Machine Learning (course 1DT071) Uppsala University – Spring 2015 Report for Assignment 1 by group 6

Ludvig Sundström and John Shaw

February 9, 2015

1 Task1: Simple classification: XOR

For the first task we were asked to create a feed forward neural network (NN) and train it to implement the boolean XOR function. Using tools built into Matlab, we could experiment with different networks and observe the training process. We don't know how the trained neural network finds its solution, but in order to understand how to train the network we somehow need to visualize the training process. Using gradient descent training, we can think of the training process for each problem as finding the lowest point in a landscape. If there exists a global minimum point in this landscape, we want the training algorithm to find it. There can exist local minima though and we don't want the algorithm to get stuck in one, so we want to adjust the parameters of the descent in such a way that we don't get stuck in a local minimum. After a session of trial and error, we found that a learning rate of 2 usually resulted in a well-trained network.

Question 1 & 2

Sometimes, as shown in for example figure {1}, the error plot converges to an value significantly larger than zero. This is because the error rate is too low and we get stuck in local minima. We are taking too small steps, and can't get out of small cavities. A too large learning rate however, will cause the algorithm to take giant leaps through the landscape. Too big steps will make it very hard to find a good solution, and an oscillating behaviour is likely to occur; see figure {3}.

A visualisation of a well trained network for the xor problem is to plot the result of various values as colors, as seen in figure {4}. Here, we can see the classification of the possible input values. We can clearly see the two hyperplanes generated that separates data.

Here, we also see the hidden node values of possible inputs of the XOR function. Inspecting the plot ranges of the hidden nodes as well as the output node, we notice that the range for the hidden nodes are [-1, 1] and the range for the output function is [0, 1].

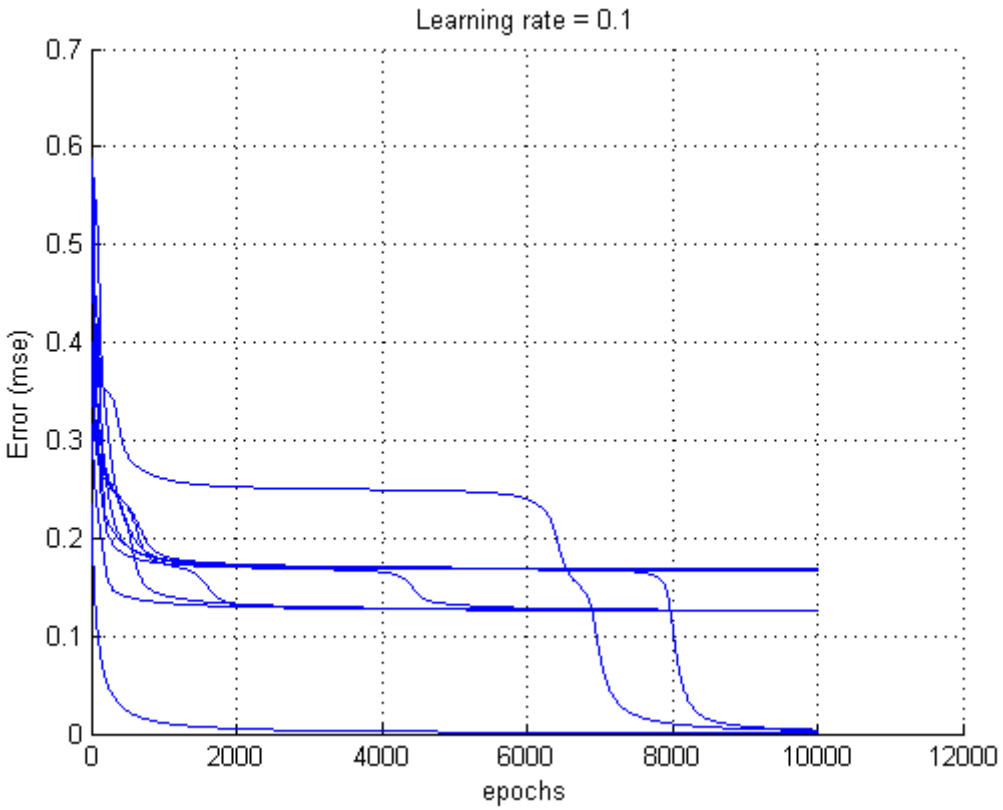

Figure 1: **Plot 1** from 10 Training sessions with a learning rate of 0.1.

