In [2]:

```
from keras.datasets import imdb
```

Data Preparation

In [4]:

```
((XT,YT),(Xt,Yt)) = imdb.load_data(num_words=10000)
```

In [27]:

```
print("Training Set",len(XT))
print("Test Set",len(Xt))
```

```
Training Set 25000
Test Set 25000
```

In [9]:

```
print(XT[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 
5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2
13, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 
4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 2
4, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 4
8, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30
92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
```

In [14]:

```
word_index = imdb.get_word_index() # word : indx
```

In [86]:

```
#print(word_index.items())
```

In [16]:

```
# Creating indx:word dict
indx_word_dict = { value:key for (key,value) in word_index.items()}
```

In [85]:

```
print(type(word_index))
print(type(indx_word_dict))
#print(indx_word_dict)
```

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

In [21]:

```
actual_review = " ".join(indx_word_dict.get(idx-3,"?") for idx in XT[0])
```

In [22]:

```
print(actual_review)
```

```
? this film was just brilliant casting location scenery story direction everyone's really suited the part they
magine being there robert ? is an amazing actor and now the same being director ? father came from the same sc
loved the fact there was a real connection with this film the witty remarks throughout the film were great it
hat i bought the film as soon as it was released for ? and would recommend it to everyone to watch and the fly
cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this
e two little boy's that played the ? of norman and paul they were just brilliant children are often left out o
the stars that play them all grown up are such a big profile for the whole film but these children are amazing
hat they have done don't you think the whole story was so lovely because it was true and was someone's life af
us all
```

Vectorize the data

Vocab Size- 10,000 we will make sure every sentence is represented by a vector of len 100

In [28]:

```
import numpy as np
```

In [31]:

```
def vectorize_sentences(sentences,dim=10000):

    outputs = np.zeros((len(sentences),dim))

    for i,idx in enumerate(sentences):
        outputs[i,idx] = 1

    return outputs
```

In [32]:

```
X_Train = vectorize_sentences(XT)
X_Test = vectorize_sentences(Xt)
```

In [33]:

```
print(X_Train.shape)
print(X_Test.shape)
```

```
(25000, 10000)
(25000, 10000)
```

In [37]:

```
Y_Train = np.array(YT).astype('float32')
Y_Test = np.array(Yt).astype('float32')
```

Build a Network

- Use Fully Connected/Dense Layers with RELU Activation
- 2 hidden layers with 16 unit each
- 1 Output layer with 1 unit (Sigmoid Activation)

In [70]:

```
from keras import models
from keras.layers import Dense
```

In [71]:

```
# Define the model
model = models.Sequential()
model.add(Dense(16,activation='relu',input_shape=(10000,)))
model.add(Dense(16,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

In [72]:

```
# Compile the Model
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
```

In [73]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 16)                160016
_________________________________________________________________
dense_5 (Dense)              (None, 16)                272
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
_________________________________________________________________
```

Training and Validation

In [74]:

```
# 20000 ex in Training Set, 5000 ex in validation set
X_val = X_Train[:5000]
X_Train_New = X_Train[5000:]

Y_val = Y_Train[:5000]
Y_Train_New = Y_Train[5000:]
```

