Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1

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

Qiu Danny, Wester Philip and Samaille Noe

September 5, 2019

1 Main objectives and scope of the assignment

Our major goals in the assignment were to implement our first **neural networks algorithms** and get a proper understanding of **learning and generalisation** capabilities of **single-layer and multi-layer perceptron networks**.

2 Methods

The programming language used in this lab is **Python**. For the first part, **numpy** library is used for matrix creation and operations. For the second part we used **keras** (python library for machine learning using Tensorflow).

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

3.1.1 Perceptron classification of linearly separable data

Given the 100 points of class A and class B each, we use the ratio of well-classified elements as evaluation. Since the training (and number of epochs) is sensitive to weights initialisation, results were made averaging 50 trainings. A bias node was added, otherwise the perceptron only classifies data separable by a line passing through (0, 0).

1. Perceptron and sequential delta rule learning have similar results, but sequential takes a little more computanional time. Our best results have been measured with a 0.006 learning rate for 4.0 epochs (sd 4.4) for perceptron learning and 3.7 epochs (sd 3.7) for sequential.
2. Delta rule with batch mode has a slower convergence. The fast computations compensates the greater number of epochs. It takes 25 epochs (sd 34) with a learning rate of 0.001.

3.1.2 Perceptron support of non-linearly separable data

During training, the perceptron is updating weights looking for a solution that does not exist until he reaches the maximum number of iterations. Delta rule will always miss-classify points, but it still minimizes the error most of the time.

Using the dataset from the lab, after training about 80% of the data are correctly classified.

1. When A and B are represented in same proportion, training has the best generalisation
2. When one class is less represented, this class is less generalised
3. When one class training set is not representative of its real distribution, subsets of that class are not generalised

Results show how even classes and representative distribution representation are important for well generalisation.

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of non-linearly separable data

Most of the time both the validation and test sets had similar mean squared error. On some more rare occasions the network seemed to learn noise. In Figure 1 both training and validation mean squared error improve until $\sim$ 60th epoch. While both sets reached a mean square error of 0, the validation set then increased its mean square error by a bit. This is supposedly because the network has learned noise from the training set.

While the results greatly varies between different test runs when the number of hidden nodes is low ($\sim$ 20), the validation set mean squared error tends to deviate even more from the training set mean squared error when the hidden nodes are many (>200).

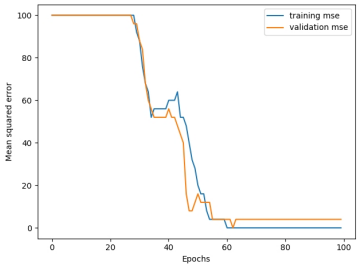


Figure 1: Mean squared error of validation and training set over 100 epochs

3.2.2 The encoder problem

Our $8-3-8$ auto-encoder network does not always converge, but we empirically noticed that it converges more often when initialising weights with values close to zero, and by increasing our learning rate η.

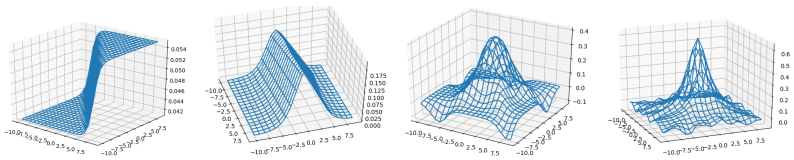
The internal code, represented by the hidden layer activations, looks like a 3 bits encoding (if we replace the -1s by zeros) of the 8 input patterns.

If we reduce the number of hidden nodes from 3 to 2, the network cannot find a 2 bits encoding of the inputs, so it never finds the correct outputs.

Such auto-encoders can serve for **dimensionality reduction** or **data compression**.

3.2.3 Function approximation

Mean squared error is used for evaluation on an input data of 100 (x, y) points.



(a) 1 hidden node (b) 2 hidden nodes (c) 3 hidden nodes (d) 50 hidden nodes

Figure 2: Function approximation depending on number of nodes in the hidden layer

- 1 hidden node: approximation is a plane or step function
- 2 hidden nodes: uni-dimensional approximation

- 3 hidden nodes and more: bi-dimensional approximation. The more hidden nodes, the more complex is the regression so the better the approximation. Above 25 nodes, there are no significant improvements.

Visually, the best model is the less complex while maintaining good approximation. A network with 20 hidden nodes is used for generalisation.

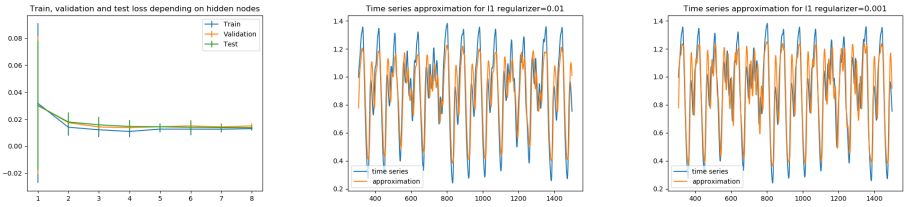
Training	Validation	Training error	Validation error	Global error
0.8	0.2	5.7e−4 (sd 1.6e−4)	8.5e−4 (sd 2.9e−4)	6.2e−4 (sd 1.8e−4)
0.6	0.4	3.9e−4 (sd 1.0e−4)	7.7e−4 (sd 1.9e−4)	5.4e−4 (sd 1.3e−4)
0.4	0.6	1.3e−3 (sd 1.1e−3)	2.4e−3 (sd 1.4e−3)	2.0e−3 (sd 1.1e−3)
0.2	0.8	1.2e−3 (sd 7.4e−4)	8.0e−3 (sd 5.5e−3)	6.7e−3 (sd 4.5e−3)

The validation and global errors are always higher than the training error, especially when the training set is small. Visually, networks succeeds at generalisation only if the training set contains more than 20% of the data (20 points). Moreover, convergence is faster when there are less training points, but it penalizes generalisation.

4 Results and discussion - Part II

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

To look at the influence of the number of hidden nodes on the training, validation and generalization, we look at the mean squared errors averaged over 20 runs given a number of nodes and a regularization strength. Data is split between 0.8 validation and 0.2 training. The optimizer is stochastic gradient descent with 0.9 momentum.



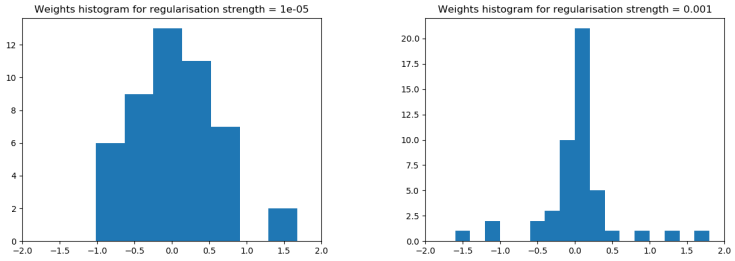
(a) Losses, l1=0.001 (b) 5 hidden nodes, l1=0.01 (c) 5 hidden nodes, l1=0.001

Figure 3: Influence of number of hidden nodes and learning rate on training

- It is hard to make a confident interpretation: at 95% confidence the errors are not distinctly different for all the nodes. If we try to interpret, there

might be some overfit with 4 nodes. A higher number of nodes gives more stable training and less overfit.

- We chose the less complex network that generalizes well, which is a two-layer network with five hidden nodes.
- A higher regularizer strength leads naturally to a higher MSE. Looking at the graph approximation shows that correctly adjusted, regularization gives a better fit for extreme values.

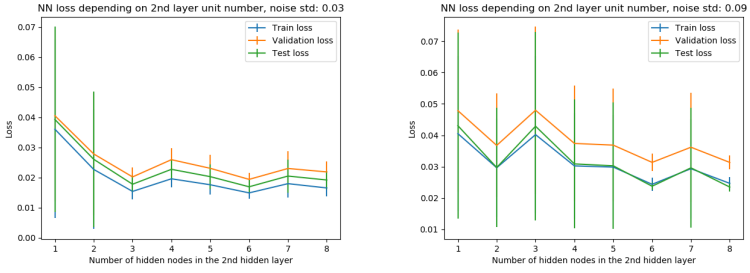


(a) lr=0.00001 (b) lr=0.0001

Figure 4: Histogram of weights depending on learning rates

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

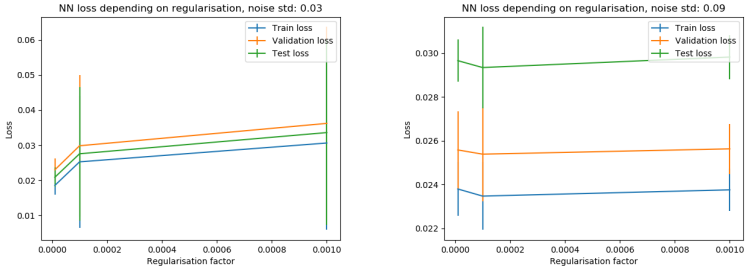
For the following results and tests, all of the data includes a zero-mean noise ($\sim \mathcal{N}(0, \sigma)$, $\sigma \in \{0.03, 0.09, 0.18\}$).



(a) $\sigma = 0.03$ (b) $\sigma = 0.09$

Figure 5: Averaged mean squared error, depending on the number of nodes in the second hidden layer (5 nodes in the first layer).

Figure 5 shows that the error tends to decrease with the number of nodes and 6 hidden nodes seem to be the most suitable in the second hidden layer.
Figure 6 shows that the mean squared error increases with the regularisation strength.



(a) $\sigma = 0.03$ (b) $\sigma = 0.09$

Figure 6: Averaged mean squared error, depending on the regularisation strength.

Based on the two previous observations, we chose a $5 - 5 - 6 - 1$ architecture for our three-layer perceptron, with a low regularisation strength (1.0e−4).

The following table compares the generalisation capabilities (mean square error of the predictions on the test set) of our two-layer architecture $(5 - 5 - 1)$ and our three-layer perceptron $(5 - 5 - 6 - 1)$ with the noisy data.

Network	Architecture	Average error (50 trainings)
two-layer	5-5-1	2.4e−2 (sd 2.1e−3)
three-layer	5-5-6-1	2.9e−2 (sd 1.2e−2)

Our two-layer perceptron has a lower error and standard deviation, which means that it has the best generalisation capabilities. Furthermore, two-layer architecture is prefered as adding a layer has a significant computation overhead in the backpropagation algorithm. Let N: number of input patterns, G: new hidden layer, H: previous layer, u: new weights matrix and n_I the number of hidden nodes in the layer I. Adding G adds a $\mathcal{O}(n_H.n_G.N) + \mathcal{O}(n_G.N)$ complexity overhead to the **forward-pass**, a $\mathcal{O}(n_G.n_H.N) + \mathcal{O}(n_G.N)$ overhead to the **backward-pass** and a $\mathcal{O}(N.n_G.n_H)$ overhead to the **weights update**.

5 Final remarks

This lab allowed us to get a proper understanding of the theorical and practical aspects of the perceptrons, how to build and use them. We also learned the impact of **noise** on the learning process and how it can improve generalisation capabilities.

Part 4 and the question about the impact of regularization and nodes on validation was a bit confusing: either the instructions were not detailed enough, or we made a mistake somewhere on the training.