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

Mastafa Foufa, Samuel Leonardo Gracio, Martin Verstraete

October 12, 2018

Main objectives and scope of the assignment

Our major goals in the assignment were :

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

• to identify key limitations of single-layer networks

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

• recognise risks associated with backpropagation and minimise them for robust learning of multi-layer perceptrons.

Methods

We worked with Python and with a Google Colab environment. We only used numpy functions and matplot.

1.1 Classification with a single-layer perceptron

- **Linearly-separable data**

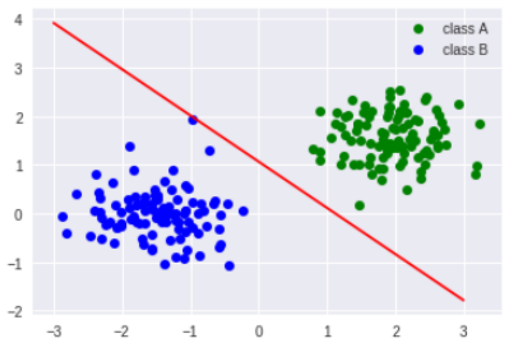


Figure 1: Delta rule

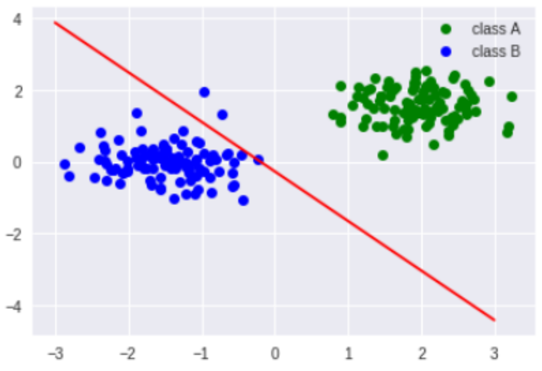


Figure 2: Perceptron learning rule

The delta rule is better than the perceptron rule, even for classifying simple, linearly-separable, patterns. When comparing sequential and batch mode for these data, both methods see their error converging to zero. Plus, the convergence does not seem to be faster with one mode.

When we remove the bias in the delta rule. The boundary goes through the origin, so the perceptron will not be able to classify the data if both classes are not on opposite sides around the origin.

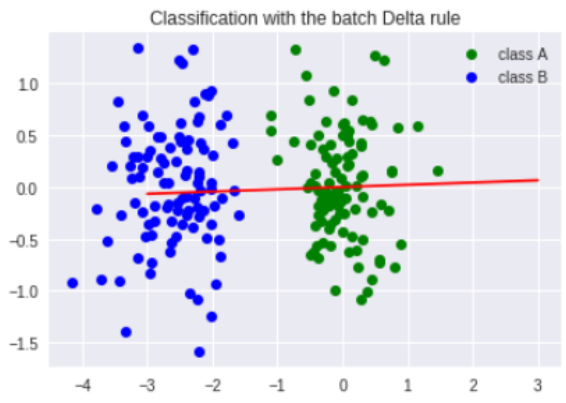


Figure 3

Here we can see that the perceptron cannot classify the data

- **Non linearly-separable data**

When the data are not linearly separable, the error does not converge to zero and some points are misclassified, whatever the mode we use.

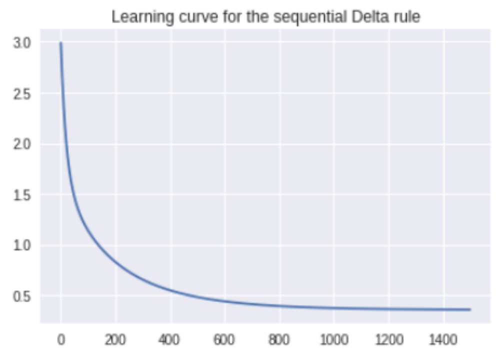


Figure 4

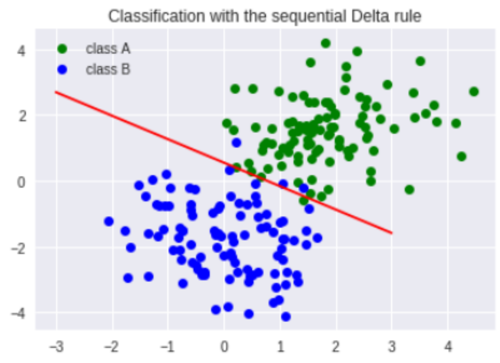


Figure 5

With even less linearly-separable data, the result is worse.

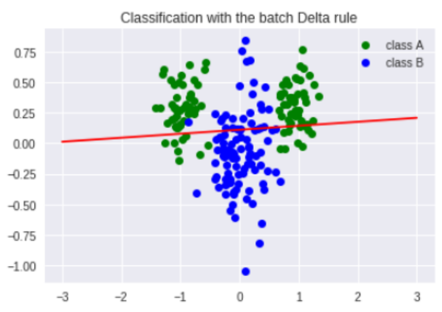


Figure 6

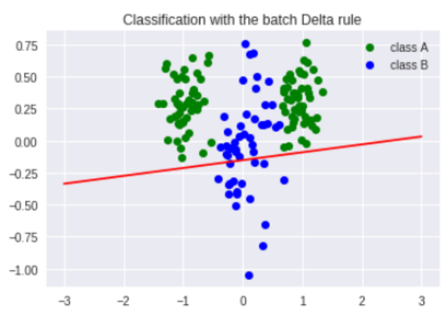


Figure 7: 50% of class B has been removed

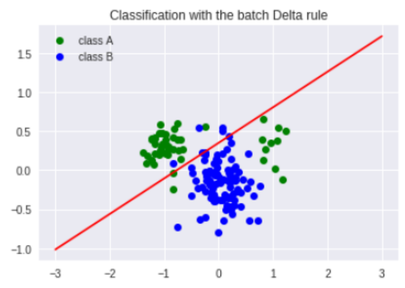


Figure 8

In the figure 8, we removed 80% of the left part of class A and 20% of its right part. We notice than the separation hardly takes the right part of class A into account.
All this study show that we need to use a two-layer perceptron to separate the data properly.

1.2 Classification and regression with a two-layer perceptron

1.2.1 Classification of linearly non-separable data

When using a two-layer perceptron with the same patterns, we immediately see that it is more to classify such non linearly-separable data:

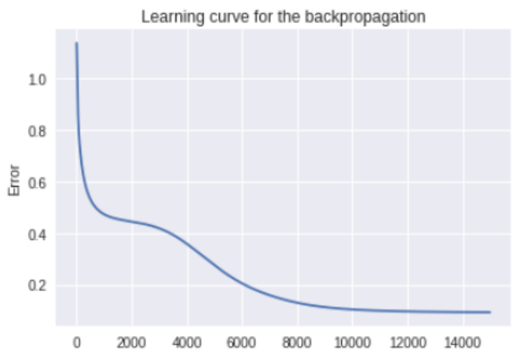


Figure 9

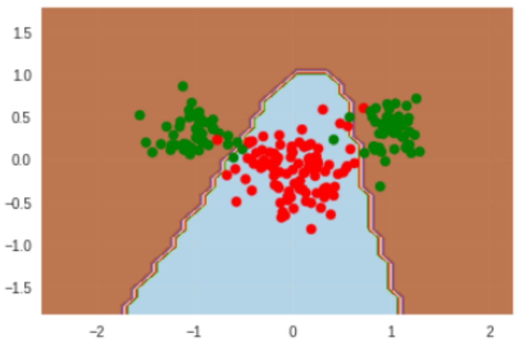


Figure 10

Let's study the influence of the learning rate eta and the number of hidden nodes.

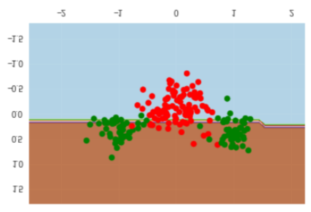


Figure 11

Just one hidden node takes us back to the one-layer perceptron

On the contrary, too much hidden nodes lead to overfitting ant that is not good for generalization. 15 hidden nodes seem to be a good trade-off between error minimization and generalization.
In addition, eta needs to be high enough to avoid getting stuck in a local minimum, but it cannot be too high or the error will diverge. We took 0.005.

Let's see the performance of the separation and the boundary in the four scenarios when we used the removed patterns as a validation set:

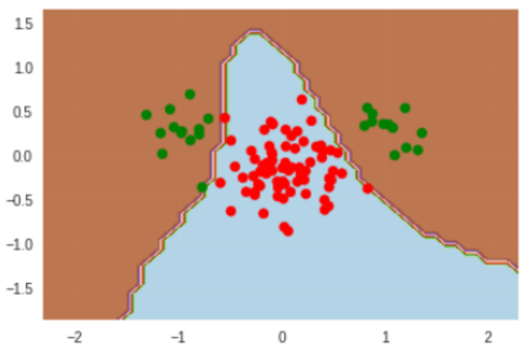


Figure 12 : First scenario
Number of misclassifications = 0.03

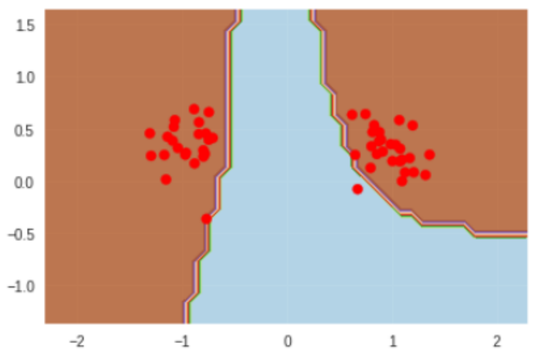


Figure 13: Second scenario
Number of misclassifications = 0.04

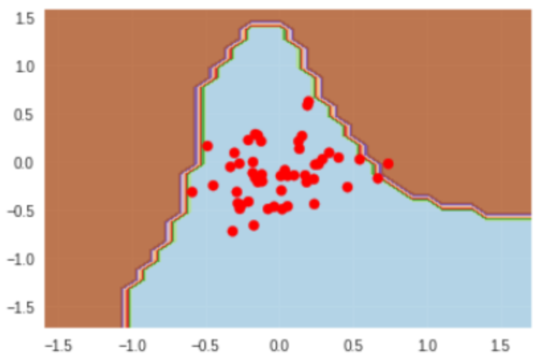


Figure 14: Third scenario
Number of misclassifications =0.08

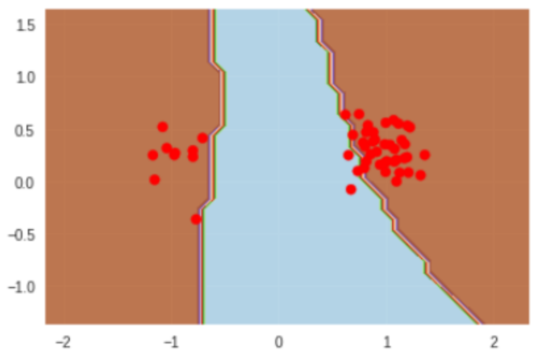


Figure 15: Fourth scenario
Number of misclassifications = 0.12

We can see that if we remove a whole group of specific data points, the number of misclassifications is higher.

The learning and error curves are pretty similar. They are the most dissimilar in the fourth scenario:

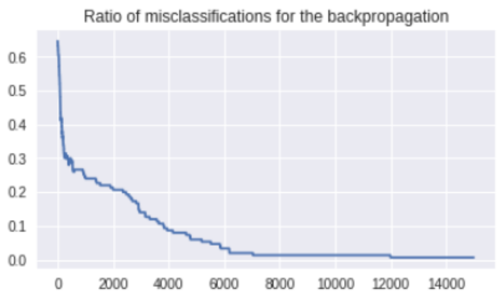


Figure 16

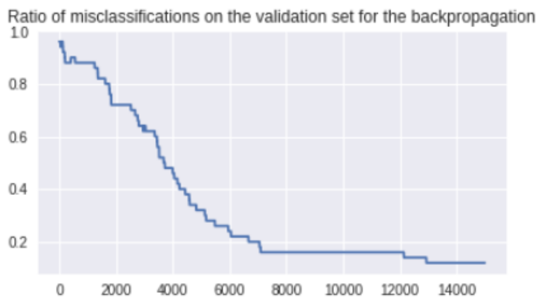


Figure 17

1.2.2 The encoder problem

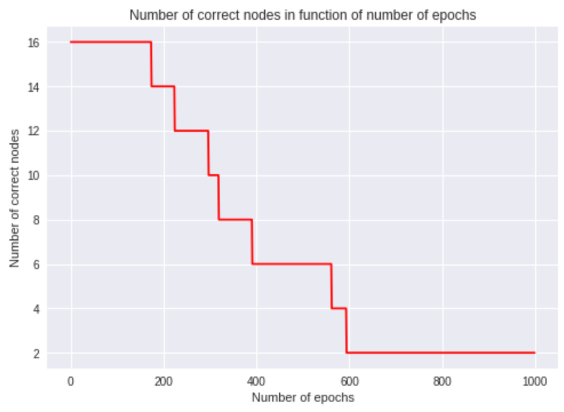


Figure 19

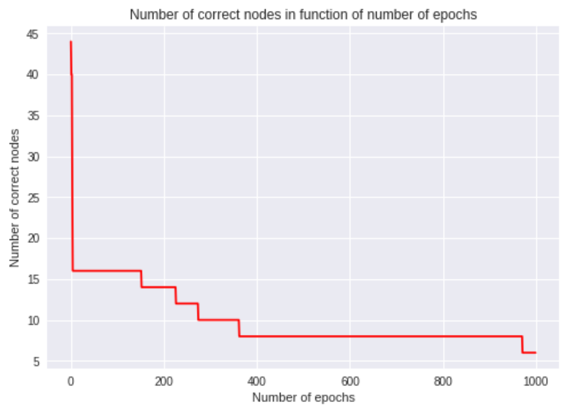


Figure 20

Our auto-encoder network is always converging in that situation of « one out of n » vectors. It is always able to map inputs to themselves for a different number of epochs.

The internal code consist of compressing the input date (size = 8) into a lower dimension (size = 3). It's like a binary code. Then, it uses this binary code to give the output in the first dimension of size 8.

For the weight matrix, the sign of the weights is used to give importance or no to a feature.

If we put the size of the hidden layer equal to 2, our network is not able to map inputs to themselves. This is because you need « $\log_2(8)$ » hidden layers to code all the values.

The autoencoders are good to reduce the dimension of the date : we were able to code in N =3 instead of N =8

Here you do not really need any illustrations, this could be a very short section reporting on your experiments in line with the assignment questions.

1.2.3 Function approximation

```
The number of hidden nodes is  3
The MSE equals  0.03904652313494096
```

```
The number of hidden nodes is  10
The MSE equals  0.0015637006670018987
```

```
The number of hidden nodes is  50
The MSE equals  0.3849235323725873
```

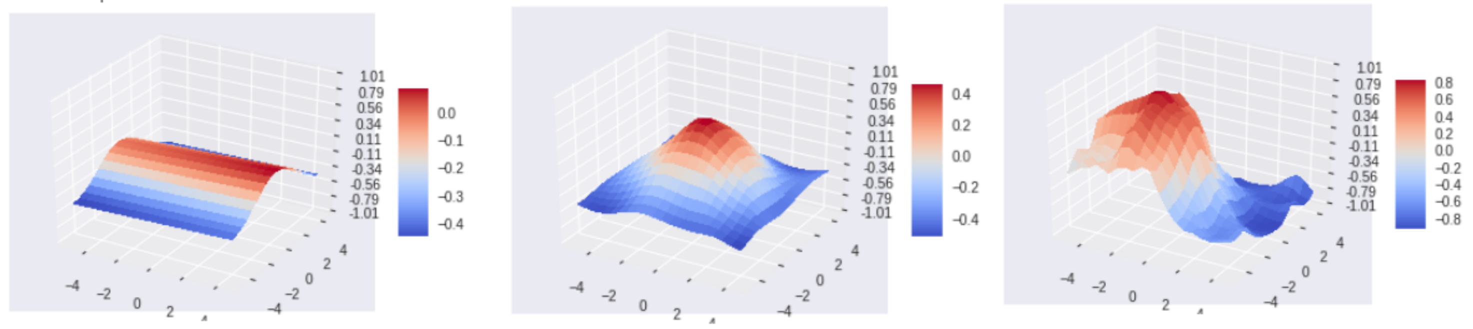


Figure 21

With too few hidden nodes, the perceptron is unable to approximate the function while with too many nodes, it overfits the samples and cannot generalize the model.

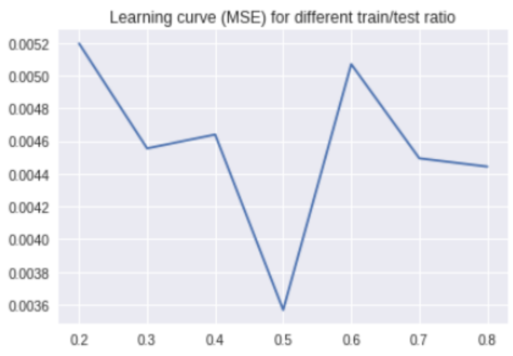


Figure 22

The train/test ratio does not have a huge impact on the error which stays relatively low. But it seems that the error is the lowest when there as many training samples as test samples.
We can try to speed up the convergence by increasing the learning rate, but we must make sure that it remains small enough to

2 Results and discussion - Part II

First, this is the representation of the time series over time :

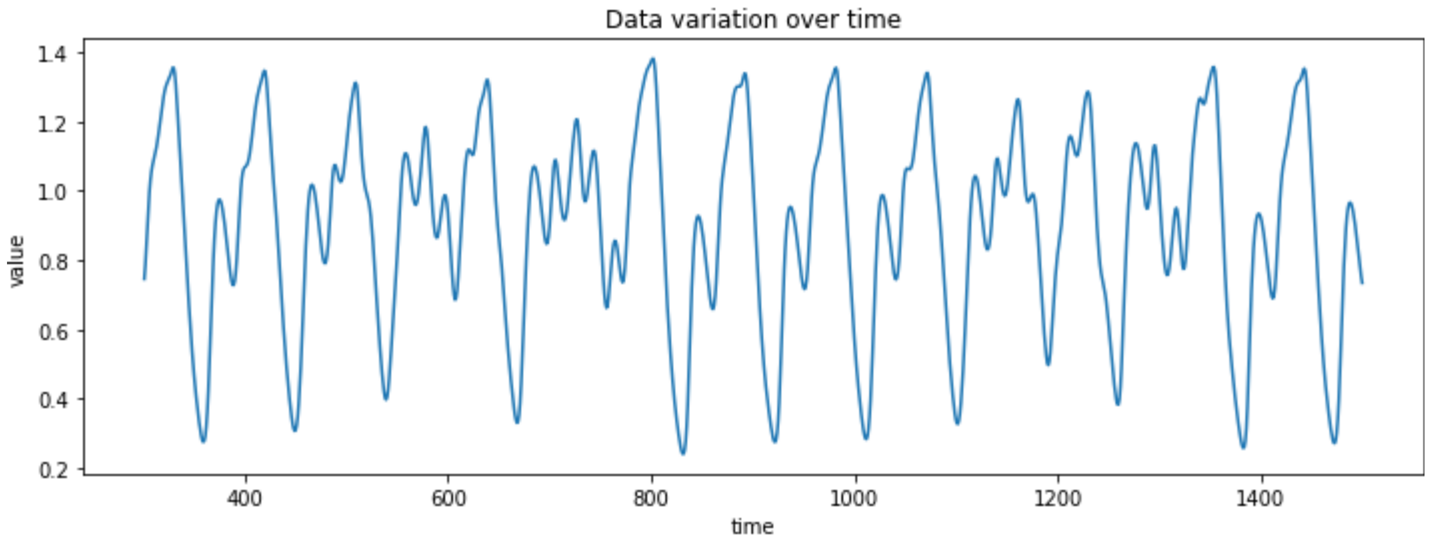


Figure 23

We tried to train a two-layer perceptron on the data and we then computed the error on the validation data for different values of the number of hidden nodes, with and without a regularization term:

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

```
Best error mse:  0.03419303819537163
```

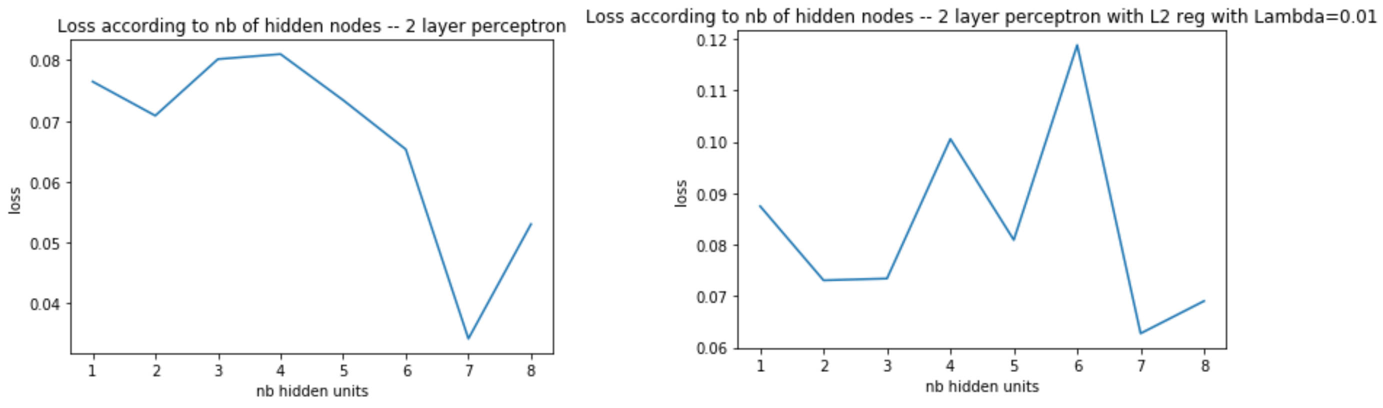


Figure 24

Without penalty, the lowest error is obtained with 7 hidden nodes and equals 0.03. With a regularization term, it is still with 7 hidden but it equals 0.06.
These two graphs explain the effect of the value of lambda, the regularization term:

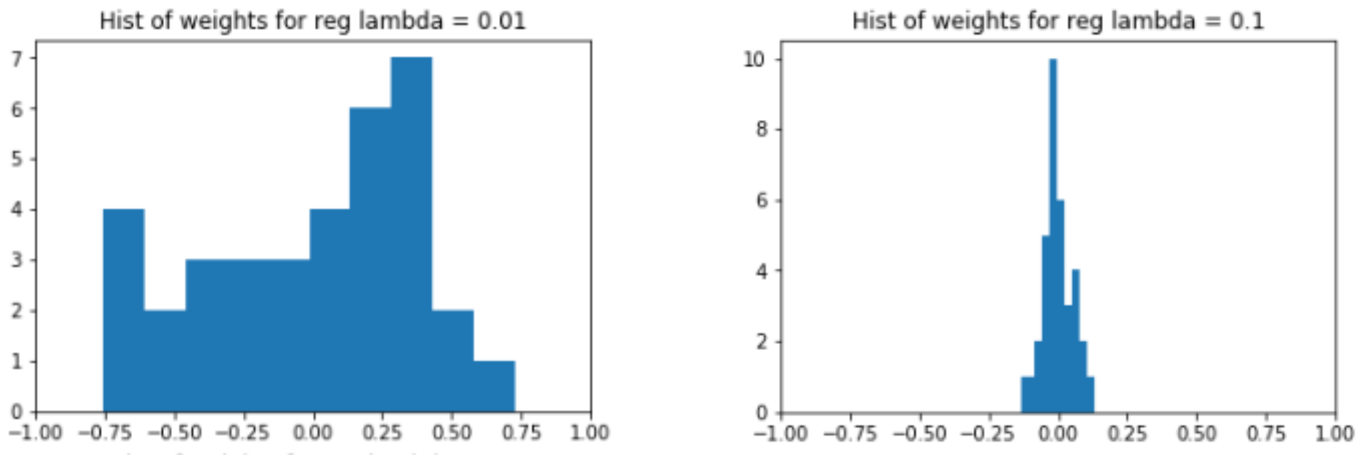
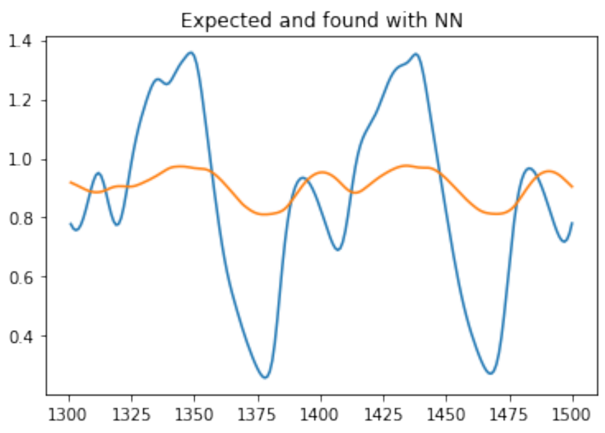


Figure 25

The higher the lambda value, the more it will penalize high weight values. From now on, we will use lambda=0.65.
Our approximation of the time series is done with 7 hidden nodes and with the aforementioned lambda.



It is not a satisfactory result, we need to use an additional layer

Figure 26

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

For the first hidden layer, we still use 7 hidden nodes and lambda = 0.65. We will now try to find the best configuration for the second hidden layer with different variance of noise.

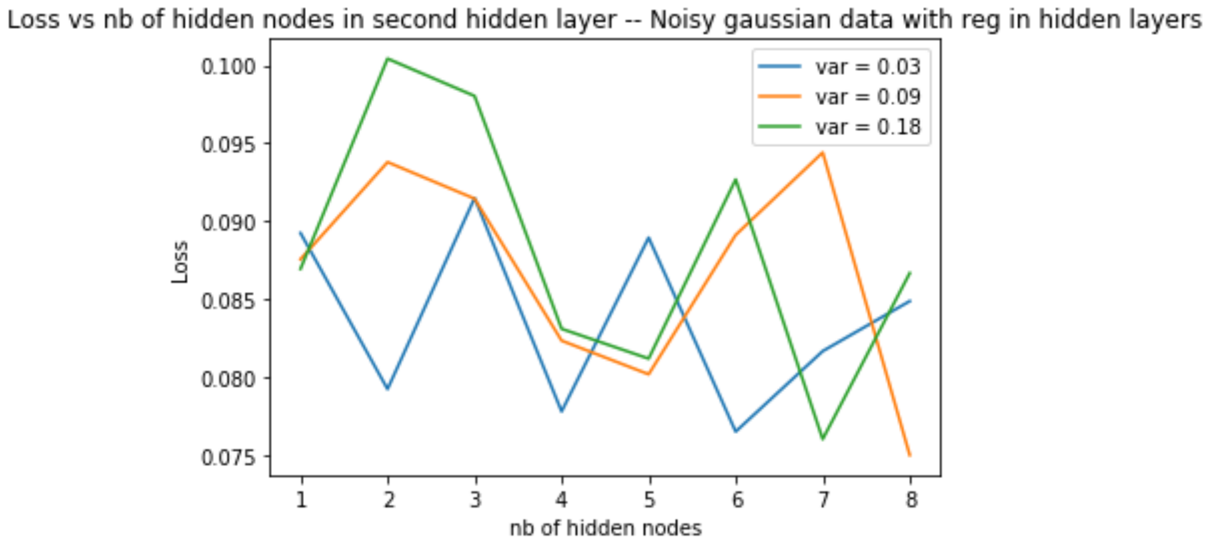


Figure 27

As expected, the best configuration depends on the value of the noise. However, 5 hidden nodes might be a good choice to keep a good generalization capacity.
Let's now do the same approach but without any regularization term.

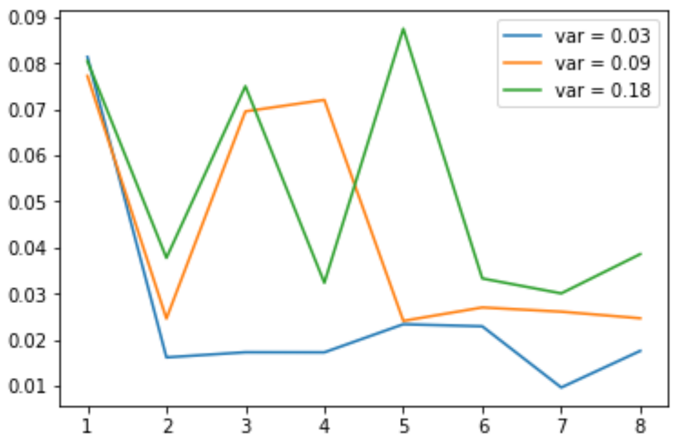


Figure 28

Using more noise makes it harder for a model to fit data distribution while reducing noise weakens regularization effect. Since the level of noise directly affects the two terms in objective function, model fitting and regularization terms, it would be desirable to maintain proper noise levels during training or develop an effective training algorithm given a noise level.
We will train our Neural Network without l2 reg since loss is higher with L2 reg when adding noise to your training data. Why? Adding noise is equivalent to adding extra term in error expression. So, you're already doing a form of reg. No need of l2.
We choose the best values for each noise distribution.

We can now display the approximation of the time series for these three values of variance of noise.

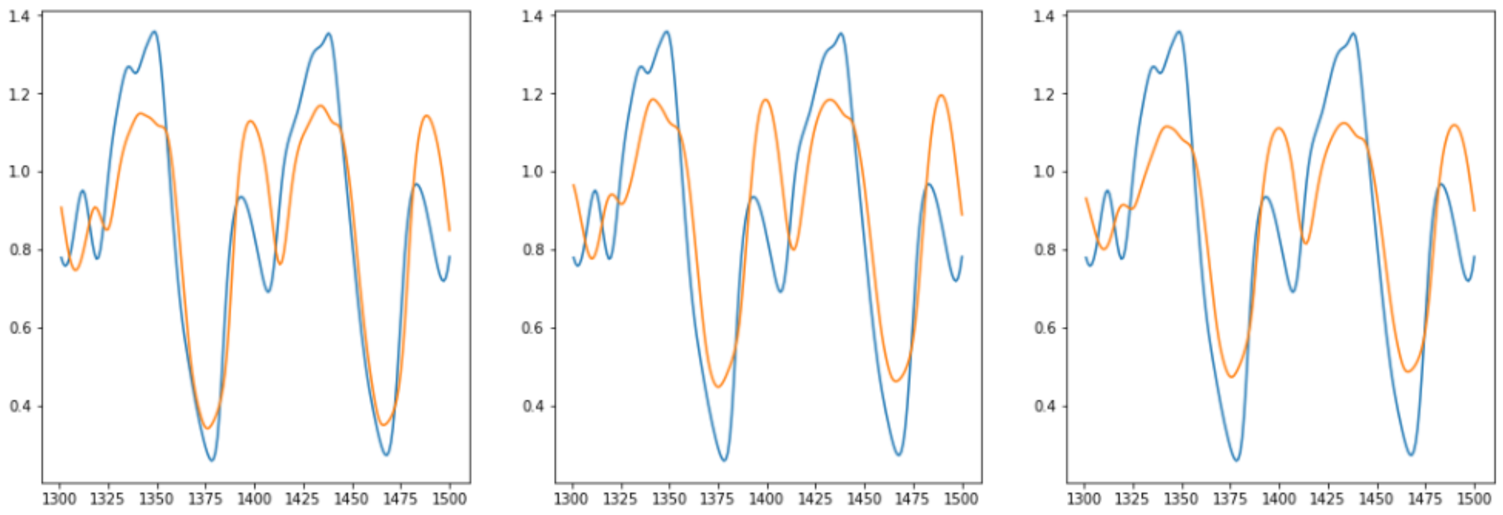


Figure 29: the variance equals 0.03,0.09 and 0.18

In the three cases, it is obvious that a three-layer perceptron is much better at approximating the time series than a two-layer perceptron.

3 Final remarks

It was great to implement our own backpropagation algorithm as it enabled us to better understand its principles.
The comparison of all the parameters for the different scenarios was tedious in our opinion.
And finally, the part on time series was really interesting because it shows a first concrete application of neural network.