Import Packages

```
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

```
# 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
```

```
train_data[:2]
```

```
array([list([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, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147,
2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2,
5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25,
1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530,
476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 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]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954,
189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4,
229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649,
26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349,
165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 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])],
      dtype=object)
```

```
train_labels
```

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

```
# Check the first label
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()

# 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
```

```
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 coul
d 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 connection with this film the witty remarks throughout the film were great it was ju
st 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 fl
y 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 norman and paul they were just brilliant children are o
ften 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 the
```

```
len(reverse_word_index)
```

```
88584
```

Preparing the data

```
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)
```

Vectorize labels

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

Model defination

```
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'))
```

```
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:]
```

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 [==============================] - 3s 83ms/step - loss: 0.5377 - binary_accuracy: 0.7878 - val_loss: 0.4385 - val_binary_accur
```

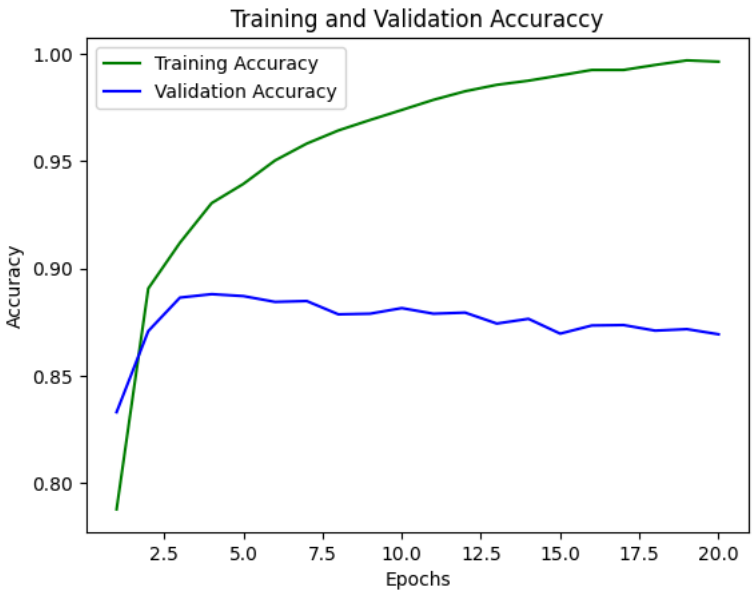
```
Epoch 2/20
30/30 [==============================] - 1s 48ms/step - loss: 0.3448 - binary_accuracy: 0.8907 - val_loss: 0.3388 - val_binary_accur
Epoch 3/20
30/30 [==============================] - 1s 35ms/step - loss: 0.2595 - binary_accuracy: 0.9121 - val_loss: 0.2898 - val_binary_accur
Epoch 4/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2100 - binary_accuracy: 0.9305 - val_loss: 0.2797 - val_binary_accur
Epoch 5/20
30/30 [==============================] - 1s 32ms/step - loss: 0.1789 - binary_accuracy: 0.9395 - val_loss: 0.2749 - val_binary_accur
Epoch 6/20
30/30 [==============================] - 1s 34ms/step - loss: 0.1516 - binary_accuracy: 0.9504 - val_loss: 0.2929 - val_binary_accur
Epoch 7/20
30/30 [==============================] - 1s 31ms/step - loss: 0.1313 - binary_accuracy: 0.9583 - val_loss: 0.2914 - val_binary_accur
Epoch 8/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1146 - binary_accuracy: 0.9644 - val_loss: 0.3153 - val_binary_accur
Epoch 9/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1002 - binary_accuracy: 0.9693 - val_loss: 0.3219 - val_binary_accur
Epoch 10/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0884 - binary_accuracy: 0.9739 - val_loss: 0.3289 - val_binary_accur
Epoch 11/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0752 - binary_accuracy: 0.9787 - val_loss: 0.3497 - val_binary_accur
Epoch 12/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0670 - binary_accuracy: 0.9827 - val_loss: 0.3623 - val_binary_accur
Epoch 13/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0572 - binary_accuracy: 0.9857 - val_loss: 0.3884 - val_binary_accur
Epoch 14/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0507 - binary_accuracy: 0.9876 - val_loss: 0.4038 - val_binary_accur
Epoch 15/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0444 - binary_accuracy: 0.9901 - val_loss: 0.4585 - val_binary_accur
Epoch 16/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0368 - binary_accuracy: 0.9926 - val_loss: 0.4425 - val_binary_accur
Epoch 17/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0329 - binary_accuracy: 0.9926 - val_loss: 0.4703 - val_binary_accur
Epoch 18/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0280 - binary_accuracy: 0.9949 - val_loss: 0.4963 - val_binary_accur
Epoch 19/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0227 - binary_accuracy: 0.9971 - val_loss: 0.5106 - val_binary_accur
Epoch 20/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0211 - binary_accuracy: 0.9965 - val_loss: 0.5352 - val_binary_accur
```

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

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

```
# 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()
```


```
# 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

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

```
Epoch 1/3
30/30 [==============================] - 3s 101ms/step - loss: 0.0209 - binary_accuracy: 0.9955 - val_loss: 0.5573 - val_binary_accu
Epoch 2/3
```

```
30/30 [==============================] - 2s 53ms/step - loss: 0.0117 - binary_accuracy: 0.9997 - val_loss: 0.5793 - val_binary_accur
Epoch 3/3
30/30 [==============================] - 2s 54ms/step - loss: 0.0137 - binary_accuracy: 0.9988 - val_loss: 0.6043 - val_binary_accur
<keras.src.callbacks.History at 0x78ed93e0cac0>
```

Model Evaluation

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

```
782/782 [==============================] - 2s 3ms/step
```

```
result
```

```
array([[0.02593851],
       [0.99999934],
       [0.9510471 ],
       ...,
       [0.00184262],
       [0.01539504],
       [0.9747988 ]], dtype=float32)
```

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

```
<ipython-input-24-d06888fff3d2>:3: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will erro
  y_pred[i] = np.round(score)
```

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