Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Tristan Perrot, Romain Darous and Mathis Pernin

February 8, 2023

Please be aware of the constraints for this document. The main intention here is that you learn how to select and organise the most relevant information into a concise and coherent report. The upper limit for the number of pages is 6 with fonts and margins comparable to those in this template and no appendices are allowed.
These short reports should be submitted to Canvas by the authors as a team before the lab presentation is made. To claim bonus points the authors should uploaded their short report a day before the bonus point deadline. The report can serve as a support for your lab presentation, though you may put emphasis on different aspects in your oral demonstration in the lab. Below you find some extra instructions in italics. Please remove them and use normal font for your text.

1 Main objectives and scope of the assignment

List here a concise list of your major intended goals, what you planned to do and what you wanted to learn/what problems you were set to address or investigate, e.g.
Our major goals in the assignment were to

- know how to build the structure and perform training of an RBF network for either classification or regression purposes
- be able to comparatively analyse different methods for initialising the structure and learning the weights in an RBF network
- know the concept of vector quantisation and learn how to use it in NN context
- be able to recognise and implement different components in the SOM algorithm

- be able to discuss the role of the neighbourhood and analyse its effect on the self-organisation in SOMs
- know how SOM-networks can be used to fold high-dimensional spaces and cluster data

Then you can write two or three sentences about the scope, limitations and assumptions made for the lab assignment

In the first part of this assignment, we'll build and study an RBF network to approximate functions in one and two dimensions with noise, while also developing a CL algorithm to automate RBF unit initialization. Then, in the second part, we'll implement the core SOM algorithm for three distinct tasks: ordering animals based on attributes, finding a circular tour passing through ten points on a plane, and creating a two-dimensional map representing the voting behavior of Swedish parliament members. Our aim across these tasks is to find low-dimensional representations of higher-dimensional data.

2 Methods

Mention here in just a couple of sentences what tools you have used, e.g. programming/scripting environment, toolboxes. If you use some unconventional method or introduce a clearly different performance measure, you can briefly mention or define it here.

For this lab, we will use the programming language **Python** and its libraries. Fir the first part, we will mainly use **torch** to build the RBF network. The graphs are displayed using the library **matplotlib.pyplot**. Some results in the second part are displayed using the library **pandas** as well.

3 Results and discussion - Part I: RBF networks and Competitive Learning *(ca. 2.5-3 pages)*

*Make effort to be **concise and to the point** in your story of what you have done, what you have observed and demonstrated, and in your responses to specific questions in the assignment. You should skip less important details and explanations. In addition, you are requested to add a **discussion** about your interpretations/predictions or other thoughts concerned with specific tasks in the assignment. This can boil down to just a few bullet points or a couple of sentences for each section of your results. Overall, structure each Results section as you like. Analogously, feel free to group and combine answers to the questions, even between different*

experiments, e.g. with noise-free and noisy function approximation, if it makes your story easier to convey.

Plan your sections and consider making combined figures with subplots rather than a set of separate figures. ***Figures*** *have to condense information, e.g. there is no point showing a separate plot for generated data and then for a decision boundary, this information can be contained in a single plot. Always carefully describe the axes, legends and add meaningful captions. Keep in mind that figures serve as a support for your description of the key findings (it is like storytelling but in technical format and academic style.*

Similarly, use ***tables*** *to group relevant results for easier communication but focus on key aspects, do not overdo it. All figures and tables attached in your report must be accompanied by captions and referred to in the text, e.g. "in Fig.X or Table Y one can see ....".*

When you report quantities such as errors or other performance measures, round numbers to a reasonable number of decimal digits (usually 2 or 3 max). Apart from the estimated mean values, obtained as a result of averaging over multiple simulations, always include also ***the second moment****, e.g. standard deviation (S.D.). The same applies to some selected plots where* ***error bars*** *would provide valuable information, especially where conclusive comparisons are drawn.*

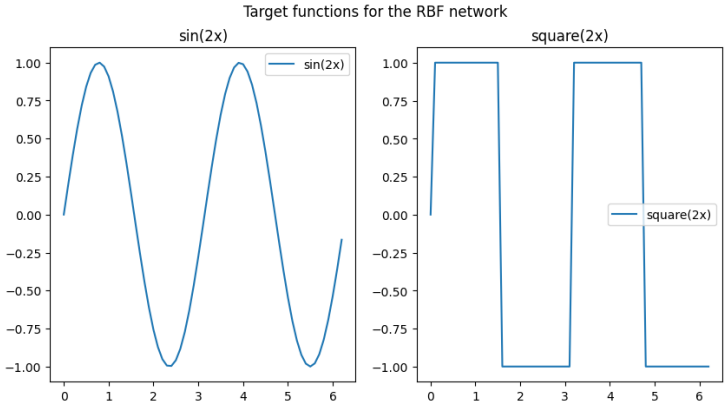
3.1 Function approximation with RBF networks
(ca. 1.5-2 pages)

Combine results and findings from RBF simulations on both noise-free and noisy function approximation (sin(2x) and square (2x)). Try to organise them into subsections, and please make sure you pay attention to the comparison between RBF- and MLP-based approaches as well as the comparative analyses of batch and on-line learning schemes. Answer the questions, quantify the outcomes, discuss your interpretations and summarise key findings as conclusions.

What we are trying to do here is to approximate the functions $x \mapsto \sin(2x)$ and $x \mapsto \text{square}(2x)$. Using a RBG network. The target functions are displayed here :

3.1.1 Network structure

The parameters of the network are the followings :



Figur 1

- The centers : they will be first chosen arbitrarily (as well as their number). Then, we will implement a CL algorithm to choose them,
- The kernel : we will use a Gaussian kernel with common standard deviation σ for all data points,
- The weights between the hidden and the output layer.

Across the lab, we will use two training methods :

- The least square method, which gives an exact solution for the weight, using matrix inversion.
- The delta rule with fixed learning rate that performs gradient descent to converge to the optimal weights.

The training dataset will be the interval $[0, 2\pi]$ regularly divided with a step of 0.01, labelled with the values of the functions introduced above.

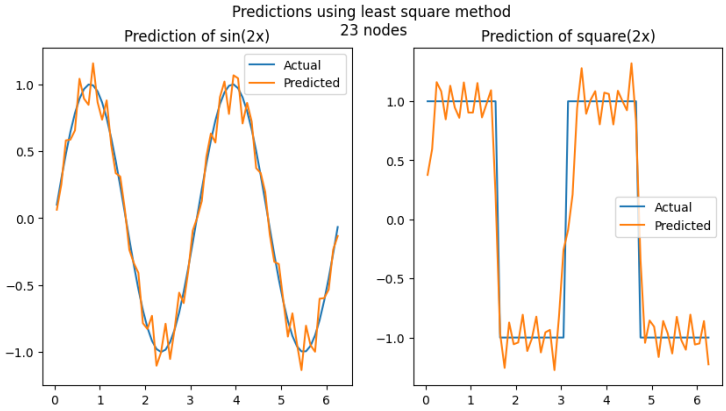
As for the test set, we will do the same, but on the interval $[0.05, 2\pi]$.

3.1.2 Choosing the centers

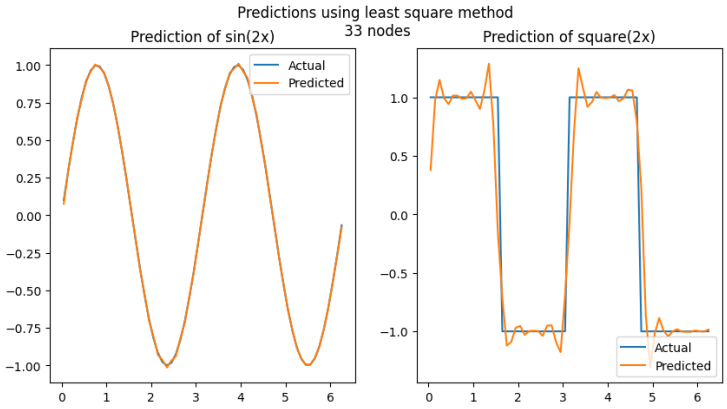
To choose the centers, we simply decided to set a number of centers and to regularly place them on the interval $[0, 2\pi]$, which avoid randomness in the prediction results and gives more comparable results. We will improve this method later using a CL algorithm.

3.1.3 Batch mode training using least squares on free noise data

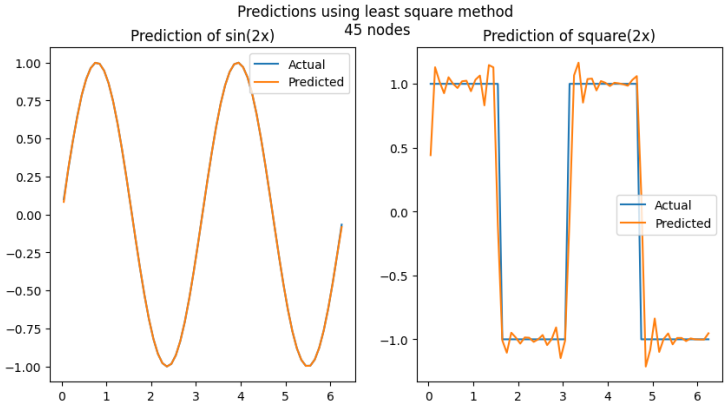
For this part, we set the $\sigma = 0.1$ arbitrarily, as it was giving good results. We will study the impact of this value when performing the delta rule.
After implementing the RBF network, we wanted to find which numbers of nodes would allow to get a residual test error of 0.1, 0.01 and 0.001 for both target functions. We obtain that for the function $x \mapsto \sin(2x)$, we need the following center numbers : $[23, 33, 45]$. However, for the function $x \mapsto \text{square}(2x)$, we could reach an error below 0.1 with 22 nodes, but we couldn't reach an error below 0.01. Still, here are displayed the evolution of the predictions with the number of centers [23, 33, 45] for both functions :



Figur 2: Error below 0.1 for both target functions



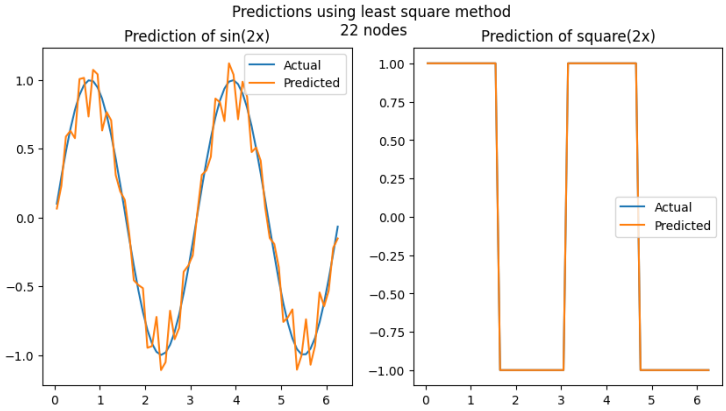
Figur 3: Error below 0.01 for sinus target function



