LAPORAN PRAKTIKUM

MACHINE LEARNING MINGGU KE-12

Jaringan Syaraf Tiruan

Dosen Pengampu: Nur Rosyid Mubtadai S.Kom., M.T.

Oleh:

Bayu Kurniawan
(3322600019)

PROGRAM STUDI D4 SAINS DATA TERAPAN

DEPARTEMEN TEKNIK INFORMATIKA DAN KOMPUTER

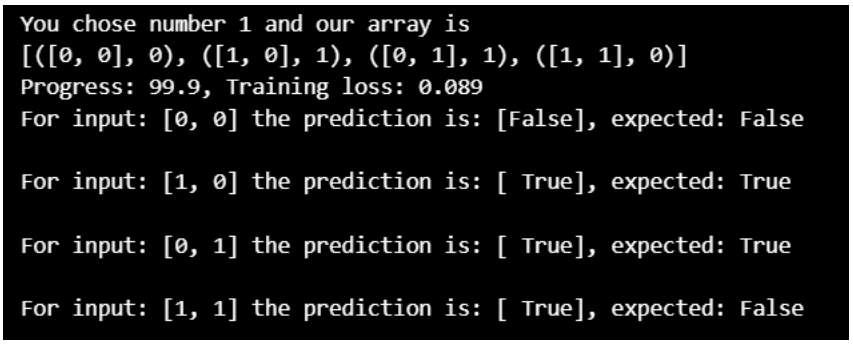
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2023

Menggunkan Library :

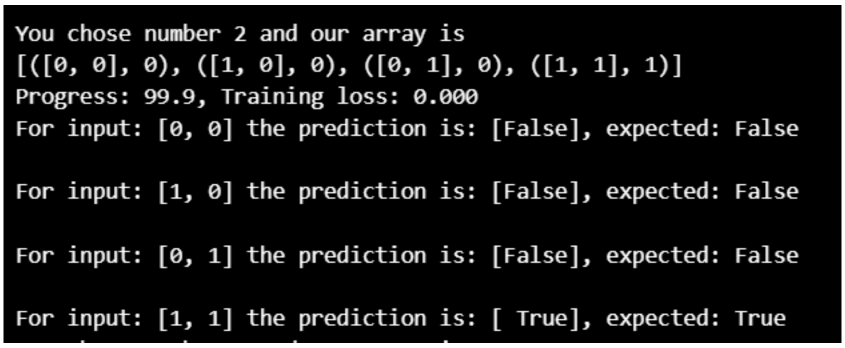
```
import numpy as np
import sys

class Logical(object):
    def __init__(self, learning_rate=0.1):
        self.weights_3_2 = np.random.normal(0.0, 2**-0.5, (3, 2))
        self.weights_3_1 = np.random.normal(0.0, 1, (1, 3))
        self.sigmoid_mapper = np.vectorize(self.sigmoid)
        self.learning_rate = np.array([learning_rate])

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

    def predict(self, inputs):
        inputs_1 = np.dot(self.weights_3_2, inputs)
        outputs_1 = self.sigmoid_mapper(inputs_1)
        inputs_2 = np.dot(self.weights_3_1, outputs_1)
        outputs_2 = self.sigmoid_mapper(inputs_2)
        return outputs_2

    def train(self, inputs, expected_predict):
        inputs_1 = np.dot(self.weights_3_2, inputs)
        outputs_1 = self.sigmoid_mapper(inputs_1)
        inputs_2 = np.dot(self.weights_3_1, outputs_1)
        outputs_2 = self.sigmoid_mapper(inputs_2)
        actual_predict = outputs_2[0]

        error_layer_1 = np.array([actual_predict - expected_predict])
        gradient_layer_1 = actual_predict * (1 - actual_predict)
        weights_delta_layer_1 = error_layer_1 * gradient_layer_1
        self.weights_3_1 -= np.dot(weights_delta_layer_1, outputs_1.reshape(1,
len(outputs_1))) * self.learning_rate

        error_layer_2 = weights_delta_layer_1 * self.weights_3_1
        gradient_layer_2 = outputs_1 * (1 - outputs_1)
        weights_delta_layer_2 = error_layer_2 * gradient_layer_2
        self.weights_3_2 -= np.dot(inputs.reshape(len(inputs), 1),
weights_delta_layer_2).T * self.learning_rate

def MSE(y, Y):
    return np.mean((y - Y) ** 2)

def choose_logical():
    train = []
    num = int(input("Choose \n"
                    " \t1. XOR \n"
                    " \t2. AND \n"
                    " \t3. OR \n"
                    "Masukkan pilihan anda : "))
    if num == 1:
        train = [([0, 0], 0),
                 ([1, 0], 1),
                 ([0, 1], 1),
                 ([1, 1], 0)]
    elif num == 2:
        train = [([0, 0], 0),
                 ([1, 0], 0),
                 ([0, 1], 0),
                 ([1, 1], 1)]
    elif num == 3:
        train = [([0, 0], 0),
                 ([1, 0], 1),
                 ([0, 1], 1),
                 ([1, 1], 1)]
    print("You chose number {} and our array is \n{}".format(num, train))
    return train

if __name__ == "__main__":
    while True:
        res = choose_logical()
        epochs = 6000
        learning_rate = 0.8
        network = Logical(learning_rate=learning_rate)
        for e in range(epochs):
            inputs_ = []
            correct_predictions = []
            for input_stat, correct_predict in res:
                network.train(np.array(input_stat), correct_predict)
                inputs_.append(np.array(input_stat))
                correct_predictions.append(np.array(correct_predict))
            train_loss = MSE(network.predict(np.array(inputs_).T),
np.array(correct_predictions))
            sys.stdout.write("\rProgress: {}, Training loss: {}
".format(str(100 * e / float(epochs))[:4],
                                                                                     str(t
rain_loss)[:5]))

        for input_stat, correct_predict in res:
            print("\nFor input: {} the prediction is: {}, expected:
{}".format(
                str(input_stat),
                str(network.predict(np.array(input_stat)) > 0.5),
                str(correct_predict == 1)))

        user_input = input("\nDo you want to continue? (Y/N): ")
        if user_input.lower() != 'y':
            break
```

Ketika memasukkan 1:



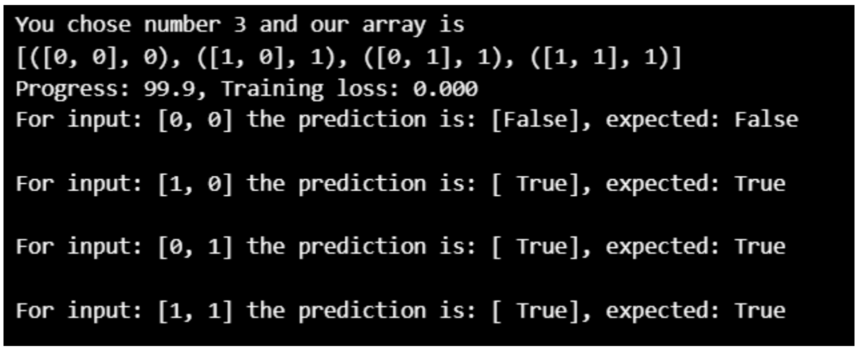
```
You chose number 1 and our array is
[([0, 0], 0), ([1, 0], 1), ([0, 1], 1), ([1, 1], 0)]
Progress: 99.9, Training loss: 0.089
For input: [0, 0] the prediction is: [False], expected: False

For input: [1, 0] the prediction is: [ True], expected: True

For input: [0, 1] the prediction is: [ True], expected: True

For input: [1, 1] the prediction is: [ True], expected: False
```

Ketika memasukkan 2:



```
You chose number 2 and our array is
[([0, 0], 0), ([1, 0], 0), ([0, 1], 0), ([1, 1], 1)]
Progress: 99.9, Training loss: 0.000
For input: [0, 0] the prediction is: [False], expected: False

For input: [1, 0] the prediction is: [False], expected: False

For input: [0, 1] the prediction is: [False], expected: False

For input: [1, 1] the prediction is: [ True], expected: True
```

Ketika memasukkan 3:



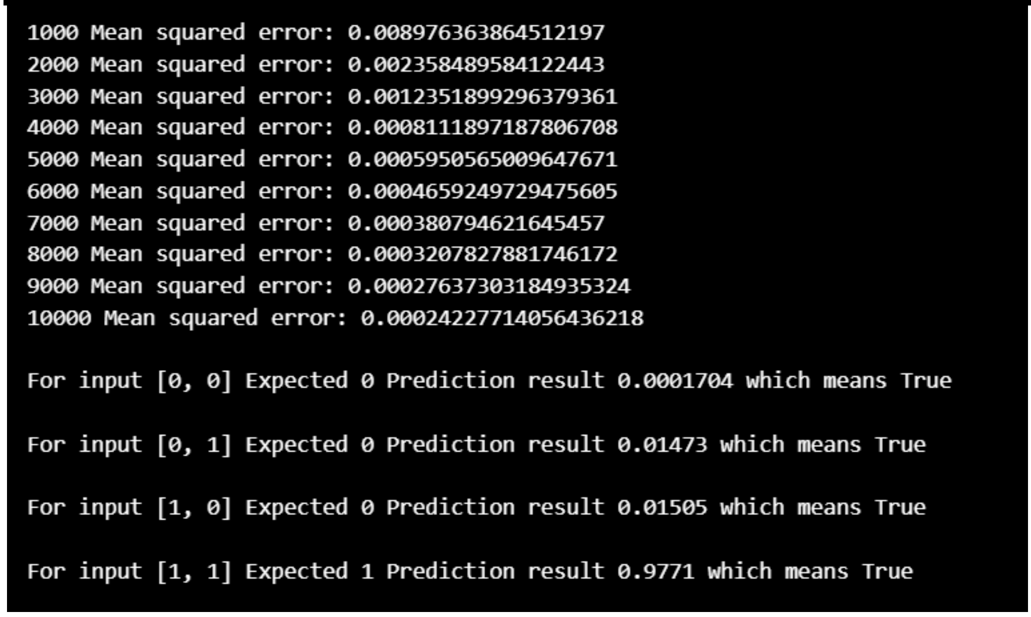
```
You chose number 3 and our array is
[([0, 0], 0), ([1, 0], 1), ([0, 1], 1), ([1, 1], 1)]
Progress: 99.9, Training loss: 0.000
For input: [0, 0] the prediction is: [False], expected: False

For input: [1, 0] the prediction is: [ True], expected: True

For input: [0, 1] the prediction is: [ True], expected: True

For input: [1, 1] the prediction is: [ True], expected: True
```

Tanpa menggunakan Library:

AND

```
import random
import math

VARIANCE_WEIGHT = 0.5

INPUTS = 2
HIDDEN_NODES = 3
OUTPUTS = 1
```

```python
hidden_weights = []
for _ in range(HIDDEN_NODES):
    hidden_weights.append([random.uniform(-VARIANCE_WEIGHT, VARIANCE_WEIGHT)
for _ in range(INPUTS)])

hidden_bias = [0] * HIDDEN_NODES

output_weights = []
for _ in range(OUTPUTS):
    output_weights.append([random.uniform(-VARIANCE_WEIGHT, VARIANCE_WEIGHT)
for _ in range(HIDDEN_NODES)])

output_bias = [0] * OUTPUTS


def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))


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


def predict(inputs):
    hiddens = []
    for i in range(HIDDEN_NODES):
        hidden = 0
        for j in range(INPUTS):
            hidden += hidden_weights[i][j] * inputs[j]
        hidden = sigmoid(hidden + hidden_bias[i])
        hiddens.append(hidden)

    outputs = []
    for i in range(OUTPUTS):
        output = 0
        for j in range(HIDDEN_NODES):
            output += output_weights[i][j] * hiddens[j]
        output = sigmoid(output + output_bias[i])
        outputs.append(output)

    return output

def learn(inputs, targets, alpha=0.1):
    global hidden_weights, hidden_bias
    global output_weights, output_bias

    hiddens = []
    for i in range(HIDDEN_NODES):
        hidden = 0
        for j in range(INPUTS):
            hidden += hidden_weights[i][j] * inputs[j]
        hidden = sigmoid(hidden + hidden_bias[i])
        hiddens.append(hidden)

    outputs = []
    for i in range(OUTPUTS):
        output = 0
        for j in range(HIDDEN_NODES):
            output += output_weights[i][j] * hiddens[j]
        output = sigmoid(output + output_bias[i])
        outputs.append(output)

    errors = []
    for i in range(OUTPUTS):
        error = targets[i] - outputs[i]
        errors.append(error)

    derrors = []
    for i in range(OUTPUTS):
        derror = errors[i] * sigmoid_prime(outputs[i])
        derrors.append(derror)

    ds = [0] * HIDDEN
    for i in range(OUTPUTS):
        for j in range(HIDDEN_NODES):
            ds[j] += derrors[i] * output_weights[i][j] *
sigmoid_prime(hiddens[j])

    for i in range(OUTPUTS):
        for j in range(HIDDEN_NODES):
            output_weights[i][j] += alpha * hiddens[j] * derrors[i]
        output_bias[i] += alpha * derrors[i]

    for i in range(HIDDEN):
        for j in range(INPUTS):
            hidden_weights[i][j] += alpha * inputs[j] * ds[i]
        hidden_bias[i] += alpha * ds[i]


# Data
inputs = [
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
]

outputs = [
    [0],
    [0],
    [0],
    [1]
]

for i in range(10000):
    indexes = [0, 1, 2, 3]
    random.shuffle(indexes)
    for j in indexes:
        learn(inputs[j], outputs[j], alpha=0.2)

    if (i + 1) % 1000 == 0:
        cost = 0
        for j in range(4):
            o = predict(inputs[j])
            cost += (outputs[j][0] - o) ** 2
        cost /= 4
        print(i + 1, "Mean squared error:", cost)

for i in range(4):
    result = predict(inputs[i])
    print("\nFor input", inputs[i], "Expected", outputs[i][0], "Prediction
result", f"{result:4.4}",
          "which means", "True" if round(result) == outputs[i][0] else
"False")
```

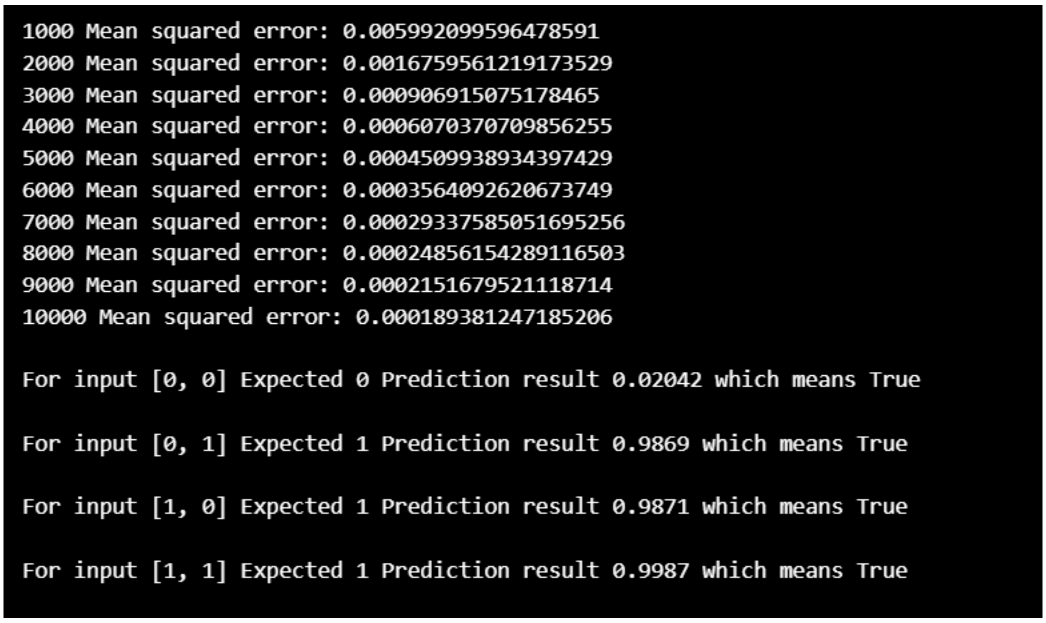


```
1000 Mean squared error: 0.008976363864512197
2000 Mean squared error: 0.002358489584122443
3000 Mean squared error: 0.0012351899296379361
4000 Mean squared error: 0.0008111897187806708
5000 Mean squared error: 0.0005950565009647671
6000 Mean squared error: 0.0004659249729475605
7000 Mean squared error: 0.000380794621645457
8000 Mean squared error: 0.0003207827881746172
9000 Mean squared error: 0.00027637303184935324
10000 Mean squared error: 0.00024227714056436218

For input [0, 0] Expected 0 Prediction result 0.0001704 which means True

For input [0, 1] Expected 0 Prediction result 0.01473 which means True

For input [1, 0] Expected 0 Prediction result 0.01505 which means True

For input [1, 1] Expected 1 Prediction result 0.9771 which means True
```

OR

```
import random
import math
```

```python
VARIANCE_WEIGHT = 0.5

INPUTS = 2
HIDDEN_NODES = 3
OUTPUTS = 1

hidden_weights = []
for _ in range(HIDDEN_NODES):
    hidden_weights.append([random.uniform(-VARIANCE_WEIGHT, VARIANCE_WEIGHT)
for _ in range(INPUTS)])

hidden_bias = [0] * HIDDEN_NODES

output_weights = []
for _ in range(OUTPUTS):
    output_weights.append([random.uniform(-VARIANCE_WEIGHT, VARIANCE_WEIGHT)
for _ in range(HIDDEN_NODES)])

output_bias = [0] * OUTPUTS


def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))


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


def predict(inputs):
    hiddens = []
    for i in range(HIDDEN_NODES):
        hidden = 0
        for j in range(INPUTS):
            hidden += hidden_weights[i][j] * inputs[j]
        hidden = sigmoid(hidden + hidden_bias[i])
        hiddens.append(hidden)

    outputs = []
    for i in range(OUTPUTS):
        output = 0
        for j in range(HIDDEN_NODES):
            output += output_weights[i][j] * hiddens[j]
        output = sigmoid(output + output_bias[i])
        outputs.append(output)

    return output

def learn(inputs, targets, alpha=0.1):
    global hidden_weights, hidden_bias
    global output_weights, output_bias

    hiddens = []
    for i in range(HIDDEN_NODES):
        hidden = 0
        for j in range(INPUTS):
            hidden += hidden_weights[i][j] * inputs[j]
        hidden = sigmoid(hidden + hidden_bias[i])
        hiddens.append(hidden)

    outputs = []
    for i in range(OUTPUTS):
        output = 0
        for j in range(HIDDEN_NODES):
            output += output_weights[i][j] * hiddens[j]
        output = sigmoid(output + output_bias[i])
        outputs.append(output)

    errors = []
    for i in range(OUTPUTS):
        error = targets[i] - outputs[i]
        errors.append(error)

    derrors = []
    for i in range(OUTPUTS):
        derror = errors[i] * sigmoid_prime(outputs[i])
        derrors.append(derror)

    ds = [0] * HIDDEN
    for i in range(OUTPUTS):
        for j in range(HIDDEN_NODES):
            ds[j] += derrors[i] * output_weights[i][j] *
sigmoid_prime(hiddens[j])

    for i in range(OUTPUTS):
        for j in range(HIDDEN_NODES):
            output_weights[i][j] += alpha * hiddens[j] * derrors[i]
        output_bias[i] += alpha * derrors[i]

    for i in range(HIDDEN):
        for j in range(INPUTS):
            hidden_weights[i][j] += alpha * inputs[j] * ds[i]
        hidden_bias[i] += alpha * ds[i]


# Data
inputs = [
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
]

outputs = [
    [0],
    [1],
    [1],
    [1]
]

for i in range(10000):
    indexes = [0, 1, 2, 3]
    random.shuffle(indexes)
    for j in indexes:
        learn(inputs[j], outputs[j], alpha=0.2)

    if (i + 1) % 1000 == 0:
        cost = 0
        for j in range(4):
            o = predict(inputs[j])
            cost += (outputs[j][0] - o) ** 2
        cost /= 4
        print(i + 1, "Mean squared error:", cost)

for i in range(4):
    result = predict(inputs[i])
    print("\nFor input", inputs[i], "Expected", outputs[i][0], "Prediction
result", f"{result:4.4}",
          "which means", "True" if round(result) == outputs[i][0] else
"False")
```

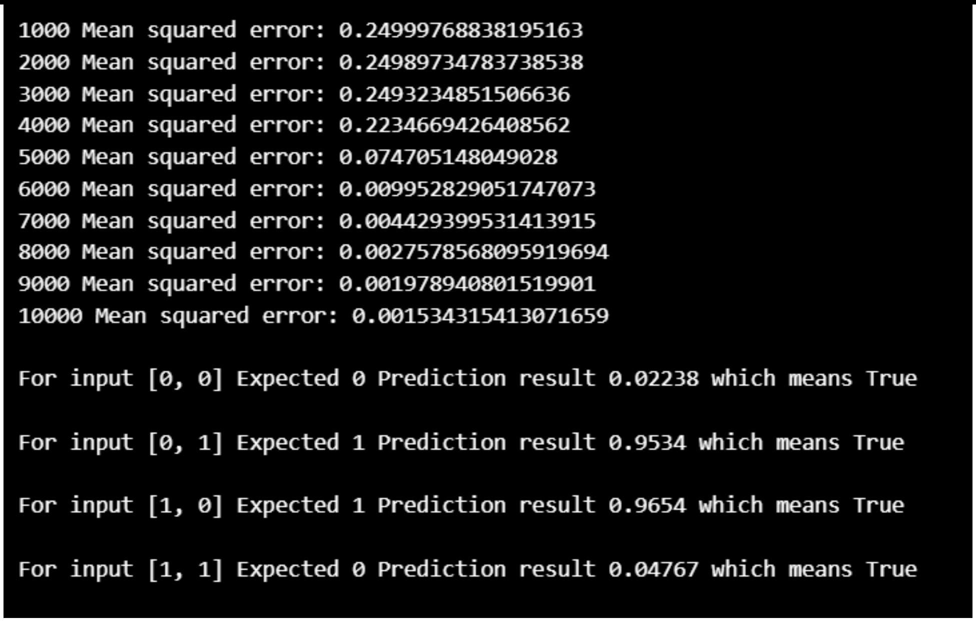


```
1000 Mean squared error: 0.005992099596478591
2000 Mean squared error: 0.0016759561219173529
3000 Mean squared error: 0.000906915075178465
4000 Mean squared error: 0.0006070370709856255
5000 Mean squared error: 0.0004509938934397429
6000 Mean squared error: 0.0003564092620673749
7000 Mean squared error: 0.00029337585051695256
8000 Mean squared error: 0.00024856154289116503
9000 Mean squared error: 0.0002151679521118714
10000 Mean squared error: 0.000189381247185206

For input [0, 0] Expected 0 Prediction result 0.02042 which means True

For input [0, 1] Expected 1 Prediction result 0.9869 which means True

For input [1, 0] Expected 1 Prediction result 0.9871 which means True

For input [1, 1] Expected 1 Prediction result 0.9987 which means True
```

XOR

```
import random
import math

VARIANCE_WEIGHT = 0.5

INPUTS = 2
HIDDEN_NODES = 3
OUTPUTS = 1

hidden_weights = []
for _ in range(HIDDEN_NODES):
    hidden_weights.append([random.uniform(-VARIANCE_WEIGHT, VARIANCE_WEIGHT)
for _ in range(INPUTS)])

hidden_bias = [0] * HIDDEN_NODES

output_weights = []
for _ in range(OUTPUTS):
    output_weights.append([random.uniform(-VARIANCE_WEIGHT, VARIANCE_WEIGHT)
for _ in range(HIDDEN_NODES)])

output_bias = [0] * OUTPUTS


def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))


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


def predict(inputs):
    hiddens = []
    for i in range(HIDDEN_NODES):
        hidden = 0
        for j in range(INPUTS):
            hidden += hidden_weights[i][j] * inputs[j]
        hidden = sigmoid(hidden + hidden_bias[i])
        hiddens.append(hidden)

    outputs = []
    for i in range(OUTPUTS):
        output = 0
        for j in range(HIDDEN_NODES):
            output += output_weights[i][j] * hiddens[j]
        output = sigmoid(output + output_bias[i])
        outputs.append(output)

    return output

def learn(inputs, targets, alpha=0.1):
    global hidden_weights, hidden_bias
    global output_weights, output_bias

    hiddens = []
    for i in range(HIDDEN_NODES):
        hidden = 0
        for j in range(INPUTS):
            hidden += hidden_weights[i][j] * inputs[j]
        hidden = sigmoid(hidden + hidden_bias[i])
        hiddens.append(hidden)

    outputs = []
    for i in range(OUTPUTS):
        output = 0
        for j in range(HIDDEN_NODES):
            output += output_weights[i][j] * hiddens[j]
        output = sigmoid(output + output_bias[i])
        outputs.append(output)

    errors = []
    for i in range(OUTPUTS):
        error = targets[i] - outputs[i]
        errors.append(error)

    derrors = []
    for i in range(OUTPUTS):
        derror = errors[i] * sigmoid_prime(outputs[i])
        derrors.append(derror)

    ds = [0] * HIDDEN
    for i in range(OUTPUTS):
        for j in range(HIDDEN_NODES):
            ds[j] += derrors[i] * output_weights[i][j] *
sigmoid_prime(hiddens[j])

    for i in range(OUTPUTS):
        for j in range(HIDDEN_NODES):
            output_weights[i][j] += alpha * hiddens[j] * derrors[i]
        output_bias[i] += alpha * derrors[i]

    for i in range(HIDDEN):
        for j in range(INPUTS):
            hidden_weights[i][j] += alpha * inputs[j] * ds[i]
        hidden_bias[i] += alpha * ds[i]


# Data
inputs = [
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
]

outputs = [
    [0],
    [1],
    [1],
    [0]
]

for i in range(10000):
    indexes = [0, 1, 2, 3]
    random.shuffle(indexes)
    for j in indexes:
        learn(inputs[j], outputs[j], alpha=0.2)

    if (i + 1) % 1000 == 0:
        cost = 0
        for j in range(4):
            o = predict(inputs[j])
            cost += (outputs[j][0] - o) ** 2
        cost /= 4
        print(i + 1, "Mean squared error:", cost)
```

```python
for i in range(4):
    result = predict(inputs[i])
    print("\nFor input", inputs[i], "Expected", outputs[i][0], "Prediction
result", f"{result:4.4}",
          "which means", "True" if round(result) == outputs[i][0] else
"False")
```



```
1000 Mean squared error: 0.24999768838195163
2000 Mean squared error: 0.24989734783738538
3000 Mean squared error: 0.2493234851506636
4000 Mean squared error: 0.2234669426408562
5000 Mean squared error: 0.074705148049028
6000 Mean squared error: 0.009952829051747073
7000 Mean squared error: 0.004429399531413915
8000 Mean squared error: 0.0027578568095919694
9000 Mean squared error: 0.001978940801519901
10000 Mean squared error: 0.001534315413071659

For input [0, 0] Expected 0 Prediction result 0.02238 which means True

For input [0, 1] Expected 1 Prediction result 0.9534 which means True

For input [1, 0] Expected 1 Prediction result 0.9654 which means True

For input [1, 1] Expected 0 Prediction result 0.04767 which means True
```