Question 3

The reason that the ranges differs is that two different sigmoid functions were used as activation functions used for the network. For the hidden layer, the hyperbolic tangent function (range [-1, 1]) were used. For the output node, the logistic function (range [0, 1]) were used. The authors of this report could not find out weather it makes a difference for the algorithm weather you use a activation function that ranges from [0, 1] or [-1, 1] but it's obviously convenient if the output activation function has the same range as the actual problem output.

Question 4

A new neural network gets distributed random weights when initialized. The result of this is that the training ends with different solutions every time the network is initialized and trained and the number of epochs needed to train the network differs as well.

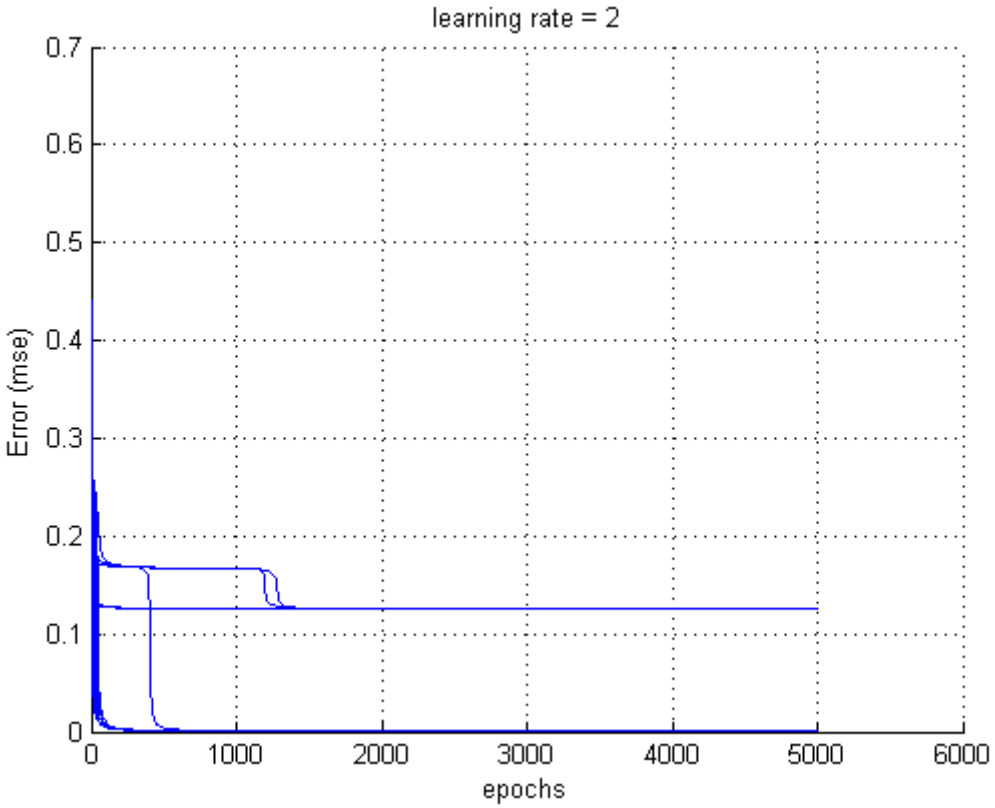

Figure 2: **Plot 1** from 10 Training sessions with a learning rate of 2.

