In [7]:
```
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 = 100
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/im
db.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz)
17464789/17464789 [==============================] - 8s 0us/step
```

In [8]:
```
# Check the first review
train_data[0]
```

Out[8]:
```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
```

In [9]:
```
# Check the first label
train_labels[0]
```

Out[9]:
```
1
```

In [10]:
```
# Since we restricted ourselves to the top 10000 frequent words, no word index should
# we'll verify this below

# Here is a list of maximum indexes in every review --- we search the maximum index i
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[10]:
```
9999
```

In [11]:
```
# Let's quickly decode a review

# 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", "Star
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-datasets/im
db_word_index.json (https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb
_word_index.json)
1641221/1641221 [==============================] - 3s 2us/step
```

Out[11]:
```
"? this film was just brilliant casting location scenery story direction everyone's
really suited the part they played and you could just imagine being there robert ? i
s an amazing actor and now the same being director ? father came from the same scott
ish island as myself so i loved the fact there was a real connection with this film
the witty remarks throughout the film were great it was just brilliant so much that
i bought the film as soon as it was released for ? and would recommend it to everyon
e to watch and the fly fishing was amazing really 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 definite
ly was also ? to the two little boy's that played the ? of norman and paul they were
just brilliant children are often left out of the ? list i think because the stars t
hat play them all grown up are such a big profile for the whole film but these child
ren are amazing and should be praised for what they have done don't you think the wh
ole story was so lovely because it was true and was someone's life after all that wa
s shared with us all"
```

In [12]:
```
len(reverse_word_index)
```

Out[12]:
```
88584
```

In [13]:
```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))    # Creates an all zero matrix o
    for i,sequence in enumerate(sequences):
        results[i,sequence] = 1                        # Sets specific indices of res
    return results

# Vectorize training Data
X_train = vectorize_sequences(train_data)

# Vectorize testing Data
X_test = vectorize_sequences(test_data)
```

In [14]:
```
X_train[0]
```

Out[14]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [15]:
```
X_train.shape
```

Out[15]:
```
(25000, 10000)
```

In [16]:
```
y_train = np.asarray(train_labels).astype('float32')
y_test  = np.asarray(test_labels).astype('float32')
```

In [17]:
```
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 [19]:
```
from keras import optimizers
from keras import losses
from keras import metrics

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

In [20]:
```
# 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 [21]:
```
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(X_val, y_val))
```

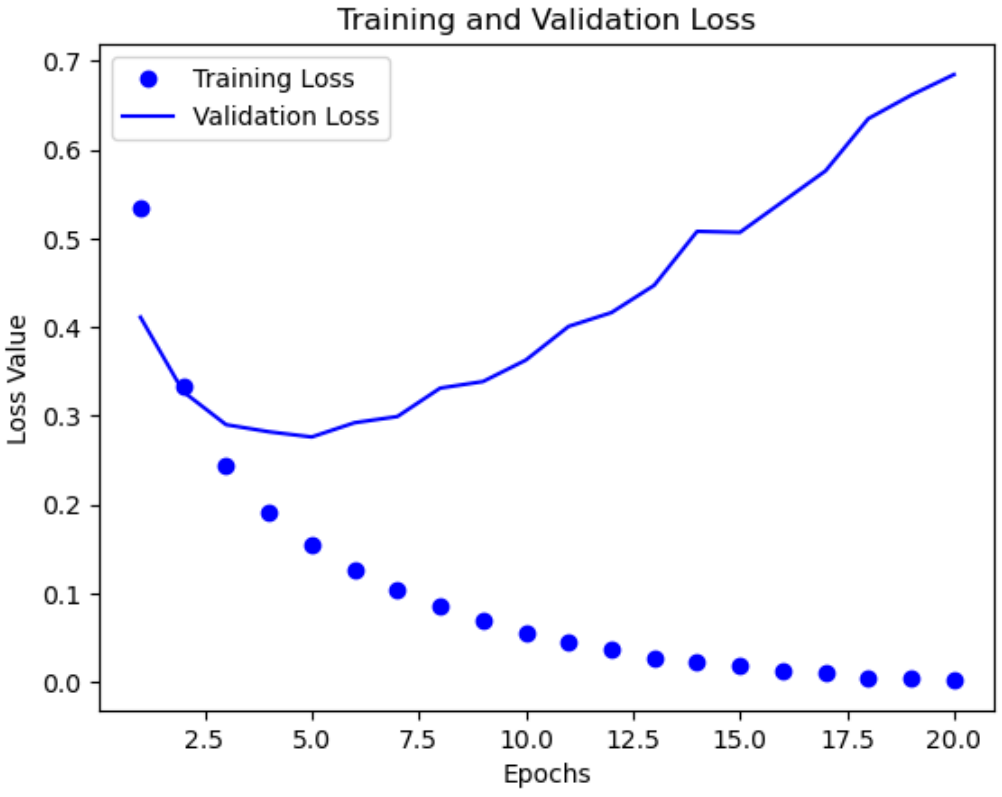
```
Epoch 1/20
30/30 [==============================] - 2s 45ms/step - loss: 0.5339 - binary_accura
cy: 0.7831 - val_loss: 0.4112 - val_binary_accuracy: 0.8661
Epoch 2/20
30/30 [==============================] - 1s 29ms/step - loss: 0.3329 - binary_accura
cy: 0.9001 - val_loss: 0.3272 - val_binary_accuracy: 0.8819
Epoch 3/20
30/30 [==============================] - 1s 23ms/step - loss: 0.2448 - binary_accura
cy: 0.9233 - val_loss: 0.2900 - val_binary_accuracy: 0.8872
Epoch 4/20
30/30 [==============================] - 1s 21ms/step - loss: 0.1912 - binary_accura
cy: 0.9413 - val_loss: 0.2820 - val_binary_accuracy: 0.8853
Epoch 5/20
30/30 [==============================] - 1s 23ms/step - loss: 0.1540 - binary_accura
cy: 0.9533 - val_loss: 0.2762 - val_binary_accuracy: 0.8913
Epoch 6/20
30/30 [==============================] - 1s 24ms/step - loss: 0.1265 - binary_accura
cy: 0.9615 - val_loss: 0.2922 - val_binary_accuracy: 0.8862
Epoch 7/20
30/30 [==============================] - 1s 21ms/step - loss: 0.1034 - binary_accura
cy: 0.9717 - val_loss: 0.2992 - val_binary_accuracy: 0.8857
Epoch 8/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0849 - binary_accura
cy: 0.9773 - val_loss: 0.3312 - val_binary_accuracy: 0.8769
Epoch 9/20
30/30 [==============================] - 1s 24ms/step - loss: 0.0698 - binary_accura
cy: 0.9827 - val_loss: 0.3386 - val_binary_accuracy: 0.8811
Epoch 10/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0561 - binary_accura
cy: 0.9871 - val_loss: 0.3628 - val_binary_accuracy: 0.8815
Epoch 11/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0444 - binary_accura
cy: 0.9912 - val_loss: 0.4008 - val_binary_accuracy: 0.8783
Epoch 12/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0372 - binary_accura
cy: 0.9927 - val_loss: 0.4163 - val_binary_accuracy: 0.8769
Epoch 13/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0265 - binary_accura
cy: 0.9959 - val_loss: 0.4471 - val_binary_accuracy: 0.8753
Epoch 14/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0220 - binary_accura
cy: 0.9970 - val_loss: 0.5078 - val_binary_accuracy: 0.8718
Epoch 15/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0183 - binary_accura
cy: 0.9965 - val_loss: 0.5067 - val_binary_accuracy: 0.8727
Epoch 16/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0118 - binary_accura
cy: 0.9987 - val_loss: 0.5413 - val_binary_accuracy: 0.8724
Epoch 17/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0103 - binary_accura
cy: 0.9987 - val_loss: 0.5760 - val_binary_accuracy: 0.8708
Epoch 18/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0053 - binary_accura
cy: 0.9996 - val_loss: 0.6348 - val_binary_accuracy: 0.8646
Epoch 19/20
30/30 [==============================] - 1s 25ms/step - loss: 0.0055 - binary_accura
cy: 0.9995 - val_loss: 0.6613 - val_binary_accuracy: 0.8641
Epoch 20/20
30/30 [==============================] - 1s 24ms/step - loss: 0.0027 - binary_accura
cy: 0.9999 - val_loss: 0.6845 - val_binary_accuracy: 0.8675
```

In [22]:
```
history_dict = history.history
history_dict.keys()
```

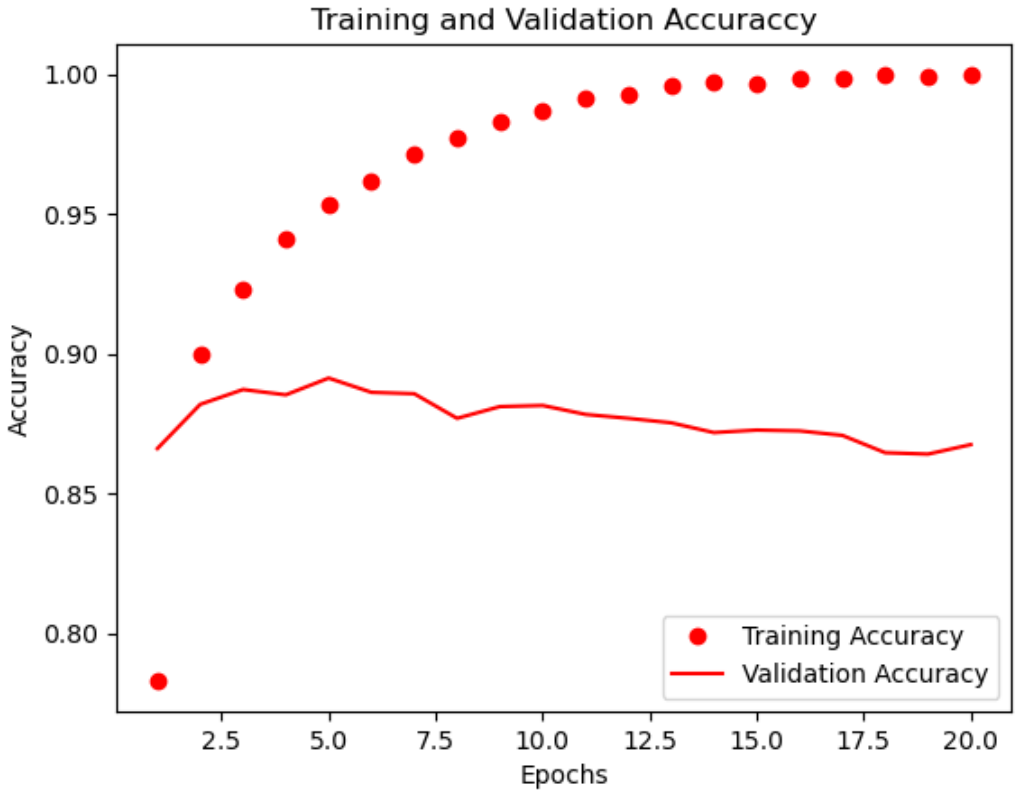
Out[22]:
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

In [23]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [24]:
```
# 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 [25]:
```
# 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 [26]:
```
model.fit(partial_X_train,
                    partial_y_train,
                    epochs=3,
                    batch_size=512,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/3
30/30 [==============================] - 1s 33ms/step - loss: 0.0021 - binary_accura
cy: 0.9999 - val_loss: 0.7997 - val_binary_accuracy: 0.8623
Epoch 2/3
30/30 [==============================] - 1s 21ms/step - loss: 0.0021 - binary_accura
cy: 0.9998 - val_loss: 0.7724 - val_binary_accuracy: 0.8667
Epoch 3/3
30/30 [==============================] - 1s 25ms/step - loss: 8.9476e-04 - binary_ac
curacy: 1.0000 - val_loss: 0.8192 - val_binary_accuracy: 0.8670
```

Out[26]:
```
<keras.callbacks.History at 0x1f286022eb0>
```

In [35]:
```
# Making Predictions for testing data
np.set_printoptions(suppress=True)
result = model.predict(X_test)
```

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

In [36]:
```
result
```

Out[36]:
```
array([[0.00977155],
       [1.        ],
       [0.9957733 ],
       ...,
       [0.00096024],
       [0.00696085],
       [0.8679086 ]], dtype=float32)
```

In [37]:
```
y_pred = np.zeros(len(result))
for i, score in enumerate(result):
    y_pred[i] = 1 if score > 0.5 else 0
```

In [38]:
```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_pred, y_test)
```

In [39]:
```
mae
```

Out[39]:
```
0.1472
```