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

Shuyuan Zhang,Xinxing Guo,Wenqin Chen

September 10, 2019

1 Main objectives and scope of the assignment

*O*ur major goals in this assignment are:

- design and apply networks in classification, function approximation and generalisation tasks
- identify key limitations of single-layer networks
- configure and monitor the behaviour of learning algorithms for single- and multi-layer perceptrons network
- recognise risks associated with back-propagation and minimise them for robust learning of multi-layer percetrons.

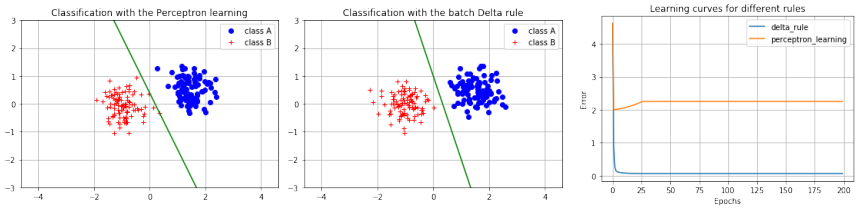
2 Methods

We worked with Python and MATLAB as programming languages in this project, and we used numpy and matplot functions in Python and the NN toolbox in MATLAB respectively.

3 Results and discussion - Part I

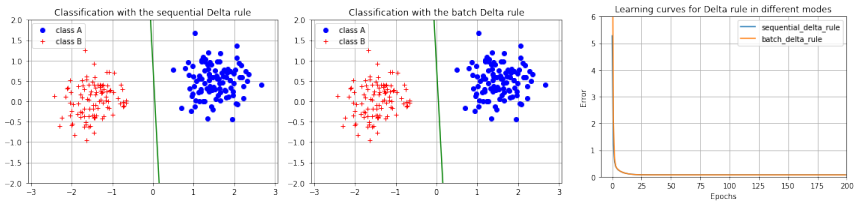
3.1 Classification with a single-layer perceptron

In this section, we compared two kinds of classification methods, namely the perceptron learning rule and the Delta learning rule in batch mode respectively. In this case,the data is linearly separable, the Delta Rule in batch mode performances better than the perceptron mode since the Figure1 shows the decision boundary for the perceptron learning rule is closer to points than that of batch Delta rule. Moreover, the Delta rule converges faster.
Figure2 shows the result of classification with Delta rule in different modes, sequential/batch mode respectively. We found that the error rates are similar in both modes when feeding the same data set with same initialisation. On the other hand, we found different random initialisations may influence the speed of convergence of the algorithms more or less.



(a) Perceptron learning (b) Batch Delta rule (c) Learning curves

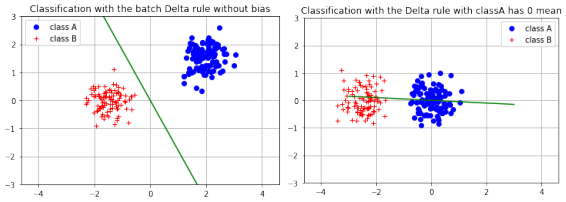
Figur 1: Classification performance with different rules



(a) Sequential Delta rule (b) Batch Delta rule (c) Learning curves

Figur 2: Classification performance with Delta rule in different modes

When removing the bias and training the network with the Delta rule in batch mode, the classification performance is shown as Figure3



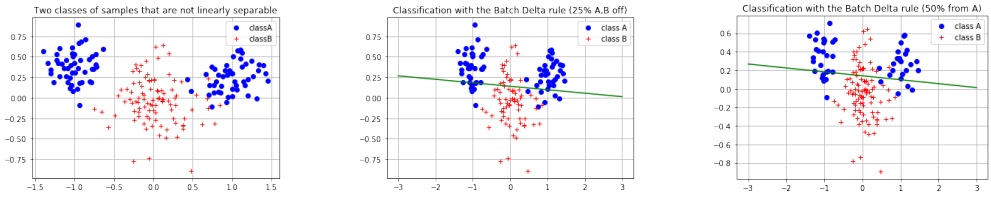
(a) Batch Delta rule without bias (b) Delta rule with classA = [0, 0]

Figur 3: Classification performance with rules without bias

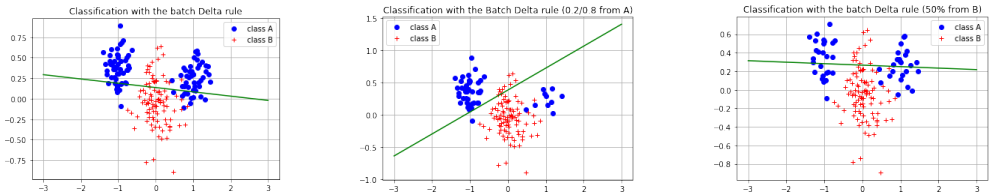
The perceptron might not work to classify all the samples correctly when the boundary goes through the origin. In sub-figure(b), we set the mean of class A to zero. In our test, the error has not converged to 0 even after 1000 epochs. Hence, the perceptron learning rule without bias is not able to classify correctly the sample from class A.

Figure 4 shows the classification performance when we remove different part of each class. Subfigure(a)(d) show the initial data set and classification with batch Delta rule.

When we remove 0.25 of each class, the decision boundary remains the same and the algorithm converges fast. If we remove half of the samples from one of the classes, the decision boundary has changed a little bit and the algorithm is a bit slower and it still cannot divide the samples correctly. In the fourth scenario, the decision boundary is totally different. Actually it seems that we have almost removed most of the samples from class A which had a positive abscissa, so it seems that the data set is now nearly linear separable.
In conclusion, if the different classes of datasets are sampled with equal probability, the generalisation will not be affected. But, if the sampling is biased, then the predictions for test data would be wrong.



(a) Dataset distribution (b) Random 0.25 from A,B (c) Random 0.5 from A



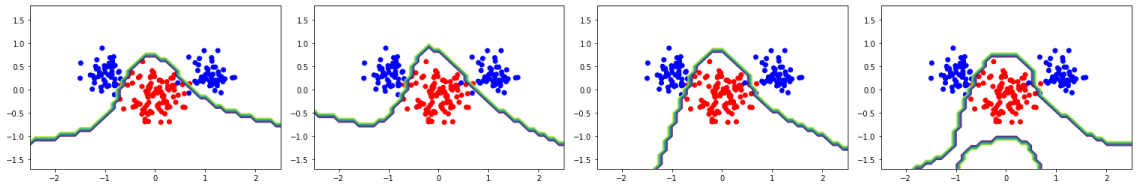
(d) Batch Delta rule (e) 0.2/0.8 form A (f) Random 0.5 from B

Figur 4: Classification performance when setting different ratios of datasets

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

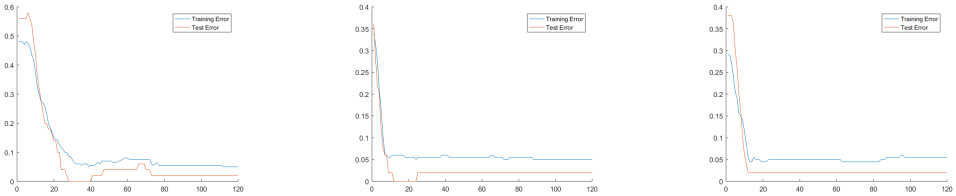
If we want to use neural network to separate them, we need at least 5 hidden neurons. In fact, we also found that increasing the number of neurons does not lead us to better results necessarily.



(a) 5 hidden nodes (b) 20 hidden nodes (c) 30 hidden nodes (d) 100 hidden nodes

Figur 5: Classification and regression with a two-layer perceptron

In most of the cases, the mean square error of validation datasets is greater than that in the training dataset. It is reasonable because the validation dataset belongs to data that has not appeared. However, the two learning curves are very similar, but as the ratio of validation dataset to training dataset increases, the error generated by the validation dataset increases. Through a comprehensive analysis of Fig.5and Fig.6, curves and the network performance depends not only on the number of hidden neurons mentioned earlier, but also on the ratio of the validation dataset to the training dataset.



(a) 2 hidden nodes (b) 4 hidden nodes (c) 6 hidden nodes

Figur 6: Mean squared error with different hidden nodes of validation and training dataset

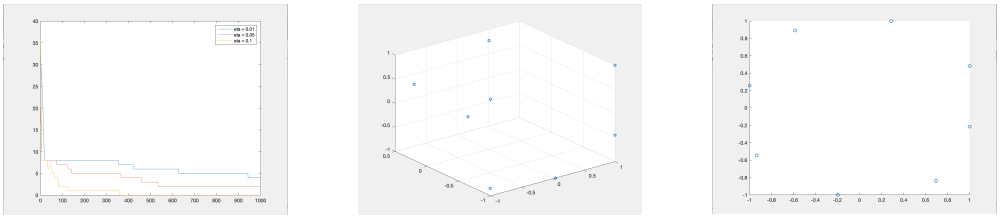
3.2.2 The encoder problem

Our network always converges. A typical example is shown in table 1.

Layer								
Input	-1	-1	1	-1	-1	-1	-1	-1
Output	-0.9993	-0.9953	0.7098	-0.7350	-0.9985	-0.7686	-0.9981	-1.0000
Intermediate	0.2641	0.2597	1.0000					

Tabell 1: Values of input, output and intermediate values for one vector

According to the table, the network maps inputs nearly identical to themselves, and generates a low-dimensional representation of the original 8D vector. Figure 8(a) shows that the network converges quickly in this small task.
The internal code represents how the network mapped 8D vectors into lower dimensional spaces.

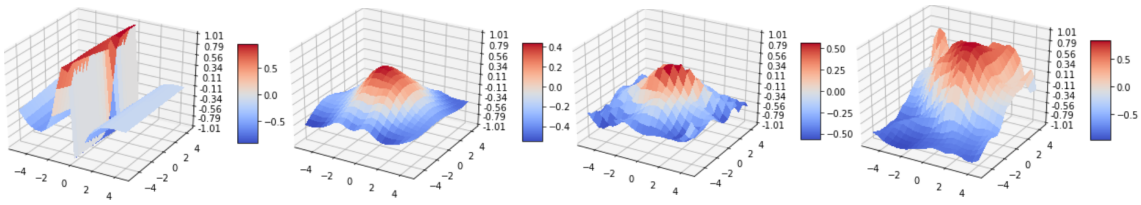


(a) Training error (b) 3D representations (c) 2D representations

Figur 7: Encoder Results

We plotted the resulting 3D vectors in figure 8(b). We can see that the orthogonal vectors in 8D space are divided sparsely in 3D space.
If we use 2 hidden nodes, then the 8D vectors are represented in a 2D space, which means we can plot them in a 2D plane as figure 8(c). Also, after sufficient training, the orthogonal vectors are distributed evenly in the 2D space.
Thus, auto-encoders can be used to reduce dimensionality of high-dimensional data. They can extract some useful features out of complicated data sets and use them to train other networks. This could help improve the performance as high dimensional data may cause some problems.

3.2.3 Function approximation



(a) 3 hidden nodes (b) 10 hidden nodes (c) 20 hidden nodes (d) 40 hidden nodes

Figur 8: Approximation with different hidden nodes

We found that if the number of hidden nodes is less than 5,the image is somehow incomplete. As the number of neurons increases, the fitting effect will gradually increase. The reason for this may be that too few hidden neurons cannot model the distribution correctly.
On the other hand, when the number of hidden neurons increases to more than 24, the neural

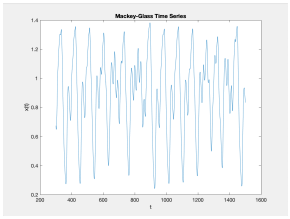
network's fitting effect will gradually decrease. The reason is that when the number of hidden neurons increases, it becomes more complex and over-fitting the data.

Ratio of train-test	0.2	0.3	0.4	0.5	0.6	0.7	0.8
MSE	0.0232	0.0176	0.0141	0.0083	0.0048	0.0033	0.0032

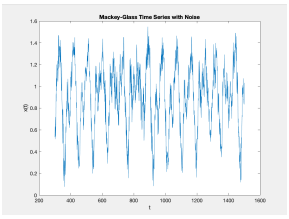
Tabell 2: MSE Results of different data ratio.

4 Results and discussion - Part II *(ca.2 pages)*

The Mackey-Glass time series is visualized in figure 9.



(a) Without noise



(b) With noise $\sigma = 0.09$

Figur 9: Mackey-Glass time series

All our results in this part are based on 50 independent NN train trials. So the results are statistically sufficient.

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

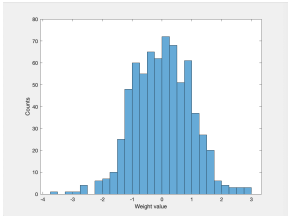
We trained neural networks of different settings on the time series and the average and standard deviations of MSE on validation set are shown in table 2.

Mean	h=2	h=4	h=8	std	h=2	h=4	h=8
r=0	5.2073e-03	9.7240e-04	8.2166e-04		3.5319e-03	3.6206e-04	2.2207e-04
r=0.5	2.6710e-03	8.9247e-04	8.6544e-04		3.4038e-03	9.5138e-04	7.4568e-04
r=1.0	3.4038e-03	9.5138e-04	7.4568e-04		3.4038e-03	9.5138e-04	7.4568e-04

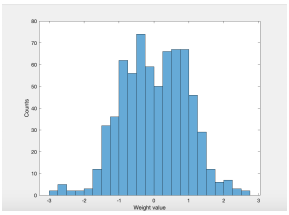
Tabell 3: MSE Results of a two-layer network with different settings of hidden layer h and regularization term r.

We may notice some interesting properties from the table. First, the overall performance of the network grows if we have more hidden nodes. MSE reaches its lowest when we use a network with 8 hidden nodes. Second, the effect of regularization seems different across networks. For $h = 2$ and $h = 4$, $r = 0.5$ gives the best results. While in a network with $h = 8$, $r = 1$ is the best choice. This is reasonable because a more complex network may need a higher degree of regularization to avoid overfitting.

The effect of regularization on the distribution of weights in a 8-layer network can be seen in figure below.
With a higher level of regularization, weights tend to be smaller and concentrated around 0. The weights are little bit smaller when $r = 1$.



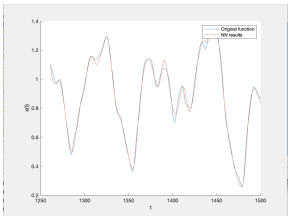
(a) Weight distribution when $r = 0$



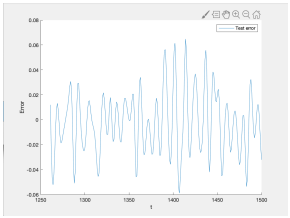
(b) Weight distribution when $r = 1$

Figur 10: Effect of regularization on weight distributions

Since the network with 8 hidden nodes and $r = 1$ has the lowest MSE on the validation set, we used it for further evaluation. We Plotted these test predictions along with the known target values. Also, we plotted the error directly.



(a) Difference with the original function



(b) Errors

Figur 11: Predictions on test set

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

In this part, the number of nodes in the first layer is 8. Regularization term $r = 1.0$.

Mean	$h_2 = 2$	$h_2 = 4$	$h_2 = 8$	std	$h_2 = 2$	$h_2 = 4$	$h_2 = 8$
$\sigma = 0.03$	3.2622e-03	3.0331e-03	2.7120e-03		1.6678e-03	2.5307e-03	8.8188e-04
$\sigma = 0.09$	1.1279e-02	1.1953e-02	1.0732e-02		9.6822e-04	4.3008e-03	1.3955e-03
$\sigma = 0.18$	3.9874e-02	4.0326e-02	3.8502e-02		6.4694e-03	4.9855e-03	4.3885e-03

Tabell 4: MSE Results of a three-layer network with different settings of hidden layer h and std term σ.

After we added noises, the model is more susceptible to over-fitting at it tries to fit the noise part of the data as well. From table 3 we can see that MSE grows as the std of the noise grows.

Mean	$r = 0.0$	$r = 0.5$	$r = 1.0$	std	$r = 0.0$	$r = 0.5$	$r = 1.0$
$\sigma = 0.03$	2.1083e-03	2.8717e-03	2.9244e-03		7.0480e-04	9.6302e-04	1.8015e-03
$\sigma = 0.09$	1.1229e-02	1.2727e-02	1.1830e-02		2.1106e-03	2.6193e-03	2.3939e-03
$\sigma = 0.18$	4.1484e-02	3.8624e-02	3.7615e-02		1.0427e-02	4.6856e-03	4.5031e-03

Tabell 5: MSE Results of a three-layer network with different settings of std term σ and regularization term r.

All results in table 4 are produced by a 3-layer network with hidden nodes set to 8-8. If the std of the noise is relatively high, it's very useful to add regularization term to prevent a complex network from over-fitting(MSE decreases as we add r when $\sigma = 0.18$)
For intermediate level of noise $\sigma = 0.09$, We compared a two layer network(5-8-1,r=1.0) with a 3-layer network(5-8-2-1,r=1.0), the 3-layer network performs slightly better.

Layer	MSE_average	MSE_std
2	1.1760e-02	2.4623e-03
3	1.1077e-02	2.6430e-03

Tabell 6: MSE Results of a three-layer network VS a two-layer network.

If the network has more hidden layers or more nodes in a hidden layer, the computation cost of backprop grows.

5 Final remarks

This lab do strengthens our understanding of the concepts from the lectures, especially the concepts of over-fitting and lack-of-fitting, and hold-out validation. At first we built perceptrons and networks from scratch, then we used them to solve some interesting problems such as encoder, function approximation and time series prediction. The only problem or difficulty we faced is that the lab has too many little questions that made us hard to find a solution.