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

Etienne Jodry, Frano Rajic, Ivan Stresec

September 9, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to implement and understand the perceptron, delta, and generalised delta learning rules
- to implement single- and multi-layer networks and observe their performance in relation to various hyperparameters
- to apply networks in solving problems of classification, encoding, and regression
- to use multi-layered networks in time-series predictions and observe the influence of regularization and noisy data sets on performance

In the first part of the lab, the focus was on the Delta rule and the generalized Delta rule for two-layer perceptrons. We designed neural networks for classification, data compression, and function approximation.

In the second part of the lab assignment we worked with multi-layer perceptrons to predict the Mackey-Glass time series. In this task we designed and evaluated the neural networks with the ambition of delivering a robust solution with good generalization capabilities.

2 Methods

We used Python 3.8 with the PyCharm IDE. Python libraries used include numpy, matplotlib, and TensorFlow.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

On linearly separable data we reached 100% accuracy with both the perceptron (PR) and the delta learning rule (DR), in either batch or sequential mode. PR converges faster than DR, as it takes more time for the latter to minimize loss when maximal accuracy is already achieved. For DR, the best learning rate for batch mode is around $\eta = 0.002$ and above that value it tends to diverge. Sequential converges faster with a learning rate on the whole range between 0.0001 and 0.2, although it has a much higher loss. Convergence speed is very sensitive on the initial weights for sequential as opposed to batch learning, which is not.

Table 1: Perceptron learning rule

Learning rate	0.001	0.005	0.01	0.1	0.25	1	10
Accuracy - mean	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Convergence epoch - mean	29.22	8.26	3.45	1.91	1.90	1.93	1.87
Convergence epoch - std	27.84	6.80	2.14	0.38	0.30	0.32	0.42

Table 2: Batch delta learning rule

Learning rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.01
Accuracy - mean	100.00%	100.00%	100.00%	diverged	diverged	diverged	diverged
Convergence epoch - mean	1,081.97	110.99	117.69	diverged	diverged	diverged	diverged
Convergence epoch - std	245.30	25.02	16.39	diverged	diverged	diverged	diverged
MSE Loss - mean	5.81E-02	5.81E-02	5.81E-02	diverged	diverged	diverged	diverged
MSE Loss - std	3.54E-07	1.51E-07	7.67E-08	diverged	diverged	diverged	diverged

Table 3: Sequential delta learning rule

Learning rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.01
Accuracy - mean	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Convergence epoch - mean	1,009.25	108.69	70.43	58.63	51.19	61.08	49.02
Convergence epoch - std	250.40	34.33	36.30	29.41	31.78	39.73	33.77
MSE Loss - mean	6.24E+00	5.83E+00	5.83E+00	5.82E+00	5.82E+00	5.81E+00	5.84E+00
MSE Loss - std	4.24E+00	4.09E-03	1.28E-02	2.39E-02	3.23E-02	3.67E-02	4.83E-02

Bias represents how distant the linear separator is from the origin; DR with no bias weights can result in successful separation of classes if a separating line can be drawn through the origin, and unsuccessful if not.
We subsampled the non-linearly separable data in various ways to see the effects on performances and decision boundaries. With uneven class representation, the drawn line penalizes the minority class. We came across a case where one part of the non-representative class was entirely misclassified by the model. In all subsampling cases, we noticed that measured accuracy on the original dataset was lower, which means that both unrepresentative sample distribution and uneven class representation decrease the generalization capability of the trained model.

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

Modifying the number of hidden nodes showed that at least 3 nodes are needed for the prepared two-class inseparable dataset for an accuracy of 100%, which is to be expected having in mind the class distribution: one sigmoid function gives a line, two sigmoid functions give two lines, so the two cannot create a complex enough boundary.

We created 4 subsets from the inseparable dataset (as described in section 3.1) and divided the data into train and validation sets. Resulting decision boundaries for all 4 subsets are shown in Figure 1.

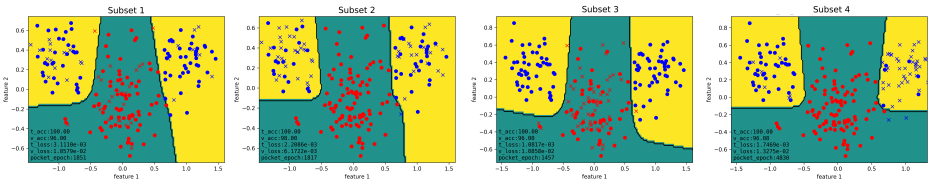


Figure 1: Multi layer perceptron results for all 4 subsets

The training error is consistently lower than the validation error, as shown in Figure 2. Increasing the number of nodes lowers the training error as the network can adapt better to the data while the validation error rises as the network loses its generalisation capability. This rise in dissimilarity also depends on the dataset we use.

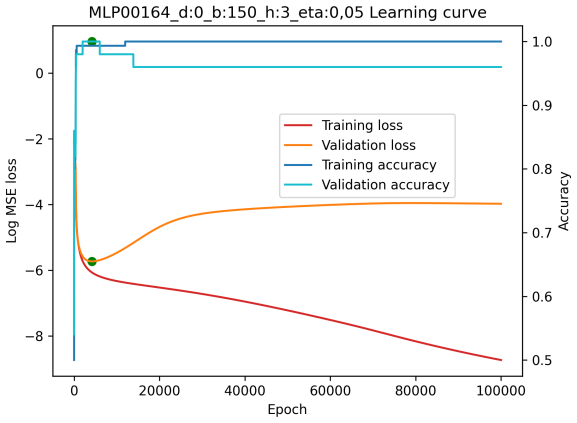


Figure 2: Learning curve of Multi layer perceptron with 3 nodes in hidden layer. Batch learning with learning rate 0.05 was used on data subset 1.

Due to its noise, sequential learning is more robust to local minima and can

therefore find a better solution than batch learning, lowering both training and validation errors. However, the noise can also negatively influence the performance of the network. In our experiments, there was no noticeable difference in performance with respect to using batch and sequential learning; they both achieved high validation accuracy, but sequential learning did so in a lower number of epochs.

3.2.2 The encoder problem

Given appropriate hyperparameters, an 8-3-8 network will always converge and map inputs to its three hidden nodes for the simple problem of mapping sparse "one out of 8" patterns. By looking into the hidden layer, we noticed the network used a binary encoding pattern to map $2^3 = 8$ input patterns. The weights of the first layer were positive for those inputs that should result in a 1 in the hidden layer and negative for that which resulted in a 0.

When using an 8-2-8 architecture, we noticed that the network still successfully maps all inputs, but now the hidden layer representation was different from a simple binary code. The representation is less interpretable but more efficient.

As shown, multi-layer perceptron networks can be used to compress data. Given various inputs, the network learns a hidden and simpler representation thereof, and thus needs less memory to store the same data.

3.2.3 Function approximation

Multi-layer perceptrons are known for their ability to approximate an arbitrary continuous function. Here, we studied how a two-layer perceptron network approximated the bell shaped Gauss function.

By varying the number of nodes in the hidden layer from 1 to 25 as shown in Figure 3, we tested all models for validation loss on a 60% data split of 1600 data points with different learning rates. We noticed that a low number of nodes, below 8, is not complex enough to lower the validation error as more complex models are capable of. We found that the best performance with respect to validation error is achieved for an architecture with 14 nodes in the hidden layer. Having more nodes than 20 starts to increase the validation loss again.

With the best model found, we run experiments with a varying number of training examples, with data splits from 80% down to 20%. We did not notice much difference in the results, which could be caused by the fact that the network is simple enough not to overfit.

The convergence of the best model can be sped up using momentum, however this introduces yet another hyperparameter to tune.

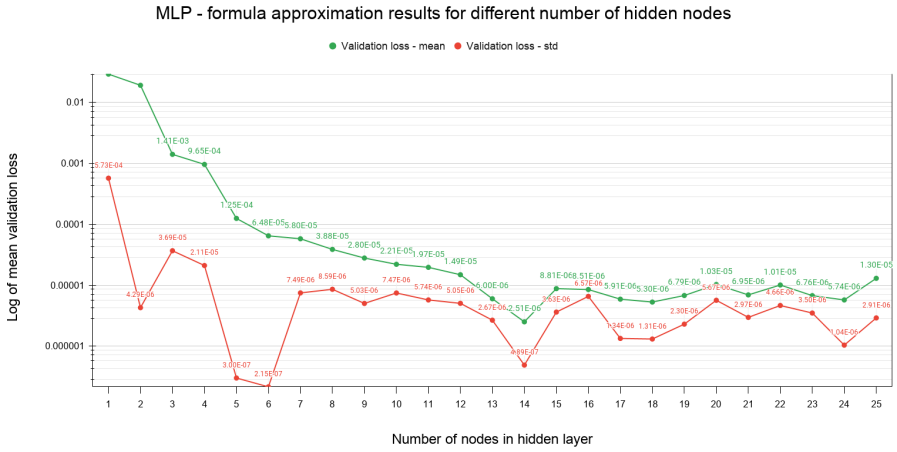


Figure 3: Graph of validation loss in relation to number of nodes in hidden layer

4 Results and discussion - Part II

We generated the Mackey-Glass time series using starting conditions and formula given in the assignment, then we split it into three consecutive non-overlapping sets: train, validation and test (respectively 800, 200, 200 samples). Weight decay (L2 norm) regularization is used. Max number of epochs is 20000 and we implemented early stopping such that if there is no improvement of at least 10^{-5} in the validation mean square error (MSE) for 50 epochs, we stop the learning.

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

Using learning rate of 0.1, as well as a medium regularization coefficient $\lambda = 0.1$, several two-layer perceptrons were tested, the results of which we can see in figure 4. Each configuration was tested 10 times for larger certainty. We opted for the networks with 8 nodes in the hidden layer as those show the best results on the test set.

The impact of regularization on this network is less important that we would have expected. As seen in Figure 5 few weights are drawn closer to zero. Our model might be too simple to overfit, plus we address the problem with early stopping.

Finally, Figure 7 shows an example of the learning process and the outputs of our selected network with 8 nodes in the hidden layer, a learning rate of 0.15 (which showed better results but slower convergence than 0.1) and an L2 weight

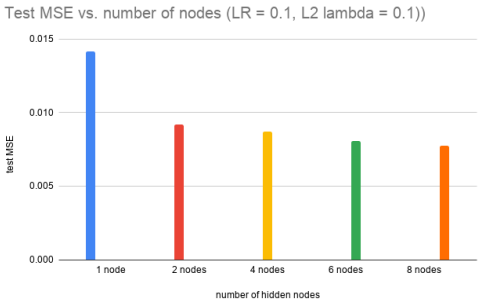


Figure 4: Test MSE for various network configurations

decay with $\lambda = 0.01$. Those networks had an MSE of around 0.0036 on the test set which demonstrates their good capability of generalisation.

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

For this section zero-mean Gaussian noise was added to all target values. Kept hyper-parameters for training were $\eta = 0.15$ and $\lambda = 0.1$

We have tested a two-layer and a three-layer network with 8 nodes in each hidden layer and these have shown that training of the three-layer network takes about 50% longer which is to be expected considering the sequential implications of the backpropagation algorithm.

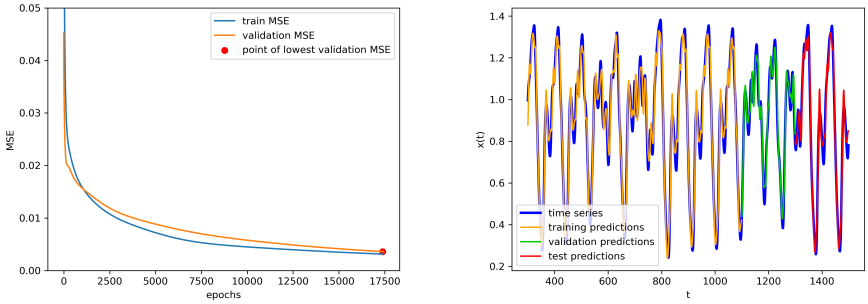
The error on the all sets is larger when working with noisy data. This is to be expected as outputs are now more chaotic and less true to the underlying function; the irreducible error itself increases.

As we add noise to the target values the network becomes more likely to overfit by learning noise instead of the underlying function. Furthermore, we are using a more complex model which can also cause overfitting. Thus, there is incentive to increase the strength of regularization methods to prevent these effects. $\lambda = 0.1$

(a) No regularization

(b) L2 regularization $\lambda = 0.1$

Figure 5: Weight distributions

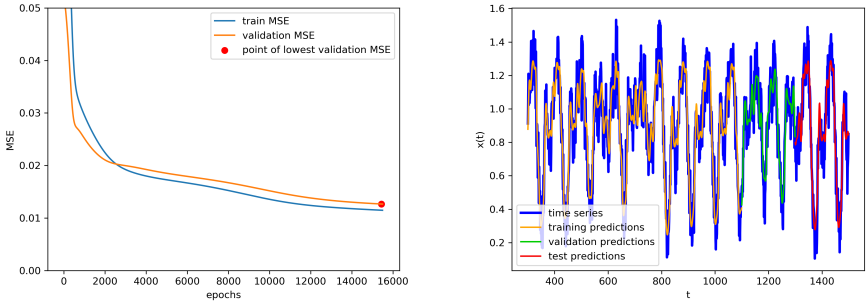


(a) Learning process (b) Predictions after training

Figure 6: Learning process

showed the best results, 10 times higher than the λ used with noiseless data.

A network with 8 nodes in the first and 4 hidden nodes in the second hidden layer showed the best results for a noisy series with $\sigma = 0.09$ with a mean test MSE of 0.0208. The network with just one hidden layer and 8 nodes had a mean test MSE of 0.0201 which shows that, in this case, the network didn't benefit from an extra hidden layer.



(a) Learning process (b) Noisy time series and all predictions

Figure 7: Typical performance of a network for 0.09 noise sigma

5 Final remarks

The lab was quite difficult in terms of creating all the needed software support and running various tests, but overall very useful for observing the effects of various hyperparameters and methods used in artificial neural network. It was insightful as a practical demonstration of theoretical concepts and showed that artificial neural networks actually do work, and do so well in both regression and classification problems.