Assignment_14_LSTM_IMDB

June 18, 2019

In [2]:
```
# Credits: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-n
# LSTM for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM,
from keras.layers import Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import os
import sqlite3

from sklearn.metrics import roc_auc_score
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
# fix random seed for reproducibility
numpy.random.seed(7)

!apt-get install -y -qq software-properties-common python-software-properties module-in
!add-apt-repository -y ppa:alessandro-strada/ppa 2>&1 > /dev/null
!apt-get update -qq 2>&1 > /dev/null
!apt-get -y install -qq google-drive-ocamlfuse fuse
from google.colab import auth
auth.authenticate_user()
from oauth2client.client import GoogleCredentials
creds = GoogleCredentials.get_application_default()
import getpass
!google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.client_secret} <
```

```
        vcode = getpass.getpass()
        !echo {vcode} | google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.
        !mkdir drive
        !google-drive-ocamlfuse drive
```

```
E: Package 'python-software-properties' has no installation candidate
Selecting previously unselected package google-drive-ocamlfuse.
(Reading database ... 130912 files and directories currently installed.)
Preparing to unpack .../google-drive-ocamlfuse_0.7.4-0ubuntu1~ubuntu18.04.1_amd64.deb ...
Unpacking google-drive-ocamlfuse (0.7.4-0ubuntu1~ubuntu18.04.1) ...
Setting up google-drive-ocamlfuse (0.7.4-0ubuntu1~ubuntu18.04.1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Please, open the following URL in a web browser: https://accounts.google.com/o/oauth2/auth?clie
ůůůůůůůůůů
Please, open the following URL in a web browser: https://accounts.google.com/o/oauth2/auth?clie
Please enter the verification code: Access token retrieved correctly.
```

Data Preprocessing :

```
In [4]: # using SQLite Table to read data.
        os.chdir("/content/drive/Colab Notebooks") #changing directory
        con = sqlite3.connect('database.sqlite')

        # filtering only positive and negative reviews i.e.
        # not taking into consideration those reviews with Score=3
        # SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
        # you can change the number to any other number based on your computing power

        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500
        # for tsne assignment you can take 5k data points

        #filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""", con)

        # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
        def partition(x):
            if x < 3:
                return 0
            return 1

        #changing reviews with score less than 3 to be positive and vice-versa
        actualScore = filtered_data['Score']
        positiveNegative = actualScore.map(partition)
        filtered_data['Score'] = positiveNegative
        print("Number of data points in our data", filtered_data.shape)
        filtered_data.head(3)
```

```
Number of data points in our data (500000, 10)
```

```
Out[4]:    Id  ...                                               Text
        0   1  ...  I have bought several of the Vitality canned d...
        1   2  ...  Product arrived labeled as Jumbo Salted Peanut...
        2   3  ...  This is a confection that has been around a fe...

        [3 rows x 10 columns]
```

```
In [0]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [6]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[6]:                 UserId  ... COUNT(*)
        0  #oc-R115TNMSPFT9I7  ...        2
        1  #oc-R11D9D7SHXIJB9  ...        3
        2  #oc-R11DNU2NBKQ23Z  ...        2
        3  #oc-R11O5J5ZVQE25C  ...        3
        4  #oc-R12KPBODL2B5ZD  ...        2

        [5 rows x 7 columns]
```

```
In [7]: display['COUNT(*)'].sum()
```

```
Out[7]: 393063
```

```
In [0]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

```
In [9]: #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
        final.shape
```

```
Out[9]: (348262, 10)
```

```
In [10]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[10]: 69.6524
```

