initializer="he_normal",kernel_regularizer=regularize
x = Dropout(0.5)(x)
x = Dense(256,activation="relu",kernel_initializer="glorot_normal",kernel_regularizer=regular
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(128,activation="relu", kernel_initializer="glorot_normal",kernel_regularizer=regula
x = Dropout(0.4)(x)
x = Dense(50,activation="relu", kernel_initializer="glorot_normal",kernel_regularizer=regular

output = Dense(2, activation='softmax', name='Output')(x)

model_1 = Model(inputs=[input_essay, input_state ,input_prefix,input_cat,
                        input_subcat ,input_grade ,num_feats ],outputs=[output])

print(model_1.summary())
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_essay (InputLayer)        [(None, 300)]        0
__________________________________________________________________________________________________
embed_essay (Embedding)         (None, 300, 300)     14217600    input_essay[0][0]
__________________________________________________________________________________________________
teacher_prefix (InputLayer)     [(None, 1)]          0
__________________________________________________________________________________________________
```

school_state (InputLayer)	[(None, 1)]	0	
grade (InputLayer)	[(None, 1)]	0	
clean_categories (InputLayer)	[(None, 1)]	0	
subject_subcategories (InputLay	[(None, 1)]	0	
lstm_1 (LSTM)	(None, 300, 100)	160400	embed_essay[0][0]
embed_prefix (Embedding)	(None, 1, 26)	1352	teacher_prefix[0][0]
embed_state (Embedding)	(None, 1, 26)	1352	school_state[0][0]
embed_grade (Embedding)	(None, 1, 26)	1352	grade[0][0]
embed_cat (Embedding)	(None, 1, 26)	1352	clean_categories[0][(
embed_subcat (Embedding)	(None, 1, 50)	19200	subject_subcategorie
numerical_features (InputLayer)	[(None, 2)]	0	
flatten_6 (Flatten)	(None, 30000)	0	lstm_1[0][0]
flatten_1 (Flatten)	(None, 26)	0	embed_prefix[0][0]
flatten_2 (Flatten)	(None, 26)	0	embed_state[0][0]
flatten_3 (Flatten)	(None, 26)	0	embed_grade[0][0]
flatten_4 (Flatten)	(None, 26)	0	embed_cat[0][0]
flatten_5 (Flatten)	(None, 50)	0	embed_subcat[0][0]
dense (Dense)	(None, 100)	300	numerical_features[0
concatenate (Concatenate)	(None, 30254)	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_1 (Dense)	(None, 512)	15490560	concatenate[0][0]
dropout (Dropout)	(None, 512)	0	dense_1[0][0]

```
#Getting all data into list
#for train data
train_data = [train_text_encode, train_prefix_encode, train_state_encode, train_grade_encode,
              train_cat_encode, train_subcat_encode, train_rem_input]
#for test data
test_data = [test_text_encode, test_prefix_encode, test_state_encode, test_grade_encode,
             test_cat_encode, test_subcat_encode, test_rem_input]
```

```
#for val data
val_data = [val_text_encode, val_prefix_encode, val_state_encode, val_grade_encode,
            val_cat_encode, val_subcat_encode, val_rem_input]
```

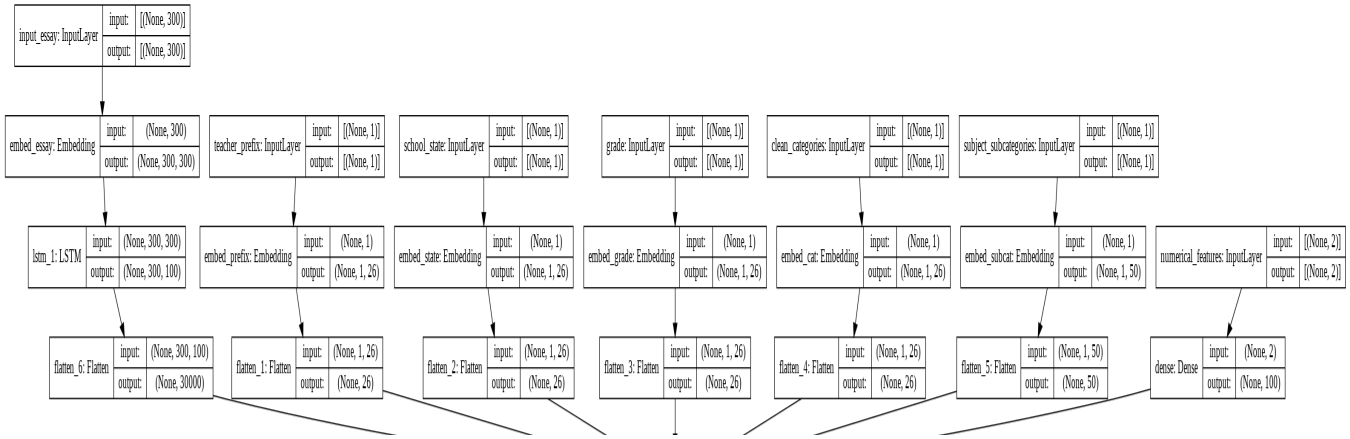
```
train_rem_input.shape
```

```
(69918, 2)
```

Network arcitecture:

```
# https://github.com/mmortazavi/EntityEmbedding-Working_Example/blob/master/EntityEmbedding.i
```

```
import pydot_ng as pydot
from keras.utils.vis_utils import plot_model
from IPython.display import Image

plot_model(model_1, show_shapes = True, show_layer_names = True, to_file = 'model_1.png')

Image(retina = True, filename = 'model_1.png')
```



Custom Callback :

```
checkpoint1 = ModelCheckpoint("model_1.h5",
                              monitor="val_auroc",
                              mode="max",
                              save_best_only = True,
                              verbose=1)
earlystop1 = EarlyStopping(monitor = 'val_auroc',
                           mode="max",
                           min_delta = 0,
                           patience = 2,
                           verbose = 1)

tensorboard1 = TensorBoard(log_dir='Model1_visualization')

callbacks_1 = [checkpoint1,earlystop1,tensorboard1]
```

```
from tensorflow.keras.optimizers import Adam
model_1.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = [auroc])
```

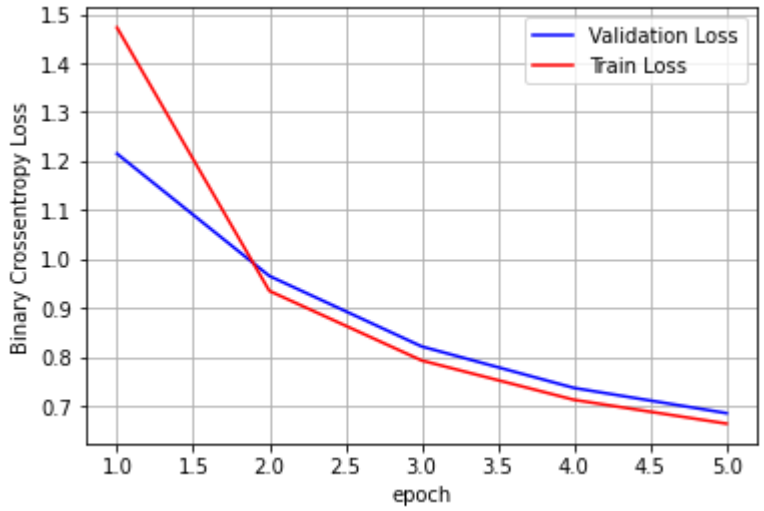
```
history = model_1.fit(train_data, y_train, batch_size=256,epochs = 10, verbose=1,callbacks=ca
```

```
Epoch 1/10
274/274 [==============================] - 689s 3s/step - loss: 1.4726 - auroc: 0.5382 -

Epoch 00001: val_auroc improved from -inf to 0.46973, saving model to model_1.h5
Epoch 2/10
274/274 [==============================] - 647s 2s/step - loss: 0.9344 - auroc: 0.5617 -

Epoch 00002: val_auroc improved from 0.46973 to 0.60970, saving model to model_1.h5
Epoch 3/10
274/274 [==============================] - 673s 2s/step - loss: 0.7922 - auroc: 0.5807 -

Epoch 00003: val_auroc improved from 0.60970 to 0.62846, saving model to model_1.h5
Epoch 4/10
274/274 [==============================] - 666s 2s/step - loss: 0.7118 - auroc: 0.5976 -

Epoch 00004: val_auroc did not improve from 0.62846
```

```
Epoch 5/10
274/274 [==============================] - 666s 2s/step - loss: 0.6632 - auroc: 0.5995 -

Epoch 00005: val_auroc did not improve from 0.62846
Epoch 00005: early stopping
```

Plot loss vs epoch

```
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()


%matplotlib inline
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
x = list(range(1,6))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

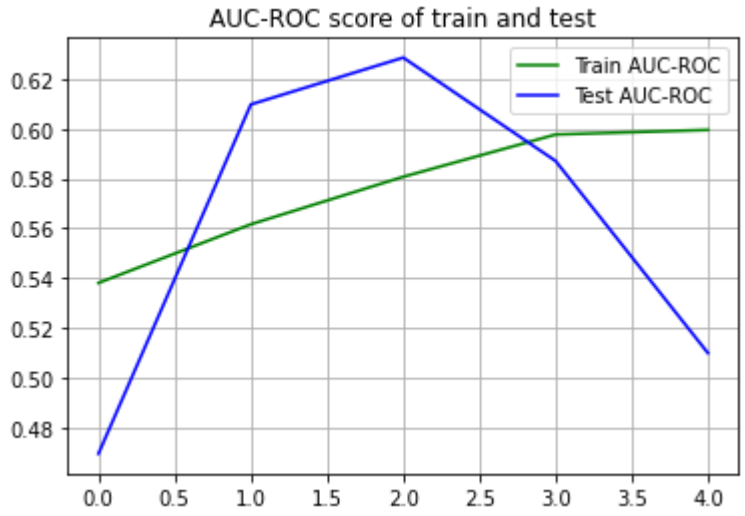


evaliting test data

```
# Evaluating test data
score_1 = model_1.evaluate(test_data, y_test, verbose = 1, batch_size = 256)
print('Test Loss:', score_1[0])
print('Test ROC-AUC score:', score_1[1], '\n')

# Plotting train and test auc roc score
plt.plot(history.history['auroc'], 'g')
plt.plot(history.history['val_auroc'], 'b')
```

```
plt.plot(history.history['val_auroc']) 
plt.title("AUC-ROC score of train and test")
plt.legend({'Train AUC-ROC': 'g', 'Test AUC-ROC':'b'})
plt.grid()
plt.show()
```

```
86/86 [==============================] - 67s 778ms/step - loss: 0.6846 - auroc: 0.5064
Test Loss: 0.684643030166626
Test ROC-AUC score: 0.5064215660095215
```



Keras Model-1.3

- Activation : { hidden_layer : by_default , output_layer : softmax }with different hidden layer number
- optimizer : { 'Adam'}
- loss : { 'categorical_crossentropy' }
- Batch_size : {800}
- epoch : {10}

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
#creating input layer
input_essay = Input(shape=(train_text_encode.shape[1],), name='input_essay')
#creating embedding layer
# This embedding layer will encode the input sequence
# into a sequence of dense 324-dimensional vectors.
x1 =Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1], weights=[embedding_
#creating lstm layer
x1 = LSTM(100, activation = 'relu',kernel_regularizer=regularizers.l2(0.001),return_sequences

layer_1 = Flatten()(x1)
```

```
x_concat = concatenate([layer_1, layer_2, layer_3, layer_4, layer_5, layer_6, layer_7])

#After concatenating text input, categorical and remaining numerical features, applying it to
from keras.layers import Concatenate
x= Dense(256,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.002))(x_concat)
x= Dropout(0.6)(x)
x= Dense(128,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.002))(x)
x= Dropout(0.5)(x)
x= Dense(64,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.002))(x)
x= Dropout(0.5)(x)
x= Dense(32,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.002))(x)
x= Dropout(0.5)(x)
x= Dense(16,activation='relu',kernel_initializer='glorot_normal',kernel_regularizer=l2(0.002)
output = Dense(2, activation='softmax', name='Output')(x)
model_1 = Model(inputs=[input_essay, input_state ,input_prefix,input_cat,
                        input_subcat ,input_grade ,num_feats ],outputs=[output])

print(model_1.summary())
```

```
embed_prefix (Embedding)        (None, 1, 26)        1352        teacher_prefix[0][0]
embed_state (Embedding)         (None, 1, 26)        1352        school_state[0][0]
embed_grade (Embedding)         (None, 1, 26)        1352        grade[0][0]
embed_cat (Embedding)           (None, 1, 26)        1352        clean_categories[0][(
embed_subcat (Embedding)        (None, 1, 50)        19200       subject_subcategorie
numerical_features (InputLayer) [(None, 2)]          0
flatten (Flatten)               (None, 51200)        0           lstm[0][0]
flatten_1 (Flatten)             (None, 26)           0           embed_prefix[0][0]
flatten_2 (Flatten)             (None, 26)           0           embed_state[0][0]
flatten_3 (Flatten)             (None, 26)           0           embed_grade[0][0]
flatten_4 (Flatten)             (None, 26)           0           embed_cat[0][0]
flatten_5 (Flatten)             (None, 50)           0           embed_subcat[0][0]
dense (Dense)                   (None, 64)           192         numerical_features[0
concatenate (Concatenate)       (None, 51418)        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_7 (Dense)                 (None, 256)          13163264    concatenate[0][0]
```

```
dropout_4 (Dropout)             (None, 256)          0           dense_7[0][0]
__________________________________________________________________________________________
dense_8 (Dense)                 (None, 128)          32896       dropout_4[0][0]
__________________________________________________________________________________________
dropout_5 (Dropout)             (None, 128)          0           dense_8[0][0]
__________________________________________________________________________________________
dense_9 (Dense)                 (None, 64)           8256        dropout_5[0][0]
__________________________________________________________________________________________
dropout_6 (Dropout)             (None, 64)           0           dense_9[0][0]
__________________________________________________________________________________________
dense_10 (Dense)                (None, 32)           2080        dropout_6[0][0]
__________________________________________________________________________________________
dropout_7 (Dropout)             (None, 32)           0           dense_10[0][0]
__________________________________________________________________________________________
dense_11 (Dense)                (None, 16)           528         dropout_7[0][0]
__________________________________________________________________________________________
Output (Dense)                  (None, 2)            34          dense_11[0][0]
==========================================================================================
Total params: 27,625,606
Trainable params: 13,451,506
Non-trainable params: 14,174,100
__________________________________________________________________________________________
```

custom callback :

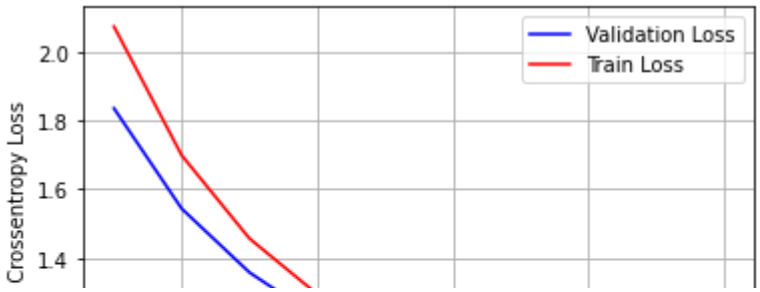
```
checkpoint1 = ModelCheckpoint("model_1.h5",
                              monitor="val_auroc",
                              mode="max",
                              save_best_only = True,
                              verbose=1)
earlystop1 = EarlyStopping(monitor = 'val_auroc',
                           mode="max",
                           min_delta = 0,
                           patience = 2,
                           verbose = 1)

tensorboard1 = TensorBoard(log_dir='Model1_visualization')

callbacks_1 = [checkpoint1,earlystop1,tensorboard1]


from tensorflow.keras.optimizers import Adam
model_1.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = [auroc])


history = model_1.fit(train_data, y_train, batch_size=800,epochs = 10, verbose=1,callbacks=ca
```

```
Epoch 1/10
88/88 [==============================] - 934s 11s/step - loss: 2.0723 - auroc: 0.6062 -

Epoch 00001: val_auroc improved from -inf to 0.69744, saving model to model_1.h5
Epoch 2/10
88/88 [==============================] - 922s 10s/step - loss: 1.6982 - auroc: 0.6400 -
```

```
Epoch 00002: val_auroc improved from 0.69744 to 0.70866, saving model to model_1.h5
Epoch 3/10
88/88 [==============================] - 911s 10s/step - loss: 1.4567 - auroc: 0.6712 -

Epoch 00003: val_auroc improved from 0.70866 to 0.71597, saving model to model_1.h5
Epoch 4/10
88/88 [==============================] - 915s 10s/step - loss: 1.2952 - auroc: 0.6881 -

Epoch 00004: val_auroc improved from 0.71597 to 0.72062, saving model to model_1.h5
Epoch 5/10
88/88 [==============================] - 920s 10s/step - loss: 1.1821 - auroc: 0.7070 -

Epoch 00005: val_auroc improved from 0.72062 to 0.72416, saving model to model_1.h5
Epoch 6/10
88/88 [==============================] - 898s 10s/step - loss: 1.1038 - auroc: 0.7185 -

Epoch 00006: val_auroc improved from 0.72416 to 0.72558, saving model to model_1.h5
Epoch 7/10
88/88 [==============================] - 885s 10s/step - loss: 1.0429 - auroc: 0.7309 -

Epoch 00007: val_auroc improved from 0.72558 to 0.72842, saving model to model_1.h5
Epoch 8/10
88/88 [==============================] - 891s 10s/step - loss: 0.9995 - auroc: 0.7364 -

Epoch 00008: val_auroc improved from 0.72842 to 0.72992, saving model to model_1.h5
Epoch 9/10
88/88 [==============================] - 888s 10s/step - loss: 0.9595 - auroc: 0.7441 -

Epoch 00009: val_auroc did not improve from 0.72992
Epoch 10/10
88/88 [==============================] - 887s 10s/step - loss: 0.9268 - auroc: 0.7495 -

Epoch 00010: val_auroc did not improve from 0.72992
Epoch 00010: early stopping
```

Plot loss vs epoch

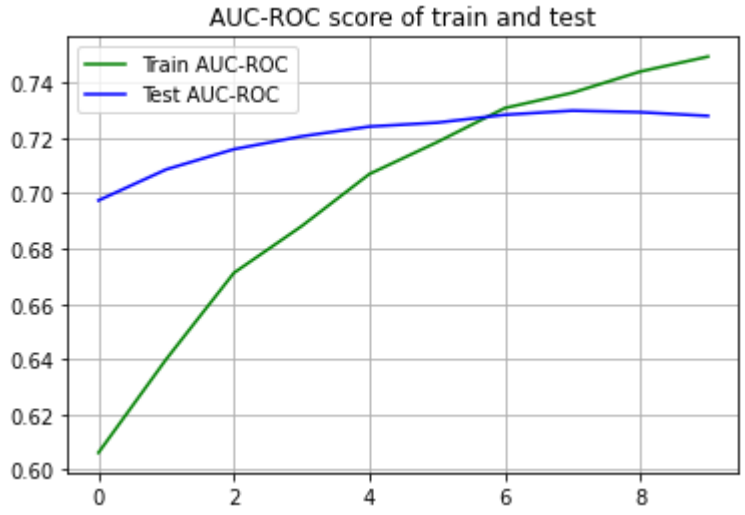
```
%matplotlib inline
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
x = list(range(1,11))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```



auc-roc score

```
# Evaluating test data
score_1 = model_1.evaluate(test_data, y_test, verbose = 1, batch_size = 800)
print('Test Loss:', score_1[0])
print('Test ROC-AUC score:', score_1[1], '\n')

# Plotting train and test auc roc score
plt.plot(history.history['auroc'], 'g')
plt.plot(history.history['val_auroc'], 'b')
plt.title("AUC-ROC score of train and test")
plt.legend({'Train AUC-ROC': 'g', 'Test AUC-ROC':'b'})
plt.grid()
plt.show()
```

```
28/28 [==============================] - 103s 4s/step - loss: 0.9231 - auroc: 0.7278
Test Loss: 0.9231275916099548
Test ROC-AUC score: 0.7277827858924866
```



TASK-2:

Applying TF-IDF vectorizer

Use the same model as above but for 'input_seq_total_text_data' give only some words in the

1. Train the TF-IDF on the Train data
2. Get the idf value for each word we have in the train data.
3. Remove the low idf value and high idf value words from our data. Do some analysis on the
4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train on tot

```
x = project_data.drop(['project_is_approved'], axis = 1)
y = project_data['project_is_approved']
```

```
x.shape
```

```
(109248, 8)
```

```
from sklearn.model_selection import train_test_split

# Splitting into x and y into train and test set
X_train, x_test, Y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42,

# Splitting train set into train and cv set
x_train, x_cv, y_train, y_cv = train_test_split(X_train, Y_train,  test_size = 0.2, random_st
```

```
print("Shape of x_train:", x_train.shape)
print("Shape of x_cv:", x_cv.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_cv:", y_cv.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of x_train: (69918, 8)
Shape of x_cv: (17480, 8)
Shape of x_test: (21850, 8)
Shape of y_train: (69918,)
Shape of y_cv: (17480,)
Shape of y_test: (21850,)
```

```
vectorizer = TfidfVectorizer(min_df=2)#applying tfidf

train_tfidf_text = vectorizer.fit_transform(x_train['essay'])

tfidf_dict = dict(zip(vectorizer.get_feature_names(),list(vectorizer.idf_)))
```

```
tfidf_df = pd.DataFrame(tfidf_dict.items(), columns=['words','idf_values'])
tfidf_df.shape
```

```
(28915, 2)
```

Removing the Rearly occuring words(High idf) and Frequently occuring words(Low idf) as they dont give much information to our model

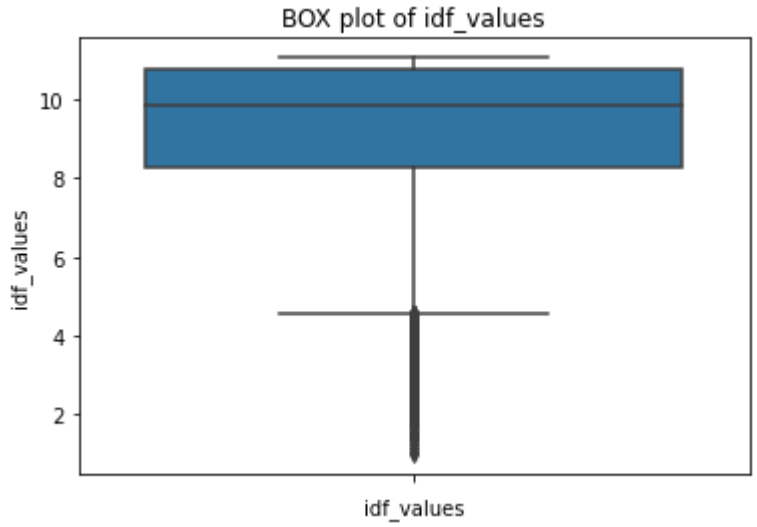
```
print("Min tf-idf value is: ",min(tfidf_df['idf_values']))

print("Max tf-idf value is: ",max(tfidf_df['idf_values']))
```

```
Min tf-idf value is:  1.0075658420924976
Max tf-idf value is:  11.056480419499536
```

Box plot

```
sns.boxplot(y = "idf_values", data = tfidf_df)
plt.xlabel("idf_values")
plt.title("BOX plot of idf_values")
```

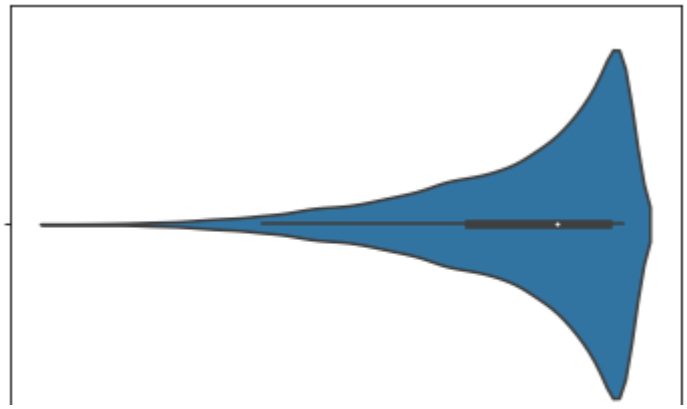
```
Text(0.5, 1.0, 'BOX plot of idf_values')
```



violin plot

```
print("Violin plot for idf values\n")
sns.violinplot(tfidf_df['idf_values'])
plt.show()
```

```
Violin plot for idf values

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```



Percentile Analysis

```
print("0th percentile: ", np.percentile(tfidf_df['idf_values'],0))
print("10th percentile: ", np.percentile(tfidf_df['idf_values'],10))
print("20th percentile: ", np.percentile(tfidf_df['idf_values'],20))
print("30th percentile: ", np.percentile(tfidf_df['idf_values'],30))
print("40th percentile: ", np.percentile(tfidf_df['idf_values'],40))
print("50th percentile: ", np.percentile(tfidf_df['idf_values'],50))
print("60th percentile: ", np.percentile(tfidf_df['idf_values'],60))
print("70th percentile: ", np.percentile(tfidf_df['idf_values'],70))
print("80th percentile: ", np.percentile(tfidf_df['idf_values'],80))
print("90th percentile: ", np.percentile(tfidf_df['idf_values'],90))
print("100th percentile: ", np.percentile(tfidf_df['idf_values'],100))
```

```
0th percentile:  1.0075658420924976
10th percentile:  6.531075201980307
20th percentile:  7.824359367881314
30th percentile:  8.689356805367918
40th percentile:  9.382503985927864
50th percentile:  9.8525076151736
60th percentile:  10.36333323893959
70th percentile:  10.545654795733544
80th percentile:  10.768798347047754
90th percentile:  11.056480419499536
100th percentile:  11.056480419499536
```

Number of Unique words in the corpus

```
filter =   (tfidf_df['idf_values']>=2) & (tfidf_df['idf_values'] <=11.05)
tfidf_best = tfidf_df[filter]
print(tfidf_best.shape)
```

```
(23543, 2)
```

Post Padding

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
def encoding_padding(input_text, max_length):
    encoded_data = tokenizer.texts_to_sequences(input_text)
    padded_data = pad_sequences(encoded_data, maxlen=max_length, padding='post')
    return padded_data
```

Tokenize:

Input data to layer should be integer. So, using tokenize inbuilt function, we will integer encode the text data.

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
train_text = x_train["essay"].values.tolist()
test_text = x_test['essay'].values.tolist()
val_text = x_cv['essay'].values.tolist()

tokenizer = Tokenizer()
tokenizer.fit_on_texts(tfidf_best['words'].values.tolist())
vocab_size = len(tokenizer.word_index) + 1

max_length = 400
train_text_encode = encoding_padding(train_text,  max_length=max_length)
test_text_encode = encoding_padding(test_text, max_length=max_length)
val_text_encode = encoding_padding(val_text, max_length=max_length)

print(train_text_encode.shape)
print(test_text_encode.shape)
print(val_text_encode.shape)
```

```
(69918, 400)
(21850, 400)
(17480, 400)
```

```
train_text_encode[0]
```

```
array([16051, 11854, 17900, 13705, 16677, 16112, 16346, 12651, 10763,
       10205, 10669,  8710,  3453,   881,   752,  6794, 13402, 15046,
       19571, 18533, 12054, 12488, 14064, 12651, 10763,  9381, 10337,
       12488,  8058, 13203, 10218,  8044,  1474, 17461,  1020,  1876,
       21297, 13128,  8909, 17621,  1712, 20852, 22794,  9209, 20237,
        2461,  7836, 18700,  8716,  6959,  3455, 21119,  6535, 10370,
       10622,  9216, 16954,  7836, 22645, 19343,  7174,  6012,  8316,
       13731, 19123,  6012, 11139,  6012, 16051, 12470, 19123, 16346,
       20219, 16506, 10084, 16681,  6958, 16501, 12619, 14358,  9167,
```

```
11932, 12350, 10201, 12350, 19082, 20852, 22797,  9814, 16500,
 2461,  6959,  7834, 15962, 11932, 16112,  7836,  8583,  6535,
 3118,  8716,  9591,  1623,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0,     0,     0,     0,     0,     0,
    0,     0,     0,     0], dtype=int32)
```

Weight Matrix

```
glove_vector = open("/content/drive/MyDrive/Lstm on donor choose/glove_vectors","rb")
glove_words = pickle.load(glove_vector)


#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

print(embedding_matrix.shape)
embedding_matrix[10]
```

```
(23544, 300)
array([-4.9187e-01, -8.9405e-02, -1.1349e-01,  2.5081e-01,  1.7025e-01,
        4.4045e-02, -8.0568e-01, -5.3650e-02, -1.5190e-01,  3.2618e-01,
       -2.6259e-01,  7.9806e-01,  7.7951e-02, -1.5277e-01,  3.0831e-01,
        1.8557e-01, -5.0393e-02,  3.9665e-02, -1.5349e-01, -9.0685e-03,
        4.3312e-01, -1.8381e-01,  2.5506e-01,  3.4104e-01, -2.2033e-01,
       -1.5333e-02, -1.7249e-01,  2.6203e-01, -2.6015e-02,  6.9308e-01,
       -1.2871e-01, -1.1590e-01,  1.1784e-01,  3.8740e-01, -1.0099e-01,
       -1.1430e-01, -3.1958e-01, -1.9918e-01,  3.3425e-01,  1.4817e-01,
        6.2132e-01, -1.9276e-01,  2.3862e-01,  1.5748e-01,  3.4096e-01,
        5.6498e-01,  1.7457e-01, -8.1865e-02, -5.1860e-01, -2.7357e-01,
        2.1368e-01, -6.6864e-02, -4.3518e-01, -1.3831e-01, -3.5474e-01,
       -4.5110e-01, -1.1627e-01, -1.8612e-01,  4.7144e-01,  9.4884e-03,
        3.9908e-02,  5.0543e-02,  6.9555e-02,  5.7015e-01,  2.9583e-01,
        4.2223e-02, -7.1775e-02,  1.7495e-01,  7.3164e-02,  2.3304e-01,
       -2.6650e-01,  5.3532e-01,  4.3914e-02, -6.5203e-01, -6.8566e-01,
       -8.2907e-02, -3.7656e-01,  5.7350e-01, -1.3352e-01, -4.4649e-01,
       -1.2475e-01, -6.5516e-01,  2.3431e-01, -4.1214e-01, -1.0343e-01,
        1.2017e-01, -3.6009e-01, -6.4713e-02,  3.1595e-01, -2.7782e-02,
       -4.3309e-01, -1.8813e-01, -1.3203e-01,  4.6149e-02, -6.4676e-02,
        1.0715e+00, -9.6998e-01,  3.4413e-01,  2.2871e-01, -1.2014e-01,
        9.0996e-02, -3.6546e-01, -5.0502e-01, -6.5768e-02,  1.2118e-01,
        6.1523e-02,  4.2013e-01,  7.4571e-01,  4.4129e-02,  3.4473e-01,
       -3.4706e-01, -2.7585e-02, -2.4270e-01, -2.7324e-01,  6.5079e-02,
        2.9341e-01, -3.8808e-03,  3.2191e-01,  4.8548e-01, -1.4890e-01,
       -1.2582e-01, -3.4970e-02,  1.8268e-01,  9.3944e-01,  2.5956e-01,
       -3.2127e-01, -2.2122e-01,  7.5386e-02,  1.2847e-01, -2.6409e-01,
       -5.8131e-02,  6.0891e-01, -8.4910e-02, -3.0767e-01,  3.0699e-01,
        3.2760e-02, -3.7458e-02, -7.6022e-01, -7.7310e-02,  8.4979e-01,
       -8.2349e-02, -1.8578e-02, -6.8491e-01, -3.8755e-01, -4.4790e-01,
       -2.5622e-01, -5.6952e-02, -8.9210e-02, -3.2741e-01,  4.7400e-01,
       -1.4748e-01, -1.0137e-01, -6.3825e-02,  2.0334e-02, -4.4217e-01,
        4.2829e-01,  2.5378e-01, -1.8118e-01,  5.5888e-01, -1.1123e-02,
        1.2857e-01,  5.3388e-02,  1.4565e-01, -3.6059e-01,  2.3134e-01,
       -6.0033e-01, -2.6475e-01, -1.0949e-01,  1.8399e-01, -2.8914e-01,
       -3.9746e-01,  5.1794e-01,  3.8130e-01,  5.3080e-03,  3.8112e-01,
       -2.9385e-01,  3.1561e-01, -3.7626e-02,  2.9905e-01,  2.1323e-02,
        7.6283e-02,  2.5145e-01,  4.7010e-01,  3.6360e-01,  3.9543e-01,
        4.2386e-01, -3.3270e-01,  3.5150e-01, -1.6398e-01, -1.3567e-02,
       -1.5612e-01, -3.3786e-01,  1.4262e-01, -3.8185e-01, -2.0750e-01,
       -3.2661e-01, -2.5169e-01,  4.6524e-01, -5.6881e-02,  4.1625e-01,
        4.9434e-02,  2.6345e-01, -2.4414e-01, -5.5193e-02,  3.1373e-01,
       -5.8117e-01,  2.9507e-02, -2.4934e-02,  9.3141e-02,  2.2387e-01,
       -3.4073e-01, -6.2888e-01, -2.2027e-01,  1.1875e-01,  1.3184e-01,
       -4.9809e-01,  6.3094e-02, -1.8543e-01, -1.2005e-01,  1.2974e-01,
        2.7520e-01, -2.1197e-01,  1.9111e-02,  3.9026e-01, -3.8508e+00,
        9.3437e-02,  1.9557e-01,  1.3566e-01,  6.4371e-01,  9.3575e-02,
        5.3613e-02,  7.2795e-01,  1.7668e-01, -3.8319e-01,  3.5036e-01,
        6.1217e-02, -1.8301e-01, -3.7741e-03, -2.2122e-01,  3.2764e-01,
        1.5131e-01, -1.4968e-01,  3.1796e-01, -1.7629e-01,  9.7184e-02,
       -1.8324e-03, -9.6001e-02,  2.7139e-01, -2.2187e-01, -6.3296e-01,
        1.8938e-01,  4.9391e-01, -8.0781e-02,  6.3653e-03,  1.9583e-01,
        4.3653e-02,  7.1280e-02,  3.4965e-02, -9.7081e-02, -7.1663e-02,
       -4.3685e-01,  1.0686e-01, -1.1850e-01,  4.6630e-02,  1.4026e-01,
       -2.8656e-01,  2.7052e-01, -8.8886e-02,  1.2114e-03,  3.6120e-01,
       -1.1939e-01, -2.4443e-01,  1.3746e-01,  4.6906e-02,  5.6918e-02,
        1.5952e-02, -1.4557e-01, -2.1556e-01,  2.9041e-02, -1.4165e-01,
```

```
            -3.0461e-01, -3.4972e-02,  6.3759e-01, -2.0046e-02, -1.0334e-01,
            -5.0095e-01, -1.0725e-01,  7.4902e-02,  1.7008e-01, -7.1300e-01,
```

```
vocab_size
```

```
23544
```

```
train_text_encode.shape[1]
```

```
400
```

Embedding layer for text data

```
# Creating an input layer
input_essay = Input(shape=(train_text_encode.shape[1],), name='input_essay')
# Creating an embedding layer
x1 =Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1], weights=[embedding_
# Creating LSTM layer
x1 = LSTM(128,kernel_regularizer=regularizers.l2(0.001),return_sequences=True)(x1)

layer_1 = Flatten()(x1)
```

Concatenating all the flattened layers

```
x_concat = concatenate([layer_1, layer_2, layer_3, layer_4, layer_5, layer_6, layer_7])
```

Keras Model-2

- Activation : { hidden_layer : relu , output_layer : softmax }
- optimizer : { 'Adam'}
- loss : { 'categorical_crossentropy' }
- Batch_size : {512}
- epoch : {10}

```
x1 = Dense(100, activation="relu", kernel_initializer="he_normal" ,kernel_regularizer=regular
x1 = Dropout(0.5)(x1)
x1 = Dense(250,activation="relu",kernel_initializer="glorot_normal" ,kernel_regularizer=regul
x1 = BatchNormalization()(x1)
x1 = Dropout(0.5)(x1)
x1 = Dense(100,activation="relu", kernel_initializer="glorot_normal" ,kernel_regularizer=regu
output = Dense(2, activation='softmax', name='output')(x1)
```

```
model_2 = Model(inputs=[input_essay, input_state ,input_prefix,input_cat,
                        input_subcat ,input_grade ,num_feats ],outputs=[output])
```

