```
import pandas as pd
import numpy as np
import nltk
import sqlite3
import os
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, LSTM, Dropout
from keras.layers.embeddings import Embedding
```

```
Using TensorFlow backend.
```

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clie

Enter your authorization code:
··········
Mounted at /content/gdrive
```

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

```
con = sqlite3.connect("/content/gdrive/My Drive/Colab Notebooks/database.sqli
filtered_data = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3""

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
```

```
#Deduplication of entries
final=filtered_data.drop_duplicates(subset={"UserId","ProfileName","Time","Te
final.shape
```

```
(364173, 10)
```

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
final = final.sample(n=50000, replace=True)
```

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'i
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', '
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'beca
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'th
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", '
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shou
            'won', "won't", 'wouldn', "wouldn't"])
```

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-re
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
```

```
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
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 50000/50000 [00:16<00:00, 3051.25it/s]
```

```
final['CleanedText'] = preprocessed_reviews
```

```
x = final['CleanedText']
y = final['Score']

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

```
#Source: https://keras.io/preprocessing/text/
#Source: https://www.kaggle.com/kredy10/simple-lstm-for-text-classification
#Source: https://keras.io/preprocessing/sequence/

max_words = 5000
max_len = 600
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(x_train)
```

```
sequences = tok.texts_to_sequences(x_train)
train_sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
```

```
test_sequences = tok.texts_to_sequences(x_test)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)
```

```
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(max_words+1, embedding_vecor_length, input_length=max_len
model.add(LSTM(100, dropout = 0.3, recurrent_dropout=0.3, return_sequences=Tr
model.add(BatchNormalization())
model.add(LSTM(100, dropout = 0.4, recurrent_dropout=0.4))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/p
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backer
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 600, 32)           160032
_________________________________________________________________
lstm_1 (LSTM)                (None, 600, 100)          53200
_________________________________________________________________
batch_normalization_1 (Batch (None, 600, 100)          400
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 294,133
Trainable params: 293,933
Non-trainable params: 200
_________________________________________________________________
None
```

```
history = model.fit(train_sequences_matrix, y_train, epochs=10, batch_size=10
```

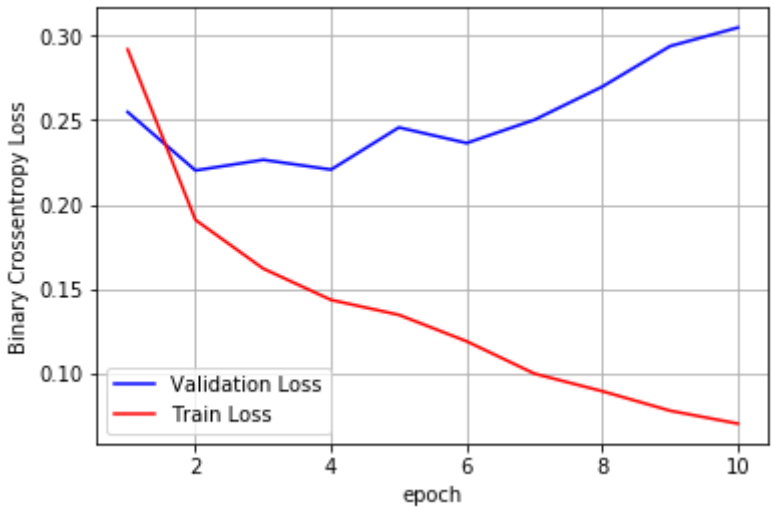
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/p
Instructions for updating:
Use tf.cast instead.
Train on 35000 samples, validate on 15000 samples
Epoch 1/10
35000/35000 [==============================] - 771s 22ms/step - loss: 0.292
Epoch 2/10
35000/35000 [==============================] - 771s 22ms/step - loss: 0.190
Epoch 3/10
```

```
scores = model.evaluate(test_sequences_matrix, y_test, verbose=0)
model2_acc = scores[1]
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 91.69%
35000/35000 [==============================] - 773s 22ms/step - loss: 0.118
```

```
print('Test score:', scores[0])
print('Test accuracy:', scores[1])

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

# list of epoch numbers
x = list(range(1,11))

vy = history.history['val_loss']
ty = history.history['loss']


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

```
Test score: 0.3049351974586646
Test accuracy: 0.9169333333651225
```




```
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(max_words+1, embedding_vecor_length, input_length=max_len
model.add(LSTM(100, dropout = 0.3, recurrent_dropout=0.3, return_sequences=Tr
model.add(BatchNormalization())
model.add(LSTM(100, dropout = 0.5, recurrent_dropout=0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 600, 32)           160032
_________________________________________________________________
lstm_3 (LSTM)                (None, 600, 100)          53200
_________________________________________________________________
batch_normalization_2 (Batch (None, 600, 100)          400
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 294,133
Trainable params: 293,933
```

```
history = model.fit(train_sequences_matrix, y_train, epochs=10, batch_size=10
```

```
Train on 35000 samples, validate on 15000 samples
Epoch 1/10
35000/35000 [==============================] - 778s 22ms/step - loss: 0.291
Epoch 2/10
35000/35000 [==============================] - 778s 22ms/step - loss: 0.189
Epoch 3/10
35000/35000 [==============================] - 780s 22ms/step - loss: 0.166
Epoch 4/10
35000/35000 [==============================] - 778s 22ms/step - loss: 0.147
Epoch 5/10
35000/35000 [==============================] - 779s 22ms/step - loss: 0.145
Epoch 6/10
35000/35000 [==============================] - 772s 22ms/step - loss: 0.131
Epoch 7/10
35000/35000 [==============================] - 777s 22ms/step - loss: 0.114
Epoch 8/10
35000/35000 [==============================] - 776s 22ms/step - loss: 0.106
Epoch 9/10
35000/35000 [==============================] - 775s 22ms/step - loss: 0.099
Epoch 10/10
35000/35000 [==============================] - 777s 22ms/step - loss: 0.099
```

```
scores = model.evaluate(test_sequences_matrix, y_test, verbose=0)
model3_acc = scores[1]
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 91.53%
```

```
print('Test score:', scores[0])
print('Test accuracy:', scores[1])

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

# list of epoch numbers
x = list(range(1,11))


vy = history.history['val_loss']
ty = history.history['loss']

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

```
Test score: 0.3190554770519336
Test accuracy: 0.9152666666348775
```


```
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(max_words+1, embedding_vecor_length, input_length=max_len
model.add(LSTM(100, dropout = 0.7, recurrent_dropout=0.7, return_sequences=Tr
model.add(BatchNormalization())
model.add(LSTM(100, dropout = 0.5, recurrent_dropout=0.5, return_sequences=Tr
model.add(BatchNormalization())
model.add(LSTM(100, dropout = 0.3, recurrent_dropout=0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 600, 32)           160032
_________________________________________________________________
lstm_5 (LSTM)                (None, 600, 100)          53200
_________________________________________________________________
batch_normalization_3 (Batch (None, 600, 100)          400
_________________________________________________________________
lstm_6 (LSTM)                (None, 600, 100)          80400
_________________________________________________________________
batch_normalization_4 (Batch (None, 600, 100)          400
_________________________________________________________________
lstm_7 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 374,933
Trainable params: 374,533
Non-trainable params: 400
_________________________________________________________________
None
```

```
history = model.fit(train_sequences_matrix, y_train, epochs=10, batch_size=10
```

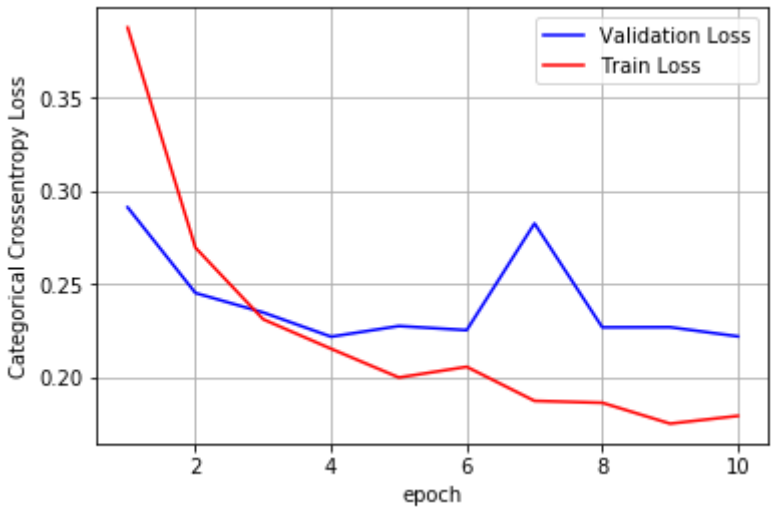
```
Train on 35000 samples, validate on 15000 samples
Epoch 1/10
35000/35000 [==============================] - 1169s 33ms/step - loss: 0.38
Epoch 2/10
35000/35000 [==============================] - 1167s 33ms/step - loss: 0.26
Epoch 3/10
35000/35000 [==============================] - 1164s 33ms/step - loss: 0.23
Epoch 4/10
35000/35000 [==============================] - 1169s 33ms/step - loss: 0.21
Epoch 5/10
35000/35000 [==============================] - 1165s 33ms/step - loss: 0.19
Epoch 6/10
35000/35000 [==============================] - 1172s 33ms/step - loss: 0.20
Epoch 7/10
```

```
scores = model.evaluate(test_sequences_matrix, y_test, verbose=0)
model4_acc = scores[1]
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 91.63%
```

```
print('Test score:', scores[0])
print('Test accuracy:', scores[1])

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

# list of epoch numbers
x = list(range(1,11))


vy = history.history['val_loss']
ty = history.history['loss']

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

```
Test score: 0.22179906602303187
Test accuracy: 0.9163333333015442
```




```
from prettytable import PrettyTable

model_metric = PrettyTable()

model_metric = PrettyTable(["Model", "# LSTM Layers", 'Test Accuracy'])

model_metric.add_row(['LSTM','2 layers','%.4f%%' %model2_acc])
model_metric.add_row(['LSTM','2 layer','%.4f%%' %model3_acc])
model_metric.add_row(['LSTM','3 layers','%.4f%%' %model4_acc])
```

```
10 print(model_metric.get_string(start=0, end=8))
```

```
+-------+---------------+---------------+
| Model | # LSTM Layers | Test Accuracy |
+-------+---------------+---------------+
|  LSTM |    2 layers   |    0.9169%    |
|  LSTM |    2 layer    |    0.9153%    |
|  LSTM |    3 layers   |    0.9163%    |
+-------+---------------+---------------+
```