In [1]:

```
from keras.datasets import imdb
# Load the data, keeping only 10,000 of the most frequently occuring words
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
asets/imdb.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/
imdb.npz)
17464789/17464789 [==============================] - 0s 0us/step
```

In [2]:

```
# Here is a list of maximum indexes in every review
print(type([max(sequence) for sequence in train_data]))
# Find the maximum of all max indexes
max([max(sequence) for sequence in train_data])
```

```
<class 'list'>
```

Out[2]:

```
9999
```

In [3]:

```
# step 1: load the dictionary mappings from word to integer index
word_index = imdb.get_word_index()
# step 2: reverse word index to map integer indexes to their respective words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
# Step 3: decode the review, mapping integer indices to words
#
# indices are off by 3 because 0, 1, and 2 are reserverd indices for "padding", "Start of
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]])
decoded_review
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
asets/imdb_word_index.json (https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/imdb_word_index.json)
1641221/1641221 [==============================] - 0s 0us/step
```

Out[3]:

```
"? this film was just brilliant casting location scenery story direction eve
ryone's really suited the part they played and you could just imagine being
there robert ? is an amazing actor and now the same being director ? father
came from the same scottish island as myself so i loved the fact there was a
real connection with this film the witty remarks throughout the film were gr
eat it was just brilliant so much that i bought the film as soon as it was r
eleased for ? and would recommend it to everyone to watch and the fly fishin
g was amazing really cried at the end it was so sad and you know what they s
ay if you cry at a film it must have been good and this definitely was also
? to the two little boy's that played the ? of norman and paul they were jus
t brilliant children are often left out of the ? list i think because the st
ars that play them all grown up are such a big profile for the whole film bu
t these children are amazing and should be praised for what they have done d
on't you think the whole story was so lovely because it was true and was som
eone's life after all that was shared with us all"
```

In [4]:

```
# Vectorize input data
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i,sequence in enumerate(sequences):
      results[i,sequence] = 1
    return results
# Vectorize training Data
X_train = vectorize_sequences(train_data)
# Vectorize testing Data
X_test = vectorize_sequences(test_data)
```

In [5]:

```
X_train[0]
```

Out[5]:

```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [6]:

```
X_train.shape
```

Out[6]:

```
(25000, 10000)
```

In [7]:

```
# Vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

In [10]:

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [11]:

```
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import metrics
model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

In [12]:

```
# Input for Validation
X_val = X_train[:10000]
partial_X_train = X_train[10000:]
# Labels for validation
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [13]:

```
history = model.fit(partial_X_train, partial_y_train, epochs=20, batch_size=512, verbose =1
```

```
Epoch 1/20
30/30 [==============================] - 2s 57ms/step - loss: 0.5501 - binar
y_accuracy: 0.7738 - val_loss: 0.4242 - val_binary_accuracy: 0.8604
Epoch 2/20
30/30 [==============================] - 1s 34ms/step - loss: 0.3422 - binar
y_accuracy: 0.8918 - val_loss: 0.3225 - val_binary_accuracy: 0.8819
Epoch 3/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2525 - binar
y_accuracy: 0.9183 - val_loss: 0.2964 - val_binary_accuracy: 0.8827
Epoch 4/20
30/30 [==============================] - 1s 32ms/step - loss: 0.2058 - binar
y_accuracy: 0.9322 - val_loss: 0.2769 - val_binary_accuracy: 0.8905
Epoch 5/20
30/30 [==============================] - 2s 56ms/step - loss: 0.1724 - binar
y_accuracy: 0.9407 - val_loss: 0.2762 - val_binary_accuracy: 0.8862
Epoch 6/20
30/30 [==============================] - 1s 48ms/step - loss: 0.1451 - binar
y_accuracy: 0.9528 - val_loss: 0.2816 - val_binary_accuracy: 0.8884
Epoch 7/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1220 - binar
y_accuracy: 0.9627 - val_loss: 0.2965 - val_binary_accuracy: 0.8852
Epoch 8/20
30/30 [==============================] - 1s 32ms/step - loss: 0.1050 - binar
y_accuracy: 0.9693 - val_loss: 0.3069 - val_binary_accuracy: 0.8853
Epoch 9/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0922 - binar
y_accuracy: 0.9728 - val_loss: 0.3223 - val_binary_accuracy: 0.8843
Epoch 10/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0748 - binar
y_accuracy: 0.9806 - val_loss: 0.3440 - val_binary_accuracy: 0.8819
Epoch 11/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0663 - binar
y_accuracy: 0.9827 - val_loss: 0.3582 - val_binary_accuracy: 0.8777
Epoch 12/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0548 - binar
y_accuracy: 0.9867 - val_loss: 0.3795 - val_binary_accuracy: 0.8804
Epoch 13/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0499 - binar
y_accuracy: 0.9881 - val_loss: 0.3949 - val_binary_accuracy: 0.8740
Epoch 14/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0366 - binar
y_accuracy: 0.9931 - val_loss: 0.4534 - val_binary_accuracy: 0.8732
Epoch 15/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0308 - binar
y_accuracy: 0.9948 - val_loss: 0.4701 - val_binary_accuracy: 0.8730
Epoch 16/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0249 - binar
y_accuracy: 0.9964 - val_loss: 0.4665 - val_binary_accuracy: 0.8761
Epoch 17/20
30/30 [==============================] - 2s 52ms/step - loss: 0.0220 - binar
y_accuracy: 0.9979 - val_loss: 0.4935 - val_binary_accuracy: 0.8744
Epoch 18/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0214 - binar
y_accuracy: 0.9959 - val_loss: 0.5008 - val_binary_accuracy: 0.8740
Epoch 19/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0147 - binar
y_accuracy: 0.9984 - val_loss: 0.5204 - val_binary_accuracy: 0.8723
```

```
Epoch 20/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0123 - binar
y_accuracy: 0.9983 - val_loss: 0.5402 - val_binary_accuracy: 0.8733
```

In [14]:

```
history_dict = history.history
history_dict.keys()
```

Out[14]:

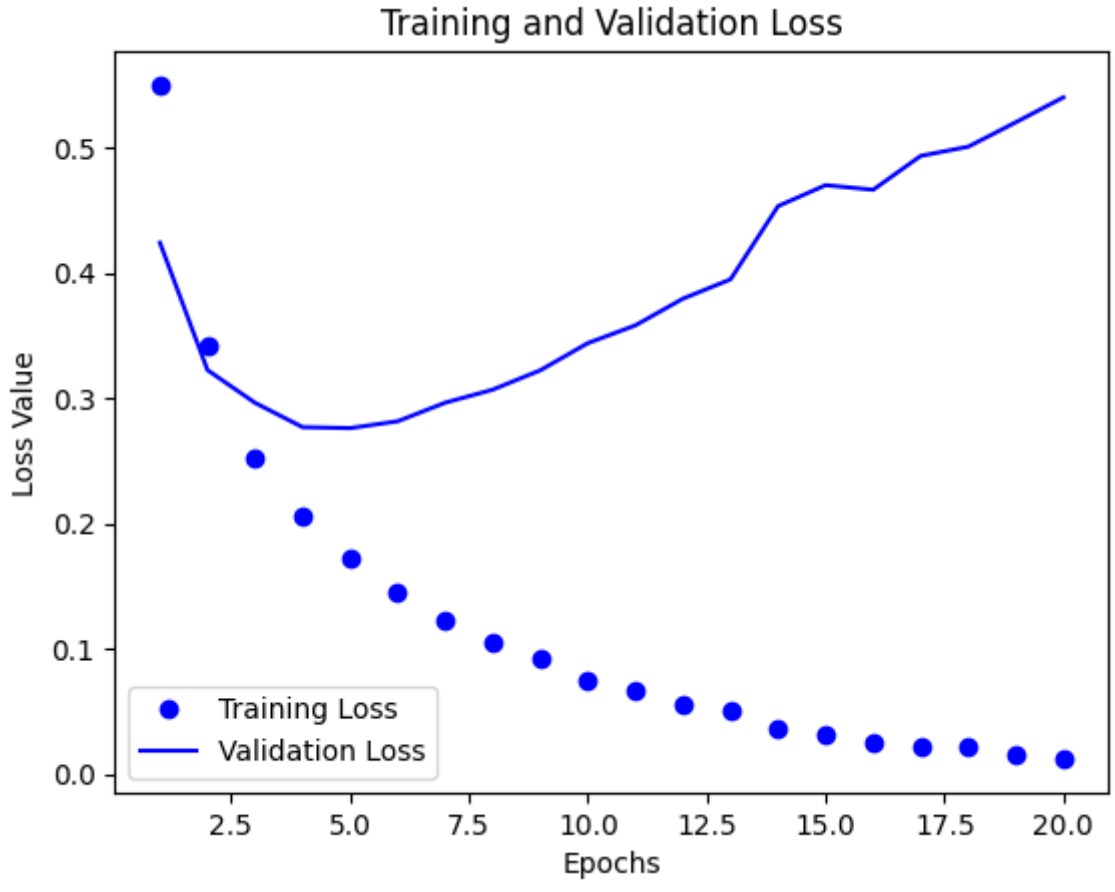
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

In [15]:

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

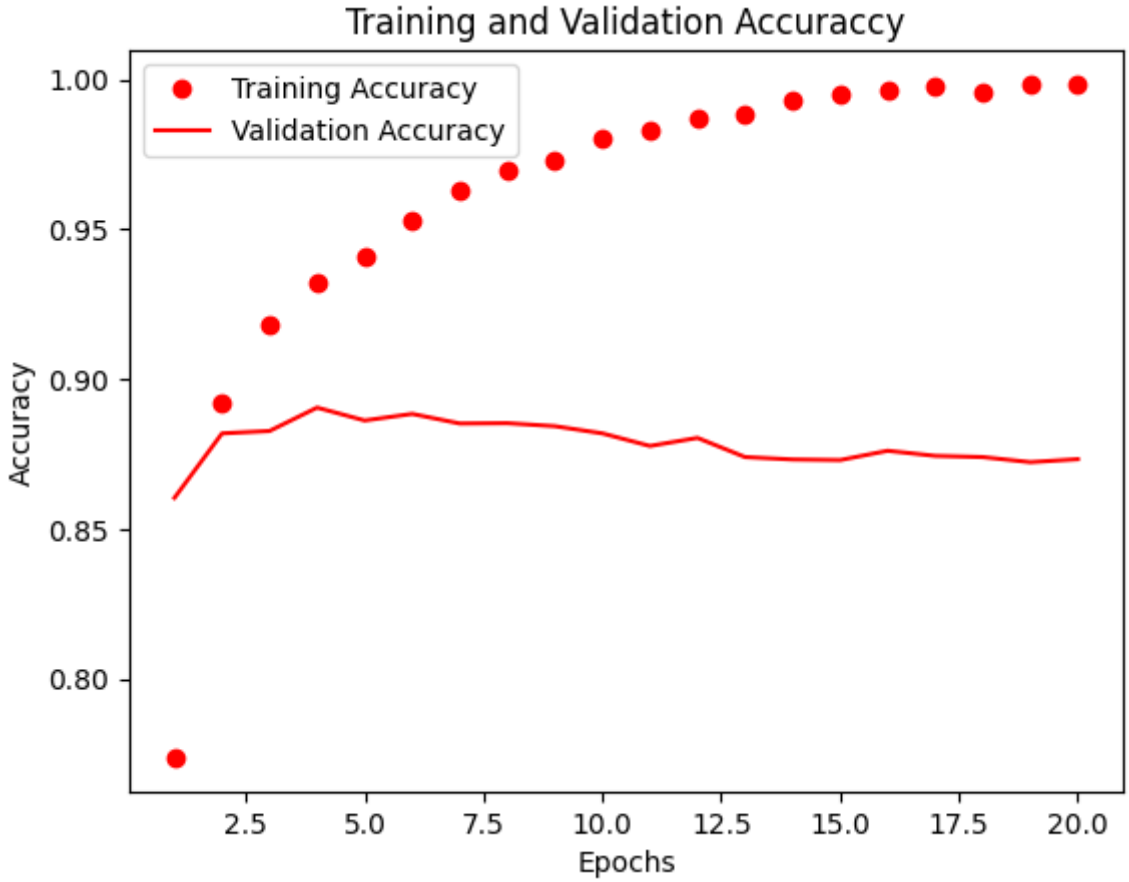
In [16]:

```
# Plotting losses
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label="Training Loss")
plt.plot(epochs, val_loss_values, 'b', label="Validation Loss")
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss Value')
plt.legend()
plt.show()
```



In [17]:

```
# Training and Validation Accuracy
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, acc_values, 'ro', label="Training Accuracy")
plt.plot(epochs, val_acc_values, 'r', label="Validation Accuracy")
plt.title('Training and Validation Accuraccy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [18]:

```
model.fit(partial_X_train, partial_y_train, epochs=3, batch_size=512, validation_data=(X_va
```

```
Epoch 1/3
30/30 [==============================] - 2s 51ms/step - loss: 0.0107 - binar
y_accuracy: 0.9984 - val_loss: 0.6217 - val_binary_accuracy: 0.8661
Epoch 2/3
30/30 [==============================] - 2s 59ms/step - loss: 0.0067 - binar
y_accuracy: 0.9998 - val_loss: 0.5804 - val_binary_accuracy: 0.8718
Epoch 3/3
30/30 [==============================] - 1s 44ms/step - loss: 0.0097 - binar
y_accuracy: 0.9987 - val_loss: 0.5961 - val_binary_accuracy: 0.8718
```

Out[18]:

```
<keras.callbacks.History at 0x7f608fb17c70>
```

In [19]:

```
# Making Predictions for testing data
np.set_printoptions(suppress=True)
result = model.predict(X_test)
```

```
782/782 [==============================] - 2s 2ms/step
```

In [20]:

```
result
```

Out[20]:

```
array([[0.00910641],
       [1.        ],
       [0.36453325],
       ...,
       [0.00241602],
       [0.00917078],
       [0.95490444]], dtype=float32)
```

In [21]:

```
y_pred = np.zeros(len(result))
for i, score in enumerate(result):
    y_pred[i] = 1 if score > 0.5 else 0
```

In [22]:

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_pred, y_test)
```

In [23]:

```
# error
mae
```

Out[23]:

```
0.14148
```

In [ ]: