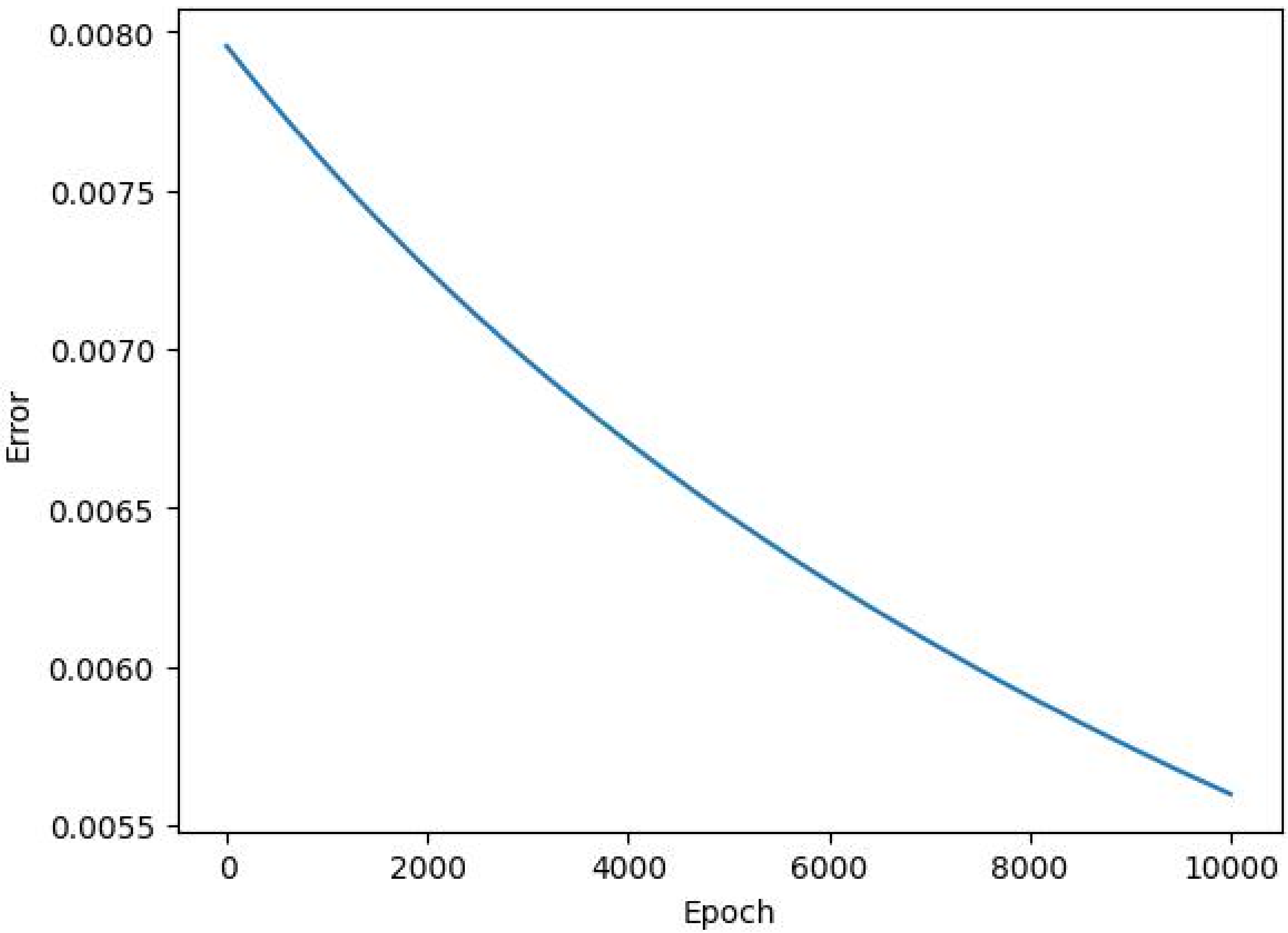
The aim of this practical is to implement a simple feed-forward neural network using backpropagation in Python.

CODE

```
import numpy as np

# sigmoid function and its derivative
def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# input dataset
inputs = np.array([[0,0,1],
                   [1,1,1],
                   [1,0,1],
                   [0,1,1]])

# output dataset
outputs = np.array([[0,1,1,0]]).T

# seed random numbers to make calculation deterministic
np.random.seed(1)

# initialize weights randomly with mean 0
weights = 2 * np.random.random((3,1)) - 1

for iter in range(10000):
    # forward propagation
    input_layer = inputs
    predictions = sigmoid(np.dot(input_layer, weights))

    # how much did we miss?
    error = outputs - predictions

    # multiply how much we missed by the slope of the sigmoid at the values in
predictions
    adjustments = error * sigmoid_derivative(predictions)

    # update weights
    weights += np.dot(input_layer.T, adjustments)

print("Output After Training:")
print(predictions)
```

```
## OUTPUT ##

Output After Training:
[[0.00966449]
 [0.99211957]
 [0.99358898]
 [0.00786506]]
```

```
## GRAPH CODE ##

import matplotlib.pyplot as plt

# assuming 'errors' is a list containing the error of the network after each epoch
errors = []

for iter in range(10000):
    # forward propagation
    input_layer = inputs
    predictions = sigmoid(np.dot(input_layer, weights))

    # how much did we miss?
    error = outputs - predictions
    errors.append(np.mean(np.abs(error)))

    # multiply how much we missed by the slope of the sigmoid at the values in
predictions
    adjustments = error * sigmoid_derivative(predictions)

    # update weights
    weights += np.dot(input_layer.T, adjustments)

# plot the error over time
plt.plot(errors)
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.show()
```


0.0080
0.0075
0.0070
0.0065
0.0060
0.0055
Error
0
2000
4000
6000
8000
10000
Epoch