Learning and generalisation in feed-forward networks — from perceptron learning to backprop

Alice Karnsund, Elin Samuelsson & Irene Natale

KTH The Royal Institute of Technology

September 12, 2018

Objectives

- design and apply networks in classification, function approximation and generalisation tasks
- identify key limitations of single-layer networks
- configure and monitor the behaviour of learning algorithms for single- and multi-layer perceptrons networks
- recognise risks associated with backpropagation and minimise them for robust learning of multi-layer perceptrons.

Classification with a single-layer perceptron

Linearly separable data

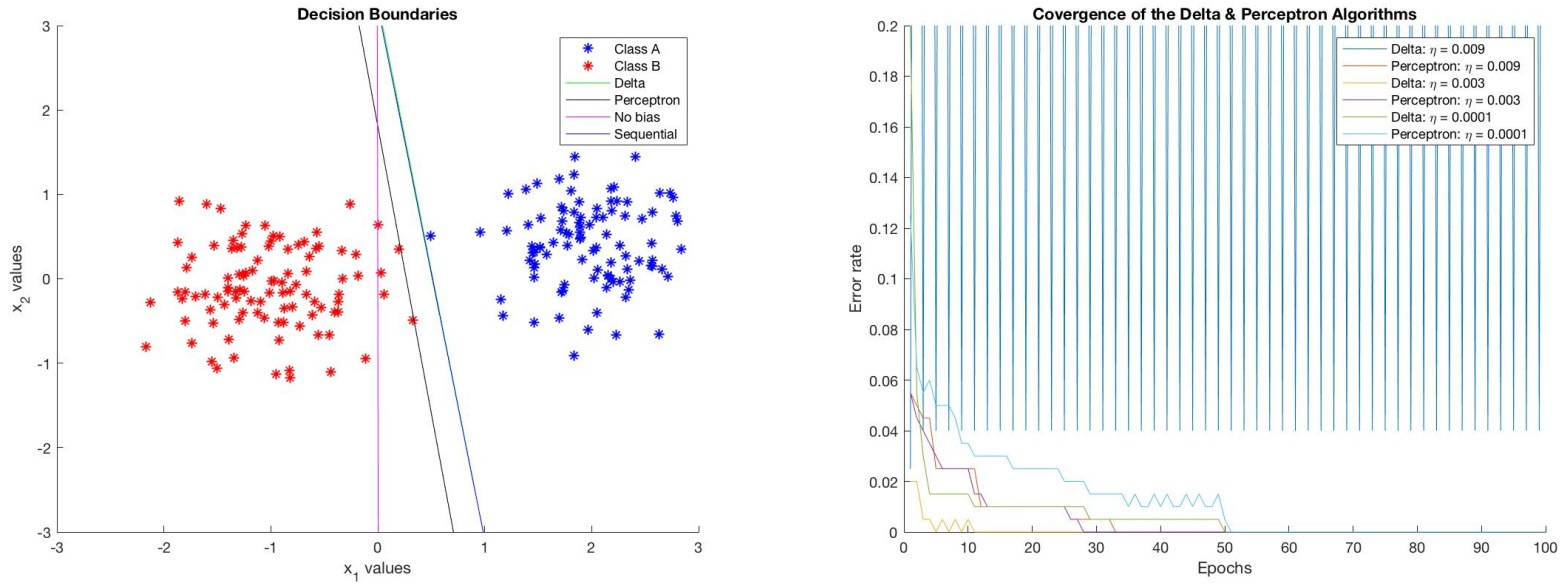
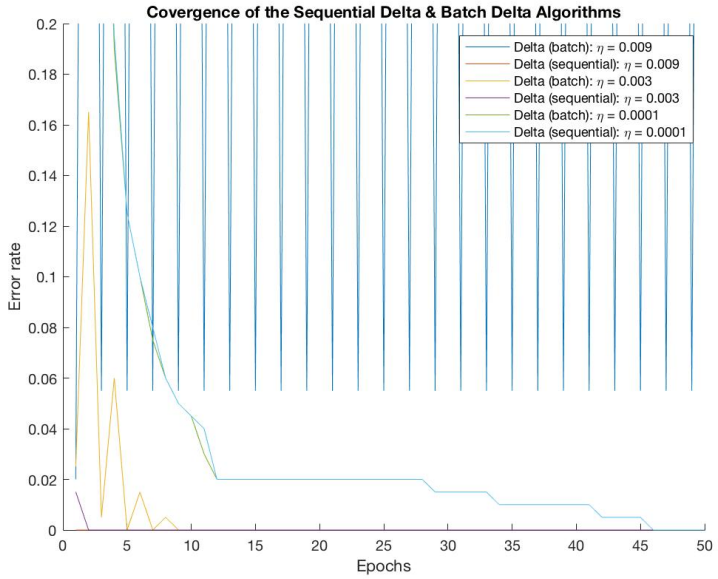


Figure 1: 1st: Four decision boundaries for $\eta = 0.003$. 2nd: Convergence plot of Delta (batch) and Perceptron algorithms.



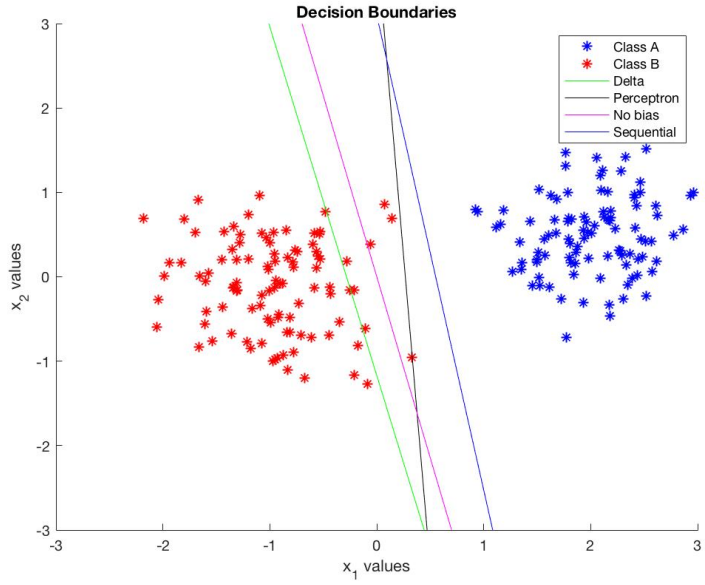


Figure 2: 1st: Learning curves for Delta batch vs Delta sequential. 2nd: Decision boundaries with $\eta = 0.009$.

	Epochs		
Algorithm	$\eta = 0.009$	$\eta = 0.003$	$\eta = 0.0001$
Delta (batch)	inf	1.8	5.7
Delta (sequential)	4.1	3.6	16.7

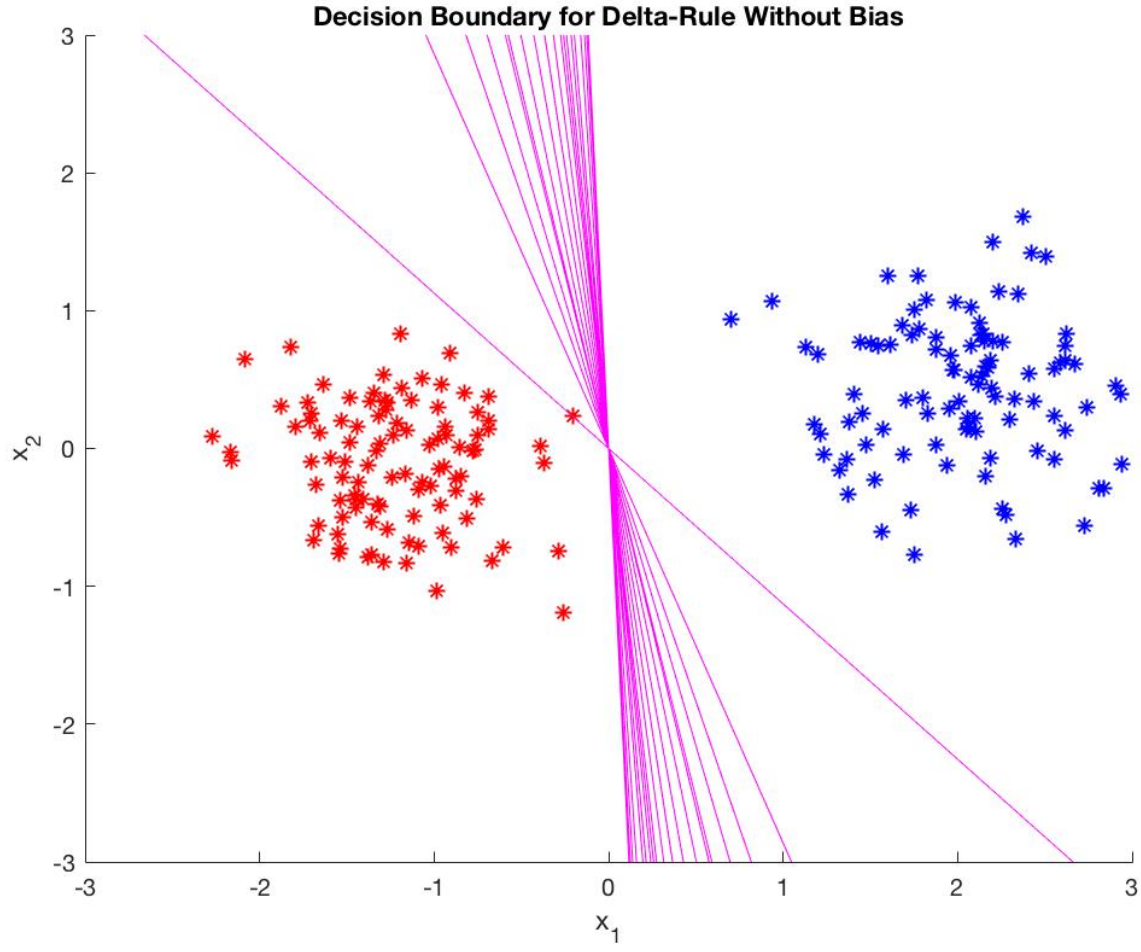
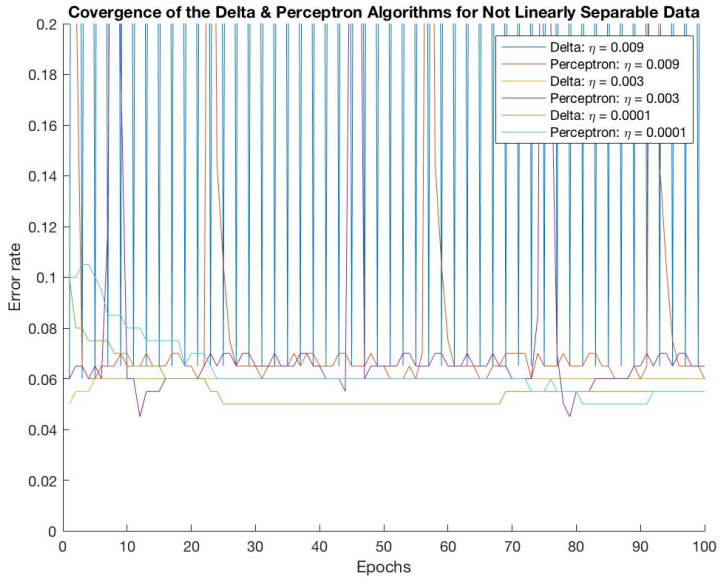


Figure 3: Decision boundary for Delta rule without bias, $\eta = 0.003$ and 20 epochs.

Non-linearly separable data



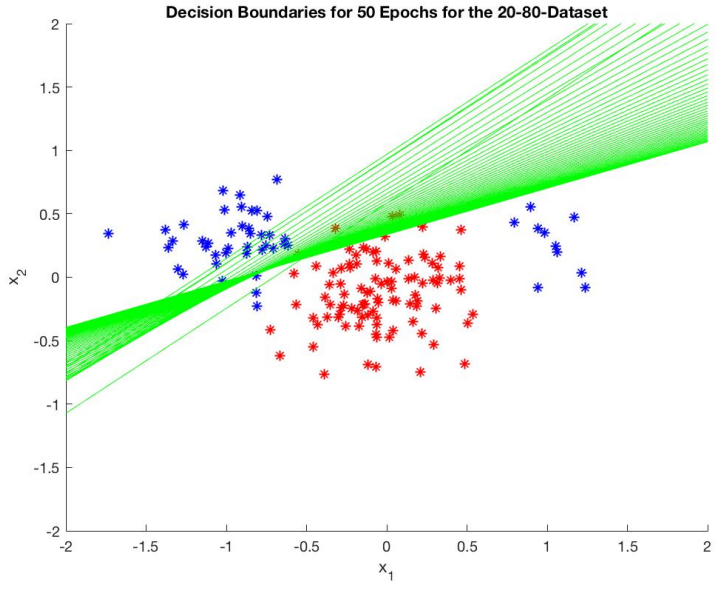


Figure 4: 1st: Convergence plot of Delta (batch) and Perceptron algorithms for non-linearly separable data, $\eta = 0.003$ and 100 epochs. 2nd: Decision boundaries for the 20-80-dataset after 50 epochs. Focus is clearly on the bigger cluster of A.

	Accuracy A/B	
Training Dataset	Training	Test
Random 75% of each class	0.162/0.240	0.217/0.284
Random 50% of A & all of B	0.695/0.061	0.669/*NaN*
Random 50% of B & all of A	0.038/0.470	*NaN*/0.565
20%-80%-ClassA & all of B	0.274/0.043	0.860/*NaN*

Table 1: Accuracy rate estimated independently for each class, i.e amount of missclasifications per group. Averaged over 10 iterations of 50 epochs each.

Classification and regression with a two-layer perceptron

Effect the size of the hidden layer has on the performance.

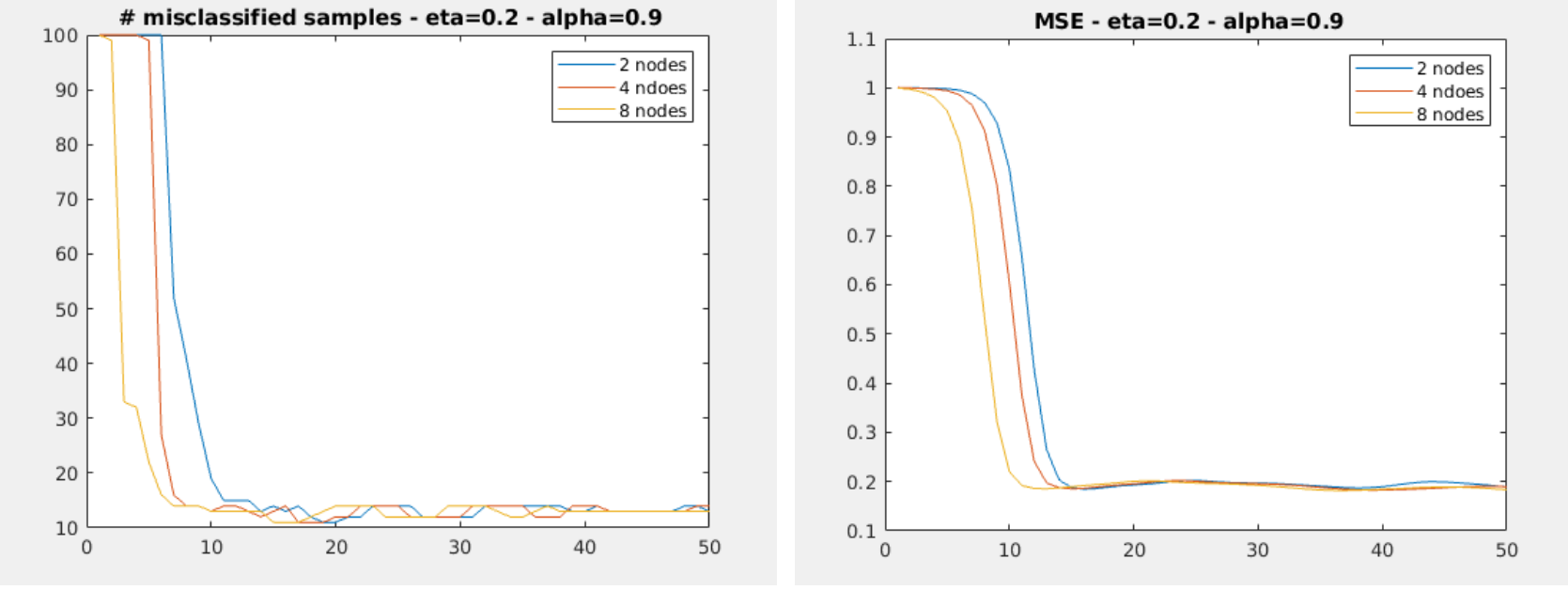


Figure 5: 1st: # MS for 2,4,8 hidden nodes at every epoch iteration 2nd: Decision boundaries for the 20-80-dataset after 50 epochs. Focus is clearly on the bigger cluster of A.

How many hidden nodes do you need to perfectly separate all the available data?

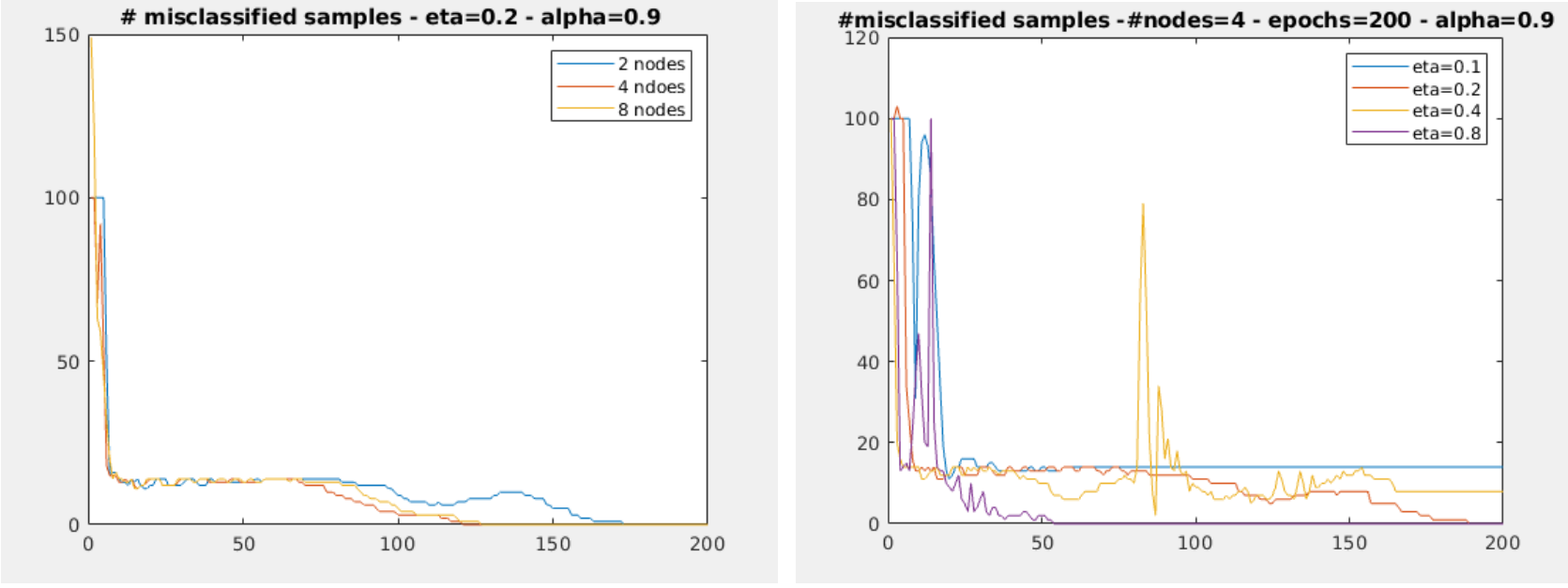


Figure 6: 1st: # MS for 2,4,8 hidden nodes at every epoch iteration (200). 2nd: Convergence for different values of learning rate.

How quickly does the learning converge depending on the learning rate?

Train/Test datasets: 25-25, 50-00, 00-50, 20-80. How do the learning/error curves for the training and the validation sets compared?

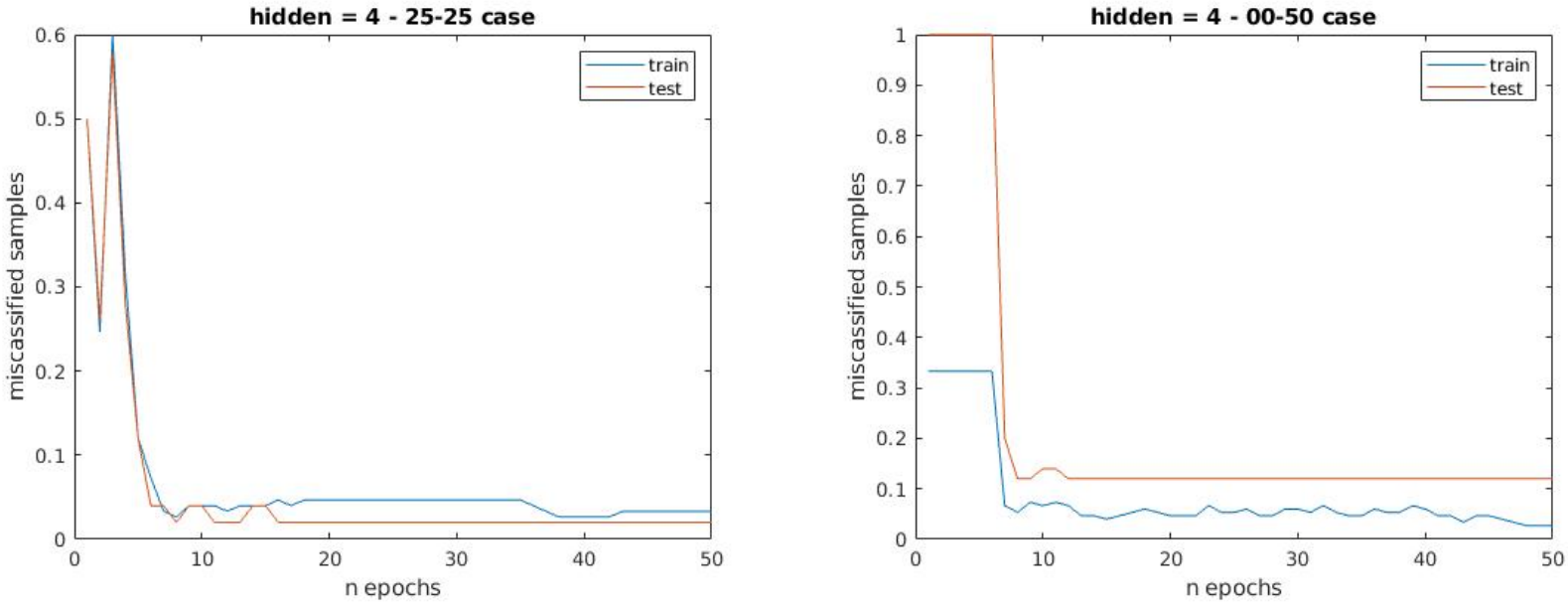


Figure 7: # MS for 4 hidden nodes.

In what cases do you observe more dissimilarity?

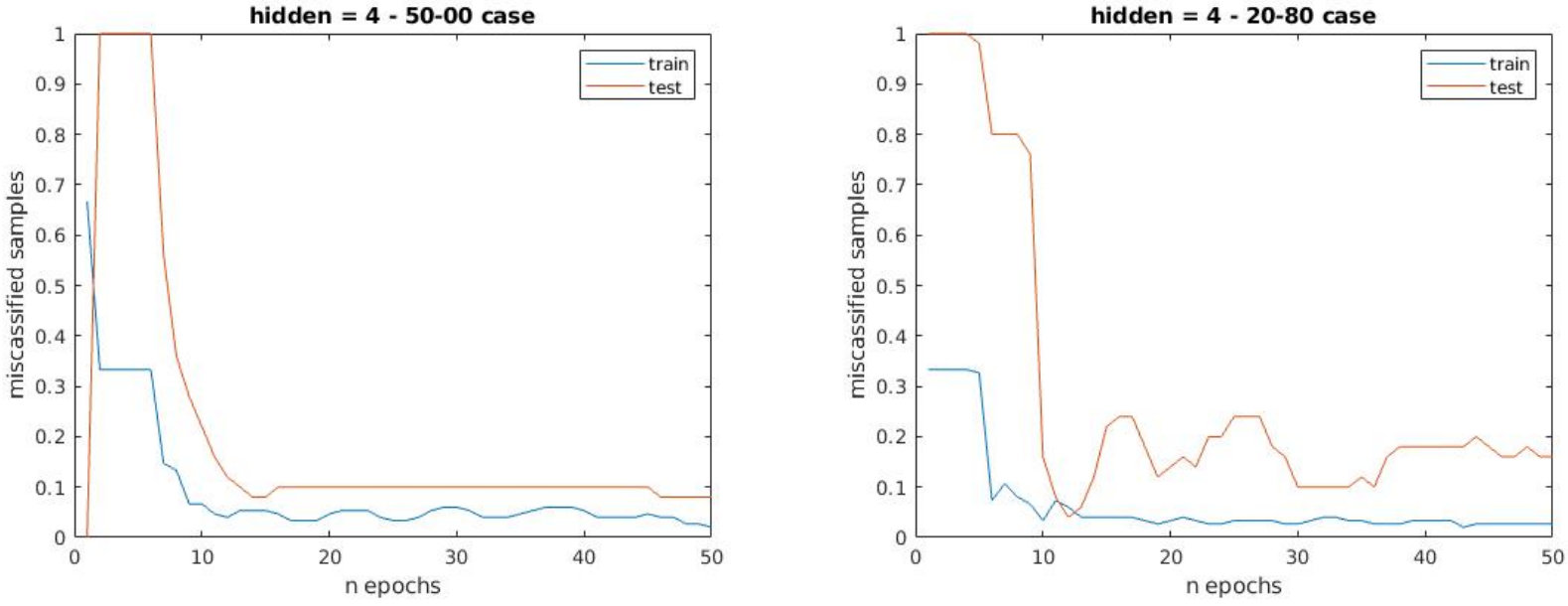


Figure 8: # MS for 4 hidden nodes.

How do these curves and the network performance depend on the size of the hidden layer in various training/validation data configurations ?

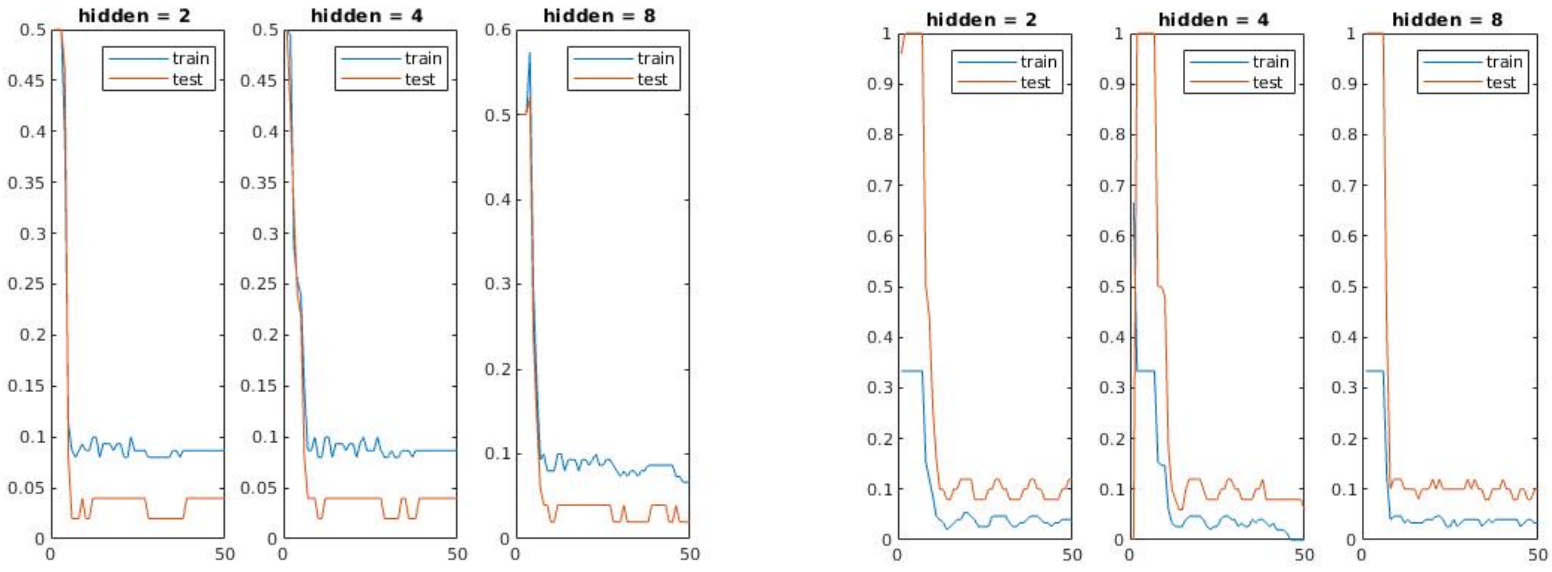


Figure 9: # MS for 2-4-8 hidden nodes in 25-25 and 50-00 cases.

Make an attempt at approximating the resulting decision boundary.

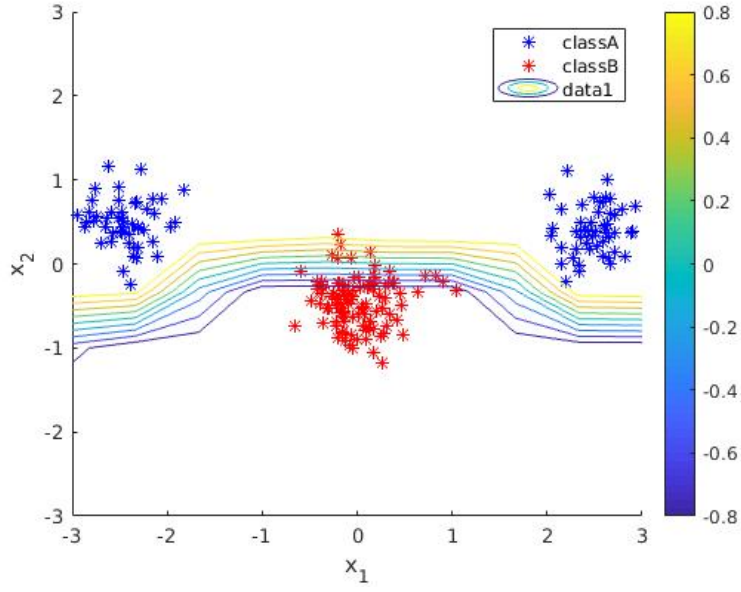


Figure 10: Boundary Layer - 4 hidden nodes.

The encoder problem

Does the network always converge?

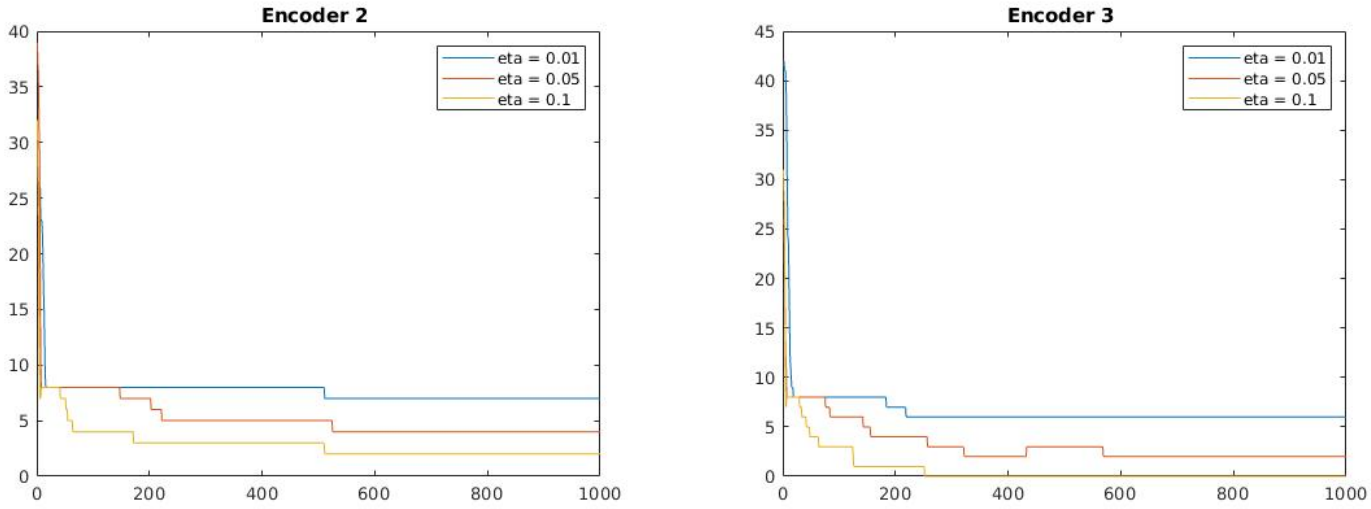


Figure 11: # MS for different η values, 2 and 3 hidden nodes.

Internal final weights encode dimensional reduction. Every input activates at least once.

Function approximation

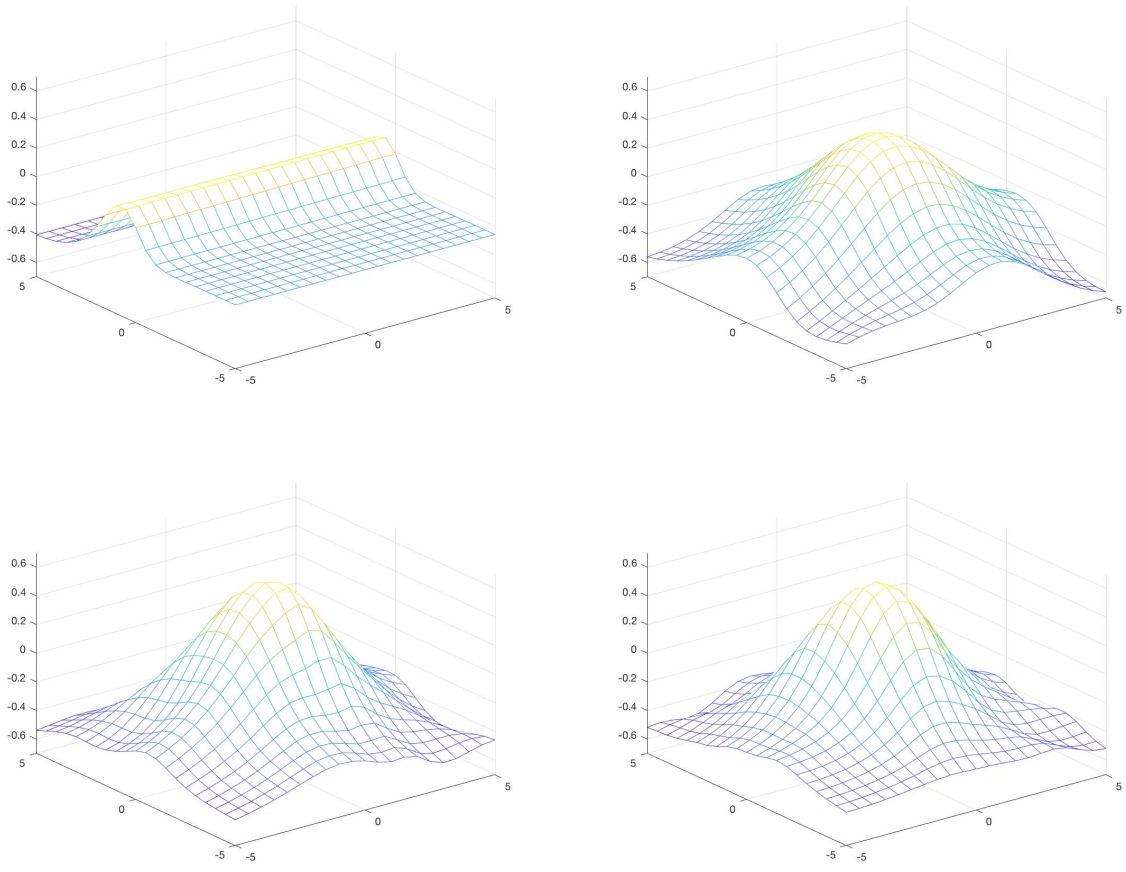
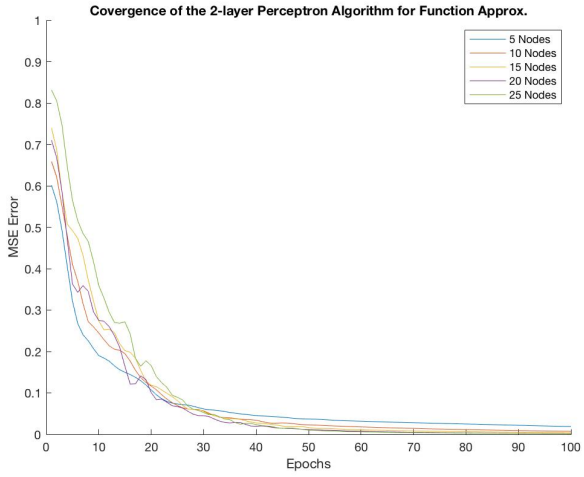


Figure 12: Function approximation with the 2-layer perceptron. The number of hidden nodes are 2, 10, 18 and 25 respectively.



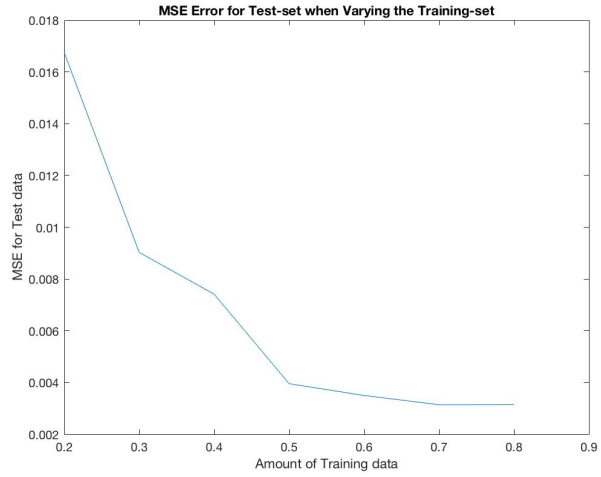


Figure 13: Averaged results over 10 iterations of 100 epochs each, $\eta = 0.009, \alpha = 0.9$. 1st: Performance of the 2-layer perceptron, varying # hidden nodes. 2nd: MSE of the model with 22 hidden nodes for all available data (training+validation).

Amount of training data	Error (MSE)
80%	0.0032
60%	0.0035
40%	0.0074
20%	0.0167

Table 2: Performance of the 22 node model, $\eta = 0.009$, $\alpha = 0.9$. Results are averaged over 10 iterations of 100 epochs. Test set is all available data (training+validation).

Part II

- Function Fitting Neural Network fitnet, a specialized version of feedforwardnet
- Default Early Stopping Criterion max_fail = 6
- Scaled conjugate gradient backpropagation trainscg
- Built in regularization method net.performParam.regularization = VARIED
- Training 60%, valdation 20%, testing 20%. No shuffling.
- Average over 100 training sessions.

Mackey-Glass Time Series

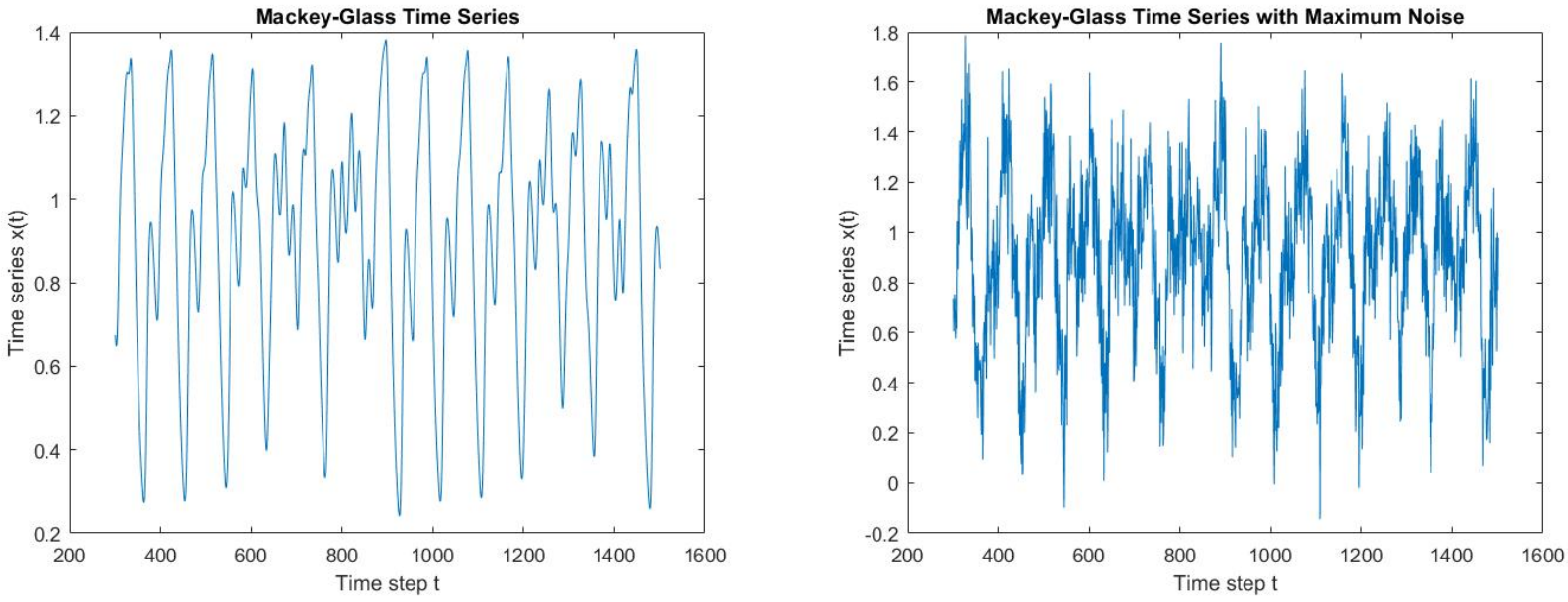


Figure 14: The time series without and with maximum noise.

Two-Layer Perceptron – Mean Validation MSE

r ↓ \ h →	2	4	6	8
0.00	3.5210e-03	1.5283e-03	1.0841e-03	1.2822e-03
0.25	4.0385e-03	1.4050e-03	1.2784e-03	1.2079e-03
0.50	4.1730e-03	1.4025e-03	1.2887e-03	1.2127e-03
0.75	4.3348e-03	1.4788e-03	1.2295e-03	1.1432e-03
1.00	3.8641e-03	1.1427e-03	1.3777e-03	1.1922e-03

Figure 15: Mean validation error for each configuration

- More hidden nodes generally performs better
- No regularisation effect? No overfitting problems?

Two-Layer Perceptron – STD Validation MSE

r ↓ \ h →	2	4	6	8
0.00	3.9177e-03	2.0926e-03	6.5759e-04	1.4436e-03
0.25	3.8827e-03	1.5827e-03	1.2878e-03	8.9432e-04
0.50	4.3518e-03	1.6371e-03	1.6802e-03	1.0136e-03
0.75	4.3886e-03	1.6897e-03	1.0746e-03	8.8037e-04
1.00	4.2039e-03	1.4321e-03	1.7566e-03	1.0648e-03

Figure 16: Standard deviation validation error for each configuration

- Small/large mean MSE often correspond to small/large std MSE

Histograms of Weights

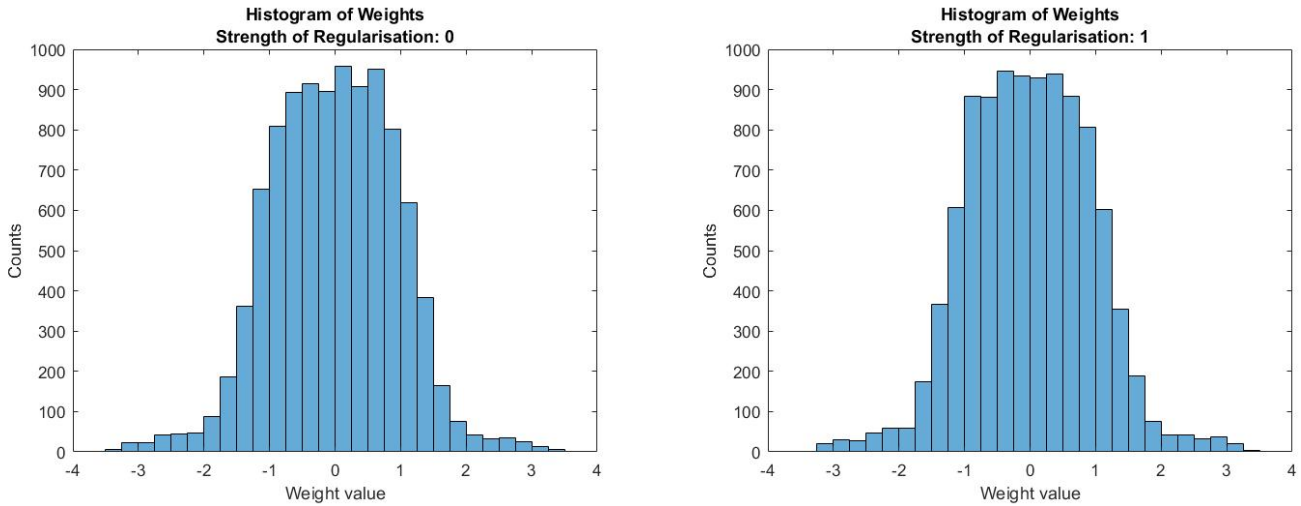


Figure 17: Weight histograms.

