In [1]:

```
# Credits: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networ
ks-python-keras/
# 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.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)

%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 sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
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

from keras.preprocessing.text import Tokenizer
from keras.layers import Dense, Dropout, Flatten

from keras.layers.normalization import BatchNormalization
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

In [2]:

```
%matplotlib notebook
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, 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()
```

In [3]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:/Users/sesha/OneDrive/Desktop/ICONS/IMP/before/MINIPJ/Personal/AMAZON foo
d review 2/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
```

```
# 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 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query("""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
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 (100000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [4]:

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

In [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

393063

EXPLORATORY DATA ANALYSIS

In [7]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

Out[9]:

```
(87775, 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]:

```
87.775
```

In [11]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [12]:

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

In [13]:

```
#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()
```

```
(87773, 10)
```

Out[13]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

PREPROCESSING

In [14]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
```

```
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [15]:

```python
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [16]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an
-element
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
```

ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers. It's definitely worth it to buy a big bag if your dog eats them a lot.

In [17]:

```
# 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]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [19]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [20]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

In [21]:

```
# 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', "y
ou'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',
```

```
            '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', 'e
ach', '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', "do
esn'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 [22]:

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

```
100%|██████████████████████████████████████████████████████████████████████| 87773/87773
[00:26<00:00, 3369.84it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'way hot blood took bite jig lol'
```

In [24]:

```
final ['preprocessed_reviews']= preprocessed_reviews
final.head(5)
```

Out[24]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Sum
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1	0	1192060800	m
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0	1	1195948800	Dog [
70677	76870	B00002N8SM	A19Q006CSFT011	Arlielle	0	0	0	1288396800	on

D

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Sum
70676	76869	B00002N8SM	A1FYH4S02BW7FN	wonderer	0	0	0	1290038400	m
70675	76868	B00002N8SM	AUE8TB5VHS6ZV	eyeofthestorm	0	0	0	1306972800	A

In [25]:

```
preprocessed_summary = []
# tqdm is for printing the status bar
for summary in tqdm(final['Summary'].values):
    summary = re.sub(r"http\S+", "", summary)
    # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
    summary = BeautifulSoup(summary, 'lxml').get_text()
    # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-fro
m-an-element
    summary = decontracted(summary)
    summary = re.sub("\S*\d\S*", "", summary).strip() #remove words with numbers python: https://st
ackoverflow.com/a/18082370/4084039
    summary = re.sub('[^A-Za-z]+', ' ', summary) #remove spacial character:
https://stackoverflow.com/a/5843547/4084039
    # https://gist.github.com/sebleier/554280
    summary = ' '.join(e.lower() for e in summary.split() if e.lower() not in stopwords)
    preprocessed_summary.append(summary.strip())
```

```
100%|██████████████████████████████████████████████████████████| 87773/87773
[00:18<00:00, 4855.39it/s]
```

In [26]:

```
preprocessed_reviews = [i + ' ' + j for i, j in zip(preprocessed_reviews,preprocessed_summary)]
print(preprocessed_reviews[1500])
```

```
way hot blood took bite jig lol hot stuff
```

Splitting the data

In [27]:

```
Y = final['Score'].values
X = np.array(preprocessed_reviews)
```

In [28]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split

#X_train, X_test, y_train, y_test = train_test_split(X, final['Score'], test_size=0.33, shuffle=Fa
lse,random_state=0)
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,
test_size=0.33,shuffle=False,random_state=0)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33) # this is random splittin
g
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random
splitting


print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
(39400,) (39400,)
(19407,) (19407,)
```

```
(28966,) (28966,)
==========================================================================================
```

Tokenizing the data

In [29]:

```
tokenize = Tokenizer(num_words=5000)
tokenize.fit_on_texts(X_train)

X_train_new = tokenize.texts_to_sequences(X_train)
X_cv_new = tokenize.texts_to_sequences(X_cv)
X_test_new = tokenize.texts_to_sequences(X_test)

print(X_train_new[1])
print(len(X_train_new))
```

```
[535, 480, 83, 133, 160, 737, 413, 291, 2, 126, 1085, 333, 1215, 497, 255, 48, 88, 370, 291, 1, 55
2, 1110, 462, 291, 83]
39400
```

In [30]:

```
print(X_train_new[1])
print(type(X_train_new[1]))
print(len(X_train_new[1]))
```

```
[535, 480, 83, 133, 160, 737, 413, 291, 2, 126, 1085, 333, 1215, 497, 255, 48, 88, 370, 291, 1, 55
2, 1110, 462, 291, 83]
<class 'list'>
25
```

Padding

In [31]:

```
# truncate and/or pad input sequences
max_review_length = 600
X_train = sequence.pad_sequences(X_train_new, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test_new, maxlen=max_review_length)
X_cv = sequence.pad_sequences(X_cv_new, maxlen=max_review_length)

print(X_train.shape)
print(X_train[1])
```

```
(39400, 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  535  480   83  133  160  737  413  291    2  126 1085  333 1215
  497  255   48   88  370  291    1  552 1110  462  291   83]
```

In [32]:

```
# create the model
embedding_vector_length = 32
model_1 = Sequential()

#Adding Embedding layer
model_1.add(Embedding(5000, embedding_vector_length, input_length=max_review_length))

#Adding 2 LSTM Layers
model_1.add(LSTM(100,return_sequences=True))
model_1.add(Dropout(0.25))

model_1.add(LSTM(100))

model_1.add(Dense(1, activation='sigmoid'))

#Compiling
model_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model_1.summary())
#Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in-an-lstm-mode
l
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 600, 32)           160000    
_________________________________________________________________
lstm_1 (LSTM)                (None, 600, 100)          53200     
_________________________________________________________________
dropout_1 (Dropout)          (None, 600, 100)          0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400     
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101       
=================================================================
Total params: 293,701
Trainable params: 293,701
Non-trainable params: 0
_________________________________________________________________
None
```

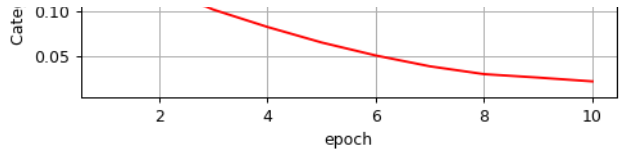
In [33]:

```
#Adding the cross validation data
history_1=model_1.fit(X_train, y_train, nb_epoch=10, batch_size=64,validation_data=(X_cv, y_cv))

# Final evaluation of the model
scores_1 = model_1.evaluate(X_test, y_test, verbose=0,batch_size=64)
print("Accuracy: %.2f%%" % (scores_1[1]*100))
```

```
Train on 39400 samples, validate on 19407 samples
Epoch 1/10
39400/39400 [==============================] - 508s 13ms/step - loss: 0.2138 - accuracy: 0.9149 -
val_loss: 0.1645 - val_accuracy: 0.9346
Epoch 2/10
39400/39400 [==============================] - 539s 14ms/step - loss: 0.1281 - accuracy: 0.9505 -
val_loss: 0.1622 - val_accuracy: 0.9384
Epoch 3/10
39400/39400 [==============================] - 531s 13ms/step - loss: 0.1018 - accuracy: 0.9626 -
val_loss: 0.1844 - val_accuracy: 0.9363
Epoch 4/10
39400/39400 [==============================] - 541s 14ms/step - loss: 0.0825 - accuracy: 0.9699 -
val_loss: 0.1906 - val_accuracy: 0.9335
Epoch 5/10
39400/39400 [==============================] - 533s 14ms/step - loss: 0.0652 - accuracy: 0.9769 -
val_loss: 0.2258 - val_accuracy: 0.9325
Epoch 6/10
39400/39400 [==============================] - 563s 14ms/step - loss: 0.0507 - accuracy: 0.9821 -
val_loss: 0.2488 - val_accuracy: 0.9251
Epoch 7/10
39400/39400 [==============================] - 534s 14ms/step - loss: 0.0386 - accuracy: 0.9874 -
val_loss: 0.2905 - val_accuracy: 0.9274
Epoch 8/10
39400/39400 [==============================] - 492s 12ms/step - loss: 0.0299 - accuracy: 0.9905 -
val_loss: 0.2903 - val_accuracy: 0.9276
Epoch 9/10
39400/39400 [==============================] - 494s 13ms/step - loss: 0.0261 - accuracy: 0.9912 -
val_loss: 0.3217 - val_accuracy: 0.9273
Epoch 10/10
39400/39400 [==============================] - 489s 12ms/step - loss: 0.0219 - accuracy: 0.9927 -
val_loss: 0.3747 - val_accuracy: 0.9260
Accuracy: 92.76%
```

In [34]:

```
#score = model3_3X3.evaluate(x_test, y_test, verbose=0)
#print('Test score:', score[0])
#print('Test accuracy:', score[1])
epochs=10
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs

vy = history_1.history['val_loss']
ty = history_1.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [35]:

```
# creating the second model
embedding_vector_length = 32
model_2 = Sequential()

#Adding Embedding layer
model_2.add(Embedding(5000, embedding_vector_length, input_length=max_review_length))

#Adding 2 LSTM Layers
model_2.add(LSTM(100))

model_2.add(Dropout(0.25))

model_2.add(Dense(50, activation='relu', kernel_initializer='he_normal'))

model_2.add(Dropout(0.25))

model_2.add(Dense(1, activation='sigmoid'))

#Compiling
model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model_2.summary())
#Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in-an-lstm-mode
l
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 600, 32)           160000
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               53200
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 218,301
Trainable params: 218,301
Non-trainable params: 0
_________________________________________________________________
None
```

In [36]:

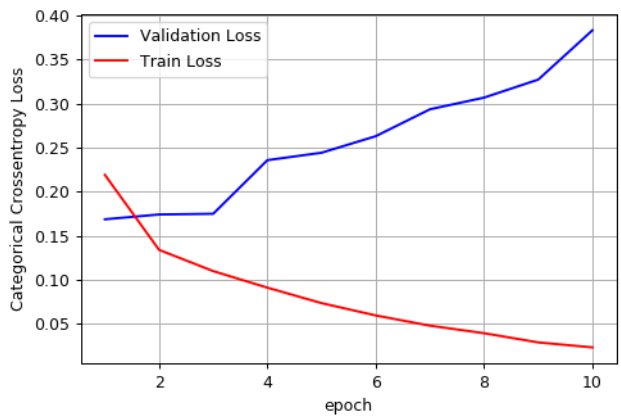
```
#Adding the cross validation data
history_2=model_2.fit(X_train, y_train, nb_epoch=10, batch_size=64,validation_data=(X_cv, y_cv))

# Final evaluation of the model
scores_2 = model_2.evaluate(X_test, y_test, verbose=0,batch_size=64)
print("Accuracy: %.2f%%" % (scores_2[1]*100))
```

```
Train on 39400 samples, validate on 19407 samples
Epoch 1/10
39400/39400 [==============================] - 246s 6ms/step - loss: 0.2190 - accuracy: 0.9157 - v
al_loss: 0.1687 - val_accuracy: 0.9351
Epoch 2/10
39400/39400 [==============================] - 241s 6ms/step - loss: 0.1340 - accuracy: 0.9494 - v
al_loss: 0.1741 - val_accuracy: 0.9363
Epoch 3/10
39400/39400 [==============================] - 240s 6ms/step - loss: 0.1099 - accuracy: 0.9594 - v
al_loss: 0.1749 - val_accuracy: 0.9357
```

```
Epoch 4/10
39400/39400 [==============================] - 241s 6ms/step - loss: 0.0912 - accuracy: 0.9665 - v
al_loss: 0.2357 - val_accuracy: 0.9327
Epoch 5/10
39400/39400 [==============================] - 241s 6ms/step - loss: 0.0736 - accuracy: 0.9738 - v
al_loss: 0.2441 - val_accuracy: 0.9324
Epoch 6/10
39400/39400 [==============================] - 241s 6ms/step - loss: 0.0597 - accuracy: 0.9785 - v
al_loss: 0.2628 - val_accuracy: 0.9274
Epoch 7/10
39400/39400 [==============================] - 241s 6ms/step - loss: 0.0481 - accuracy: 0.9828 - v
al_loss: 0.2933 - val_accuracy: 0.9287
Epoch 8/10
39400/39400 [==============================] - 241s 6ms/step - loss: 0.0395 - accuracy: 0.9864 - v
al_loss: 0.3066 - val_accuracy: 0.9307
Epoch 9/10
39400/39400 [==============================] - 248s 6ms/step - loss: 0.0291 - accuracy: 0.9899 - v
al_loss: 0.3270 - val_accuracy: 0.9279
Epoch 10/10
39400/39400 [==============================] - 258s 7ms/step - loss: 0.0235 - accuracy: 0.9920 - v
al_loss: 0.3829 - val_accuracy: 0.9283
Accuracy: 92.71%
```

In [37]:

```
#score = model3_3X3.evaluate(x_test, y_test, verbose=0)
#print('Test score:', score[0])
#print('Test accuracy:', score[1])
epochs=10
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs

vy = history_2.history['val_loss']
ty = history_2.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [4]:

```
from prettytable import PrettyTable
```

```
x = PrettyTable()

#x.field_names = ["S.NO","ARCHITECTURE","TRAIN LOSS", "TEST LOSS","TRAIN ACCURACY","TEST
ACCURACY"]

sno = [1,2]
architecture = ["Embedding-LSTM-Dropout-LSTM-Dense(Sigmoid)","Embedding-LSTM-Dropout-Dense(Relu)-D
ropout-Dese(Sigmoid)"]
train_loss=['0.0219','0.0235']
test_loss=['0.3747','0.3829']
train_accu=[' 0.9927',' 0.9920']
test_accu=['0.9260','0.9283']

x.add_column("S.NO",sno)
x.add_column("ARCHITECTURE",architecture)
x.add_column("TRAIN LOSS",train_loss)
x.add_column("TEST LOSS",test_loss)
x.add_column("TRAIN ACCURACY",train_accu)
x.add_column("TEST ACCURACY",test_accu)
# Printing the Table
print(x)
```

```
+------+-------------------------------------------------------------+------------+------------+-------
-------+---------------+
| S.NO |                        ARCHITECTURE                         | TRAIN LOSS | TEST LOSS | TRAIN
CCURACY | TEST ACCURACY |
+------+-------------------------------------------------------------+------------+------------+-------
-------+---------------+
|  1   |         Embedding-LSTM-Dropout-LSTM-Dense(Sigmoid)          |   0.0219   |   0.3747   |
0.9927     |     0.9260    |
|  2   | Embedding-LSTM-Dropout-Dense(Relu)-Dropout-Dese(Sigmoid) |   0.0235   |   0.3829   |     0.
9920     |     0.9283    |
+------+-------------------------------------------------------------+------------+------------+-------
-------+---------------+
```

In [ ]: