LP 5 - Practical 2

Classification using Deep neural network

Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset

Name: Onasvee Banarse

Roll No: 09

BE COMP 1

Import the required libraries

In [5]:

```
import pandas as pd
import numpy as np
import keras
from matplotlib import pyplot as plt
from keras.preprocessing.text import Tokenizer
from keras.utils.data_utils import pad_sequences
```

Read the Data

In [6]:

```
df_train=pd.read_csv('Train.csv')
df_val=pd.read_csv('Valid.csv')
df_train.head()
```

Out[6]:

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

In [7]:

```
df_val.head()
```

Out[7]:

	text	label
0	It's been about 14 years since Sharon Stone aw...	0
1	someone needed to make a car payment... this i...	0
2	The Guidelines state that a comment must conta...	0
3	This movie is a muddled mish-mash of clichés f...	0
4	Before Stan Laurel became the smaller half of ...	0

In [8]:

```
X_train=df_train['text'].values
Y_train=df_train['label'].values
```

In [9]:

```
X_val=df_val['text'].values
Y_val=df_val['label'].values
```

In [10]:

```
(X_train.shape,Y_train.shape),(X_val.shape,Y_val.shape)
```

Out[10]:

```
(((40000,), (40000,)), ((5000,), (5000,)))
```

Analyse the Data

In [11]:

```
df_train.iloc[:,1].describe()
```

Out[11]:

```
count    40000.000000
mean         0.499525
std          0.500006
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
Name: label, dtype: float64
```

In [12]:

```
df_val.iloc[:,1].describe()
```

Out[12]:

```
count    5000.000000
mean        0.502800
std         0.500042
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: label, dtype: float64
```

In [14]:

```
X_val_len=[len(str(i).split()) for i in X_val]
df1=pd.DataFrame(X_val_len,columns=['len'])
df1.describe()
```

Out[14]:

	len
count	5000.00000
mean	228.93260
std	169.33721
min	10.00000
25%	126.00000
50%	171.00000
75%	274.00000
max	1601.00000

In [16]:

```
X_train_len=[len(str(i).split()) for i in X_train]
df=pd.DataFrame(X_train_len,columns=['len'])
df.describe()
```

Out[16]:

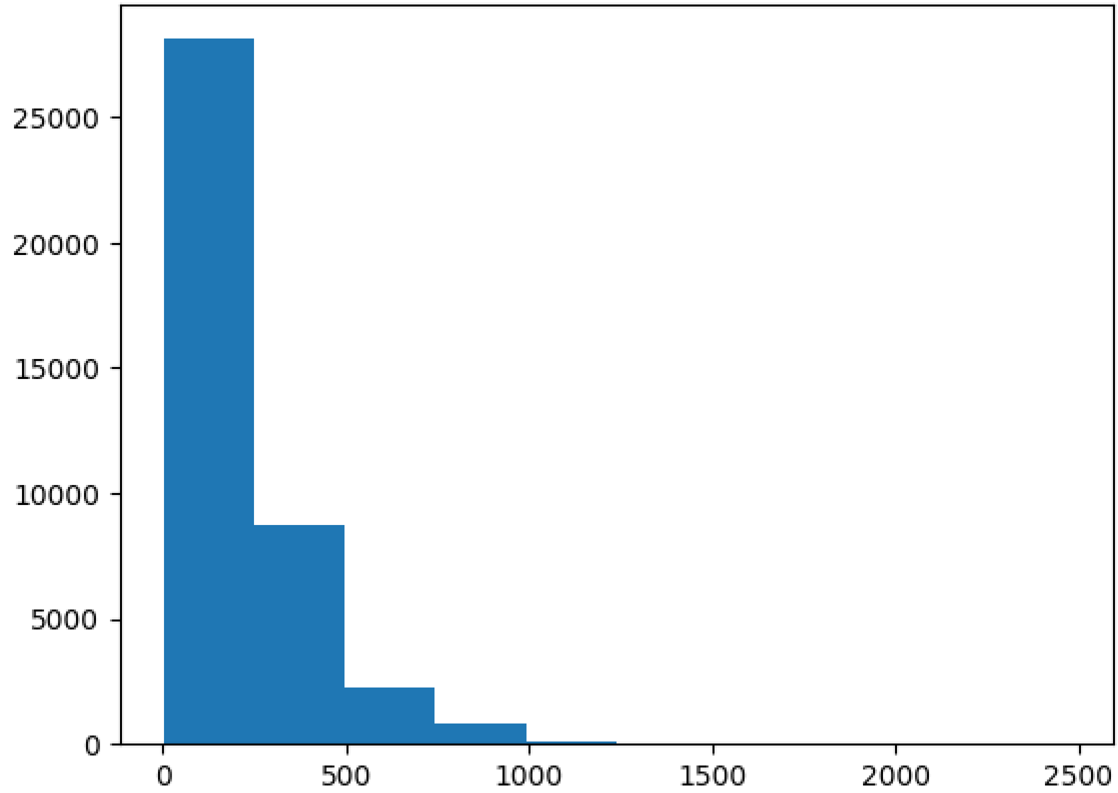
	len
count	40000.000000
mean	231.339250
std	171.194123
min	4.000000
25%	126.000000
50%	173.000000
75%	282.000000
max	2470.000000

