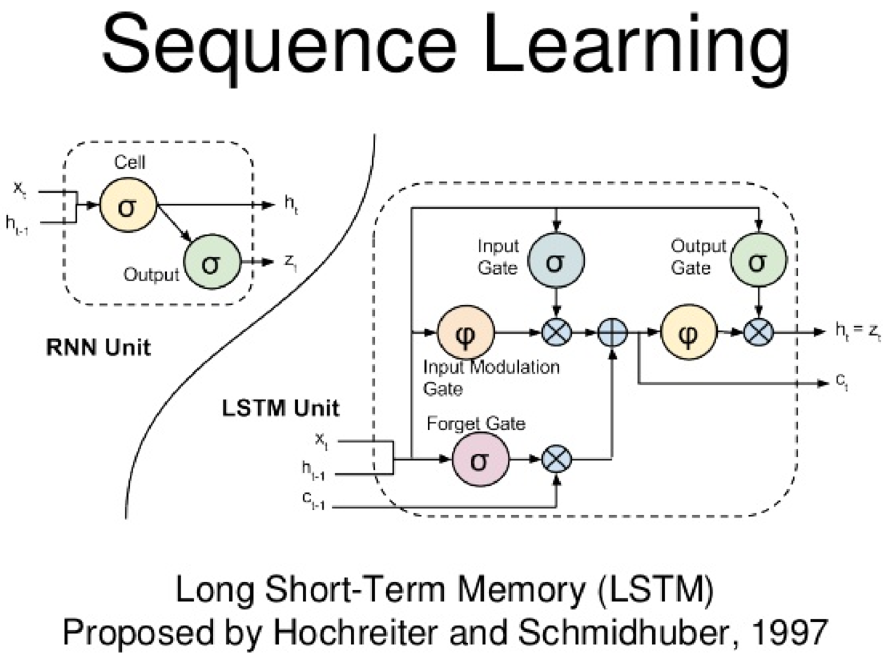
RNN MODEL ON AMAZON FINE FOOD REVIEWS DATASET

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

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.It consist of data collected from past many years. This dataset consist of approx 550k reviews.



SNIPPET

1. Calculated the frequency of each word in vocabulary.
2. Sorted the vocabulary by the rank.
3. Applied LSTM with 1-Layer & 2-Layer on dataset.
4. Plotted epoch vs losses.
5. Conclusion based on the obtained results.

DATA INFORMATION

- 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

9. Summary - brief summary of the review
10. Text - text of the review

OBJECTIVE

Convert the data according to rank and then predict the polarity of Reviews in Amazon fine food dataset using LSTM model with 1-layer and 2-layer respectively.

IMPORTING

In [69]:

```
import sqlite3
import re
import pickle
import numpy as np
import pandas as pd
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.pyplot as plt
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 sklearn.model_selection import train_test_split
from keras.preprocessing import sequence
```

FUNCTIONS

1. CLEANING

In [65]:

```
def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

2. STORING IN LIST

In [ ]:

```
def LOW(l):
    i=0
    list_of_sent=[] # list to store all the lists.
    for sent in l:
        filtered_sentence=[] # list to store each review.
        for w in sent.split():
            for cleaned_words in cleanpunc(w).split():
                if(cleaned_words.isalpha()):
                    filtered_sentence.append(cleaned_words.lower())
                else:
                    continue
        list_of_sent.append(filtered_sentence)
    return list_of_sent
```

3. USING PICKLE

In [51]:

```
'''
These functions are used to save and retrieve the information and use it afterwards for future ref
erence.
'''

# Method to Save the data.
def save(o,f):
    op=open(f+".p","wb")
    pickle.dump(o,op)

# Method to retrieve the data.
def retrieve(f):
    op=open(f+".p","rb")
    ret=pickle.load(op)
    return ret
```

4. PLOTTING TRAIN VS VAL LOSS

In [ ]:

```
def Plot(err):
    x = list(range(1,11))
    v_loss = err.history['val_loss']
    t_loss = err.history['loss']
    plt.plot(x, v_loss, '-b', label='Validation Loss')
    plt.plot(x, t_loss, '-r', label='Training Loss')
    plt.legend(loc='center right')
    plt.xlabel("EPOCHS",fontsize=15, color='black')
    plt.ylabel("Train Loss & Validation Loss",fontsize=15, color='black')
    plt.title("Train vs Validation Loss on Epoch's" ,fontsize=15, color='black')
    plt.show()
```

LOADING DATA

In [90]:

```
con = sqlite3.connect('./database.sqlite') # making a connection with sqlite

""" Assembling data from Reviews where score is not 3 as 3 will be a neutral score so we cant deci
de the polarity
 based on a score of 3.here, score of 1&2 will be considered as negative whereas score of 4&5 will
be considered as
     positive.
"""
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)

# function to map the polarity

def polarity(x):
    if x < 3:
        return 0
    return 1

filtered_data['Score']=filtered_data['Score'].map(polarity)
```

DATA PRE-PROCESSING

In [91]:

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

In [92]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print("Dimension of dataset - : ",final.shape,"\n")

#How many positive and negative reviews are present in our dataset?
print("________________________ Frequency of positive and negative reviews
_________________________")
print(final['Score'].value_counts())
```

```
Dimension of dataset - :  (364171, 10)

________________________ Frequency of positive and negative reviews _________________________
1    307061
0     57110
Name: Score, dtype: int64
```

SAMPLING DATA

In [93]:

```
# Taking 60k reviews
final = final.sample(60000)
```

SORTING

In [94]:

```
final.sort_values('Time',inplace=True)
```

In [95]:

```
print("Dimension of dataset - : ",final.shape,"\n")

#How many positive and negative reviews are present in our dataset?
print("________________________ Frequency of positive and negative reviews
_________________________")
print(final['Score'].value_counts())
```

```
Dimension of dataset - :  (60000, 10)

________________________ Frequency of positive and negative reviews _________________________
1    50687
0     9313
Name: Score, dtype: int64
```

CONVERTING THE DATA

In [96]:

```
total=[]
for i in range(60000):
    l1=final['Text'].values[i]
    l2=str(l1)
    total.append(l2)

total = LOW(total)
```

In [97]:

```
all_=[]
vocab=[]
Vocab=[]

for i in total:
    all_.extend(i)

for i in all_ :
```

```
for i in all_:
    c=0
    if i not in vocab:
        vocab.append(i)
        c = all_.count(i)
        Vocab.append((i,c))
    else:
        pass
```

VOCABULARY

In [98]:

```
l1 = sorted(Vocab,reverse=True, key=lambda x:x[1])
l2 = sorted(Vocab,reverse=False, key=lambda x:x[1])
```

In [99]:

```
mapped1 =[]
mapped2 =[]

for i in range(len(l1)):
    mapped1.append(l1[i][0])

for i in range(len(l2)):
    mapped2.append(l2[i][0])

keys=list(range(1,len(l1)+1))

data1 = dict(zip(mapped1, keys))
data2 = dict(zip(mapped2, keys))
```

In [110]:

```
'''
    Printing the vocabulary according to the frequency of the words.
    eg. the was used most of the times and queen is used least.
'''
wo= WordCloud(width = 2000, height = 1000)
wo.generate_from_frequencies(data2)
plt.figure(figsize=(20,10))
plt.imshow(wo, interpolation='bilinear')
plt.axis("off")
plt.show()
print("\n")
print("____________________ SIZE OF VOCABULARY _______________________")
print(len(vocab))
```

```
____________________ SIZE OF VOCABULARY ______________________
44579
```

CONVERTING ACCORDING TO RANK

In [115]:

```python
print("_______________________ FIRST REVIEW BEFORE CONVERTING ________________\n")
print(total[0])
```

```
_______________________ FIRST REVIEW BEFORE CONVERTING ________________

['this', 'witty', 'little', 'book', 'makes', 'my', 'son', 'laugh', 'at', 'loud', 'i', 'recite', 'i
t', 'in', 'the', 'car', 'as', 'were', 'driving', 'along', 'and', 'he', 'always', 'can', 'sing', 't
he', 'refrain', 'hes', 'learned', 'about', 'whales', 'india', 'drooping', 'i', 'love', 'all', 'the
', 'new', 'words', 'this', 'book', 'introduces', 'and', 'the', 'silliness', 'of', 'it', 'all', 'th
is', 'is', 'a', 'classic', 'book', 'i', 'am', 'willing', 'to', 'bet', 'my', 'son', 'will',
'still', 'be', 'able', 'to', 'recite', 'from', 'memory', 'when', 'he', 'is', 'in', 'college']
```

In [116]:

```python
for i in range(len(total)):
    for j in range(len(total[i])):
        rank = data1.get(total[i][j])
        total[i][j]=rank
```

In [117]:

```python
print("_______________________ FIRST REVIEW AFTER CONVERSION ________________\n")
print(total[0])
```

```
_______________________ FIRST REVIEW AFTER CONVERSION ________________

[9, 16916, 75, 1556, 161, 12, 526, 4497, 31, 4865, 2, 19954, 6, 10, 1, 1511, 21, 74, 3201, 599, 3,
103, 165, 43, 6863, 1, 12626, 990, 1345, 62, 19955, 2140, 19956, 2, 50, 40, 1, 246, 1888, 9, 1556,
15022, 3, 1, 25597, 7, 6, 40, 9, 8, 4, 1531, 1556, 2, 90, 1917, 5, 2391, 12, 526, 52, 137, 29,
346, 5, 19954, 41, 3588, 45, 103, 8, 10, 1998]
```

SPLITTING DATA INTO TRAIN & TEST

In [118]:

```python
f = final['Score'].tolist()
```

In [119]:

```python
'''
This function is used to split that data into train and test.
It uses the function to split it into 70-30 %.
It does not shuffle so the data is distributed sequentially.
'''
x_train, x_test, y_train, y_test = train_test_split(total,f,test_size=0.3,shuffle=False) # Splittin
g it in 70-30 without shuffling.
```

In [120]:

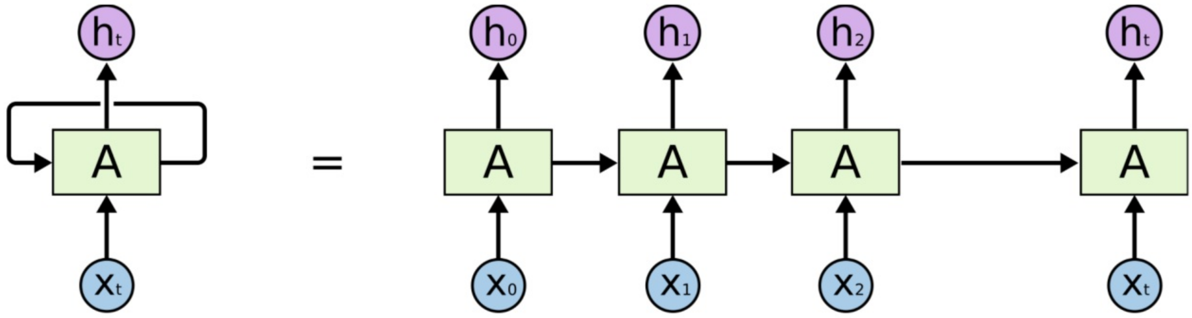
```python
print("-----------------------TRAIN DATA------------------------------------")
print(len(x_train))
print(len(y_train))
print("---------------------------------------------------------------------")
print("\n-----------------------TEST DATA------------------------------------")
print(len(x_test))
print(len(y_test))
```

```
-----------------------TRAIN DATA------------------------------------
```

```
------------------------TRAIN DATA------------------------------------
42000
42000
-----------------------------------------------------------------------

------------------------TEST DATA--------------------------------------
18000
18000
```

LSTM MODEL



PADDING

In [121]:

```
X_train = sequence.pad_sequences(x_train, maxlen = 700)
X_test = sequence.pad_sequences(x_test, maxlen = 700)
```

In [123]:

```
print("------------------------TRAIN DATA------------------------------------")
print(X_train.shape)
print(len(y_train))
print("-----------------------------------------------------------------------")
print("\n------------------------TEST DATA--------------------------------------")
print(X_test.shape)
print(len(y_test))
```

```
------------------------TRAIN DATA------------------------------------
(42000, 700)
42000
-----------------------------------------------------------------------

------------------------TEST DATA--------------------------------------
(18000, 700)
18000
```

