Assignment 3 - Problem 3

Note: This same notebook serves as our report. Suitable comments and explanations with visualizations were prodcued where and when needed. A pdf is also provided along with this notebook.

Sentiment Analysis

- Create a Neural Network to classify reviews from the IMDB movie review dataset as positive or negative.

In [2]:
```
#Importing necessary libraries

import pandas as pd
import numpy as np
import glob2
import nltk
import warnings
warnings.filterwarnings('ignore')
import re
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, LSTM,Embedding, SpatialDropout1D, Flatten, Dropout,SimpleRNN
from keras.models import Sequential
from sklearn.metrics import accuracy_score
from IPython.display import Image
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

We are using Golb2 library for reading all the files present in train and test folders. This library has the ability to capture the text matched by glob patterns, and return those matches alongside the filenames.

In [3]:
```
#Reading all the positive review files present in the train data folder

read_pos_files=glob2.glob(r'C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\train\pos\*.t

#Writing all the positive reviews present in the train data folder to the file result_pos.txt with delimiter as "\n"

with open(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\train\result_pos.txt", "w",en
    for f in read_pos_files:
        with open(f, "r",encoding="utf8") as infile:
            outfile.write(infile.read()+"\n")
```

In [4]:
```
#Reading all the negative review files present in the train data folder

read_neg_files=glob2.glob(r'C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\train\neg\*.t

#Writing all the negative reviews present in the train data folder to the file result_neg.txt with delimiter as "\n"

with open(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\train\result_neg.txt", "w",en
    for f in read_neg_files:
        with open(f, "r",encoding="utf8") as infile:
            outfile.write(infile.read()+"\n")
```

In [5]:
```
#Reading all the delimted positive reviews present in the file result_pos.txt as a dataframe

x_pos=pd.read_csv(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\train\result_pos.txt"

#Converting the dataframe to a list

x_pos_list = x_pos[0].tolist()

#Reading all the delimted negative reviews present in the file result_neg.txt as a dataframe

x_neg=pd.read_csv(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\train\result_neg.txt"

#Converting the dataframe to a list
x_neg_list = x_neg[0].tolist()
```

In [6]:
```
#Validating the number of positive and negative reviews

len(x_pos_list),len(x_neg_list)
```

Out[6]:
```
(12500, 12500)
```

Data Preprocessing on Train Data

Below are the preprocessing steps performed on the data:

1. Converting the text to lower case, removing new lines within a sentence, alphanumeric words, text in <>, http links, characters that are not alphabets, extra spaces.
2. Normalized the words in the corpus by trying to convert all of the different forms of a given word into one. We have performed Lemmatization instead of stemming since stemming just removes the last few characters of a word, often leading to incorrect meanings and spelling. Lemmatization considers the context and converts the word to its meaningful base form.
3. Removed the stop words.

In [7]:

```
x_pos_list1=[]
for i in range(len(x_pos_list)):
    x_pos_list[i] = x_pos_list[i].lower()
    text = re.sub('\n', ' ', x_pos_list[i])
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text=re.sub('[^abcdefghijklmnopqrstuvwxyz\s]', '',text)
    text = re.sub(r'\s+',' ',text)
    x_pos_list1.append(text)

lemmatizer = WordNetLemmatizer()
for i in range(len(x_pos_list1)):
    words = nltk.word_tokenize(x_pos_list1[i])
    words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
    x_pos_list1[i] = ' '.join(words)

x_neg_list1=[]
for i in range(len(x_neg_list)):
    x_neg_list[i] = x_neg_list[i].lower()
    text = re.sub('\n', ' ', x_neg_list[i])
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text=re.sub('[^abcdefghijklmnopqrstuvwxyz\s]', '',text)
    text = re.sub(r'\s+',' ',text)
    x_neg_list1.append(text)

lemmatizer = WordNetLemmatizer()
for i in range(len(x_neg_list1)):
    words = nltk.word_tokenize(x_neg_list1[i])
    words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
    x_neg_list1[i] = ' '.join(words)
```

In [8]:

```
x_train_list = x_pos_list1+x_neg_list1
len(x_train_list)
```

Out[8]: 25000

In [9]:

```
#Reading all the positive review files present in the test data folder

read_pos_files=glob2.glob(r'C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\test\pos\*.tx

#Writing all the positive reviews present in the test data folder to the file result_pos.txt with delimiter as "\n"

with open(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\test\result_pos.txt", "w",enc
    for f in read_pos_files:
        with open(f, "r",encoding="utf8") as infile:
            outfile.write(infile.read()+"\n")
```

In [10]:

```
#Reading all the negative review files present in the test data folder

read_neg_files=glob2.glob(r'C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\test\neg\*.tx

#Writing all the negative reviews present in the test data folder to the file result_neg.txt with delimiter as "\n"

with open(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\test\result_neg.txt", "w",enc
    for f in read_neg_files:
        with open(f, "r",encoding="utf8") as infile:
            outfile.write(infile.read()+"\n")
```

In [11]:

```
#Reading all the delimted positive reviews present in the file result_pos.txt as a dataframe

x_pos=pd.read_csv(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\test\result_pos.txt",

#Converting the dataframe to a list

x_pos_list = x_pos[0].tolist()
#x_pos_list=x_pos_list[0:200]

#Reading all the delimted negative reviews present in the file result_neg.txt as a dataframe

x_neg=pd.read_csv(r"C:\Users\uttej\OneDrive - University of Waterloo\Documents\657\Assignment 3\data\aclImdb\test\result_neg.txt",

#Converting the dataframe to a list
x_neg_list = x_neg[0].tolist()
#x_neg_list=x_neg_list[0:200]

#Validating the number of positive and negative reviews

len(x_pos_list),len(x_neg_list)
```

Out[11]: (12500, 12500)

Data Preprocessing on Test Data

In [12]:
```
x_pos_list1=[]
for i in range(len(x_pos_list)):
    x_pos_list[i] = x_pos_list[i].lower()
    text = re.sub('\n', ' ', x_pos_list[i])
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text=re.sub('[^abcdefghijklmnopqrstuvwxyz\s]', '',text)
    text = re.sub(r'\s+',' ',text)
    x_pos_list1.append(text)

lemmatizer = WordNetLemmatizer()
for i in range(len(x_pos_list1)):
    words = nltk.word_tokenize(x_pos_list1[i])
    words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
    x_pos_list1[i] = ' '.join(words)

x_neg_list1=[]
for i in range(len(x_neg_list)):
    x_neg_list[i] = x_neg_list[i].lower()
    text = re.sub('\n', ' ', x_neg_list[i])
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text=re.sub('[^abcdefghijklmnopqrstuvwxyz\s]', '',text)
    text = re.sub(r'\s+',' ',text)
    x_neg_list1.append(text)

lemmatizer = WordNetLemmatizer()
for i in range(len(x_neg_list1)):
    words = nltk.word_tokenize(x_neg_list1[i])
    words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
    x_neg_list1[i] = ' '.join(words)
```

In [13]:
```
x_test_list = x_pos_list1+x_neg_list1
len(x_test_list)
```

Out[13]:
```
25000
```

Model Implementation

Using CounterVectorizer and Logistic Regression

In order for the data to be understandable by our algorithm, we will need to convert each review to a numeric representation called vectorization. We are using CountVectorizer to convert the collection of review to a numeric representation.

CountVectorizer is a tool provided by the scikit-learn library in Python. It is used to alter a given text into a vector based on the count of each word that occurs in the complete text.

In [14]:
```
#Using CountVectorizer to convert the words to numeric representation

vectorizer = CountVectorizer(binary=True) #binary=True means all non zero counts are set to 1.
vectorizer.fit(x_train_list)
x = vectorizer.transform(x_train_list)

#Creating the target coulmn i.e 1 for all the positive reviews and 0 for negative reviews

target = [0 if j > 12500 else 1 for j in range(25000)]
```

We are creating a **Logistic Regression** model to classify the positive and negative reviews. Logistic regression is easy to interpret, it performs well on sparse datasets and also the model learns very fast compared to other algorithms.

In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success) or 0 (no, failure). It has hyperparameter C, which adjusts the regularization.

In [15]:
```
x_train, x_val, y_train, y_val = train_test_split(x, target, test_size = 0.2)
c_values = [0.01, 0.05, 0.25, 0.5, 1]
accuracy=[]
for i in c_values:
    model1 = LogisticRegression(C=i)
    model1.fit(x_train, y_train)
    accuracy.append(accuracy_score(y_val, model1.predict(x_val)))

d = {'Hyperparameter C':c_values,'Accuracy':accuracy}
df_acc = pd.DataFrame(d)
df_acc
```

Out[15]:

	Hyperparameter C	Accuracy
0	0.01	0.8762
1	0.05	0.8850
2	0.25	0.8794

	Hyperparameter C	Accuracy
3	0.50	0.8770
4	1.00	0.8740

Using CountVectorizer, the value of C that gives us the highest accuracy of 88.50% is 0.05

In [16]:
```
x_test = vectorizer.transform(x_test_list)
final_model1 = LogisticRegression(C=0.05)
final_model1.fit(x, target)
a = round(accuracy_score(target, final_model1.predict(x_test)),4)
print("Accuracy on Test Data: {}".format(a))
```

```
Accuracy on Test Data: 0.8762
```

From the above, we see that on the test data, the Logistic Regression model using CountVectorizer gave an accuracy of 87.62%

Using TfidfVectorizer and Logistic Regression

Another usual way to represent each document in a corpus is by using the **term frequency-inverse document frequency** for each word.The tf-idf aims to represent the number of times a given word appears in a document relative to the number of documents in the corpus that the word appears in. The words that appear in many documents have a less value and words that appear in less documents have high values.

In [17]:
```
#Using TfidfVectorizer to convert the words to numeric representation

vectorizer = TfidfVectorizer()
vectorizer.fit(x_train_list)
x = vectorizer.transform(x_train_list)

#Creating the target coulmn i.e 1 for all the positive reviews and 0 for negative reviews

target = [0 if j > 12500 else 1 for j in range(25000)]
```

In [18]:
```
x_train, x_val, y_train, y_val = train_test_split(x, target, test_size = 0.2)
c_values = [0.01, 0.05, 0.25, 0.5, 1]
accuracy=[]
for i in c_values:
    model1 = LogisticRegression(C=i)
    model1.fit(x_train, y_train)
    accuracy.append(accuracy_score(y_val, model1.predict(x_val)))

d = {'Hyperparameter C':c_values,'Accuracy':accuracy}
df_acc = pd.DataFrame(d)
df_acc
```

Out[18]:

	Hyperparameter C	Accuracy
0	0.01	0.8120
1	0.05	0.8440
2	0.25	0.8714
3	0.50	0.8794
4	1.00	0.8872

Using TfidfVectorizer, the value of C that gives us the highest accuracy of 88.72% is 1

In [19]:
```
x_test = vectorizer.transform(x_test_list)
final_model2 = LogisticRegression(C=1)
final_model2.fit(x, target)
a = round(accuracy_score(target, final_model2.predict(x_test)),4)
print("Accuracy on Test Data: {}".format(a))
```

```
Accuracy on Test Data: 0.8782
```

From the above, we see that on the test data, the Logistic Regression model using TfidfVectorizer gave an accuracy of 87.82%

Using TfidfVectorizer and Linear Support Vector Machine

The linear classifiers tend to work well on very sparse datasets like the one we have. Support Vector Machine is another algorithm that can produce great results with a less training time.

Support Vector Machine(SVM) - SVMs can solve non-linear or linear problems. They are used for both regression and classification. Different classes in the data are divided by a line or a hyerplane. The Linear SVM also, has hyperparameter C, which adjusts the regularization.

In [20]:
```
from sklearn.svm import LinearSVC

x_train, x_val, y_train, y_val = train_test_split(x, target, test_size = 0.2)
c_values = [0.01, 0.05, 0.25, 0.5, 1]
accuracy=[]
for i in c_values:
    model1 = LinearSVC(C=i)
    model1.fit(x_train, y_train)
    accuracy.append(accuracy_score(y_val, model1.predict(x_val)))

d = {'Hyperparameter C':c_values,'Accuracy':accuracy}
df_acc = pd.DataFrame(d)
df_acc
```

Out[20]:

	Hyperparameter C	Accuracy
0	0.01	0.8458
1	0.05	0.8730
2	0.25	0.8872
3	0.50	0.8846
4	1.00	0.8824

Using SVM, the value of C that gives us the highest accuracy of 88.72% is 0.25

In [21]:

```
x_test = vectorizer.transform(x_test_list)
final_model = LinearSVC(C=0.25)
final_model.fit(x, target)
a = round(accuracy_score(target, final_model.predict(x_test)),4)
print("Accuracy on Test Data: {}".format(a))
```

```
Accuracy on Test Data: 0.8788
```

From the above, we see that on the test data, the Linear SVM model using TfidfVectorizer gave an accuracy of 87.88%

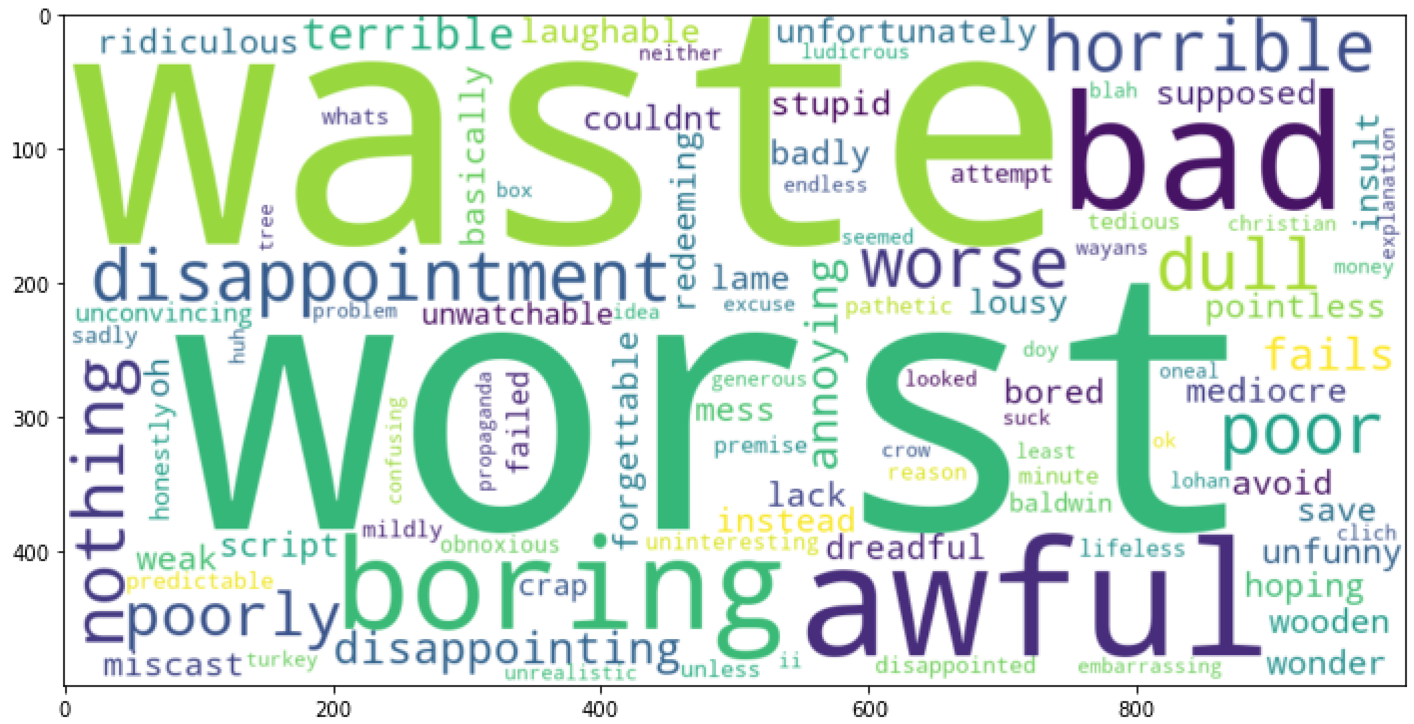
Creating a word cloud of top most discriminating words for positive and negative reviews.

In [22]:

```
word_coef = {review: coefficient for review, coefficient in zip(vectorizer.get_feature_names(), model1.coef_[0])}
# Positive reviews
top_pos_list=[]
for top_positive in sorted(word_coef.items(), key=lambda x: x[1], reverse=True)[:150]:
    top_pos_list.append(top_positive[0])

top_pos_list1=' '.join(top_pos_list[:100])

plt.figure(figsize=(12,10))
positive_text=top_pos_list1
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5,background_color='white')
positive_words=WC.generate(positive_text)
plt.imshow(positive_words,interpolation='bilinear')
plt.show;
```

In [23]:

```
# Negative reviews
top_neg_list=[]
for top_negative in sorted(word_coef.items(), key=lambda x: x[1])[:150]:
    top_neg_list.append(top_negative[0])

top_neg_list1=' '.join(top_neg_list[:100])

plt.figure(figsize=(12,10))
positive_text=top_neg_list1
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5,background_color='white')
positive_words=WC.generate(positive_text)
plt.imshow(positive_words,interpolation='bilinear')
plt.show;
```



LSTM Implementation

We have used tokenizer to vectorize the text and convert it into sequence of integers. It uses top most common 2000 words

pad_sequences is used to convert the sequences into 2-D numpy array.

In [24]:

```
data = pd.DataFrame(x_train_list, columns=['review'])
tokenizer_obj = Tokenizer(num_words=2000, split=' ')
tokenizer_obj.fit_on_texts(data['review'].values)
X = tokenizer_obj.texts_to_sequences(data['review'].values)
X = pad_sequences(X, maxlen=250)

#Preparing the test dataset for the model.
data_test = pd.DataFrame(x_test_list, columns=['review'])
X_test = tokenizer_obj.texts_to_sequences(data_test['review'].values)
X_test = pad_sequences(X_test,maxlen=250)

Y_train = [1 if j < 12500 else 0 for j in range(25000)]
Y_train = np.array(Y_train)

Y_test = [1 if j < 12500 else 0 for j in range(25000)]
Y_test = np.array(Y_test)
```

Model 1 - LSTM

In [25]:

```
model_lstm_1 = Sequential()
model_lstm_1.add(Embedding(input_dim=2000, output_dim=32, input_length=X.shape[1]))
model_lstm_1.add(Dropout(0.2))
model_lstm_1.add(LSTM(100))
model_lstm_1.add(Dense(units=256, activation='relu'))
model_lstm_1.add(Dropout(0.2))
model_lstm_1.add(Dense(units=1, activation='sigmoid'))
model_lstm_1.summary()
model_lstm_1.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 250, 32)           64000
_________________________________________________________________
dropout (Dropout)            (None, 250, 32)           0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dense (Dense)                (None, 256)               25856
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 143,313
Trainable params: 143,313
Non-trainable params: 0
_________________________________________________________________
```

a) batch_size = 64, epochs = 3

In [26]:

```
train_history = model_lstm_1.fit(X, Y_train, batch_size=64,epochs=3, verbose=True,validation_split=0.2)
```

```
Epoch 1/3
313/313 [==============================] - 50s 154ms/step - loss: 0.5417 - accuracy: 0.7200 - val_loss: 0.4896 - val_accuracy: 0.7
682
Epoch 2/3
313/313 [==============================] - 49s 155ms/step - loss: 0.2903 - accuracy: 0.8833 - val_loss: 0.4126 - val_accuracy: 0.8
248
Epoch 3/3
313/313 [==============================] - 52s 165ms/step - loss: 0.2647 - accuracy: 0.8970 - val_loss: 0.4454 - val_accuracy: 0.7
982
```

In [27]:
```
#Accuracy on the Test dataset

acc_lstm_1 = model_lstm_1.evaluate(X_test, Y_test, verbose = True, batch_size = 64)
print("Accuracy on Test Data: {}".format(round(acc_lstm_1[1],4)))
```

```
391/391 [==============================] - 15s 38ms/step - loss: 0.3436 - accuracy: 0.8492
Accuracy on Test Data: 0.8492
```

b) batch_size = 256, epochs = 10

In [37]:
```
model_lstm_3 = Sequential()
model_lstm_3.add(Embedding(input_dim=2000, output_dim=32, input_length=X.shape[1]))
model_lstm_3.add(Dropout(0.2))
model_lstm_3.add(LSTM(100))
model_lstm_3.add(Dense(units=256, activation='relu'))
model_lstm_3.add(Dropout(0.2))
model_lstm_3.add(Dense(units=1, activation='sigmoid'))
#model_lstm_2.summary()
model_lstm_3.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

In [38]:
```
train_history = model_lstm_3.fit(X, Y_train, batch_size=256,epochs=10, verbose=True,validation_split=0.2)
```

```
Epoch 1/10
79/79 [==============================] - 50s 614ms/step - loss: 0.6244 - accuracy: 0.6518 - val_loss: 0.4865 - val_accuracy: 0.772
6
Epoch 2/10
79/79 [==============================] - 51s 644ms/step - loss: 0.3017 - accuracy: 0.8744 - val_loss: 0.4508 - val_accuracy: 0.803
4
Epoch 3/10
79/79 [==============================] - 53s 671ms/step - loss: 0.2695 - accuracy: 0.8916 - val_loss: 0.4568 - val_accuracy: 0.815
2
Epoch 4/10
79/79 [==============================] - 54s 679ms/step - loss: 0.2607 - accuracy: 0.8973 - val_loss: 0.5098 - val_accuracy: 0.808
8
Epoch 5/10
79/79 [==============================] - 52s 663ms/step - loss: 0.2534 - accuracy: 0.9008 - val_loss: 0.2594 - val_accuracy: 0.886
0
Epoch 6/10
79/79 [==============================] - 53s 670ms/step - loss: 0.2645 - accuracy: 0.8913 - val_loss: 0.6360 - val_accuracy: 0.775
6
Epoch 7/10
79/79 [==============================] - 55s 690ms/step - loss: 0.2301 - accuracy: 0.9108 - val_loss: 0.3812 - val_accuracy: 0.835
4
Epoch 8/10
79/79 [==============================] - 54s 683ms/step - loss: 0.2188 - accuracy: 0.9165 - val_loss: 0.4728 - val_accuracy: 0.810
0
Epoch 9/10
79/79 [==============================] - 53s 677ms/step - loss: 0.1946 - accuracy: 0.9253 - val_loss: 0.5651 - val_accuracy: 0.790
0
Epoch 10/10
79/79 [==============================] - 52s 653ms/step - loss: 0.1926 - accuracy: 0.9270 - val_loss: 0.5457 - val_accuracy: 0.794
6
```

In [39]:
```
#Accuracy on the Test dataset

acc_lstm_3 = model_lstm_3.evaluate(X_test, Y_test, verbose = True, batch_size = 256)
print("Accuracy on Test Data: {}".format(round(acc_lstm_3[1],4)))
```

```
98/98 [==============================] - 14s 140ms/step - loss: 0.3906 - accuracy: 0.8464
Accuracy on Test Data: 0.8464
```

We have varied the parameters "batch_size" and number of epochs for Model 1 and noticed that batch_size = 64, number of epochs =3 gave a better accuracy on the test data.

Model 2 - LSTM + CNN

In [31]:
```
model_lstmcnn_1 = Sequential()
model_lstmcnn_1.add(Embedding(2000, 32, input_length=X.shape[1]))
model_lstmcnn_1.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model_lstmcnn_1.add(MaxPooling1D(pool_size=2))
model_lstmcnn_1.add(LSTM(100))
model_lstmcnn_1.add(Dense(units=256, activation='relu'))
model_lstmcnn_1.add(Dropout(0.2))
model_lstmcnn_1.add(Dense(1, activation='sigmoid'))
model_lstmcnn_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_lstmcnn_1.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 250, 32)	64000
conv1d (Conv1D)	(None, 250, 32)	3104
max_pooling1d (MaxPooling1D)	(None, 125, 32)	0
lstm_2 (LSTM)	(None, 100)	53200
dense_4 (Dense)	(None, 256)	25856
dropout_4 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 1)	257

```
Total params: 146,417
Trainable params: 146,417
```

```
Non-trainable params: 0
_________________________________________________________________
```

a) batch_size = 64, epochs = 3

In [32]:
```
train_history = model_lstmcnn_1.fit(X, Y_train, batch_size=64,epochs=3, verbose=True,validation_split=0.2)
```

```
Epoch 1/3
313/313 [==============================] - 34s 100ms/step - loss: 0.5405 - accuracy: 0.7137 - val_loss: 0.5563 - val_accuracy: 0.7
642
Epoch 2/3
313/313 [==============================] - 34s 108ms/step - loss: 0.2777 - accuracy: 0.8903 - val_loss: 0.4785 - val_accuracy: 0.7
996
Epoch 3/3
313/313 [==============================] - 33s 106ms/step - loss: 0.2447 - accuracy: 0.9039 - val_loss: 0.4695 - val_accuracy: 0.8
206
```

In [33]:
```
acc_lstm_cnn_1 = model_lstmcnn_1.evaluate(X_test, Y_test, verbose = True, batch_size = 64)
print("Accuracy on Test Data: {}".format(round(acc_lstm_cnn_1[1],4)))
```

```
391/391 [==============================] - 11s 27ms/step - loss: 0.3387 - accuracy: 0.8618
Accuracy on Test Data: 0.8618
```

b) batch_size = 256, epochs = 6

In [34]:
```
model_lstmcnn_2 = Sequential()
model_lstmcnn_2.add(Embedding(2000, 32, input_length=X.shape[1]))
model_lstmcnn_2.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model_lstmcnn_2.add(MaxPooling1D(pool_size=2))
model_lstmcnn_2.add(LSTM(100))
model_lstmcnn_2.add(Dense(units=256, activation='relu'))
model_lstmcnn_2.add(Dropout(0.2))
model_lstmcnn_2.add(Dense(1, activation='sigmoid'))
model_lstmcnn_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#model_lstmcnn_2.summary()
```

In [35]:
```
train_history = model_lstmcnn_2.fit(X, Y_train, batch_size=256,epochs=6, verbose=True,validation_split=0.2)
```

```
Epoch 1/6
79/79 [==============================] - 23s 266ms/step - loss: 0.6489 - accuracy: 0.6266 - val_loss: 0.6735 - val_accuracy: 0.751
6
Epoch 2/6
79/79 [==============================] - 20s 258ms/step - loss: 0.3511 - accuracy: 0.8486 - val_loss: 0.6299 - val_accuracy: 0.698
0
Epoch 3/6
79/79 [==============================] - 21s 272ms/step - loss: 0.2775 - accuracy: 0.8896 - val_loss: 0.4064 - val_accuracy: 0.812
0
Epoch 4/6
79/79 [==============================] - 21s 268ms/step - loss: 0.2388 - accuracy: 0.9072 - val_loss: 0.3335 - val_accuracy: 0.857
8
Epoch 5/6
79/79 [==============================] - 21s 263ms/step - loss: 0.2250 - accuracy: 0.9154 - val_loss: 0.5166 - val_accuracy: 0.797
8
Epoch 6/6
79/79 [==============================] - 21s 264ms/step - loss: 0.1948 - accuracy: 0.9303 - val_loss: 0.5173 - val_accuracy: 0.794
4
```

In [36]:
```
acc_lstm_cnn2 = model_lstmcnn_2.evaluate(X_test, Y_test, verbose = True, batch_size = 256)
print("Accuracy on Test Data: {}".format(round(acc_lstm_cnn2[1],4)))
```

```
98/98 [==============================] - 7s 72ms/step - loss: 0.3598 - accuracy: 0.8520
Accuracy on Test Data: 0.852
```

We have varied the parameters "batch_size" and number of epochs for Model 2 and noticed that batch_size = 64, number of epochs =3 gave a better accuracy on the test data.

Summary

Data Read-in

The IMDB dataset contained 25k train and 25k test sets. Postive and negative reviews for both train and test datasets were present in separate folders.

Golb2 library was used for reading all the files present in train and test folders. This library has the ability to capture the text matched by glob patterns, and return those matches alongside the filenames.

Initially, data from different text files are read and then it is written into a text file with "\n" as delimiter for the reviews. Next, the delimited text file is read as a data frame, which is used for data preprocessing.

Data Preprocessing

Below are the preprocessing steps performed on both train and test datasets:

1. Converting the text to lower case, removing new lines within a sentence, alphanumeric words, text in <>, http links, characters that are not alphabets, extra spaces.
2. Normalized the words in the corpus by trying to convert all of the different forms of a given word into one. We have performed Lemmatization instead of stemming since stemming just removes the last few characters of a word, often leading to incorrect meanings and spelling. Lemmatization considers the context and converts the word to its meaningful base form.
3. Removed the stop words.

In order for the data to be understandable by our algorithms, we converted each review to a numeric representation called vectorization. We have used different method of vectorization like CounterVectorizer, TfidfVectorizer.

Network Design

In order to classify reviews from the IMDB movie review dataset as positive or negative, we first created a **Logistic Regression** model. As it is easy to interpret, performs well on sparse datasets and quick learning rate compared to other algorithms. Logistic regression model using CountVectorizer, gave an accuracy of 87.62% on the test data.

We also tried **Linear SVM** as a classifier and it gave a accuracy of 87.88% on the test data

We then used **Long Short-Term Memory (LSTM)** type of Recurrent Neural network. A Recurrent Neural Network can learn dependencies but, it can only learn about recent information. LSTM can help solve this problem as it can understand context along with recent dependency and would be a best fit for sentiment analysis.

In order to decide the best deign for the LSTM, we have varied the parameters "batch_size" and number of epochs. We have made implementations on various combinations batch size and number of epochs. However, we have shown only few models in the code.

As number of epochs and batch size increased, we noticed that the accuracy on the test data decreased. Hence, we are considering the batch_size = 64, number of epochs as 3 for our final model.

Both Model 1(LSTM) and Model 2(LSTM & CNN) gave almost equal accuracy of **~86%** for batch_size = 64 and Number of epochs =3.

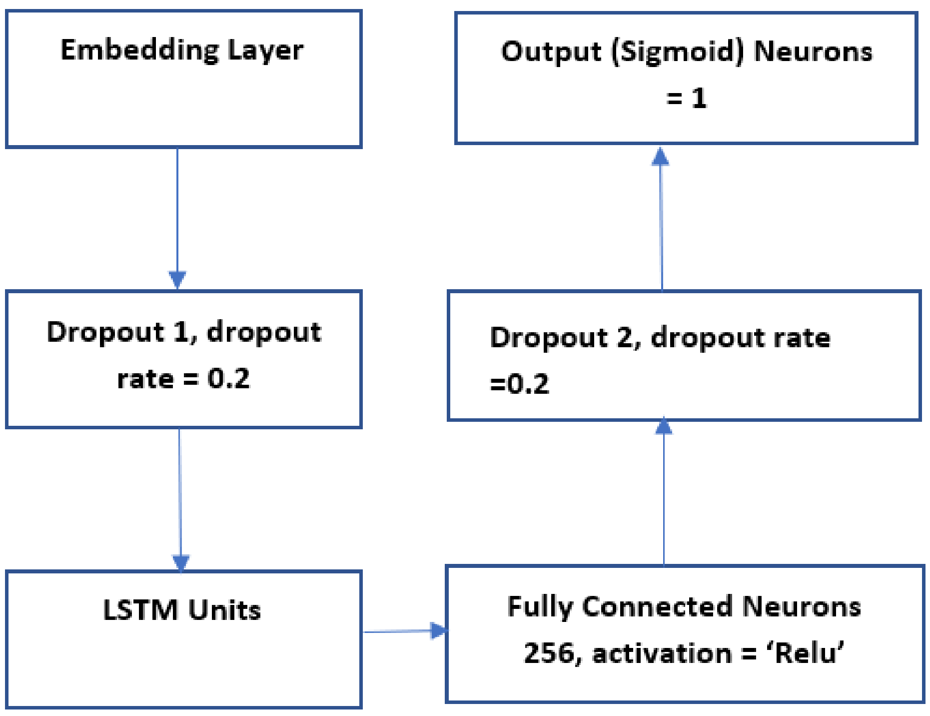
Adding a CNN layer in the Model 2 has given us same accuracy but computational time for this model has increased when compared to Model 1(LSTM). Hence we have finalized **Model 1 (LSTM)** for the Sentiment Analysis.

Final Model Architecture

In [117…

```
Image("lstm.PNG",width=500,height=400)
```

Out[117…



- The embedding layer encodes the input sequence into a sequence of dense vectors of dimension mentioned.
- Dropout - This is a regularization method where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while training a network. This has the effect of reducing overfitting and improving model performance. Dropout rate of 0.2 has been used.
- We considered 100 LSTM units for the model
- Relu - Rectified linear unit function will output the input if it is positive, otherwise it will output zero. This overcomes the vanishing gradient problem, allowing models to learn faster and perform better.
- Sigmoid - This function limits the output to a range between 0 and 1.
- We are using "Adam" optimizer as it handles sparse gradients and trains the nework efficiently.
- Loss as 'binary_crossentropy' is used as we have only two label classes.

The blog https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47 clearly explains the working of LSTM algorithm and this has been referred to better understand the concept and to implement the algorithm.