```
from keras.datasets import imdb
(train_data, train_labels),(test_data,test_labels) = imdb.load_data(num_words= 10000)
```

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

```
train_data[0]
```

```
 104,
 88,
 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,
 5535,
 18,
 51,
 36,
 28,
 224,
 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]
```

```
train_labels[0]
```

```
1
```

```
print(type([max(sequence) for sequence in train_data]))
max([max(sequence) for sequence in train_data])
```

```
<class 'list'>
9999
```

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
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/imc
1641221/1641221 [==============================] - 0s 0us/step
'? 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
```

```
len(reverse_word_index)
```

```
88584
```

```
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

xtrain = vectorize_sequences(train_data)
xtest = vectorize_sequences(test_data)
```

```
xtrain[0]
```

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

```
xtrain.shape
```

```
(25000, 10000)
```

```
ytrain = np.asarray(train_labels).astype('float32')
ytest = np.asarray(test_labels).astype('float32')
```

```
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'))
```

```
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])
```

```
xval = xtrain[:10000]
partial_xtrain = xtrain[10000:]

yval = ytrain[:10000]
partial_ytrain = ytrain[10000:]
```

```
history = model.fit(partial_xtrain,
                    partial_ytrain,
                    epochs=20,
                    batch_size= 512,
                    validation_data=(xval, yval))
```

```
Epoch 1/20
30/30 [==============================] - 5s 110ms/step - loss: 0.5294 - binary_accuracy: 0.7698 - val_loss: 0.4042 - val_binary_accu
Epoch 2/20
30/30 [==============================] - 2s 55ms/step - loss: 0.3377 - binary_accuracy: 0.8907 - val_loss: 0.3265 - val_binary_accur
Epoch 3/20
30/30 [==============================] - 2s 66ms/step - loss: 0.2581 - binary_accuracy: 0.9161 - val_loss: 0.2948 - val_binary_accur
Epoch 4/20
30/30 [==============================] - 2s 65ms/step - loss: 0.2073 - binary_accuracy: 0.9338 - val_loss: 0.2784 - val_binary_accur
Epoch 5/20
30/30 [==============================] - 3s 109ms/step - loss: 0.1760 - binary_accuracy: 0.9442 - val_loss: 0.2751 - val_binary_accu
Epoch 6/20
30/30 [==============================] - 2s 57ms/step - loss: 0.1511 - binary_accuracy: 0.9547 - val_loss: 0.2790 - val_binary_accur
Epoch 7/20
30/30 [==============================] - 1s 44ms/step - loss: 0.1301 - binary_accuracy: 0.9603 - val_loss: 0.2865 - val_binary_accur
Epoch 8/20
```

```
    30/30 [==============================] - 1s 33ms/step - loss: 0.1124 - binary_accuracy: 0.9676 - val_loss: 0.3115 - val_binary_accur
    Epoch 9/20
    30/30 [==============================] - 1s 34ms/step - loss: 0.0971 - binary_accuracy: 0.9733 - val_loss: 0.3238 - val_binary_accur
    Epoch 10/20
    30/30 [==============================] - 1s 33ms/step - loss: 0.0883 - binary_accuracy: 0.9754 - val_loss: 0.3233 - val_binary_accur
    Epoch 11/20
    30/30 [==============================] - 1s 32ms/step - loss: 0.0737 - binary_accuracy: 0.9811 - val_loss: 0.3382 - val_binary_accur
    Epoch 12/20
    30/30 [==============================] - 1s 31ms/step - loss: 0.0638 - binary_accuracy: 0.9858 - val_loss: 0.3701 - val_binary_accur
    Epoch 13/20
    30/30 [==============================] - 1s 33ms/step - loss: 0.0573 - binary_accuracy: 0.9868 - val_loss: 0.4696 - val_binary_accur
    Epoch 14/20
    30/30 [==============================] - 1s 46ms/step - loss: 0.0482 - binary_accuracy: 0.9906 - val_loss: 0.3911 - val_binary_accur
    Epoch 15/20
    30/30 [==============================] - 2s 55ms/step - loss: 0.0419 - binary_accuracy: 0.9926 - val_loss: 0.4323 - val_binary_accur
    Epoch 16/20
    30/30 [==============================] - 1s 34ms/step - loss: 0.0378 - binary_accuracy: 0.9935 - val_loss: 0.4308 - val_binary_accur
    Epoch 17/20
    30/30 [==============================] - 1s 32ms/step - loss: 0.0317 - binary_accuracy: 0.9941 - val_loss: 0.4473 - val_binary_accur
    Epoch 18/20
    30/30 [==============================] - 1s 33ms/step - loss: 0.0261 - binary_accuracy: 0.9965 - val_loss: 0.4798 - val_binary_accur
    Epoch 19/20
    30/30 [==============================] - 1s 34ms/step - loss: 0.0213 - binary_accuracy: 0.9980 - val_loss: 0.4905 - val_binary_accur
    Epoch 20/20
    30/30 [==============================] - 1s 31ms/step - loss: 0.0204 - binary_accuracy: 0.9979 - val_loss: 0.5371 - val_binary_accur
```

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

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

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

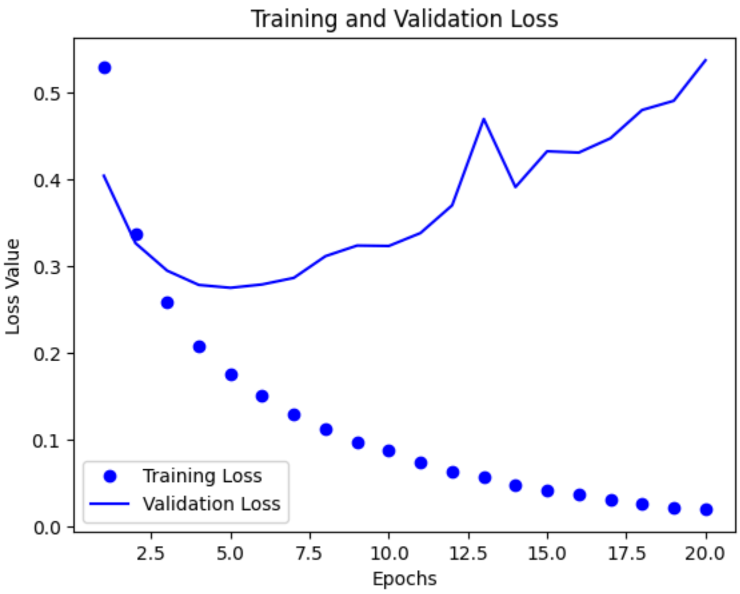
```
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

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()
```



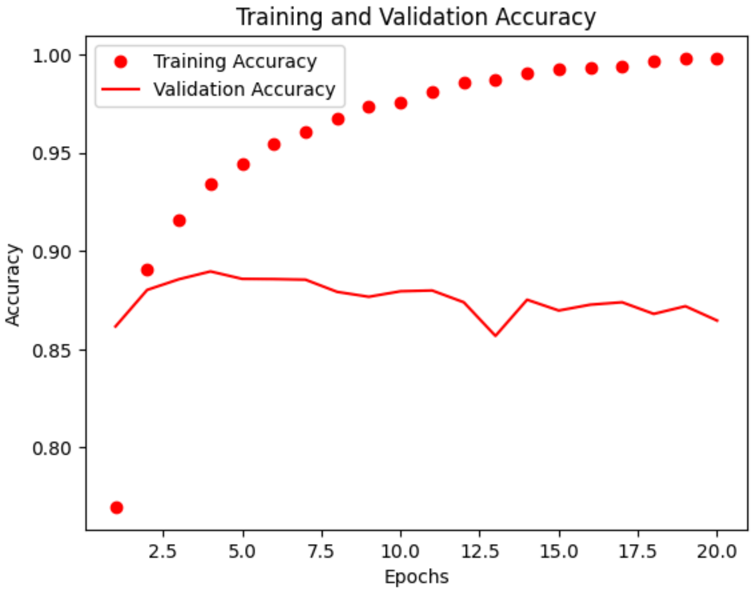
```
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

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 Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



```
model.fit(partial_xtrain,
          partial_ytrain,
          epochs= 3,
          batch_size= 512,
          validation_data=(xval,yval))
```

```
Epoch 1/3
30/30 [==============================] - 2s 61ms/step - loss: 0.0170 - binary_accuracy: 0.9979 - val_loss: 0.5386 - val_binary_accur
Epoch 2/3
30/30 [==============================] - 1s 33ms/step - loss: 0.0110 - binary_accuracy: 0.9995 - val_loss: 0.5605 - val_binary_accur
Epoch 3/3
30/30 [==============================] - 2s 52ms/step - loss: 0.0140 - binary_accuracy: 0.9977 - val_loss: 0.5759 - val_binary_accur
<keras.src.callbacks.History at 0x7c743259ec80>
```

```
np.set_printoptions(suppress= True)
result = model.predict(xtest)
```

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

```
result
```

```
array([[0.01149634],
       [0.99998546],
       [0.8626642 ],
       ...,
       [0.00099883],
       [0.00788227],
       [0.96858144]], dtype=float32)
```

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

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

```
mae
```

```
0.143
```