1 Task 2

1.1 Problem Description

Instead of trying to find the minimum of the Rosenbrock's function in task 1, we would like to solve the "exclusive or" (xor) problem in task 2. The problem exists out of two input bits which should produce their associated value of the (only) output bit. Please see Table 1 for the truth table.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Table 1: The truth table. When only one of the two input bits is true, the output bit should be true.

A perceptron cannot solve this problem, because the classes (different values of y) could not be separated linearly with one line in twodimensional space. Please see Figure 1 why.

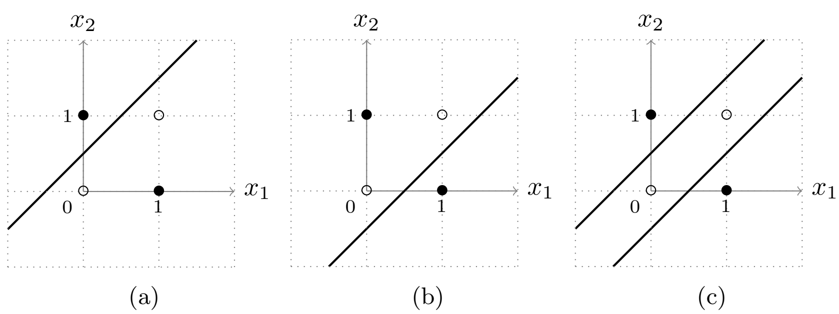

Figure 1: 3 trials to seperate the classes. The filled circles are true cases and the empty circles are false cases.

As seen in Figure 1 (c) it is possible to solve the xor problem, but not with a perceptron. We need something advanced, like a neural network.

1.2 Solution

We made a multilayer neural network with three nodes and six weights [1]. Please see Figure 2 for an illustration. The network is trained with the data of the truth table and the weights are updated with the algorithms of task 1. The algorithms

[1]The network could be trained much faster when bias nodes are used, but in sight of the assignment there is chosen to do it the hard way and make the problem more difficult.

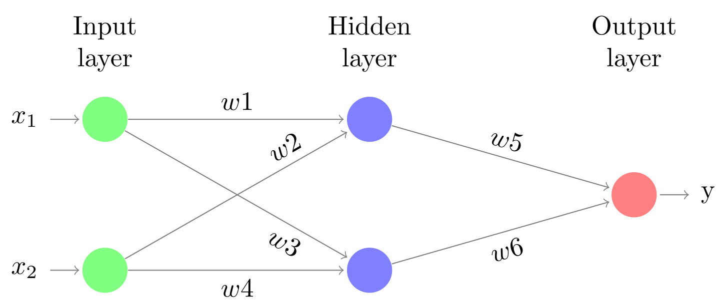

Figure 2: The xor neural network.

tries to steer the weights in such a way the errors of the predicted classes and the actual classes are minimized.

```
    function [y] = xornet(x1, x2, w)
        net1 = w(1) * x1 + w(2) * x2;
        y1 = phi(net1);
        net2 = w(3) * x1 + w(4) * x2;
        y2 = phi(net2);
        net = w(5) * y1 + w(6) * y2;
        y = phi(net);
    end
```

Figure 3: The computation of the output value, based on the input values and the weights.

```
function [d] = mysse(w)
    d = power(xornet(0, 0, w) - 0, 2) +
        power(xornet(0, 1, w) - 1, 2) +
        power(xornet(1, 0, w) - 1, 2) +
        power(xornet(1, 1, w) - 0, 2);
end
```

Figure 4: The computation of the sum squared error of the weights.

```
function [d] = dmysse(w)
    d = zeros(1, 6);

    input = [0, 0; 0, 1; 1, 0; 1, 1];
    target = [0, 1, 1, 0];

    for i = 1:4
        net1 = w(1) * input(i, 1) + w(2) * input(i, 2);
        y1 = phi(net1);
        net2 = w(3) * input(i, 1) + w(4) * input(i, 2);
        y2 = phi(net2);
        net = w(5) * y1 + w(6) * y2;
        y = phi(net);

        d(1) = d(1) + (y - target(i)) * phiprime(net) * w(5) *
            phiprime(net1) * input(i, 1);
        d(2) = d(2) + (y - target(i)) * phiprime(net) * w(5) *
            phiprime(net1) * input(i, 2);
        d(3) = d(3) + (y - target(i)) * phiprime(net) * w(6) *
            phiprime(net2) * input(i, 1);
        d(4) = d(4) + (y - target(i)) * phiprime(net) * w(6) *
            phiprime(net2) * input(i, 2);
        d(5) = d(5) + (y - target(i)) * phiprime(net) * y1;
        d(6) = d(6) + (y - target(i)) * phiprime(net) * y2;
    end
    d = d * 2;
end
```

Figure 5: The computation of the derivatives of the sum squared error of the weights.

$$\phi(x) = \frac{1}{1+e^{-x}} \tag{1}$$

$$\phi'(x) = \phi(x)(1-\phi(x)) \tag{2}$$

Figure 6: The sigmoid function (1) and the derivative (2).

analytic	diffs	delta
-0.005010573901061	-0.005010573955744	$5.4682 * 10^{11}$
-0.003205191664670	-0.003205191778655	$1.13985 * 10^{-10}$
0.064350036169591	0.064350036188543	$-0.1.8951 * 10^{-11}$
0.043391885718938	0.043391885751198	$-3.2260 * 10^{-11}$
0.185666714619300	0.185666714558330	$6.0969 * 10^{-11}$
0.181319698693786	0.181319698588922	$1.04864 * 10^{-10}$

Figure 7: Results of the gradchek function.