Question 5

Again consider the coloured plot of the XOR problem network nodes (figure {4}). We already know that this represents a well trained network which means that the output node is in fact representing the boolean XOR function. As for the hidden nodes, study the value of the different inputs. For hidden node 1, an input of (0, 0) corresponds to 0 (or at least a value very close to zero). $(0, 1)$ also yields 0, $(1, 0) = 0$ and $(1, 1) = 1$. If this node implemented any boolean function it would be the AND function. In the same way, the hidden node implements the boolean OR function.

Question 6

This is verified in the truth table {1} below.

Input	boolean XOR	Network implemented XOR
0 0	0	0.0348
0 1	1	0.9744
1 0	1	0.9745
1 1	0	0.0350

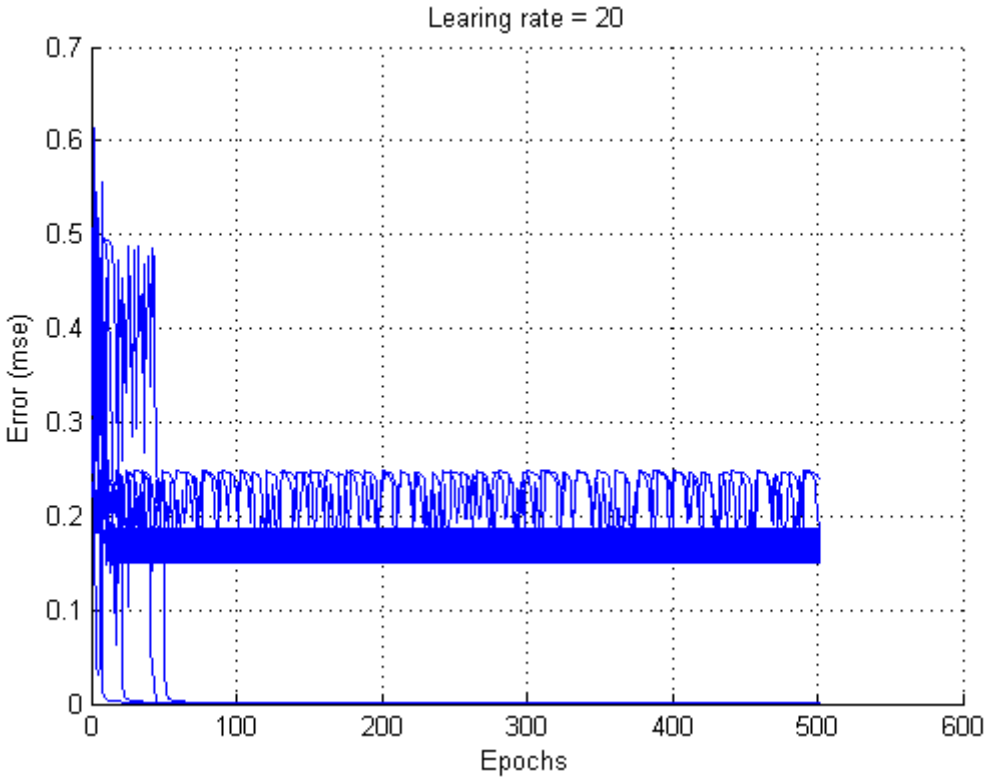

Figure 3: **Plot 1** from 10 Training sessions with a learning rate of 20.

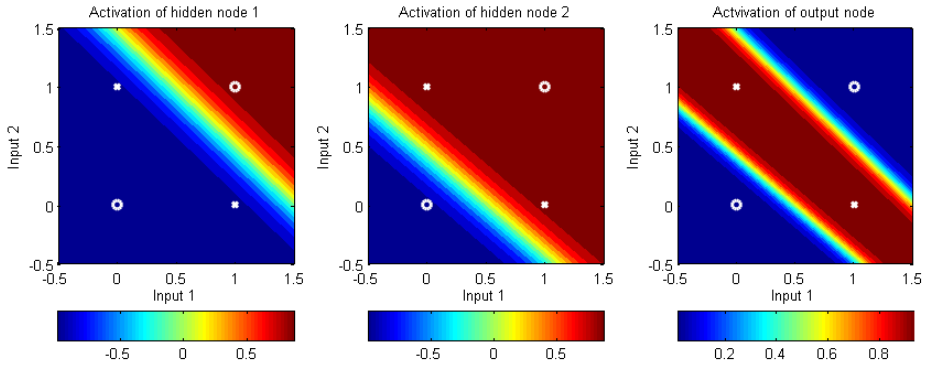

Figure 4: **Plot 2** A good solution found for the XOR problem using back-propagation

Question 7

The biggest difference is the difference in number of epoch needed. Resilient back propagation needs a lot fewer epoch to reach a minima. We think that

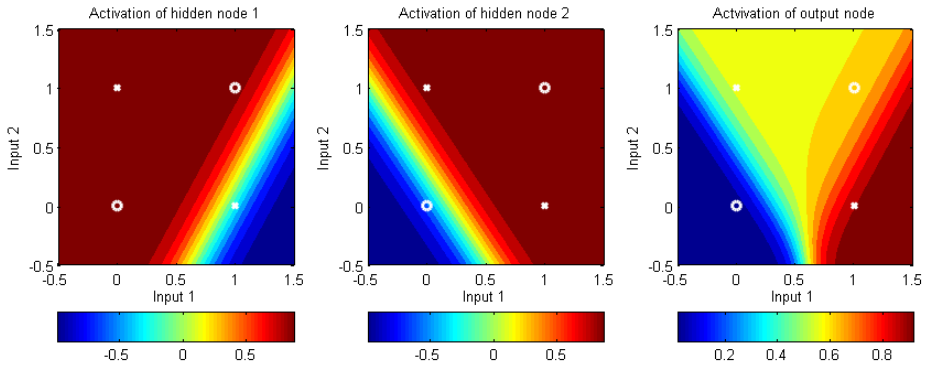

Figure 5: **Plot 2** A bad solution found for the XOR problem using back-propagation

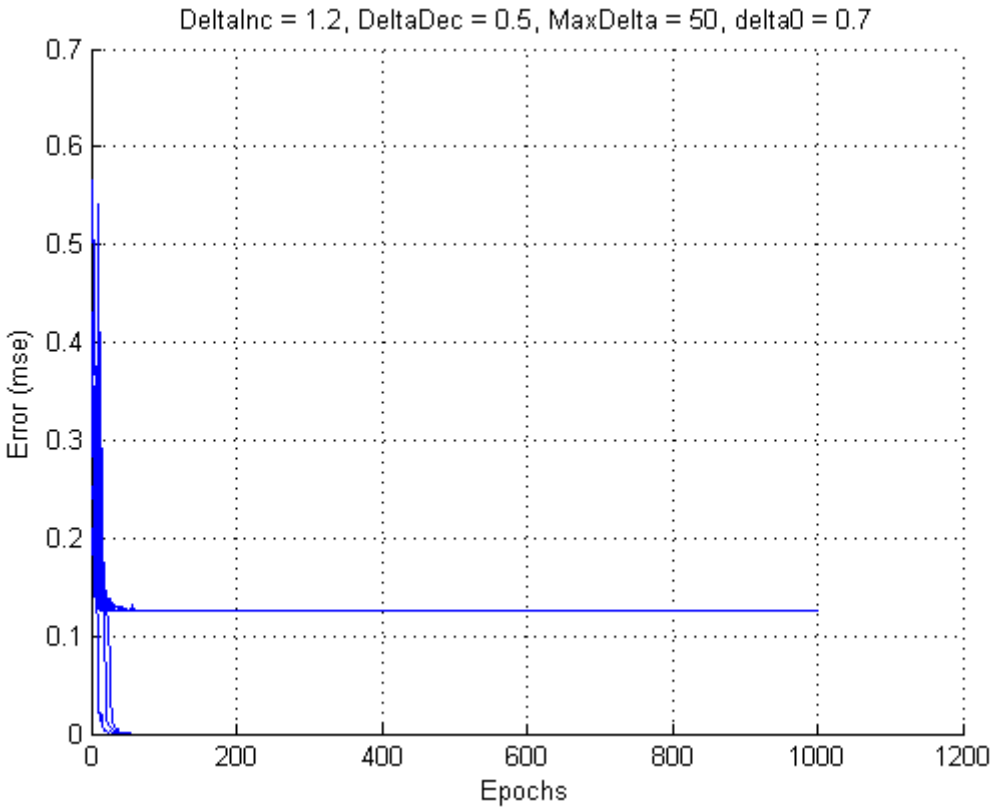

Figure 6: **Plot 3** Using the Resilient Back-propagation algorithm with the variables: $DeltaInc = 1.2, DeltaDec = 0.5, MaxDelta = 50, delta0 = 0.7$

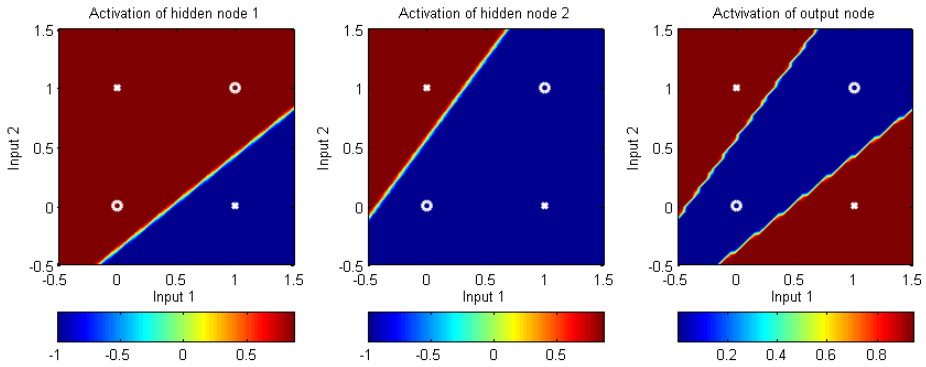

Figure 7: **Plot 3** The plot_xor of resilient back-propagation.

resilient back propagation is better suitable for this problem since it finds a minimum in about 400 times less epochs compare to the normal back propagation. This is seen in figure {6}. A interesting observation from using plot_xor, seen in figure {7} is that the way rprop have classified the groups is a mirrored image of the back-propagation.

2 Task2: Simple function approximation

In this task we are trying to approximate the function $f(x) = sin(x) \times sin(5x)$ on the interval $[0, \pi]$ with a MLP. We will use a 10 data points to approximate the function, assuming we don't know it from the start.

Question 8

When using many epochs, we notice that the higher number of nodes we have, the less MSE we get. We suspect that a AAN with more nodes can faster tweak the curve to cross the desired training targets' coordinates since they have more options to optimize the path between the targets. This gives a smaller MSE even though it doesn't necessary mean that it gives us a better answer, an example of this can be seen in the 20 nodes ANN in {11}, that performance very well in crossing the training targets' coordinates.

Question 9

The 6 node network, seen in figure {9} got the approximation that looks most like the desired function. It seems like 3 nodes is not enough to make a plot close to the training targets. This can be seen in figure {8}. We believe that each node represents a hyper plane and it seems like 3 hyper planes is not enough to plot a curve that crosses the nodes of the actual target data. It seems like the networks with bigger size tend to have to much freedom between the training targets to plot a way towards them. Furthermore the size of 6 nodes seems to limit the shape of the curve between the training target, but express enough