In [45]:

```
history = model.fit(X_Train_New,Y_Train_New,epochs=20,batch_size=512,validation_data=(
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/20
20000/20000 [==============================] - 11s 551us/step - loss: 0.4914 - accuracy: 0.8070 - val_loss: 0.
Epoch 2/20
20000/20000 [==============================] - 5s 262us/step - loss: 0.2817 - accuracy: 0.9064 - val_loss: 0.3
Epoch 3/20
20000/20000 [==============================] - 5s 259us/step - loss: 0.2127 - accuracy: 0.9287 - val_loss: 0.2
Epoch 4/20
20000/20000 [==============================] - 5s 259us/step - loss: 0.1744 - accuracy: 0.9392 - val_loss: 0.2
Epoch 5/20
20000/20000 [==============================] - 5s 270us/step - loss: 0.1472 - accuracy: 0.9486 - val_loss: 0.2
Epoch 6/20
20000/20000 [==============================] - 5s 251us/step - loss: 0.1245 - accuracy: 0.9578 - val_loss: 0.3
Epoch 7/20
20000/20000 [==============================] - 6s 279us/step - loss: 0.1080 - accuracy: 0.9639 - val_loss: 0.3
Epoch 8/20
20000/20000 [==============================] - 5s 263us/step - loss: 0.0930 - accuracy: 0.9697 - val_loss: 0.3
Epoch 9/20
20000/20000 [==============================] - 5s 265us/step - loss: 0.0799 - accuracy: 0.9754 - val_loss: 0.4
Epoch 10/20
20000/20000 [==============================] - 5s 259us/step - loss: 0.0699 - accuracy: 0.9783 - val_loss: 0.3
Epoch 11/20
20000/20000 [==============================] - 5s 263us/step - loss: 0.0612 - accuracy: 0.9825 - val_loss: 0.4
Epoch 12/20
20000/20000 [==============================] - 5s 260us/step - loss: 0.0510 - accuracy: 0.9851 - val_loss: 0.4
Epoch 13/20
20000/20000 [==============================] - 5s 258us/step - loss: 0.0451 - accuracy: 0.9880 - val_loss: 0.4
Epoch 14/20
20000/20000 [==============================] - 5s 257us/step - loss: 0.0394 - accuracy: 0.9895 - val_loss: 0.5
Epoch 15/20
20000/20000 [==============================] - 5s 254us/step - loss: 0.0300 - accuracy: 0.9930 - val_loss: 0.5
Epoch 16/20
20000/20000 [==============================] - 5s 253us/step - loss: 0.0270 - accuracy: 0.9937 - val_loss: 0.5
Epoch 17/20
20000/20000 [==============================] - 5s 255us/step - loss: 0.0250 - accuracy: 0.9939 - val_loss: 0.6
Epoch 18/20
20000/20000 [==============================] - 5s 257us/step - loss: 0.0208 - accuracy: 0.9955 - val_loss: 0.6
Epoch 19/20
20000/20000 [==============================] - 5s 258us/step - loss: 0.0151 - accuracy: 0.9976 - val_loss: 0.7
Epoch 20/20
20000/20000 [==============================] - 5s 254us/step - loss: 0.0166 - accuracy: 0.9961 - val_loss: 0.7
```

Visualize Our Results

In [46]:

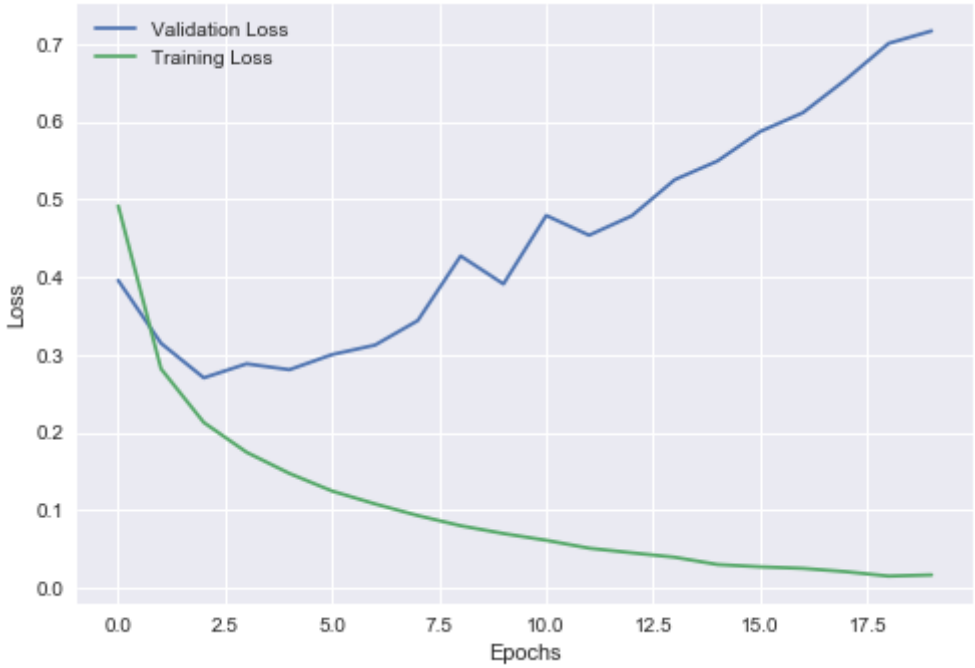
```
import matplotlib.pyplot as plt
```

In [48]:

```
h = history.history # dict containing list of loss, acc, validation loss, validation a
```

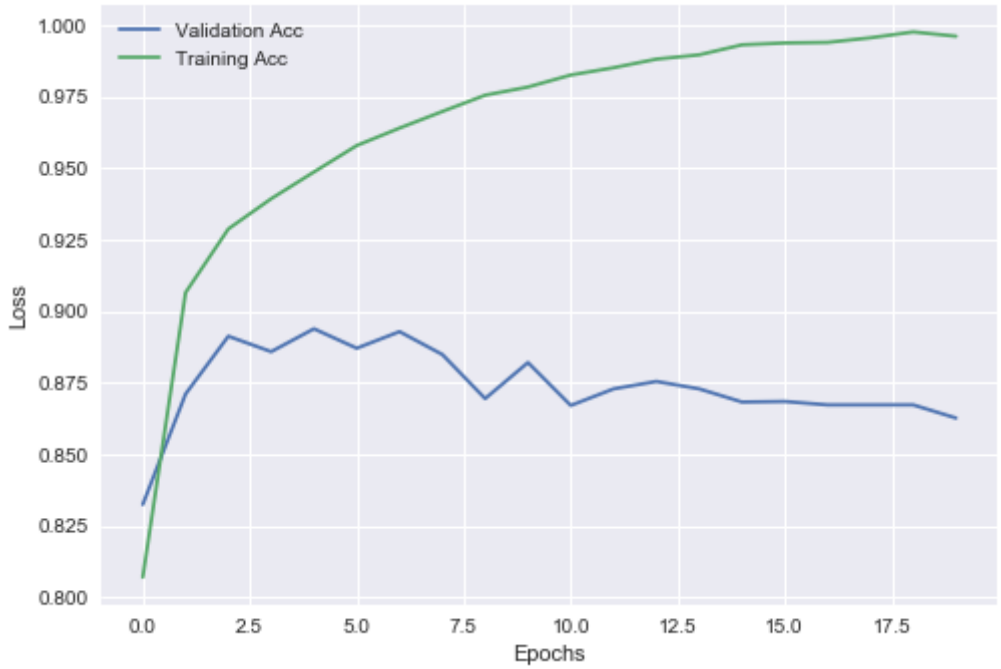
In [51]:

```
plt.style.use('seaborn')
plt.plot(h['val_loss'],label='Validation Loss')
plt.plot(h['loss'],label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [54]:

```
plt.plot(h['val_accuracy'],label='Validation Acc')
plt.plot(h['accuracy'],label='Training Acc')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [75]:

```
# Val Acc inc till around 4 epochs, then decrease.. So we stop(train) at 4 Epoch only
history2 = model.fit(X_Train_New,Y_Train_New,epochs=4,batch_size=512,validation_data=(
```

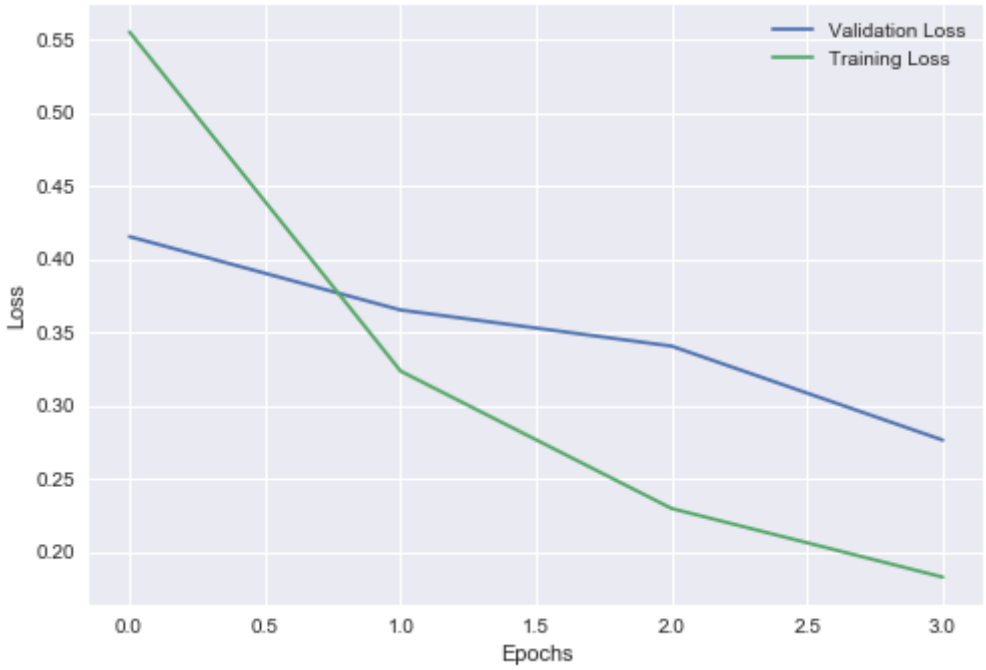
```
Train on 20000 samples, validate on 5000 samples
Epoch 1/4
20000/20000 [==============================] - 14s 698us/step - loss: 0.5550 - accuracy: 0.7508 - val_loss: 0.4
Epoch 2/4
20000/20000 [==============================] - 5s 258us/step - loss: 0.3234 - accuracy: 0.9016 - val_loss: 0.3
Epoch 3/4
20000/20000 [==============================] - 5s 247us/step - loss: 0.2295 - accuracy: 0.9262 - val_loss: 0.3
Epoch 4/4
20000/20000 [==============================] - 5s 247us/step - loss: 0.1826 - accuracy: 0.9402 - val_loss: 0.2
```

In [79]:

```
h2 = history2.history
```

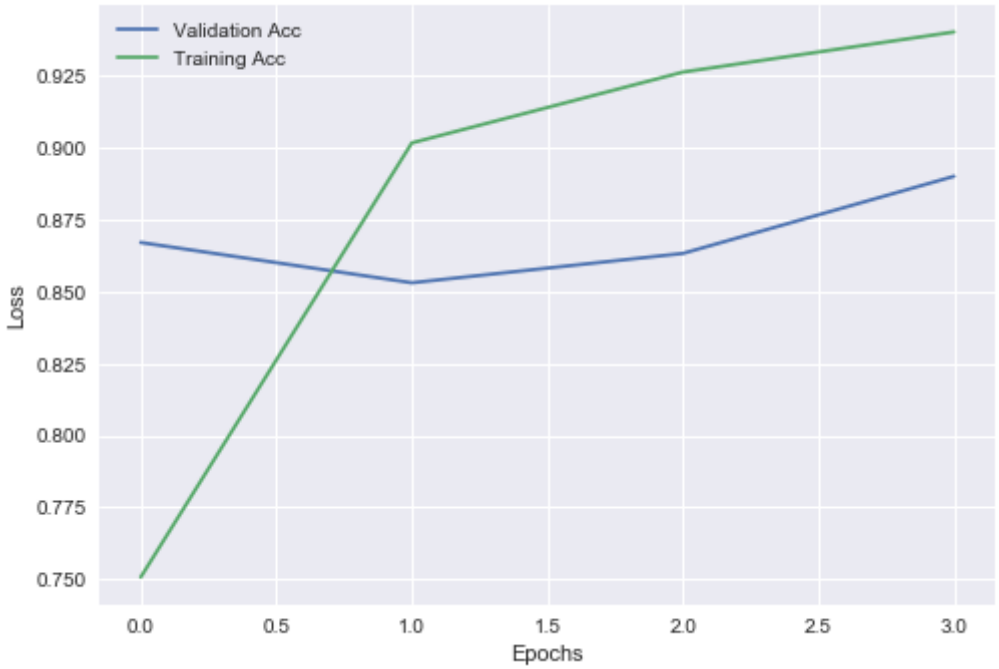
In [80]:

```
plt.plot(h2['val_loss'],label='Validation Loss')
plt.plot(h2['loss'],label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [81]:

```
plt.plot(h2['val_accuracy'],label='Validation Acc')
plt.plot(h2['accuracy'],label='Training Acc')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [82]:

```
model.evaluate(X_Test,Y_Test)[1] # Test Set Acc
```

```
 25000/25000 [==============================] - 6s 250us/step
```

```
 0.8838000297546387
```

In [83]:

```
model.evaluate(X_Train,Y_Train)[1] # Train Set Acc
```

```
 25000/25000 [==============================] - 6s 239us/step
```

```
 0.9437999725341797
```

In [87]:

```
model.predict(X_Test)
```

```
array([[0.28385603],
       [0.9999605 ],
       [0.8379575 ],
       ...,
       [0.15922514],
       [0.13209721],
       [0.60513496]], dtype=float32)
```

In [ ]: