Artificial Neural Networks and Deep Architectures, DD2437

A short report on lab assignment 1

Learning and generalization in feed-forward networks — from perceptron learning to backprop

Balint Kovacs (balintk@kth.se), Victor Castillo (vcas@kth.se),
Jun Zhang (junzha@kth.se)

January 28, 2019

1. Main objectives and scope of the assignment

Our major goals in the assignment were:

•to design and apply networks in classification, function approximation and generalization tasks

•to identify key limitations of single-layer networks

•to configure and monitor the behavior of learning algorithms for single-and multi-layer perceptrons networks

•to recognize risks associated with backpropagation and minimize them for robust learning of multi-layer perceptrons.

In order to complete the objectives, we generated data with certain characteristics that allow testing the theory of basic neural networks, and the assumptions that the same techniques can be applied to data. We tested several conditions for the different alternatives and observed the behavior that is reported on this document.

2. Methods

We used python as the programming language, with the Numpy and Matplot libraries for the first part of the assignment and Tensorflow for the second part.

3. Results and Discussion - Part I

3.1. Classification with a single-layer perceptron

In this part, we generate both linearly separable and nonlinearly separable datasets in order to compare the learning process of the delta rule and perceptron rule. The first tests were classifying with linearly separable data and we observed that:

•The initial size of the weights is relevant for the algorithms to converge. It has to be values that are near zero, otherwise, the weight update can easily overshoot if the learning rate is not

small enough (0.000001). The boundaries for linearly separated data are found by all options, but for non-linearly separable, it tends to correctly classify one class, while having a higher error for the second one. (**Figure 1**)

•Using the delta rule, convergence is achieved faster (with fewer epochs) than with perceptron rule. The perceptron rule on the other hand often leads to higher accuracy. We observed that the error is much smaller in the second case (discrete) and weights are updated in smaller amounts than with the delta rule (continuous values). **(Figures 2 and 3)**

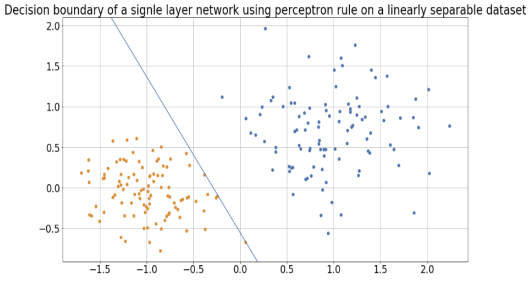
•Sequential and Batch learning methods have similar convergence rates if the learning rate is very small, but in the first case weights are updated with each sample and the boundary changes constantly. If data is not shuffled after each epoch, there is a bias in the weights after each epoch toward the last values in the dataset in the sequential case, while the batch mode is unaffected by the order of the data. Also, while the accuracy for batch and sequential is similar, the MSE for the second has a decreasing trend, but it oscillates more. **(Figure 4)**

•If the learning rate is too large (1 or 0.1), batch mode delta rule learning will end up changing from all positive predictions to all negative predictions, and weights that are very large in magnitude, with an accuracy of around 50%, but not converge. Sequential, however, can converge fast being more stochastic.

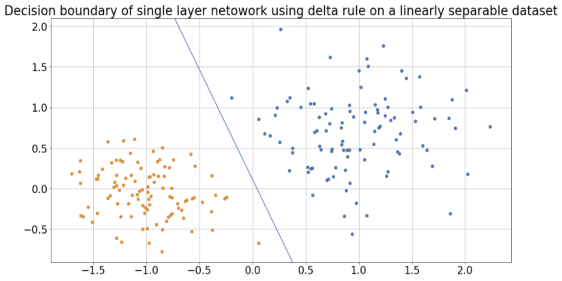
•If learning is very small, needs many epochs to converge, but sequential and batch behave almost exactly the same behavior per epoch.

•When the data is not linearly separable, delta rule can classify them better(around 70% to 85% correct) than perceptron rule (around 50%, random chance), which will continue training infinitely. The delta rule decreases the error value to local minima.