In [17]:

```
X_train_len=[len(str(i).split()) for i in X_train]
plt.hist(X_train_len)
```

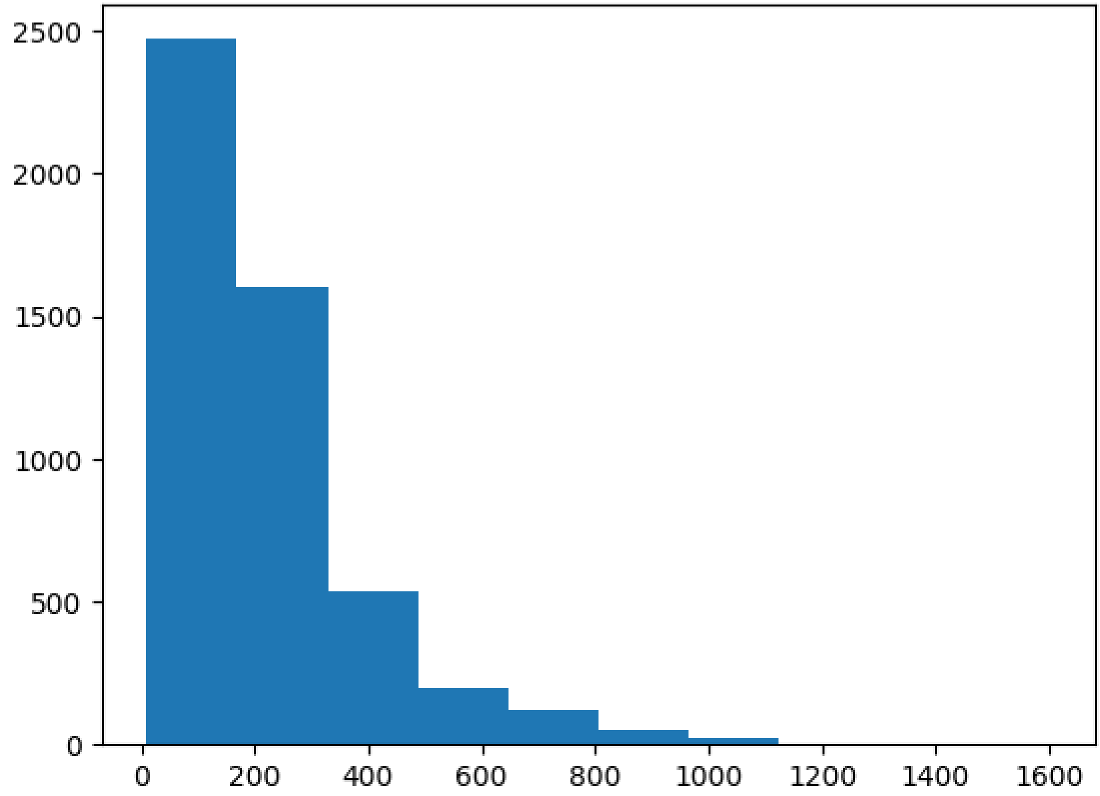
Out[17]:

```
(array([2.8097e+04, 8.6960e+03, 2.2520e+03, 8.0100e+02, 1.3800e+02,
        7.0000e+00, 3.0000e+00, 3.0000e+00, 1.0000e+00, 2.0000e+00]),
 array([   4. ,  250.6,  497.2,  743.8,  990.4, 1237. , 1483.6, 1730.2,
        1976.8, 2223.4, 2470. ]),
 <BarContainer object of 10 artists>)
```



```
In [18]: X_val_len=[len(str(i).split()) for i in X_val]
         plt.hist(X_val_len)
```

```
Out[18]: (array([2.470e+03, 1.602e+03, 5.350e+02, 1.970e+02, 1.180e+02, 5.100e+01,
                 2.300e+01, 1.000e+00, 2.000e+00, 1.000e+00]),
          array([  10. ,  169.1,  328.2,  487.3,  646.4,  805.5,  964.6, 1123.7,
                 1282.8, 1441.9, 1601. ]),
          <BarContainer object of 10 artists>)
```



