

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.

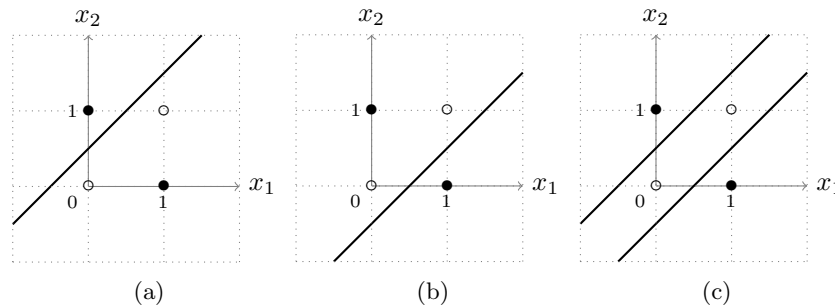


Figure 1: 3 trials to separate the classes. The filled circles are true cases and the empty circles are false cases.

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

¹The 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.

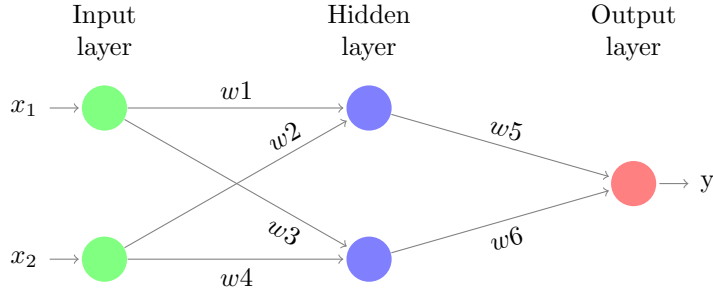


Figure 2: The xor neural network.

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

1.3 Implementation

```

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

```

Listing 1: The computation of the output value, based on the input values and the weights.

The xornet function computes the output value. The (ϕ) sigmoid function is used as the activation function, so the output value will always lay between 0 and 1. The mysse function calculates the sum squared errors of the network. For every scenario of input values (all four combinations of x_1 and x_2) the computed value is substracted with the actual value (which is found in the truth table). These values are squared and cumulated. The lower the result the better. The dmysse calculates the derivatives, which are used by the 5 algorithms. So check the correctness of this we used the gradchek function of the Netlab toolbox. The results are in Table 2. The deltas are very near to 0. The reason for this is because the gradcheck function produces an approximation.

1.4 Experiments

We ran all 5 algorithms for 100 iterations, starting from a random point. This is done 100 times to get the averages of the function evaluations, gradient evaluation and the successrate. The relative amount of times an algorithm is successful

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

$$\phi'(x) = \phi(x)(1 - \phi(x)) \quad (2)$$

Figure 3: The sigmoid function (1) and the derivative (2).

```

1 function [d] = mysse(w)
2     d = power(xornet(0, 0, w) - 0, 2) +
3         power(xornet(0, 1, w) - 1, 2) +
4         power(xornet(1, 0, w) - 1, 2) +
5         power(xornet(1, 1, w) - 0, 2);
6 end

```

Listing 2: The calculation of the sum squared error of the weights.

```

1 function [d] = dmysse(w)
2     d = zeros(1, 6);
3
4     input = [0, 0; 0, 1; 1, 0; 1, 1];
5     target = [0, 1, 1, 0];
6
7     for i = 1:4
8         net1 = w(1) * input(i, 1) + w(2) * input(i, 2);
9         y1 = phi(net1);
10        net2 = w(3) * input(i, 1) + w(4) * input(i, 2);
11        y2 = phi(net2);
12        net = w(5) * y1 + w(6) * y2;
13        y = phi(net);
14
15        d(1) = d(1) + (y - target(i)) * phiprime(net) * w(5) *
            phiprime(net1) * input(i, 1);
16        d(2) = d(2) + (y - target(i)) * phiprime(net) * w(5) *
            phiprime(net1) * input(i, 2);
17        d(3) = d(3) + (y - target(i)) * phiprime(net) * w(6) *
            phiprime(net2) * input(i, 1);
18        d(4) = d(4) + (y - target(i)) * phiprime(net) * w(6) *
            phiprime(net2) * input(i, 2);
19        d(5) = d(5) + (y - target(i)) * phiprime(net) * y1;
20        d(6) = d(6) + (y - target(i)) * phiprime(net) * y2;
21    end
22    d = d * 2;
23 end

```

Listing 3: The computation of the derivatives of the sum squared error of the weights.

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	$1.04864 * 10^{-10}$

Table 2: Results of the gradchek function.

```

1 success = round(xorner(0, 0, w)) == 0 & ...
2           round(xorner(1, 1, w)) == 0 & ...
3           round(xorner(0, 1, w)) == 1 & ...
4           round(xorner(1, 0, w)) == 1

```

Listing 4: Success boolean of an set of weights.

after 100 iterations is the successrate. An algorithm is succesful when it can predict all four scenarios correct. Listing 4 gives more details about this definition.