```
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)


print(train_data[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 28
```

+ Code + Text

```
print(train_labels[0])
```

```
1
```

```
# Since we restricted ourselves to the top 10000 frequent words, no word index should exceed 10000
# we'll verify this below

# Here is a list of maximum indexes in every review --- we search the maximum index in this list of max indexes
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
```

```
# Let's quickly decode a review

# step 1: load the dictionary mappings from word to integer index
word_index = imdb.get_word_index()
#print(word_index.items())
# 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 sequence" and "unknown"
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]])

decoded_review
```

```
'? 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 t
he same being director ? father came from the same scottish island as myself so i loved the fact t
here 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 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 ?
```

Vectorize input data

```
#one hot encoding of each word
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))    # Creates an all zero matrix of shape (len(sequences),10K)
    for i,sequence in enumerate(sequences):
        results[i,sequence] = 1                        # Sets specific indices of results[i] to 1s
    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)
```

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

```
print(y_train.shape)
print(y_test.shape)
```

```
(25000,)
(25000,)
```

```
#Model definition
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'))
```

```
#Compiling the model
from keras import optimizers
from keras import losses
from keras import metrics

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

```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:143: UserWarning: The `lr` argument is deprecated, use `
  super().__init__(name, **kwargs)
```

```
#Setting up Validation
# 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
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 [==============================] - 4s 84ms/step - loss: 0.5167 - binary_accuracy: 0.7948 - val_loss: 0.3972 - val_binary_accuracy:
Epoch 2/20
30/30 [==============================] - 1s 45ms/step - loss: 0.3153 - binary_accuracy: 0.9010 - val_loss: 0.3251 - val_binary_accuracy:
Epoch 3/20
30/30 [==============================] - 1s 45ms/step - loss: 0.2318 - binary_accuracy: 0.9277 - val_loss: 0.3152 - val_binary_accuracy:
Epoch 4/20
30/30 [==============================] - 1s 46ms/step - loss: 0.1830 - binary_accuracy: 0.9429 - val_loss: 0.2945 - val_binary_accuracy:
Epoch 5/20
30/30 [==============================] - 2s 56ms/step - loss: 0.1475 - binary_accuracy: 0.9564 - val_loss: 0.2828 - val_binary_accuracy:
Epoch 6/20
30/30 [==============================] - 2s 75ms/step - loss: 0.1233 - binary_accuracy: 0.9632 - val_loss: 0.2920 - val_binary_accuracy:
Epoch 7/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1001 - binary_accuracy: 0.9711 - val_loss: 0.3078 - val_binary_accuracy:
Epoch 8/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0832 - binary_accuracy: 0.9773 - val_loss: 0.3332 - val_binary_accuracy:
Epoch 9/20
30/30 [==============================] - 2s 51ms/step - loss: 0.0677 - binary_accuracy: 0.9826 - val_loss: 0.3503 - val_binary_accuracy:
Epoch 10/20
30/30 [==============================] - 2s 55ms/step - loss: 0.0529 - binary_accuracy: 0.9883 - val_loss: 0.3891 - val_binary_accuracy:
Epoch 11/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0433 - binary_accuracy: 0.9914 - val_loss: 0.3969 - val_binary_accuracy:
Epoch 12/20
```

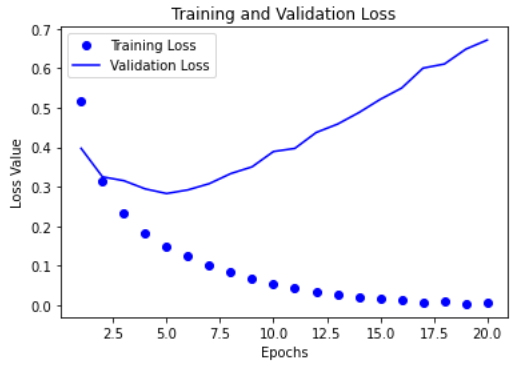
```
30/30 [==============================] - 1s 45ms/step - loss: 0.0331 - binary_accuracy: 0.9937 - val_loss: 0.4373 - val_binary_accuracy:
Epoch 13/20
30/30 [==============================] - 2s 54ms/step - loss: 0.0254 - binary_accuracy: 0.9959 - val_loss: 0.4585 - val_binary_accuracy:
Epoch 14/20
30/30 [==============================] - 3s 92ms/step - loss: 0.0195 - binary_accuracy: 0.9977 - val_loss: 0.4877 - val_binary_accuracy:
Epoch 15/20
30/30 [==============================] - 2s 68ms/step - loss: 0.0153 - binary_accuracy: 0.9984 - val_loss: 0.5212 - val_binary_accuracy:
Epoch 16/20
30/30 [==============================] - 3s 89ms/step - loss: 0.0139 - binary_accuracy: 0.9973 - val_loss: 0.5497 - val_binary_accuracy:
Epoch 17/20
30/30 [==============================] - 2s 72ms/step - loss: 0.0065 - binary_accuracy: 0.9997 - val_loss: 0.6002 - val_binary_accuracy:
Epoch 18/20
30/30 [==============================] - 2s 64ms/step - loss: 0.0094 - binary_accuracy: 0.9982 - val_loss: 0.6106 - val_binary_accuracy:
Epoch 19/20
30/30 [==============================] - 2s 70ms/step - loss: 0.0036 - binary_accuracy: 0.9999 - val_loss: 0.6482 - val_binary_accuracy:
Epoch 20/20
30/30 [==============================] - 2s 78ms/step - loss: 0.0069 - binary_accuracy: 0.9983 - val_loss: 0.6713 - val_binary_accuracy:
```

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

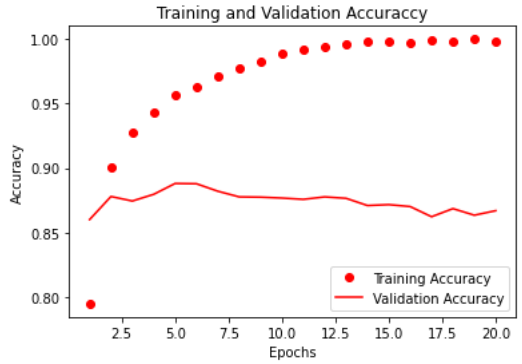
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

```
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()
```




```
#Retraining our model
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 76ms/step - loss: 0.0020 - binary_accuracy: 0.9999 - val_loss: 0.6992 - val_binary_accuracy:
Epoch 2/3
30/30 [==============================] - 1s 43ms/step - loss: 0.0022 - binary_accuracy: 0.9999 - val_loss: 0.7383 - val_binary_accuracy:
Epoch 3/3
30/30 [==============================] - 1s 43ms/step - loss: 0.0012 - binary_accuracy: 0.9999 - val_loss: 0.7695 - val_binary_accuracy:
<keras.callbacks.History at 0x7f1495b16100>
```

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

```
782/782 [==============================] - 3s 3ms/step
```

```
result
```

```
array([[0.0029159 ],
       [1.        ],
       [0.9368664 ],
       ...,
       [0.00138323],
       [0.01167707],
       [0.93595177]], 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)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, auc
accuracy_score(y_test,y_pred)
```

```
0.85144
```

```
# Error
mae
```

```
0.14856
```