The modified performance function:

$$msereg = \gamma * msw + (1-\gamma) * mse, \qquad msw = \frac{1}{n}\sum_{j=1}^{n} w_j^2$$

Using this performance function causes the network to have smaller weights and biases, and this forces the network response to be smoother and less likely to overfit.

The Best Configuration

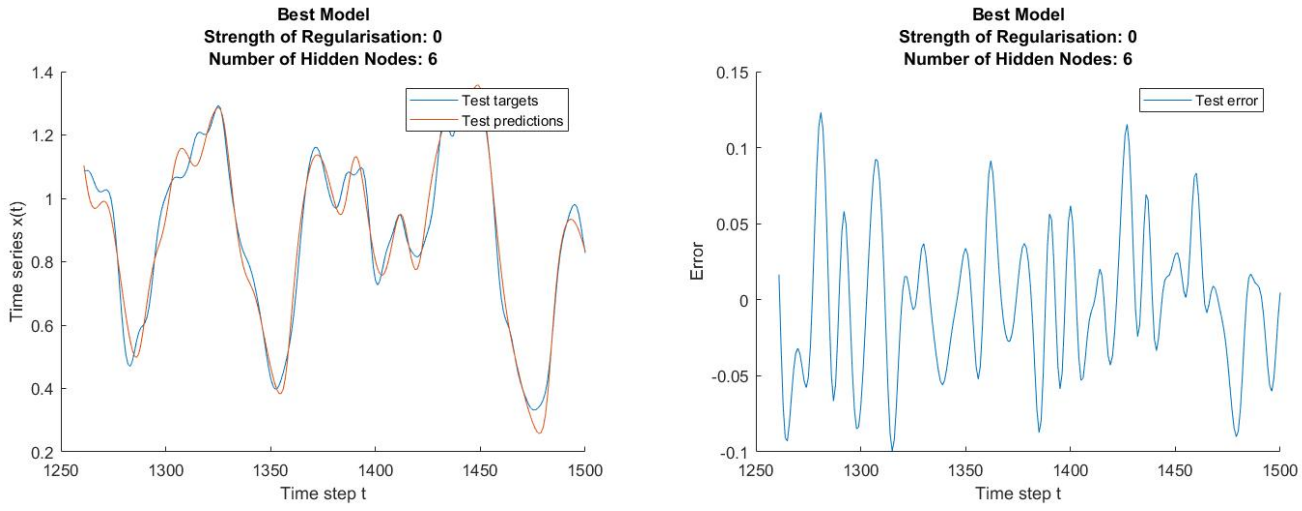


Figure 18: Performance on the test set of one example network of the best configuration.

Two- and Three-Layer Perceptron - Test MSE

type	sigma	configuration	mean test mse	std test mse
three-layer	0.03	r=0.25, h_1=6, h_2=4	2.5985e-03	1.5633e-03
three-layer	0.09	r=1.00, h_1=6, h_2=6	1.1283e-02	3.0197e-03
three-layer	0.18	r=0.00, h_1=6, h_2=8	3.8537e-02	5.6040e-03
two-layer	0.03	r=0.00, h=6	2.3104e-03	1.0587e-03
two-layer	0.09	r=0.00, h=6	1.0813e-02	2.3046e-03
two-layer	0.18	r=0.00, h=6	3.9353e-02	6.2151e-03

Figure 19: Mean validation error for the best configurations.

- No large differences between two- and three-layer

Three-Layer Perceptron - Test MSE

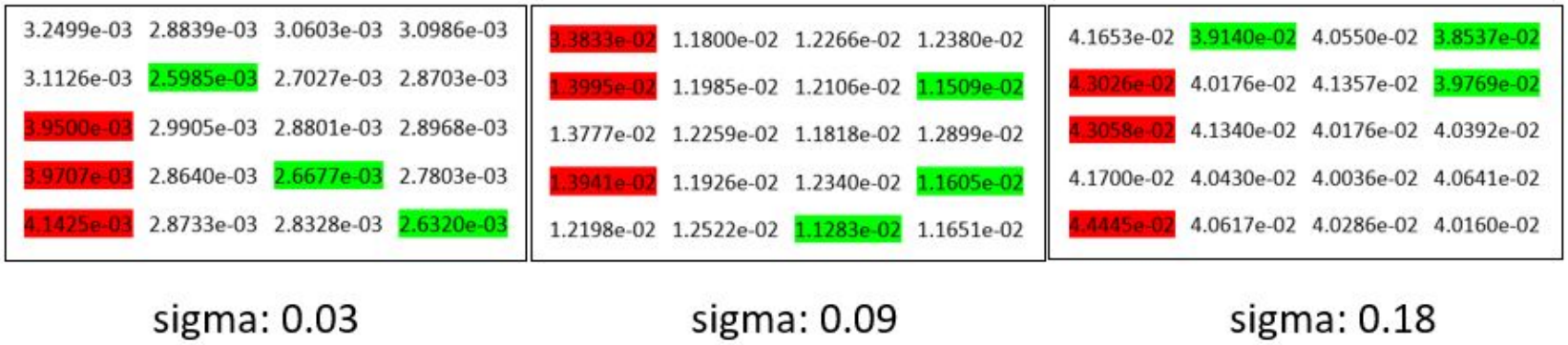


Figure 20: Mean validation error for all configurations.

- 2 nodes never enough
- The more noise, the less difference between configurations
- No regularisation effect? Because of the built-in regularisation method?

Two- and Three-Layer Perceptron - CPU Time

type	configuration	mean CPU time	std CPU time
three-layer	r=0.00, h_1=6, h_2=2	6.2484e-01	2.0276e-01
three-layer	r=0.00, h_1=6, h_2=4	5.4406e-01	1.6815e-01
three-layer	r=0.00, h_1=6, h_2=6	6.0766e-01	1.8415e-01
three-layer	r=0.00, h_1=6, h_2=8	6.6391e-01	1.5914e-01
two-layer	r=0.00, h=6	5.5203e-01	1.8845e-01

Figure 21: CPU time error for the best configurations.

- A trade-off:
- More nodes = more weights to train = increased CPU time
- More nodes = may overfit and lead to early stopping = decreased CPU time