Setting the parameters

```
In [19]: vocab_size=30000 #went for an average vocab size
         embedding_dimension=64 #high dimensions would result in finding better parameters
         max_length=120 #used a maximum length of 120 words
         turnc='post'#preprocessing step for pad_sequences
         oov_tok='<OOV>'#oov stands for out of vocabulary tokens
```

Tokenizing and converting the data into Sequences

```
In [20]: tokenizer=Tokenizer(num_words=vocab_size,filters='''!"#$%&'()*+,-./:;<=>?@[\]^_`{|
         tokenizer.fit_on_texts(X_train)
         X=tokenizer.texts_to_sequences(X_train)
         X_padded=pad_sequences(X,maxlen=max_length,padding='post',truncating=turnc)
         X_val_seq=tokenizer.texts_to_sequences(X_val)
         X_val_padded=pad_sequences(X_val_seq,maxlen=max_length,padding='post',truncating=tu
```

```
In [21]: X_padded.shape,X_val_padded.shape
```

```
Out[21]: ((40000, 120), (5000, 120))
```

The Model

```
In [22]: from keras.layers import LSTM,Bidirectional,Embedding,Dense,SpatialDropout1D,Flatte
         from keras.models import Sequential
```

```
In [23]: model=Sequential()
         model.add(Embedding(vocab_size,embedding_dimension,input_length=max_length))
         model.add(SpatialDropout1D(0.4))
         model.add(Bidirectional(LSTM(120,activation='tanh',return_sequences=True)))
         model.add(Dropout(0.3))
         model.add(Bidirectional(LSTM(120,activation='tanh',return_sequences=False)))
         model.add(Dropout(0.2))
         model.add(Dense(300,activation='relu'))
         model.add(Dropout(0.3))
         model.add(Dense(1,activation='sigmoid'))
         print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 120, 64)           1920000   
                                                                 
 spatial_dropout1d (SpatialD  (None, 120, 64)          0         
 ropout1D)                                                       
                                                                 
 bidirectional (Bidirectiona  (None, 120, 240)         177600    
 l)                                                              
                                                                 
 dropout (Dropout)           (None, 120, 240)          0         
                                                                 
 bidirectional_1 (Bidirectio  (None, 240)              346560    
 nal)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 240)               0         
                                                                 
 dense (Dense)               (None, 300)               72300     
                                                                 
 dropout_2 (Dropout)         (None, 300)               0         
                                                                 
 dense_1 (Dense)             (None, 1)                 301       
                                                                 
=================================================================
Total params: 2,516,761
Trainable params: 2,516,761
Non-trainable params: 0
_________________________________________________________________
None
```

In [24]:
```
model.compile(optimizer="rmsprop",loss='binary_crossentropy',metrics=['accuracy'])
```

In [25]:
```
hist=model.fit(X_padded,Y_train,epochs=7,batch_size=16,validation_data=(X_val_padde
```

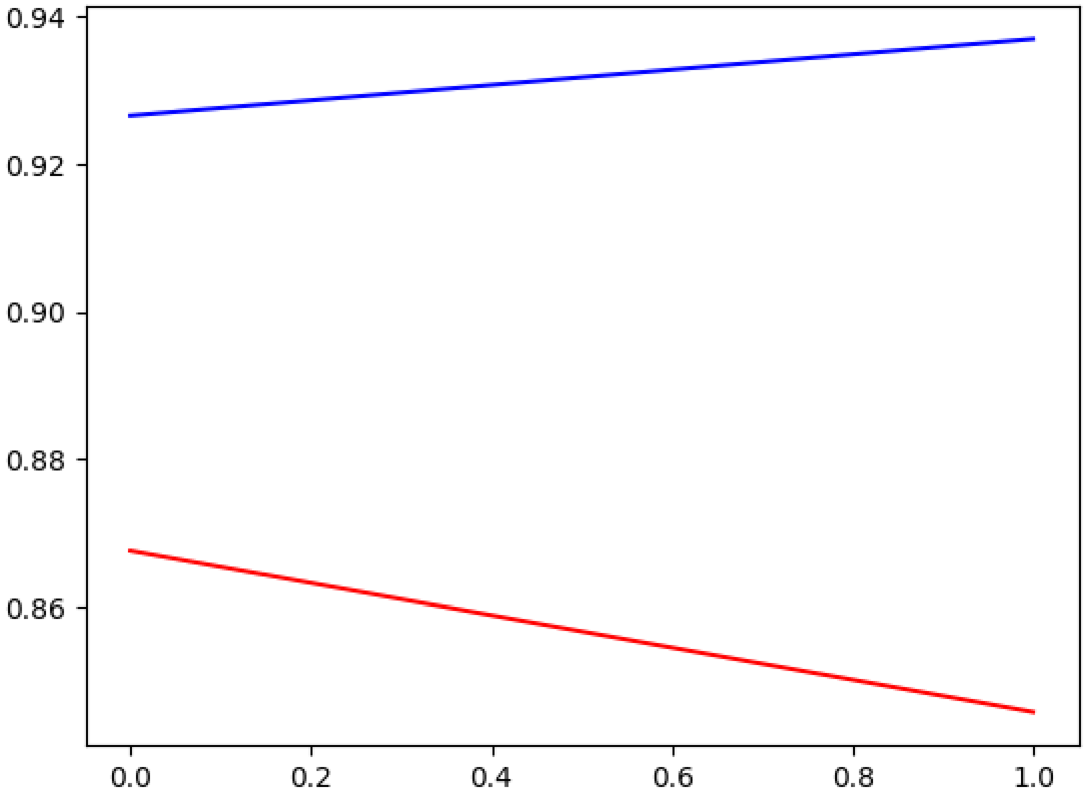
```
Epoch 1/7
2500/2500 [==============================] - 565s 224ms/step - loss: 0.4687 - accu
racy: 0.7755 - val_loss: 0.3797 - val_accuracy: 0.8342
Epoch 2/7
2500/2500 [==============================] - 581s 232ms/step - loss: 0.3510 - accu
racy: 0.8540 - val_loss: 0.4503 - val_accuracy: 0.8194
Epoch 3/7
2500/2500 [==============================] - 455s 182ms/step - loss: 0.3212 - accu
racy: 0.8702 - val_loss: 0.3702 - val_accuracy: 0.8534
Epoch 4/7
2500/2500 [==============================] - 417s 167ms/step - loss: 0.2935 - accu
racy: 0.8848 - val_loss: 0.3486 - val_accuracy: 0.8514
Epoch 5/7
2500/2500 [==============================] - 423s 169ms/step - loss: 0.2735 - accu
racy: 0.8938 - val_loss: 0.3364 - val_accuracy: 0.8584
Epoch 6/7
2500/2500 [==============================] - 430s 172ms/step - loss: 0.2508 - accu
racy: 0.9048 - val_loss: 0.3705 - val_accuracy: 0.8558
Epoch 7/7
2500/2500 [==============================] - 442s 177ms/step - loss: 0.2262 - accu
racy: 0.9149 - val_loss: 0.3905 - val_accuracy: 0.8620
```

