Kungliga Tekniska Högskolan

DD2437 Artificial Neural Networks and Deep Architectures

Lab 1: Perceptron rule and robust backpropagation in multi-layer perceptrons

Quentin LEMAIRE
Quentin VECCHIO
Kévin YERAMIAN

February 22, 2018

1 Introduction

This exercise is about supervised (error-based) learning approaches for feed-forward neural networks, both single-layer and multi-layer perceptron.

1.1 Aim and scope

The goal of this lab is to learn how to design artificial neural network and more precisely how the delta rule and perceptron rule work.

During this lab we learned to :

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

2 Assumptions and tools

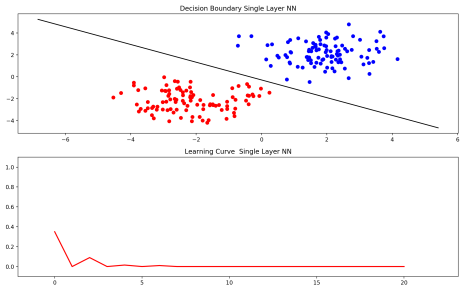
Our assumptions for this lab were that the perceptron and single-layer neural network were able to separate linear data. The multiple-layer neural network was able to separate not linearly separable data like the XOR problem. For the part 2 of this lab, we used *sklearn* to create and manipulate artificial neural network.

3 Results

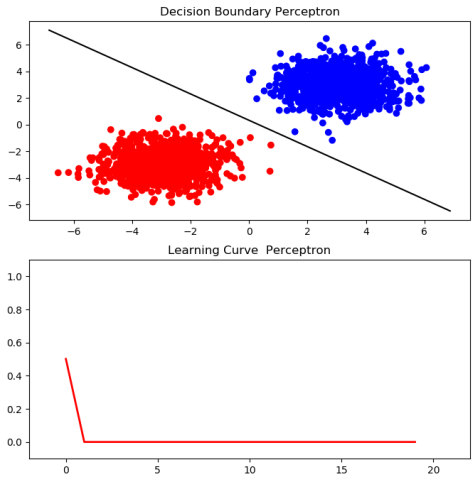
3.1 Part 1

3.1.1 Classification with a single-layer perceptron

Our assumptions for this first part were that a single-layer perceptron was able to separate linear data only. We experimented two learning rules in this part, the delta rule and the perceptron rule. Both methods converged after a few number of epoch. We also tried several parameters by changing the learning rate and the batch size.

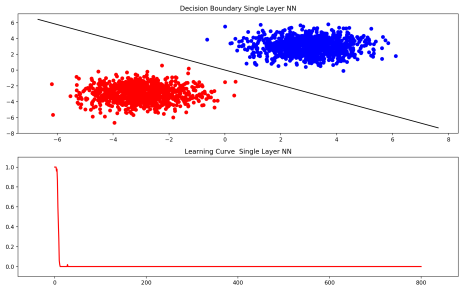


(a) single-layer

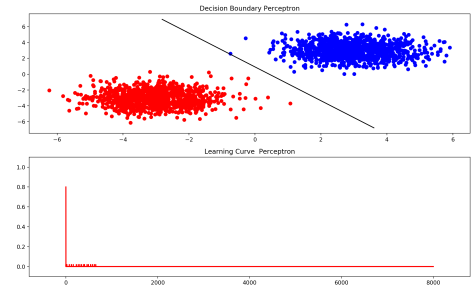


(b) perceptron

In both case we see that the convergence is very fast and we are able to separate the two set. The batch size doesn't influence on the result that much.

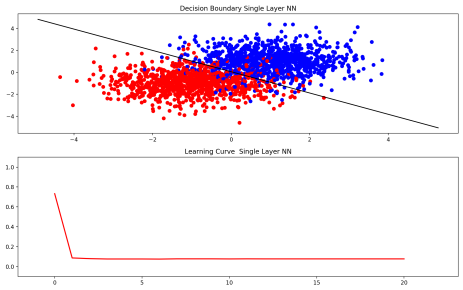


(c) single-layer BatchSize =50

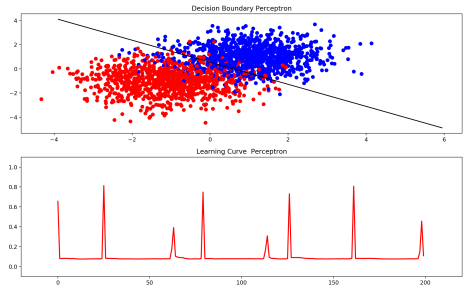


(d) perceptron BatchSize =50

We also tried to separate non linearly separate data, but as we thought we can't separate this kind of data with a perceptron or a single layer neural network. In this case, we see that the single layer converge quickly like the perceptron but the perceptron is less stable. Indeed, we can notice some spike due to the lack of stability of the method.

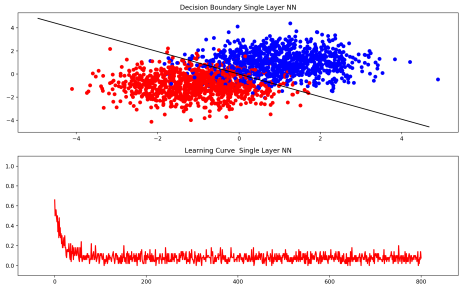


(e) single-layer BatchSize =50

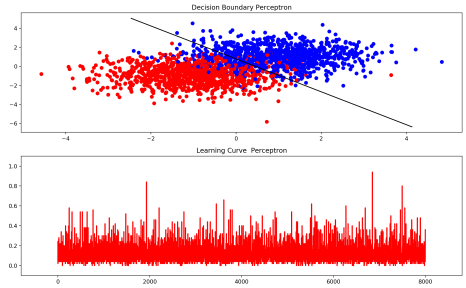


(f) perceptron BatchSize =50

In this case, we used batch learning. We see that the perceptron isn't stable at all. It's normal, the method changes the weights only if there is a mistake of classification. So for small batch, the error (distance) doesn't count. For each batch the perceptron overfits to be able to predict the batch only. The single layer is more stable we see some spike due to the batch which is overfitted but the method doesn't change the weights that much even if the classification is false. It's only fix the weights according to the distance.



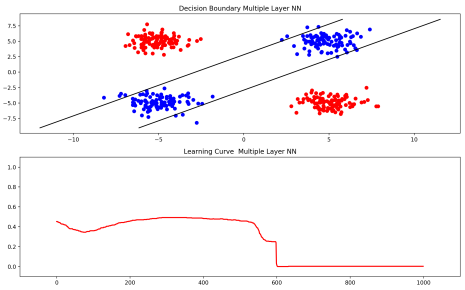
(g) single-layer BatchSize =50



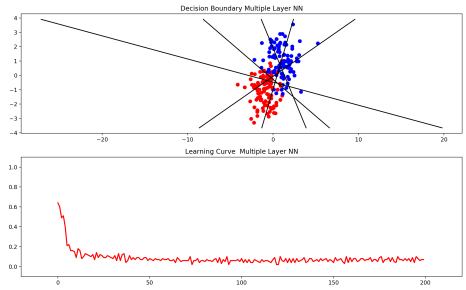
(h) perceptron BatchSize =50

3.1.2 Classification and regression with a two-layer perceptron

For the two-layer perceptron, we tried to separate not linearly separable data. We generated different data sets. We also used the two-layer perceptron to resolve the XOR function. We tried to separate complex data with different architectures, different numbers of neurons in the hidden layer, and with different values of learning rate and batch size.

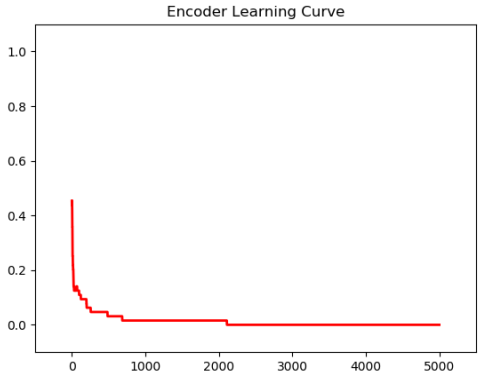


(i) Two-layer LayerSize=2

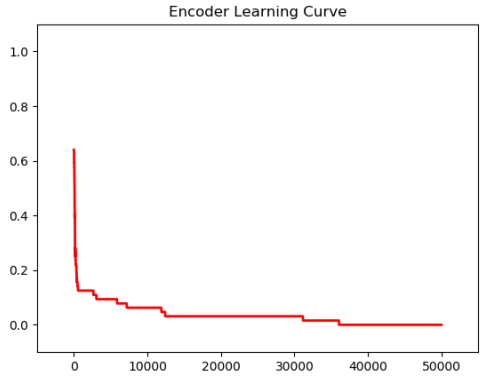


(j) Two-layer BatchSize =100 LayerSize=5

We also used our two-layer neural network to create an encoder. The architecture was given, we used a 8-3-8 neural network. We experimented several parameters to find the best configuration.



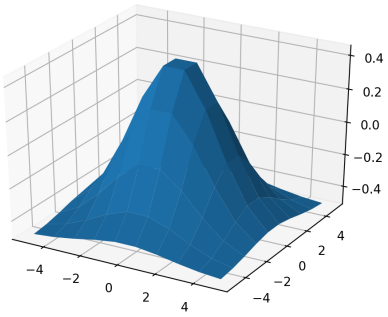
(k) Encoder Learning rate=0.01



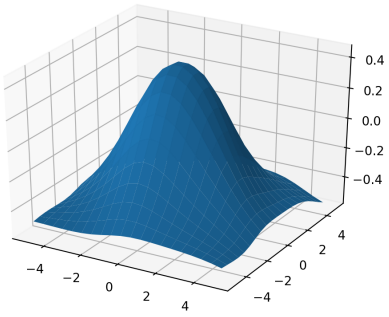
(l) Encoder Learning rate=0.001

3.1.3 Function approximation

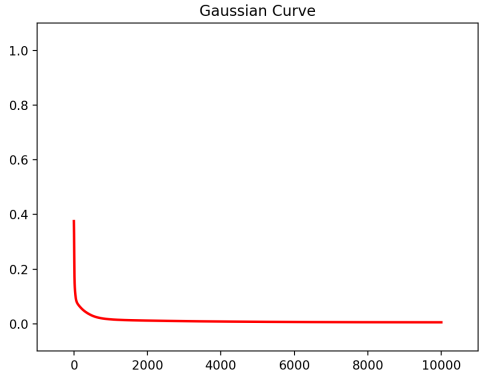
The goal was to approximate a function. We had in input (x,y) and z in output. We used two-layer neural network to approximate it.



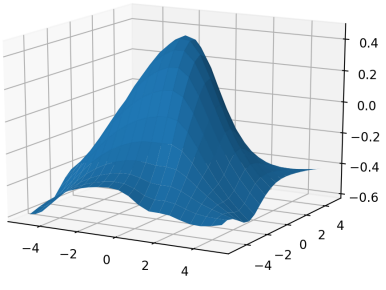
(m) True shape of the function



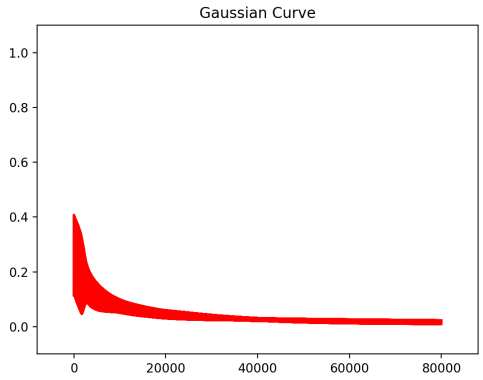
(n) Our result with 10 neurals, 100 samples and lr = 0.0001



(o) Lose function



(p) Approximation with 10 neurals , 50 batch and samples = 400 and lr=0.0001

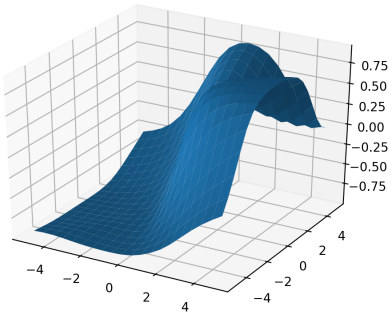


(q) Lose function

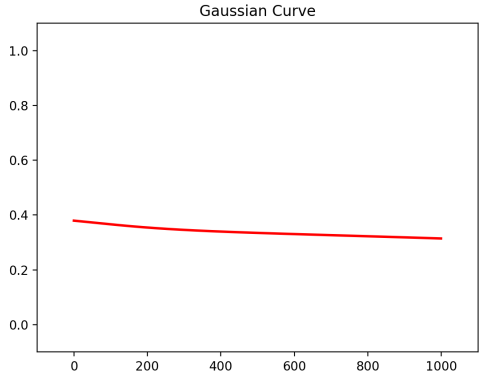
The batch approximation is fair but the lose function is chaotic. The result is close to the true shape but at the extremity is not fitted very well. A possible

reason is that we took more sample (400) this time and the batch size is only 50 so we overfit locally. We have also try with more neural, the result was close to the true shape but seem less smooth (more stiff on the extremity) probably due to some overfitting.

We also do the last part "Evaluate generalisation performance". The result was not as expected. Our approximation has a high lose function but the shape has some similarities. There is a possibility that we got an error or that we failed the manipulation but we got very good result for the simulation.



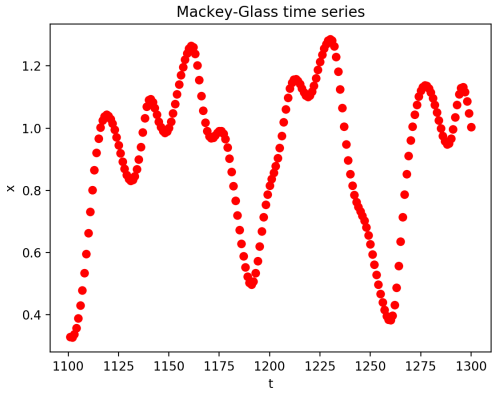
(r) Approximation with 20 neurals , n =20 and lr=0.0001



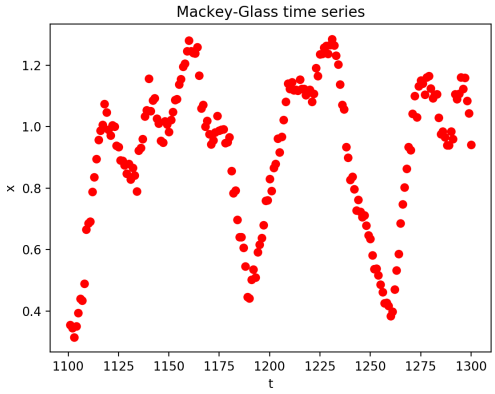
(s) Lose function

3.2 Part 2

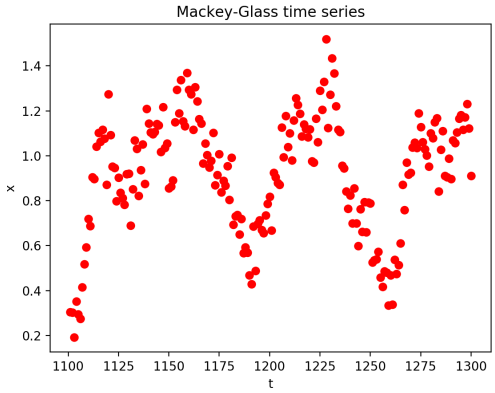
The aim of this part is to implement a multi-layer perceptron network for chaotic time-series prediction. Moreover, we have add noise to the data in some cases to have a more real use case.



