```
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-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

```
# 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'>
9999
```

```
# 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-datasets/imdb_word_index.json
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 ? is an amazing actor and now the same being director ? father came from the sam
e scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout th
e film were great it was just brilliant so much that i bought the film as soon as it was released for ? and would recommen
d it to everyone 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 definitely was also ? to the two little boy's that played the ? of no
rman 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 are such a big profile for the whole film but these children are amazing and should be praised for what they
```

```
# 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)
```

```
X_train[0]
```

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

```
X_train.shape
```

```
(25000, 10000)
```

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

```
# 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:]
```

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

```
Epoch 1/20
30/30 [==============================] - 2s 57ms/step - loss: 0.5501 - binary_accuracy: 0.7738 - val_loss: 0.4242 - val_binary_accu
Epoch 2/20
30/30 [==============================] - 1s 34ms/step - loss: 0.3422 - binary_accuracy: 0.8918 - val_loss: 0.3225 - val_binary_accu
Epoch 3/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2525 - binary_accuracy: 0.9183 - val_loss: 0.2964 - val_binary_accu
Epoch 4/20
30/30 [==============================] - 1s 32ms/step - loss: 0.2058 - binary_accuracy: 0.9322 - val_loss: 0.2769 - val_binary_accu
Epoch 5/20
30/30 [==============================] - 2s 56ms/step - loss: 0.1724 - binary_accuracy: 0.9407 - val_loss: 0.2762 - val_binary_accu
Epoch 6/20
30/30 [==============================] - 1s 48ms/step - loss: 0.1451 - binary_accuracy: 0.9528 - val_loss: 0.2816 - val_binary_accu
Epoch 7/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1220 - binary_accuracy: 0.9627 - val_loss: 0.2965 - val_binary_accu
Epoch 8/20
30/30 [==============================] - 1s 32ms/step - loss: 0.1050 - binary_accuracy: 0.9693 - val_loss: 0.3069 - val_binary_accu
Epoch 9/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0922 - binary_accuracy: 0.9728 - val_loss: 0.3223 - val_binary_accu
Epoch 10/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0748 - binary_accuracy: 0.9806 - val_loss: 0.3440 - val_binary_accu
Epoch 11/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0663 - binary_accuracy: 0.9827 - val_loss: 0.3582 - val_binary_accu
Epoch 12/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0548 - binary_accuracy: 0.9867 - val_loss: 0.3795 - val_binary_accu
Epoch 13/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0499 - binary_accuracy: 0.9881 - val_loss: 0.3949 - val_binary_accu
Epoch 14/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0366 - binary_accuracy: 0.9931 - val_loss: 0.4534 - val_binary_accu
Epoch 15/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0308 - binary_accuracy: 0.9948 - val_loss: 0.4701 - val_binary_accu
Epoch 16/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0249 - binary_accuracy: 0.9964 - val_loss: 0.4665 - val_binary_accu
Epoch 17/20
30/30 [==============================] - 2s 52ms/step - loss: 0.0220 - binary_accuracy: 0.9979 - val_loss: 0.4935 - val_binary_accu
Epoch 18/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0214 - binary_accuracy: 0.9959 - val_loss: 0.5008 - val_binary_accu
Epoch 19/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0147 - binary_accuracy: 0.9984 - val_loss: 0.5204 - val_binary_accu
Epoch 20/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0123 - binary_accuracy: 0.9983 - val_loss: 0.5402 - val_binary_accu
```

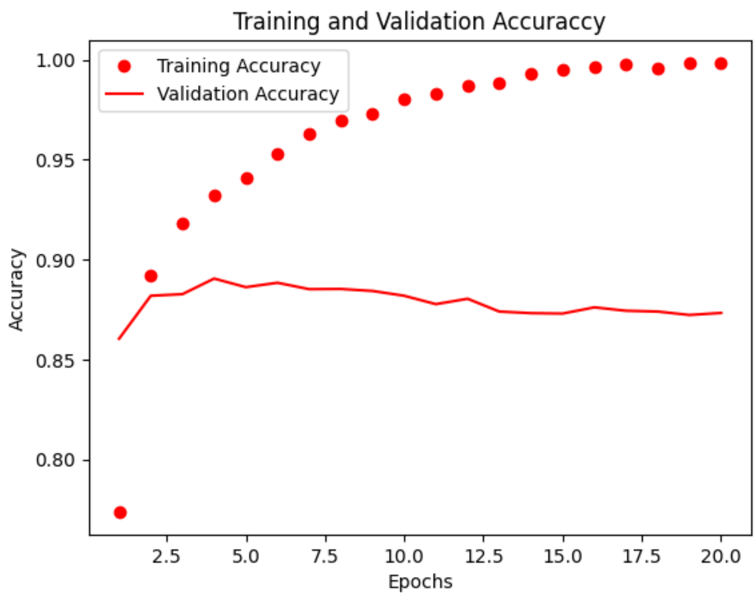
```
history_dict = history.history
history_dict.keys()
```

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

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

```
# 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()
```


```
# 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()
```




```
model.fit(partial_X_train, partial_y_train, epochs=3, batch_size=512, validation_data=(X_val, y_val))
```

```
Epoch 1/3
30/30 [==============================] - 2s 51ms/step - loss: 0.0107 - binary_accuracy: 0.9984 - val_loss: 0.6217 - val_binary_accu
Epoch 2/3
30/30 [==============================] - 2s 59ms/step - loss: 0.0067 - binary_accuracy: 0.9998 - val_loss: 0.5804 - val_binary_accu
Epoch 3/3
30/30 [==============================] - 1s 44ms/step - loss: 0.0097 - binary_accuracy: 0.9987 - val_loss: 0.5961 - val_binary_accu
<keras.callbacks.History at 0x7f608fb17c70>
```

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

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

```
result
```

```
array([[0.00910641],
       [1.        ],
       [0.36453325],
       ...,
       [0.00241602],
       [0.00917078],
       [0.95490444]], 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, y_test)
```

```
# error
mae
```

```
0.14148
```

✓ 0s completed at 10:56 PM