Import Packages

In [1]:
```python
import numpy as np
from keras.datasets import imdb
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras import metrics


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

Loading the Data

In [2]:
```python
# 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
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

In [3]:
```python
train_data[:2]
```

```
 656,
 245,
 2350,
 5,
 4,
 9837,
 131,
 152,
 491,
 18,
 2,
 32,
 7464,
 1212,
 14,
 9,
 6,
 371,
 78,
 22,
 625,
 64,
 1382,
 9,
 8,
 168,
 145,
 23,
 4,
 1690,
 15,
 16,
 4,
 1355,
 5,
 28,
 6,
 52,
 154,
 462,
 33,
 89,
 78,
 285,
 16,
 145,
 95]]
```

In [4]:

```
train_labels
```

Out[4]:

```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [5]:

```
# Check the first label
train_labels[0]
```

Out[5]:

```
1
```

In [6]:

```
# Since we restricted ourselves to the top 10000 frequent words, no word index s
# we'll verify this below
# Here is a list of maximum indexes in every review --- we search the maximum in
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[6]:

```
9999
```

In [7]:

```
# 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",
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[

decoded_review
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

Out[7]:

```
"? this film was just brilliant casting location scenery story direction everyo
ne'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 connect
ion 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 w
ould recommend it to everyone to watch and the fly fishing was amazing really c
ried at the end it was so sad and you know what they say if you cry at a film i
t must have been good and this definitely was also ? to the two little boy's th
at played the ? of norman and paul they were just brilliant children are often
left out of the ? list i think because the stars that play them all grown up ar
e such a big profile for the whole film but these children are amazing and shou
ld be praised for what they have done don't you think the whole story was so lo
vely because it was true and was someone's life after all that was shared with
us all"
```

In [8]:

```
len(reverse_word_index)
```

Out[8]:

```
88584
```

Preparing the data

In [9]:

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

# Vectorize training Data
X_train = vectorize_sequences(train_data)
```

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

In [10]:

```
X_train[0]
```

Out[10]:

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

In [11]:

```
X_train.shape
```

Out[11]:

```
(25000, 10000)
```

Vectorize labels

In [12]:

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

Model defination

In [13]:

```
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'))
```

```
C:\Users\admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras
\src\layers\core\dense.py:88: UserWarning: Do not pass an `input_shape`/`input_
dim` argument to a layer. When using Sequential models, prefer using an `Input
(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [14]:

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

In [15]:

```
# 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:]
```

Training our model

In [16]:

```
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 ━━━━━━━━━━━━━━━━━━━━ 28s 476ms/step - binary_accuracy: 0.6663 - loss: 0.6
049 - val_binary_accuracy: 0.8397 - val_loss: 0.4093
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - binary_accuracy: 0.8871 - loss: 0.339
3 - val_binary_accuracy: 0.8856 - val_loss: 0.3089
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - binary_accuracy: 0.9272 - loss: 0.236
5 - val_binary_accuracy: 0.8902 - val_loss: 0.2819
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - binary_accuracy: 0.9392 - loss: 0.192
4 - val_binary_accuracy: 0.8894 - val_loss: 0.2747
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - binary_accuracy: 0.9475 - loss: 0.164
4 - val_binary_accuracy: 0.8882 - val_loss: 0.2796
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - binary_accuracy: 0.9552 - loss: 0.138
8 - val_binary_accuracy: 0.8869 - val_loss: 0.2902
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - binary_accuracy: 0.9645 - loss: 0.118
7 - val_binary_accuracy: 0.8854 - val_loss: 0.2957
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - binary_accuracy: 0.9726 - loss: 0.099
5 - val_binary_accuracy: 0.8814 - val_loss: 0.3300
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - binary_accuracy: 0.9749 - loss: 0.086
1 - val_binary_accuracy: 0.8829 - val_loss: 0.3273
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - binary_accuracy: 0.9798 - loss: 0.075
4 - val_binary_accuracy: 0.8781 - val_loss: 0.3631
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - binary_accuracy: 0.9839 - loss: 0.065
2 - val_binary_accuracy: 0.8799 - val_loss: 0.3592
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - binary_accuracy: 0.9889 - loss: 0.054
3 - val_binary_accuracy: 0.8694 - val_loss: 0.4106
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - binary_accuracy: 0.9888 - loss: 0.048
6 - val_binary_accuracy: 0.8762 - val_loss: 0.3982
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - binary_accuracy: 0.9918 - loss: 0.040
3 - val_binary_accuracy: 0.8743 - val_loss: 0.4255
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - binary_accuracy: 0.9935 - loss: 0.035
9 - val_binary_accuracy: 0.8631 - val_loss: 0.4878
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - binary_accuracy: 0.9953 - loss: 0.030
3 - val_binary_accuracy: 0.8675 - val_loss: 0.4822
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - binary_accuracy: 0.9967 - loss: 0.024
5 - val_binary_accuracy: 0.8700 - val_loss: 0.4908
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - binary_accuracy: 0.9986 - loss: 0.020
2 - val_binary_accuracy: 0.8482 - val_loss: 0.6391
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - binary_accuracy: 0.9960 - loss: 0.023
6 - val_binary_accuracy: 0.8610 - val_loss: 0.5668
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - binary_accuracy: 0.9984 - loss: 0.016
6 - val_binary_accuracy: 0.8698 - val_loss: 0.5586
```

In [17]:
```
history_dict = history.history
history_dict.keys()
```

Out[17]:
```
dict_keys(['binary_accuracy', 'loss', 'val_binary_accuracy', 'val_loss'])
```

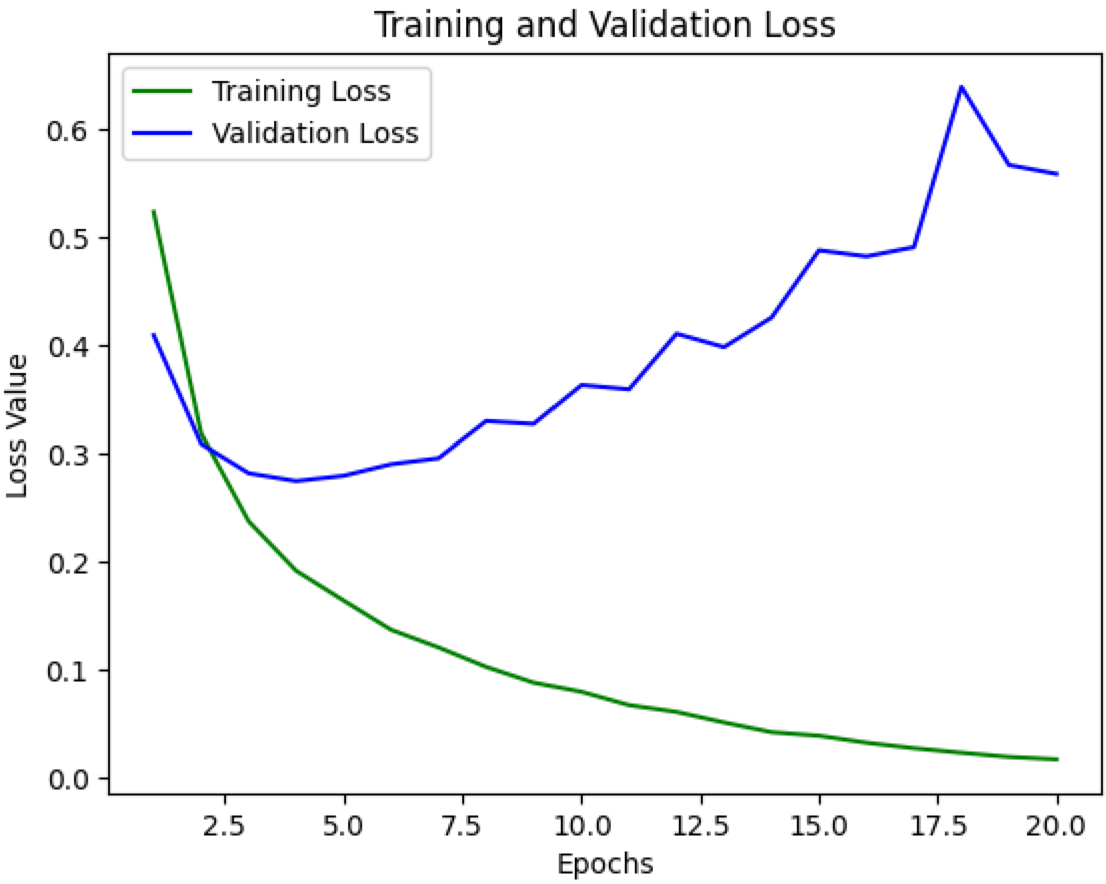
In [18]:
```
# Plotting losses
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'g', 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()
```



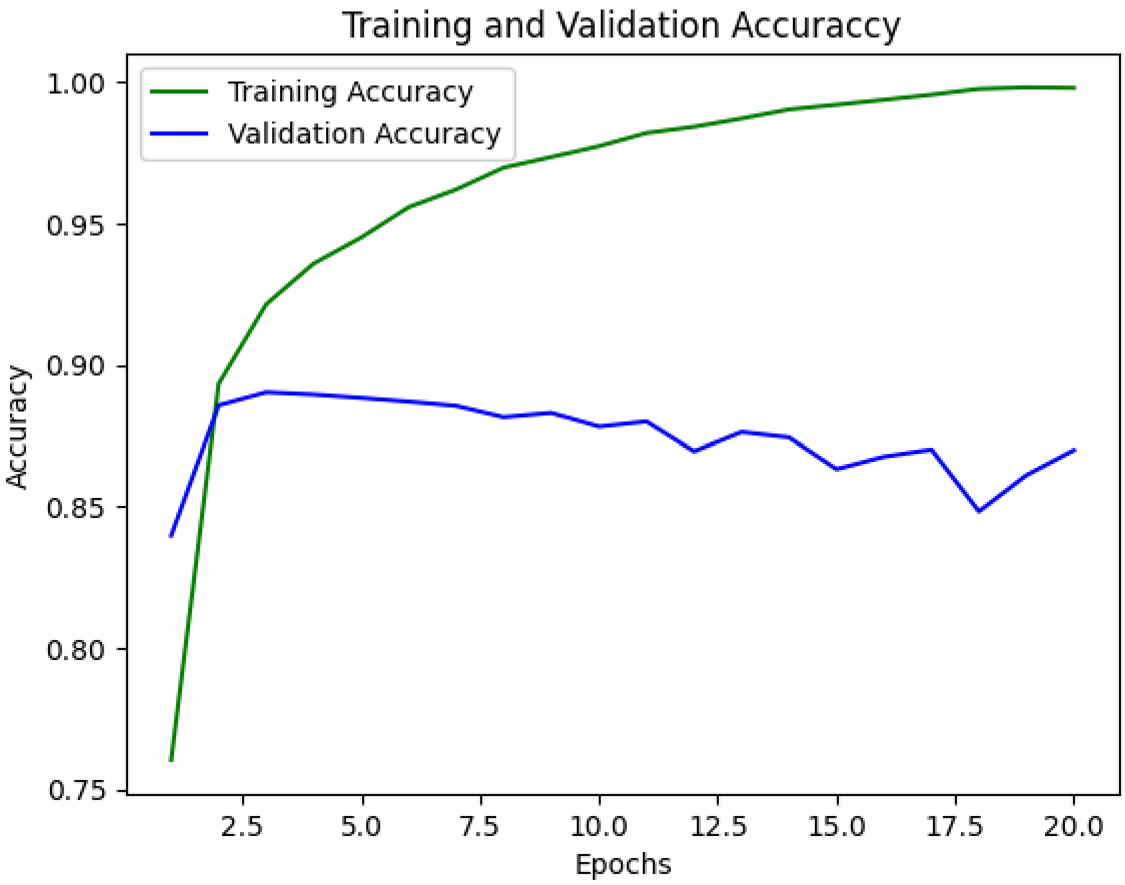
In [19]:
```
# Training and Validation Accuracy

