LSTM - Amazon Fine Food Reviews

May 31, 2019

1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

1.1 Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
pd.set_option('display.max_colwidth', -1)
from matplotlib import pyplot as plt
import seaborn as sns
import keras
from keras.layers import Dense, Embedding, LSTM, Dropout
from keras.models import Sequential
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
import sqlite3
from tqdm import tqdm
import re
from bs4 import BeautifulSoup

from sklearn.model_selection import train_test_split
```

```
Using TensorFlow backend.
```

In [2]:
```
conn = sqlite3.connect('../input/database.sqlite')
filtered_data = pd.read_sql_query(''' SELECT * FROM REVIEWS LIMIT 100000''', conn)

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

def findMinorClassPoints(df):
    posCount = int(df[df['Score']==1].shape[0]);
    negCount = int(df[df['Score']==0].shape[0]);
    if negCount < posCount:
        return negCount
    return posCount

#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

#Performing Downsampling
# samplingCount = findMinorClassPoints(filtered_data)
# postive_df = filtered_data[filtered_data['Score'] == 1].sample(n=5000)
```

```
        # negative_df = filtered_data[filtered_data['Score'] == 0].sample(n=5000)

        # filtered_data = pd.concat([postive_df, negative_df])

        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[2]:

	Id
0	1
1	2
2	3

[3 rows x 10 columns]

1.2 Data Preprocessing

```
In [3]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False

        #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='f
        final.shape

        #Removing the anamolies
        final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]

        #Preprocessing
        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


        preprocessed_reviews = []
```

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

        ## Similartly you can do preprocessing for review summary also.
        def concatenateSummaryWithText(str1, str2):
            return str1 + ' ' + str2

        preprocessed_summary = []
        # tqdm is for printing the status bar
        for sentence in tqdm(final['Summary'].values):
            sentence = re.sub(r"http\S+", "", sentence)
            #sentence = BeautifulSoup(sentence, 'lxml').get_text()
            sentence = decontracted(sentence)
            sentence = re.sub("\S*\d\S*", "", sentence).strip()
            sentence = re.sub('[^A-Za-z]+', ' ', sentence)
            # https://gist.github.com/sebleier/554280
            # sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwo
            preprocessed_summary.append(sentence.strip())

        preprocessed_reviews = list(map(concatenateSummaryWithText, preprocessed_reviews, prepro
        final['CleanedText'] = preprocessed_reviews
        final['CleanedText'] = final['CleanedText'].astype('str')
```

```
100%|| 88461/88461 [00:27<00:00, 3203.76it/s]
100%|| 88461/88461 [00:01<00:00, 56663.24it/s]
```

Segregating the input and output data from the dataset.
We will be using the Cleaned Text i.e preprocessed data from the dataset and score for that text

```
In [4]: X = final['CleanedText']
        y = final['Score']
```

```
In [5]: del final
        del preprocessed_reviews
        del preprocessed_summary
        del sorted_data
        del filtered_data
```

1.2.1 Splitting the data

```
In [6]: X_t, X_test, y_t, y_test = train_test_split(X, y, test_size=0.20, stratify=y, shuffle=Tr
        X_train, X_cv, y_train, y_cv = train_test_split(X_t, y_t, test_size=0.20, stratify=y_t,
        print("Shape of Input  - Train:", X_train.shape)
        print("Shape of Output - Train:", y_train.shape)
        print("Shape of Input  - CV   :", X_cv.shape)
        print("Shape of Output - CV   :", y_cv.shape)
        print("Shape of Input  - Test :", X_test.shape)
        print("Shape of Output - Test :", y_test.shape)
```

```
Shape of Input  - Train: (56614,)
Shape of Output - Train: (56614,)
Shape of Input  - CV   : (14154,)
Shape of Output - CV   : (14154,)
Shape of Input  - Test : (17693,)
Shape of Output - Test : (17693,)
```

1.2.2 Tokenizing the dataset

```
In [7]: 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))
```

```
[9, 46, 6, 2027, 1818, 3, 2502, 3, 1697, 14, 121, 94, 40, 128, 945, 5, 53, 225, 171, 8, 4, 829,
56614
```

1.2.3 Padding the dataset

This is just to give batch input to the RNN

