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

André Silva, Axel Myrberger and Beatrice Lovely

January 28, 2021

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- Study and draw conclusions about the performance of a single-layer perceptron with different learning rules, with different data sets and varying settings
- Study and draw conclusions about the performance of a multi-layer perceptron on non-linearly separable data, compared to a single-layer perceptron
- Learn how to implement neural networks with libraries and tweak the setting of the network to better fit different assignments and problems.

2 Methods

We implemented part 1 in Python using numpy for matrix operations, and part 2 in Keras using Google Colab. Note for the first experiments with a single layer perceptron, the error metric used for measuring performance of the delta rule is the SSE (sum of squared errors), rather than MSE. This results in rather higher error rates but the results are still internally comparable. For perceptron rule the error is measured in terms of fraction of total points misclassified.

We experimented with the following generated multi-variate normal distributed data sets, with two classes:

- Linearly separable data: classA means [1.0, 0.5], classB means [-1.0, 0.0], sigma (standard deviation) 0.25
- non-linearly separable data: classA means [1.0, 0.3], separated in two clouds, classB means [0.0, -0.1], sigmaA 0.2, sigmaB 0.3

3 Results and discussion - Part I

In this section of experiments, unless otherwise noted we used a learning rate of 0.001, and initial weights randomly initialized using a normal distribution with mean 0 and variance 1. The number of epochs was limited to a maximum of 200, but the algorithms typically converged in around 40 epochs at this learning rate.

3.1 Classification with a single-layer perceptron

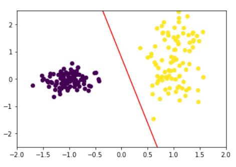
3.1.1 convergence speed with varying learning rates and learning modes

We experimented with different learning rates for the perceptron and delta rules respectively. Predictably learning coverges slower with a lower learning rate. The perceptron rule is able to handle higher learning rates (up to 0.1). However if learning rate for the delta rule is set any higher than about 0.006 error starts ballooning and the algorithm fails to converge. The delta rule converged in as little as 12 epochs at learning rate 0.005, but this learning rate was too fast for later experiments. Similarly a very low learning rate (0.0001) fails to converge within the time limit of 200 epochs. We settled on 0.001 as a safe, if somewhat slow, learning rate for the later experiments.

For the delta rule we also experimented with batch learning vs sequential learning with different random initializations (changing the distribution of the initial weights. With this type of easily separable data, there is no marked difference in convergence between batch and sequential learning although in later sections, with a two-layer perceptron and non-linearly separable data, batch learning converges faster. Weights initialization affects the initial error but has no marked effect on convergence speed. Likely sequential learning would present a significant bottle neck on performance in real, larger data sets with much higher dimensionality.

3.1.2 decision boundaries

When plotting the decision boundary, as expected, the Delta rule finds a better decision boundary than the Perceptron rule.


(a) Decision boundary for perceptron rule

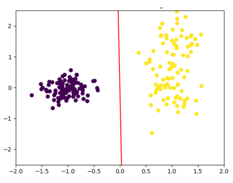
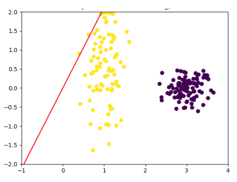

(b) Decision boundary for delta rule

Figure 1: comparison of decision boundaries for perceptron and delta rule learning

3.1.3 the importance of the bias term


(a) no bias learning

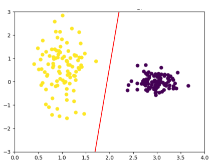

(b) with bias learning

Figure 2: decision boundary of delta rule with and without bias, classB moved from [-1.0, 0.0] to [3.0, 0.0]

If bias is removed, the perceptron is only able to classify data points that are centered around 0 (the origin). We proved this by first moving one of the classes (mB = [3.0, 0.0] instead of [-1.0, 0.0]) and removing the bias, upon which the decision boundary remains with y-intercept 0. When bias is used this repositioned data is once again successfully classified. See Figure 2.

3.1.4 Non-linearly separable data

We experimented by moving the means of the data points closer and increasing the spread, both the delta rule and perceptron rule perform surprisingly well even with somewhat overlapping data, although the error is predictably higher. For example there is a ten-fold increase in the sum-squared error (from around 5.5 to 55) for the delta rule when sigma is increased to 1 for both point clouds, and their means are shifted closer. We also created a different, non-linearly separable data set and experimented with subsampling, see below.

Subsampling method	Delta rule total error	False positive rate (perceptron)	False negative rate (perceptron)
removing 25% of each class	48.2	0.29	0.28
removing 50% of classA	17.7	0.12	0.08
removing 50% of classB	52.8	0.16	0.3
removing outliers in classA	43.3	0.3	0.09

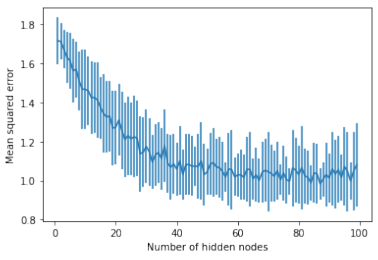
Table 1: error rates for subsampling of non-linearly separable data

See Table 1 for a performance comparison of delta rule and perceptron rules on different subsamples of non-linearly separable data. Unsurprisingly, both the delta and perceptron learning rules perform quite badly on this non-linearly separable data. Visually, the decision boundary cuts both classes in half, with no meaningful classification. Subsampling by removing a portion of one class or the other shifts the decision boundary in the direction of this class, with the result that more points from teh underrepresented class are misclassified. This illustrates the effect of bias that can be introduced if data is not subsampled correctly eg when splitting data into training and test sets (if the classes are unevenly represented). Removing outliers in class A had a negative effect on misclassification for that class, but a positive effect for class B, since there were more points remaining from class B.

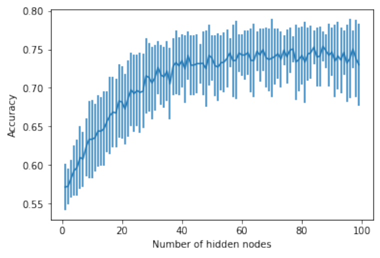
3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

We performed an experiment to better understand the effect the size of the hidden layer has on the performance of the model. All experiments used, by default, a learning rate of 0.001, α of 0.9 and were trained for 1000 epochs.



(a) Mean squared error in function of the size of the hidden layer.



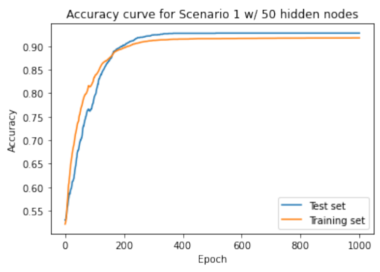
(b) Accuracy in function of the size of the hidden layer.

Figure 3: Model performance in function of the number of hidden nodes.

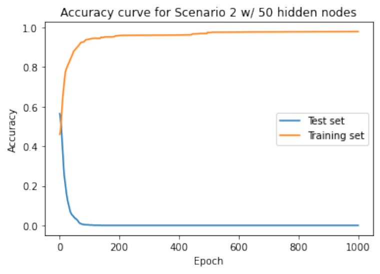
Figure 3 (a) plots the mean squared error in function of the number of hidden nodes. The blue line represents the mean of the 50 experiments for each problem instance, while the blue bars represent the standard deviation. Figure 3 (b) plots the accuracy in function of the number of hidden nodes. The blue line represents, again, the mean of the 50 experiments for each problem instance, while the blue bars represent the standard deviation.

We can see that, as we increase the number of hidden nodes we get a smaller error and bigger accuracy, hitting a plateau after approximately 40 nodes. This plateau might indicate that our problem is not complex enough to require more nodes, or that our learning process isn't optimized enough to take advantage of them, either in learning rate or epochs.

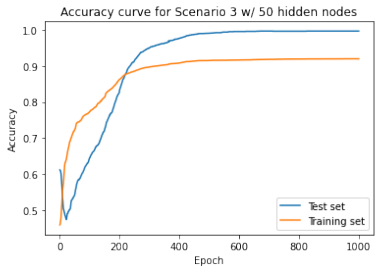
We then experimented with different training/validation set splits, as in 3.1.3, and acquired the results showed in Figure 4 in terms of accuracy. Figure 5 shows us the decision boundaries for the different scenarios.



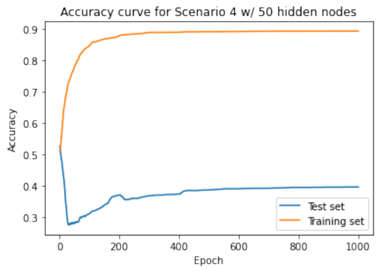
(a) Accuracy for test and validation sets for Scenario 1.



(b) Accuracy for test and validation sets for Scenario 2.



(c) Accuracy for test and validation sets for Scenario 3.



(d) Accuracy for test and validation sets for Scenario 4.

Figure 4: Learning curves for different validation set splits.

We can see that for Scenario 1, where we remove 25% of each class for the test set, our curves are very identical. For the remaining scenarios, where we have sweked training and test sets, our curves do not align. The most dissimilarity is observed in Scenario 2 and Scenario 4, where class A is the only class present in the test set. We can conclude that, in order to have a model that generalizes well, representative training and validation sets are essential.

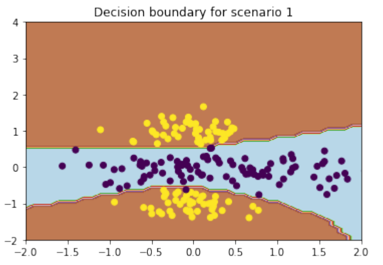
Our experiments to understand the behavior of the network depending on the size of the hidden layer in these various configurations did not present any different result from what we could conclude in the unsplit data experiment.

Figure 6 shows the results of our experiment to understand the difference between sequential and batch learning. By looking at the results, we can conclude that sequential learning presents itself as a bottlneck of the model's performance when compared to batch learning.

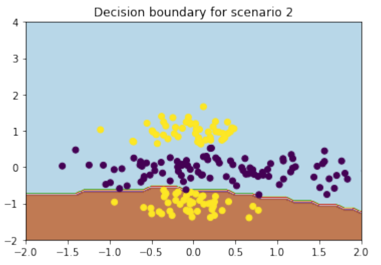
3.2.2 Function approximation

We designed and performed an experiment to observe the behaviour of the network in approximating a continuous function (regression) in function of the number of hidden nodes.

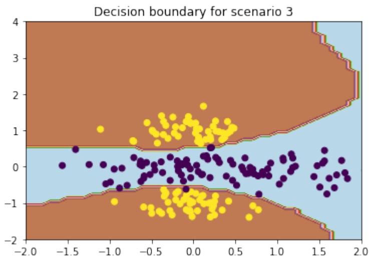
Figure 7 shows the plots for the approximations given by models with 1, 5, 10 and 25 hidden nodes. We used, by default, a learning rate of 0.001, α of 0.9 and trained each model for 1000 epochs.



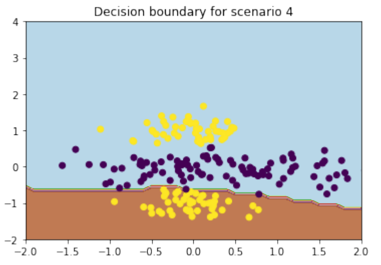
(a) Decision boundary for Scenario 1.



(b) Decision boundary for Scenario 2.

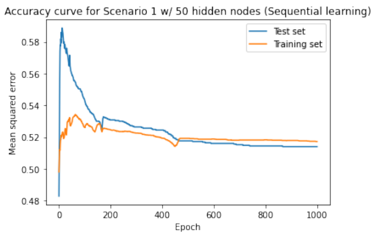


(c) Decision boundary for Scenario 3.

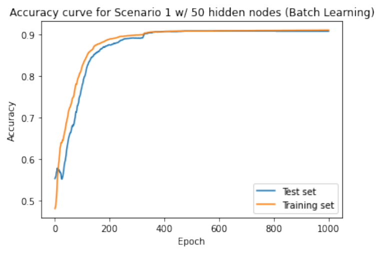


(d) Decision boundary for Scenario 4.

Figure 5: Decision boundaries for different validation set splits.



(a) Learning curve for sequential learning.



(b) Learning curve for batch learning.

Figure 6: Model performance for sequential and batch learning methods.

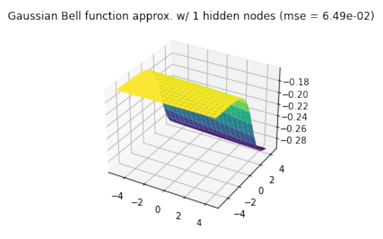
We selected a model with 25 hidden nodes, as this was the one which provided the best results.

In Figure 8, we show the results for our experiments to understand the behaviour of the network in function of the number of hidden nodes, the number of training samples, and the learning rate. The blue lines represent the mean of 50 experiments of each problem instance, while the blue bars represent standard deviation.

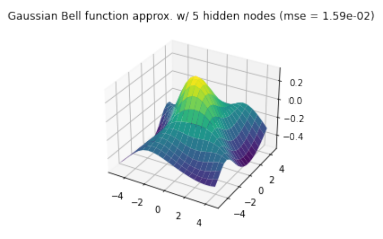
We can conclude that, as in classfication, the more nodes we have in the hidden layer the better the performance. It hit a plateau at approximately 13 hidden nodes, which can be explained by the same reasons as in the classification problem.

In terms of training ratio, we also observe that the larget the training ratio the better the model can generalize, achieving better results in terms of performance when predicting the test set.

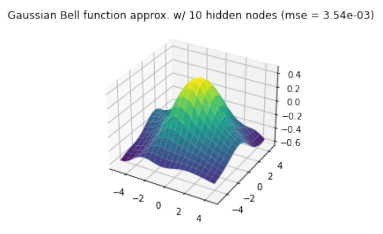
As for learning rate, we observe that low training rates don't achieve the best performance, while large training rates also miss this target. Low learning rates fail to achieve a good performance due to doing a slow gradient descent, while large learning rates tend to overshoot when performing the gradient descent.



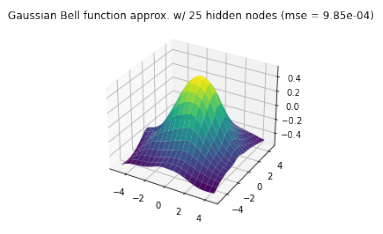
(a) Approx. Gaussian Bell function by a network w/ 1 hidden nodes.



(b) Approx. Gaussian Bell function by a network w/ 5 hidden nodes.

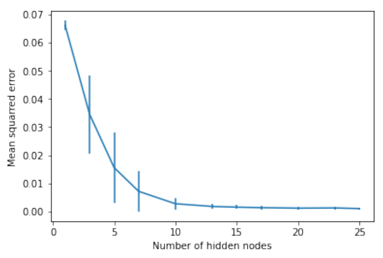


(c) Approx. Gaussian Bell function by a network w/ 10 hidden nodes.

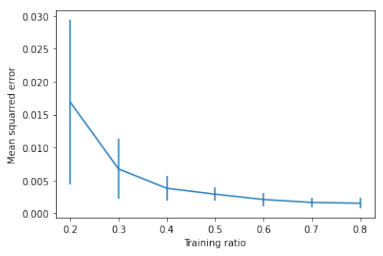


(d) Approx. Gaussian Bell function by a network w/ 25 hidden nodes.

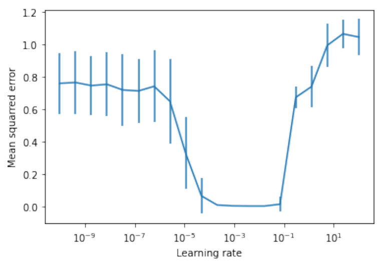
Figure 7: 3D Plots of the Gaussian Bell function experiment.



(a) Mean squared error in function of the number of hidden nodes.



(b) Mean squared error in function of the training ratio.



(c) Mean squared error in function of the learning rate.

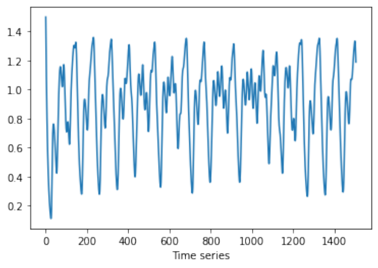
Figure 8: 3D Plots of the Gaussian Bell function experiment.

4 Results and discussion - Part II

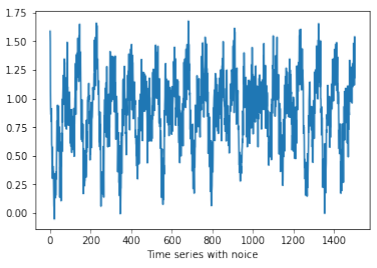
4.1 Three-layer perceptron for time series prediction - model selection, validation

Of the available data 1200 data points, the data was split into three sets of data. Training was given the first 700 data points, validating the data points with index 700 to 1000 and lastly the test data set was given the last 200 data points. Every data point consisted of five inputs and one output each part of the Mackey-Glass time series se figure 10. The input data was also given white noise with $\sigma = 0.15$ and $\sigma = 0.05$, resulting in three different data sets in total. Stocastic gradient decent was used as optimizer with mean square error as metric. The learning rate used was 0.001 for al the experiments in this part. 40 epochs where used for all the expermients because the validation error converged satisfactorily for all experiments.

We experimented with different numbers of nodes in the hidden layers (given by nh1/nh2 = number of hidden nodes in each layer). With nh1 = 3,4,5 and nh2 = 2,4,6 there is in total 9 different combinations of nodes in the hidden layers. Early stopping was used with patience = 5, meaning that if no improvement occurred for 5 epochs the learning stopped to prevent overfitting.



(a) Time series gernerated by Mackey-Glass time series



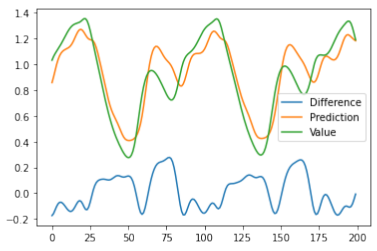
(b) Time series with white noise $\sigma = 0.015$

Figure 9: Time series generated with and without noise

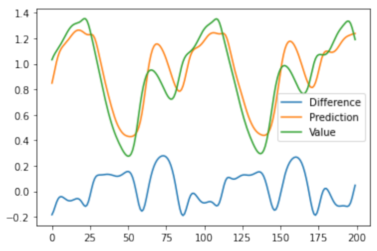
Metrics	3,2	3,4	3,6	4,2	4,4	4,6	5,2	5,4	5,6
Min validation loss	0.014	0.011	0.020	0.022	0.020	0.015	0.018	0.011	0.017
Min mse	0.015	0.014	0.022	0.025	0.024	0.019	0.022	0.015	0.020

Table 2: Performace of the validation data set

By this we can draw the conclusion that [5,4] appears the be the most stable architecture and [4,2] the least stable. For the most stable architecture we achieved average difference of 0.013 and standard deviation of 0.13 for the difference between the predicted and real results. For the least stable architecture we achieved average difference of 0.02 and standard deviation of 0.13.

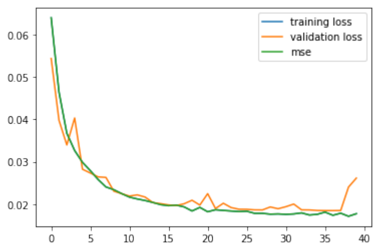


(a) Prediction of timeseries without noise, most stable prediction

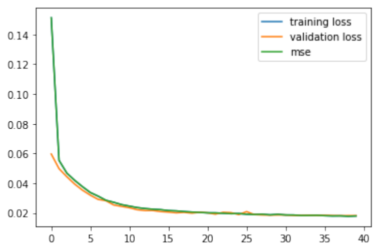


(b) Prediction of timeseries without noise, least stable prediction

Figure 10: Performance for classification of time series without noise



(a) results, $\sigma = 0.015$ and $\lambda = 0.01$



(b) results, $\sigma = 0.015$ and $\lambda = 0.001$

Figure 11: Validation results of network with 5 nodes in the first layer and 9 nodes in the secound.

4.2 Three-layer perceptron for noisy time series prediction with penalty regularisation

- Noise drastically increases the variability of the shape of the output predictions, making both the predictions and the actual value noisy. This can also be seen in the difference, as the NN predicts a noise and the output also has noise the volatility of the difference increases even further.
- With higher regularization the number of epochs needed for converges is increasing as the volatility of the mse and validation error increases.
- The regularization decreases the average error in the predictions but the standard deviations is not notably different with the regularization or without.
- Higher regularization constants tend to lead to higher mse validation errors. With more nodes in the second layer the regularization constant is more important to prevent the network from overfitting.
- Early stopping and regularization techniques prevent overfitting. When regularization is used the early stopping never occurred.
- We found that regularization and the noise of the signals did not interfere, and the results where similar in both cases regardless of the how noisy the signals where.

5 Final remarks

This was an interesting and useful first introduction to neural networks. Implementing simple networks by hand gives a good understanding of the internal workings. Experimenting with removing bias, subsampling etc gave a good illustration and confirmation of topics covered in lectures.

Keras is a great tool for implementing neural networks with good documentation and resources available online. This was a good learning opportunity because constructing a simple neural network for time series prediction is useful knowledge to many real-world scenarios. With the many default setting in Keras it is possible to implement a network with almost no knowledge at all about neural networks.