Figur 4: Error below 0.001 for sinus target function

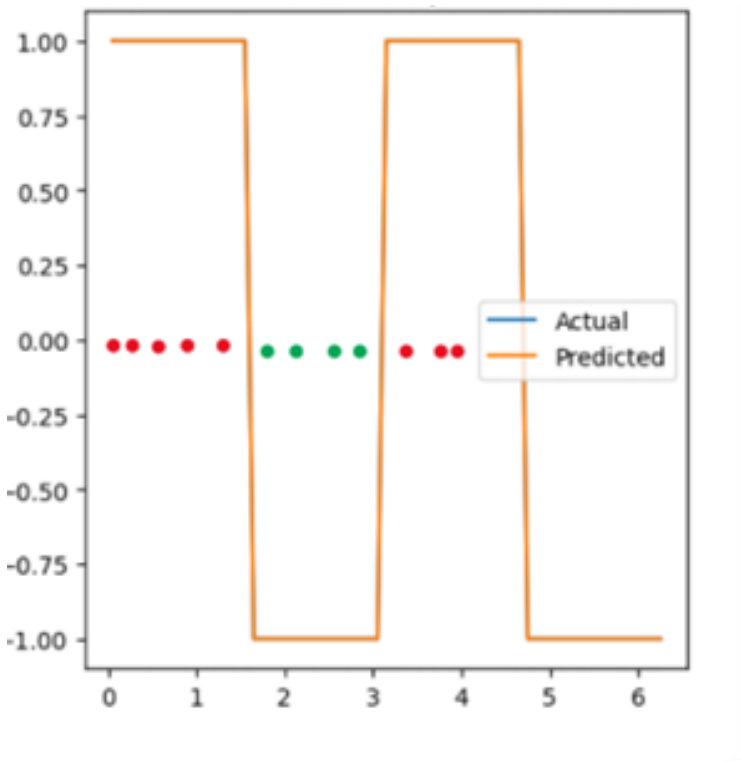
We can see how precise gets the prediction when the number of centers increases.

To improve the residual error for the square function, one idea is to apply the function $x \mapsto sign(x)$ on the prediction. Using this methods gives us a **zero error** starting from 22 nodes. The result is displayed below :



Figur 5

Using a sign function allows to considerably reduce the number of needed nodes, and we get a zero error.
This method is particularly suitable for **classification problems**. The associated classification problem would be this :



Figur 6: Example of a 1-D binary classification problem that would be solved using the square target function. The two class are displayed with red and green dots.

3.1.4 Regression with noise

When performing the delta rule, we shuffle the data after each epoch. We will use the MSE to compare the least square method and the delta rule, as we will observe a bias-variance tradeoff.

To find the best parameters for our noisy data using the delta rule, we performed a grid search over the number of centers and the value of σ.

We get the following results for the $x \mapsto sin(2x)$ target function :

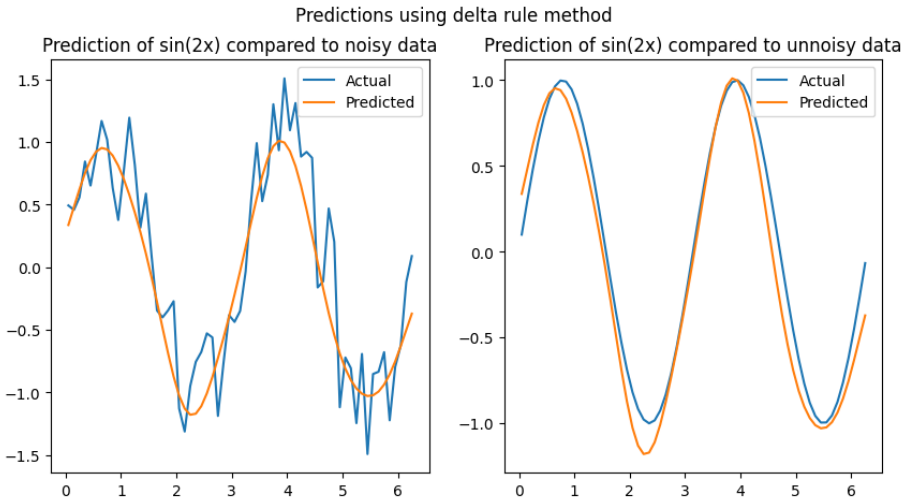
- $\sigma = 0.5$
- number of centers : 37

Regarding the target function $x \mapsto \text{square}(2x)$.

