Sentiment analysis on Amazon-fine-food reviews using LSTM

In [64]:
```
import datetime as dt
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.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras import regularizers
from keras.regularizers import l1
from keras.regularizers import l2
from keras.layers import Dense, Dropout, Flatten
from keras.layers import  GlobalMaxPool1D
from keras.layers.normalization import BatchNormalization
from sklearn import preprocessing
from array import array
from sklearn.preprocessing import LabelEncoder
```

Connecting to the pre-processed sql-ite file

In [3]:
```
#Connecting to the SQL table
con = sqlite3.connect('final.sqlite')

#Reading data from the database

Data = pd.read_sql_query("""
SELECT *
FROM Reviews """,con)
Data.shape

# Drop index column
Data.drop(columns=['index'],inplace=True)
```

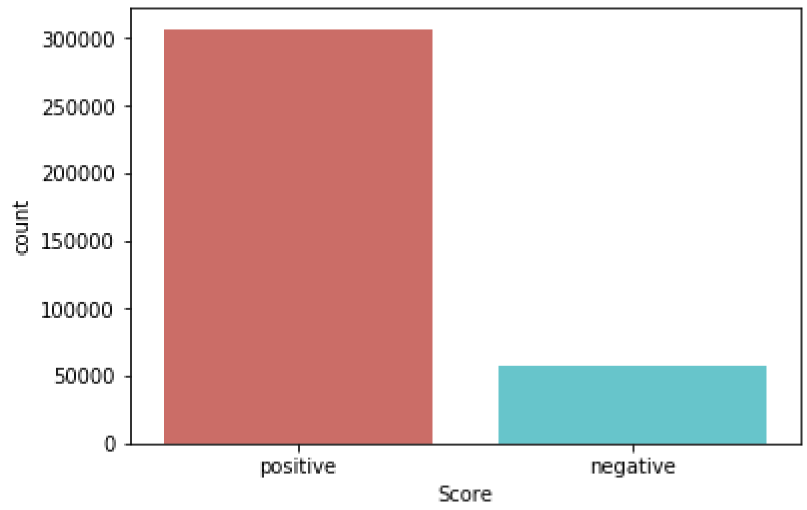
Preparing the data by using its temporal structure

In [4]:
```
# Convert timestamp to datetime.

Data['Time'] = Data[['Time']].applymap(lambda x: dt.datetime.fromtimestamp(x))

#Setting Time column as index of the dataframe
Data.set_index("Time",inplace=True)

#Sampling the above data

Sorted=Data.sort_index()

Sorted.head()
```

Out[4]:

Time	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Sun
1999-10-08 05:30:00	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive	E b educ
1999-10-25 05:30:00	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	positive	This se great sper
1999-12-02 05:30:00	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	positive	Entert I
1999-12-06 05:30:00	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	positive	A n day fa
2000-01-03 05:30:00	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	positive	FANTA

In [5]:
```
polarity=Sorted["Score"]
sns.countplot(x="Score",data=Sorted,palette="hls")
plt.show()
plt.savefig("count_plot")
```




```
<Figure size 432x288 with 0 Axes>
```

Utility function for plotting the error plot of LSTM network

In [35]:
```python
# Plot train and cross validation loss
def plot_train_cv_loss(trained_model, epochs, colors=['b']):
    fig, ax = plt.subplots(1,1)
    ax.set_xlabel('epoch')
    ax.set_ylabel('Categorical Crossentropy Loss')
    x_axis_values = list(range(1,epochs+1))

    validation_loss = trained_model.history['val_loss']
    train_loss = trained_model.history['loss']

    ax.plot(x_axis_values, validation_loss, 'b', label="Validation Loss")
    ax.plot(x_axis_values, train_loss, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()


# Plot weight distribution using violin plot
def plot_weights(model):
    w_after = model.get_weights()

    o1_w = w_after[0].flatten().reshape(-1,1)
    o2_w = w_after[2].flatten().reshape(-1,1)
    out_w = w_after[4].flatten().reshape(-1,1)


    fig = plt.figure(figsize=(10,7))
    plt.title("Weight matrices after model trained\n")
    plt.subplot(1, 3, 1)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=o1_w,color='b')
    plt.xlabel('Hidden Layer 1')

    plt.subplot(1, 3, 2)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=o2_w, color='r')
    plt.xlabel('Hidden Layer 2 ')

    plt.subplot(1, 3, 3)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=out_w,color='y')
    plt.xlabel('Output Layer ')
    plt.show()
```

Processing the data by using Tokeneization and padding in keras

In [31]:
```python
from  keras.preprocessing.text import Tokenizer
# defining the size of the vocabulary 5000
max_Reviws = 5000 #Maximum length of the reviews to be taken
x = Sorted["CleanedText"]
y = polarity

X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size = 0.33, random_state = None)

# label encode the target variable
encoder = preprocessing.LabelEncoder()
train_y = encoder.fit_transform(Y_train)
test_y = encoder.fit_transform(Y_test)

# create the tokenizer
t = Tokenizer(num_words=max_Reviws, split=' ')
# fit the tokenizer on the documents
t.fit_on_texts(X_train)

#Using the Tokenizers API of keras
list_tokenized_train = t.texts_to_sequences(X_train)
list_tokenized_test = t.texts_to_sequences(X_test)
```

In [45]:
```python
t.word_counts
```

```
Out[45]: OrderedDict([('year', 25759),
             ('ago', 6474),
             ('product', 80140),
             ('disappear', 887),
             ('store', 31647),
             ('shelv', 871),
             ('thought', 11510),
             ('mayb', 6371),
             ('downgrad', 37),
             ('season', 6323),
             ('item', 11079),
             ('everi', 15925),
             ('trip', 2220),
             ('groceri', 10346),
             ('shop', 4150),
             ('final', 4338),
             ('given', 3272),
             ('hope', 7613),
             ('ever', 14936),
             ('find', 35781),
             ('search', 3737),
             ('net', 475),
             ('last', 13865),
             ('resort', 302),
             ('found', 23857),
             ('amazon', 40041),
             ('trust', 1358),
             ('level', 2749),
             ('low', 8948),
             ('first', 25381),
             ('could', 19049),
             ('didnt', 14344),
             ('come', 18031),
             ('brown', 4970),
             ('box', 28810),
             ('took', 5575),
             ('chanc', 1938),
             ('base', 4516),
             ('review', 19633),
             ('bought', 21432),
             ('case', 9172),
             ('sure', 13298),
             ('enough', 13791),
             ('one', 78271),
             ('ate', 3490),
             ('even', 33358),
             ('set', 4312),
             ('dont', 38926),
             ('tell', 5614),
             ('give', 24028),
             ('star', 9341),
             ('complet', 4546),
             ('ridicul', 865),
             ('two', 19558),
             ('use', 80004),
             ('sell', 5526),
             ('contain', 13913),
             ('chicken', 11053),
             ('tikka', 73),
             ('masala', 256),
             ('rice', 11272),
             ('theyv', 857),
             ('cut', 5059),
             ('portion', 2036),
             ('ctm', 4),
             ('half', 8626),
             ('replac', 4037),
             ('cheap', 3024),
             ('bland', 2135),
             ('price', 37309),
             ('entir', 3390),
             ('packag', 24316),
             ('gone', 3562),
             ('though', 12328),
             ('get', 57698),
             ('much', 38306),
             ('exact', 5239),
             ('email', 1193),
             ('compani', 9472),
             ('complain', 1426),
             ('claim', 2010),
             ('mani', 17834),
             ('peopl', 10865),
             ('demand', 364),
             ('assum', 1215),
             ('lie', 485),
             ('buy', 45596),
```

```
('entrepreneurship', 1),
('rainforest', 184),
('conserv', 94),
('suggest', 3758),
('itll', 392),
('panko', 99),
('heavier', 218),
('crumb', 1006),
('aesthet', 73),
('sushi', 670),
('chef', 911),
('allow', 2657),
('reseal', 832),
('mean', 4586),
('desir', 1017),
('finish', 4114),
('tasteless', 766),
('pleasant', 3954),
('dens', 729),
('brick', 315),
('boss', 116),
('said', 8488),
('issu', 4877),
('raisin', 2467),
('germani', 523),
('wienerschnitzl', 1),
('jager', 3),
('delight', 3024),
('wienershnitzel', 1),
('aldi', 26),
('abl', 7600),
('obtain', 337),
('boon', 61),
('send', 2463),
('prohibit', 127),
('walmart', 1420),
('folk', 1201),
('charg', 1744),
('slow', 862),
('vendor', 1452),
('earlier', 690),
('caramel', 2669),
('macchiato', 92),
...])
```

In [44]:

```
#Top 25 most frequent occurence of words
count=list(t.word_counts)
print(count[:25])
```

```
['year', 'ago', 'product', 'disappear', 'store', 'shelv', 'thought', 'mayb', 'downgrad', 'season', 'it
em', 'everi', 'trip', 'groceri', 'shop', 'final', 'given', 'hope', 'ever', 'find', 'search', 'net', 'l
ast', 'resort', 'found']
```

In [47]:

```
t.word_index
```

Out[47]:
```
{'like': 1,
 'tast': 2,
 'flavor': 3,
 'good': 4,
 'product': 5,
 'use': 6,
 'one': 7,
 'love': 8,
 'great': 9,
 'tri': 10,
 'tea': 11,
 'coffe': 12,
 'get': 13,
 'make': 14,
 'food': 15,
 'would': 16,
 'buy': 17,
 'time': 18,
 'realli': 19,
 'eat': 20,
 'amazon': 21,
 'order': 22,
 'dont': 23,
 'much': 24,
 'price': 25,
 'also': 26,
 'find': 27,
 'littl': 28,
 'bag': 29,
 'best': 30,
 'dog': 31,
 'even': 32,
 'well': 33,
 'drink': 34,
 'store': 35,
 'ive': 36,
 'better': 37,
 'box': 38,
 'chocol': 39,
 'mix': 40,
 'day': 41,
 'water': 42,
 'sugar': 43,
 'recommend': 44,
 'look': 45,
 'year': 46,
 'sweet': 47,
 'first': 48,
 'want': 49,
 'packag': 50,
 'cup': 51,
 'brand': 52,
 'give': 53,
 'found': 54,
 'purchas': 55,
 'high': 56,
 'think': 57,
 'treat': 58,
 'made': 59,
 'work': 60,
 'way': 61,
 'bought': 62,
 'enjoy': 63,
 'say': 64,
 'need': 65,
 'thing': 66,
 'know': 67,
 'delici': 68,
 'nice': 69,
 'review': 70,
 'differ': 71,
 'two': 72,
 'sinc': 73,
 'add': 74,
 'bit': 75,
 'pack': 76,
 'could': 77,
 'cat': 78,
 'lot': 79,
 'favorit': 80,
 'come': 81,
 'still': 82,
 'mani': 83,
 'keep': 84,
 'perfect': 85,
 'cant': 86,
 'got': 87,
```

```
 'negat': 958,
 'boost': 959,
 'pantri': 960,
 'damag': 961,
 'steak': 962,
 'crispi': 963,
 'liter': 964,
 'kit': 965,
 'yellow': 966,
 'dent': 967,
 'premium': 968,
 'pain': 969,
 'sorri': 970,
 'weird': 971,
 'carbon': 972,
 'play': 973,
 'grab': 974,
 'grey': 975,
 'yeast': 976,
 'eye': 977,
 'stuck': 978,
 'heart': 979,
 'shes': 980,
 'lost': 981,
 'handl': 982,
 'whether': 983,
 'pocket': 984,
 'occasion': 985,
 'remain': 986,
 'anim': 987,
 'broth': 988,
 'contact': 989,
 'head': 990,
 'freez': 991,
 'major': 992,
 'crumbl': 993,
 'scoop': 994,
 'nasti': 995,
 'chop': 996,
 'overwhelm': 997,
 'pull': 998,
 'equal': 999,
 'warn': 1000,
 ...}
```

In [46]:
```
#Top 25 words according to the index position
#for index of words
index=list(t.word_index)
print(index[:25])
```

```
['like', 'tast', 'flavor', 'good', 'product', 'use', 'one', 'love', 'great', 'tri', 'tea', 'coffe', 'g
et', 'make', 'food', 'would', 'buy', 'time', 'realli', 'eat', 'amazon', 'order', 'dont', 'much', 'pric
e']
```

Observations

- Here first a vocabulary is created which contains most frequent 5000 words in the whole dataset.After that by using the keras texts_to_sequences method following steps have taken place:

  1. The frequency of each words which are present in the vocabulary is calculated in a tuple format.
  2. The words are arranged in a dictionary by assigning the index to the most frequent words in the vocabulary.

- But there is still a problem since the length of the reviews are not same throughout the data and batch training cannot be done on this type of data which can slow-up the whole training process.
- One solution is to use the padding concept and make the length of all the reviews same to avoid the above problems and speed up the training time.

In [13]:
```
#Defining the maximum length of the padding
maxlen = 300
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

In [48]:
```
#Sample input of the train data
X_t[1]
```

Out[48]:
```
array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0, 3467,  461,  497,
        234,  413, 1141, 1184,  565,  402,   27,  180,  212,   15, 1270,
        330,   15,   13,    7,  169,    1,    4,  497,  234,  383,    2,
          1,  223,   60])
```

In [27]:
```
#Implementing the elastic-net regularization
reg=regularizers.l1_l2(l1=0.01, l2=0.01)
```

Building the 5-Layer LSTM architecture model

In [ ]:
```
##create the sequential model
embedding_vecor_length = 32 #Total number of inputs size of the embedding layer
model = Sequential()

#Embedding layer
model.add(Embedding(max_Reviws , embedding_vecor_length, input_length=maxlen))
model.add(BatchNormalization())
model.add(Dropout(0.3))

#First LSTM layer
model.add(LSTM(100 , return_sequences=True,bias_regularizer=reg))
model.add(Dropout(0.4))

#Second LSTM layer
model.add(LSTM(80 , return_sequences=True,bias_regularizer=reg))
model.add(Dropout(0.5))

#Third LSTM layer
model.add(LSTM(60 , return_sequences=True,bias_regularizer=reg))
model.add(BatchNormalization())
model.add(Dropout(0.6))

#Fourth LSTM layer
model.add(LSTM(40 , return_sequences=True,bias_regularizer=reg))
model.add(Dropout(0.70))

#Fifth LSTM layer
model.add(LSTM(20 ))
model.add(BatchNormalization())
model.add(Dropout(0.50))

#Final dense layer
model.add(Dense(10, activation="relu"))
model.add(Dropout(0.8))
model.add(Dense(1, activation="sigmoid"))

#Compiling the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

Training the model over the train set

In [32]:
```
batch_size = 192
LSTM_5l=model.fit(X_t, train_y, epochs = 7, batch_size=batch_size, verbose = 2,validation_data=(X_te,
test_y))
```

```
Train on 243994 samples, validate on 120177 samples
Epoch 1/7
 - 5204s - loss: 2.7172 - acc: 0.7912 - val_loss: 0.4748 - val_acc: 0.8428
Epoch 2/7
 - 5260s - loss: 0.4515 - acc: 0.8433 - val_loss: 0.4394 - val_acc: 0.8428
Epoch 3/7
 - 5125s - loss: 0.4375 - acc: 0.8434 - val_loss: 0.4364 - val_acc: 0.8428
Epoch 4/7
 - 5143s - loss: 0.4355 - acc: 0.8434 - val_loss: 0.4362 - val_acc: 0.8428
Epoch 5/7
 - 5157s - loss: 0.4353 - acc: 0.8434 - val_loss: 0.4361 - val_acc: 0.8428
Epoch 6/7
 - 5085s - loss: 0.4353 - acc: 0.8434 - val_loss: 0.4362 - val_acc: 0.8428
Epoch 7/7
 - 5117s - loss: 0.4353 - acc: 0.8434 - val_loss: 0.4362 - val_acc: 0.8428
```

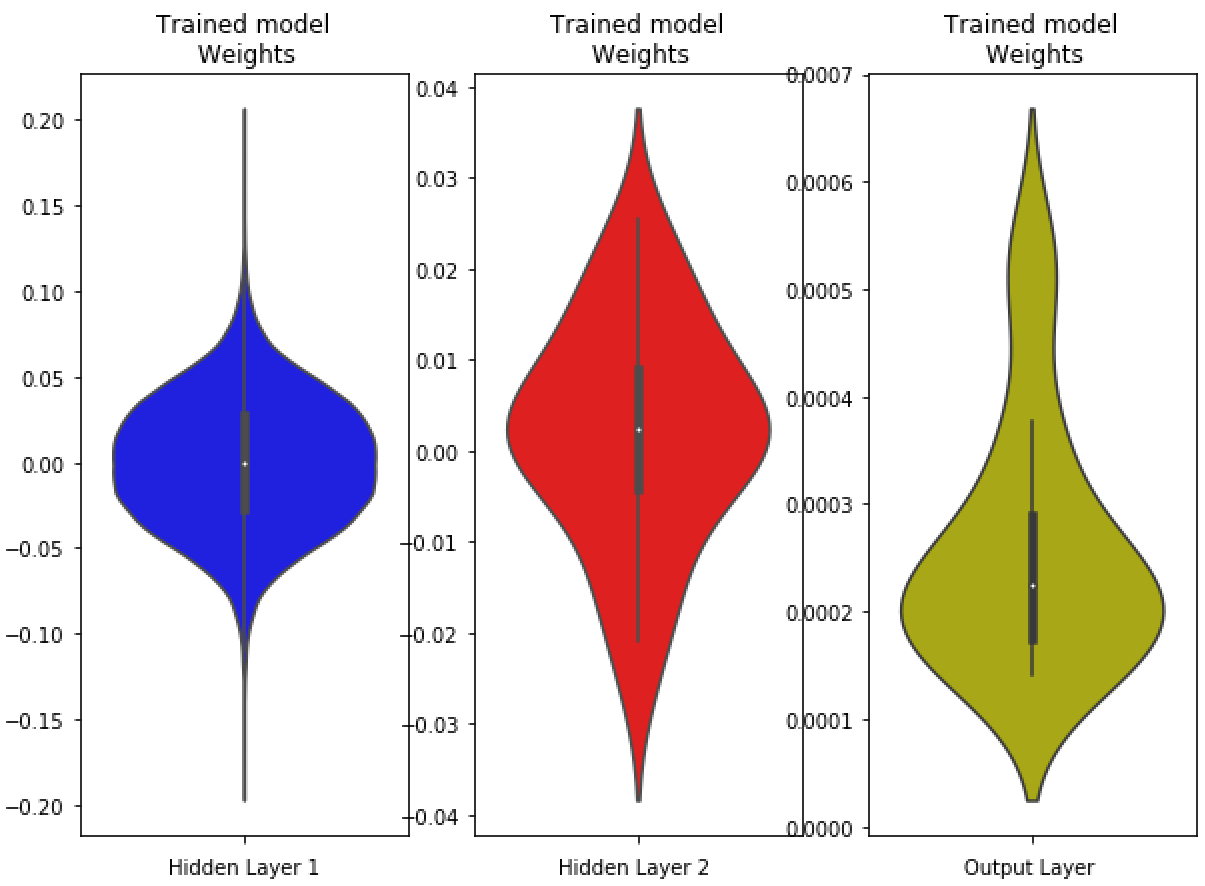
Final accuracy score on the test set

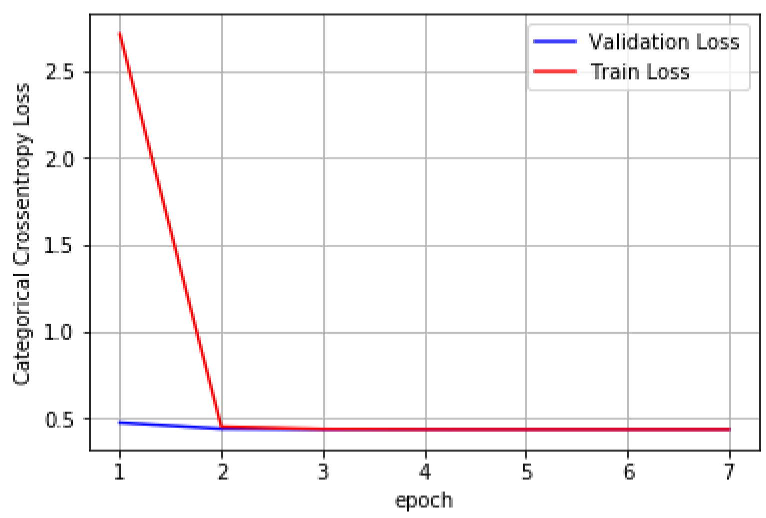
In [34]:
```
score = model.evaluate(X_te, test_y, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.4361937159713344
Test accuracy: 0.8427985388228963
```

Plotting the weights and error-plots

In [38]:
```
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(LSTM_5l, 7)
```





Conclusion

- The input text data must be arranged in a proper format before feeding to the LSTM network.
- Building a proper network architecture is very much important in LSTM models which must be done very carefully.
- The train and test accuracy of the above 5-Layered LSTM model is about 84% which is good for a binary classification model but still can be improved if trained for larger epochs and layers.
- The above LSTM model is not over-fitting which can be seen on the above error plot and the model is very sensible and doing a very good job in classiffying the model properly.
- using the dropout and batch normalization layers reduces the high variance in the network and solves the problem of internal covariance-shift.
- But training an LSTM model is not a trivial task as it took me about 1.5 hrs for each epochs on C.P.U so thats why I trained for a smaller number of epochs.

In [ ]: