LSTM Model on Amazon Fine Food Reviews Dataset

Exercise :

1. Download Amazon Fine Food Reviews dataset from Kaggle. (https://www.kaggle.com/snap/amazon-fine-food-reviews)
2. Get vocabulary for each word in corpus.
3. Also get the frequencies for each word and index them from most frequent to less frequent.
4. Now run LSTM Models on the dataset.
5. Also try 2-layers of LSTM.
6. Also use dropout and batch normalization and plot train-test error vs epochs for each model.
7. Write your observations in English as crisply and unambiguously as possible. Always quantify your results.

Information regarding data set :

1. **Title**: Amazon Fine Food Reviews Data
2. **Sources**: Stanford Network Analysis Project(SNAP)
3. **Relevant Information**: This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~568,454 reviews up to October 2012(Oct 1999 - Oct 2012). Reviews include product and user information, ratings, and a plain text review.
4. **Attribute Information**:
   ProductId - unique identifier for the product
   UserId - unqiue identifier for the user
   ProfileName - name of the user
   HelpfulnessNumerator - number of users who found the review helpful
   HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
   Score - rating between 1 and 5.(rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored)
   Time - timestamp for the review
   Summary - brief summary of the review
   Text - text of the review

Objective :

> It is a 2-class classification task, where we have to analyze, transform and perform LSTM to find the polarity of the dataset.

In [1]:

```
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

import sqlite3
import datetime as dt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from itertools import islice

from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.preprocessing import sequence
from keras.initializers import he_normal
from keras.layers import BatchNormalization, Dense, Dropout, Flatten, LSTM
from keras.layers.embeddings import Embedding
from keras.regularizers import L1L2
from prettytable import PrettyTable
```

```
Using TensorFlow backend.
```

Load Dataset

In [2]:
```
# This dataset is already gone through data deduplication and text preprocessing, so it is approx ~364
K

# For Data Cleaning Steps follow this link -
# ipython notebook - https://drive.google.com/open?id=1JXCva5vXdIPgHbfNdD9sgnySqELoVtpy
# dataset - https://drive.google.com/open?id=1IoDoTT8TfDu53N6cyKg6xVCU-FDPHyIF

# For Text Preporcessing Steps follow this link -
# ipython notebook - https://drive.google.com/open?id=18-AkTzzEhCwM_hflIbDNBMAP-imX4k4i
# dataset - https://drive.google.com/open?id=1SfDwwXFhDpjgtfIE5O_E80SO89xRc8Sa

# Load dataset
def load_review_dataset(do_not_sample=True, sample_count=1):
    # Create connection object to load sqlite dataset
    connection = sqlite3.connect('finalDataSet.sqlite')

    # Load data into pandas dataframe.
    reviews_df = pd.read_sql_query(""" SELECT * FROM Reviews """,connection)

    # Drop index column
    reviews_df = reviews_df.drop(columns=['index'])

    # Sample dataset
    if do_not_sample == False:
        reviews_df = reviews_df.sample(sample_count)

    # Convert timestamp to datetime.
    reviews_df['Time'] = reviews_df[['Time']].applymap(lambda x: dt.datetime.fromtimestamp(x))

    # Sort the data on the basis of time.
    reviews_df = reviews_df.sort_values(by=['Time'])

    return reviews_df


# Load 'finalDataSet.sqlite' in panda's daraframe.
reviews_df = load_review_dataset(do_not_sample = True,sample_count = 1)


# Make CleanedText as a dataset for clustering
cleaned_text = reviews_df['CleanedText'].values

print("Dataset Shape : \n",cleaned_text.shape)

reviews_df['Score'] = reviews_df['Score'].map(lambda x : 1 if x == 'positive' else 0)
reviews_df.head(5)
```

```
Dataset Shape :
 (351237,)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
382	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1
250	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	1
383	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	1
269	374422	B00004CI84	A1048CYU0OV4O8	Judy L. Eans	2	2	1
369	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	23	0

Lets calculate frequencies for each word and index them from most frequent to less frequent.

In [3]:

```
all_words=[]
for sentence in cleaned_text:
    words = sentence.split()
    all_words += words

print("Shape of the data : ",cleaned_text.shape)
print("Number of sentences present in complete dataset : ",len(all_words))

counts = Counter(all_words)
print("Number of unique words present in whole corpus: ",len(counts.most_common()))
vocab_size = len(counts.most_common()) + 1
top_words_count = 5000
sorted_words = counts.most_common(top_words_count)

word_index_lookup = dict()
i = 1
for word,frequency in sorted_words:
    word_index_lookup[word] = i
    i += 1

print()
print("Top 25 words with their frequencies:")
print(counts.most_common(25))
print()
print("Top 25 words with their index:")
print(list(islice(word_index_lookup.items(), 25)))
```

```
Shape of the data :  (351237,)
Number of sentences present in complete dataset :  12901678
Number of unique words present in whole corpus:  93072

Top 25 words with their frequencies:
[('like', 160957), ('tast', 153682), ('flavor', 122605), ('good', 120139), ('love', 111232), ('use', 1
10705), ('product', 110217), ('one', 108864), ('great', 104928), ('tri', 96815), ('tea', 89507), ('cof
fe', 88109), ('get', 80051), ('make', 79567), ('food', 69353), ('would', 67655), ('buy', 63884), ('tim
e', 60622), ('realli', 57989), ('eat', 57247), ('amazon', 55670), ('order', 55535), ('dont', 53855),
('much', 53351), ('price', 53027)]

Top 25 words with their index:
[('like', 1), ('tast', 2), ('flavor', 3), ('good', 4), ('love', 5), ('use', 6), ('product', 7), ('on
e', 8), ('great', 9), ('tri', 10), ('tea', 11), ('coffe', 12), ('get', 13), ('make', 14), ('food', 1
5), ('would', 16), ('buy', 17), ('time', 18), ('realli', 19), ('eat', 20), ('amazon', 21), ('order', 2
2), ('dont', 23), ('much', 24), ('price', 25)]
```

Lets add new column to our existing review dataframe with the index value of the words, which are present in 'CleanedText' columns.

In [4]:
```python
def apply_text_index(row):
    holder = []
    for word in row['CleanedText'].split():
        if word in word_index_lookup:
            holder.append(word_index_lookup[word])
        else:
            holder.append(0)
    return holder


reviews_df['CleanedText_Index'] = reviews_df.apply(lambda row: apply_text_index(row),axis=1)
reviews_df.head(5)
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
382	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1
250	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	1
383	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	1
269	374422	B00004CI84	A1048CYU0OV4O8	Judy L. Eans	2	2	1
369	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	23	0

Split dataset into 70 : 30 split.

In [5]:
```python
# Split data into train and test
x_train, x_test, y_train, y_test = train_test_split(reviews_df['CleanedText_Index'].values,
                                                    reviews_df['Score'],
                                                    test_size=0.3,
                                                    shuffle=False,
                                                    random_state=0)
```

In [6]:
```python
print("Total number words present in first review:\n",len(x_train[1]))
print()
print("List of word indexes present in first review:\n", x_train[1])
print()
```

```
Total number words present in first review:
 20

List of word indexes present in first review:
 [1555, 0, 3692, 2656, 212, 2557, 0, 0, 0, 933, 3049, 0, 0, 0, 4623, 111, 2385, 542, 2843, 1495]
```

Lets apply padding to force every review to have equal length.

In [7]:
```python
max_review_length = 500
x_train = sequence.pad_sequences(x_train, maxlen=max_review_length)
x_test = sequence.pad_sequences(x_test, maxlen=max_review_length)

print("Total number words present in first review after padding:\n",len(x_train[1]))
print()
print("List of word indexes present in first review padding:\n", x_train[1])
print()
```

```
Total number words present in first review after padding:
 500

List of word indexes present in first review padding:
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0 1555    0 3692 2656  212 2557    0    0    0  933
 3049    0    0    0 4623  111 2385  542 2843 1495]
```

Model 1 - with 1 - LSTM Layer

In [8]:

```
# Batch size
batch_size = 192

# Number of time whole data is trained
epochs = 10

# Embedding vector size
embedding_vecor_length = 32

# Bias regularizer value - we will use elasticnet
reg = L1L2(0.01, 0.01)

# 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()
```

In [9]:

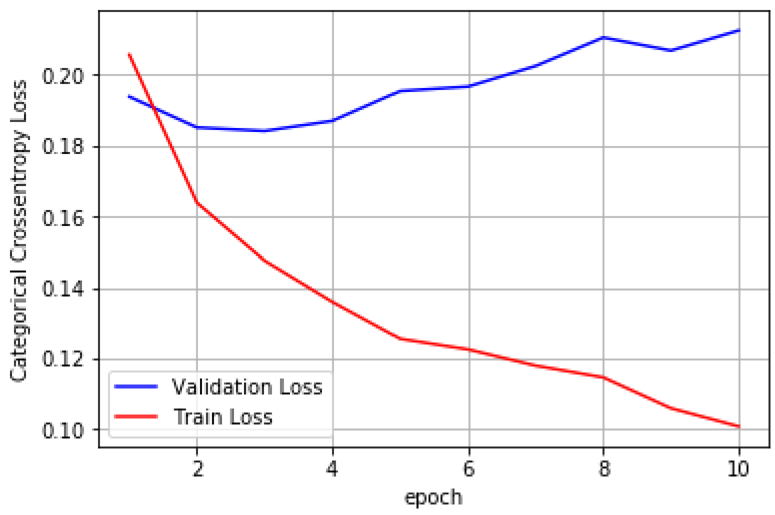
```
# Instantiate sequntial model
model = Sequential()

# Add Embedding Layer
model.add(Embedding(vocab_size, embedding_vecor_length, input_length=max_review_length))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer
model.add(LSTM(100))

# Add dropout
model.add(Dropout(0.20))

# Add Dense Layer
model.add(Dense(1, activation='sigmoid'))

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, np.array(y_train), batch_size = batch_size, epochs = epochs, verbos
e=1, validation_data=(x_test, y_test))
```

```
Model Summary:

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           2978336   
_________________________________________________________________
batch_normalization_1 (Batch (None, 500, 32)           128       
_________________________________________________________________
dropout_1 (Dropout)          (None, 500, 32)           0         
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200     
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101       
=================================================================
Total params: 3,031,765
Trainable params: 3,031,701
Non-trainable params: 64
_________________________________________________________________


Train on 245865 samples, validate on 105372 samples
Epoch 1/10
245865/245865 [==============================] - 1138s 5ms/step - loss: 0.2057 - acc: 0.9195 - val_los
s: 0.1938 - val_acc: 0.9251
Epoch 2/10
245865/245865 [==============================] - 1146s 5ms/step - loss: 0.1639 - acc: 0.9360 - val_los
s: 0.1851 - val_acc: 0.9261
Epoch 3/10
245865/245865 [==============================] - 1304s 5ms/step - loss: 0.1476 - acc: 0.9425 - val_los
s: 0.1841 - val_acc: 0.9277
Epoch 4/10
245865/245865 [==============================] - 1446s 6ms/step - loss: 0.1359 - acc: 0.9470 - val_los
s: 0.1870 - val_acc: 0.9296
Epoch 5/10
245865/245865 [==============================] - 1332s 5ms/step - loss: 0.1255 - acc: 0.9512 - val_los
s: 0.1954 - val_acc: 0.9288
Epoch 6/10
245865/245865 [==============================] - 1400s 6ms/step - loss: 0.1225 - acc: 0.9524 - val_los
s: 0.1967 - val_acc: 0.9271
Epoch 7/10
245865/245865 [==============================] - 1423s 6ms/step - loss: 0.1179 - acc: 0.9541 - val_los
s: 0.2025 - val_acc: 0.9263
Epoch 8/10
245865/245865 [==============================] - 1415s 6ms/step - loss: 0.1146 - acc: 0.9557 - val_los
s: 0.2105 - val_acc: 0.9261
Epoch 9/10
245865/245865 [==============================] - 1405s 6ms/step - loss: 0.1059 - acc: 0.9592 - val_los
s: 0.2069 - val_acc: 0.9279
Epoch 10/10
245865/245865 [==============================] - 1437s 6ms/step - loss: 0.1008 - acc: 0.9616 - val_los
s: 0.2125 - val_acc: 0.9277
```

In [10]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy: {0:.2f}%'.format(score[1]*100))
```

```
Test accuracy: 92.77%
```

In [11]:
```
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```



After 1st epoch we got 92.51% accuracy, and if we train further we starts to overfit, as validation error does not decreses.

Model 2 - With 2 - LSTM Layers

In [12]:

```
%%time
# Instantiate sequntial model
model = Sequential()

# Add Embedding Layer
model.add(Embedding(vocab_size, embedding_vecor_length, input_length=max_review_length))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 1
model.add(LSTM(100,return_sequences=True))

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 2
model.add(LSTM(100))

# Add dropout
model.add(Dropout(0.20))

# Add Dense Layer
model.add(Dense(1, activation='sigmoid'))

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, np.array(y_train), batch_size = batch_size, epochs = epochs, verbos
e=1, validation_data=(x_test, y_test))
```

```
Model Summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           2978336   
_________________________________________________________________
batch_normalization_2 (Batch (None, 500, 32)           128       
_________________________________________________________________
dropout_3 (Dropout)          (None, 500, 32)           0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 500, 100)          53200     
_________________________________________________________________
dropout_4 (Dropout)          (None, 500, 100)          0         
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400     
_________________________________________________________________
dropout_5 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101       
=================================================================
Total params: 3,112,165
Trainable params: 3,112,101
Non-trainable params: 64
_________________________________________________________________


Train on 245865 samples, validate on 105372 samples
Epoch 1/10
245865/245865 [==============================] - 2416s 10ms/step - loss: 0.2031 - acc: 0.9202 - val_lo
ss: 0.1907 - val_acc: 0.9230
Epoch 2/10
245865/245865 [==============================] - 2653s 11ms/step - loss: 0.1621 - acc: 0.9366 - val_lo
ss: 0.1783 - val_acc: 0.9281
Epoch 3/10
245865/245865 [==============================] - 2531s 10ms/step - loss: 0.1457 - acc: 0.9434 - val_lo
ss: 0.1800 - val_acc: 0.9289
Epoch 4/10
245865/245865 [==============================] - 2466s 10ms/step - loss: 0.1343 - acc: 0.9478 - val_lo
ss: 0.1920 - val_acc: 0.9276
Epoch 5/10
245865/245865 [==============================] - 2469s 10ms/step - loss: 0.1246 - acc: 0.9519 - val_lo
ss: 0.1864 - val_acc: 0.9287
Epoch 6/10
245865/245865 [==============================] - 2468s 10ms/step - loss: 0.1166 - acc: 0.9549 - val_lo
ss: 0.1965 - val_acc: 0.9292
Epoch 7/10
245865/245865 [==============================] - 2485s 10ms/step - loss: 0.1095 - acc: 0.9578 - val_lo
ss: 0.2001 - val_acc: 0.9277
Epoch 8/10
245865/245865 [==============================] - 2506s 10ms/step - loss: 0.1041 - acc: 0.9597 - val_lo
ss: 0.2028 - val_acc: 0.9274
Epoch 9/10
245865/245865 [==============================] - 2527s 10ms/step - loss: 0.0990 - acc: 0.9622 - val_lo
ss: 0.1951 - val_acc: 0.9278
Epoch 10/10
245865/245865 [==============================] - 2541s 10ms/step - loss: 0.0952 - acc: 0.9636 - val_lo
ss: 0.2113 - val_acc: 0.9267
Wall time: 6h 57min 46s
```

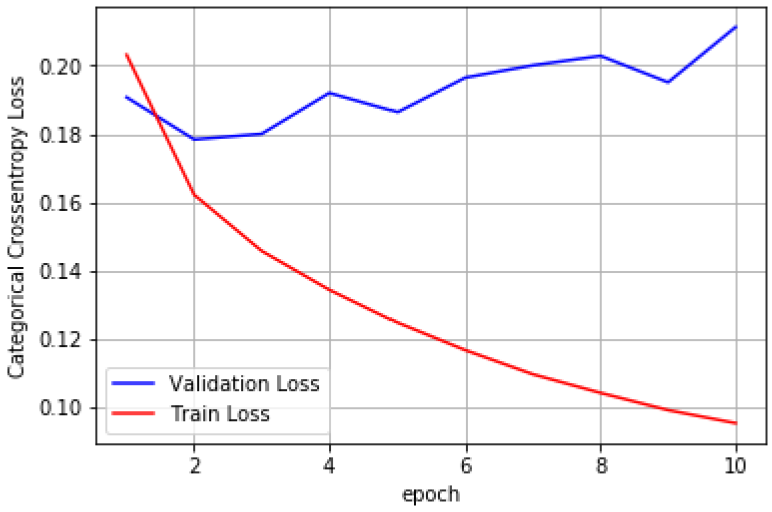
In [13]:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy: {0:.2f}%'.format(score[1]*100))
```

```
Test accuracy: 92.67%
```

In [14]:

```
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```



After 1st epoch we got 92.30% accuracy, and if we train further we starts to overfit, as validation error does not decreses.

Model 3 - With 5 - LSTM Layers

In [9]:
```
# Instantiate sequntial model
model = Sequential()

# Add Embedding Layer
model.add(Embedding(vocab_size, embedding_vecor_length, input_length=max_review_length))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 1
model.add(LSTM(100,return_sequences=True,bias_regularizer=reg))

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 2
model.add(LSTM(80,return_sequences=True,bias_regularizer=reg))

# Add dropout
model.add(Dropout(0.20))

# Add LSTM Layer 3
model.add(LSTM(60,return_sequences=True,bias_regularizer=reg))

# Add dropout
model.add(Dropout(0.30))

# Add LSTM Layer 4
model.add(LSTM(40,return_sequences=True,bias_regularizer=reg))

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.40))

# Add LSTM Layer 5
model.add(LSTM(20))

# Add dropout
model.add(Dropout(0.50))

# Add Dense Layer
model.add(Dense(1, activation='sigmoid'))

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, np.array(y_train), batch_size = batch_size, epochs = epochs, verbos
e=1, validation_data=(x_test, y_test))
```

```
Model Summary:

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           2978336
_________________________________________________________________
batch_normalization_1 (Batch (None, 500, 32)           128
_________________________________________________________________
dropout_1 (Dropout)          (None, 500, 32)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 500, 100)          53200
_________________________________________________________________
dropout_2 (Dropout)          (None, 500, 100)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, 500, 80)           57920
_________________________________________________________________
dropout_3 (Dropout)          (None, 500, 80)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 500, 60)           33840
_________________________________________________________________
dropout_4 (Dropout)          (None, 500, 60)           0
_________________________________________________________________
lstm_4 (LSTM)                (None, 500, 40)           16160
_________________________________________________________________
batch_normalization_2 (Batch (None, 500, 40)           160
_________________________________________________________________
dropout_5 (Dropout)          (None, 500, 40)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 20)                4880
_________________________________________________________________
dropout_6 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 21
=================================================================
Total params: 3,144,645
Trainable params: 3,144,501
Non-trainable params: 144
_________________________________________________________________


Train on 245865 samples, validate on 105372 samples
Epoch 1/10
245865/245865 [==============================] - 4146s 17ms/step - loss: 2.3025 - acc: 0.9107 - val_lo
ss: 0.2145 - val_acc: 0.9182
Epoch 2/10
245865/245865 [==============================] - 3659s 15ms/step - loss: 0.1803 - acc: 0.9323 - val_lo
ss: 0.1930 - val_acc: 0.9227
Epoch 3/10
245865/245865 [==============================] - 3656s 15ms/step - loss: 0.1625 - acc: 0.9389 - val_lo
ss: 0.1959 - val_acc: 0.9268
Epoch 4/10
245865/245865 [==============================] - 3654s 15ms/step - loss: 0.1507 - acc: 0.9431 - val_lo
ss: 0.1977 - val_acc: 0.9220
Epoch 5/10
245865/245865 [==============================] - 3650s 15ms/step - loss: 0.1404 - acc: 0.9475 - val_lo
ss: 0.1885 - val_acc: 0.9283
Epoch 6/10
245865/245865 [==============================] - 3648s 15ms/step - loss: 0.1331 - acc: 0.9504 - val_lo
ss: 0.2017 - val_acc: 0.9257
Epoch 7/10
245865/245865 [==============================] - 3664s 15ms/step - loss: 0.1255 - acc: 0.9525 - val_lo
ss: 0.1880 - val_acc: 0.9279
Epoch 8/10
245865/245865 [==============================] - 3659s 15ms/step - loss: 0.1192 - acc: 0.9559 - val_lo
ss: 0.2126 - val_acc: 0.9275
Epoch 9/10
245865/245865 [==============================] - 3665s 15ms/step - loss: 0.1145 - acc: 0.9572 - val_lo
ss: 0.2138 - val_acc: 0.9257
Epoch 10/10
245865/245865 [==============================] - 3751s 15ms/step - loss: 0.1092 - acc: 0.9594 - val_lo
ss: 0.2201 - val_acc: 0.9252
```