```
In [8]: # truncate and/or pad input sequences
        max_review_length = 1000
        X_train_new = sequence.pad_sequences(X_train_new, maxlen=max_review_length)
        X_cv_new = sequence.pad_sequences(X_cv_new, maxlen=max_review_length)
        X_test_new = sequence.pad_sequences(X_test_new, maxlen=max_review_length)

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

```
(56614, 1000)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    9   46    6 2027 1818    3 2502    3 1697   14  121   94   40  128
  945    5   53  225  171    8    4  829   57    1  313  444    2  311
 2019   16   11   54   14    6    4   26  215   12  382    9   46   53
  118    5  168    3  247    7  551   13  541 1472  178  135 1309    4
 2753 3051   47    4  340   46   81  178  139    8    1    6  823  444
   18    8    1  747   65  655  119   11  630   46  266    9  300    6
    4  229 1063   46   35  808]
```

```
In [9]: # 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()

        n_epochs = 5
        batchsize = 512

        final_output = pd.DataFrame(columns=["Model", "Architecture",
                                             "TRAIN_LOSS", "TEST_LOSS", "TRAIN_ACC", "TEST_ACC"]
```

1.2.4 Model M1 (Embedding -> LSTM -> Output(Sigmoid))

```
In [10]: # create the model
         embed_vector_length = 32
         model = Sequential()
         model.add(Embedding(5000, embed_vector_length, input_length=max_review_length))
         model.add(LSTM(100))
```

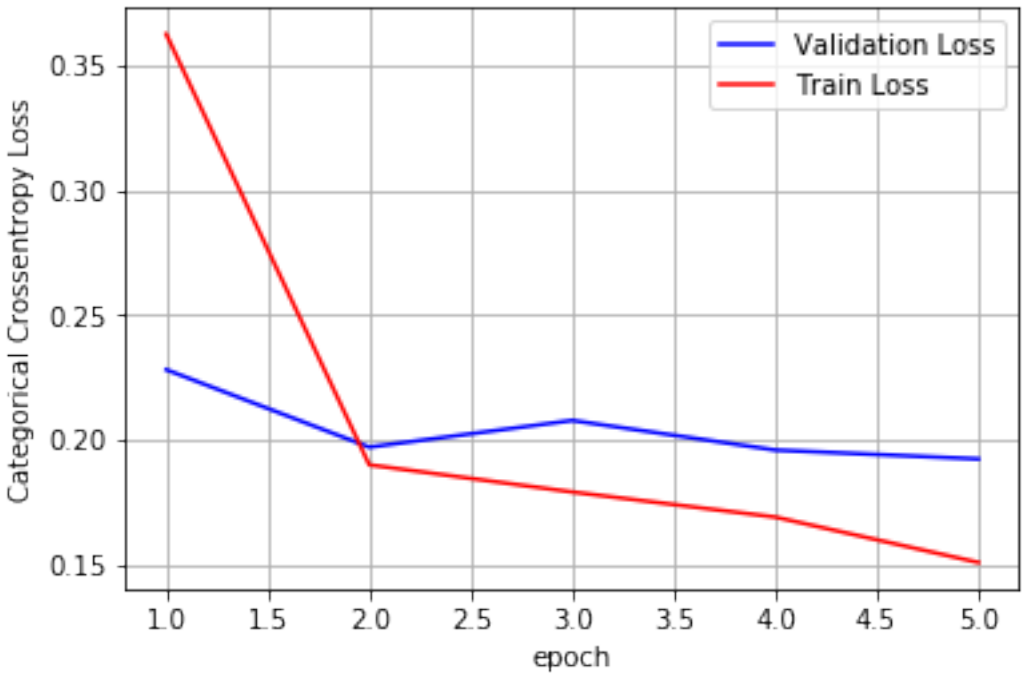
```
        model.add(Dense(1, activation='sigmoid'))
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        print("***********************************************")
        print("Printing the Model Summary")
        print(model.summary())
        print("***********************************************")
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op_de
Instructions for updating:
Colocations handled automatically by placer.
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 32)          160000
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
```

```
In [11]: m_hist = model.fit(X_train_new, y_train, epochs=n_epochs,
                            batch_size=batchsize, verbose=1, validation_data=(X_cv_new, y_cv))

        score = model.evaluate(X_test_new, y_test, batch_size=batchsize)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        final_output = final_output.append({"Model": 1,
                                            "Architecture": 'Embedding-LSTM-Sigmoid',
                                            "TRAIN_LOSS": '{:.5f}'.format(m_hist.history["loss"
                                            "TEST_LOSS": '{:.5f}'.format(score[0]),
                                            "TRAIN_ACC": '{:.5f}'.format(m_hist.history["acc"][
                                            "TEST_ACC": '{:.5f}'.format(score[1])}, ignore_inde

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

        # list of epoch numbers
```

```
        x = list(range(1,n_epochs+1))

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

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py
Instructions for updating:
Use tf.cast instead.
Train on 56614 samples, validate on 14154 samples
Epoch 1/5
56614/56614 [==============================] - 154s 3ms/step - loss: 0.3626 - acc: 0.8625 - val_
Epoch 2/5
56614/56614 [==============================] - 152s 3ms/step - loss: 0.1901 - acc: 0.9211 - val_
Epoch 3/5
56614/56614 [==============================] - 152s 3ms/step - loss: 0.1792 - acc: 0.9263 - val_
Epoch 4/5
56614/56614 [==============================] - 152s 3ms/step - loss: 0.1692 - acc: 0.9311 - val_
Epoch 5/5
56614/56614 [==============================] - 152s 3ms/step - loss: 0.1510 - acc: 0.9393 - val_
17693/17693 [==============================] - 15s 871us/step
Test score: 0.19941727656365546
Test accuracy: 0.9197987909537473
```

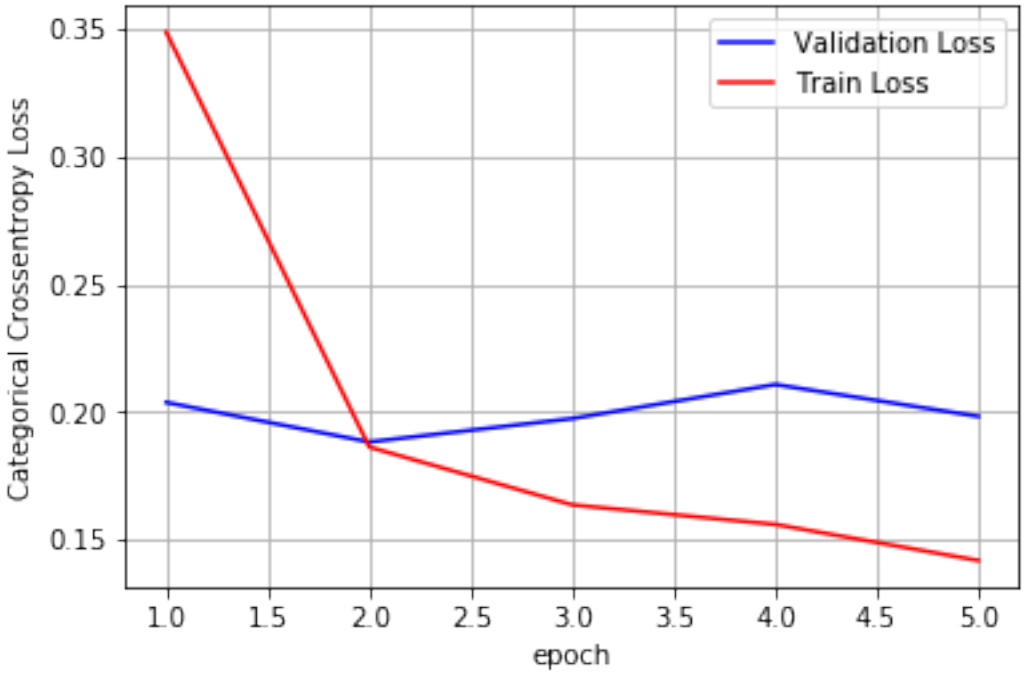


1.2.5 Model M2 (Embedding -> LSTM -> Dropout -> Dense(128-Relu) -> Dropout -> Dense (64-Relu) -> Dropout -> Output(Sigmoid))

```
In [12]: # create the model
         embed_vector_length = 32
         model = Sequential()
         model.add(Embedding(5000, embed_vector_length, input_length=max_review_length))
         model.add(LSTM(100))
         model.add(Dropout(rate=0.5))
         model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
         model.add(Dropout(rate=0.5))
         model.add(Dense(64, activation='relu', kernel_initializer='he_normal'))
         model.add(Dropout(rate=0.5))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         print("***********************************************")
         print("Printing the Model Summary")
         print(model.summary())
         print("***********************************************")
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/tensorflow_backend.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1000, 32)          160000
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               53200
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               12928
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
Total params: 234,449
Trainable params: 234,449
Non-trainable params: 0
_________________________________________________________________
None
```

```
**********************************************


In [13]: m_hist = model.fit(X_train_new, y_train, epochs=n_epochs,
                            batch_size=batchsize, verbose=1, validation_data=(X_cv_new, y_cv))

         score = model.evaluate(X_test_new, y_test, batch_size=batchsize)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"Model": 2,
                                             "Architecture": 'Embedding-LSTM-Dropout-Dense(128-R
                                             "TRAIN_LOSS": '{:.5f}'.format(m_hist.history["loss"
                                             "TEST_LOSS": '{:.5f}'.format(score[0]),
                                             "TRAIN_ACC": '{:.5f}'.format(m_hist.history["acc"][
                                             "TEST_ACC": '{:.5f}'.format(score[1])}, ignore_inde

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

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

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

Train on 56614 samples, validate on 14154 samples
Epoch 1/5
56614/56614 [==============================] - 154s 3ms/step - loss: 0.3486 - acc: 0.8666 - val_
Epoch 2/5
56614/56614 [==============================] - 153s 3ms/step - loss: 0.1862 - acc: 0.9255 - val_
Epoch 3/5
56614/56614 [==============================] - 153s 3ms/step - loss: 0.1636 - acc: 0.9359 - val_
Epoch 4/5
56614/56614 [==============================] - 153s 3ms/step - loss: 0.1559 - acc: 0.9384 - val_
Epoch 5/5
56614/56614 [==============================] - 154s 3ms/step - loss: 0.1418 - acc: 0.9446 - val_
17693/17693 [==============================] - 15s 864us/step
Test score: 0.20497340852693577
Test accuracy: 0.916011982321746
```



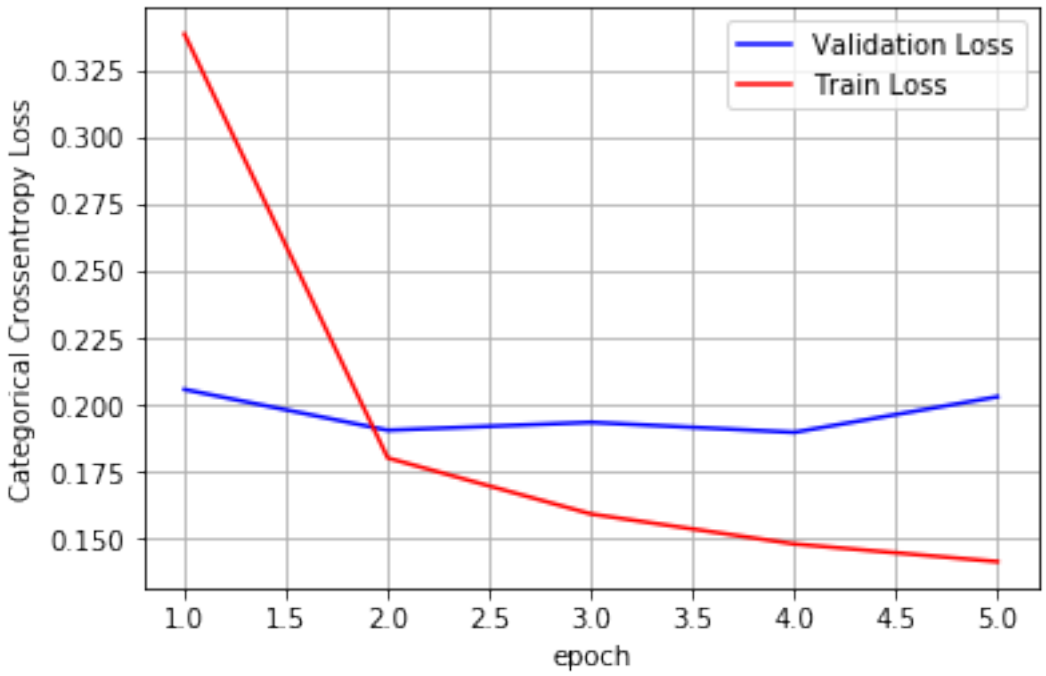
1.2.6 Model M3 (Embedding -> LSTM -> LSTM -> Output(Sigmoid))

```
In [14]: # create the model
         embed_vector_length = 32
         model = Sequential()
         model.add(Embedding(5000, embed_vector_length, input_length=max_review_length))
         model.add(LSTM(100, return_sequences=True))
         model.add(LSTM(100))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         print("***********************************************")
         print("Printing the Model Summary")
         print(model.summary())
         print("***********************************************")
```

```
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 1000, 32)          160000
_________________________________________________________________
lstm_3 (LSTM)                (None, 1000, 100)         53200
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               80400
```

```
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
=================================================================
Total params: 293,701
Trainable params: 293,701
Non-trainable params: 0
_________________________________________________________________
None
**********************************************


In [15]: m_hist = model.fit(X_train_new, y_train, epochs=n_epochs,
                            batch_size=batchsize, verbose=1, validation_data=(X_cv_new, y_cv))

         score = model.evaluate(X_test_new, y_test, batch_size=batchsize)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"Model": 3,
                                             "Architecture": 'Embedding-LSTM-LSTM-Sigmoid',
                                             "TRAIN_LOSS": '{:.5f}'.format(m_hist.history["loss"
                                             "TEST_LOSS": '{:.5f}'.format(score[0]),
                                             "TRAIN_ACC": '{:.5f}'.format(m_hist.history["acc"][
                                             "TEST_ACC": '{:.5f}'.format(score[1])}, ignore_inde

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

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

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

Train on 56614 samples, validate on 14154 samples
Epoch 1/5
56614/56614 [==============================] - 307s 5ms/step - loss: 0.3385 - acc: 0.8720 - val_
Epoch 2/5
56614/56614 [==============================] - 306s 5ms/step - loss: 0.1803 - acc: 0.9266 - val_
Epoch 3/5
56614/56614 [==============================] - 306s 5ms/step - loss: 0.1594 - acc: 0.9355 - val_
Epoch 4/5
56614/56614 [==============================] - 306s 5ms/step - loss: 0.1482 - acc: 0.9407 - val_
Epoch 5/5
56614/56614 [==============================] - 306s 5ms/step - loss: 0.1416 - acc: 0.9437 - val_
17693/17693 [==============================] - 31s 2ms/step
```

```
Test score: 0.2099661622818616
Test accuracy: 0.9195727120015537
```



2 Conclusion

```
In [16]: final_output

Out[16]:   Model   ...    TEST_ACC
         0  1      ...     0.91980
         1  2      ...     0.91601
         2  3      ...     0.91957

         [3 rows x 6 columns]
```

Here,

The dataset which we used is Amazon fine food reviews dataset. There are a couple different models that we tried - * Model 1 was having architecture with one LSTM layer. * Model 2 was having architecture with one LSTM layer, intermediate dropouts set to 0.5 and 2 dense hidden layers with ReLU activation * Model 3 was having architecture with 2 LSTM layers.

Conclusion that can be drawn from the above models is that All the models are performing great in terms of execution. All of them are converging very faster.

Though Model 3 with 2 LSTM layers converges a little bit faster but it is requiring more training time.