IMDB SIMPLE RNN

In [ ]:

```
import numpy as np
import pandas as pd
```

In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:

```
from scipy import stats
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.embeddings import Embedding
from keras.layers import SimpleRNN,Dense,Activation
```

Loading the Dataset

In [ ]:

```
(X_train,Y_train),(X_test,Y_test) = imdb.load_data(path="imdb.npz",num_words=None,skip_
top=0,maxlen=None,start_char=1,seed=13,oov_char=2,index_from=3)
```

In [ ]:

```
print("Type: ", type(X_train))
print("Type: ", type(Y_train))
```

```
Type:  <class 'numpy.ndarray'>
Type:  <class 'numpy.ndarray'>
```

In [ ]:

```
print("X train shape: ",X_train.shape)
print("Y train shape: ",Y_train.shape)
```

```
X train shape:  (25000,)
Y train shape:  (25000,)
```

Exploratory data Analysis

In [ ]:

```
print("Y train values: ",np.unique(Y_train))
print("Y test values: ",np.unique(Y_test))
```

```
Y train values:  [0 1]
Y test values:  [0 1]
```

In [ ]:

```
unique,counts = np.unique(Y_train,return_counts=True)
print("Y train distribution: ", dict(zip(unique,counts)))
```

```
Y train distribution:  {0: 12500, 1: 12500}
```

In [ ]:

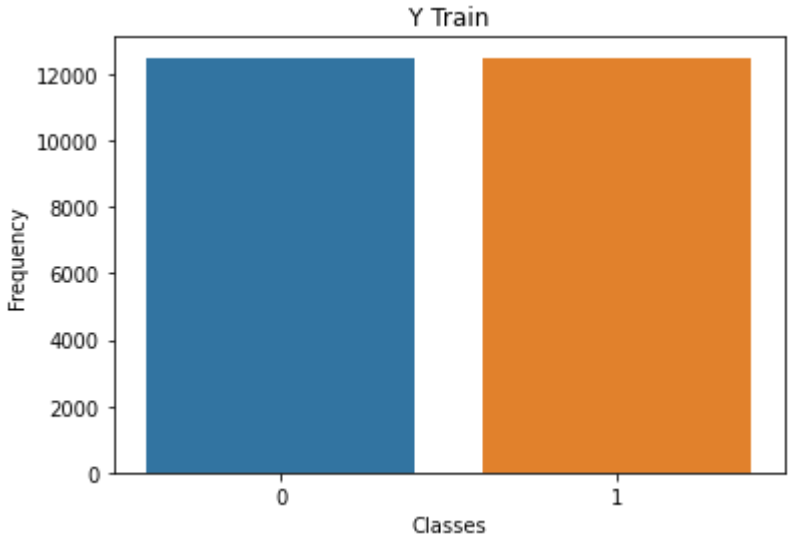
```
unique,counts = np.unique(Y_test,return_counts=True)
print("Y test distribution: ", dict(zip(unique,counts)))
```

```
Y test distribution:  {0: 12500, 1: 12500}
```

In [ ]:

```
plt.figure();
sns.countplot(Y_train);
plt.xlabel("Classes");
plt.ylabel("Frequency");
plt.title("Y Train");
```

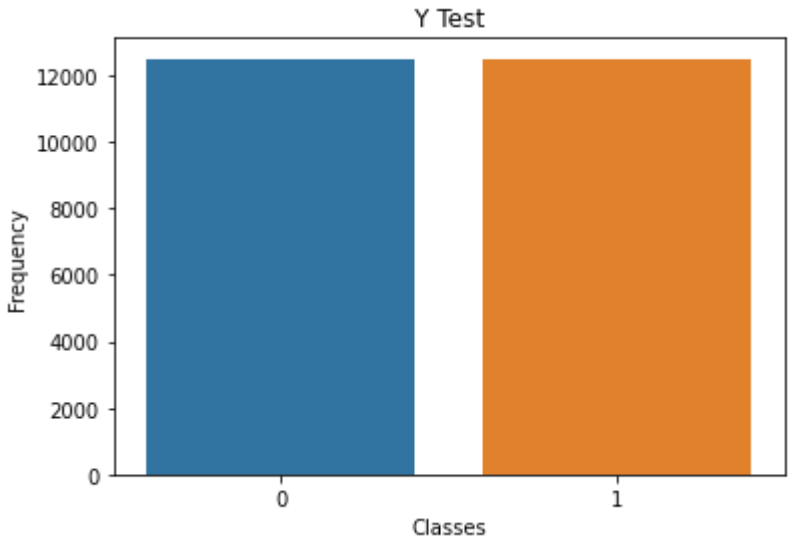
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```