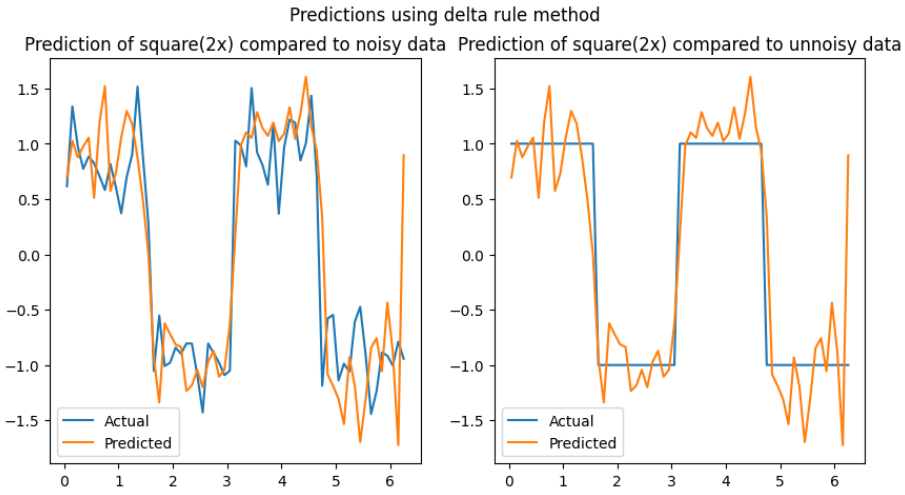
- $\sigma = 0.07$

- number of centers = 45

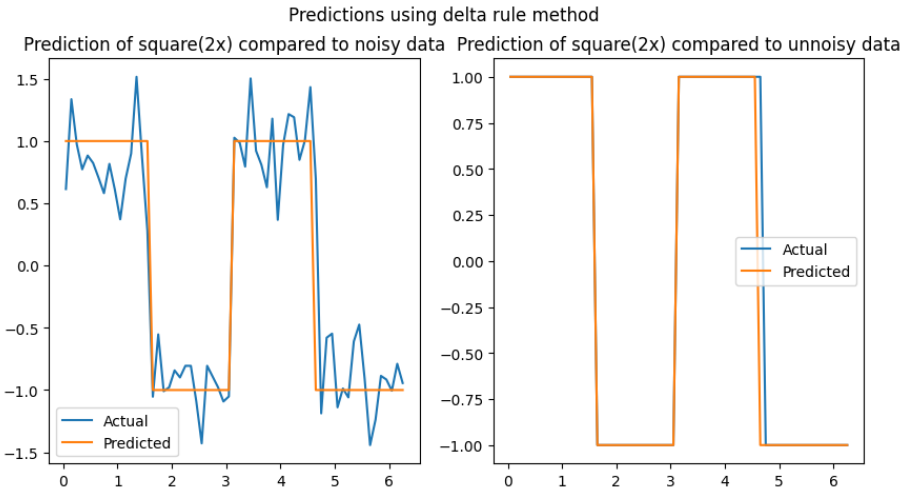
The results of our trainings for those best parameters are displayed below. The prediction is the one trained on noisy data :



Figur 7: Sinus approximation with 37 centers and $\sigma = 0.5$



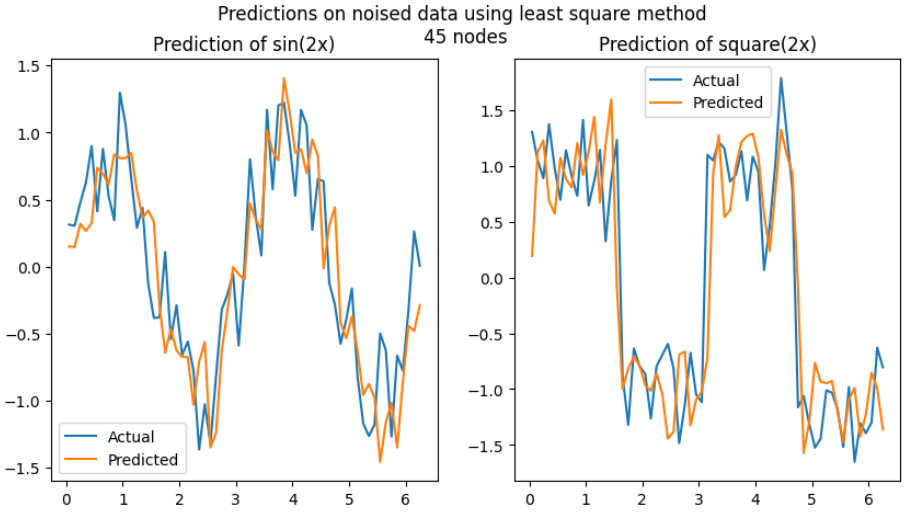
Figur 8: Square approximation with 45 centers and $\sigma = 0.07$



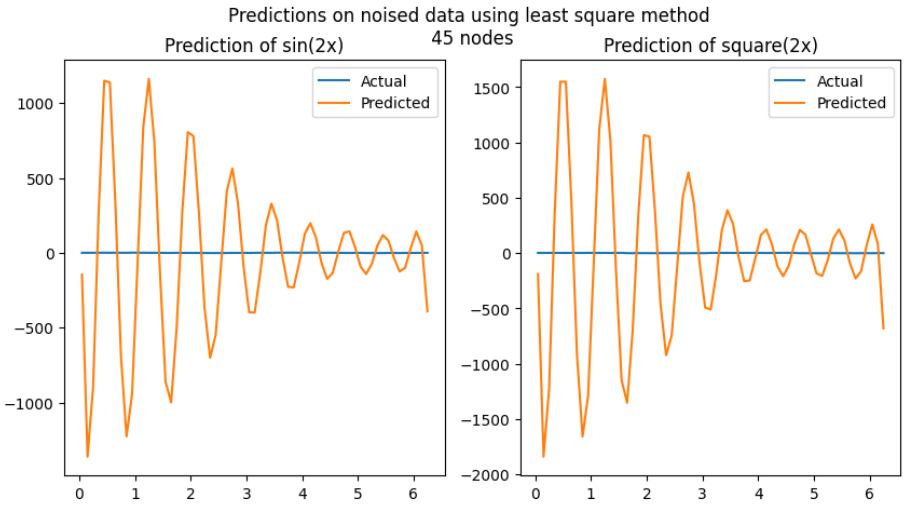
Figur 9: Square approximation with 45 centers and $\sigma = 0.07$ and a classification rule

We can see that the delta rule almost allows to recover the underlying sinus function despite the noise. The square function prediction is more sensitive to the noise, but applying a classification rule make it more accurate.

Let's compare to what we obtain with the least square method and the best node number of before :