```
In [0]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [12]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(348260, 10)
```

Out[12]:
```
1    293516
0     54744
Name: Score, dtype: int64
```

In [0]:
```
# https://stackoverflow.com/a/47091490/4084039
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
```

In [18]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is defini
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a t
==================================================
```

```
In [19]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
         sent_150 = re.sub(r"http\S+", "", sent_1500)
         sent_4900 = re.sub(r"http\S+", "", sent_4900)

         print(sent_0)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
```

```
In [20]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
         from bs4 import BeautifulSoup

         soup = BeautifulSoup(sent_0, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1000, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1500, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_4900, 'lxml')
         text = soup.get_text()
         print(text)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is defini
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
```

```
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a t
```

```
In [0]: # https://gist.github.com/sebleier/554280
        # we are removing the words from the stop words list: 'no', 'nor', 'not'
        # <br /><br /> ==> after the above steps, we are getting "br br"
        # we are including them into stop words list
        # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

        stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves
                    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [21]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 348260/348260 [02:52<00:00, 2016.77it/s]
```

```
In [22]: preprocessed_reviews[1500]
```

```
Out[22]: 'great product healthy dogs first food love eat helped older dog lose weight year old
```

Splitting the data :

```
In [0]: from sklearn.model_selection import train_test_split
        final['Text'] = preprocessed_reviews
```

```
        finalp = final[final.Score == 1].sample(25000,random_state =2)
        finaln = final[final.Score == 0].sample(25000,random_state =2)
        finalx = pd.concat([finalp,finaln],ignore_index=True)
        finalx = finalx.sort_values('Time')
        y = finalx.Score.values
        X = finalx.Text.values
        X_train, X_test , y_train, y_test = train_test_split(X,y,test_size=0.2)
        X_tr, X_cv , y_tr, y_cv = train_test_split(X_train,y_train,test_size=0.2)
```

```
In [0]: from keras.preprocessing.text import Tokenizer
        tokens = Tokenizer(num_words=5000)
        tokens.fit_on_texts(X_tr)

        X_tr = tokens.texts_to_sequences(X_tr)
        X_cv = tokens.texts_to_sequences(X_cv)
        X_test = tokens.texts_to_sequences(X_test)
```

```
In [66]: print(X_tr[1])
         print(type(X_tr[1]))
         print(len(X_tr[1]))
```

```
[17, 15, 167, 65, 246, 164, 135, 167, 31, 471, 95, 598, 2285, 309, 213, 441, 3, 221, 67]
<class 'list'>
19
```

```
In [67]: # truncate and/or pad input sequences
         max_review_length = 600
         X_tr = sequence.pad_sequences(X_tr, maxlen=max_review_length)
         X_cv = sequence.pad_sequences(X_cv, maxlen=max_review_length)
         X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

         print(X_tr.shape)
         print(X_tr[1])
         print(X_cv.shape)
         print(X_cv[1])
         print(X_test.shape)
         print(X_test[1])
```

```
(32000, 600)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0   17   15  167   65  246  164  135
  167   31  471   95  598 2285  309  213  441    3  221   67]
(8000, 600)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0   60   41 1288  150   58    4]
(10000, 600)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0 3279  143  336   68  546    1  177   58
  117    7  168  179 1092  336  711 1073  101   35  259  172    5  208
    2   35 1267  947   31  326   30  589    3  711   78   42  357  579
  943   35  259  172 3206 1604 1763  115    7  227  182  179    2 1037
  412  168  179  867  143  729 1050   11    3  168  276  130  796  453
 4994  561  945 2353 1403  247  412   85  168  247 1993   26 4224  247
   87  647   97   32  582 3719   43 2443 2498  336   28    1  268 4702
 2498   14  742  797  140  190  179  326  780  168    1  143  336    1
  472 2091  922  272  143  336   95  168  247  550  197    6  144   43
   74   10 2240   33   68    6  255   28  809  544 2443  209]
```

```
In [0]: import matplotlib.pyplot as plt
        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()
```

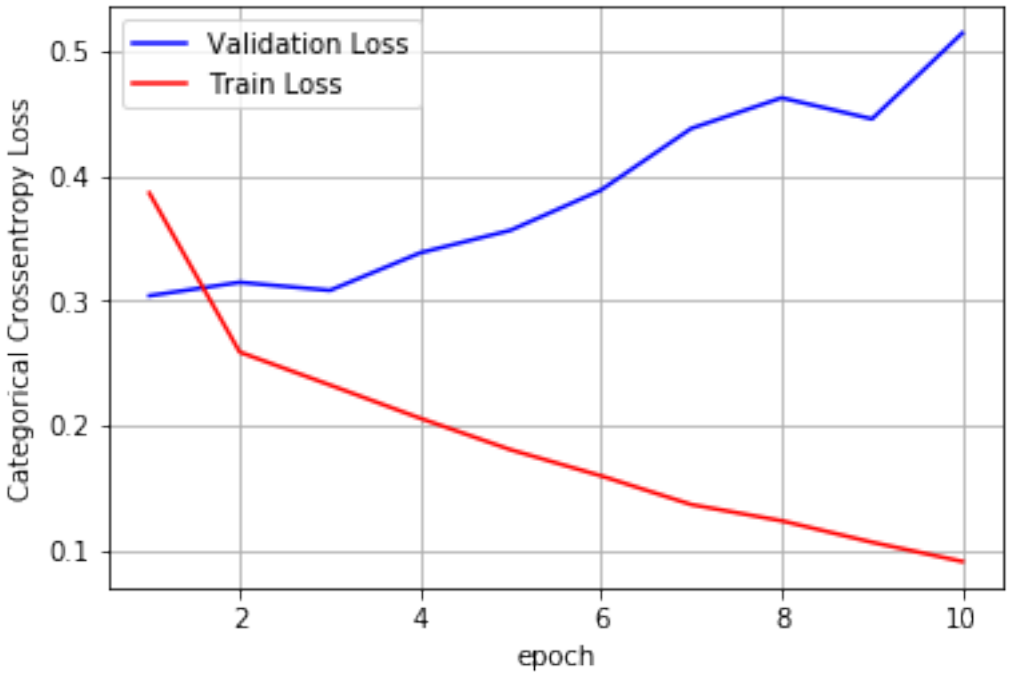
SINGLE LSTM LAYER :

```
In [72]: # create the model
         embedding_vecor_length = 32
         top_words = 5000
         def model():
           model = Sequential()
           model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_le
           model.add(LSTM(100))
           model.add(Dense(1, activation='sigmoid'))
           model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
           print(model.summary())
           #Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-
```

```
    nb_epoch = 10
    history = model.fit(X_tr, y_tr, nb_epoch=nb_epoch, batch_size=64,validation_data=(X_
    # Final evaluation of the model
    scores = model.evaluate(X_test, y_test, verbose=0)
    print("Accuracy: %.2f%%" % (scores[1]*100))
    fig,ax = plt.subplots(1,1)
    ax.set_xlabel('epoch')
    ax.set_ylabel('Categorical Crossentropy Loss')
    x = list(range(1,nb_epoch+1))
    vy = history.history['val_loss']
    ty = history.history['loss']
    plt_dynamic(x, vy, ty, ax)
model()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_16 (Embedding)     (None, 600, 32)           160032
_________________________________________________________________
lstm_23 (LSTM)               (None, 100)               53200
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 101
=================================================================
Total params: 213,333
Trainable params: 213,333
Non-trainable params: 0
_________________________________________________________________
None
Train on 32000 samples, validate on 8000 samples
Epoch 1/10
32000/32000 [==============================] - 419s 13ms/step - loss: 0.3861 - acc: 0.8264 - va
Epoch 2/10
32000/32000 [==============================] - 420s 13ms/step - loss: 0.2588 - acc: 0.8973 - va
Epoch 3/10
32000/32000 [==============================] - 418s 13ms/step - loss: 0.2324 - acc: 0.9090 - va
Epoch 4/10
32000/32000 [==============================] - 417s 13ms/step - loss: 0.2058 - acc: 0.9208 - va
Epoch 5/10
32000/32000 [==============================] - 419s 13ms/step - loss: 0.1806 - acc: 0.9324 - va
Epoch 6/10
32000/32000 [==============================] - 419s 13ms/step - loss: 0.1598 - acc: 0.9395 - va
Epoch 7/10
32000/32000 [==============================] - 419s 13ms/step - loss: 0.1367 - acc: 0.9488 - va
Epoch 8/10
32000/32000 [==============================] - 417s 13ms/step - loss: 0.1238 - acc: 0.9555 - va
Epoch 9/10
32000/32000 [==============================] - 417s 13ms/step - loss: 0.1066 - acc: 0.9602 - va
Epoch 10/10
```

```
32000/32000 [==============================] - 417s 13ms/step - loss: 0.0912 - acc: 0.9677 - va
Accuracy: 85.83%
```

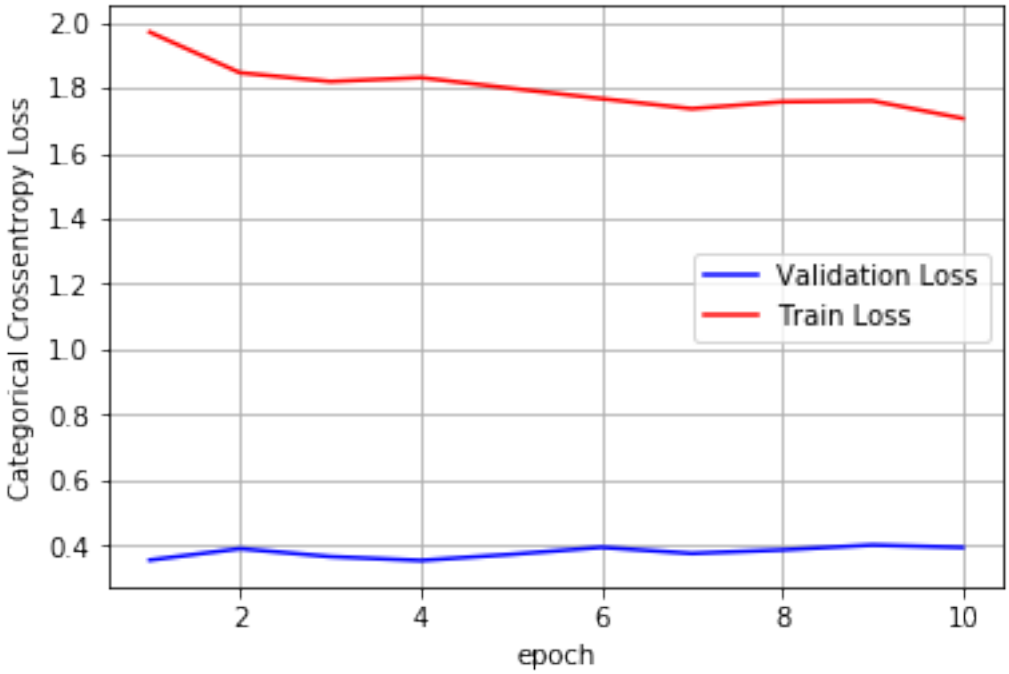


Model with 2 LSTM Layers and droputs:

```
In [73]: def model():
             model = Sequential()
             model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_le
             model.add(LSTM(100,return_sequences=True))
             model.add(Dropout(0.5))
             model.add(LSTM(50))
             model.add(Dense(1, activation='sigmoid'))
             model.add(Dropout(0.2))
             model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
             print(model.summary())
             #Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-
             nb_epoch = 10
             history = model.fit(X_tr, y_tr, nb_epoch=nb_epoch, batch_size=64,validation_data=(X_
             # Final evaluation of the model
             scores = model.evaluate(X_test, y_test, verbose=0)
             print("Accuracy: %.2f%%" % (scores[1]*100))
             fig,ax = plt.subplots(1,1)
             ax.set_xlabel('epoch')
             ax.set_ylabel('Categorical Crossentropy Loss')
             x = list(range(1,nb_epoch+1))
```

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_17 (Embedding)     (None, 600, 32)           160032
_________________________________________________________________
lstm_24 (LSTM)               (None, 600, 100)          53200
_________________________________________________________________
dropout_10 (Dropout)         (None, 600, 100)          0
_________________________________________________________________
lstm_25 (LSTM)               (None, 50)                30200
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 51
_________________________________________________________________
dropout_11 (Dropout)         (None, 1)                 0
=================================================================
Total params: 243,483
Trainable params: 243,483
Non-trainable params: 0
_________________________________________________________________
None
Train on 32000 samples, validate on 8000 samples
Epoch 1/10
32000/32000 [==============================] - 760s 24ms/step - loss: 1.9702 - acc: 0.7525 - va
Epoch 2/10
32000/32000 [==============================] - 766s 24ms/step - loss: 1.8459 - acc: 0.8067 - va
Epoch 3/10
32000/32000 [==============================] - 767s 24ms/step - loss: 1.8191 - acc: 0.8213 - va
Epoch 4/10
32000/32000 [==============================] - 761s 24ms/step - loss: 1.8311 - acc: 0.8253 - va
Epoch 5/10
32000/32000 [==============================] - 760s 24ms/step - loss: 1.7983 - acc: 0.8317 - va
Epoch 6/10
32000/32000 [==============================] - 761s 24ms/step - loss: 1.7668 - acc: 0.8395 - va
Epoch 7/10
32000/32000 [==============================] - 760s 24ms/step - loss: 1.7360 - acc: 0.8458 - va
Epoch 8/10
32000/32000 [==============================] - 761s 24ms/step - loss: 1.7572 - acc: 0.8512 - va
Epoch 9/10
32000/32000 [==============================] - 760s 24ms/step - loss: 1.7601 - acc: 0.8546 - va
Epoch 10/10
32000/32000 [==============================] - 761s 24ms/step - loss: 1.7065 - acc: 0.8627 - va
Accuracy: 87.30%
```

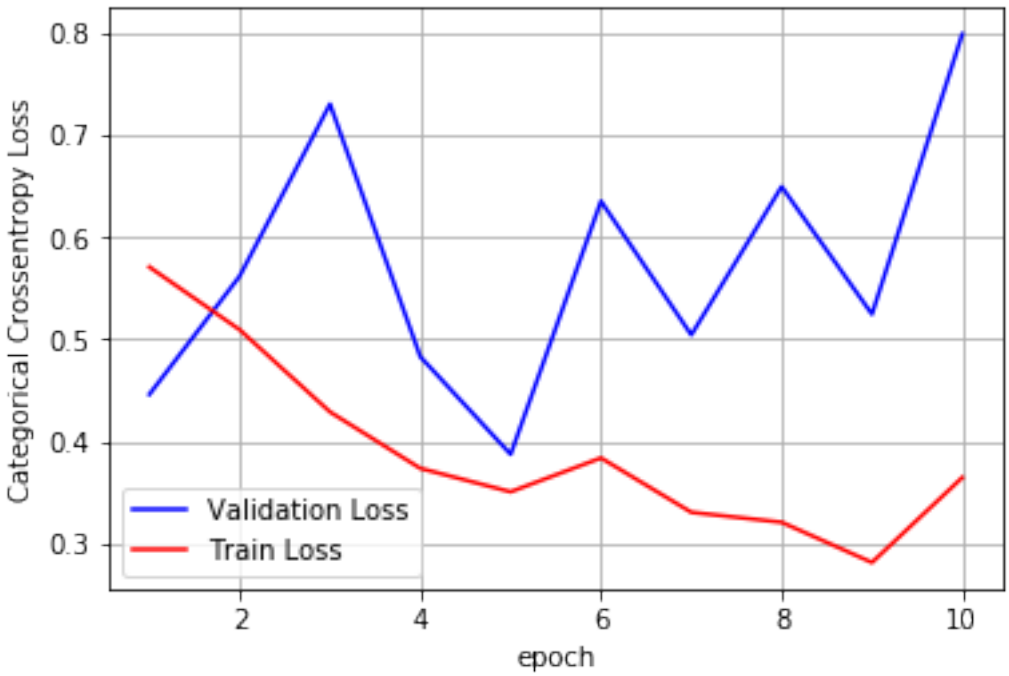


Model with 3 LSTM layers and dropouts(Using ReLu here):

```
In [84]: def model():
             model = Sequential()
             model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_le
             model.add(LSTM(100,return_sequences=True))
             model.add(Dropout(0.5))
             model.add(LSTM(40,return_sequences=True))
             model.add(Dropout(0.5))
             model.add(LSTM(20))
             model.add(Dense(1,activation='relu',kernel_initializer='he_normal'))
             model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
             print(model.summary())
             #Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-
             nb_epoch = 10
             history = model.fit(X_tr, y_tr, nb_epoch=nb_epoch, batch_size=64,validation_data=(X_
             # Final evaluation of the model
             scores = model.evaluate(X_test, y_test, verbose=0)
             print("Accuracy: %.2f%%" % (scores[1]*100))
             fig,ax = plt.subplots(1,1)
             ax.set_xlabel('epoch')
             ax.set_ylabel('Categorical Crossentropy Loss')
             x = list(range(1,nb_epoch+1))
             vy = history.history['val_loss']
             ty = history.history['loss']
```

```
          plt_dynamic(x, vy, ty, ax)
        model()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_28 (Embedding)     (None, 600, 32)           160032
_________________________________________________________________
lstm_51 (LSTM)               (None, 600, 100)          53200
_________________________________________________________________
dropout_34 (Dropout)         (None, 600, 100)          0
_________________________________________________________________
lstm_52 (LSTM)               (None, 600, 40)           22560
_________________________________________________________________
dropout_35 (Dropout)         (None, 600, 40)           0
_________________________________________________________________
lstm_53 (LSTM)               (None, 20)                4880
_________________________________________________________________
dense_24 (Dense)             (None, 1)                 21
=================================================================
Total params: 240,693
Trainable params: 240,693
Non-trainable params: 0
_________________________________________________________________
None
Train on 32000 samples, validate on 8000 samples
Epoch 1/10
32000/32000 [==============================] - 921s 29ms/step - loss: 0.5708 - acc: 0.7575 - va
Epoch 2/10
32000/32000 [==============================] - 913s 29ms/step - loss: 0.5093 - acc: 0.7662 - va
Epoch 3/10
32000/32000 [==============================] - 912s 28ms/step - loss: 0.4287 - acc: 0.8385 - va
Epoch 4/10
32000/32000 [==============================] - 910s 28ms/step - loss: 0.3734 - acc: 0.8599 - va
Epoch 5/10
32000/32000 [==============================] - 909s 28ms/step - loss: 0.3504 - acc: 0.8782 - va
Epoch 6/10
32000/32000 [==============================] - 910s 28ms/step - loss: 0.3836 - acc: 0.8439 - va
Epoch 7/10
32000/32000 [==============================] - 909s 28ms/step - loss: 0.3303 - acc: 0.8679 - va
Epoch 8/10
32000/32000 [==============================] - 911s 28ms/step - loss: 0.3206 - acc: 0.8698 - va
Epoch 9/10
32000/32000 [==============================] - 912s 29ms/step - loss: 0.2811 - acc: 0.8729 - va
Epoch 10/10
32000/32000 [==============================] - 916s 29ms/step - loss: 0.3644 - acc: 0.8334 - va
Accuracy: 84.19%
```




```
In [88]: from prettytable import PrettyTable
         x = PrettyTable()
         x.title = "LSTM"
         x.field_names = [ "Architecture","Overfitting","Accuracy"]
         x.add_row([ "Embedding->LSTM->Sigmoid", "less","85.83%" ])
         x.add_row([ "Embedding->LSTM->Dropout->LSTM->Sigmoid->Dropout","More", "87.3%"])
         x.add_row(["Embedding->LSTM->Dropout->LSTM->Dropout->LSTM->ReLU","less", "84.19%"])
         print(x)
```

```
+-----------------------------------------------------+-------------+----------+
|                     Architecture                    | Overfitting | Accuracy |
+-----------------------------------------------------+-------------+----------+
|               Embedding->LSTM->Sigmoid              |     less    |  85.83%  |
|   Embedding->LSTM->Dropout->LSTM->Sigmoid->Dropout  |     More    |  87.3%   |
| Embedding->LSTM->Dropout->LSTM->Dropout->LSTM->ReLU |     less    |  84.19%  |
+-----------------------------------------------------+-------------+----------+
```

CONCLUSIONS: Tried 3 different achitectures: single LSTM layer, 2 LSTM layers, and 3 LSTM layers.

Observed that dropout decreases overfitting, but not much of a difference in these models.

Though the second model gave more accuracy, it has comparatively more overfitting.

ReLU worked better than sigmoid as observed from last model.

CNN worked better than LSTM