In [ ]:

```
plt.figure();
sns.countplot(Y_test);
plt.xlabel("Classes");
plt.ylabel("Frequency");
plt.title("Y Test");
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretat
ion.
  FutureWarning
```



In [ ]:

```
print(X_train[0])
```

```
[1, 608, 13, 6467, 14, 22, 13, 80, 1109, 14, 20, 584, 18, 231, 72, 141, 6,
783, 254, 189, 7060, 13, 100, 115, 106, 14, 20, 584, 207, 82, 557, 111, 11
1, 537, 7, 4, 962, 12, 69, 11, 45, 204, 766, 33, 4, 8334, 23, 94, 797, 104
8, 991, 527, 1987, 538, 2629, 4, 4518, 125, 72, 449, 8295, 68, 3385, 2500,
93, 14, 1190, 22, 13, 119, 12, 13, 197, 4, 226, 22, 16, 542, 5, 221, 14, 2
0, 9, 38, 629, 14, 9, 4, 6128, 20, 13, 28, 126, 110, 11, 61, 113, 24, 15,
51, 571, 11, 4, 22, 5, 4, 326, 7, 4, 22, 26, 24, 629, 195, 21, 51, 210, 18
8, 72, 16, 21849, 2726, 116, 118, 189, 22, 126, 164, 70, 126, 30, 14, 629,
174, 2195, 829, 33, 94, 61124]
```

In [ ]:

```
review_len_train = []
review_len_test = []
for i,j in zip(X_train,X_test):
    review_len_train.append(len(i))
    review_len_test.append(len(j))
```

In [ ]:

```
print("min: ", min(review_len_train), "max: ", max(review_len_train))
```

```
min:  11 max:  2494
```

In [ ]:

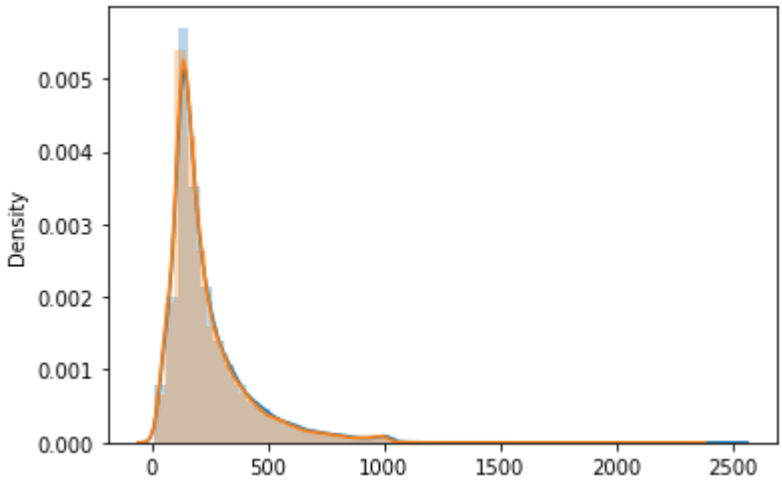
```
print("min: ", min(review_len_test), "max: ", max(review_len_test))
```

```
min:  7 max:  2315
```

In [ ]:

```
sns.distplot(review_len_train,hist_kws={"alpha":0.3});
sns.distplot(review_len_test,hist_kws={"alpha":0.3});
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```



In [ ]:

```
print("Train mean: ",np.mean(review_len_train))
print("Train median: ",np.median(review_len_train))
print("Train mode: ",stats.mode(review_len_train))
```

```
Train mean:  238.71364
Train median:  178.0
Train mode:  ModeResult(mode=array([132]), count=array([196]))
```

In [ ]:

```
word_index = imdb.get_word_index()
print(type(word_index))
```

```
<class 'dict'>
```

In [ ]:

```
print("length of word_index: ",len(word_index))
```

```
length of word_index:  88584
```

In [ ]:

```
for keys,values in word_index.items():
    if values == 1:
        print(keys)
```

```
the
```

In [ ]:

```
def whatItSay(index=24):
    reverse_index = dict([(value,key) for (key,value) in word_index.items()])
    decode_review = " ".join([reverse_index.get(i-3, "!") for i in X_train[index]])
    print(decode_review)
    print(Y_train[index])
    return decode_review

decoded_review = whatItSay()
```

```
! this movie was extremely funny i would like to own this for my vintage c
ollection of 1970s movie must see again list i know this cast of character
s they are people that i have met over the years and that prompt me to sea
rch out this comedy unfortunately this was never put to dvd or vhs redd fo
xx always a clown of comedy pearl baily a great match as his wife witty an
d sassy norman a son with a secret not sure if he will have a future if it
is out dennis dugan crazy funny man miss dobson hooker with a heart and li
ttle conscience love lust strange family ties this movie qualifies for a c
ome back encore performance situation comedy with a mix of events as this
could and should find its way as a remake i do think finding cast would be
extremely difficult maybe impossible except jerry seinfeld playing dennis
dugan role this earmarks a couple of seinfeld episodes that also brought m
e back to norman is that you keeping them in the closest was surely imposs
ible as impossible to reform pretend hooker girl friend and infidelity of
a parent this movie was a wild ride advise of a cabbie remind me of episod
e kramer takes advice of his caddie over his lawyer episode from seinfeld
the parents have there jaw dropping moment fun over fun it is screaming br
ing me back
1
```

In [ ]:

```
decoded_review = whatItSay(5)
```

```
! quite possibly how francis veber one of the best comedy directors in the
world at least when sticking to his native france managed to turn in a fil
m so completely unwatchable is beyond the reason of mere mortal man to dis
cern it's not just that the characters are so unlikeable or that the film
is so utterly devoid of even the lowest form of wit it's genuinely physica
lly painful to watch such an endless parade of inept writing acting and fi
lm making that you cannot believe this is the work of experienced and tale
nted filmmakers for once the near eternity spent in the cutting room and o
n the shelf before its blink and you'll miss it theatrical release tells t
he whole story what were they thinking
0
```

Preprocessing

In [ ]:

```
num_words = 20000
(X_train,Y_train),(X_test,Y_test) = imdb.load_data(num_words=num_words)
```

In [ ]:

```
maxlen=80
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

In [ ]:

```
print("X train shape: ",X_train.shape)
```

```
X train shape:  (25000, 80)
```

In [ ]:

```
print(X_train[5])
```

```
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     1   778   128    74    12   630   163    15     4  1766  7982
  1051     2    32    85   156    45    40   148   139   121   664   665
    10    10  1361   173     4   749     2    16  3804     8     4   226
    65    12    43   127    24 15344    10    10]
```

In [ ]:

```
for i in X_train[0:10]:
    print(len(i))
```

```
80
80
80
80
80
80
80
80
80
80
```

In [ ]:

```
decoded_review = whatItSay(5)
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
! begins better than it ends funny that the russian submarine crew ! all o
ther actors it's like those scenes where documentary shots br br spoiler p
art the message ! was contrary to the whole story it just does not mesh br
br
0
```

RNN Model 1

In [ ]:

```
rnn = Sequential()

rnn.add(Embedding(num_words,32,input_length =len(X_train[0]))) # num_words=15000
rnn.add(SimpleRNN(16,input_shape = (num_words,maxlen), return_sequences=False,activatio
n="relu"))
rnn.add(Dense(1)) #flatten
rnn.add(Activation("sigmoid")) #using sigmoid for binary classification

print(rnn.summary())
rnn.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accuracy"])
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 80, 32)            640000

 simple_rnn_1 (SimpleRNN)    (None, 16)                784

 dense_1 (Dense)             (None, 1)                 17

 activation_1 (Activation)   (None, 1)                 0

=================================================================
Total params: 640,801
Trainable params: 640,801
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:

```
history = rnn.fit(X_train,Y_train,validation_data = (X_test,Y_test),epochs = 5,batch_si
ze=128,verbose = 1)
```

```
Epoch 1/5
196/196 [==============================] - 8s 35ms/step - loss: 0.6692 - a
ccuracy: 0.5930 - val_loss: 0.6507 - val_accuracy: 0.6323
Epoch 2/5
196/196 [==============================] - 7s 35ms/step - loss: 0.5005 - a
ccuracy: 0.7727 - val_loss: 0.4561 - val_accuracy: 0.7965
Epoch 3/5
196/196 [==============================] - 7s 34ms/step - loss: 0.3624 - a
ccuracy: 0.8451 - val_loss: 0.3919 - val_accuracy: 0.8296
Epoch 4/5
196/196 [==============================] - 6s 32ms/step - loss: 0.2786 - a
ccuracy: 0.8863 - val_loss: 0.4382 - val_accuracy: 0.8084
Epoch 5/5
196/196 [==============================] - 6s 33ms/step - loss: 0.2292 - a
ccuracy: 0.9090 - val_loss: 0.3622 - val_accuracy: 0.8438
```

Accuracy

In [ ]:

```
score = rnn.evaluate(X_test,Y_test)
```

```
782/782 [==============================] - 4s 5ms/step - loss: 0.3622 - ac
curacy: 0.8438
```

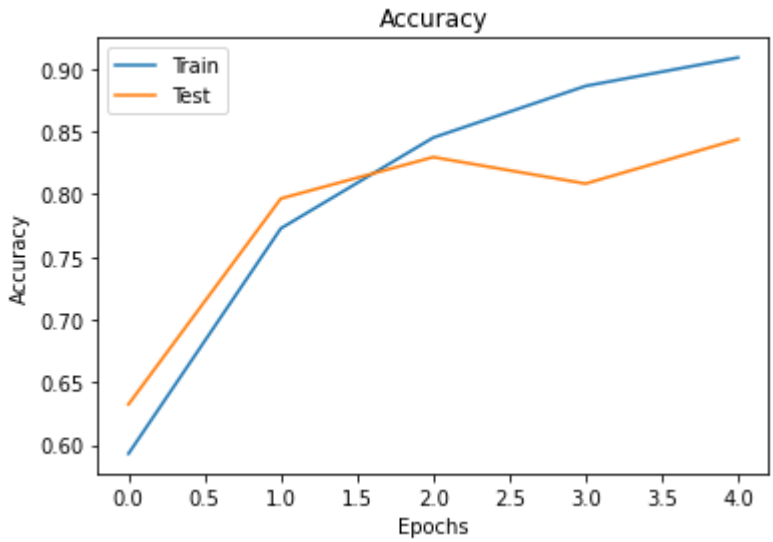
In [ ]:

```
print("accuracy:", score[1]*100)
```

```
accuracy: 84.38000082969666
```

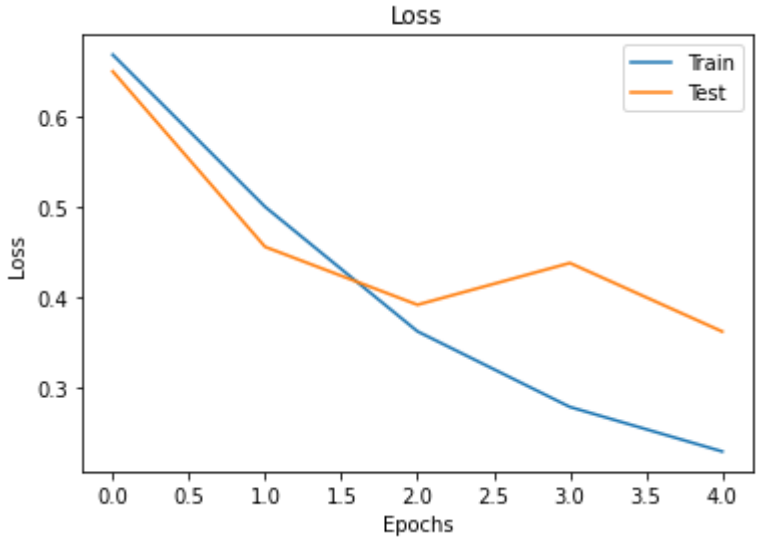
In [ ]:

```
plt.figure()
plt.plot(history.history["accuracy"],label="Train");
plt.plot(history.history["val_accuracy"],label="Test");
plt.title("Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```



In [ ]:

```
plt.figure()
plt.plot(history.history["loss"],label="Train");
plt.plot(history.history["val_loss"],label="Test");
plt.title("Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```



Different max_features and max_len

In [ ]:

```
num_words = 30000
(X_train,Y_train),(X_test,Y_test) = imdb.load_data(num_words=num_words)
```

In [ ]:

```
maxlen=130
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

In [ ]:

```
print("X train shape: ",X_train.shape)
```

```
X train shape:  (25000, 130)
```

In [ ]:

```
print(X_train[5])
```

```
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     1   778   128    74    12   630   163    15     4
  1766  7982  1051     2    32    85   156    45    40   148   139   121
   664   665    10    10  1361   173     4   749     2    16  3804     8
     4   226    65    12    43   127    24 15344    10    10]
```

In [ ]:

```
for i in X_train[0:10]:
    print(len(i))
```

```
130
130
130
130
130
130
130
130
130
130
```

In [ ]:

```
decoded_review = whatItSay(5)
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
! ! ! ! ! ! ! ! ! ! ! ! ! begins better than it ends funny that the russ
ian submarine crew ! all other actors it's like those scenes where documen
tary shots br br spoiler part the message ! was contrary to the whole stor
y it just does not mesh br br
0
```

RNN model2

In [ ]:

```
rnn = Sequential()

rnn.add(Embedding(num_words,32,input_length =len(X_train[0]))) # num_words=15000
rnn.add(SimpleRNN(16,input_shape = (num_words,maxlen), return_sequences=False,activatio
n="relu"))
rnn.add(Dense(1)) #flatten
rnn.add(Activation("sigmoid")) #using sigmoid for binary classification

print(rnn.summary())
rnn.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accuracy"])
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 130, 32)           960000

 simple_rnn_2 (SimpleRNN)    (None, 16)                784

 dense_2 (Dense)             (None, 1)                 17

 activation_2 (Activation)   (None, 1)                 0

=================================================================
Total params: 960,801
Trainable params: 960,801
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:

```
history = rnn.fit(X_train,Y_train,validation_data = (X_test,Y_test),epochs = 5,batch_si
ze=128,verbose = 1)
```

```
Epoch 1/5
196/196 [==============================] - 12s 55ms/step - loss: 0.6295 -
accuracy: 0.6558 - val_loss: 0.5207 - val_accuracy: 0.7932
Epoch 2/5
196/196 [==============================] - 10s 51ms/step - loss: 0.4494 -
accuracy: 0.8294 - val_loss: 0.4151 - val_accuracy: 0.8220
Epoch 3/5
196/196 [==============================] - 10s 49ms/step - loss: 0.3415 -
accuracy: 0.8604 - val_loss: 0.3927 - val_accuracy: 0.8316
Epoch 4/5
196/196 [==============================] - 10s 49ms/step - loss: 0.2844 -
accuracy: 0.8872 - val_loss: 1.4192 - val_accuracy: 0.6930
Epoch 5/5
196/196 [==============================] - 9s 48ms/step - loss: 0.2509 - a
ccuracy: 0.9032 - val_loss: 0.4282 - val_accuracy: 0.8183
```

Accuracy

In [ ]:

```
score = rnn.evaluate(X_test,Y_test)
```

```
782/782 [==============================] - 7s 9ms/step - loss: 0.4282 - ac
curacy: 0.8183
```

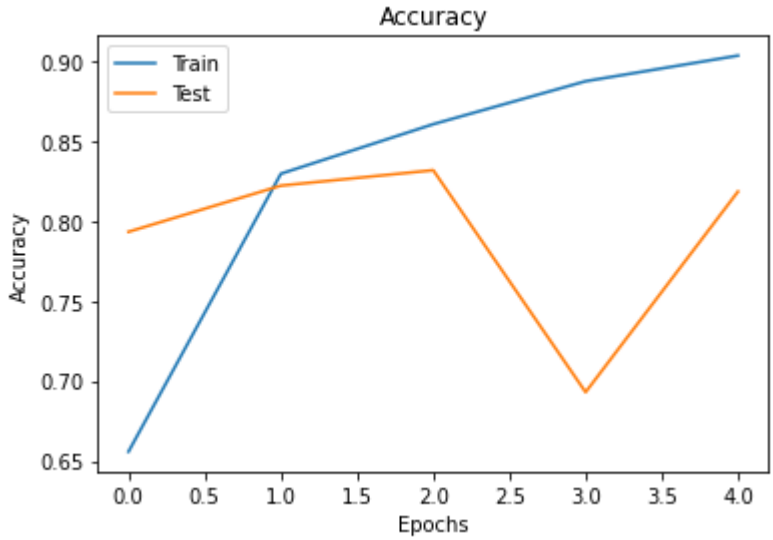
In [ ]:

```
print("accuracy:", score[1]*100)
```

```
accuracy: 81.83199763298035
```

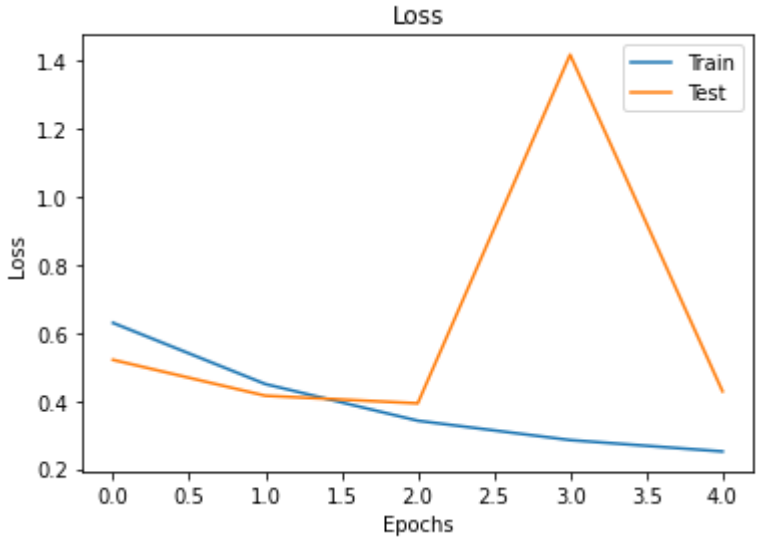
In [ ]:

```
plt.figure()
plt.plot(history.history["accuracy"],label="Train");
plt.plot(history.history["val_accuracy"],label="Test");
plt.title("Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```



In [ ]:

```
plt.figure()
plt.plot(history.history["loss"],label="Train");
plt.plot(history.history["val_loss"],label="Test");
plt.title("Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```



In [ ]:

```
num_words = 40000
(X_train,Y_train),(X_test,Y_test) = imdb.load_data(num_words=num_words)
```

In [ ]:

```
maxlen=180
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

In [ ]:

```
print("X train shape: ",X_train.shape)
```

```
X train shape:  (25000, 180)
```

In [ ]:

```
print(X_train[5])
```

```
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1   778   128    74    12   630   163
    15     4  1766  7982  1051     2    32    85   156    45    40   148
   139   121   664   665    10    10  1361   173     4   749     2    16
  3804     8     4   226    65    12    43   127    24 15344    10    10]
```

In [ ]:

```
for i in X_train[0:10]:
    print(len(i))
```

```
180
180
180
180
180
180
180
180
180
180
```

In [ ]:

```
decoded_review = whatItSay(5)
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! begins better than i
t ends funny that the russian submarine crew ! all other actors it's like
those scenes where documentary shots br br spoiler part the message ! was
contrary to the whole story it just does not mesh br br
0
```

RNN Model3

In [ ]:

```
rnn = Sequential()

rnn.add(Embedding(num_words,32,input_length =len(X_train[0]))) # num_words=15000
rnn.add(SimpleRNN(16,input_shape = (num_words,maxlen), return_sequences=False,activatio
n="relu"))
rnn.add(Dense(1)) #flatten
rnn.add(Activation("sigmoid")) #using sigmoid for binary classification

print(rnn.summary())
rnn.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accuracy"])
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 180, 32)           1280000

 simple_rnn_3 (SimpleRNN)    (None, 16)                784

 dense_3 (Dense)             (None, 1)                 17

 activation_3 (Activation)   (None, 1)                 0

=================================================================
Total params: 1,280,801
Trainable params: 1,280,801
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:

```
history = rnn.fit(X_train,Y_train,validation_data = (X_test,Y_test),epochs = 5,batch_si
ze=128,verbose = 1)
```