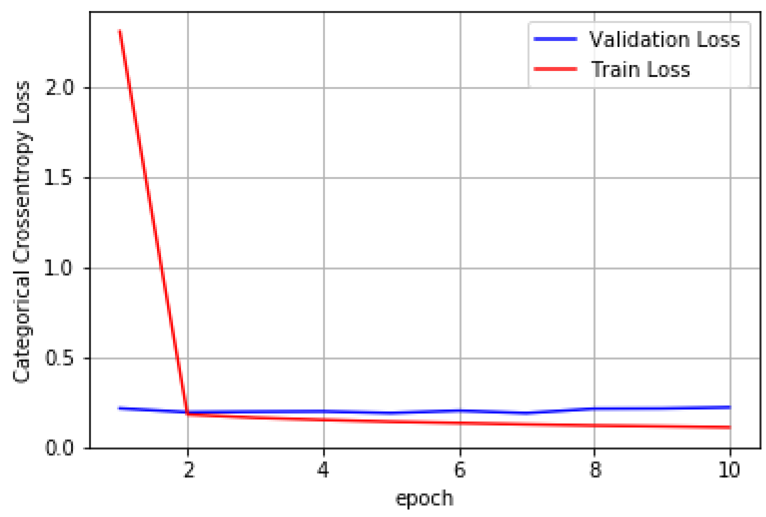
In [10]:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy: {0:.2f}%'.format(score[1]*100))
```

```
Test accuracy: 92.52%
```

In [11]:

```
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```



After 7th epoch, gap between train error and test error increases drastically, so to avoid overfitting we should train our model upto 7th epoch.

We can see that, as we increase number of LSTM layers chance of overfitting the model reduces.

In [2]:
```
# Pretty table instance
ptable = PrettyTable()
ptable.title = "Comparison between different LSTM Models"
ptable.field_names = ['Number of LSTM Layers','Epoch','Testing Accuracy','Does Overfit']
ptable.add_row(["1","10","90.77","Yes"])
ptable.add_row(["2","10","92.67","Yes"])
ptable.add_row(["5","10","92.52","No - Optimal Epoch value is 7"])

# Print pretty table values
print(ptable)
```

```
+---------------------------------------------------------------------------------+
|                      Comparison between different LSTM Models                   |
+-----------------------+-------+------------------+-------------------------------+
| Number of LSTM Layers | Epoch | Testing Accuracy |          Does Overfit         |
+-----------------------+-------+------------------+-------------------------------+
|           1           |   10  |      90.77       |              Yes              |
|           2           |   10  |      92.67       |              Yes              |
|           5           |   10  |      92.52       | No - Optimal Epoch value is 7 |
+-----------------------+-------+------------------+-------------------------------+
```

Observations :

> 1. Tried different LSTM architectures on Amazon Fine Food Review Dataset.
> 2. 'sigmoid' is used as an activation function to develop LSTM network.
> 3. 'Adam' is used as an optimizer to develop LSTM network.
> 4. Introduced batch normalization and dropout in between hidden layers.

Note:

To avoid overfitting we can try below following measures:

We can increase the number of epochs to some reasonable number like 100 - 300 ,

We can introduce 'recurrent_regularizer' on LSTM layer, for different values of L1 or L2 or elasticnet ,

We can also try 'kernel_regularizer' on LSTM layer, for different values of L1 or L2 or elasticnet ,

We can combine CNN + MaxPooling + LSTM and observe the any decrease in validation error.