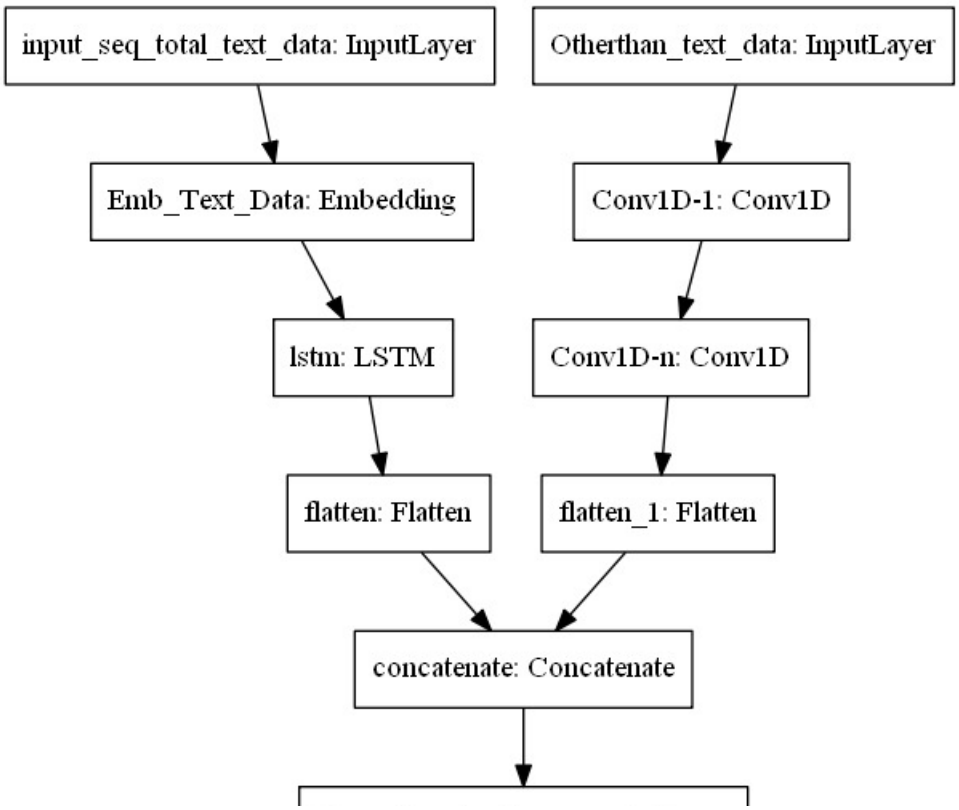
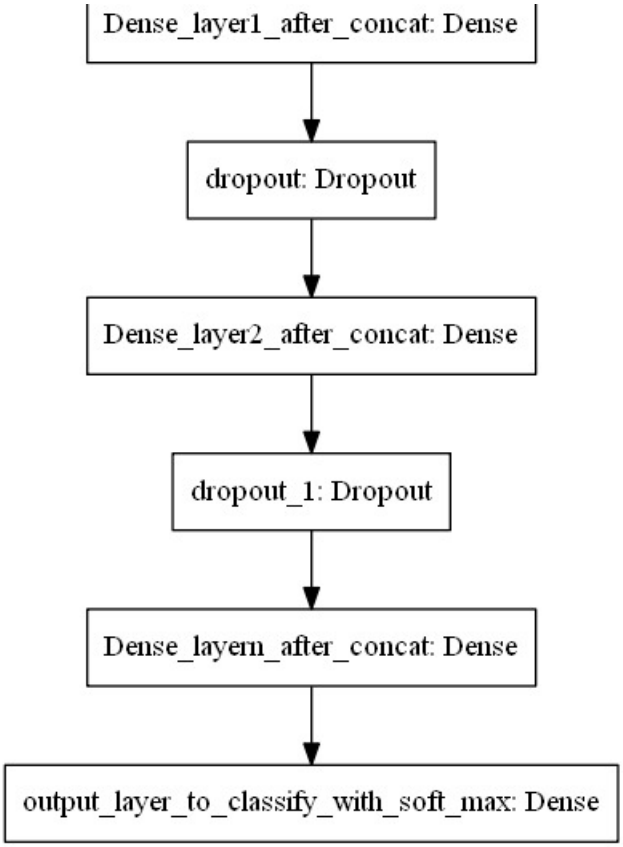
LSTM Assignment - 14 [Model - 3]

In [37]:

```
# Importing all necessary files
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from pickle import load,dump

from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer


import pickle
from tqdm import tqdm
import os
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

3. Model 3





In [2]:

```
# keras imports
from sklearn.preprocessing import StandardScaler
from keras.layers import BatchNormalization,Dense,Dropout,Input,Embedding,LSTM,Flatten
from keras.layers import Conv1D
from keras.models import Model,Sequential
from keras.layers.merge import concatenate
from keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras.callbacks import TensorBoard
```

```
Using TensorFlow backend.
```

In [3]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [4]:

```
project_data.isnull().sum()
```

Out[4]:

```
Unnamed: 0                                      0
id                                              0
teacher_id                                      0
teacher_prefix                                  3
school_state                                    0
project_submitted_datetime                      0
project_grade_category                          0
project_subject_categories                      0
project_subject_subcategories                   0
project_title                                   0
project_essay_1                                 0
project_essay_2                                 0
project_essay_3                            105490
project_essay_4                            105490
project_resource_summary                        0
teacher_number_of_previously_posted_projects    0
project_is_approved                             0
```

```
dtype: int64
```

In [5]:

```
#filling 3 null teacher prefix values with Teacher

project_data["teacher_prefix"].fillna("Teacher",inplace = True)
project_data.isnull().sum()
```

Out[5]:

```
Unnamed: 0                                          0
id                                                  0
teacher_id                                          0
teacher_prefix                                      0
school_state                                        0
project_submitted_datetime                          0
project_grade_category                              0
project_subject_categories                          0
project_subject_subcategories                       0
project_title                                       0
project_essay_1                                     0
project_essay_2                                     0
project_essay_3                                105490
project_essay_4                                105490
project_resource_summary                            0
teacher_number_of_previously_posted_projects        0
project_is_approved                                 0
dtype: int64
```

In [6]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [7]:

```
# comibining total text_data
project_data["combine"] = project_data["essay"] + project_data["project_title"]
```

In [8]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')

project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
Data columns (total 21 columns):
Unnamed: 0                                        109248 non-null int64
id                                                109248 non-null object
teacher_id                                        109248 non-null object
teacher_prefix                                    109248 non-null object
school_state                                      109248 non-null object
project_submitted_datetime                        109248 non-null object
project_grade_category                            109248 non-null object
project_subject_categories                        109248 non-null object
project_subject_subcategories                     109248 non-null object
project_title                                     109248 non-null object
project_essay_1                                   109248 non-null object
project_essay_2                                   109248 non-null object
project_essay_3                                   3758 non-null object
project_essay_4                                   3758 non-null object
project_resource_summary                          109248 non-null object
teacher_number_of_previously_posted_projects      109248 non-null int64
project_is_approved                               109248 non-null int64
essay                                             109248 non-null object
combine                                           109248 non-null object
```

```
price                                           109248 non-null float64
quantity                                        109248 non-null int64
dtypes: float64(1), int64(4), object(16)
memory usage: 18.3+ MB
```

In [9]:

```
project_data.columns
```

Out[9]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'essay', 'combine', 'price', 'quantity'],
      dtype='object')
```

In [10]:

```
from sklearn.utils import resample
p_d = resample(project_data)
```

In [11]:

```
#splitting data as 30% to test
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

y = p_d["project_is_approved"]
X = p_d.drop("project_is_approved",axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=42
)


print(X_train.shape," ",y_train.shape)
print(X_test.shape," ",y_test.shape)
print(X_val.shape," ",y_val.shape)
```

```
(65548, 20)   (65548,)
(21850, 20)   (21850,)
(21850, 20)   (21850,)
```

Preprocessing Text Data

In [12]:

```
#using function and stopwords form assignemnt

import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

# we are removing the words from the stop words list: 'no', 'nor', 'not'
```

```
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [14]:

```
from tqdm import tqdm

#for train data
preprocessed_combine = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['combine'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_combine.append(sent.lower().strip())

test_preprocessed_combine = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['combine'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    test_preprocessed_combine.append(sent.lower().strip())

val_preprocessed_combine = []
# tqdm is for printing the status bar
for sentance in tqdm(X_val['combine'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    val_preprocessed_combine.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 65548/65548 [01:59<00:00, 548.32it/s]
100%|████████████████████████████████████████████████████████████████| 21850/21850 [00:45<00:00, 481.36it/s]
100%|████████████████████████████████████████████████████████████████| 21850/21850 [00:
```

```
35<00:00, 608.34it/s]
```

Embeding Essay

Considering all the words.

In [15]:

```
from keras.preprocessing.text import Tokenizer

token = Tokenizer()

# fitting on train data.
token.fit_on_texts(preprocessed_essays)
```

Caution:

Tokenizer considers only top words provided by num_words while converting to sequences. i.e. if word is not present the it skips the word. But token.index_word keeps all the record of words in whole vocabulary.So, remoove unnecessary words while using token.index_words as a dictionary.