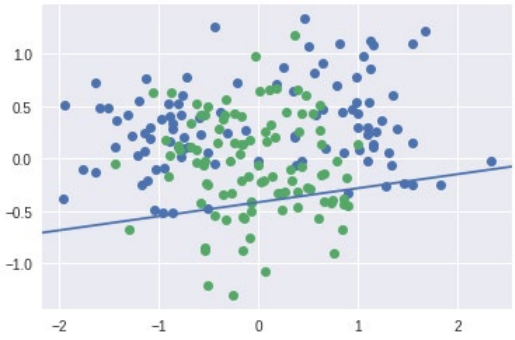
•By removing the bias, we found that the perceptron converges in the cases when the means of the two classes are on the same line with the origin and in the same quadrant



(a)



(b)



(c)

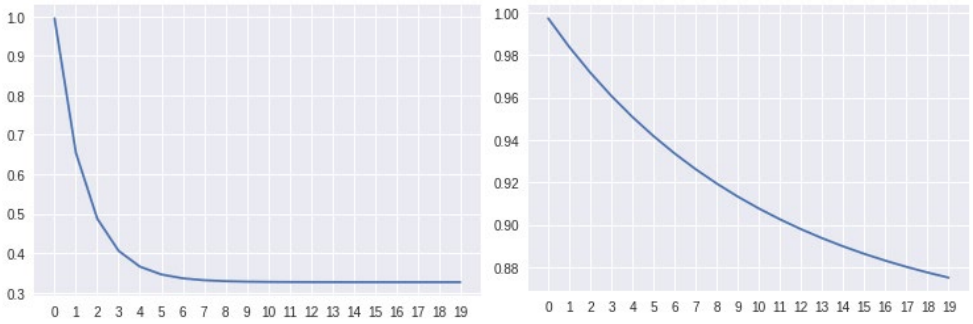
(a) Plot of linearly separable data and the boundary of batch learning (perceptron rule)

(b) Plot of linearly separable data and the boundary of batch learning (delta rule)

(c) Predictions for non linearly separable data and the boundary of batch learning (perceptron

learning)

Figure 1: Generated data used for the assignment

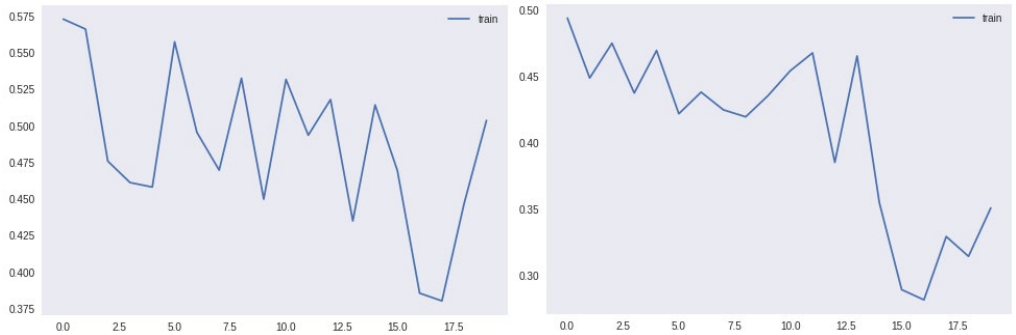


(a) for linearly separable data (b) for non-linearly separable data

Figure 2: Plot MSE with delta learning (Batch)

(a) for linearly separable data (b) for non-linearly separable data

Figure 3: Plot MSE with perceptron learning (Batch)



(a) with delta rule learning (b) with perceptron learning

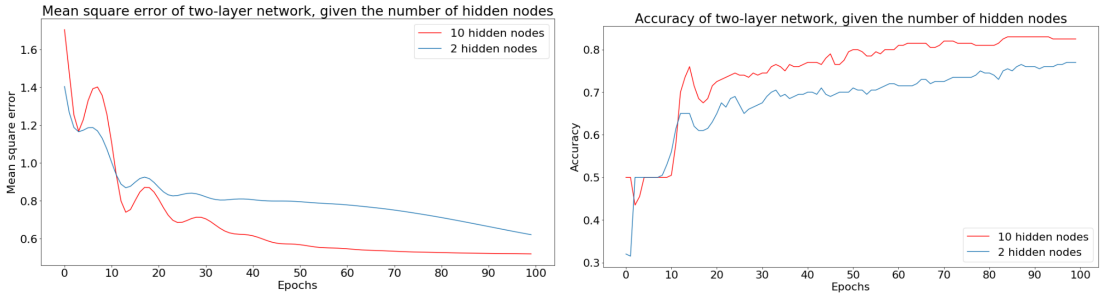
Figure 4: Plot MSE for linearly separable data (**Sequential**)

3.2. Classification and regression with a two-layer perceptron

3.2.1. Classification of linearly non-separable data

For the two-layer network, we can see the performance comparison between using 10 nodes and 2 nodes on **figure 5.** We found that a higher number of hidden nodes can make the model reach the convergence faster, and the more complex our classifier can be, thus the more complex data it can fit. However, if we use too many hidden nodes for a simple problem, it can

cause overfitting.



(a) (b)

(a) MSE of the two-layer networks, 10 nodes (red) and 2 nodes (blue)
(b) Accuracy of the two-layer networks, 10 nodes (red) and 2 nodes (blue)

Figure 5: Comparing two layer networks

When we select 25% of each class as test samples, the performance is similar to the original case. When 50% of class B will be the training set, the performance is high with a simple model on the test set but low on the training set(f.e. hidden_nodes=5, epochs=20, learning_rate=0.001, alpha=0.9), with a complex model(f.e. hidden_nodes=5, epochs=100, learning_rate=0.001, alpha=0.9), the accuracy is high on the training set and low on the test set

3.2.2. The encoder problem

In this exercise, we tested that using batch learning mode, the network always converges if the learning rate is big enough (in relation to the number of epochs and the momentum).
The hour-glass (8-3-8) shaped network is used in this case to train the network with input and output patterns that are identical, so the network itself represents a projection from the 8 8-dimensional space of the inputs into the three-dimensional space of the internal neurons, and then back to 8-dimensional space.
When using a hidden layer with a size of only two, we need more epochs to train a network to fit our data, or bigger step size, but it's also possible to encode the information. With the correct configuration, perfect accuracy can be reached, as shown in **Figure 6**.
Autoencoders seem really efficient for compressing data, or can be useful for encrypting messages as well, being able to recover the original information with the trained model, dinding compact coding of mostly sparse data.

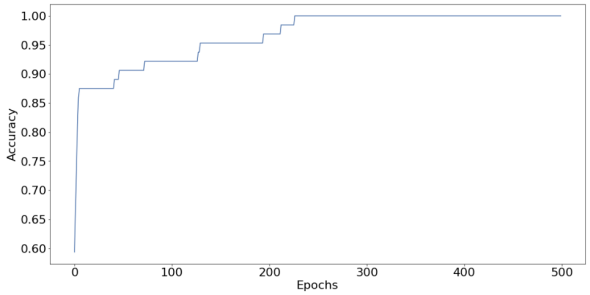


Figure 6: Perfect accuracy on encoding

3.2.3. Function approximation

For this exercise, we trained the two-layer network in order to visualize the approximation of the true continuous function in a higher dimension to a lower one. The data used was generated following the provided instructions (using a Gaussian function) and we tested

different number of nodes in the network for the training process. In the **Figure 7** we show the Gaussian function provided.

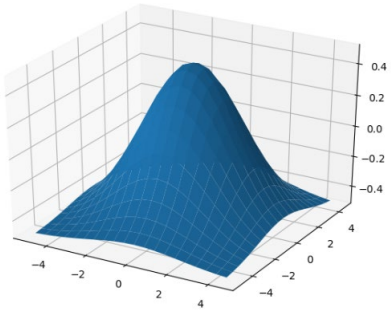


Figure 7: true function

When there are very few nodes, the representation of the true function is limited. In the the 1 node case, the result is a monotonous function that defines a plane and saturates at the extremes. As the number of hidden nodes increases, the resulting function is closer to the original, as can be seen in **Figure 8**. However, as the number of nodes increases above 20 nodes, it becomes distorted, which is a sign that it's overfitting to the data. It is possible to increase the convergence speed using regularization mechanisms to prevent overfitting.
If a subset of the data is used, the approximated functions vary in comparison to the true function, but they change randomly, depending on the sampled subset**.**

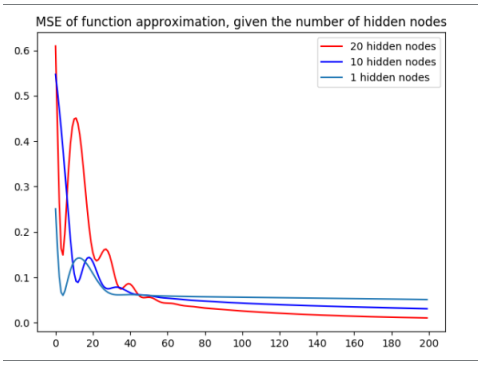


Figure 8: MSE of predictions with different number of hidden nodes

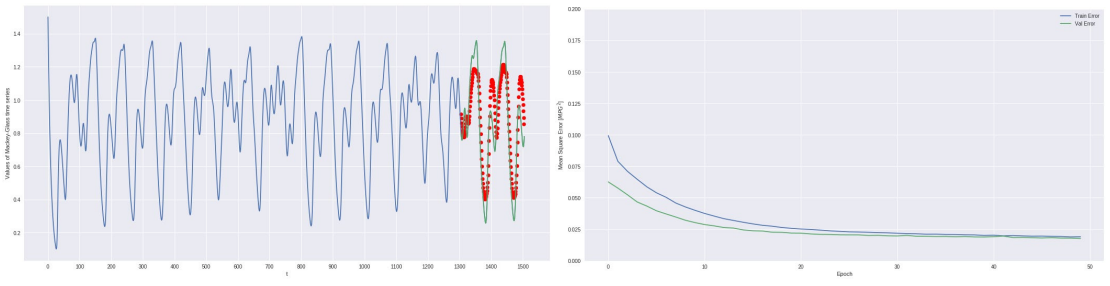
4. Results and discussion - Part II

We have generated the Mackey-Glass time series as explained in the assignment for chaotic prediction, splitting the data into three groups (training, validation and testing) and trained a multi-layer network with different parameters to observe the results.

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

The best parameters found for the time-series are a 8-1 network training using gradient descent and sigmoid activation function. Also the chosen regularization method is weight decay (L2 regularization) and early stopping was defined for 10 epochs without improvement on the

validation set MSE. When the learning rate is too small, the error stays constant after a few number of epochs, pointing towards a premature convergence. On **Figure 9** the predictions and MSE using the described parameters can be found. When we use a high value for weight decay (very strong regularization, e. g. lambda=10), the validation performance is constant, the value of the weights stay close to zero and the prediction approximates a constant around the mean of the function, as can be seen on **Figure 10**. The performance of the network on the test is not affected in a significant way by the number of nodes. With 8 nodes, convergence is faster, but overfitting on the training data is bigger. More nodes allows the model to have higher capacity, but needs more regularization to prevent overfitting on the training data and generalize better. The predictions of our best performing model can be seen on the top left corner of **Figure 10**. We used 8 hidden nodes for this network with 0.1 learning rate and 0.01 lambda for weight decay. It produced an MSE of 0.023.

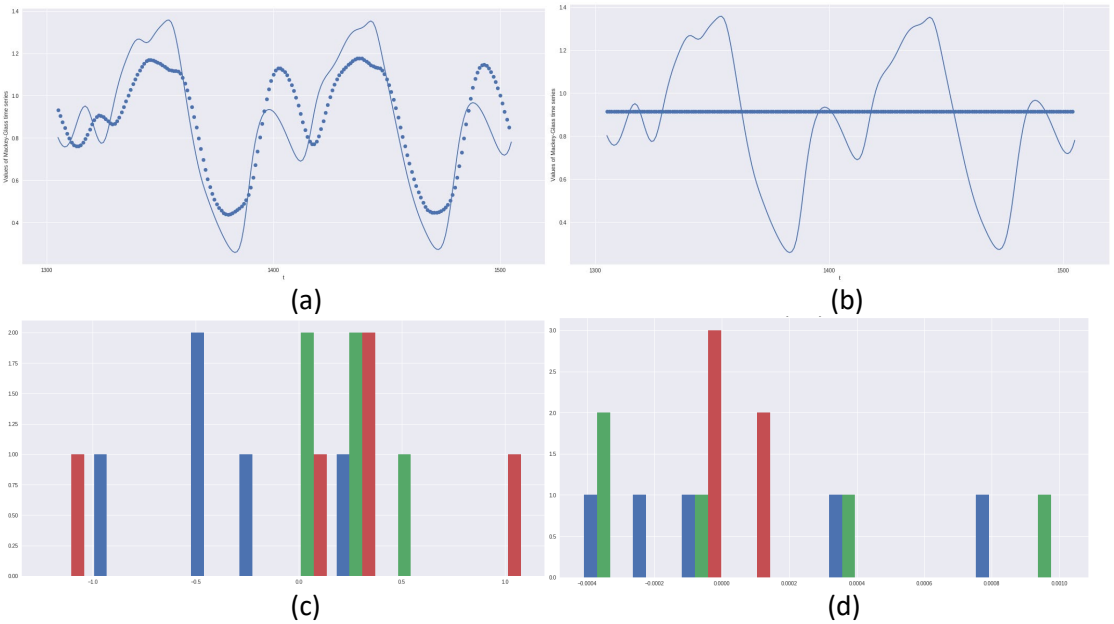


(a) (b)

(a) Training (+Validation 20%), Testing data and Predictions, Two layers, no noise, no regularization

(b) Training (blue) and Validation MSE over epochs

Figure 9



(a) (b)

(c) (d)

(a) Two layers, no noise, no regularization

(b) Two layers, no noise, high regularization (lambda = 10)

(c) Weight histogram with no regularization (range -1 to 1)

(d) Weight histogram with lambda = 10 (range -0.0004 to 0.0010)

Figure 10

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

In this part, we appreciate the effects of noise for generalization of the network. If the noise is high (0.18) and there is no regularization, the prediction tries to follow the noise very closely, meaning a higher variance. However, the predicative performance becomes worse with more noise. With enough regularization (more is needed as the noise increases), the prediction becomes smoother and similar to the true function, even if the MSE increases and the weights are kept small. As is the case with two layers, too much regularization and the prediction is almost a flat line around the mean of the function.

The intuition here is that regularization could be helpful for noisy data sets. A two-layer network have a natural limit to the complexity that it can handle which helps it prevent overfitting with the noise and regularization becomes much more important in the three-layer network. In **Figure 11** we show the prediction for the three-layer network for the test data with noise with a standard deviation of 0.18 and how the regularization smoothens the curve. Our best performing three layer network reached 0.032 mean square error on the dataset with medium noise, which is worse by roughly 50% compared to the 2-layer one on the clean set. Computational cost (time) increases with the number of nodes*layer. A more complex network takes more epochs to train.

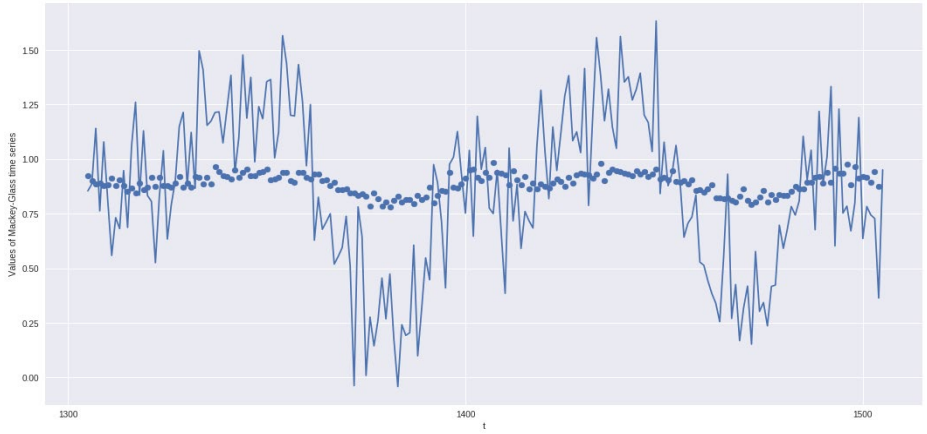


Figure 11: Three layers (5-5-3), noisy data (0.18), L2 regularization (0.1)

5. Final remarks

The insights gained in this lab from understanding how different parameters affect neural networks was invaluable. Some of the most interesting points to learn were how to closely approximate a true function without overfitting by using regularization, how noise can affect predictions and how important the representation of the data is for the algorithms to perform better.