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

Magnus Tronstad, Teo Jansson Minne and Quintus Roos

January 28, 2021

1 Main objectives and scope of the assignment

The main objective with this assignment is to further extend our knowledge of neural networks by implementing and evaluating single-layer perceptrons (SLP) and multi-layer perceptrons (MLP) on various machine learning problems.

2 Methods

This assignment is written in Python 3. The first part uses the libraries Numpy for scientic calculations, Scipy for statistical functions, and Matplotlib for creating plots. For the second part, the libraries Tensorflow and Keras were used to construct the layered networks in addition to the previously mentioned.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

For the linearly-separable data, both delta rule and perceptron learning have relatively similar performance increases in general. With a normalized learning rate of 0.1 and with an average of over 100 weight initialization, we can observe that perceptron learning performs better than delta rule learning. The difference is slight, but reasonable as perceptron learning classifies by creating linear combination of the used variables compared to calculating the gradient of a cost function. Both implementations start to converge at around 20-25 epochs.

For sequential and batch learning for the delta rule, we can observe that perceptron learning outperforms batch learning. Since it converges faster than delta rule for a given learning rate, this also reduces the variance in the observed error curves since the trajectories are already very close to the uniquely defined minimum.

For delta rule learning without bias, we can observe that the delta learning rule obtains a misclassification rate of 50%, as the decision boundary gets "stuck" in the origin. Since the boundary cannot be shifted in the xy-plane, it is only able to classify data samples which are on opposite sides of the origin (i.e. 50/50 chance for each side).

The results from running simulations with the perceptron learning algorithm and the delta rule on linearly non-separable data for different learning rates are shown in Figure 1. Here, $m_A = [-1.5, 2.5], m_B = [-2.5, 1.5], \sigma_A = 0.15, \sigma_B = 0.08$. Clearly, the single perceptron had no way of perfectly separating the classes from each other—a consequence of it only being able to produce a linear separating hyperplane. In particular, we see that the perceptron learning rule never converges.

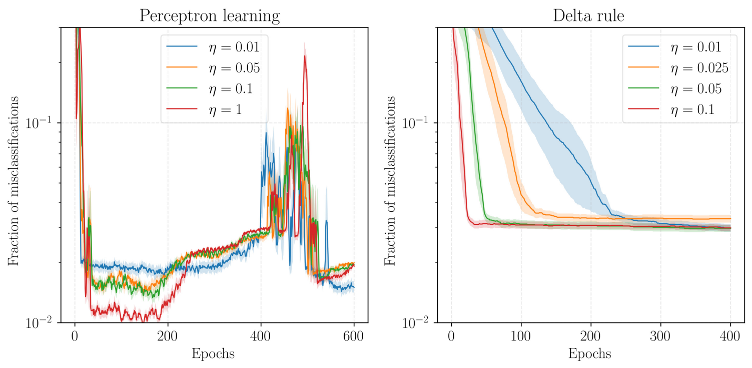


Figure 1: Convergence plots of error (fraction of misclassified points) for perceptron learning algorithm (left) and delta rule (right). Averaged over 30 weight initializations. (No momentum was used here)

Next, we study effects of subsampling data: remove 25% from each class (Subsampling 'A'), remove 50% from class A (Subsampling 'B'), remove 50% from class B (Subsampling 'C'), remove 20% from class A in the left half plane, and 80% from class A in the right half plane (Subsampling 'D').

Of the four data sets (A,B,C,D), subsamplings B and C produced the most interesting results. For these sets, the fraction of misclassified points (FMP) for each class appeared to be in conflict: especially early on during training, we see that as FMP for one class decreases, it *increases* for the other, this situation is illustrated in Figure 2 for subsampling C. Furthermore, because of the way the classes are distributed, this seems to favor a decision boundary that may neglect a large area of the plane, and we suspect this should lead to poor generalization error when the subsampled class actually has points within that region. Without subsampling, the corresponding training error converged to ~ 0.24 for class B and ~ 0.15 for class A.

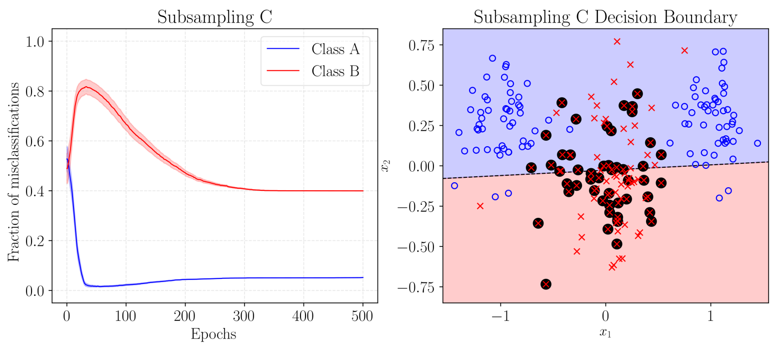


Figure 2: **Left:** Convergence plots of training error (fraction of misclassified points for each class) for subsampling C, averaged over 50 weight initializations. **Right:** Plot showing the data distribution for subsampling C as well as the decision boundary obtained from training with delta rule for 500 epochs with $\eta = 0.1$ learning rate. Black spots indicate the removed data points.

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

A two-layer perceptron was trained on the subsampling problem defined above. A momentum term was also added to the weight updates. Using a momentum paramter value of 0.9 and increasing the learning rate improved the speed of learning slightly. In the first experiment we tried different number of hidden neurons, $h_n \in \{2, 4, 8, 16\}$ and recorded the (training sample) MSE values during training. Both too few neurons (2) and too many (16) led to slow learning. A possible explanation is that having too few neurons yields a too simple model, not flexible enough, whereas having many neurons might make the error “landscape” highly complex, making it harder for gradient descent to navigate towards a favorable local minimum. With our particular data distribution, the fraction of misclassified points never converged under 0.005, corresponding to precisely *one* misclassified point, regardless of how many neurons we tried.

Training and validation error curves are shown in Figure 3 for each subsampling (A,B,C,D) and both for few (2) and many (12) hidden neurons. First, the validation loss was consistently lower for subsampling B when 2 neurons were used instead of 12. Second, subsampling C shows overfitting, as the validation error starts to increase after around 200 epochs. This is even more evident with subsampling D. To understand why, we may look at Figure 4. The network produces a skewed decision boundary, and consequently, generalizes so poorly (there are very few points belonging to class A (blue circles) in the right half plane!).

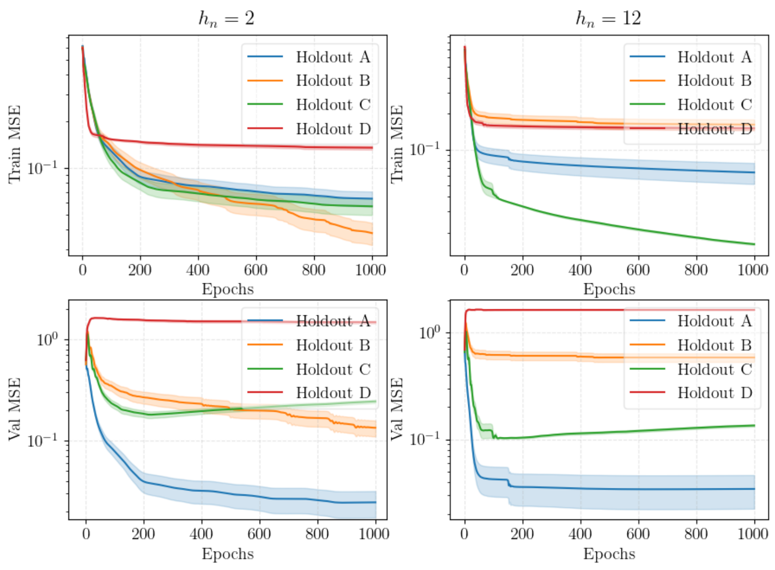


Figure 3: Training and validation error curves for each subsampling routine (A,B,C,D) and form 2 and 12 hidden layers (left and right columns respectively). Errors are averaged over 30 weight initializations and the faint color regions correspond to the standard error of the mean.

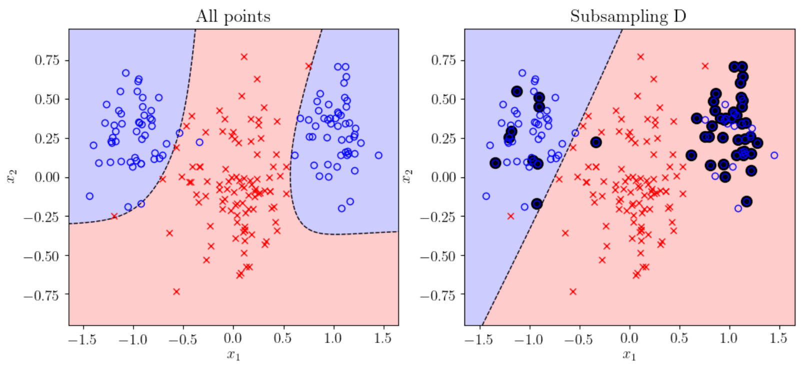


Figure 4: Decision boundaries when the MLP is trained with all data points (left) and using the training data according to subsampling D (right). 300 epochs with an 8 layer network was used to produce both plots.

Finally, sequential learning converged much faster than batch in terms of the number of epochs. However, we noted that the trajectories could become very noisy unless we slightly adjusted the learning rate so that it was a bit lower than when we used batch (that is, after having normalized by the number of training points). A downside to sequential that was noted was that it was much slower in terms of *wall clock time.*

3.2.2 Function approximation

Comparing different number of hidden neurons (h_n) between 3 and 25 we found that the MLP tended to get better at approximating the Gaussian shape with more neurons. This is exemplified in Figure 5 which shows the difference in the approximations with the equal number of epochs taken. Visualization together with training loss curves made it easy to spot and compare good vs bad models. The best model we found had 25 neurons. As seen in Figure 5, it does a very good job at approximating the underlying Gaussian distribution of data. More neurons allow the MLP more flexibility when performing the least squares minimization, and since the training data is uniformly spaced and completely noiseless in this case, so we do not expect big problems with overfitting. This may explain why the results seemed to improve with an increased number of hidden neurons.

Experiments were then run that explored different ratios, α, used for partitioning the data into training and validation sets ($\alpha = 1$ means all data is used for training). Results are shown in Figure 6 (left plot). We note that the final MSE (calculated on training+validation data) decreases as more points are used during training. However, as Gaussian noise is added to the targets ($\sigma = 0.1$), and only 20% of the data is used for training, the error starts to increase as training progresses (right plot of Figure 6), indicating problems of overfitting.

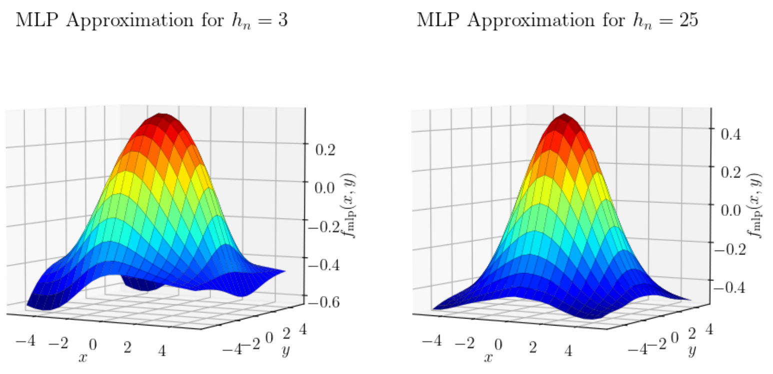


Figure 5: Plots showing MLP approximations after 10000 epochs for 3 hidden neurons (left plot) and 25 hidden neurons (right plot). Momentum parameter value 0.9 and learning rate $\eta = 1$ was used.

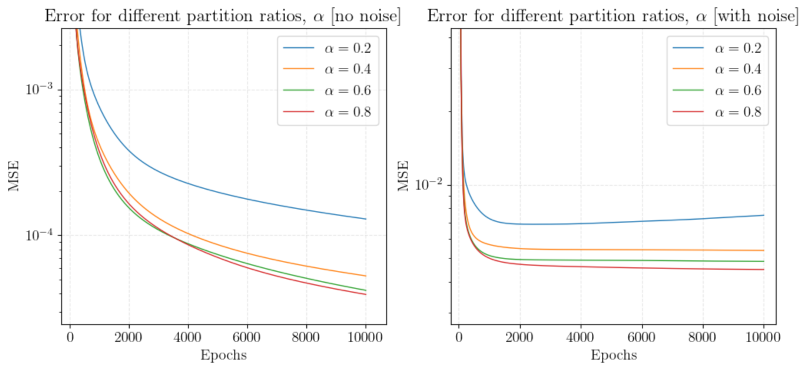


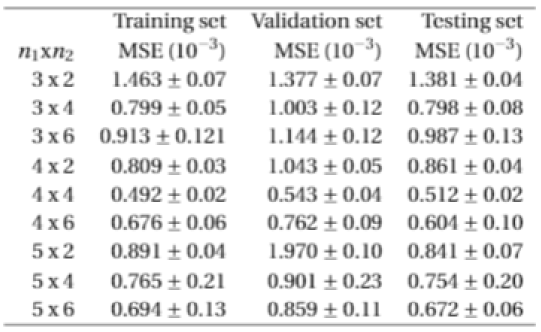
Figure 6: Plots showing MSE during training for different ratios of training data to validation to. Left: no noise on the targets. Right: Gaussian noise with $\sigma = 0.1$ added to targets.

Finally, it was found that we could reach a comparable MSE around 10 times earlier (1000 instead of 10000) epochs, when the learning rate was increased from 1 to 10 (with a fixed momentum parameter value of 0.9), greatly reducing the speed of training.

4 Results and discussion - Part II

4.1 Three-layer perceptron for time series prediction

A three-layer perceptron was used to predict the Mackey-Glass time series. Data was gathered from the Euler solution of the problem and then split into a training set, a validation set and a testing set using a $2/3 : 1/6 : 1/6$ ratio. This data was then used directly to train the network for model selection and six different architectures were tested with combinations of $n_1 = [3,4,5]$ and $n_2 = [2,4,6]$ where n_1 and n_2 represent the number of neurons in the first and second hidden layer respectively. Early stopping was implemented to prevent overfitting and a heuristic approach was used to decide on good testing parameters for the model selection. The model was set to run for 1000 epochs and the early stopping was given a patience of 30 epochs however as it proved too unreliable for lower values. A few different activation functions were tested for the hidden layers and the hyperbolic tangent function was settled upon as it showed the best performance during preliminary testing. In both the model selection and the later prediction test all combinations of architectures and hyperparameters were ran 20 times to reduce the effect of the stochastic nature of the weight initialization. The results of this first test can be seen in table 1 below. 4x4 had the best performance. Using 5 neurons in the first layer sometimes gave the best results but suffered from a lot more variance and sensitivity to initialization.



	Training set	Validation set	Testing set
n_1xn_2	MSE (10^{-3})	MSE (10^{-3})	MSE (10^{-3})
3 x 2	1.463 ± 0.07	1.377 ± 0.07	1.381 ± 0.04
3 x 4	0.799 ± 0.05	1.003 ± 0.12	0.798 ± 0.08
3 x 6	0.913 ± 0.121	1.144 ± 0.12	0.987 ± 0.13
4 x 2	0.809 ± 0.03	1.043 ± 0.05	0.861 ± 0.04
4 x 4	0.492 ± 0.02	0.543 ± 0.04	0.512 ± 0.02
4 x 6	0.676 ± 0.06	0.762 ± 0.09	0.604 ± 0.10
5 x 2	0.891 ± 0.04	1.970 ± 0.10	0.841 ± 0.07
5 x 4	0.765 ± 0.21	0.901 ± 0.23	0.754 ± 0.20
5 x 6	0.694 ± 0.13	0.859 ± 0.11	0.672 ± 0.06

Figure 7: Mean accuracy of three-layer perceptron training on "clean" data. (Mean $\pm$ std)

4.2 Three-layer perceptron for noisy time series

After settling upon the 4x4 structure being superior in the previous section a more realistic problem was posed for the network. Gaussian noise was added to the training data, first from a normal distribution with $\sigma = 0.05$ and then with $\sigma = 0.15$, this drastically increased the prediction error for the model as is visualized in figure 9.

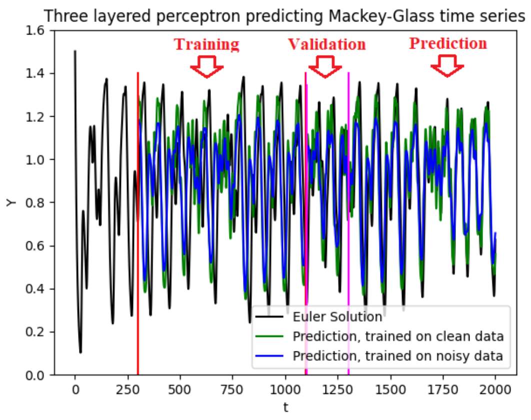
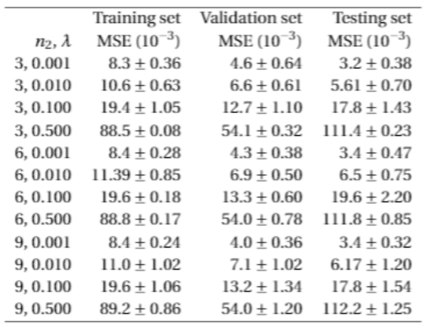
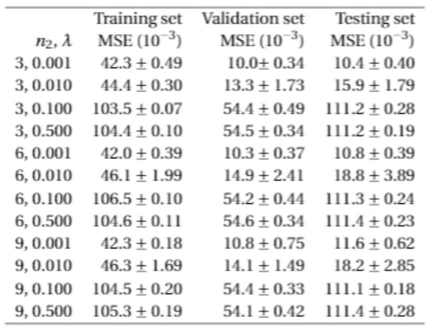


Figure 8: Time series prediction after both models has trained for same amount of epochs

Instead of using early stopping another regularization technique was used here, namely weight decay or l2 regularization. In practice this technique serves to de-incentivize complexity in the model, preventing it from fitting too well to the noise in the training set. The first hidden layer was given four neurons as decided upon in the previous section and the amount in the second layer was varied. The results of these tests can be seen in the tables below.Interestingly adding neurons to the second layer did not seem to vastly affect the accuracy in this case, a low value of lambda led to great improvement in the testing set accuracy compared to the model being ran without the weight decay, however increasing it did not improve it in either case. For the noisiest training set a high value of λ led to it taking the shape of the simplest possible model, a straight line through the mean value of the training set which is why the values for $\lambda = 0.1$ and 0.5 are almost the same.



	Training set	Validation set	Testing set
n_2, λ	MSE (10^{-3})	MSE (10^{-3})	MSE (10^{-3})
3, 0.001	8.3 ± 0.36	4.6 ± 0.64	3.2 ± 0.38
3, 0.010	10.6 ± 0.63	6.6 ± 0.61	5.61 ± 0.70
3, 0.100	19.4 ± 1.05	12.7 ± 1.10	17.8 ± 1.43
3, 0.500	88.5 ± 0.08	54.1 ± 0.32	111.4 ± 0.23
6, 0.001	8.4 ± 0.28	4.3 ± 0.38	3.4 ± 0.47
6, 0.010	11.39 ± 0.85	6.9 ± 0.50	6.5 ± 0.75
6, 0.100	19.6 ± 0.18	13.3 ± 0.60	19.6 ± 2.20
6, 0.500	88.8 ± 0.17	54.0 ± 0.78	111.8 ± 0.85
9, 0.001	8.4 ± 0.24	4.0 ± 0.36	3.4 ± 0.32
9, 0.010	11.0 ± 1.02	7.1 ± 1.02	6.17 ± 1.20
9, 0.100	19.6 ± 1.06	13.2 ± 1.34	17.8 ± 1.54
9, 0.500	89.2 ± 0.86	54.0 ± 1.20	112.2 ± 1.25



	Training set	Validation set	Testing set
n_2, λ	MSE (10^{-3})	MSE (10^{-3})	MSE (10^{-3})
3, 0.001	42.3 ± 0.49	10.0 ± 0.34	10.4 ± 0.40
3, 0.010	44.4 ± 0.30	13.3 ± 1.73	15.9 ± 1.79
3, 0.100	103.5 ± 0.07	54.4 ± 0.49	111.2 ± 0.28
3, 0.500	104.4 ± 0.10	54.5 ± 0.34	111.2 ± 0.19
6, 0.001	42.0 ± 0.39	10.3 ± 0.37	10.8 ± 0.39
6, 0.010	46.1 ± 1.99	14.9 ± 2.41	18.8 ± 3.89
6, 0.100	106.5 ± 0.10	54.2 ± 0.44	111.3 ± 0.24
6, 0.500	104.6 ± 0.11	54.6 ± 0.34	111.4 ± 0.23
9, 0.001	42.3 ± 0.18	10.8 ± 0.75	11.6 ± 0.62
9, 0.010	46.3 ± 1.69	14.1 ± 1.49	18.2 ± 2.85
9, 0.100	104.5 ± 0.20	54.4 ± 0.33	111.1 ± 0.18
9, 0.500	105.3 ± 0.19	54.1 ± 0.42	111.4 ± 0.28

Figure 9: Mean accuracy of three-layer perceptron training on data with Gaussian noise. (Mean $\pm$ std) Left table : $\sigma = 0.05$, right table: $\sigma = 0.15$

5 Final remarks

The lab helped elucidate many aspects of how to implement a neural network in practice. For example the shortcomings of the single layer perceptron model and need of a bias was made very clear, but also the capabilities of a rather simple model to make rather accurate regression and classification. The breadth of different problem formulations was also appreciated since it exposed us to different ways of applying a neural network model.

Since batch learning works by summing up many sample gradients, it is hard to make comparisons with sequential learning unless you normalize the learning rate (or, equivalently, make sure the divide the gradient with the number of training points). It might be easy to miss this detail when one is implementing the network for the first time, since there are so many things to think about. Perhaps this could be included as a "heads-up".