```
print(model_2.summary())
```

```
__________________________________________________________________________________________________
clean_categories (InputLayer)   [(None, 1)]          0
__________________________________________________________________________________________________
subject_subcategories (InputLay [(None, 1)]          0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 400, 128)     219648      embed_essay[0][0]
__________________________________________________________________________________________________
embed_prefix (Embedding)        (None, 1, 26)        1352        teacher_prefix[0][0]
__________________________________________________________________________________________________
embed_state (Embedding)         (None, 1, 26)        1352        school_state[0][0]
__________________________________________________________________________________________________
embed_grade (Embedding)         (None, 1, 26)        1352        grade[0][0]
__________________________________________________________________________________________________
embed_cat (Embedding)           (None, 1, 26)        1352        clean_categories[0][(
__________________________________________________________________________________________________
embed_subcat (Embedding)        (None, 1, 50)        19200       subject_subcategorie
__________________________________________________________________________________________________
numerical_features (InputLayer) [(None, 2)]          0
__________________________________________________________________________________________________
flatten_6 (Flatten)             (None, 51200)        0           lstm_1[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 26)           0           embed_prefix[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 26)           0           embed_state[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 26)           0           embed_grade[0][0]
__________________________________________________________________________________________________
flatten_4 (Flatten)             (None, 26)           0           embed_cat[0][0]
__________________________________________________________________________________________________
flatten_5 (Flatten)             (None, 50)           0           embed_subcat[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 64)           192         numerical_features[0
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 51418)        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)          5141900     concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 100)          0           dense_5[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 250)          25250       dropout_3[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 250)          1000        dense_6[0][0]
__________________________________________________________________________________________________
```

```
dropout_4 (Dropout)             (None, 250)          0           batch_normalization_
__________________________________________________________________________________________
dense_7 (Dense)                 (None, 100)          25100       dropout_4[0][0]
__________________________________________________________________________________________
output (Dense)                  (None, 2)            202         dense_7[0][0]
==========================================================================================
Total params: 12,501,100
Trainable params: 5,437,400
Non-trainable params: 7,063,700
```

Getting all data into list

```
#Getting all data into list
#for train data
train_data = [train_text_encode, train_prefix_encode, train_state_encode, train_grade_encode,
              train_cat_encode, train_subcat_encode, train_rem_input]
#for test data
test_data = [test_text_encode, test_prefix_encode, test_state_encode, test_grade_encode,
             test_cat_encode, test_subcat_encode, test_rem_input]
#cv data
val_data = [val_text_encode, val_prefix_encode, val_state_encode, val_grade_encode,
            val_cat_encode, val_subcat_encode, val_rem_input]


checkpoint2 = ModelCheckpoint("model_2.h5",
                              monitor="val_auroc",
                              mode="max",
                              save_best_only = True,
                              verbose=1)
earlystop2 = EarlyStopping(monitor = 'val_auroc',
                           mode="max",
                           min_delta = 0,
                           patience = 3,
                           verbose = 1)

tensorboard2 = TensorBoard(log_dir='Model2_visualization')

callbacks_2 = [checkpoint2,earlystop2,tensorboard2]


from keras.utils import np_utils

#np_utils.to_categorical is used to convert array of labeled data(from 0 to nb_classes-1) to

y_train = np_utils.to_categorical(y_train, 2)
y_test = np_utils.to_categorical(y_test, 2)
y_cv = np_utils.to_categorical(y_cv, 2)


y_cv.shape
```

```
(17480, 2)
```

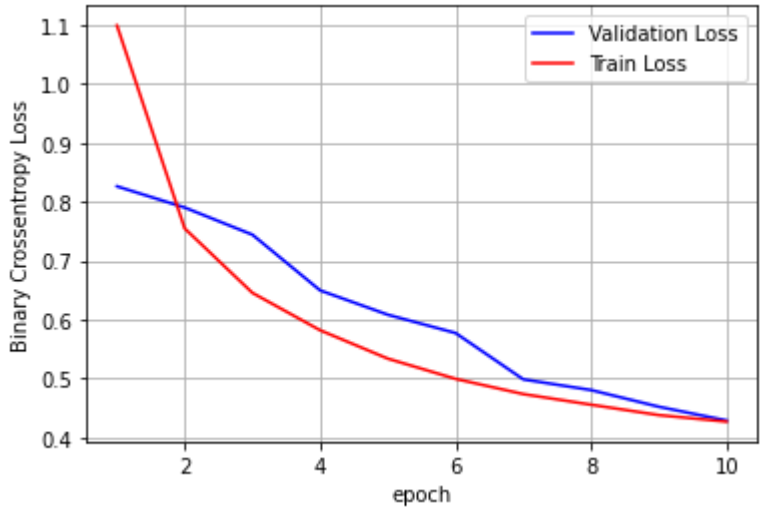
```
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auroc])

history1 = model_2.fit(train_data, y_train, batch_size=512, epochs=10, verbose=1,callbacks=ca
```

```
Epoch 1/10
137/137 [==============================] - 732s 5s/step - loss: 1.0989 - auroc: 0.6023 -

Epoch 00001: val_auroc improved from -inf to 0.69881, saving model to model_2.h5
Epoch 2/10
137/137 [==============================] - 723s 5s/step - loss: 0.7547 - auroc: 0.6885 -

Epoch 00002: val_auroc improved from 0.69881 to 0.72176, saving model to model_2.h5
Epoch 3/10
137/137 [==============================] - 706s 5s/step - loss: 0.6454 - auroc: 0.7106 -

Epoch 00003: val_auroc improved from 0.72176 to 0.73648, saving model to model_2.h5
Epoch 4/10
137/137 [==============================] - 725s 5s/step - loss: 0.5820 - auroc: 0.7184 -

Epoch 00004: val_auroc improved from 0.73648 to 0.73683, saving model to model_2.h5
Epoch 5/10
137/137 [==============================] - 744s 5s/step - loss: 0.5338 - auroc: 0.7268 -

Epoch 00005: val_auroc improved from 0.73683 to 0.74679, saving model to model_2.h5
Epoch 6/10
137/137 [==============================] - 724s 5s/step - loss: 0.4995 - auroc: 0.7355 -

Epoch 00006: val_auroc did not improve from 0.74679
Epoch 7/10
137/137 [==============================] - 731s 5s/step - loss: 0.4737 - auroc: 0.7428 -

Epoch 00007: val_auroc improved from 0.74679 to 0.74943, saving model to model_2.h5
Epoch 8/10
137/137 [==============================] - 699s 5s/step - loss: 0.4557 - auroc: 0.7452 -

Epoch 00008: val_auroc improved from 0.74943 to 0.75173, saving model to model_2.h5
Epoch 9/10
137/137 [==============================] - 700s 5s/step - loss: 0.4380 - auroc: 0.7513 -

Epoch 00009: val_auroc did not improve from 0.75173
Epoch 10/10
137/137 [==============================] - 690s 5s/step - loss: 0.4269 - auroc: 0.7510 -

Epoch 00010: val_auroc did not improve from 0.75173
```

plot epoch vs loss

```
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()
```

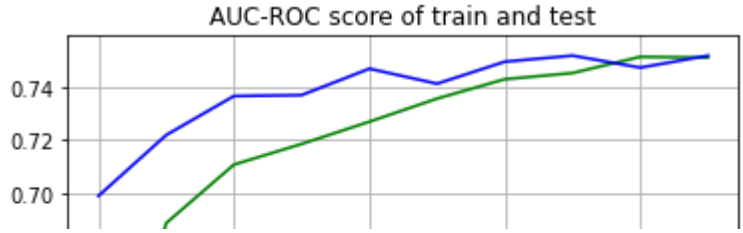
```
    plt.grid()
    fig.canvas.draw()


%matplotlib inline
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
x = list(range(1,11))
vy = history1.history['val_loss']
ty = history1.history['loss']
plt_dynamic(x, vy, ty, ax)
```



auc-roc score

```
# Evaluating test data
score_1 = model_2.evaluate(test_data, y_test, verbose = 1, batch_size = 512)
print('Test Loss:', score_1[0])
print('Test ROC-AUC score:', score_1[1], '\n')

# Plotting train and test auc roc score
plt.plot(history1.history['auroc'], 'g')
plt.plot(history1.history['val_auroc'], 'b')
plt.title("AUC-ROC score of train and test")
plt.legend({'Train AUC-ROC': 'g', 'Test AUC-ROC':'b'})
plt.grid()
plt.show()
```

```
43/43 [==============================] - 78s 2s/step - loss: 0.4272 - auroc: 0.7572
Test Loss: 0.42724213004112244
Test ROC-AUC score: 0.7571699619293213
```

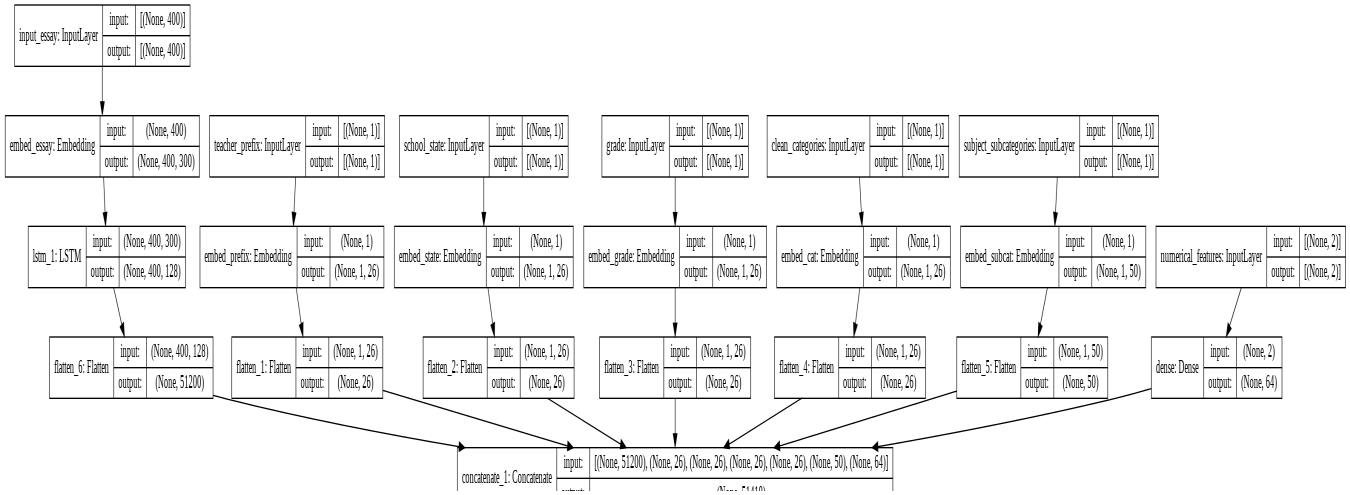


Observation:

- Test loss - 0.4272
- Test AUC-ROC - 0.7571

Network Artitecture

```
#https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
from keras.utils.vis_utils import plot_model
import pydot_ng as pydot
from IPython.display import Image

plot_model(model_2, show_shapes = True, show_layer_names = True, to_file = 'model_2.png')

Image(retina = True, filename = 'model_2.png')
```



Keras Model-2.2

- Activation : { hidden_layer : relu , output_layer : softmax }same as model2.1 with different hidden number.
- optimizer : { 'Adam'}
- loss : { 'categorical_crossentropy' }
- Batch_size : {256}
- epoch : {10}

```
# Creating an input layer
input_essay = Input(shape=(train_text_encode.shape[1],), name='input_essay')
# Creating an embedding layer
x1 =Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1], weights=[embedding_
# Creating LSTM layer
x1 = LSTM(128,kernel_regularizer=regularizers.l2(0.001),return_sequences=True)(x1)

layer_1 = Flatten()(x1)


x_concat = concatenate([layer_1, layer_2, layer_3, layer_4, layer_5, layer_6, layer_7])


x1 = Dense(256, activation="relu", kernel_initializer="he_normal" ,kernel_regularizer=regular
x1 = Dropout(0.5)(x1)
x1 = Dense(128,activation="relu",kernel_initializer="glorot_normal" ,kernel_regularizer=regul
x1 = BatchNormalization()(x1)
x1 = Dropout(0.5)(x1)
x1 = Dense(64,activation="relu", kernel_initializer="glorot_normal" ,kernel_regularizer=regul
output = Dense(2, activation='softmax', name='output')(x1)

model_2 = Model(inputs=[input_essay, input_state ,input_prefix,input_cat,
                        input_subcat ,input_grade ,num_feats ],outputs=[output])
```

```
print(model_2.summary())
```

clean_categories (InputLayer)	[(None, 1)]	0	
subject_subcategories (InputLay	[(None, 1)]	0	
lstm_2 (LSTM)	(None, 400, 128)	219648	embed_essay[0][0]
embed_prefix (Embedding)	(None, 1, 26)	1352	teacher_prefix[0][0]
embed_state (Embedding)	(None, 1, 26)	1352	school_state[0][0]
embed_grade (Embedding)	(None, 1, 26)	1352	grade[0][0]
embed_cat (Embedding)	(None, 1, 26)	1352	clean_categories[0][(
embed_subcat (Embedding)	(None, 1, 50)	19200	subject_subcategorie
numerical_features (InputLayer)	[(None, 2)]	0	
flatten_7 (Flatten)	(None, 51200)	0	lstm_2[0][0]
flatten_1 (Flatten)	(None, 26)	0	embed_prefix[0][0]
flatten_2 (Flatten)	(None, 26)	0	embed_state[0][0]
flatten_3 (Flatten)	(None, 26)	0	embed_grade[0][0]
flatten_4 (Flatten)	(None, 26)	0	embed_cat[0][0]
flatten_5 (Flatten)	(None, 50)	0	embed_subcat[0][0]
dense (Dense)	(None, 64)	192	numerical_features[0
concatenate_2 (Concatenate)	(None, 51418)	0	flatten_7[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_8 (Dense)	(None, 256)	13163264	concatenate_2[0][0]
dropout_5 (Dropout)	(None, 256)	0	dense_8[0][0]
dense_9 (Dense)	(None, 128)	32896	dropout_5[0][0]
batch_normalization_2 (BatchNor	(None, 128)	512	dense_9[0][0]
dropout_6 (Dropout)	(None, 128)	0	batch_normalization_
dense_10 (Dense)	(None, 64)	8256	dropout_6[0][0]
output (Dense)	(None, 2)	130	dense_10[0][0]

```
==================================================================================================
Total params: 20,512,706
Trainable params: 13,449,250
Non-trainable params: 7,063,456
__________________________________________________________________________________________________
```

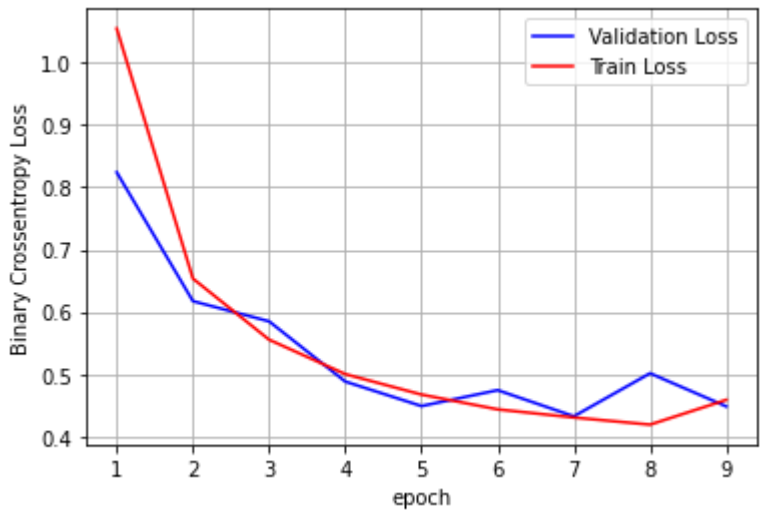
```
#Getting all data into list
#for train data
train_data = [train_text_encode, train_prefix_encode, train_state_encode, train_grade_encode,
              train_cat_encode, train_subcat_encode, train_rem_input]
#for test data
test_data = [test_text_encode, test_prefix_encode, test_state_encode, test_grade_encode,
             test_cat_encode, test_subcat_encode, test_rem_input]
#cv data
val_data = [val_text_encode, val_prefix_encode, val_state_encode, val_grade_encode,
            val_cat_encode, val_subcat_encode, val_rem_input]


checkpoint2 = ModelCheckpoint("model_2.h5",
                              monitor="val_auroc",
                              mode="max",
                              save_best_only = True,
                              verbose=1)
earlystop2 = EarlyStopping(monitor = 'val_auroc',
                           mode="max",
                           min_delta = 0,
                           patience = 2,
                           verbose = 1)

tensorboard2 = TensorBoard(log_dir='Model2_visualization')

callbacks_2 = [checkpoint2,earlystop2,tensorboard2]


model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auroc])

history1 = model_2.fit(train_data, y_train, batch_size=256, epochs=10, verbose=1,callbacks=ca
```

```
Epoch 1/10
274/274 [==============================] - 59s 189ms/step - loss: 1.0549 - auroc: 0.5936

Epoch 00001: val_auroc improved from -inf to 0.69715, saving model to model_2.h5
Epoch 2/10
274/274 [==============================] - 50s 181ms/step - loss: 0.6533 - auroc: 0.6763

Epoch 00002: val_auroc improved from 0.69715 to 0.71510, saving model to model_2.h5
Epoch 3/10
274/274 [==============================] - 51s 186ms/step - loss: 0.5552 - auroc: 0.7025

Epoch 00003: val_auroc improved from 0.71510 to 0.73516, saving model to model_2.h5
Epoch 4/10
274/274 [==============================] - 51s 186ms/step - loss: 0.4999 - auroc: 0.7208
```

```
Epoch 00004: val_auroc improved from 0.73516 to 0.73803, saving model to model_2.h5
Epoch 5/10
274/274 [==============================] - 51s 186ms/step - loss: 0.4670 - auroc: 0.7256

Epoch 00005: val_auroc improved from 0.73803 to 0.73899, saving model to model_2.h5
Epoch 6/10
274/274 [==============================] - 50s 181ms/step - loss: 0.4433 - auroc: 0.7318

Epoch 00006: val_auroc improved from 0.73899 to 0.74337, saving model to model_2.h5
Epoch 7/10
274/274 [==============================] - 51s 186ms/step - loss: 0.4302 - auroc: 0.7346

Epoch 00007: val_auroc improved from 0.74337 to 0.74347, saving model to model_2.h5
Epoch 8/10
274/274 [==============================] - 51s 186ms/step - loss: 0.4188 - auroc: 0.7288

Epoch 00008: val_auroc did not improve from 0.74347
Epoch 9/10
274/274 [==============================] - 50s 182ms/step - loss: 0.4585 - auroc: 0.4999

Epoch 00009: val_auroc did not improve from 0.74347
Epoch 00009: early stopping
```

plot epoch vs loss

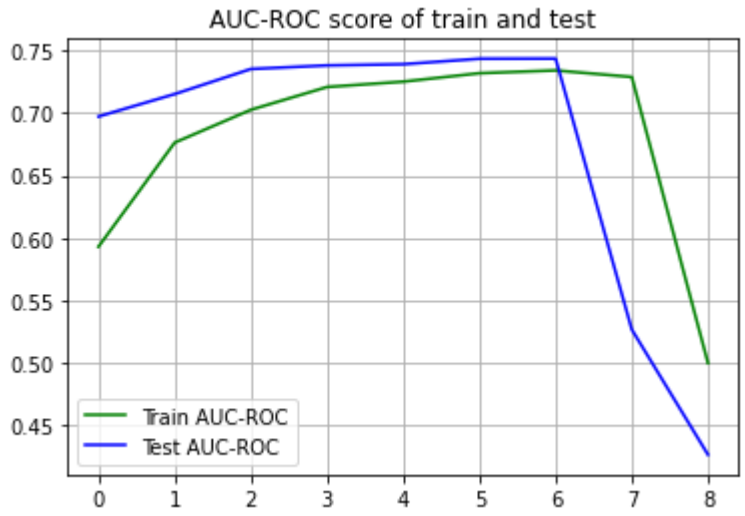
```
%matplotlib inline
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
x = list(range(1,10))
vy = history1.history['val_loss']
ty = history1.history['loss']
plt_dynamic(x, vy, ty, ax)
```



auc-roc score

```
# Evaluating test data
score_1 = model_2.evaluate(test_data, y_test, verbose = 1, batch_size = 512)
print('Test Loss:', score_1[0])
print('Test ROC-AUC score:', score_1[1], '\n')

# Plotting train and test auc roc score
plt.plot(history1.history['auroc'], 'g')
plt.plot(history1.history['val_auroc'], 'b')
plt.title("AUC-ROC score of train and test")
plt.legend({'Train AUC-ROC': 'g', 'Test AUC-ROC':'b'})
plt.grid()
plt.show()
```

```
43/43 [==============================] - 4s 97ms/step - loss: 0.4480 - auroc: 0.4117
Test Loss: 0.4480125904083252
Test ROC-AUC score: 0.41168510913848877
```



Keras Model-2.3

- Activation : { hidden_layer : relu,sigmoid , output_layer : softmax }with different hidden layer and recuurent dropout.
- optimizer : { Adam(0.0005, decay=1e-6)}
- loss : { binary_crossentropy' }
- Batch_size : {850}
- epoch : {10}

```
# Creating an input layer
input_essay = Input(shape=(train_text_encode.shape[1],), name='input_essay')
# Creating an embedding layer
x1 =Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1], weights=[embedding_
# Creating LSTM layer
```

```
x1 = LSTM(100,recurrent_dropout=0.5,kernel_regularizer=regularizers.l2(0.001),return_sequence

layer_1 = Flatten()(x1)
```

```
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernels since it doesn't meet the cri
```

```
x_concat = concatenate([layer_1, layer_2, layer_3, layer_4, layer_5, layer_6, layer_7])


x = Dense(50, activation="relu", kernel_initializer="he_normal" ,kernel_regularizer=regulariz
x = Dropout(0.25)(x)
x = Dense(200,activation="relu",kernel_initializer="glorot_normal" ,kernel_regularizer=regula
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(80,activation="sigmoid", kernel_initializer="glorot_normal" ,kernel_regularizer=reg
output = Dense(2, activation='softmax', name='output')(x)

model2_1 = Model(inputs=[input_essay, input_state ,input_prefix,input_cat,
                         input_subcat ,input_grade ,num_feats ],outputs=[output])



print(model2_1.summary())
```

```
__________________________________________________________________________________________
clean_categories (InputLayer)   [(None, 1)]          0
__________________________________________________________________________________________
subject_subcategories (InputLay [(None, 1)]          0
__________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 400, 100)     160400      embed_essay[0][0]
__________________________________________________________________________________________
embed_prefix (Embedding)        (None, 1, 26)        1352        teacher_prefix[0][0]
__________________________________________________________________________________________
embed_state (Embedding)         (None, 1, 26)        1352        school_state[0][0]
__________________________________________________________________________________________
embed_grade (Embedding)         (None, 1, 26)        1352        grade[0][0]
__________________________________________________________________________________________
embed_cat (Embedding)           (None, 1, 26)        1352        clean_categories[0][(
__________________________________________________________________________________________
embed_subcat (Embedding)        (None, 1, 50)        19200       subject_subcategorie

__________________________________________________________________________________________
numerical_features (InputLayer) [(None, 2)]          0
__________________________________________________________________________________________
flatten_9 (Flatten)             (None, 40000)        0           lstm_4[0][0]
__________________________________________________________________________________________
flatten_1 (Flatten)             (None, 26)           0           embed_prefix[0][0]
__________________________________________________________________________________________
flatten_2 (Flatten)             (None, 26)           0           embed_state[0][0]
__________________________________________________________________________________________
flatten_3 (Flatten)             (None, 26)           0           embed_grade[0][0]
__________________________________________________________________________________________
flatten_4 (Flatten)             (None, 26)           0           embed_cat[0][0]
__________________________________________________________________________________________
flatten_5 (Flatten)             (None, 50)           0           embed_subcat[0][0]
```

```
dense (Dense)                   (None, 64)           192         numerical_features[0
__________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 40218)        0           flatten_9[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_15 (Dense)                (None, 50)           2010950     concatenate_3[0][0]
__________________________________________________________________________________________
dropout_9 (Dropout)             (None, 50)           0           dense_15[0][0]
__________________________________________________________________________________________
dense_16 (Dense)                (None, 200)          10200       dropout_9[0][0]
__________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 200)          800         dense_16[0][0]
__________________________________________________________________________________________
dropout_10 (Dropout)            (None, 200)          0           batch_normalization_4
__________________________________________________________________________________________
dense_17 (Dense)                (None, 80)           16080       dropout_10[0][0]
__________________________________________________________________________________________
output (Dense)                  (None, 2)            162         dense_17[0][0]
==========================================================================================
Total params: 9,286,592
Trainable params: 2,222,992
Non-trainable params: 7,063,600
__________________________________________________________________________________________
```

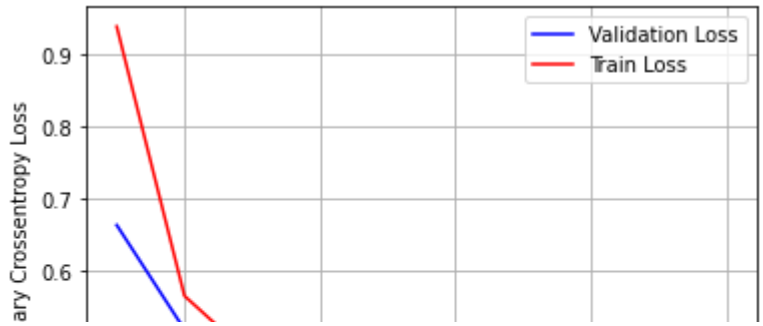
```
checkpoint2 = ModelCheckpoint("model2_1.h5",
                              monitor="val_auroc",
                              mode="max",
                              save_best_only = True,
                              verbose=1)
earlystop2 = EarlyStopping(monitor = 'val_auroc',
                           mode="max",
                           min_delta = 0,
                           patience = 2,
                           verbose = 1)

tensorboard2 = TensorBoard(log_dir='model2_1_visualization')

callbacks_2 = [checkpoint2,earlystop2,tensorboard2]


from tensorflow.keras import optimizers
model2_1.compile(optimizer=optimizers.Adam(0.0005, decay=1e-6), loss='binary_crossentropy', m
history2_1= model2_1.fit(train_data, y_train, batch_size=850, epochs=10, verbose=1,callbacks=
```

```
Epoch 1/10
83/83 [==============================] - 161s 2s/step - loss: 0.9378 - auroc: 0.5799 - \

Epoch 00001: val_auroc improved from -inf to 0.68978, saving model to model2_1.h5
```

```
Epoch 2/10
83/83 [==============================] - 142s 2s/step - loss: 0.5649 - auroc: 0.6925 - \

Epoch 00002: val_auroc improved from 0.68978 to 0.70634, saving model to model2_1.h5
Epoch 3/10
83/83 [==============================] - 145s 2s/step - loss: 0.4797 - auroc: 0.7071 - \

Epoch 00003: val_auroc improved from 0.70634 to 0.71776, saving model to model2_1.h5
Epoch 4/10
83/83 [==============================] - 146s 2s/step - loss: 0.4437 - auroc: 0.7187 - \

Epoch 00004: val_auroc improved from 0.71776 to 0.72383, saving model to model2_1.h5
Epoch 5/10
83/83 [==============================] - 141s 2s/step - loss: 0.4236 - auroc: 0.7260 - \

Epoch 00005: val_auroc did not improve from 0.72383
Epoch 6/10
83/83 [==============================] - 141s 2s/step - loss: 0.4112 - auroc: 0.7319 - \

Epoch 00006: val_auroc improved from 0.72383 to 0.72614, saving model to model2_1.h5
Epoch 7/10
83/83 [==============================] - 140s 2s/step - loss: 0.4034 - auroc: 0.7365 - \

Epoch 00007: val_auroc improved from 0.72614 to 0.73078, saving model to model2_1.h5
Epoch 8/10
83/83 [==============================] - 144s 2s/step - loss: 0.3974 - auroc: 0.7411 - \

Epoch 00008: val_auroc improved from 0.73078 to 0.73309, saving model to model2_1.h5
Epoch 9/10
83/83 [==============================] - 145s 2s/step - loss: 0.3927 - auroc: 0.7454 - \

Epoch 00009: val_auroc improved from 0.73309 to 0.73313, saving model to model2_1.h5
Epoch 10/10
83/83 [==============================] - 144s 2s/step - loss: 0.3893 - auroc: 0.7504 - \

Epoch 00010: val_auroc did not improve from 0.73313
```

plot epoch vs loss

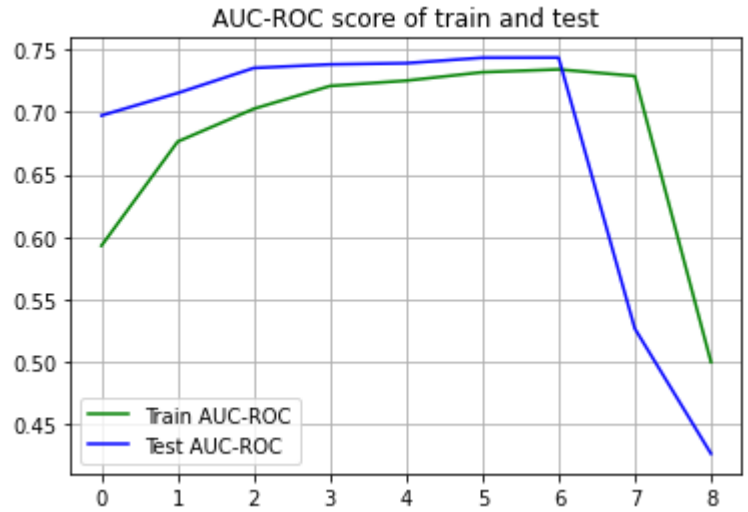
```
%matplotlib inline
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
x = list(range(1,11))
vy = history2_1.history['val_loss']
ty = history2_1.history['loss']
plt_dynamic(x, vy, ty, ax)
```



auc-roc score


```
# Evaluating test data
score_1 = model2_1.evaluate(test_data, y_test, verbose = 1, batch_size = 850)
print('Test Loss:', score_1[0])
print('Test ROC-AUC score:', score_1[1], '\n')

# Plotting train and test auc roc score
plt.plot(history1.history['auroc'], 'g')
plt.plot(history1.history['val_auroc'], 'b')
plt.title("AUC-ROC score of train and test")
plt.legend({'Train AUC-ROC': 'g', 'Test AUC-ROC':'b'})
plt.grid()
plt.show()
```

```
26/26 [==============================] - 8s 312ms/step - loss: 0.4093 - auroc: 0.7426
Test Loss: 0.40928637981414795
Test ROC-AUC score: 0.7425810694694519
```



Task-3:

- **input_seq_total_text_data**:

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

. 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 outp

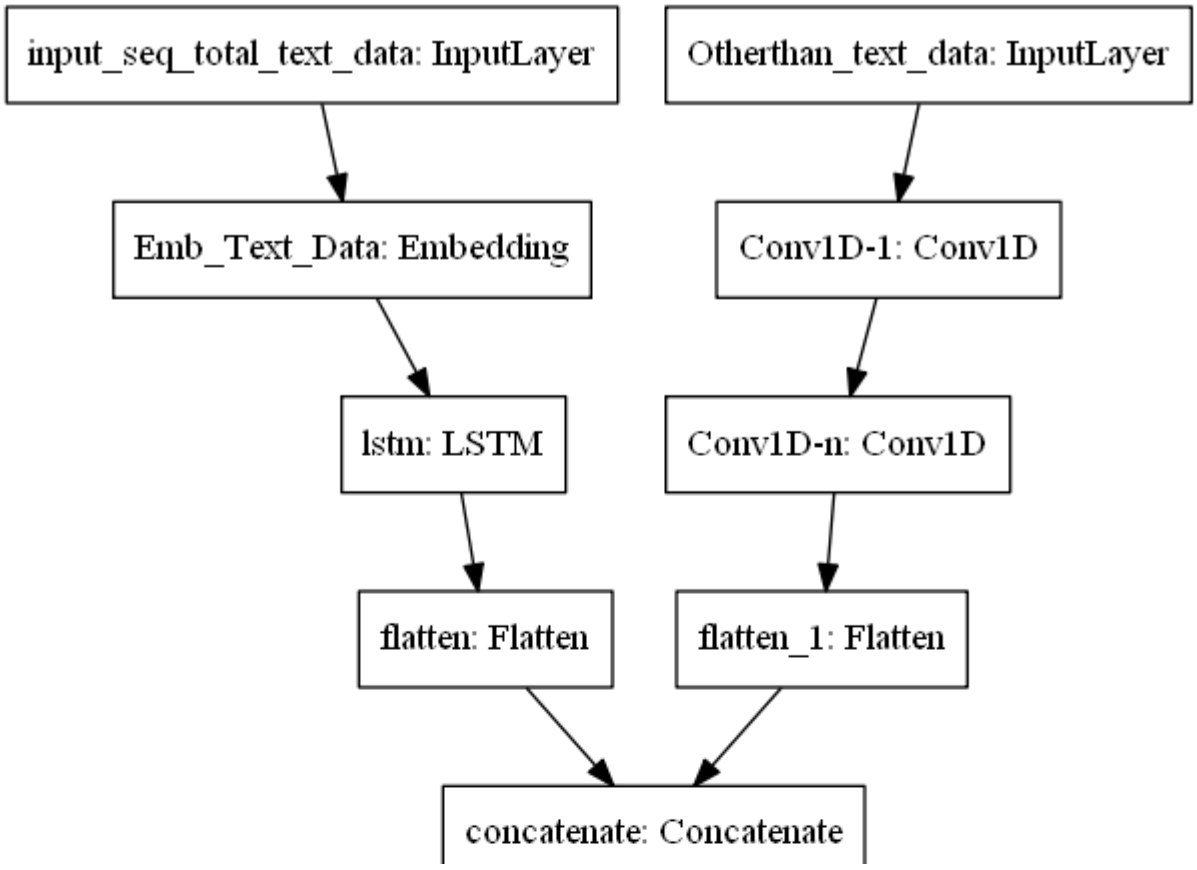
. You are free to preprocess the input text as you needed.

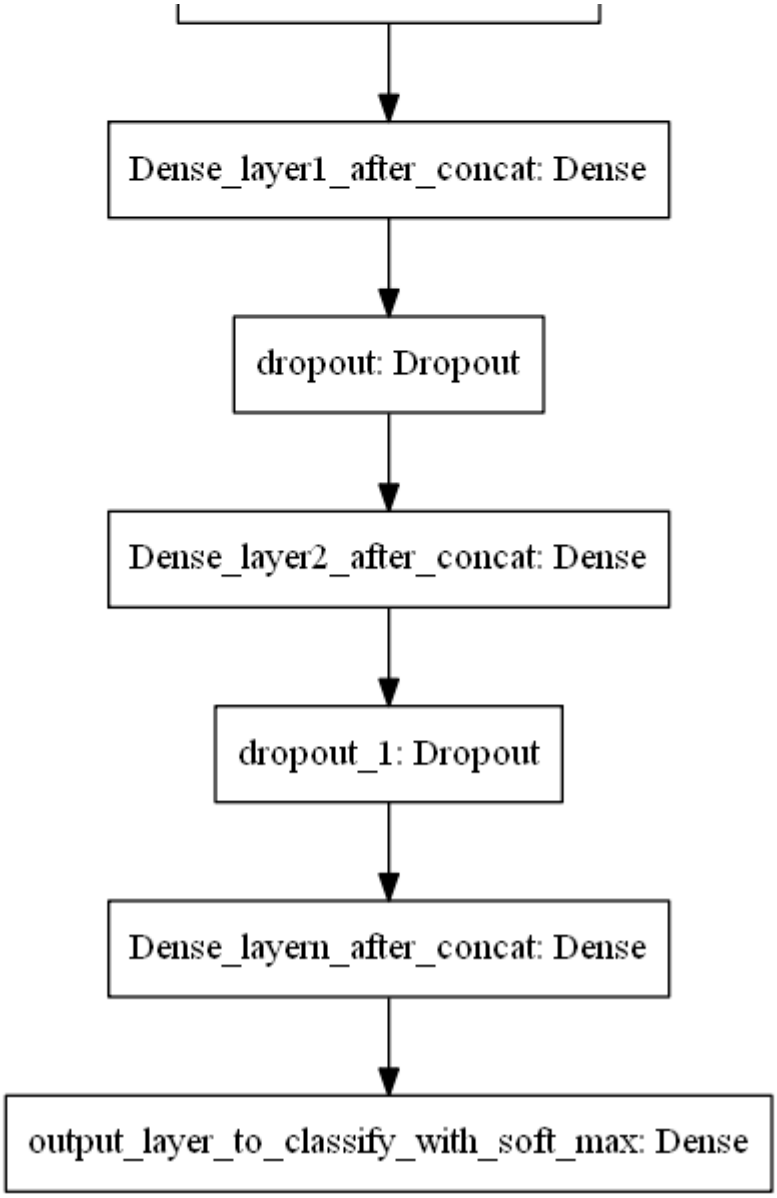
- **Other_than_text_data**:

. Convert all your Categorical values to onehot coded and then concatenat

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

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





ref:

https://i.imgur.com/fkQ8nGo.png

```
x_train.columns
```

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

TEXT DATA

1) school_state

```
# Convert all your Categorical values to onehot coded and then concatenate all these onehot v
# Neumerical values
```

```
# One hot encoding of Categorical Feature
# - school_state : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(x_train['school_state'].values)# Training

x_train_school_state_ohe = vectorizer.transform(x_train['school_state'].values)
x_cv_school_state_ohe = vectorizer.transform(x_cv['school_state'].values)
x_test_school_state_ohe = vectorizer.transform(x_test['school_state'].values)

school_state_features = vectorizer.get_feature_names()
print(vectorizer.get_feature_names())
print('*'*100)
print(x_train_school_state_ohe.shape, y_train.shape)
print(x_cv_school_state_ohe.shape, y_cv.shape)
print(x_test_school_state_ohe.shape, y_test.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il
****************************************************************************************
(69918, 51) (69918, 2)
(17480, 51) (17480, 2)
(21850, 51) (21850, 2)
```

2) teacher_prefix

```
# Convert all your Categorical values to onehot coded and then concatenate all these onehot v
# Neumerical values

# One hot encoding of Categorical Feature
# - teacher_prefix : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(x_train['teacher_prefix'].values)# Training

x_train_teacher_prefix_ohe = vectorizer.transform(x_train['teacher_prefix'].values)
x_cv_teacher_prefix_ohe = vectorizer.transform(x_cv['teacher_prefix'].values)
x_test_teacher_prefix_ohe = vectorizer.transform(x_test['teacher_prefix'].values)

teacher_prefix_features = vectorizer.get_feature_names()
print(vectorizer.get_feature_names())
print('*'*100)
print(x_train_teacher_prefix_ohe.shape, y_train.shape)
print(x_cv_teacher_prefix_ohe.shape, y_cv.shape)
print(x_test_teacher_prefix_ohe.shape, y_test.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
****************************************************************************************
(69918, 5) (69918, 2)
(17480, 5) (17480, 2)
(21850, 5) (21850, 2)
```

3) project_grade_category

```
# Convert all your Categorical values to onehot coded and then concatenate all these onehot v
# Neumerical values

# One hot encoding of Categorical Feature
# - project_grade_category : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(x_train['project_grade_category'].values)# Training

x_train_project_grade_category_ohe = vectorizer.transform(x_train['project_grade_category'].v
x_cv_project_grade_category_ohe = vectorizer.transform(x_cv['project_grade_category'].values)
x_test_project_grade_category_ohe = vectorizer.transform(x_test['project_grade_category'].val

project_grade_category_features = vectorizer.get_feature_names()
print(vectorizer.get_feature_names())
print('*'*100)
print(x_train_project_grade_category_ohe.shape, y_train.shape)
print(x_cv_project_grade_category_ohe.shape, y_cv.shape)
print(x_test_project_grade_category_ohe.shape, y_test.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
****************************************************************************************
(69918, 4) (69918, 2)
(17480, 4) (17480, 2)
(21850, 4) (21850, 2)
```

4) clean_categories

```
# Convert all your Categorical values to onehot coded and then concatenate all these onehot v
# Neumerical values

# One hot encoding of Categorical Feature
# - clean_categories : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(x_train['clean_categories'].values)# Training

x_train_clean_categories_ohe = vectorizer.transform(x_train['clean_categories'].values)
x_cv_clean_categories_ohe = vectorizer.transform(x_cv['clean_categories'].values)
x_test_clean_categories_ohe = vectorizer.transform(x_test['clean_categories'].values)

project_grade_category_features = vectorizer.get_feature_names()
print(vectorizer.get_feature_names())
print('*'*100)
print(x_train_clean_categories_ohe.shape, y_train.shape)
```

```
print(x_cv_clean_categories_ohe.shape, y_cv.shape)
print(x_test_clean_categories_ohe.shape, y_test.shape)
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language
**********************************************************************************
(69918, 9) (69918, 2)
(17480, 9) (17480, 2)
(21850, 9) (21850, 2)
```

5) clean_subcategories

```
# Convert all your Categorical values to onehot coded and then concatenate all these onehot v
# Neumerical values

# One hot encoding of Categorical Feature
# - clean_subcategories : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(x_train['clean_subcategories'].values)# Training

x_train_clean_subcategories_ohe = vectorizer.transform(x_train['clean_subcategories'].values)
x_cv_clean_subcategories_ohe = vectorizer.transform(x_cv['clean_subcategories'].values)
x_test_clean_subcategories_ohe = vectorizer.transform(x_test['clean_subcategories'].values)

project_grade_subcategory_features = vectorizer.get_feature_names()
print(vectorizer.get_feature_names())
print('*'*100)
print(x_train_clean_subcategories_ohe.shape, y_train.shape)
print(x_cv_clean_subcategories_ohe.shape, y_cv.shape)
print(x_test_clean_subcategories_ohe.shape, y_test.shape)
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_c
**********************************************************************************
(69918, 30) (69918, 2)
(17480, 30) (17480, 2)
(21850, 30) (21850, 2)
```

Numerical Data

1- price

```
# You no need to perform standardization/normalization on numerical data,
# because you will classify data by using gini impurity in decision tree classifier.
# - price,teacher_number_of_previously_posted_projects   : numerical

x_train_price_input_norm = x_train['price'].values.reshape(-1,1)
x_cv_price_input_norm = x_cv['price'].values.reshape(-1,1)
```

```
x_test_price_input_norm = x_test['price'].values.reshape(-1,1)

print("After vectorizations")
print("*"*100)
print(x_train_price_input_norm.shape, y_train.shape)
print(x_cv_price_input_norm.shape, y_cv.shape)
print(x_test_price_input_norm.shape, y_test.shape)
```

```
After vectorizations
****************************************************************************************************
(69918, 1) (69918, 2)
(17480, 1) (17480, 2)
(21850, 1) (21850, 2)
```

2 - teacher_number_of_previously_posted_projects

```
# You no need to perform standardization/normalization on numerical data,
# because you will classify data by using gini impurity in decision tree classifier.
# - price,teacher_number_of_previously_posted_projects   : numerical

x_train_teacher_number_of_previously_posted_projects_input_norm = x_train['teacher_number_of_
x_cv_teacher_number_of_previously_posted_projects_input_norm = x_cv['teacher_number_of_previo
x_test_teacher_number_of_previously_posted_projects_input_norm = x_test['teacher_number_of_pr

print("After vectorizations")
print("*"*100)
print(x_train_teacher_number_of_previously_posted_projects_input_norm.shape, y_train.shape)
print(x_cv_teacher_number_of_previously_posted_projects_input_norm.shape, y_cv.shape)
print(x_test_teacher_number_of_previously_posted_projects_input_norm.shape, y_test.shape)
```

```
After vectorizations
****************************************************************************************************
(69918, 1) (69918, 2)
(17480, 1) (17480, 2)
(21850, 1) (21850, 2)
```

Stacking one on top of the other in the form of a dense matrix

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
# X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
# X.shape

X_train_final = hstack((x_train_school_state_ohe, x_train_clean_categories_ohe,x_train_clean_

X_cv_final = hstack((x_cv_school_state_ohe, x_cv_clean_categories_ohe,x_cv_clean_subcategorie
```

```
X_test_final = hstack((x_test_school_state_ohe, x_test_clean_categories_ohe,x_test_clean_subc
```

```
print(X_train_final.shape)
print(X_cv_final.shape)
print(X_test_final.shape)
```

```
(69918, 101)
(17480, 101)
(21850, 101)
```

Adding new dimension as '2' as we want a 2 Dimensional output

```
train_final = X_train_final.todense()
cv_final = X_cv_final.todense()
test_final = X_test_final.todense()
```

```
X_train_final_new = np.resize(train_final,new_shape=(69918,101,1))
X_cv_final_new = np.resize(cv_final,new_shape=(17480,101,1))
X_test_final_new = np.resize(test_final,new_shape=(21850,101,1))
```

```
X_train_final_new.shape
```

```
(69918, 101, 1)
```

```
print("Final Train data shape",X_train_final_new.shape)
print("Final cv data shape",X_cv_final_new.shape)
print("Final Test data shape",X_test_final_new.shape)
```

```
Final Train data shape (69918, 101, 1)
Final cv data shape (17480, 101, 1)
Final Test data shape (21850, 101, 1)
```

```
from keras.utils import np_utils

#np_utils.to_categorical is used to convert array of labeled data(from 0 to nb_classes-1) to

y_train = np_utils.to_categorical(y_train, 2)
y_test = np_utils.to_categorical(y_test, 2)
y_cv = np_utils.to_categorical(y_cv, 2)
```

```
y_cv.shape
```

```
(17480, 2)
```

Keras Model-3

1) LSTM for TEXT DATA

2) CONV1D for OTHER THAN TEXT DATA

3) Dense Layer after Concatenating both

```
train_text_encode.shape[1]
```

```
400
```

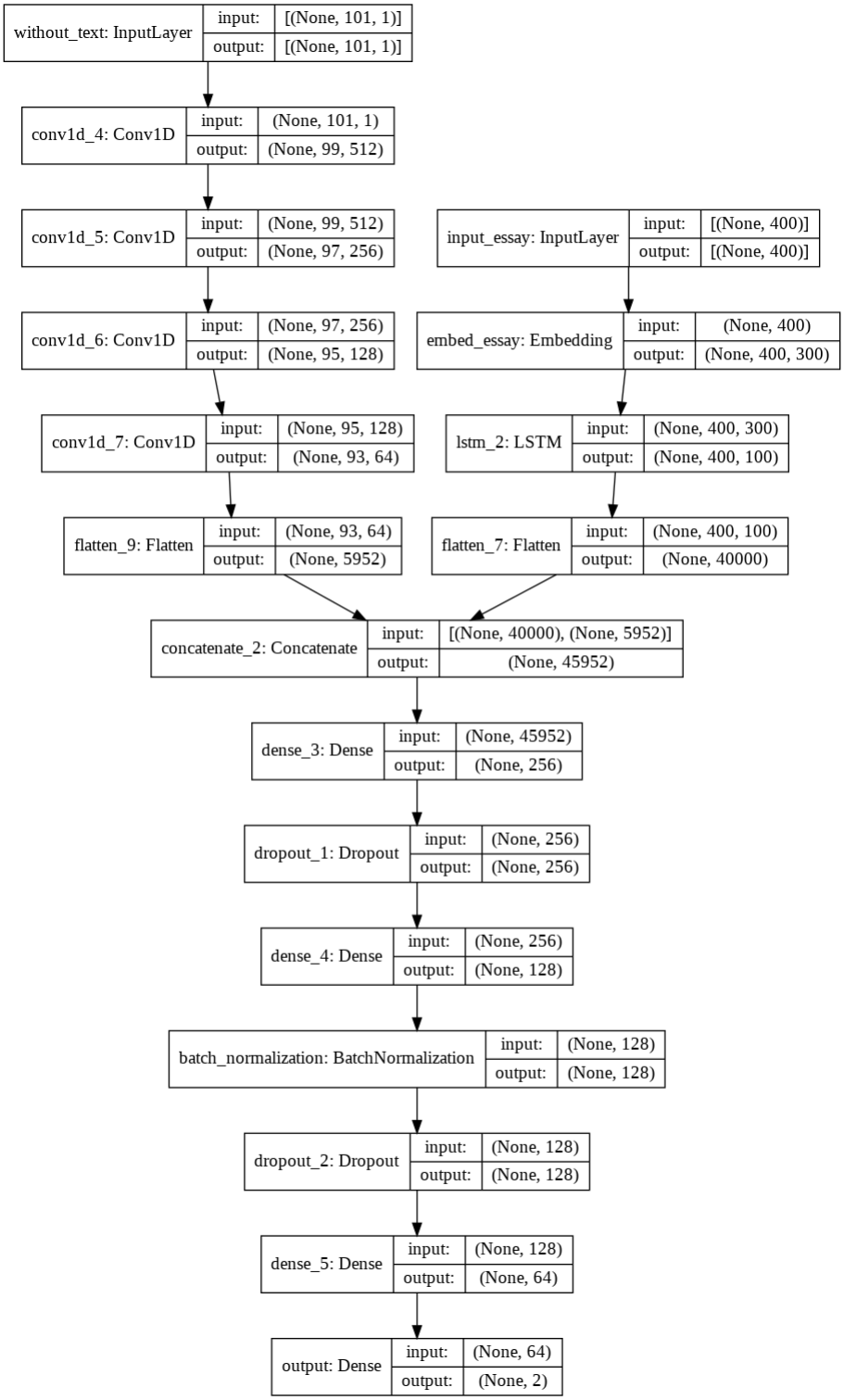
```
# Creating an input layer
input_essay = Input(shape=(train_text_encode.shape[1],), name='input_essay')
# Creating an embedding layer
x1 =Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1], weights=[embedding_
# Creating LSTM layer
x1 = LSTM(100,recurrent_dropout=0.5,kernel_regularizer=regularizers.l2(0.001),return_sequence

layer_1 = Flatten()(x1)


from keras.layers import MaxPooling1D

input_without_text  = Input(shape=(101,1), name='without_text')
convo = Conv1D(512 , 3 , activation='relu' ,  kernel_initializer=he_normal(seed=None) , paddi
convo = Conv1D(256 , 3 , activation='relu' ,  kernel_initializer=he_normal(seed=None) , paddi
convo = Conv1D(128 , 3 , activation='relu' ,  kernel_initializer=he_normal(seed=None) , paddi
convo = Conv1D(64 , 3 , activation='relu' ,  kernel_initializer=he_normal(seed=None) , paddin
flatten_without_text = Flatten()(convo)
x_concat = concatenate([layer_1 , flatten_without_text])

x = Dense(256, activation="relu", kernel_initializer="he_normal" ,kernel_regularizer=regulari
x=Dropout(0.5)(x)
x = Dense(128,activation="sigmoid",kernel_initializer="glorot_normal" ,kernel_regularizer=reg
x = BatchNormalization()(x)
x=Dropout(0.5)(x)
x = Dense(64,activation="relu", kernel_initializer="he_normal" ,kernel_regularizer=regularize
output = Dense(2, activation='softmax', name='output')(x)
model_3 = Model(inputs=[input_essay, input_without_text],outputs=[output])

print(model_3.summary())
```

```
Model: "model"
__________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==========================================================================================
without_text (InputLayer)    [(None, 101, 1)]          0
__________________________________________________________________________________________
conv1d_4 (Conv1D)            (None, 99, 512)           2048        without_text[0][0]
```

```
input_essay (InputLayer)        [(None, 400)]        0
conv1d_5 (Conv1D)               (None, 97, 256)      393472      conv1d_4[0][0]
embed_essay (Embedding)         (None, 400, 300)     7063200     input_essay[0][0]
conv1d_6 (Conv1D)               (None, 95, 128)      98432       conv1d_5[0][0]
lstm_2 (LSTM)                   (None, 400, 100)     160400      embed_essay[0][0]
conv1d_7 (Conv1D)               (None, 93, 64)       24640       conv1d_6[0][0]
flatten_7 (Flatten)             (None, 40000)        0           lstm_2[0][0]
flatten_9 (Flatten)             (None, 5952)         0           conv1d_7[0][0]
concatenate_2 (Concatenate)     (None, 45952)        0           flatten_7[0][0]
                                                                 flatten_9[0][0]
dense_3 (Dense)                 (None, 256)          11763968    concatenate_2[0][0]
dropout_1 (Dropout)             (None, 256)          0           dense_3[0][0]
dense_4 (Dense)                 (None, 128)          32896       dropout_1[0][0]
batch_normalization (BatchNorma (None, 128)          512         dense_4[0][0]
dropout_2 (Dropout)             (None, 128)          0           batch_normalization[0]
dense_5 (Dense)                 (None, 64)           8256        dropout_2[0][0]
output (Dense)                  (None, 2)            130         dense_5[0][0]
==================================================================================
Total params: 19,547,954
Trainable params: 12,484,498
Non-trainable params: 7,063,456
__________________________________________________________________________________
None
```