Figur 10: $\sigma = 0.1$

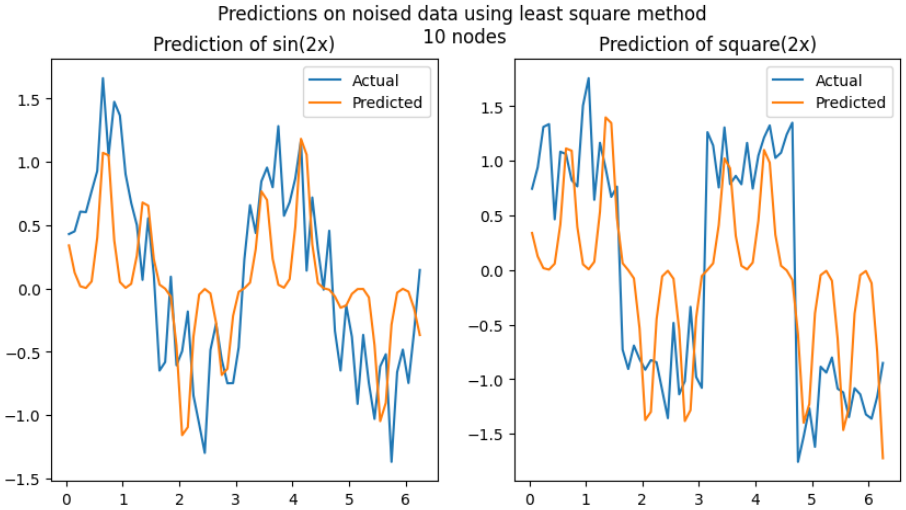


Figur 11: $\sigma = 0.5$

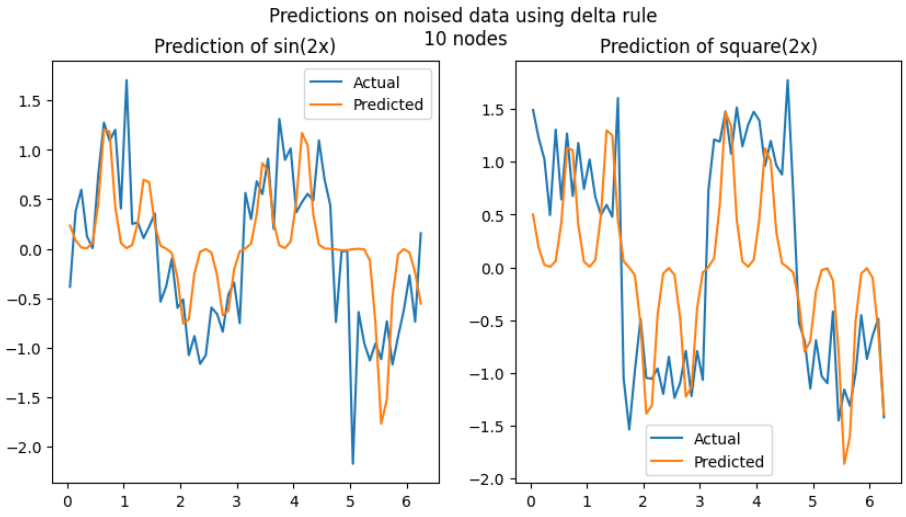
As we can see, the least square methods tends to overfit the data and is too sensitive to noise. Moreover, we cannot adjust σ as we get computations issues if it gets too far from 0.1.

- Effect of RBF units for both methods

First, let's reduce the number of units and compare the result to the ones above and from the first part :



Figur 12: $\sigma = 0.1$



Figur 13: $\sigma = 0.1$

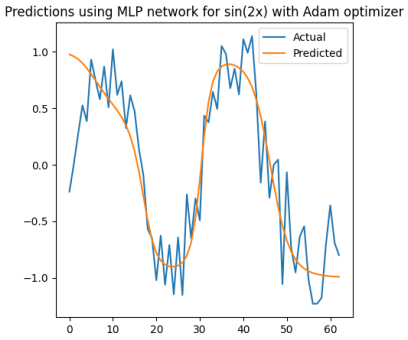
We can clearly see that the model **underfits** the data. To have more information about the data structure, we need more units. However, a balance must be found between fitting the training data and generalisation. It was the purpose of the grid search performed earlier.

We add here some general observation that we made while training our models, that can apply to other machine learning or deep learning models :

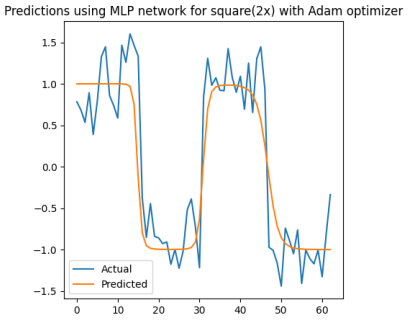
- Reducing the learning rate increases the number of epochs required to make the algorithm converge.
- Impact of the width : increasing the standard deviation σ makes the model smoother for the delta rule. The model gets less sensitive to the noise and we can almost recover the underlying sinusoidal function. It decreases the variance of the prediction, but it reduces the bias. An example can be observed by comparing the model predictions using the delta rule with the best parameters for sinus and square functions. the σ value of the square function is lower, and we can see that the prediction is more sensitive to the noise of the data points.
- Impact of the centers : placing the centers is important. For example, in a classification problem, if the centers are only present among one of the classes, the model will fail classify the data points. Rather than random centers (which becomes close to our strategy for a high number of centers), using competitive learning methods to optimize the centers would be more relevant.

Finally, let's compare the results we obtained with the ones using a MLP with

45 nodes in the hidden layer for the square function and for the sin function using least square and 37 for the sin function using delta rule:



(a) Sinus function approximated by MLP

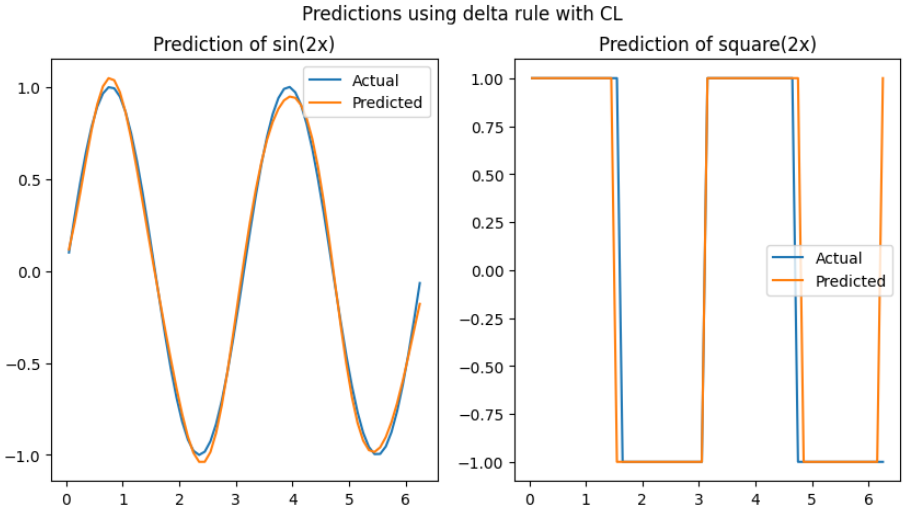


(b) Square function approximated by MLP

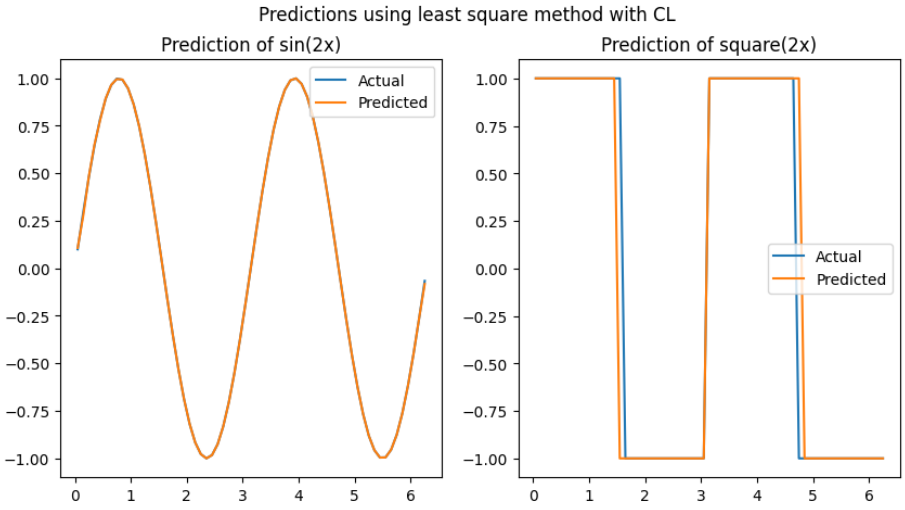
Here we can see that the MLP generalize nicely but not more than the delta rule applied to RBF. The least square method clearly over-fit the data so compared to that, MLP have a greater generalisation ratio. Furthermore, we have the training time for least square that is really close to zero but for the MLP the training time is about 0.07s so it's clearly longer to process. What's important to notice also is that here the MLP clearly depended on the initial condition and not all training lead to a good generalization.

3.2 Competitive learning for RBF unit initialisation *(ca. 1 page)*

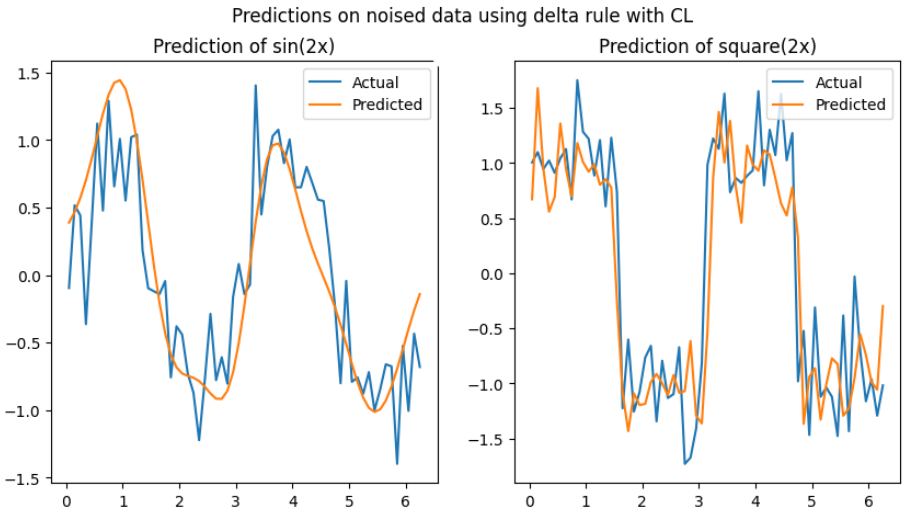
Please refer first to the results in the previous section, i.e. those obtained without any automated initialisation. Then in the next subsection focus on two-dimensional regression with RBF networks.



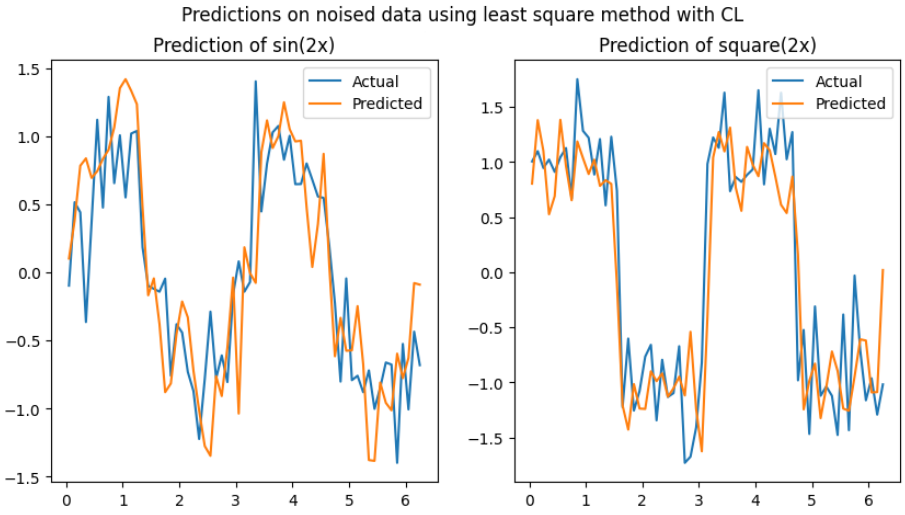
Figur 15: Competitive learning with delta rule noise free



Figur 16: Competitive learning with least square noise free



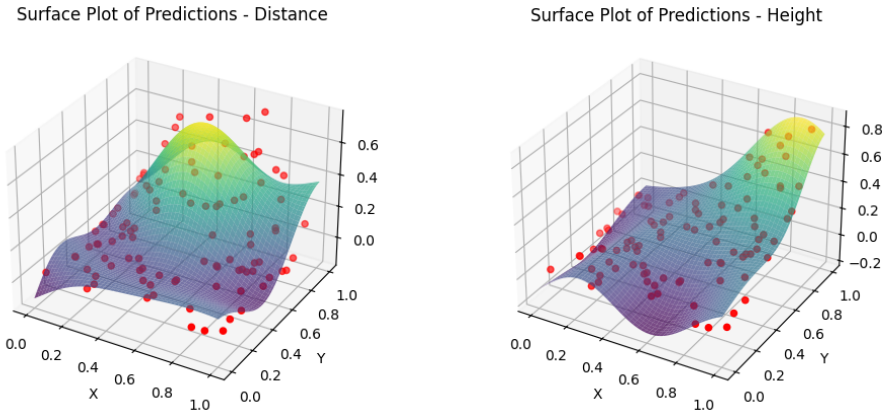
Figur 17: Competitive learning with delta rule noisy



Figur 18: Competitive learning with least square noisy

Here, without noise, we can clearly see that because we had very good result without competitive learning with a very well generalized network, it is not necessary to had competitive learning. However, with noise, we can see that the delta rule is well generalized for the sin function but the square function is still overfitting. Moreover, the least square still overfitt as it still learn to much from the training data.
To avoid dead units, it could be good for the firsts epochs to update every weights and then apply the Winner-Takes-All rule to finish the job and apply the competitive learning.

Moreover we have trained the RBF with a 2D representation with the provided. Here are displayed the surface predictions on each dimension of the output :



(a) 2D trained - distance

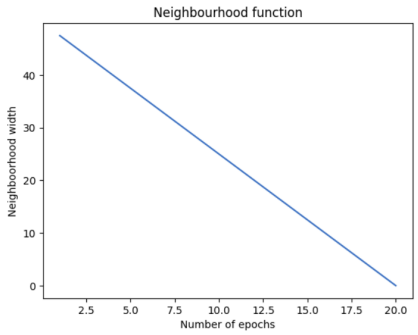
(b) 2D trained - height

Here we can see that the RBF algorithm generalize pretty well the result after an optimization of the hyper-parameters. The surface is really following the trend of the data.

4 Results and discussion - Part II: Self-organising maps *(ca. 2 pages)*

4.1 Topological ordering of animal species

Implementing and running this SOM was fairly easy, as it was the simplest of the three exercises. We did not need to tune parameters such as the learning rate in order to obtain convincing results. Throughout the exercise, we worked with three different neighbourhood functions: the first with linear decay, the second with exponential decay and the third with Gaussian decay. Here, the simplest (linear decay, as shown below) provided satisfactory results. However, after several trials, we noticed that this neighbourhood empirically produced fewer clusters than the other two. The network converges rapidly (around 20 epochs). Finally, we notice that as the learning rate increases, there are more clusters, which is logical since the nodes first move away more quickly when the neighbourhood is high, and can no longer rejoin when the neighbourhood is low in the end. For these results we used $\eta = 0,2$.



	Cluster	Animals
0	0	[camel, giraffe, pig]
1	3	[antelop]
2	6	[horse]
3	11	[bat, elephant]
4	14	[rabbit, rat]
5	19	[kangaroo]
6	24	[skunk]
7	27	[cat, lion]
8	30	[dog]
9	36	[ape]
10	39	[bear, hyena]
11	45	[walrus]
12	52	[crocodile, seaturtle]
13	58	[frog]
14	64	[ostrich]
15	67	[penguin]
16	70	[duck, pelican]
17	80	[spider]
18	88	[housefly, moskito]
19	93	[butterfly]
20	96	[beetle, grasshopper]
21	99	[dragonfly]

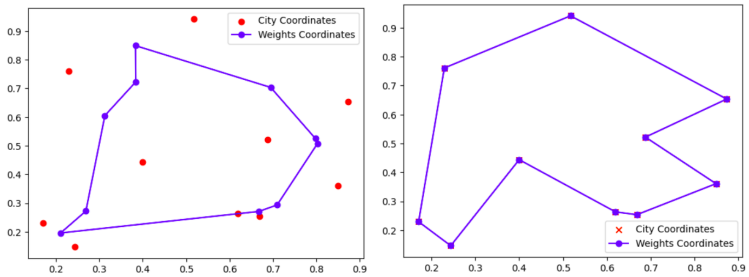
Figur 20: Topological ordering of animals using a SOM

4.2 Cyclic tour

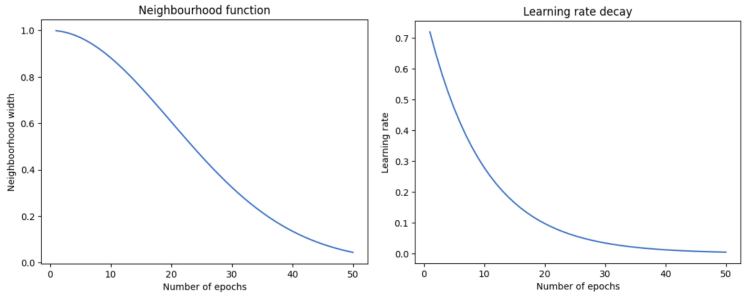
In this case, it is necessary to tune the parameters to obtain a truly satisfactory result. For example, we had to opt for a decreasing learning rate, because without it the proposed solution was poor. In addition, this example highlights the importance of defining the neighbourhood correctly: if the initial neighbourhood is 2, the contour never reaches the points, whereas it often succeeds when the initial neighbourhood is 1. The best performing neighbourhood is the Gaussian neighbourhood (shown below). This time the algorithm requires more epochs (around 50) to achieve a satisfactory result. Finally, we note that the algorithm sometimes has difficulty taking into account nearby points, which often have the same BMU when we would like them to have a different one. In addition, the result was sometimes quite sensitive to the initialization of the weights, and we were sometimes unable to compensate for the poor initial layout, resulting in a twisted contour.