In [16]:

```
# index to words are assigned according to frequency. i.e the most frequent word has index of 1
ix_to_word = token.index_word
len(ix_to_word)
```

Out[16]:

```
15359
```

In [18]:

```
# for k in list(ix_to_word):
#     if k>=MAX_WORDS:
#         ix_to_word.pop(k, None)

word_to_ix = dict()
for k,v in ix_to_word.items():
    word_to_ix[v] = k

print(len(word_to_ix))
print(len(ix_to_word))
```

```
15359
15359
```

In [19]:

```
combine_sequence = token.texts_to_sequences(preprocessed_combine)
test_combine_sequence = token.texts_to_sequences(test_preprocessed_combine)
val_combine_sequence = token.texts_to_sequences(val_preprocessed_combine)

print(len(combine_sequence))
print(len(test_combine_sequence))
print(len(val_combine_sequence))
```

```
65548
21850
21850
```

In [20]:

```
combine_sequence = pad_sequences(combine_sequence,padding="post")
test_combine_sequence = pad_sequences(test_combine_sequence,maxlen=combine_sequence.shape[1],paddin
g="post")
val_combine_sequence = pad_sequences(val_combine_sequence,maxlen=combine_sequence.shape[1],padding
="post")
```

In [21]:

```
print(combine_sequence.shape)
print(test_combine_sequence.shape)
print(val_combine_sequence.shape)
```

```
(65548, 333)
(21850, 333)
(21850, 333)
```

In [30]:

```
ix_to_word[0] = "start_seq"
word_to_ix["start_seq"] = 0
```

In [31]:

```
MAX_LENGTH = combine_sequence.shape[1]
MAX_WORDS = len(word_to_ix)
print("Maximum sequence length is {}".format(MAX_LENGTH))
print(combine_sequence.shape)
```

```
Maximum sequence length is 333
(65548, 333)
```

In [32]:

```
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    glove = load(f)
    glove_words =  set(glove.keys())
```

In [33]:

```
EMBEDDING_SIZE = 300
VOCAB_SIZE = MAX_WORDS

# Get 300-dim dense vector for each of the words in vocabulary
embedding_matrix = np.zeros((VOCAB_SIZE,EMBEDDING_SIZE))
embedding_matrix.shape
```

Out[33]:

```
(15360, 300)
```

In [34]:

```
# code for embedding matrix. considering top 5000 words and using already present glove vectors

# Get 300-dim dense vector for each of the words in vocabulary
embedding_matrix = np.zeros(((VOCAB_SIZE),EMBEDDING_SIZE))

for word, i in word_to_ix.items():
    embedding_vector = np.zeros(300)
    if word in glove_words:
        embedding_vector = glove[word]
        embedding_matrix[i] = embedding_vector
    else:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector
```

In [35]:

```
# save the embedding matrix to file
with open("embedding_matrix_3.pkl","wb") as f:
    dump(embedding_matrix,f)
```

Functional API for Esay

In [36]:

```
# functional api for essay
LSTM_units = 16
input_ess = Input(shape=(MAX_LENGTH,))
em1 = Embedding(MAX_WORDS,EMBEDDING_SIZE,input_length=MAX_LENGTH)(input_ess)
lstm = LSTM(LSTM_units,input_shape = (1,MAX_LENGTH), return_sequences=True,)(em1)
flt_ess  = Flatten()(lstm)
```

```
WARNING:tensorflow:From C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

One hot encoded categorical features

In [38]:

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

project categories

In [39]:

```
catogories = list(X_train['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_train['clean_categories'] = cat_list
X_train.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in X_train['clean_categories'].values:
    my_counter.update(word.split())

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




# project subject categories for test data

catogories = list(X_test['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_test['clean_categories'] = cat_list
X_test.drop(['project_subject_categories'], axis=1, inplace=True)
```

```
# project subject categories for val data

catogories = list(X_val['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_val['clean_categories'] = cat_list
X_val.drop(['project_subject_categories'], axis=1, inplace=True)
```

In [40]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)

# fitting on train data
vectorizer.fit(X_train['clean_categories'].values)
print(vectorizer.get_feature_names())

# for train data
categories_one_hot = vectorizer.transform(X_train['clean_categories'].values)

# for val data
val_categories_one_hot = vectorizer.transform(X_val['clean_categories'].values)

# for test data
test_categories_one_hot = vectorizer.transform(X_test['clean_categories'].values)


print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (65548, 9)
```

project subject sub_categories

In [41]:

```
sub_catogories = list(X_train['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_train['clean_subcategories'] = sub_cat_list
X_train.drop(['project_subject_subcategories'], axis=1, inplace=True)

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

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

```
sub_catogories = list(X_test['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_test['clean_subcategories'] = sub_cat_list
X_test.drop(['project_subject_subcategories'], axis=1, inplace=True)

sub_catogories = list(X_val['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_val['clean_subcategories'] = sub_cat_list
X_val.drop(['project_subject_subcategories'], axis=1, inplace=True)
```

In [42]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)

# fitting on train data
vectorizer.fit(X_train['clean_subcategories'].values)
print(vectorizer.get_feature_names())

# for train data
sub_categories_one_hot = vectorizer.transform(X_train['clean_subcategories'].values)
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)

# for val data
val_sub_categories_one_hot = vectorizer.transform(X_val['clean_subcategories'].values)


# for test data
test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subcategories'].values)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'Warmth', 'Care_Hunger', 'NutritionEducation',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'EarlyDevelopment', 'ESL',
'Health_LifeScience', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness',
'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (65548, 30)
```

Teacher Prefix

In [43]:

```
#preprocessing teacher prefix
prefix = list(X_train['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
```

```
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_train['clean_prefix'] = prefix_list

my_counter = Counter()
for word in X_train['clean_prefix'].values:
  my_counter.update(word.split())

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


#preprocessing teacher prefix for test data
prefix = list(X_test['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_test['clean_prefix'] = prefix_list

#preprocessing teacher prefix for val data
prefix = list(X_val['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_val['clean_prefix'] = prefix_list
```

```
{'Dr': 8, 'Teacher': 1425, 'Mr': 6396, 'Ms': 23527, 'Mrs': 34192}
```

In [44]:

```
vectorizer = CountVectorizer(vocabulary=list(prefix_dict.keys()), lowercase=False, binary=True)

# fitting on train data
vectorizer.fit(X_train['clean_prefix'].values)
print(vectorizer.get_feature_names())

# for train data
prefix_one_hot = vectorizer.transform(X_train['clean_prefix'].values)
print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

# for val data
val_prefix_one_hot = vectorizer.transform(X_val['clean_prefix'].values)

# for test data
test_prefix_one_hot = vectorizer.transform(X_test['clean_prefix'].values)
```

```
['Mr', 'Mrs', 'Ms', 'Teacher', 'Dr']
Shape of matrix after one hot encodig  (65548, 5)
```

Grade Category

In [45]:

```
grade = list(X_train['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
```

```
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_train['clean_grade'] = grade_list

my_counter = Counter()
for word in X_train['clean_grade'].values:
  my_counter.update(word.split())

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



# preprocessing of grade category for test data

grade = list(X_test['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_test['clean_grade'] = grade_list

# preprocessing of grade category for val data

grade = list(X_val['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_val['clean_grade'] = grade_list
```

```
{'nine_twelve': 6495, 'six_eight': 10356, 'three_five': 22317, 'prek_two': 26380}
```

In [46]:

```
vectorizer = CountVectorizer(vocabulary=list(grade_dict.keys()), lowercase=False, binary=True)

# fitting on train data
vectorizer.fit(X_train['clean_grade'].values)
print(vectorizer.get_feature_names())
```

```
# for train data
grade_one_hot = vectorizer.transform(X_train['clean_grade'].values)
print("Shape of matrix after one hot encodig ",grade_one_hot.shape)

# for val data
val_grade_one_hot = vectorizer.transform(X_val['clean_grade'].values)

# for test data
test_grade_one_hot = vectorizer.transform(X_test['clean_grade'].values)
```

```
['six_eight', 'prek_two', 'nine_twelve', 'three_five']
Shape of matrix after one hot encodig  (65548, 4)
```

School State

In [47]:

```
#no need of preprocessing on school state

state = X_train["school_state"].value_counts()
sorted_state = dict(state)
sorted_state_dict = dict(sorted(sorted_state.items(), key=lambda kv: kv[1]))
X_train["clean_state"] = X_train["school_state"]
print(sorted_state_dict)

#similarly for X_test
X_test["clean_state"] = X_test["school_state"]

#similarly for X_val
X_val["clean_state"] = X_val["school_state"]
```

```
{'VT': 49, 'WY': 51, 'ND': 69, 'MT': 136, 'SD': 169, 'RI': 183, 'NE': 187, 'NH': 187, 'DE': 200,
'AK': 225, 'WV': 289, 'HI': 289, 'ME': 298, 'DC': 313, 'NM': 325, 'KS': 397, 'ID': 407, 'IA': 416,
'AR': 630, 'CO': 663, 'OR': 708, 'MN': 720, 'MS': 758, 'NV': 792, 'KY': 819, 'MD': 923, 'CT': 990,
'TN': 1008, 'UT': 1019, 'WI': 1039, 'AL': 1080, 'VA': 1206, 'WA': 1354, 'AZ': 1355, 'NJ': 1355, 'OK
': 1366, 'LA': 1443, 'MA': 1444, 'OH': 1445, 'MO': 1528, 'IN': 1639, 'MI': 1866, 'PA': 1954, 'SC':
2343, 'GA': 2430, 'IL': 2630, 'NC': 3068, 'FL': 3750, 'NY': 4316, 'TX': 4413, 'CA': 9304}
```

In [48]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['clean_state'].values)
print(vectorizer.get_feature_names())
state_one_hot = vectorizer.transform(X_train['clean_state'].values)

print("Shape of matrix after one hot encodig ",state_one_hot.shape)

# for val data
val_state_one_hot = vectorizer.transform(X_val['clean_state'].values)

test_state_one_hot = vectorizer.transform(X_test['clean_state'].values)
```

```
['VT', 'WY', 'ND', 'MT', 'SD', 'RI', 'NE', 'NH', 'DE', 'AK', 'WV', 'HI', 'ME', 'DC', 'NM', 'KS', 'I
D', 'IA', 'AR', 'CO', 'OR', 'MN', 'MS', 'NV', 'KY', 'MD', 'CT', 'TN', 'UT', 'WI', 'AL', 'VA', 'WA',
'AZ', 'NJ', 'OK', 'LA', 'MA', 'OH', 'MO', 'IN', 'MI', 'PA', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix after one hot encodig  (65548, 51)
```

Preprocessing Numerical Feature¶

In [49]:

```
from sklearn.preprocessing import StandardScaler


price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
```

```
#train data price standardization
price_standardized = price_scalar.transform(X_train['price'].values.reshape(-1, 1))

#val data price stanardization. Fit method applied on X_train
val_price_standardized = price_scalar.transform(X_val['price'].values.reshape(-1, 1))

#test data price stanardization. Fit method applied on X_train
test_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [50]:

```
warnings.filterwarnings("ignore")
price_scalar = StandardScaler()
price_scalar.fit(X_train["quantity"].values.reshape(-1, 1))
print(f"Mean of Quantity : {price_scalar.mean_[0]}, Standard deviation of Quantity :
{np.sqrt(price_scalar.var_[0])}")

#train data quantity standardization
quantity_standardized = price_scalar.transform(X_train["quantity"].values.reshape(-1, 1))

#val data quantity stanardization. Fit method applied on X_train
val_quantity_standardized = price_scalar.transform(X_val["quantity"].values.reshape(-1, 1))


#test data quantity stanardization. Fit method applied on X_train
test_quantity_standardized = price_scalar.transform(X_test["quantity"].values.reshape(-1, 1))
```

```
Mean of Quantity : 16.890995911393176, Standard deviation of Quantity : 25.737157296905973
```

In [51]:

```
price_scalar = StandardScaler()
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

#train data ppp standardization
number_ppp_standardized =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))

#val data price stanardization. Fit method applied on X_train
val_number_ppp_standardized =
price_scalar.transform(X_val['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))

#test data price stanardization. Fit method applied on X_train
test_number_ppp_standardized =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
```

```
Mean : 11.263287972173064, Standard deviation : 27.829998026252415
```

In [52]:

```
# concatinating features as below
# category + sub_category + teacher_prefix + school_state + grade + price + quantity + number_ppp
from scipy.sparse import hstack

other_than =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,state_one_hot,grade_one_hot,price_
standardized,quantity_standardized,number_ppp_standardized))
val_other_than =
hstack((val_categories_one_hot,val_sub_categories_one_hot,val_prefix_one_hot,val_state_one_hot,val
_grade_one_hot,val_price_standardized,val_quantity_standardized,val_number_ppp_standardized))
test_other_than = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_prefix_one_hot,
test_state_one_hot,test_grade_one_hot,test_price_standardized,test_quantity_standardized,test_numbe
r_ppp_standardized))

other_than = np.expand_dims(other_than.toarray(),axis=2)
test_other_than = np.expand_dims(test_other_than.toarray(),axis=2)
```

```
val_other_than = np.expand_dims(val_other_than.toarray(),axis=2)

print("Shape of other than features {}".format(other_than.shape))
print("Shape of val other than features {}".format(val_other_than.shape))
print("Shape of test other than features {}".format(test_other_than.shape))
```

```
Shape of other than features (65548, 102, 1)
Shape of val other than features (21850, 102, 1)
Shape of test other than features (21850, 102, 1)
```

Model Preperation

In [53]:

```
# other_than functional layer

# please refer for i/p shape of conv1D https://stackoverflow.com/questions/43396572/dimension-of-shape-in-conv1d/43399308

IN_SHAPE = other_than.shape[1:]
input2 = Input(shape=IN_SHAPE)

# using 64 kernels of size 1x1
con1 = Conv1D(8,5,activation="relu")(input2)

# using 20 kernels of size 1x1
con2 = Conv1D(4,3,activation="relu")(con1)
flt2 = Flatten()(con2)
```

In [54]:

```
# concatenating essay_ip + other_ip
cnt = concatenate([flt_ess,flt2])
dense = Dense(8,activation="relu")(cnt)
dp = Dropout(0.4)(dense)
dense2 = Dense(4,activation="relu")(dp)
dp = Dropout(0.4)(dense2)
dense3 = Dense(2)(dp)
output_1 = Dense(1,activation="sigmoid")(dense3)
```

```
WARNING:tensorflow:From C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

Model Preperation

In [56]:

```
model3 = Model(inputs = [input_ess,input2],outputs = output_1)

# Freezing essay embedding_layer from training

model3.layers[2].set_weights([embedding_matrix])
model3.layers[2].trainable = False
```

In [57]:

```
model3.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         (None, 333)               0
_________________________________________________________________________________________________
input_2 (InputLayer)         (None, 102, 1)            0
_________________________________________________________________________________________________
embedding_1 (Embedding)      (None, 333, 300)          4608000     input_1[0][0]
```

```
conv1d_1 (Conv1D)               (None, 98, 8)         48          input_2[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 333, 16)       20288       embedding_1[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 96, 4)         100         conv1d_1[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 5328)          0           lstm_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 384)           0           conv1d_2[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 5712)          0           flatten_1[0][0]
                                                                  flatten_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 8)             45704       concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 8)             0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 4)             36          dropout_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 4)             0           dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 2)             10          dropout_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1)             3           dense_3[0][0]
==================================================================================================
Total params: 4,674,189
Trainable params: 66,189
Non-trainable params: 4,608,000
__________________________________________________________________________________________________
```

In [229]:

```
# please refer https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8
# data generator, intended to be used in a call to model.fit_generator()
from numpy import array

def data_generator(df,batch_size,data_type = 'Train'):
    X1, X2, y = list(), list(), list()
    flag = True
    if data_type == 'Val':
        flag = False
    n=0
    # loop for ever over images
    while 1:
        for i in range(len(df)):
            n+=1
            if flag:
                X1.append(combine_sequence[i])
                X2.append(other_than[i])
                y.append(df.iloc[i])
            else:
                X1.append(val_combine_sequence[i])
                X2.append(val_other_than[i])
                y.append(df.iloc[i])
            if n==batch_size:
                yield [[array(X1), array(X2)],array(y)]
                X1, X2,y = list(), list(), list()
                n=0
```

In [230]:

```
epochs = 25
batch_size = 64
steps = len(y_train)//batch_size
val_steps = len(y_val)//32
```

In [231]:

```
# https://datascience.stackexchange.com/questions/35775/how-to-find-auc-metric-value-for-keras-model
from sklearn import metrics
from keras import backend as K
```

```
from sklearn.metrics import roc_auc_score
import tensorflow as tf

# https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-keras

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

def auc(y_true, y_pred):
    auc = tf.metrics.auc(y_true, y_pred)[1]
    K.get_session().run(tf.local_variables_initializer())
    return auc
```

In [232]:

```
# using tensorboard instance for callbacks
from time import time
from datetime import datetime

tensorboard = TensorBoard(log_dir="model3_logs/{}".format(time()))

# model compilation
model3.compile(loss='binary_crossentropy', optimizer='adam',metrics=[auroc])
```

In [233]:

```
for i in range(epochs):
    print("Epoch {} start at time ".format(i),datetime.now())
    generator = data_generator(y_train,batch_size)
    val_generator = data_generator(y_val,batch_size,"Val")
    model3.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=2,callbacks=[tensorboard],validation_data=val_generator,validation_steps=val_steps)
    model3.save_weights("model3_1_epoch_{}.h5".format(i))
```

