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

Tobias Bezner, Ricardo Del Razo and Nik Joel Dorndorf

January 30, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to learn how to implement a simple ANN from scratch
- to see how different tasks can be solved by this ANN
- to look at different behaviours of the ANN under certain conditions

All of the above mentioned tasks should be solved by writing code that can be used across different tasks. The code should be readable and as simple as possible, so that others can understand and adapt it in a short time. When running experiments for multiple times to observe the variability and randomness of the models, we keep the number of runs in a feasible range.

2 Methods

As programming language we used Python. For including different libraries, we used Anaconda. The used libraries include numpy, scipy, matplotlib and tensorflow. For sharing our code repository, we created a github project.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

In order to analyse the classification with a single-layer perceptron, we made different investigations. First, we compared the convergence of the perceptron

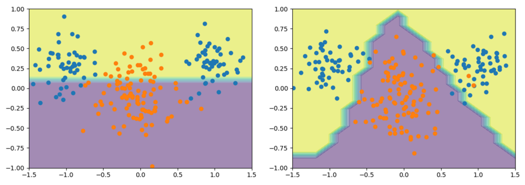

Figure 2: left: decision boundary of single-layer perceptron (delta rule), right: decision boundary of two-layer perceptron (backprop)

learning and the Delta learning rule for batch learning. For different learning rates the Delta learning rule converges commonly faster. We could not detect any difference between the convergence of the iterative and the batch learning for the Delta learning rule.

In general the random initialisation of the weight matrix W has a great influence on the convergence and the final loss. If the initial values are far away from the optimal solution, all learning rules tend to converge to a local minima with a high loss. Therefore, we had to run the training several times to calculate mean values to get comparable values. For this part of the assignment we used 30 runs on the same dataset with 15 epochs and modified the learning rate.

Without the bias term, the Delta learning rule approach is not always able to separate the linearly separable classes. The border line of the trained network is a straight line through the origin and there is no linear displacement possible. As you can see in the Figure 1, it is not possible to separate data samples with both classes located in the same direction of the origin of the coordinate system.

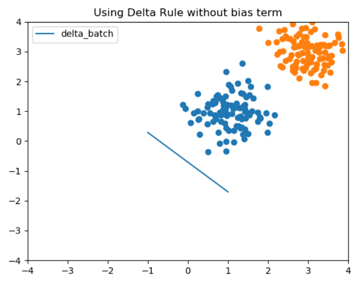

Figure 1: Delta learning rule without bias term

When using a data samples of not linearly separable classes (point clouds overlapping), it turned out that the Delta learning rule (batch and iterative) converge slower and with a higher loss. In this case, the perceptron learning rule is not converging generally and becomes more unstable with higher learning rates.

The linearly non-separable problem could not be solved sufficiently by a single-layer perceptron as one can see in Figure 2. Removing different points from the training set just moved and turned around the boundary.

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

In the case of the two-layer perceptron with at least 6 hidden neurons the linearly non-separable data can be classified by the model as one can see in Figure 2 (100% correct classification is not possible because of the data distribution). Comparing sequential and batch training for this ANN, the batch one was converging way faster than the other one.

For the rest of this task, we took a separate validation set to compare the validation loss to the training loss. When doing this, we observed that the error curves for both sets looked very similar during the training with the validation error being a little bit higher than the training error. When we increased the sigma for the distribution, meaning a higher variance between the points in the cloud, the dissimilarity got a little bit larger. The same happens when we leave out specific datapoints in the training set from only one class. What is interesting to notice is that the dissimilarity between the validation and the training learning curves was getting even bigger when we used more than 20 hidden neurons. This is probably the case because these models are adapting to the training data even more and that is hurting the generalisation.

3.2.2 The encoder problem

The 8-3-8 encoder is always able to converge and map the inputs to itself but we had to increase the learning rate for this behaviour. When looking at the internal representation, we can see that to every one of the eight different options the three activations of the hidden layer perform another pattern regarding their signs (e.g. -++, +++, -+-, ...). There are nine different options to combine these three signs which is enough to map the eight inputs to itself. When we decrease the encoder to a size of 8-2-8 the mapping is not fully working anymore. Autoencoders could be a nice way to create lowerdimensional representations of data e.g. for images.

3.2.3 Function approximation

The surface of the function to approximate can be seen in Figure 3. We tried different configurations of the ANN in terms of number of hidden nodes (while fixing the number of epochs and the learning rate). The results can be seen in Table 1. It shows that adding more nodes definitely helps the ANN to approximate this function. If you look at a single hidden node, the circular structure cannot be captured by the model. Still, compared to the ANN with only 4 hidden nodes but full training set, the model with 20 hidden layers and only

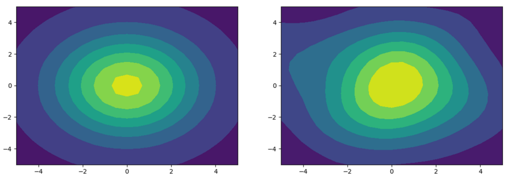

Figure 3: left: original function, right: approximation of best ANN with 80% training data

Hidden nodes	1	4	8	16	20	25
Val. Loss	0.259 (0.000)	0.157 (0.019)	0.103 (0.039)	0.069 (0.014)	0.053 (0.019)	0.076 (0.022)

Table 1: Mean validation loss (over 5 runs) for different numbers of hidden nodes (std in brackets)

20% of the dataset performs better.

In Table 2 the validation loss on the whole dataset is depicted for models (with 20 hidden nodes) trained on different percentages of the dataset. The ANN is already loosing a lot of approximation power when we only take 80% of the data for training. This trend continues when we remove even more points from the training set. It shows that for a good generalisation, model tuning is very important.

4 Results and discussion - Part II

For the whole section we chose the RMSprop optimizer which is a simple but powerful one that worked well for us in early tests. The maximum number of epochs was set to 1000. This point was never reached since all models stopped through early stopping before that. The dataset consisting of 1200 samples was split into train (800), validation (200) and test set (200).

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

Concerning the regularisation technique we decided to choose l2 weight decay. The reason for this was that a lot of sources stated that this is one of the best

Percentage	80%	60%	40%	20%
Val. Loss	0.081 (0.030)	0.092 (0.024)	0.091 (0.028)	0.130 (0.008)

Table 2: Mean validation loss (over 5 runs) for training the ANN on different percentages of the whole dataset (std in brackets)

	1	2	4	6	8
no reg.	0.053 (0.000)	0.024 (0.021)	0.004 (0.004)	**0.002 (0.000)**	0.010 (0.001)
	63.3 (2.6)	210.0 (111.2)	526.6 (171.8)	**295.3 (67.4)**	283.0 (44.5)
0.0001	0.011 (0.002)	0.009 (0.001)	0.004 (0.004)	0.005 (0.004)	**0.002 (0.001)**
	204.0 (180.4)	203.6 (152.6)	389.0 (95.5)	114.3 (57.6)	**258.6 (102.7)**
0.001	0.024 (0.020)	0.007 (0.004)	**0.002 (0.000)**	**0.002 (0.001)**	0.007 (0.004)
	226.0 (219.2)	508.6 (216.3)	**292.6 (30.9)**	**336.6 (38.0)**	209.0 (100.4)
0.01	0.025 (0.020)	0.010 (0.001)	**0.002 (0.000)**	**0.002 (0.000)**	0.003 (0.001)
	129.0 (39.8)	335.0 (145.7)	**378.0 (30.1)**	**254.0 (51.0)**	290.3 (48.8)
0.1	0.026 (0.019)	0.010 (0.003)	0.010 (0.003)	0.008 (0.001)	0.007 (0.001)
	309.0 (196.7)	405.0 (65.3)	313.6 (49.5)	259.0 (22.0)	203.0 (14.9)

Table 3: Mean validation loss (upper value) and number of epochs (lower value) over three runs for different numbers of hidden nodes (columns) and regularisation factors (rows). (Best loss values are bold. Stdev in brackets.)