Network Artitecture

```
# https://github.com/mmortazavi/EntityEmbedding-Working_Example/blob/master/EntityEmbedding.i

import pydot_ng as pydot
from keras.utils.vis_utils import plot_model
from IPython.display import Image

plot_model(model_3, show_shapes = True, show_layer_names = True, to_file = 'model_3.png')

Image(retina = True, filename = 'model_3.png')
```




```
train_text_encode.shape
```

```
(69918, 400)
```

getting into one list

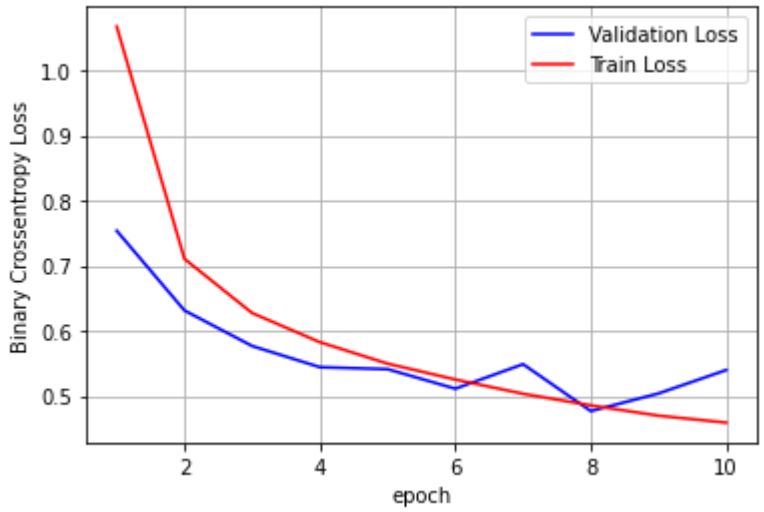
```
train_data_3 = [train_text_encode,X_train_final_new]
cv data 3 = [val text encode,X cv final new]
```

```
test_data_3 = [test_text_encode,X_test_final_new]


checkpoint3_1 = ModelCheckpoint("model_3.h5",
                             monitor="val_auroc",
                             mode="max",
                             save_best_only = True,
                             verbose=1)
earlystop3_1 = EarlyStopping(monitor = 'val_auroc',
                            mode="max",
                            min_delta = 0,
                            patience = 3,
                            verbose = 1)
tensorboard3_1 = TensorBoard(log_dir='Model3_visualization')

callbacks_3_1 = [checkpoint3_1,earlystop3_1,tensorboard3_1]


from tensorflow.keras import optimizers
model_3.compile(optimizer=optimizers.Adam(0.0005, decay=1e-6),loss='binary_crossentropy', met

history_3= model_3.fit(train_data_3, y_train, batch_size=850, epochs=10, verbose=1,callbacks=

     Epoch 1/10
     83/83 [==============================] - 1362s 16s/step - loss: 1.0672 - auroc: 0.5316 -

     Epoch 00001: val_auroc improved from -inf to 0.61596, saving model to model_3.h5
     Epoch 2/10
     83/83 [==============================] - 1291s 16s/step - loss: 0.7104 - auroc: 0.5874 -

     Epoch 00002: val_auroc improved from 0.61596 to 0.68146, saving model to model_3.h5
     Epoch 3/10
     83/83 [==============================] - 1328s 16s/step - loss: 0.6272 - auroc: 0.6512 -

     Epoch 00003: val_auroc improved from 0.68146 to 0.71602, saving model to model_3.h5
     Epoch 4/10
     83/83 [==============================] - 1325s 16s/step - loss: 0.5824 - auroc: 0.6857 -

     Epoch 00004: val_auroc improved from 0.71602 to 0.72266, saving model to model_3.h5
     Epoch 5/10
     83/83 [==============================] - 1277s 15s/step - loss: 0.5495 - auroc: 0.7048 -

     Epoch 00005: val_auroc improved from 0.72266 to 0.72630, saving model to model_3.h5
     Epoch 6/10
     83/83 [==============================] - 1350s 16s/step - loss: 0.5251 - auroc: 0.7146 -

     Epoch 00006: val_auroc improved from 0.72630 to 0.73047, saving model to model_3.h5
     Epoch 7/10
     83/83 [==============================] - 1339s 16s/step - loss: 0.5032 - auroc: 0.7233 -

     Epoch 00007: val_auroc improved from 0.73047 to 0.73605, saving model to model_3.h5
     Epoch 8/10
     83/83 [==============================] - 1297s 16s/step - loss: 0.4854 - auroc: 0.7298 -

     Epoch 00008: val_auroc improved from 0.73605 to 0.73846, saving model to model_3.h5
```

```
Epoch 9/10
83/83 [==============================] - 1347s 16s/step - loss: 0.4696 - auroc: 0.7358 -

Epoch 00009: val_auroc improved from 0.73846 to 0.74130, saving model to model_3.h5
Epoch 10/10
83/83 [==============================] - 1360s 16s/step - loss: 0.4589 - auroc: 0.7389 -

Epoch 00010: val_auroc did not improve from 0.74130
```

Plot epoch vs loss

```
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()


%matplotlib inline
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
x = list(range(1,11))
vy = history_3.history['val_loss']
ty = history_3.history['loss']
plt_dynamic(x, vy, ty, ax)
```

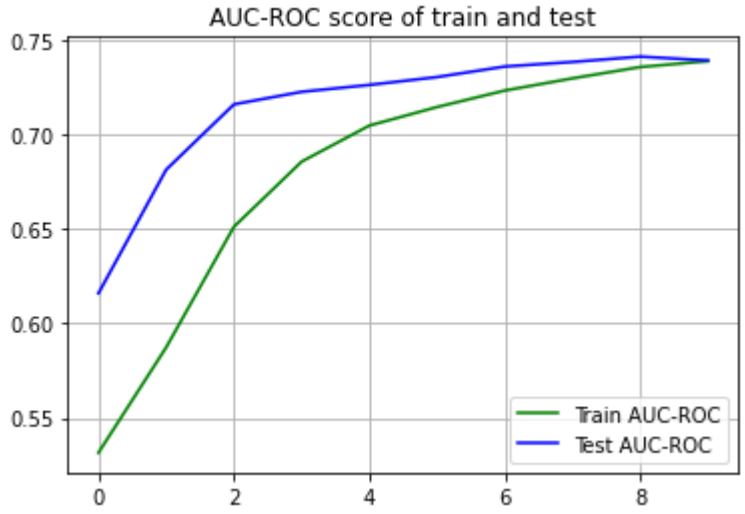


auc-roc score

```
# Evaluating test data
score_1 = model_3.evaluate(test_data_3, y_test, verbose = 1, batch_size = 850)
print('Test Loss:', score_1[0])
print('Test ROC-AUC score:', score_1[1], '\n')
```

```
# Plotting train and test auc roc score
plt.plot(history_3.history['auroc'], 'g')
plt.plot(history_3.history['val_auroc'], 'b')
plt.title("AUC-ROC score of train and test")
plt.legend({'Train AUC-ROC': 'g', 'Test AUC-ROC':'b'})
plt.grid()
plt.show()
```

```
26/26 [==============================] - 107s 4s/step - loss: 0.5370 - auroc: 0.7483
Test Loss: 0.5370233654975891
Test ROC-AUC score: 0.7483065724372864
```



Observation:

```
from prettytable import PrettyTable

a = PrettyTable()

a.field_names = ['S.No', 'Model', 'Optimizer','val_loss','val_auroc' ,'Test Loss', 'Test AUC-

a.add_row([1, 'Model- 1.1', 'adam', 0.4541,0.7538,0.4524, 0.7566])
a.add_row([1, 'Model- 1.2', 'adam', 0.6848,0.6284,0.6632, 0.5064])
a.add_row([1, 'Model- 1.3', 'adam', 0.9218,0.7299,0.9231, 0.7278])
a.add_row([2, 'Model- 2.1', 'adam',0.4289,0.7517 ,0.4272, 0.7571])
a.add_row([2, 'Model- 2.2', 'adam',0.3893,0.7504 ,0.4481, 0.7434])
a.add_row([2, 'Model- 2.3', 'adam',0.4118,0.7327 ,0.4093, 0.7426])
a.add_row([3, 'Model- 3', 'adam', 0.4651 ,07413.,0.5370 ,0.7483])

print(a.get_string(title = "LSTM on Donors Result"))
```

```
+-------------------------------------------------------------------------------+
|                            LSTM on Donors Result                              |
+------+------------+-----------+----------+-----------+-----------+-------------+
| S.No |   Model    | Optimizer | val_loss | val_auroc | Test Loss | Test AUC-ROC |
```

1	Model- 1.1	adam	0.4541	0.7538	0.4524	0.7566
1	Model- 1.2	adam	0.6848	0.6284	0.6632	0.5064
1	Model- 1.3	adam	0.9218	0.7299	0.9231	0.7278
2	Model- 2.1	adam	0.4289	0.7517	0.4272	0.7571
2	Model- 2.2	adam	0.3893	0.7504	0.4481	0.7434
2	Model- 2.3	adam	0.4118	0.7327	0.4093	0.7426
3	Model- 3	adam	0.4651	7413.0	0.537	0.7483

refernce;

https://github.com/Dharaniraj1997/LSTM-on-DonorsChoose/blob/main/LSTM%20on%20DonorsChoose.ipynb

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

✓ 0s completed at 11:45 PM