(1)

Concept

This problem is a simple version of problem **(2-1)**. There is only two major change. One is that the hidden layer is removed from the structure of problem **(2-1)**. The other is that the problem gives “Y=1 when net>0, Y=0 when net≦0”, which means that the activation function is a step function. Other concepts are the same. For further explanation please refer to problem **(2-1)**.

Hand Writing

Given:

	x_1	x_2	d
①	-1	-1	0
②	-1	1	1
③	1	-1	1
④	1	1	0

$Net = W_{11}x_1 + W_{12}x_2$

$Y = \begin{cases} 1, & Net > 0 \\ 0, & Net \leq 0 \end{cases}$

Initial Weight: $W_{11} = +1$, $W_{12} = -1$

$\theta = 0$, $\eta = 0.1$

$err = d - Y$, $\Delta W_{11} = \eta \cdot err \cdot x_1$, $\Delta W_{12} = \eta \cdot err \cdot x_2$

Epoch 1:

① $Net = 1\times(-1) + (-1)\times(-1) = 0 \Rightarrow Y = 0$; $err = d - Y = 0 - 0 = 0$

$\therefore \Delta W_{11} = 0.1\times 0\times(-1) = 0$, $W_{11} + \Delta W_{11} = 1 + 0 = 1$

$\Delta W_{12} = 0.1\times 0\times(-1) = 0$, $W_{12} + \Delta W_{12} = -1 + 0 = -1$

② $Net = 1\times(-1) + (-1)\times 1 = -2 \Rightarrow Y = 0$, $err = d - Y = 1 - 0 = 1$

$\therefore \Delta W_{11} = 0.1\times 1\times(-1) = -0.1$, $W_{11} + \Delta W_{11} = 1 + (-0.1) = 0.9$

$\Delta W_{12} = 0.1\times 1\times 1 = 0.1$, $W_{12} + \Delta W_{12} = -1 + 0.1 = -0.9$

③④ ... and Epoch 2.3.4... goes the same.

Code (Python)

```
import numpy as np

# scipy.special for the sigmoid function expit()
import scipy.special
```

```
import matplotlib.pyplot as plt

class neuralNetwork:

    def __init__(self, learningrate):

        # w_i_j, from node i to node j in the next layer
        # w11 w21
        # w12 w22 etc
        # theta 3 is the threshold with input -1
        # [W11, W12, W13=0], [W21, W22, W23=0], [theta 31, theta 32, theta 33=1]
        self.wio = np.array([1.0, -1.0, 0.0])

        # learning rate
        self.lr = learningrate

        # activation function: sigmoid function
        self.activation_function = lambda x: 1 if x>0 else 0

        pass

    def train(self, inputs_list, target):
        # convert inputs list to 2d array
        inputs = np.array(inputs_list, ndmin=1).T
        targets = np.array(target, ndmin=1).T #  ndmin=1 changed from the
standard

        # calculate signals into hidden layer
        final_inputs = np.dot(self.wio, inputs)
        # calculate the signals emerging from hidden layer
        final_outputs = self.activation_function(final_inputs)

        # output layer error is the (target - actual)
        final_errors = targets - final_outputs

        # update the weights for the links between the input and hidden layers
        self.wio += self.lr * final_errors * np.transpose(inputs)
```

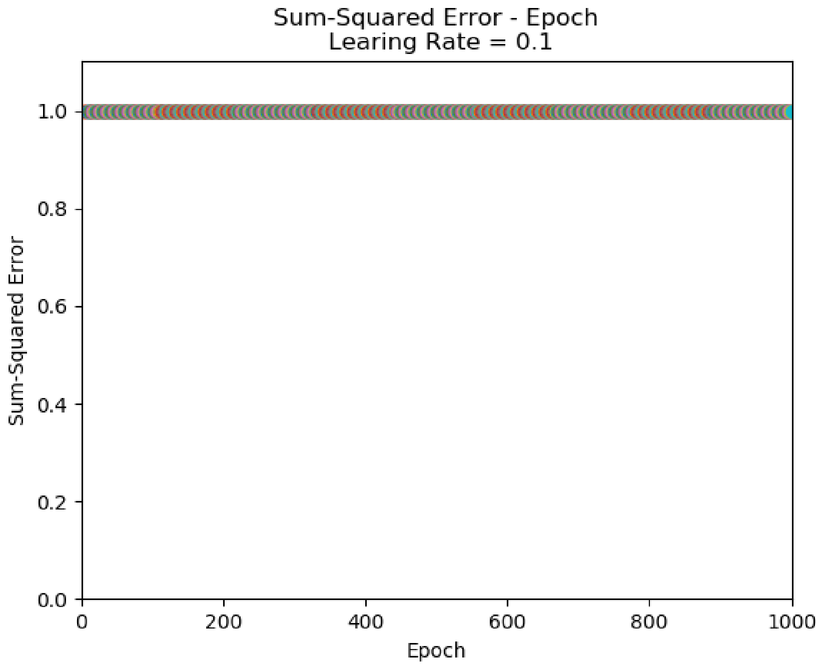
```
        pass

    # query the neural network
    def query(self, inputs_list):
        # convert inputs list to 2d array
        inputs = np.array(inputs_list, ndmin=2).T

        # calculate signals into final output layer
        final_inputs = np.dot(self.wio, inputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        return final_outputs

def main():

    input_list = []
    target_list = []

    # Number of Epoch
    epoch = 1000

    # learning rate
    learing_rate = 0.1

    # Inputs & Targets

    input_list.append([-1, -1]); target_list.append(0)
    input_list.append([-1, 1]); target_list.append(1)
    input_list.append([1, -1]); target_list.append(1)
    input_list.append([1, 1]); target_list.append(0)

    # Create an instance of neuralNetwork with the learning rate specified
    nn = neuralNetwork(learing_rate)

    # Add the threshold input
    for i in range(len(input_list)):
```

```
        input_list[i].append(-1)

    # Plot the Sum-Squared Error - Epoch
    plt.axis([0, epoch+1, 0, 1.1])
    plt.title('Sum-Squared Error - Epoch\n Learing Rate = 0.1')
    plt.xlabel('Epoch')
    plt.ylabel('Sum-Squared Error')

    # Train & Plot
    for x in range(0, epoch):
        for i in range(len(input_list)):
            nn.train(input_list[i], target_list[i])

        sum_squared_errors = 0

        for i in range(len(input_list)):
            sum_squared_errors += (nn.query(input_list[i])-target_list[i])**2

        plt.scatter(x+1, sum_squared_errors)

    plt.show()

if __name__ == '__main__':
    main()
```

Results

The result shown below is not surprise because the XOR logic problem with no hidden layer and activation function being step function.

Figure 1



Step by step check with excel explains the same as shown below.

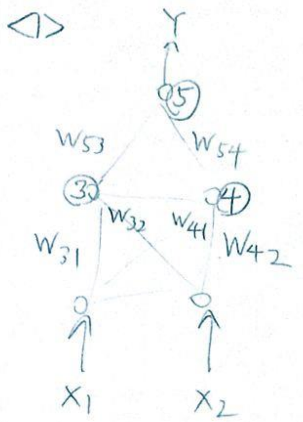
Table 1

epoch	sum-squared-error	x1	x2	d	Y	err	delta W11	delta W12	W11	W12
1	1.000	-1	-1	0	0.000	0.000	0.000	0.000	1.000	-1.000
		-1	1	1	0.000	1.000	-0.100	0.100	0.900	-0.900
		1	-1	1	1.000	0.000	0.000	0.000	0.900	-0.900
		1	1	0	0.000	0.000	0.000	0.000	0.900	-0.900
2	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.900	-0.900
		-1	1	1	0.000	1.000	-0.100	0.100	0.800	-0.800
		1	-1	1	1.000	0.000	0.000	0.000	0.800	-0.800
		1	1	0	0.000	0.000	0.000	0.000	0.800	-0.800
3	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.800	-0.800
		-1	1	1	0.000	1.000	-0.100	0.100	0.700	-0.700
		1	-1	1	1.000	0.000	0.000	0.000	0.700	-0.700
		1	1	0	0.000	0.000	0.000	0.000	0.700	-0.700
4	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.700	-0.700
		-1	1	1	0.000	1.000	-0.100	0.100	0.600	-0.600
		1	-1	1	1.000	0.000	0.000	0.000	0.600	-0.600
		1	1	0	0.000	0.000	0.000	0.000	0.600	-0.600
5	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.600	-0.600
		-1	1	1	0.000	1.000	-0.100	0.100	0.500	-0.500
		1	-1	1	1.000	0.000	0.000	0.000	0.500	-0.500
		1	1	0	0.000	0.000	0.000	0.000	0.500	-0.500
6	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.500	-0.500
		-1	1	1	0.000	1.000	-0.100	0.100	0.400	-0.400
		1	-1	1	1.000	0.000	0.000	0.000	0.400	-0.400
		1	1	0	0.000	0.000	0.000	0.000	0.400	-0.400
7	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.400	-0.400
		-1	1	1	0.000	1.000	-0.100	0.100	0.300	-0.300
		1	-1	1	1.000	0.000	0.000	0.000	0.300	-0.300
		1	1	0	0.000	0.000	0.000	0.000	0.300	-0.300
8	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.300	-0.300
		-1	1	1	0.000	1.000	-0.100	0.100	0.200	-0.200
		1	-1	1	1.000	0.000	0.000	0.000	0.200	-0.200
		1	1	0	0.000	0.000	0.000	0.000	0.200	-0.200
9	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.200	-0.200
		-1	1	1	0.000	1.000	-0.100	0.100	0.100	-0.100
		1	-1	1	1.000	0.000	0.000	0.000	0.100	-0.100
		1	1	0	0.000	0.000	0.000	0.000	0.100	-0.100
10	1.000	-1	-1	0	0.000	0.000	0.000	0.000	0.100	-0.100
		-1	1	1	0.000	1.000	-0.100	0.100	0.000	0.000
		1	-1	1	1.000	0.000	0.000	0.000	0.000	0.000
		1	1	0	0.000	0.000	0.000	0.000	0.000	0.000

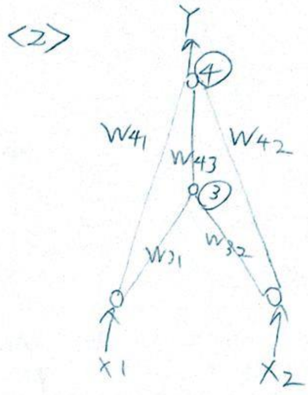
(2)

Hand Writing

<1>



<2>



Desired output : d

★ Δ 由外而內調

Actual output (in each step) : Y

★ ~~Outer Layer~~ Hidden → Outer

error: $\delta_j = (d_j - Y_j)\, f_j'(net_j)$, 其中 $f(x) = \dfrac{1}{1+e^{-x}}$

$net_j = \sum W_{ji} x_i - \theta_j$

$\Rightarrow f'(x) = f(x)\cdot(1-f(x))$

★ Input Layer → Hidden

$\Delta W_{ji} = \eta\, \delta_j\, x_i$

★ ~~Hidden Layer~~ Input → Hidden

$\delta_j = \delta_{(j\_outer)}\, W_{(j\_outer)j}\, f_j'(net_j)$

★ Hidden → Outer

$\Delta W_{ji} = \eta\, \delta_j\, y_i$

$\Delta\theta_j = -\eta\, \delta_j$, learning rate $\eta = 0.1$

<1> $y_3 = sigmoid(x_1 W_{31} + x_2 W_{32} - \theta_3)$

$y_4 = sigmoid(x_1 W_{41} + x_2 W_{42} - \theta_4)$

$Y = y_5 = sigmoid(y_3 W_{53} + y_4 W_{54} - \theta_5)$

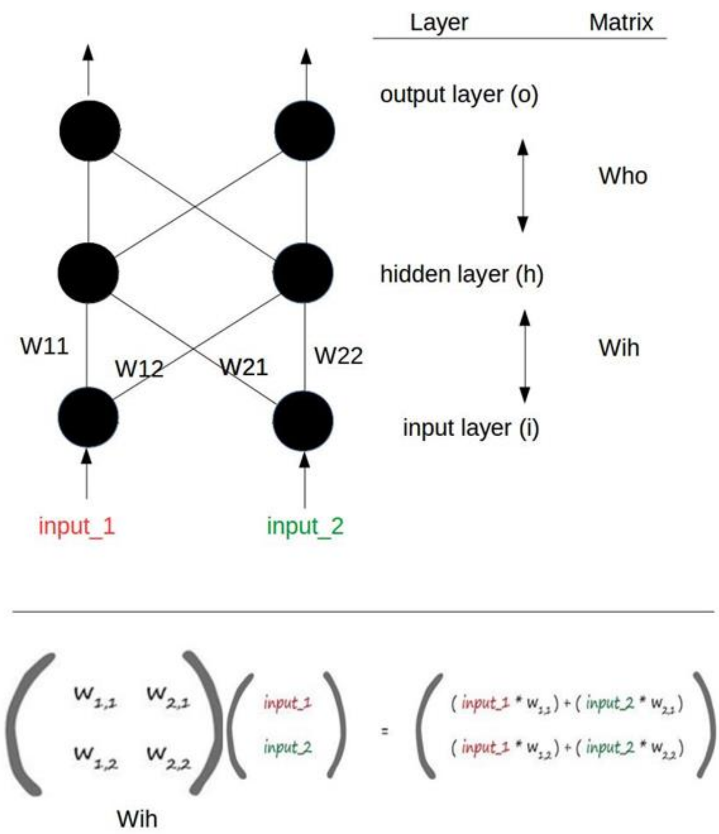
<2> $y_3 = sigmoid(x_1 W_{31} + x_2 W_{32} - \theta_3)$

$Y = y_4 = sigmoid(x_1 W_{41} + x_2 W_{42} + y_3 W_{43} - \theta_4)$

(2-1) Structure 1

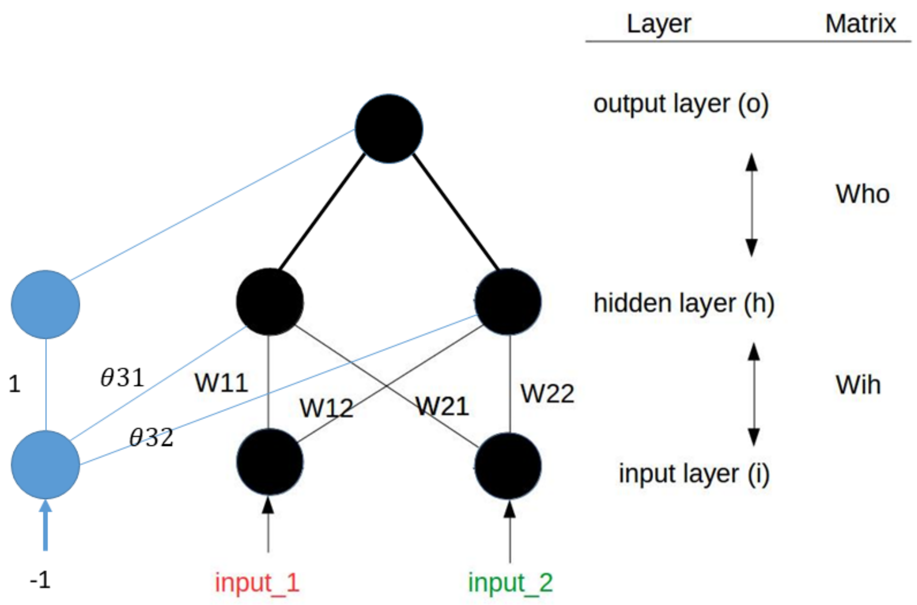
Concept

The basic concept of the code is to build a 3-layer back-propagation structure shown below. This is an expandable basic structure mentioned in the “Make Your Own Neural Network”. Note that the indexing is different from which the problem gives. Figure 2



However, modification has to be made to satisfy this problem.

Figure 3



First, make the output layer only one node (one output).

Next, realize the thresholds in activation functions with input -1 and thresholds as weights.

Given $\delta_j=(d_j-Y_j)\bullet f'(net)$ and $f(x)= 1/(1+e^{-x})$

The activation function is a sigmoid function.

Code (Python)

The code utilizes the concept mentioned above to build an expandable 3-layer neural network structure.

```
import numpy as np

# scipy.special for the sigmoid function expit()
import scipy.special

import matplotlib.pyplot as plt
```

```
class neuralNetwork:

    def __init__(self, learningrate):

        # w_i_j, from node i to node j in the next layer
        # theta 3 is the threshold with input -1
        # [W11, W12, W13=0], [W21, W22, W23=0], [theta 31, theta 32, theta 33=1]
        self.wih = np.array([[0.2, -0.4, 0], [0.2, -0.2, 0], [0.8, -0.1, 1]])
        self.who = np.array([[0.1, -0.4, 0.3]])

        # learning rate
        self.lr = learningrate

        # activation function: sigmoid function
        self.activation_function = lambda x: scipy.special.expit(x)

        pass

    def train(self, inputs_list, target):
        # convert inputs list to 2d array
        inputs = np.array(inputs_list, ndmin=2).T
        targets = np.array(target, ndmin=1).T #  ndmin=1 changed from the
standard

        # calculate signals into hidden layer
        hidden_inputs = np.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = np.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        # output layer error is the (target - actual)
        output_errors = targets - final_outputs
        # hidden layer error is the output_errors, split by weights, recombined at
hidden nodes
```

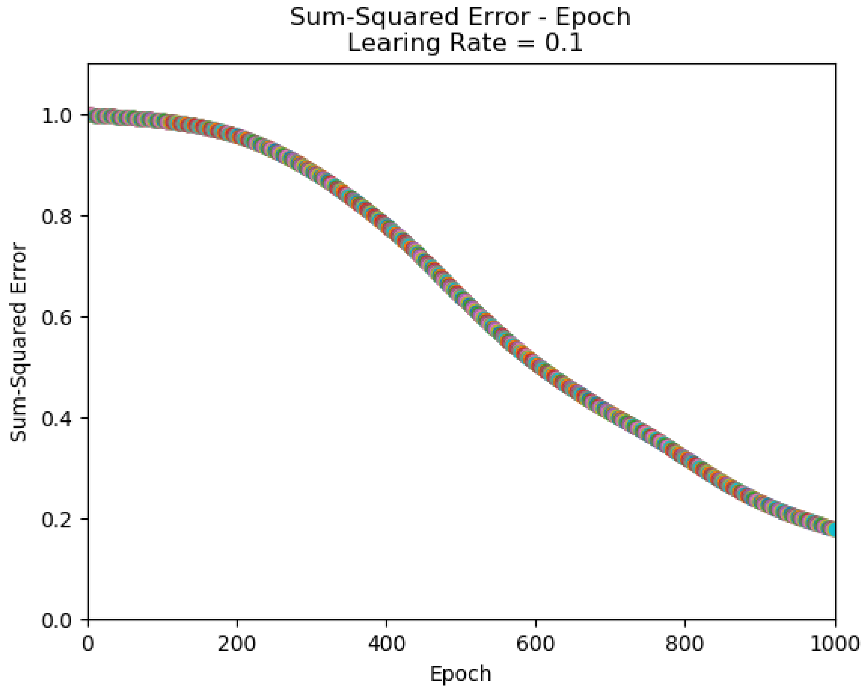
```
        hidden_errors = np.dot(self.who.T, output_errors)

        # update the weights for the links between the hidden and output layers
        self.who += self.lr * np.dot((output_errors * final_outputs * (1.0 - 
final_outputs)), np.transpose(hidden_outputs))

        # update the weights for the links between the input and hidden layers
        self.wih += self.lr * np.dot((hidden_errors * hidden_outputs * (1.0 - 
hidden_outputs)), np.transpose(inputs))

        pass

    # query the neural network
    def query(self, inputs_list):
        # convert inputs list to 2d array
        inputs = np.array(inputs_list, ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = np.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = np.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        return final_outputs

def main():
    input_list = []
    target_list = []

    # Number of Epoch
    epoch = 1000

    # learning rate
    learing_rate = 1
```

```
    # Inputs & Targets
    input_list.append([-1, -1]); target_list.append(0)
    input_list.append([-1, 1]); target_list.append(1)
    input_list.append([1, -1]); target_list.append(1)
    input_list.append([1, 1]); target_list.append(0)

    # Create an instance of neuralNetwork with the learning rate specified
    nn = neuralNetwork(learing_rate)

    # Add the threshold input
    for i in range(len(input_list)):
        input_list[i].append(-1)

    # Plot the Sum-Squared Error - Epoch
    plt.axis([0, epoch+1, 0, 1.1])
    plt.title('Sum-Squared Error - Epoch\n Learing Rate = 0.1')
    plt.xlabel('Epoch')
    plt.ylabel('Sum-Squared Error')

    # Train & Plot
    for x in range(0, epoch):
        for i in range(len(input_list)):
            nn.train(input_list[i], target_list[i])

        sum_squared_errors = 0

        for i in range(len(input_list)):
            sum_squared_errors += (nn.query(input_list[i])-target_list[i])**2

        plt.scatter(x+1, sum_squared_errors)

    plt.show()

if __name__ == '__main__':
    main()
```

Results

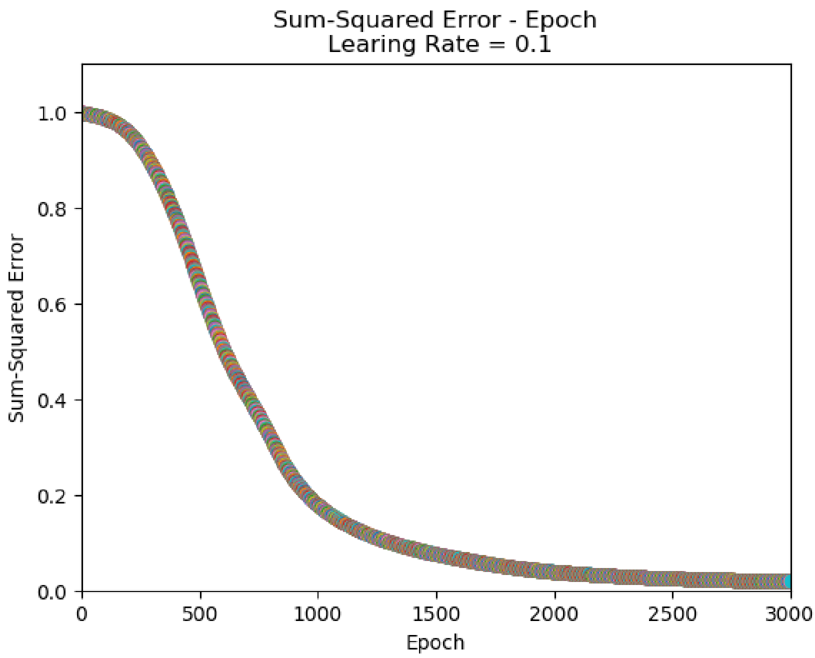
For the learning rate 0.1 specified by the problem, as shown below, the sum-squared error is still 0.2 even when epoch goes to 1000.

Figure 4



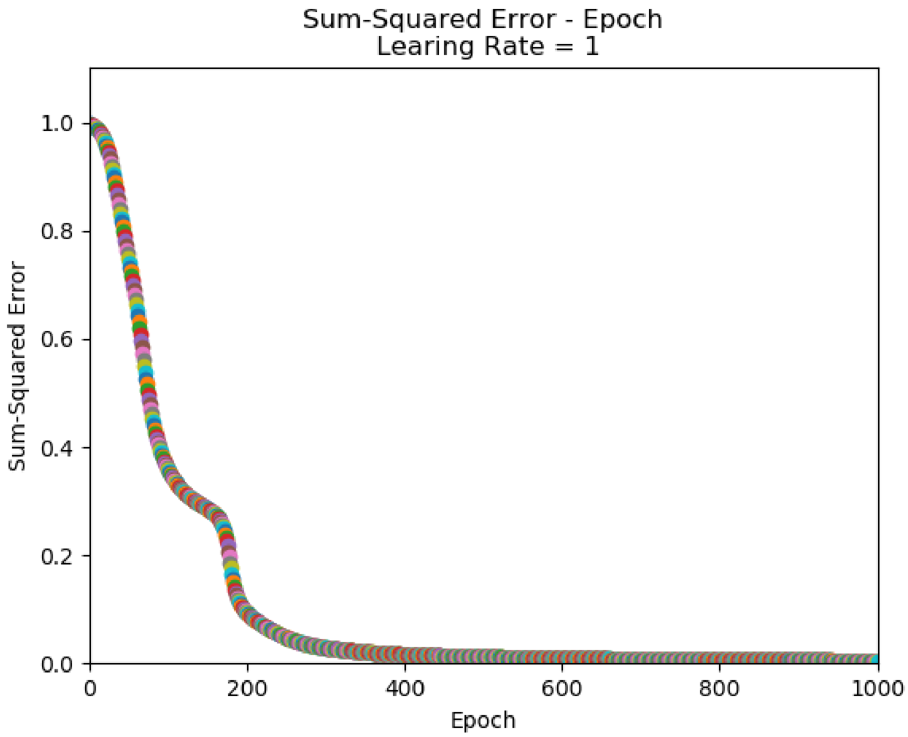
To show the convergence, increase the epoch to 3000 as shown below.

Figure 5



Now increase the learning rate to 1. The sum-squared error converges much more rapidly as shown below compared to Figure 5.

Figure 6



(2-2) Structure 2

Concept

The Structure is not a basic type defined in Structure 1. The code will not build an expandable structure for simplicity.

Code (Python)

```
import numpy as np

# scipy.special for the sigmoid function expit()
import scipy.special

import matplotlib.pyplot as plt
```

```
class neuralNetwork:

    def __init__(self):

        self.w31 = 0.2
        self.w32 = -0.4
        self.w41 = 0.2
        self.w42 = -0.2
        self.w43 = -0.4
        self.w43 = -0.4

        self.theta3 = 0.8
        self.theta4 = 0.3

        self.learningRate = 0.1
        pass


    def train(self, x1, x2, target):
        net3 = x1*self.w31+x2*self.w32-self.theta3
        y3 = scipy.special.expit(net3)

        net4 = x1*self.w41+x2*self.w42+y3*self.w43-self.theta4
        Y = scipy.special.expit(net4)

        error4 = (target - Y)* scipy.special.expit(net4)* (1 - scipy.special.expit(net4))
        error3 = error4* self.w43* scipy.special.expit(net3)* (1 -
scipy.special.expit(net3))

        self.w43 += self.learningRate* error4* y3

        self.w41 += self.learningRate* error4* x1
        self.w42 += self.learningRate* error4* x2

        self.w31 += self.learningRate* error3* x1
        self.w32 += self.learningRate* error3* x2

        self.theta3 += -self.learningRate* error3
```

```
        self.theta4 += -self.learningRate* error4
        pass

    def query(self, x1, x2):
        net3 = x1*self.w31+x2*self.w32-self.theta3
        y3 = scipy.special.expit(net3)

        net4 = x1*self.w41+x2*self.w42+y3*self.w43-self.theta4
        Y = scipy.special.expit(net4)

        return Y

def main():

    epoch = 10000

    plt.axis([0, epoch+1, 0, 1.5])
    plt.title('Sum-Squared Error - Epoch\n Learing Rate = 0.1')
    plt.xlabel('Epoch')
    plt.ylabel('Sum-Squared Error')

    for x in range(0, epoch):

        nn = neuralNetwork()
        nn.train(-1, -1, 0)
        nn.train(-1, 1, 1)
        nn.train(1, -1, 1)
        nn.train(1, 1, 0)

        sum_squared_errors = (0 - nn.query(-1,-1))**2+(1 - nn.query(-1,1))**2+(1 - 
nn.query(1,-1))**2+(0 - nn.query(1,1))**2

        plt.scatter(x+1, sum_squared_errors)

    plt.show()

if __name__ == '__main__':
    main()
```

Result

The sum-squared error will not converge as shown below.

Figure 7