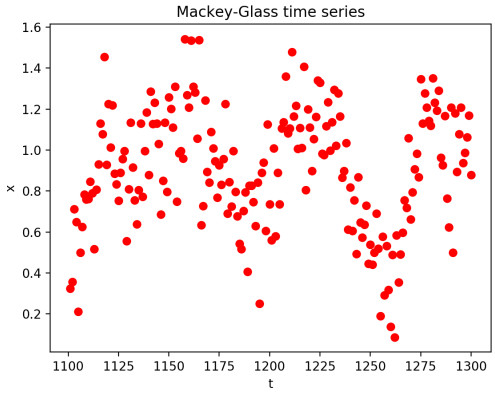
(t) noise = 0



(u) noise = 0.03



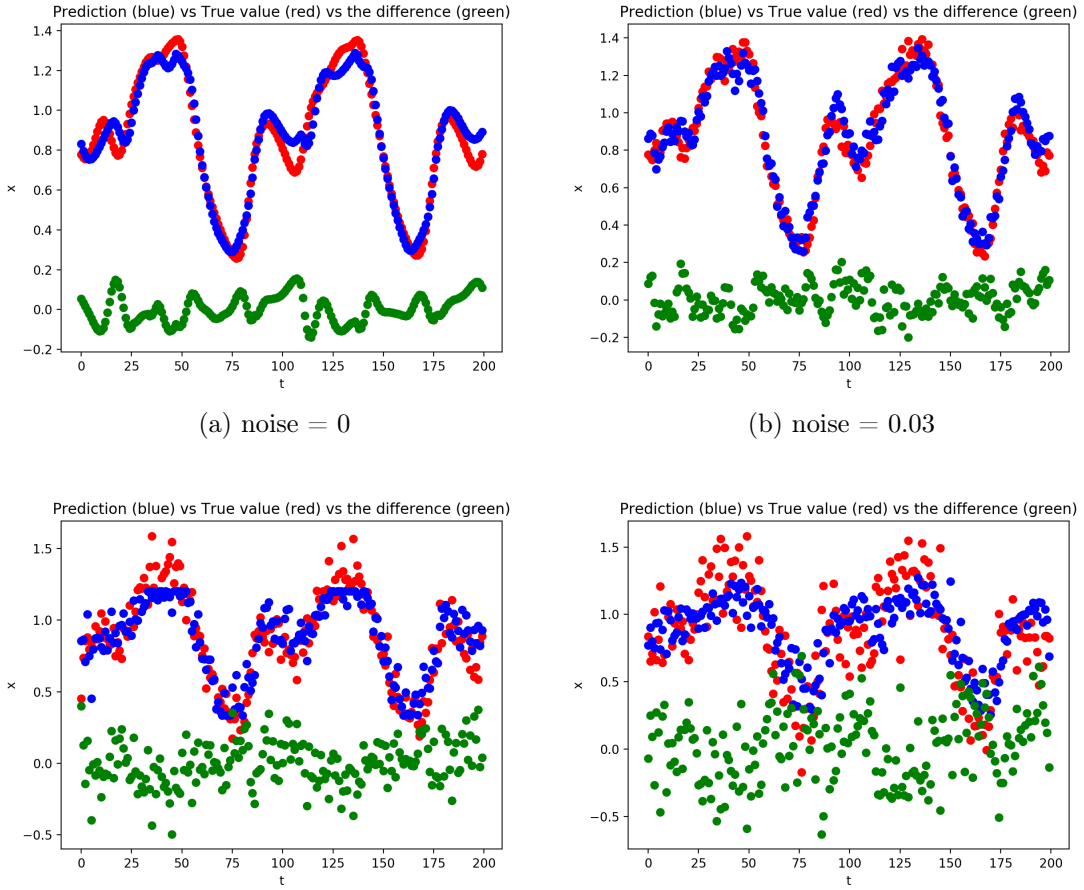
(v) noise = 0.09



(w) noise = 0.18

Figure 1: Sample of the Mackey-Glass time series for different variance of the noise

In the figure 1, we have plotted a sub-sequence of the used series for different versions of the noise. We can see that in the extreme case it is hard, even for a human, the recognize clearly the shape of the series.



(a) noise = 0

(b) noise = 0.03

(c) noise = 0.09

(d) noise = 0.18

Figure 2: Comparison between the true time serie (red) and the prediction (blue), the difference between the two is shown in green. A one-layer neural network was used for the first case and a two-layer neural network for all the other cases.

In the previous figure (Figure 2), we have plotted our prediction, the real values and the difference between both.

We can see that the result is quite good and the shape is well followed. When the noise is increasing, the result becomes less and less like the original function but still has a good prediction of the function with the noise.

Noise	learning rate	node number	momentum	alpha	early stopping	learning decay	MSE
0	0.1	6	0.9	0.00001	True	False	0.0027
0.03	0.1	6	0.9	0.00001	True	False	0.1002
0.09	0.1	6	0.9	0.00001	True	False	0.1331
0.18	0.1	6	0.9	0.00001	True	False	0.0713
0.03	0.1	5, 7	0.9	0.001	True	False	0.0063
0.09	0.1	5, 7	0.9	0.0001	True	False	0.029
0.18	0.01	6, 4	0.9	0.0001	True	False	0.073

Table 1: Parameters of our best networks for a 1-hidden layer network (noise 0-0.03-0.09-0.18) and for a 2-hidden layers network (noise 0.03-0.09-0.18).

In the table 1, we presents our best 1-hidden layer network for a 0 noise input. Then we test this network on a dataset with a bigger noise, the result is way worse. Then we try to put a 2-hidden layers network instead to have better results.

Something which is also interesting is to see the parameters of the worse result, to see which parameter can have a very important effect. The network with only 1-3 nodes have very bad results. We can also see that the best models have a momentum value.

Noise	Computation time 1 layer	Computation time 2 layers
0.03	0.0064	0.0339
0.09	0.0071	0.0155
0.18	0.0230	0.0584

Table 2: Computation time for different values of the noise and both 1 and 2 hidden layers network.

Finally, we took a look at the computation time of the different networks for different values of the noise. It was interesting to change the noise because it can delay the early-stopping and, therefore, it was increase the computation time.

We can see that for the same network (1 hidden layer), the computation time is effectively increasing the computation time.

We can also see that there is a clear difference between the 1 and the 2-hidden layer, there is at least a factor 2 between the two.

We have to keep in mind the fact that a computation time can vary a lot between two datasets due to the randomness and between two computers. The values are, therefore, not relevant but we have an estimation of the evolution between 2 parameters.

4 Reflections, open questions and conclusions

4.1 Part I

To conclude on the first path of the assignment, we learned that perceptron can only separate linearly data. There are two important rules of learning for this model, the delta rule and the perceptron rule. We also learned that a multi-layer network can separate not linearly data and approximate maths function. We also found useful applications of encoder with ANN.

We can also approximate very complex function with ANN. Our case was a 2D exponential function and in few iteration the network is able to fit with a complex shape.

4.2 Part II

To conclude on the second part of the assignment, we can say that it is pretty hard to find the best values for each parameter for this problem.

We have computed 800 networks to find the best parameters for the 1-hidden layer network but the best can change according to the randomness of the dataset (separation between the train set and the validation set for example), we should have done it several times to find the parameters with the bigger number of occurrence.

Unfortunately, it can be done for a very simple model like this one but it is not possible to test such many combination of parameters in complex the deep-learning networks used today for imagery for example.

Increasing some parameters as the number of hidden layers can be very important for the generalization of our network. We have seen that a 2 hidden-layer network has a better generalization that a 1-hidden layer.

The noise decreases the performances of our network but at the same time it is a good indicator of the robustness of our network.

Finally, we have also seen that the best network trained on a noised train set has a bigger alpha value which is the coefficient of regularization. The MSE on the validation set is better for a strongest regularization in this case because the network is less likely to overfeet the noise.