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, 'g', label="Training Accuracy")
plt.plot(epochs, val_acc_values, 'b', label="Validation Accuracy")
plt.title('Training and Validation Accuraccy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Retraining our model

```
In [20]: model.fit(
             partial_X_train,
             partial_y_train,
             epochs=3,
             batch_size=512,
             validation_data=(X_val, y_val)
         )
```

```
Epoch 1/3
30/30 ━━━━━━━━━━━━━━━━━━━━ 14s 417ms/step - binary_accuracy: 0.9977 - loss: 0.0
153 - val_binary_accuracy: 0.8689 - val_loss: 0.5857
Epoch 2/3
30/30 ━━━━━━━━━━━━━━━━━━━━ 11s 379ms/step - binary_accuracy: 0.9994 - loss: 0.0
104 - val_binary_accuracy: 0.8677 - val_loss: 0.6046
Epoch 3/3
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - binary_accuracy: 0.9983 - loss: 0.011
6 - val_binary_accuracy: 0.8684 - val_loss: 0.6298
```

```
Out[20]: <keras.src.callbacks.history.History at 0x229ed331ed0>
```

Model Evaluation

In [21]:

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

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step
```

In [22]:

```
result
```

Out[22]:

```
array([[0.00919514],
       [0.9999993 ],
       [0.9390242 ],
       ...,
       [0.00086875],
       [0.00614974],
       [0.96308315]], dtype=float32)
```

In [23]:

```
y_pred = np.zeros(len(result))
for i, score in enumerate(result):
    y_pred[i] = np.round(score)
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_12356\3135542042.py:3: DeprecationW
arning: Conversion of an array with ndim > 0 to a scalar is deprecated, and wil
l error in future. Ensure you extract a single element from your array before p
erforming this operation. (Deprecated NumPy 1.25.)
  y_pred[i] = np.round(score)
```

In [24]:

```
mae = metrics.mean_absolute_error(y_pred, y_test)
mae
```

Out[24]:

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.14388>
```

In [ ]: