Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1b

Learning with backpropagation and generalisation in multi-layer perceptrons

Stefan Christiansson, Rasmus Hammer and George Malki

9 februari 2022

1 Main objectives and scope of the assignment

Our main objectives for this assigment were:

- to design and implement the necessary building-blocks for a working MLP (*forward pass,backward pass and weight update*).
- to monitor the behaviour of learning across different task (*classification, function approximation and generalisation*).
- to gain a deeper understanding of the risks associated with back-propagation with the aim to minimise them.

The scope of this lab is concerned with the generalised delta rule for a two-layer perceptron. It is also of interest to study the capability of an MLP when it comes to classification, function approximation and generalization. The assignment was coded using python since it's more used within the Machine Learning community than the MATLAB recommendation.

2 Methods

For the first part of this two part lab, a two-layered MLP was implemented using batch mode and momentum for the weight update. We also used a bipolar representation (-1/1) for our data-set. For our performance measure we used two types of measures. The first measure was an MSE in accordance to the following formula:

$$MSE = ||(y-t)^2|| \tag{1}$$

where y is the predicted label and t is the target. The other measure was the classification error given by:

$$\begin{aligned} \textit{Classification error} &= ||\frac{|f(y)-t|}{2}|| \\ f &= \begin{cases} 1, & \text{if } y < 0 \\ 0, & \text{if } y = 0 \\ -1, & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

where y is the predicted label and t is the target.

In the second part of the lab an MLP was developed for chaotic time-series prediction. The MLP was tested on the Mackey-Glass time series, which was generated using:

$$x(t+1) = x(t) + \frac{0.2x(t-25)}{1+x^{10}(t-25)} - 0.1x(t)$$

First 1200 data point was generated, which was then divided into training, validation, and test subsets. This dataset was then used to test 4 different configurations of the MLP with the first hidden layer having 9, 3, 4, 5 nodes, and the second hidden layer having 6, 2, 4, 6 nodes respectively. An MLP with only two layers was also tested with the hidden layer having 9, 3, 4, 5 nodes. Early stopping was used with the validation subset.

Second zero-mean Gaussian noise was generated and added to the training data, to examine the effect of noise on the performance when training an MLP. The most sutable architecture from previously was picked and the nodes in the first layer was held constant, while the number of nodes in the second layer varied according to 6, 2, 4 nodes. The Gaussian noise also varied with parameter σ according to $\sigma = 0.05, 0.15$. Further, the weight regularization with parameter λ was experimented with using values $\lambda = 10^{-6}, 10^{-3}, 10^{-1}$. When training on the data with noise early stopping was not used. For training, both with and without noise, the MSE was used as a metric of the error on the validation and the test subsets and all metrics was averaged over 100 iterations. All hidden layers had random weight initialization. The libraries used during part II of the lab was tensorflow and keras, as well as sklearn for MSE.

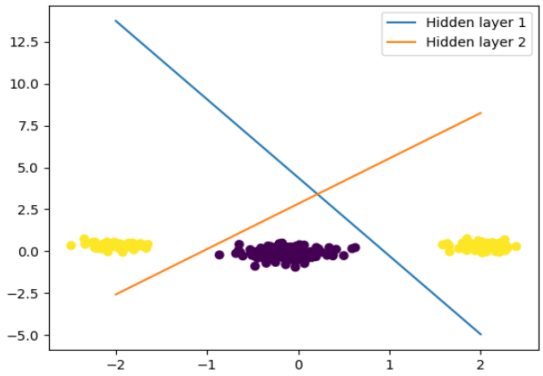
We used python as the programming language for both parts. Visual studio code was used as text-editor, and Git was used for code-sharing. For the plots we used the matplotlib library.

3 Results and discussion

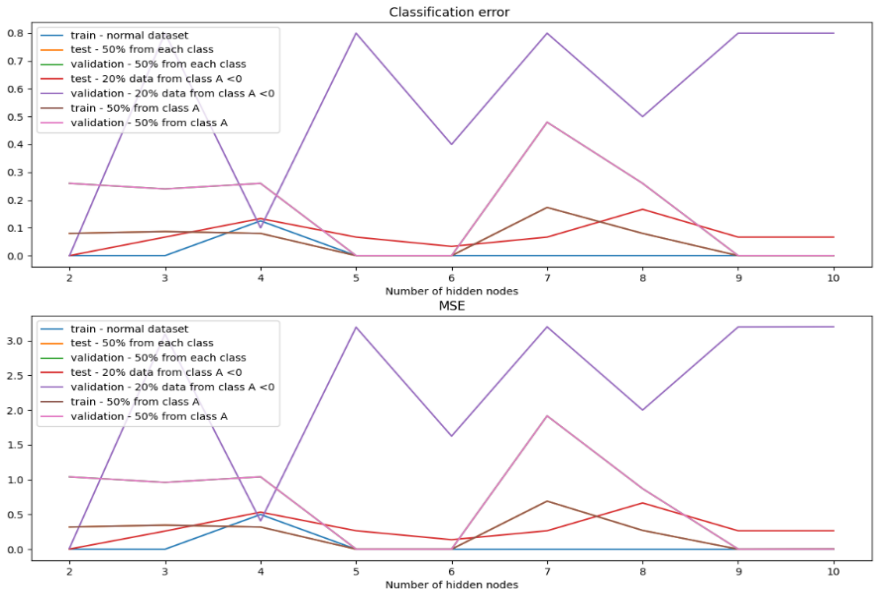
3.1 Classification and regression with a two-layer perceptron

3.1.1 Classification of linearly non-separable data

Our implementation discussed in the method section was indeed able to find a solution for non-linearly separable data as shown in figure 1. We also conducted an experiment where we varied the amount of hidden nodes in each layer for different epochs. For the representation of these results we chose 500 epochs, since it performed the best. The result for this is shown in figure 2. For both the classification error and the MSE we can see a trend where it starts high goes down and then goes up again, until it finally settles by going down. Judging by the dark-blue line from the plots in figure 2, 2-3 hidden nodes should on average suffice to separate all available data.



Figur 1: 2 layer MLP solution for non-linearly separable data using 2 hidden nodes, 100 epochs, and learning rate of 0.1



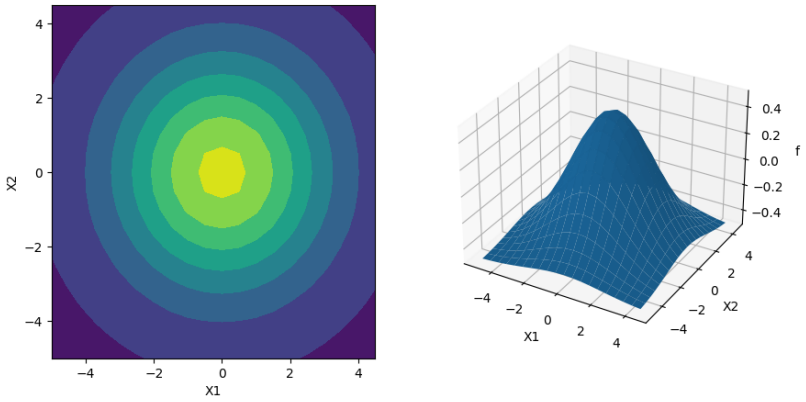
Figur 2: Effect of number of hidden nodes with a learning rate of 0.1 running for 500 epochs on non-linearly separable data.

We see a pattern in figure 2 for how MSE/classification error curves for training/validation sets compare for many of the samplings scenarios. We see that the error curves are higher for the validation sets than the training sets, which is to be expected since these validation sets are used to probe the generalisation performance of our model. This is the case for all of our curves including the dark-purple curve where the error curves are considerably higher for validation sets than training sets. This makes sense since in this case the data being validated on is more representative of the under-represented points of training data, since 80% of the points from the first class in validation is only 20% of the points

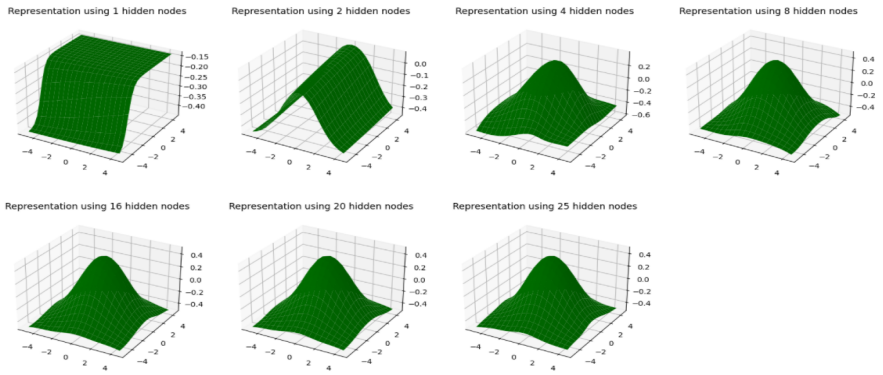
in training. Another thing that we could derive from our testing data is that increasing the number of hidden nodes, up to a point, increases the performance of our model. Too few hidden nodes could result in our model converging slower/ converging to a local-minima. Increasing the number of hidden nodes to 20+ resulted in error-curves that were hard to interpret. We expect the validation performance to be a bit more noisy using sequential learning, which in this case is called stochastic gradient descent, since we are updating the inner weights in a point by point case. What we mean by noisy is that the model might update its weights after using a new value to learn, which could result in a 'bad' update which would move us away from the local minima.

3.1.2 Function approximation

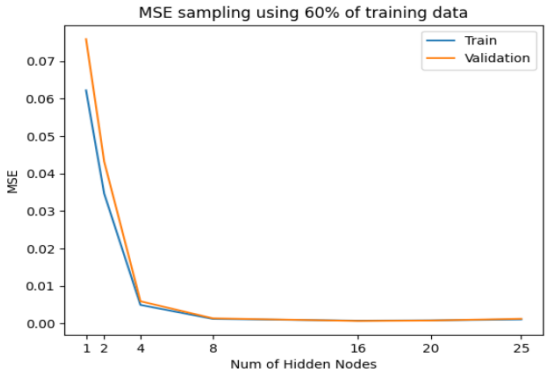
For this part we were asked to use our 2 layered MLP to approximate the bell Gauss function visualized in figure 3. Figure 4 shows the approximation our MLP achieved for different number of hidden nodes. As we can see from both the visual plot and the MSE comparison in figure 5, 8 hidden nodes seems to be the sweet spot for maximum performance, which is why we chose 8 hidden nodes for our model.



Figur 3: General 3D plot of bell shaped Gauss function



Figur 4: Visualization of approximated bell shaped Gauss function for different hidden nodes trained using our MLP network with learning rate of 0.1 and 100 epochs



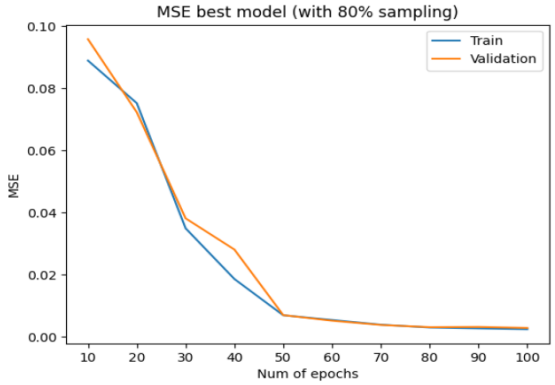
Figur 5: MSE of bell shaped Gauss function approximation for different hidden nodes on 60% of data

After finding the optimal amount of hidden nodes for our model we ran an experiment varying both the learning rate and number of epochs to try and find the best combination of learning rate and epochs to minimize the MSE. The results from this experiment are shown in table 1. As we can see from the results, the learning rate is on the higher side, which makes sense since we had implemented the momentum update for our weights. These results may not be the most optimal speed wise since the number of epochs are quite high which leads to slower convergence, but they were the most precise given the learning rates and epochs we tested.

Best learning rate	Best epochs	MSE
0.1	5000	8.56e-05

Tabell 1: Best parameters for 8 hidden node model (best model) based on the lowest MSE

Finally, after optimizing the number of hidden nodes and hyper parameters, we tried to optimize convergence speed by varying the number of epochs between 10 - 100 and comparing MSE. The results from this are shown in figure 6. The results show that we don't need more than around 80 epochs before the MSE gives no gain in performance.



Figur 6: MSE for best model varying the amount of epochs

3.2 Multi-layer perceptron for time series prediction

Table 2 shows the MSE for each configuration of the network, averaged over 100 iterations for each configuration. For all of the configurations the learning rate was 0.01 and the weight decay was 0.9. Random weight initialization was used. From the table it can be seen that the most robust model was the model with 5 nodes in the first layer and 6 in the second layer. The worst performing model was the one with 3 nodes in the first layer and 2 in the second.

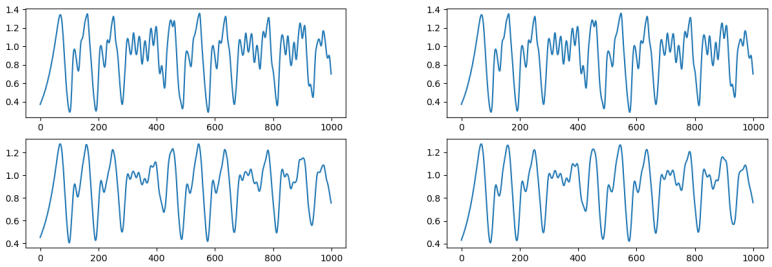
	Average MSE
$l1 = 9$, $l2 = 6$	0.412
$l1 = 3$ $l2 = 2$	0.499
$l1 = 4$ $l2 = 4$	0.442
$l1 = 5$ $l2 = 6$	0.366

Tabell 2: MSE for the different network configurations.

Figure 7a shows the comparison of the real data for the time series and the prediction made on the test set, when using the best configuration of the model. 7b shows the same thing but for the worst configuration. It is difficult to see if there is any difference in performance just by comparing the graphs, but by looking in Table 3, we are able to see that the first model performs better on average (over 100 iterations) on the test set.

	Average MSE
l1 = 5, l2 = 6	0.3550
l1 = 3 l2 = 2	0.4428

Tabell 3: MSE for the different network configurations on the test set.



(a) Real data (upper), and prediction (lower) for l1 = 5, l2 = 6.

(b) Real data (upper), and prediction (lower) for l1 = 3, l2 = 2.

Figur 7: Comparison of time series with real data and prediction made on the test set

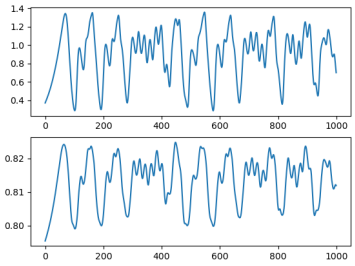
Table 4 shows the average MSE on the test-set for different configurations of the model with 100 points of noise added to the dataset. The results indicate overall lower MSE when training with added noise, than for training on the data-set without added noise. Table 5 shows the same thing but for 1000 added points of noise. The results from table 5 does not show a significant decrease in MSE when compared to training on the regular data-set, and it is therefore hard to draw any conclusions from this. Figure 8a show the prediction for the seemingly best model ($\lambda = 10^{-3}$, l1 = 5, l2 = 6), compared to the real data. 8b show the prediction for the model with ($\lambda = 10^{-6}$, l1 = 5, l2 = 6), on data with 10 000 points of added noise. As can be seen by the graph the model seem to overfit the noise, which can also be seen by looking at the average MSE for this case which is 0.749. When looking at the validation error when training with noise, it was also found that the average validation error is higher when training on data with noise, and it also seems more volatile.

	$\lambda = 10^{-6}$	$\lambda = 10^{-3}$	$\lambda = 10^{-1}$
l1 = 5, l2 = 6	0.236	0.156	0.157
l1 = 5 l2 = 2	0.236	0.317	0.478
l1 = 5 l2 = 4	0.235	0.236	0.157

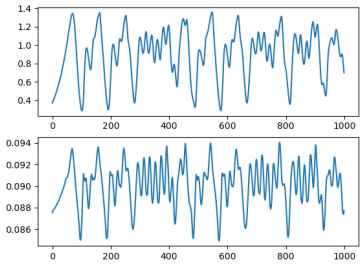
Tabell 4: Average MSE for the different network configurations with $\lambda = 1e-6, 1e-3, 1e-1$, $\sigma = 0.05$ and 100 points of noise.

	$\lambda = 10^{-6}$	$\lambda = 10^{-3}$	$\lambda = 10^{-1}$
l1 = 5, l2 = 6	0.318	0.387	0.380
l1 = 5 l2 = 2	0.449	0.386	0.448
l1 = 5 l2 = 4	0.383	0.326	0.257

Tabell 5: Average MSE for the different network configurations with $\lambda = 10^{-6}, 10^{-3}, 10^{-1}$, $\sigma = 0.05$ and 1000 points of noise.



(a) Prediction for model with $\lambda = 1e-3$ and l1 = 5, l2 = 6 on data with 100 points of noise with $\sigma = 0.05$.



(b) Prediction for model with $\lambda = 1e-6$ and l1 = 5, l2 = 6 on data with 10 000 points of noise with $\sigma = 0.05$.

Figur 8: Comparison of time series with training data and prediction made on the test set

Conclusions are that training with a low amount of added noise from a Gaussian increase the performance of the model on the test set, however it also increases validation error. As the quantity of noise grows, the model starts to overfit the noise, and therefore it is most likely beneficial to decrease λ, increasing the regularization, to avoid overfitting the weights to the noise.

4 Final remarks

We found lab1b to be a bit more interesting than lab1a, albeit a lot more complex and harder to implement. We gained knowledge about how to utilise MLPs, with different amounts of hidden layers and nodes, to find solutions for non-linearly separable data-sets using generalised delta-rule. We also learned how to utilise the training data more effectively by dividing it into training and validation subsets, which should improve the generalisation performance since the model is less prone to over-fit. Further the lab gave us insight in how big of an impact the hyper parameters and number of hidden nodes has on both convergence speed and accuracy. Simply by adjusting the amount of hidden nodes and learning rate in accordance to MSE we could decrease the amount of epochs by hundredfolds. A problem that we faced was deciding how to present data without risking information-overload and crossing the hard page-limit.