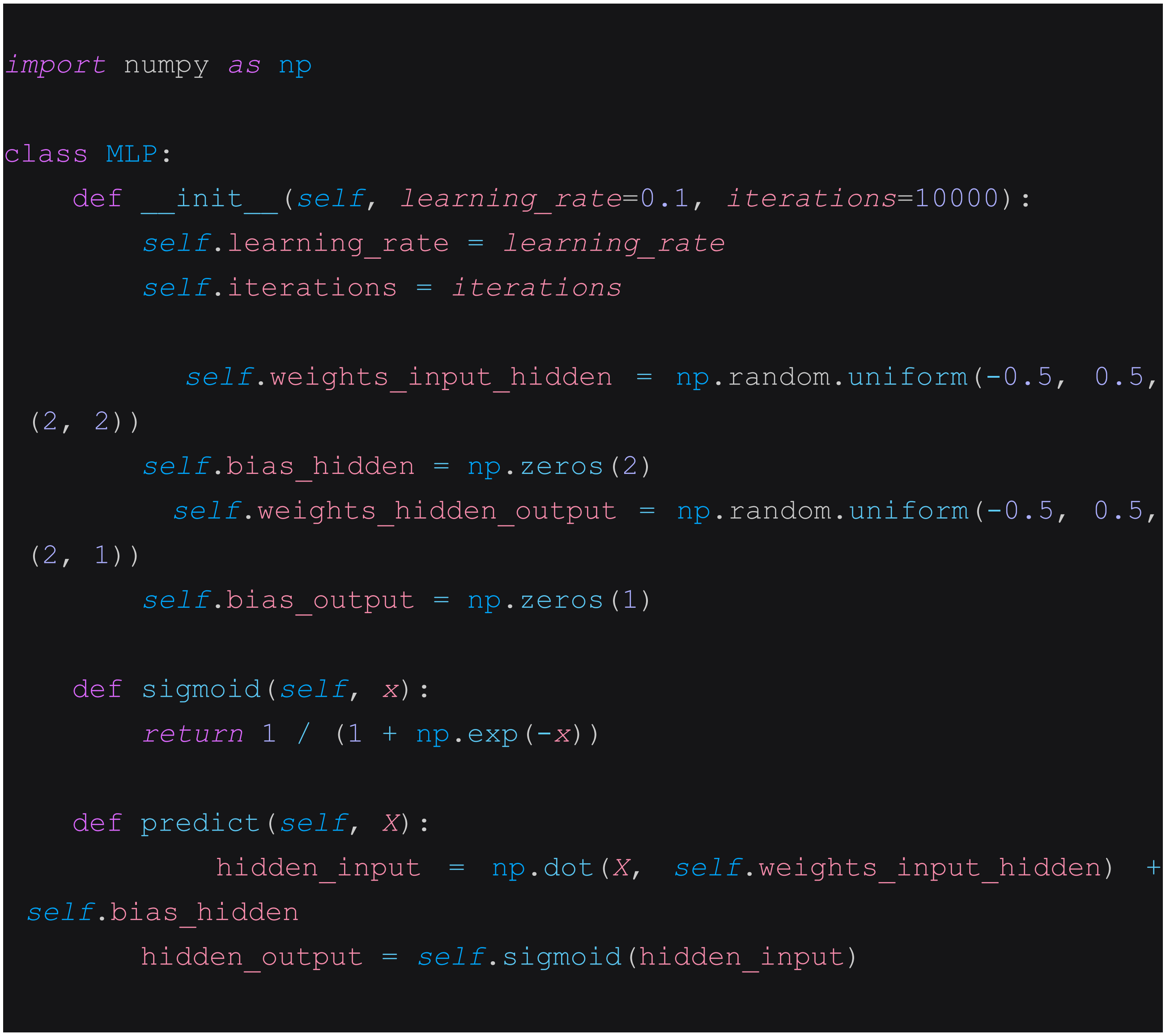
SOFT COMPUTING ASSIGNMENT -4

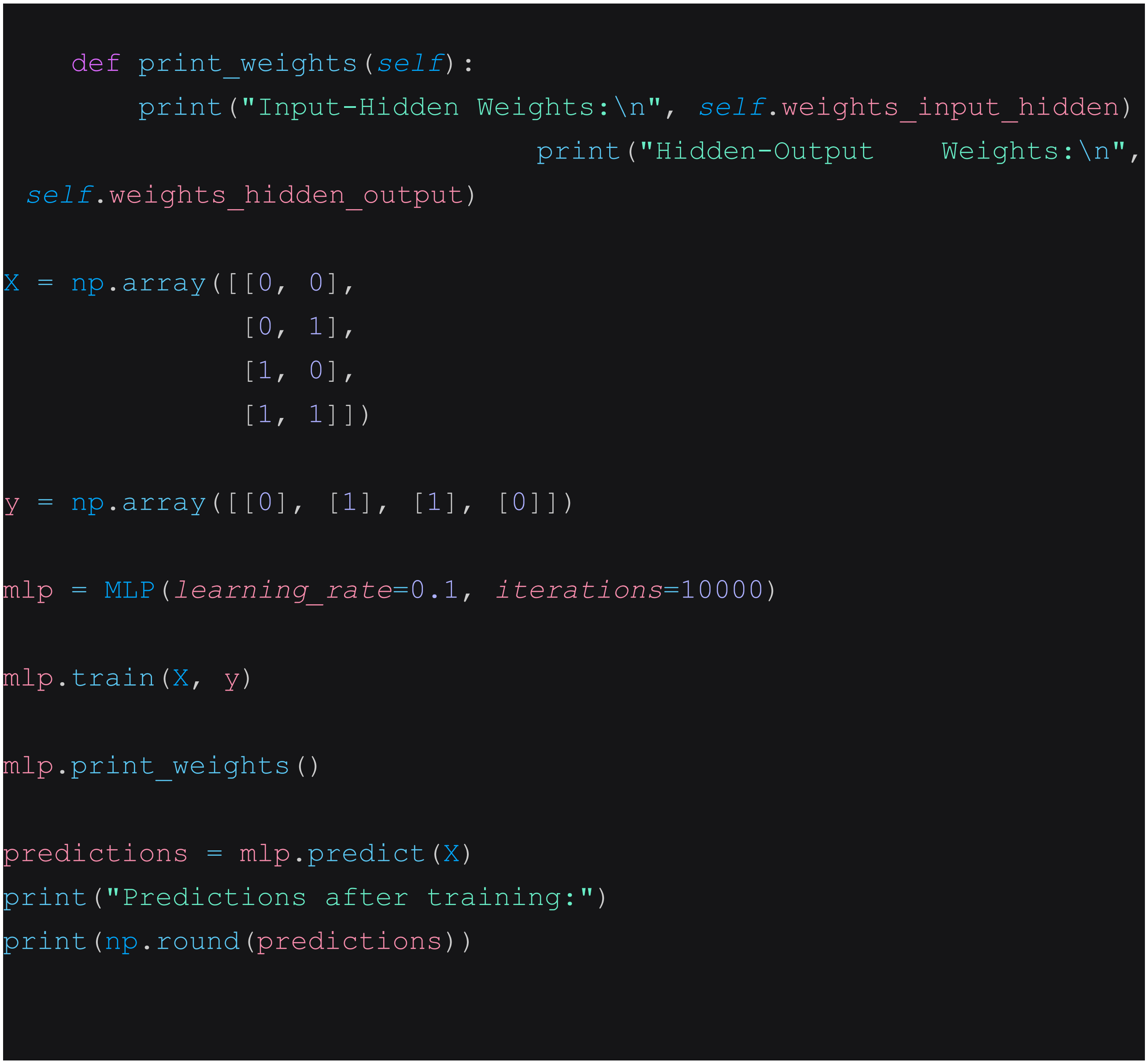
Perumalla Dharan
AP21110010201

Write a Python program to implement a Multi-Layer Perceptron (MLP) on logical XOR function. Take the binary inputs and outputs.

Note: Upload the source file of the Python program and a Word document file that contains the Python program along with the results.

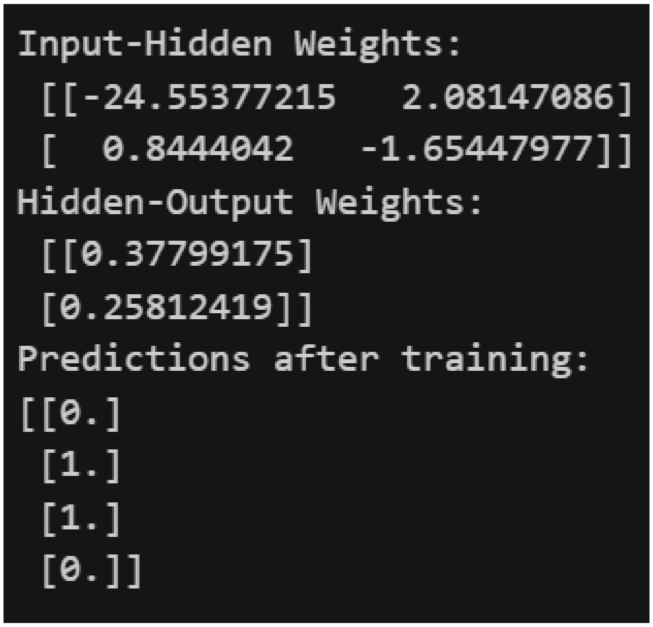


```
import numpy as np

class MLP:
    def __init__(self, learning_rate=0.1, iterations=10000):
        self.learning_rate = learning_rate
        self.iterations = iterations

          self.weights_input_hidden = np.random.uniform(-0.5, 0.5,
 (2, 2))
        self.bias_hidden = np.zeros(2)
         self.weights_hidden_output = np.random.uniform(-0.5, 0.5,
 (2, 1))
        self.bias_output = np.zeros(1)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def predict(self, X):
            hidden_input = np.dot(X, self.weights_input_hidden) +
 self.bias_hidden
        hidden_output = self.sigmoid(hidden_input)
        final_input = np.dot(hidden_output, self.weights_hidden_output) + self.bias_output
        final_output = self.sigmoid(final_input)
        return np.round(final_output)

    def train(self, X, y):
        for epoch in range(self.iterations):
            for i in range(len(X)):

                hidden_input = np.dot(X[i], self.weights_input_hidden) + self.bias_hidden
                hidden_output = self.sigmoid(hidden_input)

                final_input = np.dot(hidden_output, self.weights_hidden_output) + self.bias_output
                final_output = self.sigmoid(final_input)

                y_pred = np.round(final_output)

                if y_pred != y[i]:

                    output_error = y[i] - final_output
                    hidden_error = output_error.dot(self.weights_hidden_output.T)

                    self.weights_hidden_output += self.learning_rate * np.outer(hidden_output, output_error)
                    self.bias_output += self.learning_rate * output_error

                    self.weights_input_hidden += self.learning_rate * np.outer(X[i], hidden_error)
                    self.bias_hidden += self.learning_rate * hidden_error
```



```
    def print_weights(self):
        print("Input-Hidden Weights:\n", self.weights_input_hidden)
                                print("Hidden-Output    Weights:\n",
 self.weights_hidden_output)

X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

y = np.array([[0], [1], [1], [0]])

mlp = MLP(learning_rate=0.1, iterations=10000)

mlp.train(X, y)

mlp.print_weights()

predictions = mlp.predict(X)
print("Predictions after training:")
print(np.round(predictions))
```

Output -



```
Input-Hidden Weights:
 [[-24.55377215   2.08147086]
 [  0.8444042   -1.65447977]]
Hidden-Output Weights:
 [[0.37799175]
 [0.25812419]]
Predictions after training:
[[0.]
 [1.]
 [1.]
 [0.]]
```