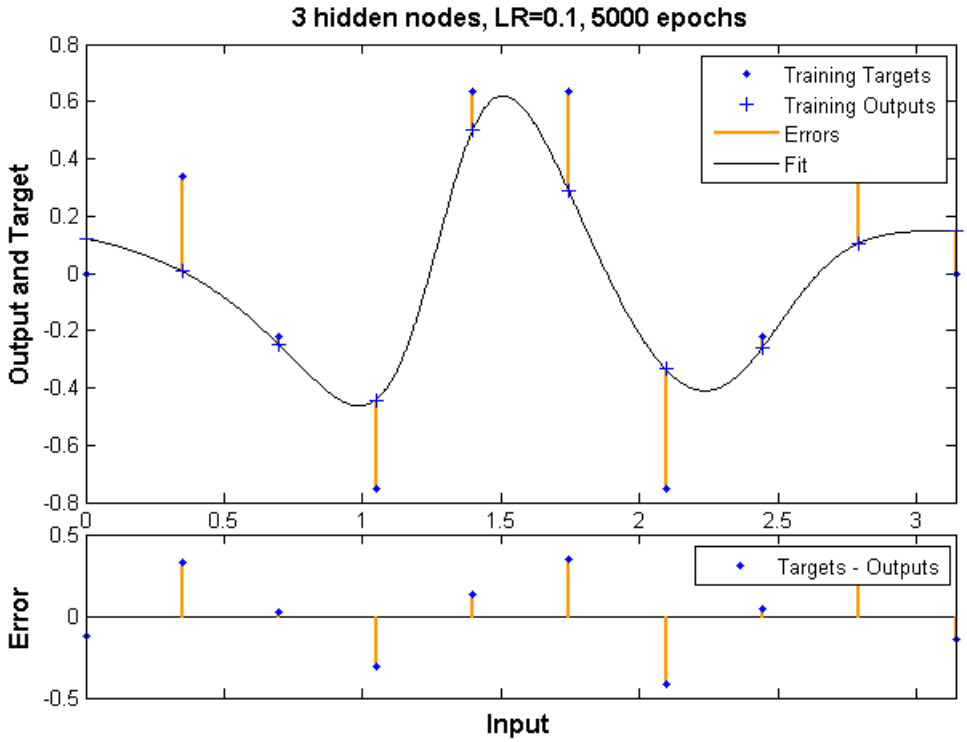

Figure 8: **Plot 4** A ANN consisting of 3 nodes with a learning rate of 0.1 for 5000 epochs

freedom to actually cross the target data's coordinates, to represent the desired function. That would as well mean that if we have more target nodes we would benefit if we increase the number of nodes in the network. The AAN with 10 nodes, seen in {10}, shows less freedom of expression between the lines than the 20 node network but a lot more than the 6 node network.

Question 10

With few hidden nodes we get to few hyper planes to classify the actual target nodes in this problem instance. This is strongly co-related to the number of coordinates the target data express and their location.

Question 11

If we have to many hidden nodes, the errors we are testing against is getting less severe, but the approximation towards the function is suffering. There are to many hyperplanes that the AAN can adjust so it's not likely that it will plot it as a smooth curve between the target nodes. Instead it just get rewards for reaching the target nodes, regardless of the angle of the extra not hyper planes between the target nodes.

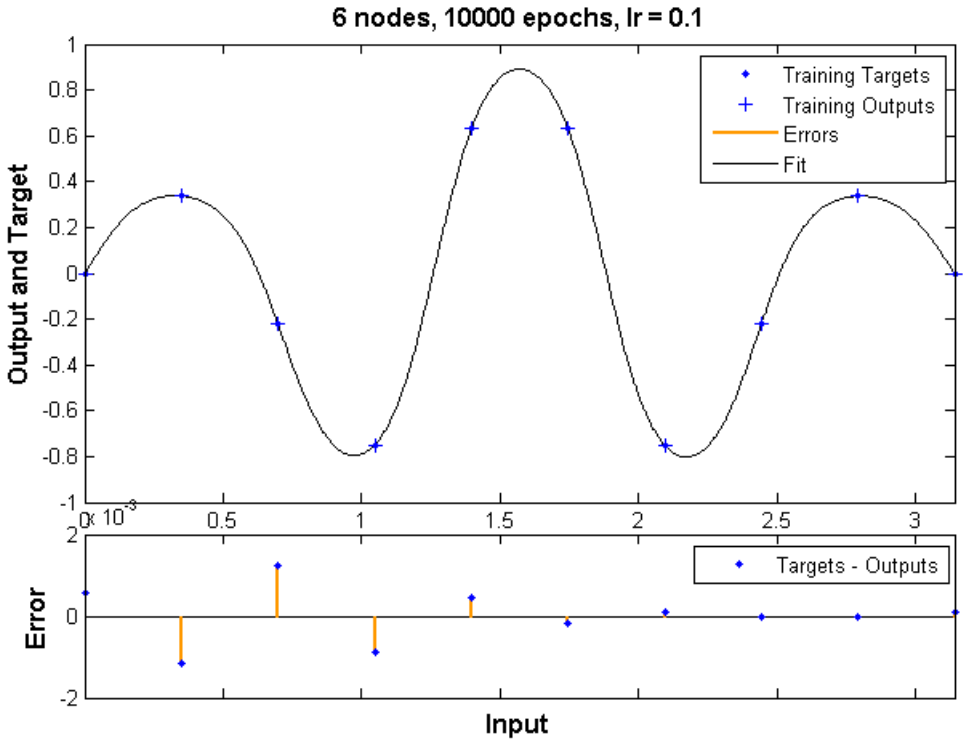

Figure 9: **Plot 4** A ANN consisting of 6 nodes with a learning rate of 0.1 for 10000 epochs

Question 12

We know that the hyperbolic tangent function can only create a few types of hyperplanes, where the most simple is similar to a line. By looking at the desired function we can imagine how many hyperplanes we need to get a good approximation of the function. For every hyperplane we need we should add one hidden node to the MLP. However we need to make sure we have enough nodes in the target data to actually represent this shape.

Question 13

With resilient back-propagation we get very fast a result that is quite similar to the desired function shape. If we let it run for a long time we don't experience any improvement. If we want more precision we can use normal back propagation but the shape will be more distorted the first iterations but quite fast it will catch up and become better than the resilient back-propagation. The choice is between training time and how close the approximation need to be.

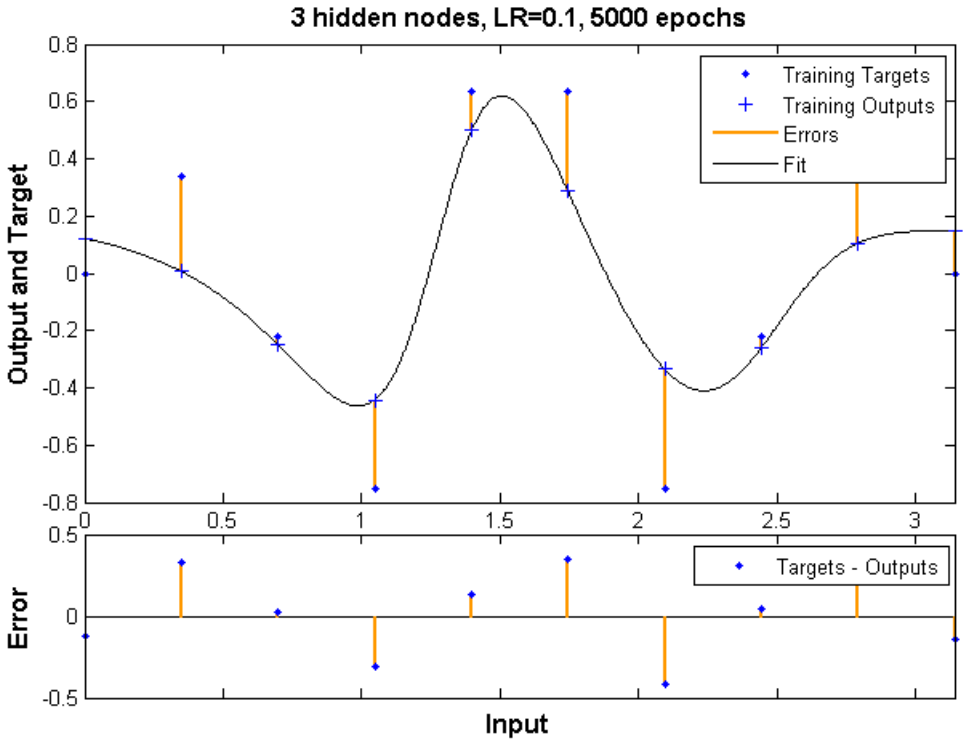

Figure 10: **Plot 4** A ANN consisting of 10 nodes with a learning rate of 0.1 for 10000 epochs

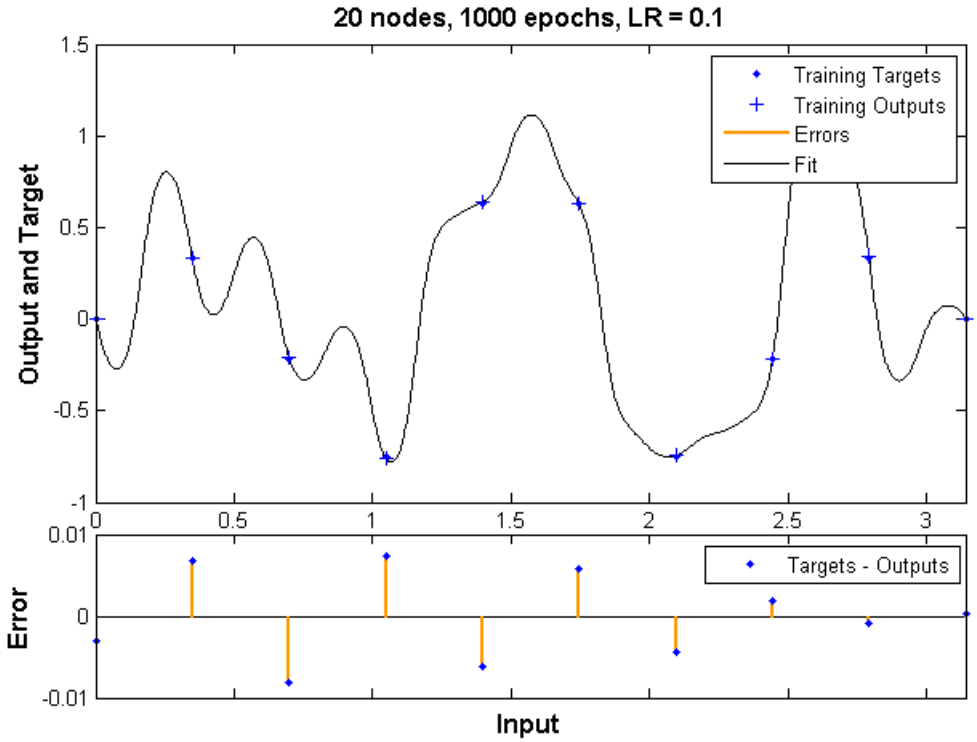

Figure 11: **Plot 4** A ANN consisting of 20 nodes with a learning rate of 0.1 for 1000 epochs

3 Task3: Classification of wine data

Question 14

Question 15

Question 16

4 Task4: Approximating house prices

Question 17

Question 18

Question 19

5 Task 5: Wrapping up

Question 20

6 Appendix

References

[1] T.H Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *Introduction to Algorithms*, 3rd edition 2009, p. 708-722, 658-659

[2] `http://en.wikipedia.org/wiki/Dijkstra's_algorithm`

[3] `http://en.wikipedia.org/wiki/Priority_queue`

[4] `http://www.geeksforgeeks.org/bipartite-graph/`

[5] `http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm`

Ideas, and general pointers how to solve the various problems have been extracted from these resources. Other then that, we declare that the content of this report is entirely pure, coming purely and directly from the it's authors brains.