```
Epoch 1/5
196/196 [==============================] - 23s 109ms/step - loss: 0.5950 -
accuracy: 0.6995 - val_loss: 0.4741 - val_accuracy: 0.8181
Epoch 2/5
196/196 [==============================] - 16s 83ms/step - loss: 0.3874 -
accuracy: 0.8511 - val_loss: 0.4826 - val_accuracy: 0.7725
Epoch 3/5
196/196 [==============================] - 15s 74ms/step - loss: 0.3048 -
accuracy: 0.8810 - val_loss: 0.3645 - val_accuracy: 0.8431
Epoch 4/5
196/196 [==============================] - 13s 68ms/step - loss: 0.2460 -
accuracy: 0.9052 - val_loss: 0.3975 - val_accuracy: 0.8316
Epoch 5/5
196/196 [==============================] - 14s 72ms/step - loss: 0.2143 -
accuracy: 0.9191 - val_loss: 0.3606 - val_accuracy: 0.8498
```

Accuracy

In [ ]:

```
score = rnn.evaluate(X_test,Y_test)
```

```
782/782 [==============================] - 9s 12ms/step - loss: 0.3606 - a
ccuracy: 0.8498
```

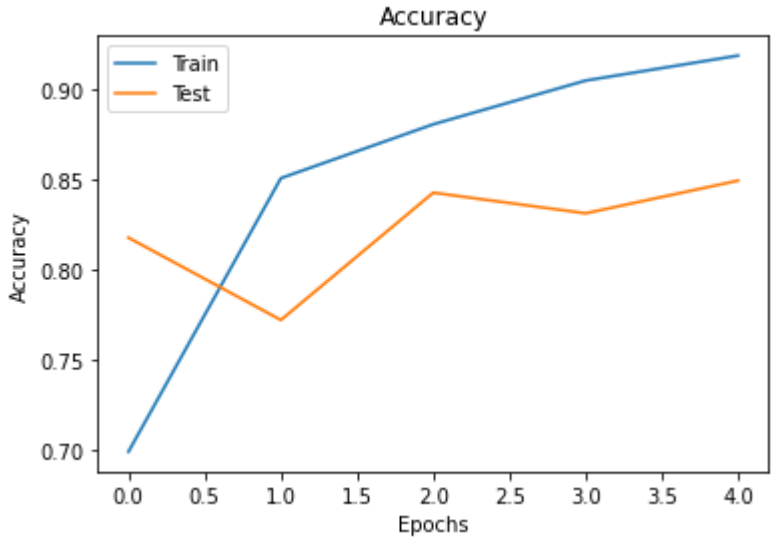
In [ ]:

```
print("accuracy:", score[1]*100)
```

```
accuracy: 84.97599959373474
```

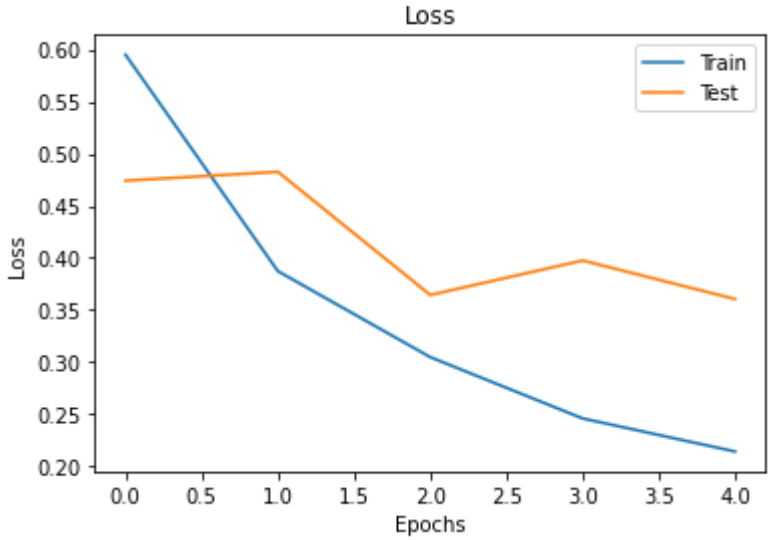
In [ ]:

```
plt.figure()
plt.plot(history.history["accuracy"],label="Train");
plt.plot(history.history["val_accuracy"],label="Test");
plt.title("Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```



In [ ]:

```
plt.figure()
plt.plot(history.history["loss"],label="Train");
plt.plot(history.history["val_loss"],label="Test");
plt.title("Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```



As we increased max_features from 20000 to 30000 and max_len from 80 to 130 accuracy increased

But when further increasing the accuracy decreased