Artificial Neural Networks and Deep Architectures, DD2437, Group 44

Short report on lab assignment 1

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

Daniel Wass, Patrik Ekman, Eric Hartmanis and Martin Koling

September 10, 2020

1 Main objectives and scope of the assignment

This report describes the work and results of assignment 1 of course DD2437 - Artificial Neural Networks and Deep Architectures. Our major goals in the assignment were to further extend our foundational knowledge and understanding of neural networks and basic learning dynamics. This has been done by implementing and evaluating single-layer perceptrons (SLP), and multi-layer perceptrons (MLP) of two and three layers, on different data-scenarios and settings as defined in the assignment description. Many problems arose from practical coding issues that were often time consuming, which made planning in order to present on the deadline more challenging.

2 Methods

We have programmed this assignment in Python 3 using the developer environment Visual Studio Code as it provides a convenient collaborative live sharing feature. The libraries numpy and matplotlib were used for computations and to visualize results. For part II, the Python libraries Tensorflow and Keras were used to construct the three-layer networks. Mean square error was used as the main performance metric throughout, although accuracy was also used for classification tasks.

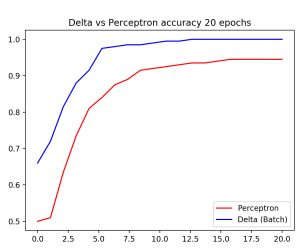
3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

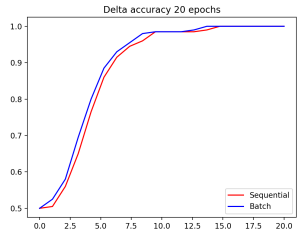
For the linearly separated data and $\eta = 0.01$ the weights of the single-layer perceptron (SLP) converged after 20 epochs, but as η increased, the number of epochs required decreased and vice versa. The Delta rule instead, implemented with batch learning, converged faster even with lower η, i.e. required fewer epochs. In *Fig 1. a* with $\eta = 0.001$, it can be seen that the learning rate is too low for perceptron to correctly separate all samples whereas the Delta rule is sufficiently trained after 13 epochs.

Delta rule with sequential learning was slightly more sensitive than batch learning to random initialization, but both needed more epochs until convergence, see *Fig 1. b, c.* Both classified all samples correctly after 13 epochs, W initialized from a zero mean, unit variance normal distribution and $\eta = 0.001$, see *Fig 1. b.* For higher learning rates, e.g. $\eta = 0.005$, the batch learning showed an undesirable behaviour as the iterative updates ΔW became too large. With these large changes of W, the decision boundary appeared to jump back and forth 180 degrees, oscillating between 100 and 0 percent accuracy. The sequential learning algorithm was thus less sensitive to higher learning rates.

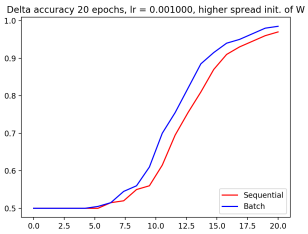
Without bias, the delta rule got "stuck" at the origin and could thus only correctly classify data samples that a straight line going through the origin could separate, as the decision boundary could not be shifted in the xy-plane.



(a) Delta (batch) vs perceptron accuracy comparison.



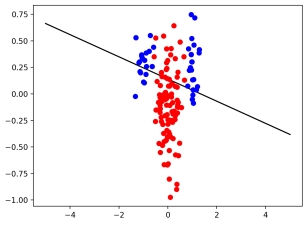
(b) Delta rule accuracy comparison.



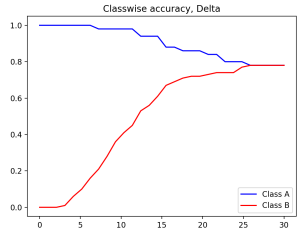
(c) Delta accuracy comparison, but with higher spread of W-initialization.

Figure 1: Learning curves for perceptron and delta rule after 20 epochs with $\eta = 0.001$

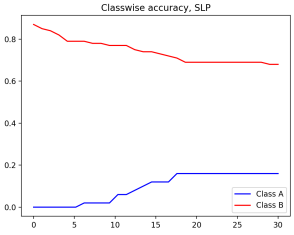
Shifting the mean of the data-generating distributions to make the data linearly non-separable, and comparing SLP and Delta rule (batch), showed that both algorithms performed similar when (in vain) trying to linearly separate the data if their parameters are individually tuned. Removing datapoints of the two classes symmetrically do not change that behaviour, but the learning curve became flatter. However, asymmetric deletion of points lead to Delta rule outperforming the SLP and the classwise accuracy implies that SLP is more sensitive to unbalanced data - see Fig 2.



(a) Delta rule.



(b) Classwise acc (delta).



(c) Classwise acc (SLP).

Figure 2: Linearly non-separable data, according to the given distribution parameters of the assignment, $\eta = 0.001$.

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

The number of nodes needed to perfectly separate the data varied with the random initialization of data, and for some random seeds it was non-separable for the two-layer perceptron (TLP). Its behaviour was somewhat in line with rules of thumb for the desired number of nodes to be found on the web - e.g. to have no more nodes than the input features. In most cases one node was sufficient for optimal classification in terms of MSE and accuracy, but two nodes tended to generate a more reasonable decision boundary. When increasing the number of nodes the performance eventually decreased, see Fig 3.

The MSE and accuracy curves comparing the network's performance on training and validation data (scenario 4) can be seen in figure 4. It is evident that the network requires a number of epochs (about 250) before learning the actual underlying data pattern being shared by the training and validation data. This is also where the plots are dissimilar - while the training MSE decreases quickly from the start, the validation MSE increases from the start - until the underlying pattern is actually learnt, which is when the MSE curves become very similar in terms of convergence. Figure 4c shows the learnt decision boundary

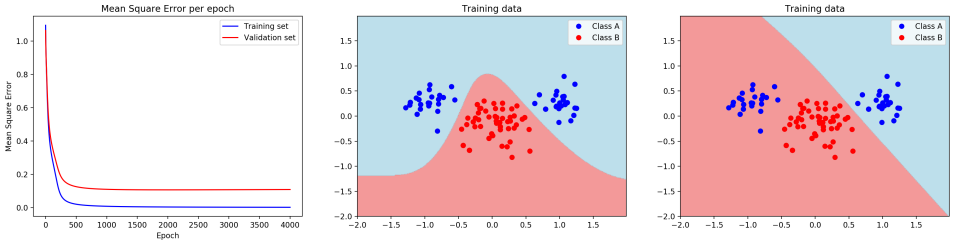
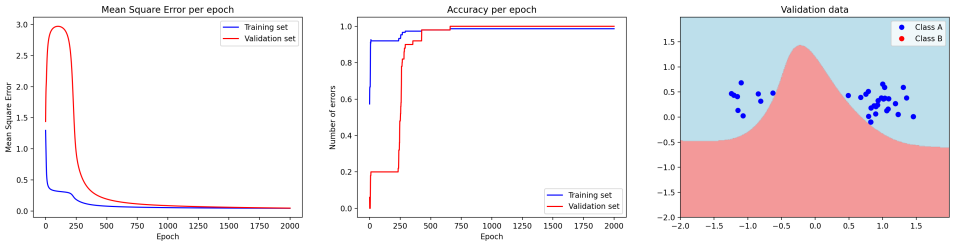


Figure 3: MSE for 2 nodes and decision boundaries for 2 and 100 nodes respectively. Scenario 1.

and the validation data, indicating that the network generalises well with regards to the validation data. However, the fourth listed scenario for sub-sampling (scenario 4) seems to be the hardest pattern to learn. With the same configurations and the other scenarios, the learning curves of the two data sets followed each other more smoothly during the whole training period as the underlying pattern was understood faster. Using more hidden nodes and scenario 4, the learning period to understand the underlying pattern were shorter and the curves coincided after a fewer number of epochs - up until a certain level as explained in the previous section.



(a) MSE *(b) Accuracy* *(c) Decision boundary*

Figure 4: MSE and accuracy curves (a and b), decision boundary and validation data (c). 2 layer perceptron, 3 hidden nodes. Scenario 4.

3.2.2 The encoder problem

The network always converged when being trained for the encoder problem, i.e. mapped the inputs to themselves. However the learning time deviated when initiating the weights differently, as well as when using different learning rates.

The activations of the hidden layer corresponding to the input patterns can be seen in the top part of table 1 and its binary representation in the bottom part. Every column represents the ”3 dimensional memory” for one of the input data points originally having dimension 8. Along with the learnt weights, which after the learning process are fixed, one such column should thus be able to recreate a data point of dimension 8 in terms of 1s and -1s. The fact that the number of hidden nodes are three is thus crucial, enabling 8 different combinations to be stored, i.e. all the binary numbers/combinations from 0 (000) to 7 (111), but in this case 8 combinations of 1s and -1s. It thus becomes evident that our learnt weights along with the compact representation of the 8 dimensional data makes it possible for the positive 1 to be in 8 different places of the 8 dimensional data. This is what the internal code represents and is how the encoder can learn a representation/encoding of the data.

If the size of the hidden layer would be 2 instead of 3 only 4 combinations could be stored, i.e. from 0 (00) to 3 (11). This means that we could only be sure that 4 of the 8 dimensions would be correctly stored in the hidden layer (the compressed data). In fact, it is required $log_2 8$ nodes, which equals 3.

Autoencoders could serve as a solution when dimensionality reduction is desirable, such as for e.g. image compression.

0.9993	0.5222	0.9978	-0.6277	-0.9989	-0.6411	0.5238	-0.9995
0.9999	0.9999	0.9999	0.9998	0.9999	0.9994	0.9999	0.9997
-0.4109	-0.9984	0.6632	0.9973	0.5581	-0.9986	0.9994	-0.5105
1	1	1	0	0	0	1	0
1	1	1	1	1	1	1	1
0	0	1	1	1	0	1	0

Table 1: Example of activations of the hidden layers corresponding to the input patterns and their respective binary representation.

3.2.3 Function approximation

When having very few nodes in the hidden layer, the function approximation becomes insufficient and the visual representation cannot be likened with the Gaussian function. When increasing the number of nodes however, a more accurate approximation can be observed. After increasing the number of nodes step wise, an accurate representation could be observed using as few as five nodes (after very many epochs, e.g. 1000). After that, continuing to increase the number of nodes did not generate any significant differences in the visual representation. However, the model converges faster and the MSE decreases with higher numbers of nodes, and after 100 epochs the models with 10 and more nodes performed the best, see Fig 5. Following the MSE results, the trend appears to be that the more nodes, the more is the optimal split of data weighted towards validation. For 10 nodes the lowest errors were generated from a train/validation-distribution around 70/30, whereas more towards 40/60 for 25 nodes.

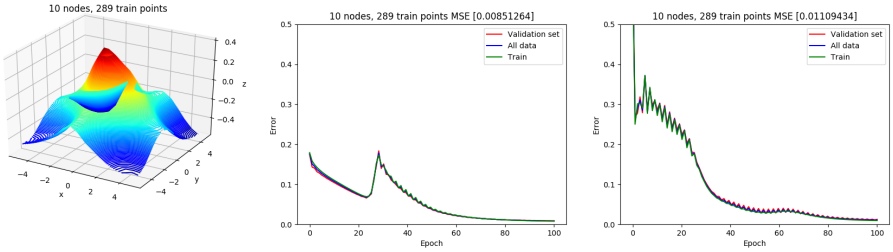
Rate of training data

Nodes	0.91	0.82	0.73	0.66	0.58	0.51	0.44	0.38	0.33	0.27	0.23	0.18	0.15	0.11	0.08	0.06
1	0,0670	0,0669	0,0664	0,0655	0,0652	0,0643	0,0640	0,0642	0,0657	0,0668	0,0682	0,0706	0,0766	0,0835	0,0840	0,0815
4	0,0365	0,0383	0,0392	0,0421	0,0449	0,0456	0,0464	0,0476	0,0481	0,0492	0,0517	0,0546	0,0573	0,0662	0,0772	0,0869
7	0,0427	0,0432	0,0425	0,0436	0,0500	0,0536	0,0596	0,0766	0,0797	0,0915	0,1069	0,1327	0,1932	0,4510	0,8420	1,0120
10	0,0281	0,0128	0,0099	0,0085	0,0104	0,0110	0,0143	0,0181	0,0221	0,0320	0,0449	0,0668	0,0860	0,0918	0,1004	0,1125
13	0,0308	0,0259	0,0256	0,0260	0,0362	0,0420	0,0492	0,0581	0,0658	0,0891	0,1061	0,1139	0,1233	0,1221	0,1303	0,1225
16	0,0221	0,0213	0,0193	0,0199	0,0184	0,0156	0,0115	0,0106	0,0120	0,0205	0,0186	0,0278	0,0457	0,0844	0,0944	0,1287
19	0,0218	0,0241	0,0197	0,0198	0,0233	0,0268	0,0275	0,0294	0,0255	0,0224	0,0166	0,0137	0,0156	0,0185	0,0240	0,0320
22	0,0372	0,0430	0,0420	0,0426	0,0384	0,0410	0,0289	0,0341	0,0373	0,0626	0,0819	0,1013	0,1349	0,1409	0,1538	0,1582
25	0,0466	0,0394	0,0306	0,0258	0,0258	0,0186	0,0125	0,0108	0,0116	0,0144	0,0188	0,0236	0,0260	0,0316	0,0469	0,0679

MSE

Figure 5: MSE of the function approximation with different node and data-ratio setups after 100 epochs. Mean of five runs, $\eta = 0.001$.

One measure for tweaking the best model for faster convergence is to increase the η. The experiments described in the previous section was performed with $\eta = 0.001$. With larger η, convergence is faster until a tipping point, where learning rate is too high and the learning curve becomes oscillating. In Fig 6c), with $\eta = 0.0035$, the algorithm converges after around 50 epochs, but to the cost of MSE compared to Fig 6b) where η was 0.001. Generalization was still good, but somewhat worse than the model with $\eta = 0.001$, which can be seen in the larger distance between the learning curves for validation and train (validation was for this experiment not used during training, and thus treated as a test dataset).



(a) Approximation of the function. *(b) MSE curve, $\eta = 0.001$.* *(c) MSE curve, $\eta = 0.0035$.*

Figure 6: 10 nodes model on 66/34 train/val-ratio after 100 epochs.

4 Results and discussion - Part II

4.1 Three-layer perceptron for noisy time series prediction

The following experiments, trained and tested on the Mackey Glass chaotic time series were performed with five nodes in the first hidden layer, shifting number of nodes in the second layer and L2 regularisation as the regularisation method of choice. Early stopping was implemented by comparing the development of the validation MSE throughout the training process - If the validation MSE did not improve in 15 epochs, the training process was stopped. Each test was performed five times of which the mean and standard deviation was computed.

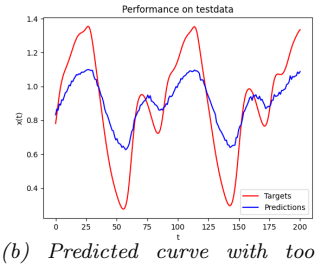
Figure 7a showcases the results of tests performed with a three-layer architecture with different amounts of nodes in the second hidden layer and with different levels of noise. With little noise, the differences in performance between models with different no. of nodes were low, i.e. increasing the no. of nodes had little effect, whereas for much noise there were significant improvements in MSE (train, validation and test) for a higher no. of nodes. Eight and seven nodes were significantly superior for the highest noise level, sampled from a zero-mean Gaussian with $\sigma = 0.18$.

Noise std

Nodes	0,03	0,09	0,18
1	-3,1583787	-1,4868289	-1,7658319
3	-2,9433637	-2,5513522	-1,6313266
5	-2,8572611	-3,0794861	-2,9020738
7	-2,8864723	-3,2589943	-2,9073648

Log MSE

(a) Mean Log MSE per nodes in layer 2 vs. std of noise generation.



(b) Predicted curve with too high regularisation.

Noise std

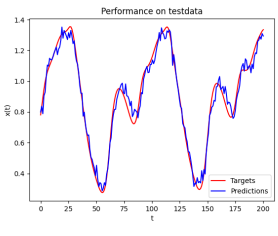
Reg. Rate	0,03	0,09	0,18
0	-6,379144815	-3,038491034	-2,70539204
0,0005	-4,877511775	-4,318888218	-3,706236621
0,001	-2,711848208	-4,678433198	-3,222975166
0,01	-3,274775872	-2,894374297	-2,903415817
0,05	-1,957991367	-3,041914703	-2,279641233
0,1	-2,716096314	-2,386096203	-2,436641928

Log MSE

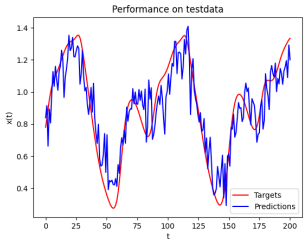
(c) Mean Log MSE per regularisation rate (scalar) and std of noise generation.

Figure 7: 750 epochs, 5 nodes in first hidden layer. Mean of five runs.

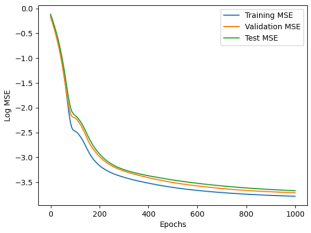
Generally, regularisation pushes the weights down, making the model less complex and the predicted curve more "smooth", especially improving generalisation on noisy data. Experiments performed on the seven-layer model showed that regularisation improved the model performance significantly for the very noisy data, whereas it had negative impact on performance on data with less noise. The regularisation term must thus be tuned carefully - if it is too large considering the noise level, the curve becomes too smooth and does thus not take enough variance into account (see Fig 7b), while if it is too small the curve becomes too wiggly, taking unwanted small differences (e.g. created by noise) into account and does thus not generalise as well.



(a) $\sigma = 0.03$.



(b) $\sigma = 0.18$.



(c) MSE learning curve

Figure 8: Three Layer Network, 7 nodes in second hidden layer

The best model, with 7 nodes, was evaluated further by trying to look for the best regularisation rate, as can be seen in figure 7c. Looking at the table in figure 7c it is evident that less noise requires a smaller regularisation rate, while more noise requires a larger regularisation rate. This makes sense, as more noise normally would increase the variance of the data and naturally make the predicted curve more "wiggly". For such a case, more

regularisation would thus be required to decrease the complexity of the model (decrease the size of the weights), smoothening the curve and decreasing the variance predicted by the model, ultimately increasing the model's generalisation capability. This is also how noise affects the generalisation performance, i.e. more noise increases the variance of the data, leading to increased difficulties when trying to train the model understanding the true underlying pattern of the data.

For the intermediate level of noise ($\sigma = 0.09$), our best three-layer model was the model with 7 nodes in the second hidden layer. Figure 8 showcases its prediction performance on the Mackey Glass data with $\sigma = 0.03$ (a) and $\sigma = 0.18$ (b), as well as the MSE learning curve (c). The best model was decided by comparing the validation data MSE of the different models. In a "real life-situation", the validation MSE serves as the best and only fair pointer towards which model performs the best on unseen data (generalisation) - the test data should be kept entirely out of the equation and should be viewed as unknown, and the training data is only used for training, hence entailing nothing of whether the model performs well on unseen data or not - merely of whether the model performs well on already seen data.

Yet another reason for using more nodes than just a few can be seen in figure 9. Figure 9a suggests that as the number of nodes increases, the convergence is more robust (standard deviation lower during a number of simulations), consistently performing well. Moreover, as can be seen in figure 9b, more regularisation seems to decrease the standard deviation. This goes in line with previous discussions, entailing that more regularisation smoothens the predicted curve, thus making the predictions more consistent over a number of simulations.

Noise std

Nodes	0,03	0,09	0,18
1	1,881	1,534	1,140
3	0,530	0,973	0,496
5	0,234	0,277	0,807
7	0,647	0,088	0,060

Std of MSE

(a) Standard deviation (log) of MSE

Noise std

Reg. Rate	0,03	0,09	0,18
0	1,3244	0,6249	0,5424
0,0005	1,8751	0,8063	0,3479
0,001	1,0096	0,5015	0,3370
0,01	0,2457	0,4036	0,2763
0,05	0,1547	0,4400	0,8045
0,1	0,8105	0,8488	0,1814

STD of MSE

(b) Standard deviation (log) of MSE

Figure 9: Standard deviation over five runs and different hyperparameters.

Adding more nodes and more hidden layers increases the computation cost for the learning algorithm. The main difference of adding layers and nodes comes from the matrix multiplications, and the linear increase, such as from activation functions, are thus assumed negligible. If n_i is the no. of nodes in layer i, where n_0 denotes the input dimension, k is the number of layers, d is the number of input datapoints and t is the number of epochs, then the complexity of the MLP is:

$$\mathcal{O}(d \cdot t \sum_{i=1}^{k} n_i \cdot n_{i-1}). \tag{1}$$

Adding one layer n_j thus increases the complexity by $\mathcal{O}(d \cdot t \cdot n_j \cdot n_{j-1})$. Adding x nodes to a layer n_j increases the complexity by $\mathcal{O}(d \cdot t \cdot (x \cdot n_{j-1} + x \cdot n_{j+1}))$.

5 Final remarks

This assignment has been good in the way the questions have been formed to promote analytical thinking - most questions required an extra thought from the initial one. We though, sometimes, experienced the lab instruction as quite unclear in what was expected to be answered on each point - some questions were formed on bullet points and some hidden in the text, and some of the instructions were even in the report template.

Most problems that we experienced were practical and not theoretical, which in one sense is good as we have challenged our coding skills, but on the compromise of challenging theoretical analysis.