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

Clara Tump, Romina Arriaza Barriga and Konstantinos Saitas - Zarkias

January 2019

1 Main objectives and scope of the assignment

The intended outcome of this assignment was to provide a more thorough understanding of how Artificial Neural Networks work and how to work with them to perform machine learning tasks in practice. Our major goals in the assignment were:

- to understand single and multi-layer networks, the different learning rules, and understand the differences and the limitations of each approach
- to implement and understand approaches for classification, data compression and function approximation tasks
- to gain experience on advanced features of neural networks such as regularization and early stopping.

In this assignment we implemented different networks for different tasks and evaluated and analysed their performance. In the first part of this assignment the learning models were implemented from scratch. In the second part, we focused on more practical aspects of training neural networks, thus a pre-installed library was used.

2 Methods

For the results reported here, Python 2.6 and 3.7 was used. section 4 (Part II) makes use of the keras library in Python.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

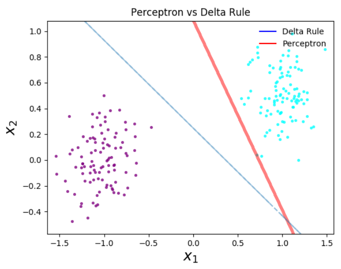
3.1.2 Linearly separable data

For this part, a simple two-dimensional data set was created. It has two classes (-1: purple, 1: cyan) that are linearly separable. This will be used as an example in order to reflect the capabilities of a single-layer perceptron in different aspects.

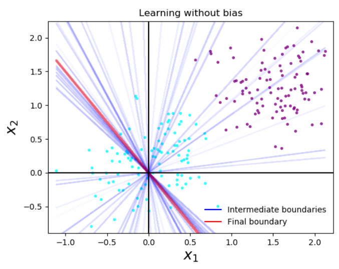
In order to train the single-layer network we used two different learning algorithms, which are responsible on how the weights are updated in each iteration. In Figure 1a, a comparison is presented on the perceptron and delta learning rule. We can see that both learning rules provide a correct decision boundary, but the delta rule's boundary also maximizes the distance to the clusters. This is because the perceptron learning rule terminates whenever all points are classified correctly, where the delta rule continues to minimize the mean squared error.

From the experiments implemented above, we observe that the convergence rate of the algorithms heavily depends on the learning rate. With appropriate values, both algorithms converge in around 10 epochs. Moreover, the random initialisation of the weights can affect the convergence rate of the algorithm. After some experimentation, we conclude that initialising the weights with a zero mean and a small variance (< 0.2) helps to converge faster. The sigmoid activation function is the steepest in the range closest to 0, hence this initialization causes the learning to be fast in the beginning.

Figure 1b shows the influence of bias in learning. It shows that bias is needed for classification with a decision boundary that does not go through the origin.



(a) The decision boundaries found through perceptron and delta rule (found with max. 100 iterations of batch learning)



(b) Not separable without bias, decision boundary needs to be shifted.

Figure 1: Both with a perceptron which learns with delta rule, 200 points and learning rate 0.2. In blue the transition boundaries along each epoch and in red the final boundary.

3.1.3 Non Linearly separable data

For this part of the assignment, non-linearly separable data clusters were created. As can be seen in Figure 2a the single-layer perceptron cannot classify this perfectly, independent of the used learning rule. The figure shows clearly how the perceptron rule minimizes the misclassification error and the delta rule minimises the mean squared error. In figure 2b 25% of the data of the right cluster of class 1 was removed. As a result, there are more samples from class 1 on the left cluster, and both learning rules make use of this by sacrificing misclassification on the right, to correctly classify the larger number of data points on the left.

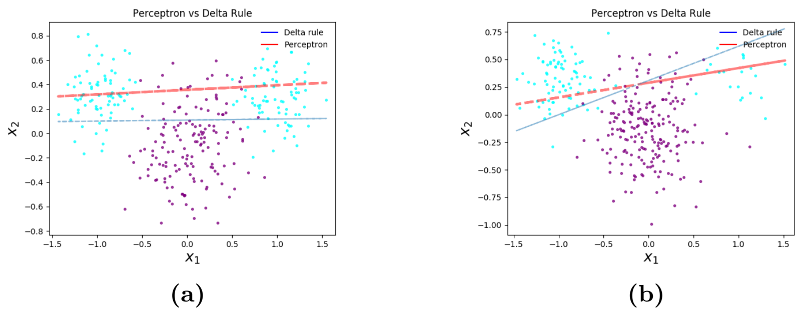


Figure 2: (a) Data of class 1 (cyan) equally distributed over 2 clusters. (b) data removed from class 1 on the right cluster

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

The same setup of data was tested in a two-layer perceptron network. The hidden layer had 10 nodes and 1 output node in the output layer. The two-layer network was able to classify better than the single layer since its decision boundary could be a non-linear one. Due to the overlapping nature of the data the network was still not able to achieve a 100% accuracy.

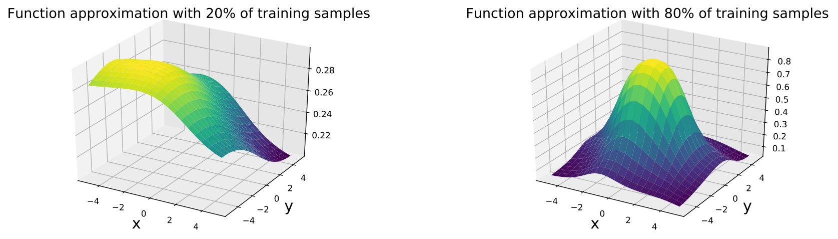
3.2.2 The encoder problem

The auto-encoder receives an 8x8 input matrix, containing 8 data samples with 8 dimensions each, and has a 3-node hidden layer. The input and output training data is identical, thus the goal is that the auto-encoder learns to map inputs to themselves. It can be seen that the mapping reaches perfect accuracy in most cases. However, due to the random initialisation of the weights, the model gets stuck in a local minimum in some runs, and gives an output which does not match the input.

Because the hidden layer is 3-dimensional, the model learns to represent the 8-dimensional data in a 3-dimensional space. Each node in the hidden layer can be either positive or negative, and there are 3 such nodes. In this way, the hidden layer can represent 2^n different one-hot encoded inputs, where n is the number of nodes in the hidden layer. In principle, the values of the nodes of the hidden layers are compressed versions of the input vectors. Thus, auto-encoders can be simply used for data compression and generalisation. Moreover, the conversion from input vector to the hidden layer encoded in the weights signify a projection onto the most important features of the input. This property can be used by, for example, using unlabelled data to learn a latent representation of the input data. This is done in the pre-training of neural networks.

3.2.3 Function approximation

In this section, we approach a different kind of problem with our two layer network. We create a dataset of a simple Gaussian distribution and given the X, Y axis as input we try to predict the values in Z (the actual values of the Gaussian). A comparison of the performance of the network with different sizes of sample training data is presented in figure 3a. Moreover, we run experiments with different number of hidden nodes and observed that very few nodes (2-3) were unable to converge to a correct solution compared to more nodes (19 as in the case of 3b). Finally, by using momentum during training we managed to converge faster (in less epochs) to the same solution.



(a) **(b)**

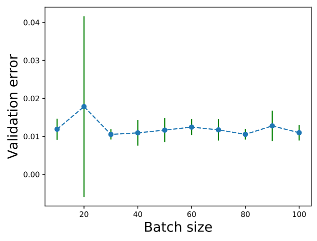
Figure 3: Comparison of performance with different size of training data.

When comparing very few nodes (2-5) with many, we can observe that they are not performing as well, since it is harder to map a representation of the data in so few weights. On the other hand, when we use 25 nodes the convergence rate had a slight decrease since a lot of weights are being adjusted even for very similar data points.

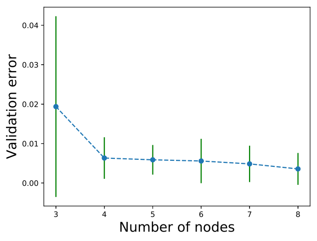
4 Results and discussion - Part II *(ca.2 pages)*

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

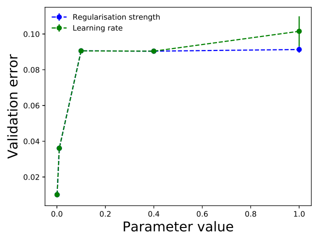
Several models where trained under different conditions. The batch size, number of hidden nodes, learning rate and regularisation strength were varied. In order to pick each parameter, the model was trained several times using early stopping. The performance was evaluated based on the validation error. Figures 4 (a) - (c) display examples of this analysis. We found that the optimal model has the following characteristics: 80 samples per batch, 8 nodes in the hidden layer, a regularisation strength of 0.001, a learning rate of 0.001, 700 training samples and 300 validation samples. In figure 4 (d) shows the resulting prediction for the test data (200 samples) with a training and validation error of 0.0136 and 0.0089 respectively.



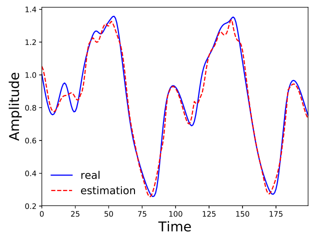
(a)



(b)



(c)

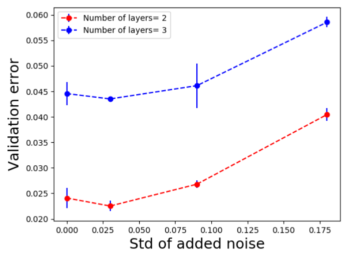


(d)

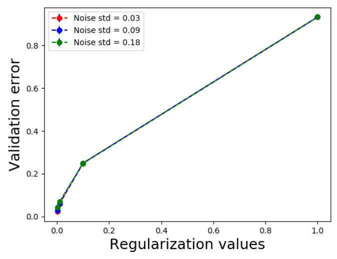
Figure 4: (a), (b), (c): validation error (mean and standard deviation) for different values of the parameters. (d) Prediction (test data) obtained with the final model.

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

When we add Gaussian noise to the train data we can observe an interesting effect on the validation error. By adding small amounts of noise ($\sigma = 0.03, 0.09$) the validation error slightly decreases. This can be attributed to the fact that the model is able to generalise more and not overfit on the training data. On the other hand, by adding even more noise the validation error starts to increase again as the model starts to become more generalised than we want. This effect can be seen in figure 5a.



(a) Validation error on two- and three layer networks when training with noisy data.



(b) Validation error on different values of the L2 regularizer.

In Figure 5a also shows that the two-layer network performs consistently better than the three layer network, independent of whether and how much noise is added. In figure 5b, different values of λ of L2 regularisation are presented with respect to the validation error in a three-layer network with 7 nodes in each hidden layer.

From the experiments run above it is clear that adding more hidden nodes to a neural network increases the computational cost of training. Even more, adding a third layer to a two-layer network increases the cost exponentially. This makes sense since in each iteration another level of equations needs to be computed for each layer.

5 Final remarks

This lab provided us a good opportunity to explore different aspects of the process of building a neural network and thus acquiring a more intuitive and insightful understanding of them. It was interesting to come in contact with applications of neural networks that we did not know existed, such as the encoder. We spent a lot of time trying to find a bug in the linear algebra calculations of backpropagation. This was not very useful, as we believe we could have used this time for fundamental understanding of other concepts. However, it is probably good to have implemented this learning method from scratch at least once.

We learned a lot about the comparison between single-layer networks two- and three-layer networks. But an open question for us is still how this all generalises to very deep networks, with considerably more hidden layers than in this assignment. Finally, many more experiments were conducted during this assignment than what are presented but due to the restriction of number of pages a considerate amount graphs and tables were left out.