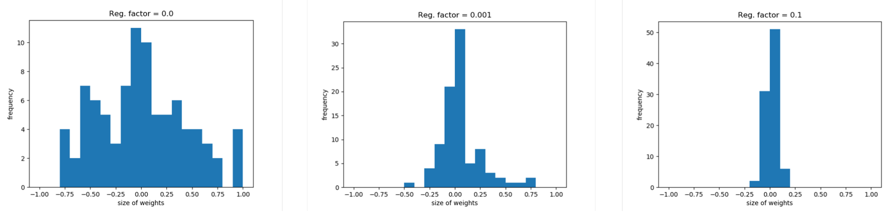

Figure 4: Histogram of weights for different regularisation factors

methods. That was the reason why we tried this method first. The results we obtained were good (shown below) why we decided to stay with this method.

Looking first at the lower values in Table 3 which show the number of epochs until the training was stopped through early stopping, we notice that the standard deviation for most of the configurations is high in general. By taking a closer look, one can see that by increasing the number of hidden nodes and the regularisation strength the standard deviation gets a little smaller which stands for a more stable training process. The more important value is the first one which is the validation loss after the last epoch. The model with only one hidden node performs by far the worst. Providing the network with more nodes is improving the results a lot until four hidden neurons but afterwards the performance of the models stays approximately the same. So using more than four hidden nodes seems to be unnecessary in this case. Looking at the overall best configurations, we notice a group of four that all reached a validation loss of 0.002. These results are achieved for 4 and 6 hidden neurons and a regularisation factor of 0.01 and 0.001. Increasing the regularisation even further seems to hurt the performance.

In Figure 4 one can see histograms of the weights of an ANN with 6 hidden neurons for different regularisation factors. One can immediately see the impact of the weight regularisation on the weight sizes. The stronger the regularisation the smaller and more centered the model weights.

We continued working with the model consisting of four hidden neurons and a regularisation factor of 0.001 since four neurons seemed enough and low regularisation did not hurt performance too much. Test results can be seen in Figure 5.

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

After adding different amounts of noise and another hidden layer, we evaluated the influence of the number of hidden nodes in the new layer and the regularisation factor. The noise is hurting the performance a lot and neither regularisation nor more nodes seem to effectively counter this. The only time we could observe a real trend was for the smallest amount of noise where stronger regularisation seemed to increase the performance.

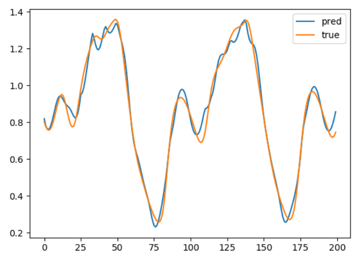

Figure 5: True test series compared to the prediction of the 4-node-model

Comparing the performance of the two-layer perceptron (4 hidden nodes) with the three-layer perceptron (two times 4 hidden nodes) on the medium noisy data, the performance of the bigger model was only slightly better (0.124 mean test loss compared to 0.131) but it needed 100 epochs less on average (147 compared to 251). Looking at the training time, the bigger model was trained faster (6.1 seconds compared to 21.7 seconds) which may be explained by the smaller amount of epochs. To check this we performed an experiment where we set a fixed number of 100 epochs without early stopping. For this experiments, there was no notable difference in training time for the both models but when increasing the number of hidden nodes for the three-layer perceptron the training time increased slightly simultaneously.

5 Final remarks

This lab was a jump in at the deep end. Implementing the different learning rules and especially the backpropagation was quite challenging and very interesting. It was nice to play around with a few parameters of the network and see how this affects the outcome.

For the second part with the regularisation our results were not that significant to say much about where the best points where and to what degree regularisation was really helping. Normally, you would guess that for more noise more regularisation is needed, but this was not always the case for use.