DEFINING MODEL

1 - LAYER LSTM

In [124]:

```
'''
    In the embedding layer we put the total vocabulary as first parameter followed by output_dim
    and then the input length which we obtained after padding.
'''
```

```
model = Sequential()
model.add(Embedding(44580, 32, input_length = 700))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))

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

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_4 (Embedding)      (None, 700, 32)           1426560   
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               53200     
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101       
=================================================================
Total params: 1,479,861
Trainable params: 1,479,861
Non-trainable params: 0
_________________________________________________________________
None
```

FITTING THE MODEL

In [130]:

```
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_data=(X_test, y_test))
```

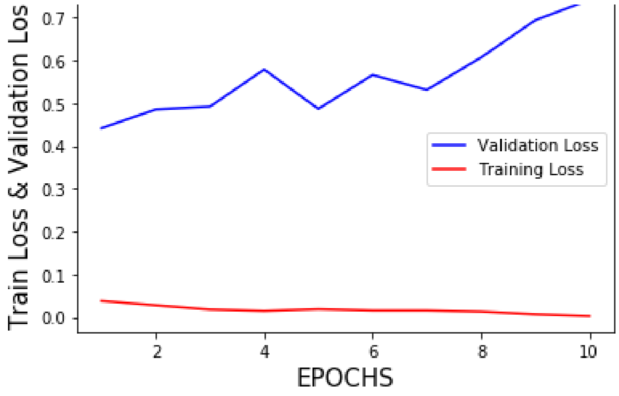
```
Train on 42000 samples, validate on 18000 samples
Epoch 1/10
42000/42000 [==============================] - 580s 14ms/step - loss: 0.0382 - acc: 0.9874 - val_l
oss: 0.4414 - val_acc: 0.9006
Epoch 2/10
42000/42000 [==============================] - 544s 13ms/step - loss: 0.0277 - acc: 0.9907 - val_l
oss: 0.4852 - val_acc: 0.9025
Epoch 3/10
42000/42000 [==============================] - 560s 13ms/step - loss: 0.0180 - acc: 0.9941 - val_l
oss: 0.4918 - val_acc: 0.9003
Epoch 4/10
42000/42000 [==============================] - 550s 13ms/step - loss: 0.0147 - acc: 0.9956 - val_l
oss: 0.5784 - val_acc: 0.8998
Epoch 5/10
42000/42000 [==============================] - 543s 13ms/step - loss: 0.0189 - acc: 0.9940 - val_l
oss: 0.4866 - val_acc: 0.8938
Epoch 6/10
42000/42000 [==============================] - 555s 13ms/step - loss: 0.0157 - acc: 0.9953 - val_l
oss: 0.5656 - val_acc: 0.8968
Epoch 7/10
42000/42000 [==============================] - 549s 13ms/step - loss: 0.0157 - acc: 0.9948 - val_l
oss: 0.5310 - val_acc: 0.8974
Epoch 8/10
42000/42000 [==============================] - 648s 15ms/step - loss: 0.0131 - acc: 0.9961 - val_l
oss: 0.6062 - val_acc: 0.8906
Epoch 9/10
42000/42000 [==============================] - 1071s 25ms/step - loss: 0.0068 - acc: 0.9981 - val_
loss: 0.6934 - val_acc: 0.8974
Epoch 10/10
42000/42000 [==============================] - 1071s 26ms/step - loss: 0.0027 - acc: 0.9994 - val_
loss: 0.7410 - val_acc: 0.8966
```

TRAIN VS VAL LOSS

In [177]:

```
Plot(history)
```

Train vs Validation Loss on Epoch's



Here we can see clearly that our model is overfitting.

EVALUATING THE MODEL

In [178]:

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

```
Accuracy: 89.66%
```

2 - LAYER LSTM

In [181]:

```
'''

    If we have to apply 2 layers in LSTM then the output of the above layer must be
    in a 3-dimensional space and by applying return_sequence=True that is achieved.

'''
model = Sequential()
model.add(Embedding(44580, 32, input_length = 700))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(30))
model.add(Dense(1, activation='sigmoid'))

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

# printing the structure of the model.
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_7 (Embedding)      (None, 700, 32)           1426560   
_________________________________________________________________
lstm_9 (LSTM)                (None, 700, 50)           16600     
_________________________________________________________________
lstm_10 (LSTM)               (None, 30)                9720      
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 31        
=================================================================
Total params: 1,452,911
Trainable params: 1,452,911
Non-trainable params: 0
_________________________________________________________________
None
```

FITTING THE MODEL

In [182]:

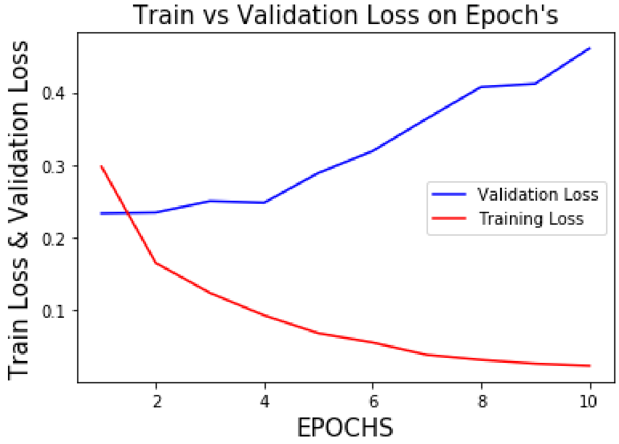
```
history1 = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_data=(X_test, y_test))
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/10
42000/42000 [==============================] - 621s 15ms/step - loss: 0.2977 - acc: 0.8846 - val_l
oss: 0.2334 - val_acc: 0.9068
Epoch 2/10
42000/42000 [==============================] - 474s 11ms/step - loss: 0.1651 - acc: 0.9371 - val_l
oss: 0.2345 - val_acc: 0.9115
Epoch 3/10
42000/42000 [==============================] - 475s 11ms/step - loss: 0.1238 - acc: 0.9544 - val_l
oss: 0.2502 - val_acc: 0.9139
Epoch 4/10
42000/42000 [==============================] - 474s 11ms/step - loss: 0.0928 - acc: 0.9665 - val_l
oss: 0.2480 - val_acc: 0.9054
Epoch 5/10
42000/42000 [==============================] - 482s 11ms/step - loss: 0.0682 - acc: 0.9773 - val_l
oss: 0.2890 - val_acc: 0.9086
Epoch 6/10
42000/42000 [==============================] - 476s 11ms/step - loss: 0.0555 - acc: 0.9809 - val_l
oss: 0.3191 - val_acc: 0.8999
Epoch 7/10
42000/42000 [==============================] - 474s 11ms/step - loss: 0.0384 - acc: 0.9880 - val_l
oss: 0.3639 - val_acc: 0.9034
Epoch 8/10
42000/42000 [==============================] - 474s 11ms/step - loss: 0.0316 - acc: 0.9902 - val_l
oss: 0.4074 - val_acc: 0.9021
Epoch 9/10
42000/42000 [==============================] - 474s 11ms/step - loss: 0.0263 - acc: 0.9912 - val_l
oss: 0.4120 - val_acc: 0.9019
Epoch 10/10
42000/42000 [==============================] - 475s 11ms/step - loss: 0.0233 - acc: 0.9932 - val_l
oss: 0.4605 - val_acc: 0.8973
```

TRAIN VS VAL LOSS

In [183]:

```
Plot(history1)
```

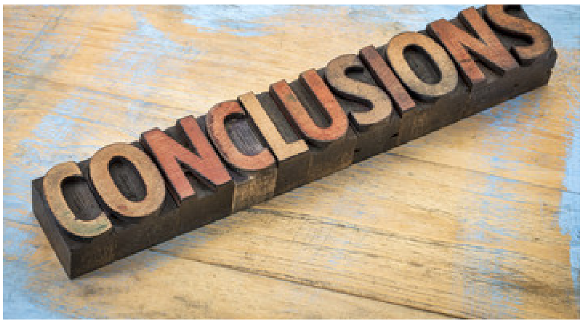


EVALUATING THE MODEL

In [184]:

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

```
Accuracy: 89.73%
```



1. In the first model we had obtained an accuracy of 89.66%.

2. In the second model the accuracy obtained was 89.73%.

3. In the first model we can see that the model is overfitting as the train loss and validation loss are seprated by big margin.

4. We can analyze from the results that LSTM works very good on data as in 1st epoch it was giving a training loss of 0.0382 which is a very big deal .