Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1

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

Alice Karnsund, Elin Samuelsson and Irene Natale

September 10, 2018

1 Main objectives and scope of the assignment

Our major goals in the assignment are

- to implement a one- and two-layer perceptron, investigate its performance, properties and dependencies
- to use NN toolbox to investigate important properties of two- and three-layer perceptrons.

2 Methods

We used `Matlab` throughout this lab. For part II, we specifically used the NN Toolbox.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron *(ca.1 page)*

Conclusions from Figure 1: The Delta algorithm never converges for $\eta = 0.009$, since it takes too large steps and misses the minimum. The Delta-rule without bias can classify correctly if the decision boundary can go through $x = [0, 0]$. The convergence of both Delta Batch and Sequential respond to the initial values of W, but even more importantly to the learning rate. Table 1 shows that all algorithms converge relatively fast when the clusters are easily separable and initial W-values not too randomly chosen.

	Epochs		
Algorithm	$\eta = 0.009$	$\eta = 0.003$	$\eta = 0.0001$
Delta-rule	inf	1.8	5.7
Perceptron	4.1	3.6	16.7
Delta-rule (sequential)	4.1	3.6	16.7
Delta-rule (no Bias)	1	1	1

Table 1: Convergence with criteria = 5/200 misclassified samples. Results after taking the mean of 10 iterations of 100 epochs each.

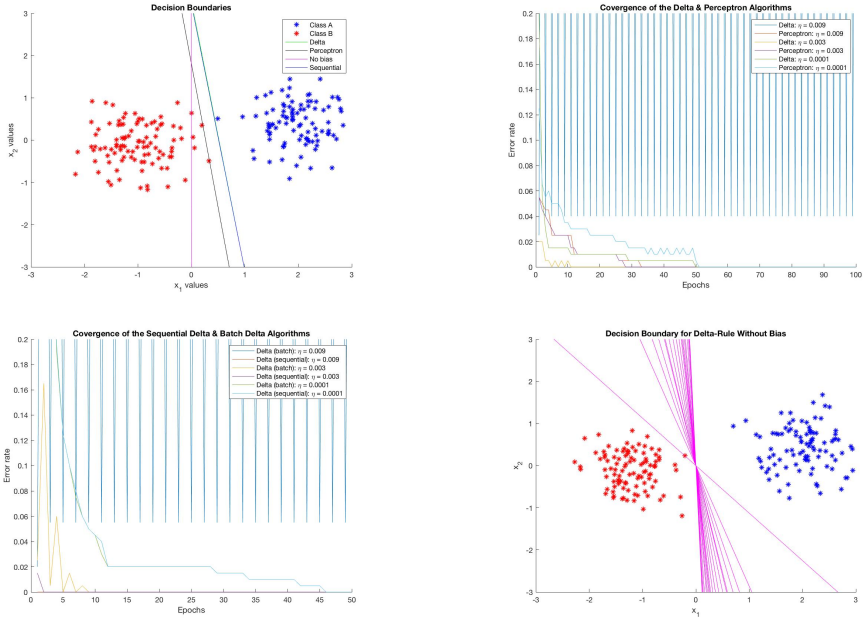

Figure 1: 1st: Four decision boundaries for $\eta = 0.003$. 2nd: Convergence plot of Delta and Perceptron algorithms. 3rd: Learning curves for Delta batch vs Delta sequential. 4th: Decision boundary for Delta rule without bias, $\eta = 0.003$ and 20 epochs.

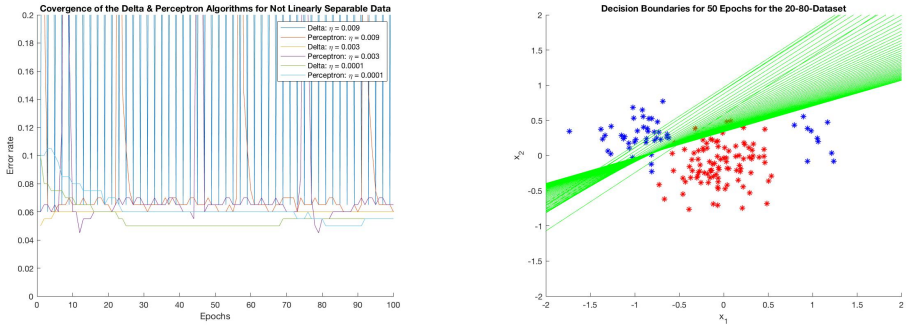

Figure 2: 1st: Convergence plot of Delta (batch) and Perceptron algorithms for non-linearly separable data, $\eta = 0.003$ and 100 epochs. 2nd: Decision boundaries for the 20-80-dataset after 50 epochs. Focus is clearly on the bigger cluster of A.

Training Dataset	Accuracy A/B (Training)	Accuracy A/B (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 2: Accuracy rate estimated independently for each class, i.e amount of missclasifications per group. Averaged over 10 iterations of 50 epochs each.

3.2 Classification and regression with a two-layer perceptron *(ca.2 pages)*

3.2.1 Classification of linearly non-separable data

Figure 5 shows that # misclassified samples (MS) and mean square error (MSE) decrease as # hidden nodes increases. Two nodes are enough for convergence, and faster if η increase. For all but the 25-25 case, the MS test curve is significantly worse than the train curve in figure 4. Increasing # nodes can improve performance, but a sequential approach would not give different results.

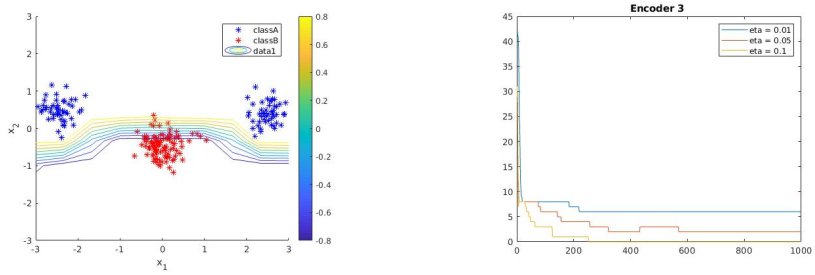

Figure 3: Left: Decision boundary for a two-layer perceptron with 4 hidden nodes. Right: Encoder problem - hidden = 3 - different η values - $\alpha = 0.9$

3.2.2 The encoder problem - dimensional reduction

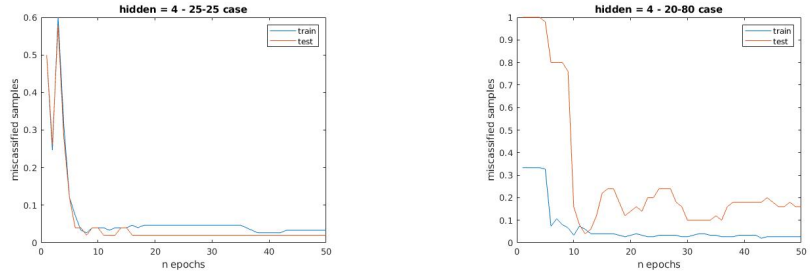

Figure 4: Percentage of MS for 4 hidden nodes. 1st: 25-25 case. 2nd: 20-80 case.

Figure 3 show that three nodes are enough for convergence, if η is sufficiently big. The final hidden weights encode the dimensional reduction of the input data, in fact every input entry is activated (positive weight) at least once in one of the hidden nodes.

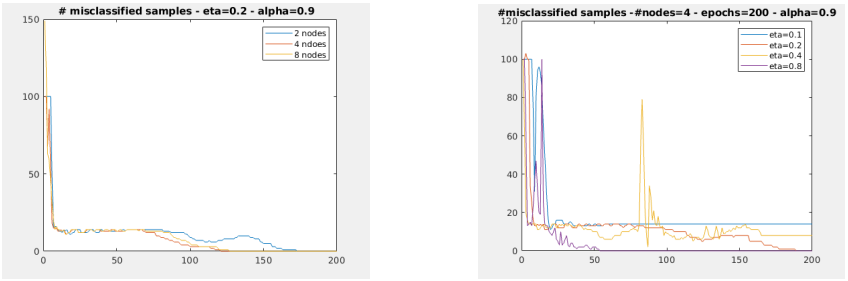

Figure 5: (x-axis = epochs number) First: MS for 2,4,8 hidden nodes, 200 maximum number of epochs. Second: MS with 4 hidden nodes, and $\eta = 0.1, 0.2, 0.4, 0.8$

3.2.3 Function approximation

Figure 6: Function approximation with the 2-layer perceptron. The number of hidden nodes are 2 and 25 respectively.

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

Table 3: 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).*

When the # hidden nodes are < 5, the network do not catch enough information to approximate the Gaussian function. When # nodes = 10-20, it approaches the bell shape. When # nodes $>>$ 20, the network takes in more information than needed. The best model is with 22 nodes. By minimizing the number of epochs, and increasing step-size η to 0.05, the convergence can be sped up slightly, without losing generalization.

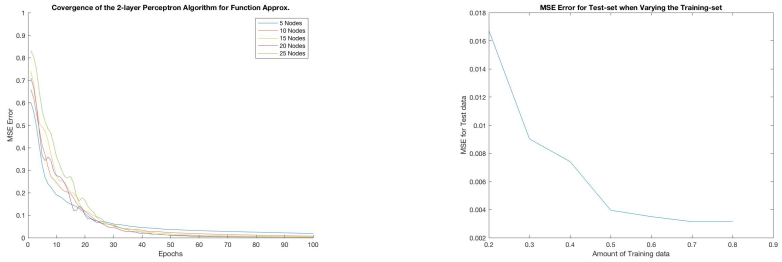

Figure 7: Averaged results over 10 iterations of 100 epochs each, $\eta = 0.009, \alpha = 0.9$. 1st: Performance of the 2-layer perceptron, varying # hidden nodes. 2nd: MSE of the model with 22 hidden nodes for all available data (training+validation).

4 Results and discussion - Part II *(ca.2 pages)*

4.1 Two-layer perceptron for time series prediction - model selection, regularisation and validation

We chose the network architecture **fitnet**, with the default early stopping criterion, **max_fail = 6**, the recommended algorithm **trainscg** and a built-in regularization method. We varied # hidden nodes h and strength of regularization r. For each configuration we performed 100 training sessions with independent addition of noise and initialization of weights.

configuration	mean val. mse	std val. mse
r=0.00, h=6	1.0841e-03	6.5759e-04
r=1.00, h=4	1.1427e-03	1.4321e-03
r=0.75, h=8	1.1432e-03	8.8037e-04

Table 4: Mean and standard deviation of validation mse for top three configurations.

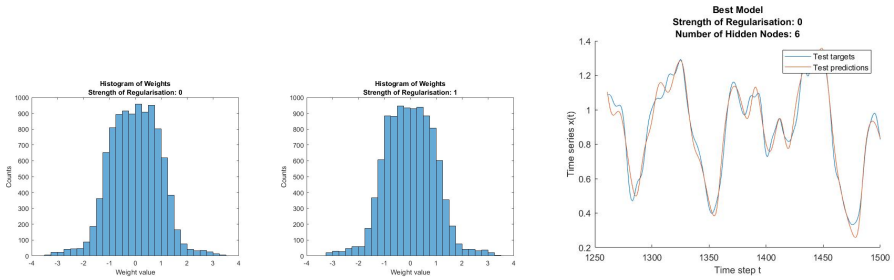

Figure 8: Histograms of weights for r=0 (left) and r=1 (middle). Right: Plot of the test predictions along with the true target values for one example network.

Figure 8 displays histograms of all weights generated for r=0 and r=1, which are disturbingly similar. Theoretically, stronger regularization should result in smaller weights and biases, and thus less overfitting. However, a two-layer perceptron and no-noise data may not suffer from overfitting, which can explain the lack of regularisation effect. Table 4 indicates that the best model is h=6 and r=0. Figure 8 also shows the performance of one example network with that specific configuration.

4.2 Comparison of two- and three-layer perceptron for noisy time series prediction

The test mse determined the best configurations in Table 5. Theory suggests that more noisy data benefits from regularisation, but this could not be validated. Intuitively, we would have expected the CPU time in Table 6 to increase with the number of hidden nodes. On the other hand, more neurons may also lead to overfitting quicker, which causes earlier stopping of the training based on the validation error.

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

Table 5: Mean and standard deviation of test mse over 100 networks for the best three-layer and former best two-layer configuration trained with noisy data.

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

Table 6: Computational cost of training in terms of CPU time for different network sizes (r=0.00 and sigma=0.03 for all networks).

5 Final remarks *(max 0.5 page)*

In our opinion the lab was well structured and we have been guided to a reasonable understanding of feed-forward networks.