4.3 Clustering with SOM

The last exercise is the most complex and interesting. It was more difficult to converge this SOM towards an interpretable result: the parameters had to be tuned precisely by successive trials. In particular, it was difficult to separate the entry points to obtain a large number of clusters and fill the map. In particular,

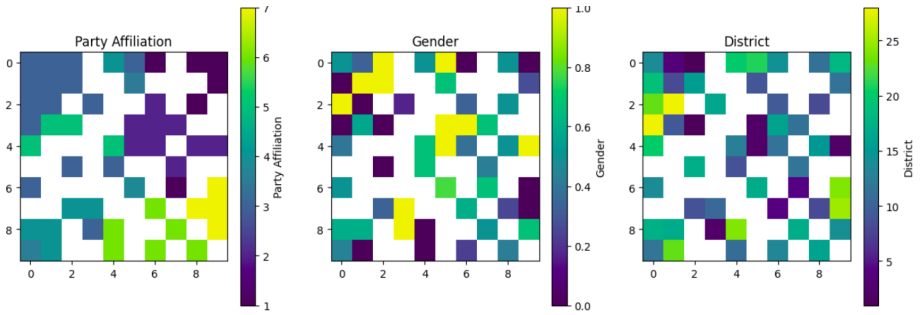


Figur 21: Cyclic tour with maximal neighbourhood width respectively 1 and 2



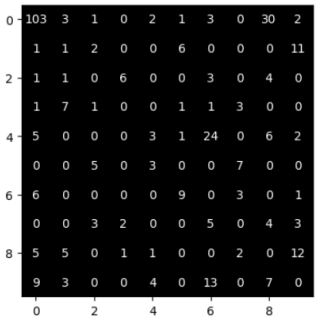
Figur 22: Neighbourhood and learning rate as a function of epochs

we never managed to separate a cluster of 103 fairly centrist MPs whose votes were supposed to be very close. We opted for a Gaussian decay neighbourhood. With a learning rate that was too high or a neighbourhood that was too wide, we observed very few clusters. Convergence was nevertheless satisfactory fairly quickly: we obtained more or less the same results with 20, 50 or 100 epochs. We decided to represent for each BMU the average, according to the criterion studied, of the entry points attached to it.

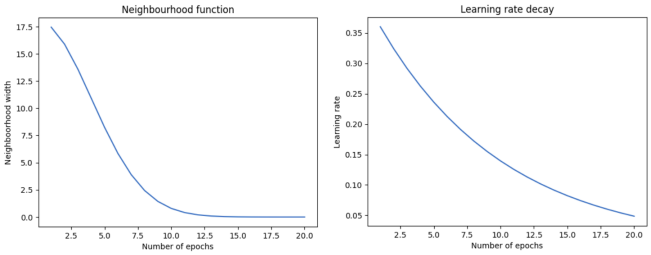


Figur 23: Clustering of Swedish MP votes using SOM with $\eta = 0,4$ and 20 epochs

As for the interpretation of the results, it is clear that the most decisive parameter in the vote is the party. On this map, we can clearly see marked clusters according to party. While the algorithm easily manages to separate the most extreme votes, it is more difficult to distinguish between the centrist votes. This



Figur 24: Number of MPs per cluster



Figur 25: Neighbourhood and learning rate as a function of epochs

is probably due to the use of the average. It is interesting to note that the votes for the two extremes (extreme left or extreme right) are closer together than they are to the centrist vote, although we might expect the opposite. It is more difficult to distinguish a general trend linked to district or gender. The votes are more mixed. There are more points close to the average, and the extreme points are more mixed. Finally, it can be said that representing the colours according to the average in the first approach was not optimal, especially for the average points. If we had taken the study further, we would have had to study the distribution of points attached to each BMU, and this would have produced much more accurate results.

5 Final remarks *(max 0.5 page)*

Please share your final reflections on the lab, its content and your own learning. Which parts of the lab assignment did you find confusing or not necessarily helping in understanding important concepts and which parts you have found interesting and relevant to your learning experience?
Here you can also formulate your opinion, interpretation or speculation about some of the simulation outcomes. Please add any follow-up questions that you might have regarding the lab tasks and the results you have produced.

In section 1, we performed RBF using two methods : batch learning using least square optimization, and online learning using delta rule. We saw that delta rule was offering better generalisation perspectives, as adjusting the number of nodes allowed to recover the underlying functions without noise pretty accurately.
Adjusting the center positions using competitive learning allowed us to get even better approximations of the underlying functions.
Implementing the RBF network on 2D data was a way to test the network on more complex data. Results seem to offer a relevant predictions on the test dataset.

In section 2, we firstly performed exploration of topological ordering of animal species through Self-Organizing Maps. The simplicity of the SOM implementation in the first exercise allowed for a clear understanding of neighborhood functions, with linear decay yielding satisfactory results. The cyclic tour exercise emphasized the importance of parameter tuning, showing the effectiveness of a Gaussian neighborhood. The final clustering exercise presented the most complexity, revealing the decisive influence of party affiliation on voting patterns. Challenges in distinguishing centrist votes and interpreting results underscored the nuances of SOM applications. Overall, the exercises provided valuable insights into the capabilities and considerations of SOMs, emphasizing the need for careful parameter selection and result interpretation.