```
Epoch 0 start at time  2019-07-03 00:00:38.096461
Epoch 1/1
 - 406s - loss: 0.4613 - auroc: 0.5346 - val_loss: 0.4155 - val_auroc: 0.7068
Epoch 1 start at time  2019-07-03 00:07:34.744413
Epoch 1/1
 - 401s - loss: 0.4165 - auroc: 0.5547 - val_loss: 0.4118 - val_auroc: 0.7211
Epoch 2 start at time  2019-07-03 00:14:19.651700
Epoch 1/1
 - 399s - loss: 0.4145 - auroc: 0.5471 - val_loss: 0.4112 - val_auroc: 0.7056
Epoch 3 start at time  2019-07-03 00:21:03.039287
Epoch 1/1
 - 401s - loss: 0.4127 - auroc: 0.5579 - val_loss: 0.4065 - val_auroc: 0.7423
Epoch 4 start at time  2019-07-03 00:27:48.179986
Epoch 1/1
 - 401s - loss: 0.4099 - auroc: 0.5978 - val_loss: 0.3967 - val_auroc: 0.7408
Epoch 5 start at time  2019-07-03 00:34:33.059392
Epoch 1/1
 - 400s - loss: 0.4069 - auroc: 0.6148 - val_loss: 0.3947 - val_auroc: 0.7505
Epoch 6 start at time  2019-07-03 00:41:17.022177
Epoch 1/1
 - 400s - loss: 0.4015 - auroc: 0.6507 - val_loss: 0.3916 - val_auroc: 0.7259
Epoch 7 start at time  2019-07-03 00:48:01.146367
Epoch 1/1
 - 399s - loss: 0.3974 - auroc: 0.6703 - val_loss: 0.3865 - val_auroc: 0.7559
Epoch 8 start at time  2019-07-03 00:54:44.655848
Epoch 1/1
 - 404s - loss: 0.3958 - auroc: 0.6725 - val_loss: 0.3848 - val_auroc: 0.7598
Epoch 9 start at time  2019-07-03 01:01:32.580948
Epoch 1/1
 - 400s - loss: 0.3937 - auroc: 0.6810 - val_loss: 0.3830 - val_auroc: 0.7610
Epoch 10 start at time  2019-07-03 01:08:16.580513
Epoch 1/1
 - 403s - loss: 0.3933 - auroc: 0.6811 - val_loss: 0.3814 - val_auroc: 0.7493
Epoch 11 start at time  2019-07-03 01:15:04.180188
Epoch 1/1
 - 362s - loss: 0.3912 - auroc: 0.6886 - val_loss: 0.3784 - val_auroc: 0.7656
Epoch 12 start at time  2019-07-03 01:21:11.473738
Epoch 1/1
 - 394s - loss: 0.3891 - auroc: 0.6959 - val_loss: 0.3754 - val_auroc: 0.7686
```

```
Epoch 13 start at time  2019-07-03 01:27:49.549383
Epoch 1/1
 - 400s - loss: 0.3796 - auroc: 0.7239 - val_loss: 0.3677 - val_auroc: 0.7771
Epoch 14 start at time  2019-07-03 01:34:33.755273
Epoch 1/1
 - 399s - loss: 0.3713 - auroc: 0.7425 - val_loss: 0.3645 - val_auroc: 0.7800
Epoch 15 start at time  2019-07-03 01:41:17.045430
Epoch 1/1
 - 398s - loss: 0.3661 - auroc: 0.7513 - val_loss: 0.3580 - val_auroc: 0.7820
Epoch 16 start at time  2019-07-03 01:47:58.923634
Epoch 1/1
 - 399s - loss: 0.3595 - auroc: 0.7611 - val_loss: 0.3554 - val_auroc: 0.7854
Epoch 17 start at time  2019-07-03 01:54:42.010034
Epoch 1/1
 - 399s - loss: 0.3539 - auroc: 0.7679 - val_loss: 0.3515 - val_auroc: 0.7862
Epoch 18 start at time  2019-07-03 02:01:24.698776
Epoch 1/1
 - 398s - loss: 0.3494 - auroc: 0.7750 - val_loss: 0.3490 - val_auroc: 0.7918
Epoch 19 start at time  2019-07-03 02:08:07.286893
Epoch 1/1
 - 397s - loss: 0.3425 - auroc: 0.7860 - val_loss: 0.3451 - val_auroc: 0.7957
Epoch 20 start at time  2019-07-03 02:14:48.009221
Epoch 1/1
 - 398s - loss: 0.3376 - auroc: 0.7883 - val_loss: 0.3436 - val_auroc: 0.7969
Epoch 21 start at time  2019-07-03 02:21:30.117382
Epoch 1/1
 - 399s - loss: 0.3320 - auroc: 0.7962 - val_loss: 0.3423 - val_auroc: 0.7967
Epoch 22 start at time  2019-07-03 02:28:13.086297
Epoch 1/1
 - 382s - loss: 0.3267 - auroc: 0.7994 - val_loss: 0.3462 - val_auroc: 0.7934
Epoch 23 start at time  2019-07-03 02:34:39.395761
Epoch 1/1
 - 418s - loss: 0.3278 - auroc: 0.8000 - val_loss: 0.3436 - val_auroc: 0.7979
Epoch 24 start at time  2019-07-03 02:41:42.876569
Epoch 1/1
 - 372s - loss: 0.3223 - auroc: 0.8040 - val_loss: 0.3450 - val_auroc: 0.7958
```

Model Prediction

In [234]:

```
model3.load_weights("model3_1_epoch_24.h5")

# model compilation
model3.compile(loss='binary_crossentropy', optimizer='adam')
```

In [235]:

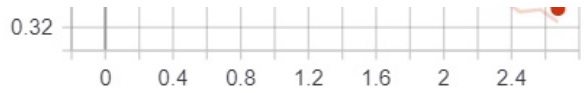
```
history = model3.predict([test_essay_sequence,test_other_than])
```

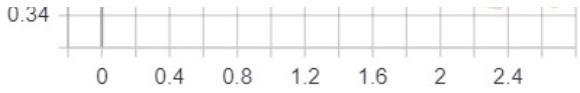
In [236]:

```
y_pred = (history > 0.5).astype(np.int)
```

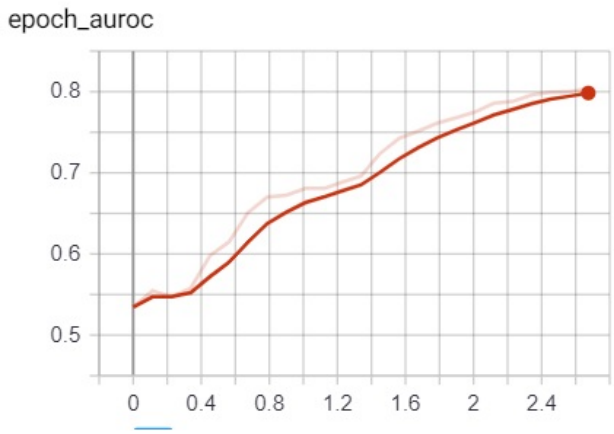
Loss and AUC

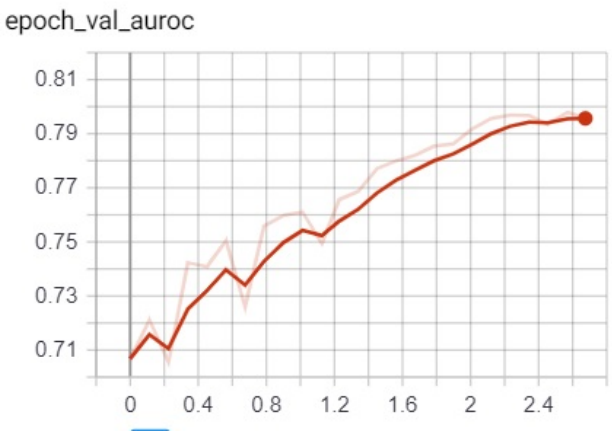
Epoch Loss And Validation Loss





Epoch AUC and Validation AUC





AUC Score

In [238]:

```
from sklearn.metrics import roc_auc_score

# AUC for test data

print("AUC score is {}".format(roc_auc_score(y_test,history)))
```

```
AUC score is 0.7954055260587393
```

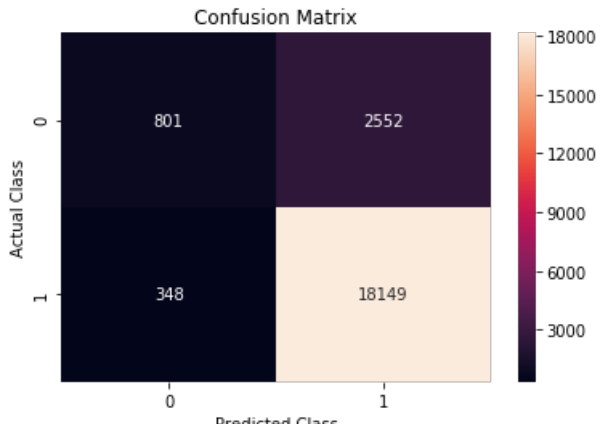
Confusion Matrix

In [237]:

```
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test,y_pred)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm1, annot=True, fmt="d")
plt.ylabel("Actual Class")
plt.xlabel("Predicted Class")
plt.title("Confusion Matrix")
```

Out[237]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



Predicted Class