In [26]:
```
hist=model.fit(X_padded,Y_train,epochs=2,batch_size=16,validation_data=(X_val_padde
Epoch 1/2
2500/2500 [==============================] - 477s 191ms/step - loss: 0.2029 - accu
racy: 0.9265 - val_loss: 0.3551 - val_accuracy: 0.8676
Epoch 2/2
2500/2500 [==============================] - 447s 179ms/step - loss: 0.1785 - accu
racy: 0.9369 - val_loss: 0.4937 - val_accuracy: 0.8458
```

This Plot is for the last two Epochs

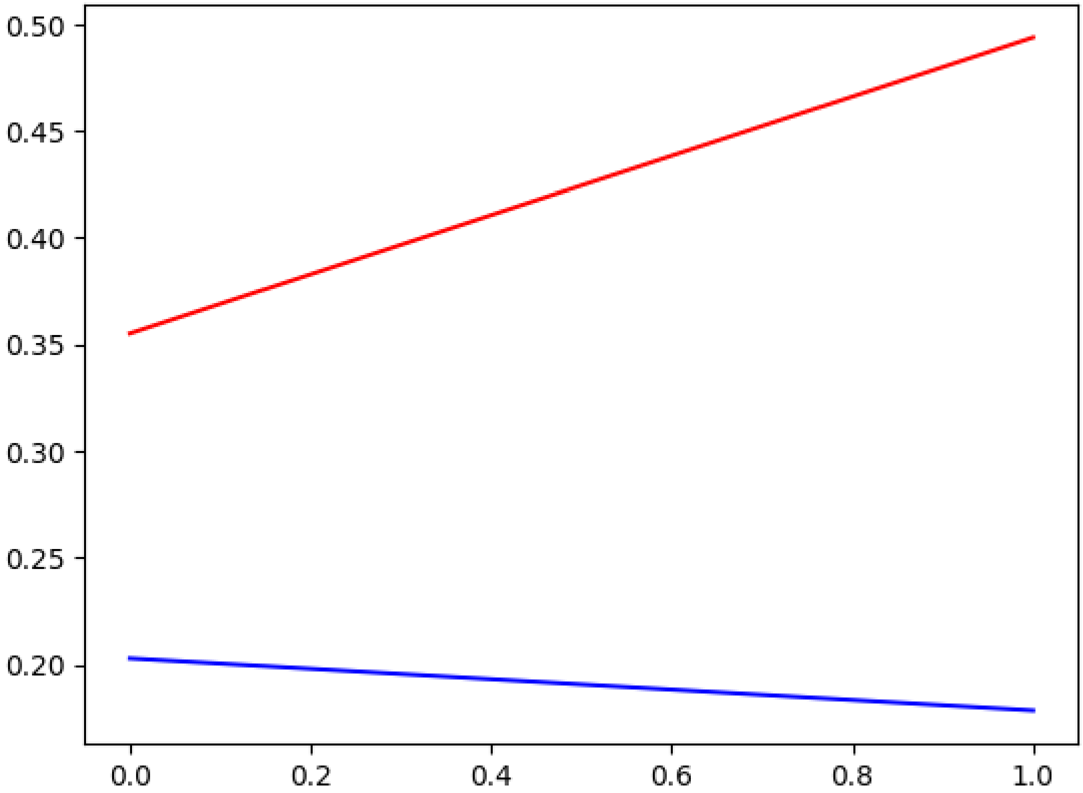
In [27]:

```
plt.plot(hist.history['accuracy'],c='b')
plt.plot(hist.history['val_accuracy'],c='r')
plt.show()
```



In [28]:

```
plt.plot(hist.history['loss'],c='b')
plt.plot(hist.history['val_loss'],c='r')
plt.show()
```



Reading the Test Data

In [123…

```
df_test=pd.read_csv('Test.csv')
df_test.head()
```

Out[123]:

	text	label
0	I always wrote this series off as being a comp...	0
1	1st watched 12/7/2002 - 3 out of 10(Dir-Steve ...	0
2	This movie was so poorly written and directed ...	0
3	The most interesting thing about Miryang (Secr...	1
4	when i first read about "berlin am meer" i did...	0

In [31]:

```
X_test=df_test['text'].values
Y_test=df_test['label'].values
```

Converting into Sequential Data

In [32]:

```
X_test_seq=tokenizer.texts_to_sequences(X_test)
```

In [33]:

```
X_test_padded=pad_sequences(X_test_seq,maxlen=max_length,padding='post',truncating
X_test_padded[0]
```

Out[33]:

```
array([   11,   212,  1082,    12,   200,   125,    16,   112,     4,
         570,  7688,  3306,    86,  1242,  5463,    15,   565,    10,
           9,     3,  2713,    20,    93,    30,   255,     4,  1650,
       16887,  4040,   102,     4,   450,  8685,   459,  1004,    11,
        1679,    13,     2,  2894,    15,    32,     2,    97,    24,
           2,    79,   490,     5,     2,   677,   831,   149,    11,
          98,    29,    43,  1916,    57,     3,  2247,   621,     2,
         677,     6,    76,     2,  2894,    42,    60,     6,     2,
         240,     6,   480,     2,  1190,    20,    93,   136,    25,
          43,    76,    57,     3,  1194,   621,     2,   649,     6,
         105,   240,    10,   165,  1051,  4891,     6,    13,    11,
         307,    38,    11,   861,     6,    43,  2988,  2503,    24,
           2,  5596,     3,   191,   847,  2008,    69,    10,  1074,
          18,    70,    49])
```

In [34]:

```
X_test_padded.shape
```

Out[34]:

```
(5000, 120)
```

In [35]:

```
model.evaluate(X_test_padded,Y_test)
```

```
157/157 [==============================] - 11s 70ms/step - loss: 0.4858 - accurac
y: 0.8502
```

Out[35]:

```
[0.485765665769577, 0.8501999974250793]
```

Check for your own Reviews

In [124…

```
def Check(x):
    test_case1=[x]
    test_case=tokenizer.texts_to_sequences(test_case1)
    test_case_padded=pad_sequences(test_case,padding='post',truncating=turnc)
    predict_x=model.predict(test_case_padded)
    print(predict_x)
    if predict_x>=0.5:
        print("Positive")
    else:
        print("Negative")
```

In [128…

```
test_review=str(input("Enter the review :  "))
Check(test_review)
```

```
Enter the review :  This is an epic film about the unification of the ancient king
doms of China in the third century BC. What makes it interesting is the tragic dow
nfall of the king and all the palace intrigue going on around him. It reminded me
a bit of "King Lear" and some of the other Shakespeare plays.<br /><br />The king
starts out with noble ambitions, to unify the kingdoms under one ruler and to stop
all the quarrelling so that the people can prosper and lead better lives. He and h
is childhood sweetheart, played beautifully by Li Gong, concoct a scheme whereby s
he pretends to go into exile in a rival kingdom in order to recruit an assassin to
kill the king, thus giving him a pretext to go to war. But while she's away, the k
ing becomes sadistic in his lust for power and goes on a killing spree.<br /><br /
>There are numerous side plots that keep the action going. There is the Marquis, w
ho pretends to be stupid and foppish but who's really very clever and wants to bec
ome king himself. He fathers two children with the king's mother and manages to ke
ep it secret for years. Then there is the Prime Minister, a political rival to the
king, who turns out to really be his father. <br /><br />The assassin is a complex
character himself. An adept swordsman and killer, he is undergoing a reformation w
hen the king's lover comes to recruit him. He wants nothing more with killing, but
is eventually won over by Li Gong (who wouldn't be?) when he sees how cruel and vi
cious the king has become.<br /><br />Some spectacular cinematography, especially
the battle scenes that are carried out on a grand scale - like they used to say, a
cast of thousands, literally. The acting is OK, nothing special. It's the story th
at's interesting, though at over two and a half hours, it pushes the limit.<br /><
br />Definitely worth viewing.
1/1 [==============================] - 0s 47ms/step
[[0.99055874]]
Positive
```

I just checked for one random imdb review

In [129…

```
test_review="You will get A to Z all details of this scam, i may be wrong but due t
Check(test_review)
```

```
1/1 [==============================] - 0s 23ms/step
[[0.02230704]]
Negative
```

In [ ]: