Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Jesús Ventas Muñoz de Lucas, Willem van der Schoot

September 12, 2019

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- Build RBF networks from scratch, test their performance and find ways to initialise the centers
- Build SOMs from scratch and test their performance on a range of problems

2 Methods

The programming language used was Matlab, all networks were built from scratch so no toolboxes were utilised.

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

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

3.1.1 Batch mode learning using Least Squares

Simulations were carried out with a fixed sigma and the Absolute Residual Error for sin(2x) and square(2x) were computed to study the effect of sparse RBF space compared to a crowded RBF space. In Figure 1 it can be seen that to reduce the validation error, approximately between 9 and 12 RBFs are suitable, for a std deviation = 1.2. In the case of sin(2x) the error is below 0.001 between 9 and 12 RBFs, below 0.01 between 7 and 12 and below 0.1 between 5 and 15. In the case of the square(2x) the error is never below 0.1. For reducing the error in square(2x) the output of our NN was transformed by computing the sign, which could be very useful in classification tasks. In that case, the error was 0 if the number of RBFs was between 5 and 14.

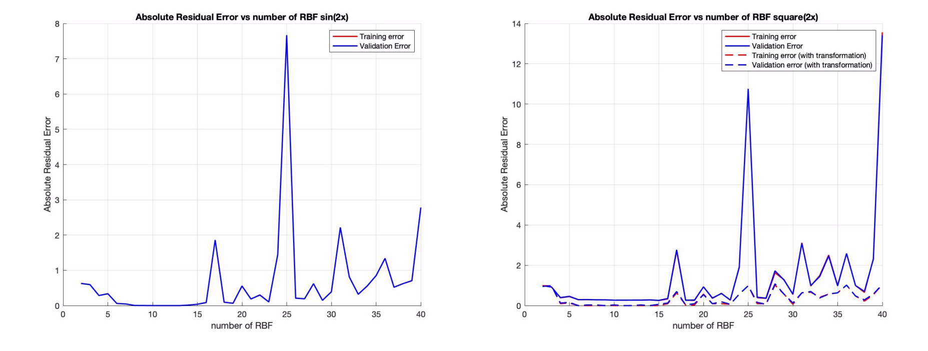

Figure 1: Absolute Residual Error in sin(2x) (left) and in Square(2x)(right) for std.deviation = 1.2 and means uniformly-spaced

3.1.2 Regression with noise

For comparison between models we have used the validation error. The training error oscillates a lot in the case of the Delta Rule since the error is estimated using only one input sample each time. In Figure 2, the Delta Rule and Least Squares methods are compared. When noisy data is used, the Delta Rule performs slightly better, whereas Least Squares is best when the data is clean.

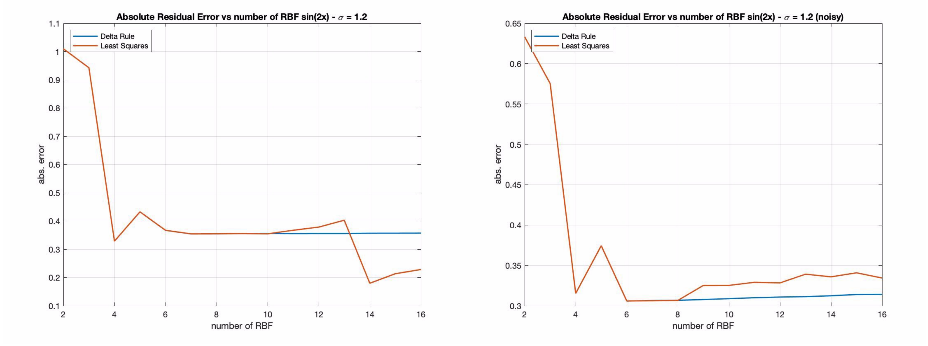

Figure 2: Comparison Delta Rule and Least Squares for a varying number of RBF with clean data (left) and noisy data (right)

It is necessary to choose the learning rate carefully in delta rule. As seen in Lab 1, and also in Figure 3 (left), if the learning rate is very large, the obtained error keeps oscillating around the minimum. In the case of Least Squares, it does not make sense to speak about learning rate and convergence since it obtains optimum weights in one step. The effect of RBFs width was also studied. If RBFs are very wide all neurons fire independently of the input, but if RBFs are very narrow, neurons only fire for a very few inputs. A width given by sigma between 0.9 and 1.9 performs well in the network, as seen in Figure 3 (right).

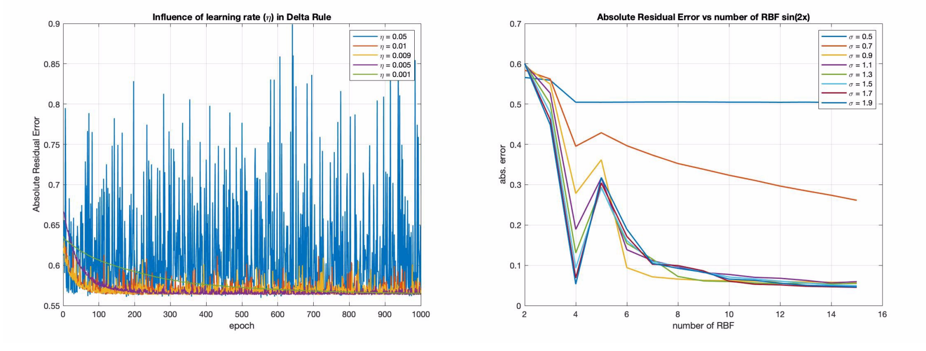

Figure 3: Effect of learning rate (left) and RBFs width (right)

The performance of the MLP developed in lab1 was then compared with a NN using RBFs. As seen in Figure 4, RBFs performs better than MLP when there are enough RBFs. RBFs nodes were uniformly-distributed along the input space. Furthermore, the MLP networks took more time to converge than the RBF networks.

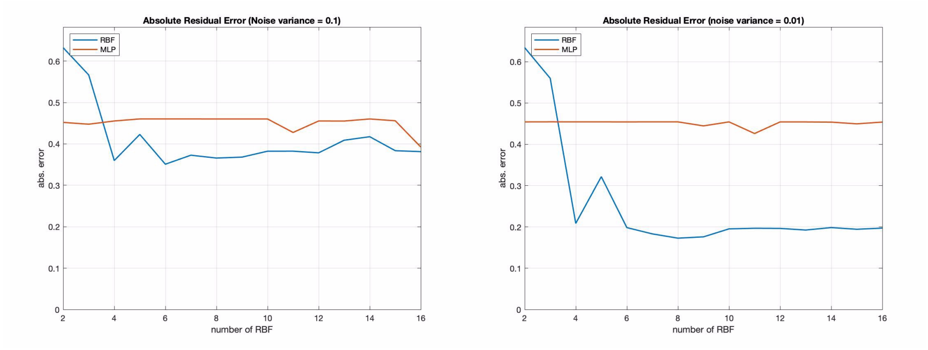

Figure 4: MLP vs RBFs

Finally, random centered RBFs were then compared with manual uniformly spaced centers. After hundred simulations with random positions and averaging, the performance obtained was similar to when positions were selected manually. Therefore, the effect of selecting centers randomly (with uniform distribution) reduces stability, since sometimes center are wrongly positioned.

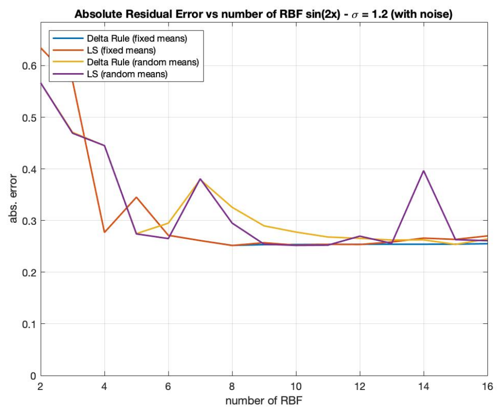

Figure 5: Comparison between fixed and random RBF centers

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

A comparison between using CL and fixing RBFs positions was carried out. As seen in Figure 6 (left), the network performs better when trained with noisy data , whereas with clean it performs similarly in both cases. This is because fixed RBFs are uniformly positioned and the given data also distributes uniformly along the input interval. Some strategies for avoiding dead units were implemented and compared in terms of Absolute Residual Error as seen in Figure 6 (right). With leaky learning looser neurons move in the direction of the winner, moving more when closer to the winner.

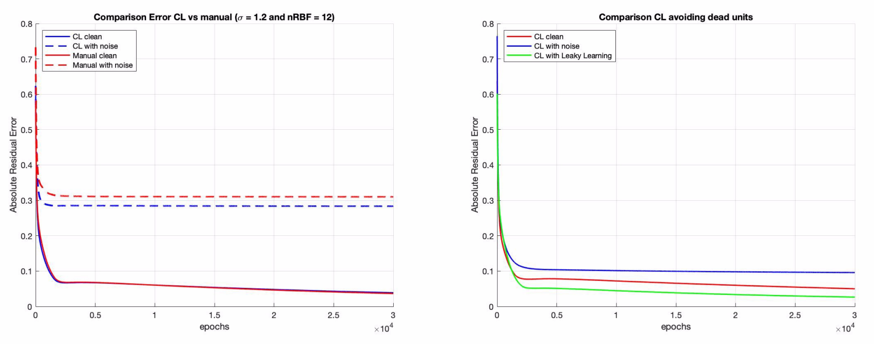

Figure 6: RBF positions fixed vs using CL (left) and performance comparison of different strategies for avoiding dead units (right)

The last experiment with RBFs consisted of using an RBF network to approximate a R2 to R2 function. After some simulations, 12 multivariate-gaussian-RBFs with sigma = 1.2 performed well for this purpose. The initial and final RBF positions and input positions are shown in Figure 7 (left). After applying the delta rule validation errors obtained for both outputs are shown in Figure 7 (right), where it can be seen that the output 2 error decreases while output 1 error remains almost constant increasing slightly. However, the whole error norm decreases with epochs so the algorithm converges.

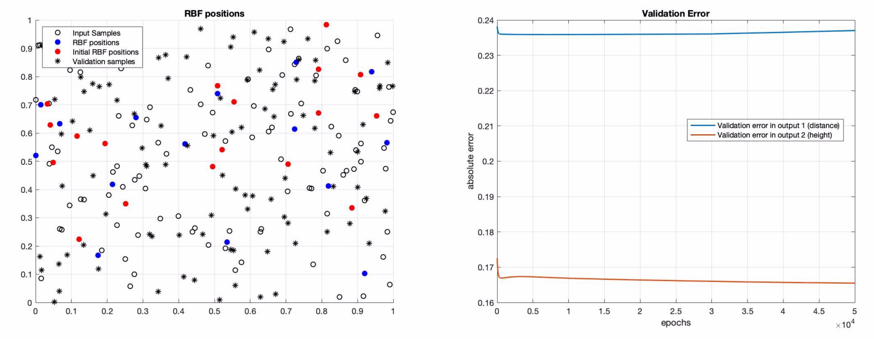

Figure 7: RBF and input positions (left) and validation error in the outputs (right)

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

4.1 Topological ordering of animal species

The aim was to assign a natural order to 32 animal species characterized by 84 binary attributes. Three parameters were used to ensure the algorithm converged and produced a reasonable result, they were:

- Number of epochs - In order to be able to first learn then converge, 20 epochs were used.
- Step size - This dictated essentially the 'learning rate' - how much the weight distances changed each epoch for the winning node and neighbours, set at 0.2 for this SOM.
- Neighbourhood - The neighbourhood needed to be large enough at the beginning to ensure no dead weights, before being small at the end to converge. The neighbourhood was simply the range of nodes indexed next to the winning node in 1D.

The ordering produced by the SOM : [giraffe, camel, pig, horse, antelop, kangaroo, bat, elephant, rabbit, rat, skunk, hyena, dog, lion, cat, ape, bear, walrus, crocodile, seaturtle, frog, penguin, ostrich, duck, pelican, spider, beetle, dragonfly, grasshopper, butterfly, housefly, moskito]. Looking at the ordering it is clear the insects are positioned together at the end, the water animals/birds towards the middle, and the bigger animals towards the front.

4.2 Cyclic tour

Given 10 cities, each with two coordinates and a total of 10 nodes, the travelling salesman problem was configured. The neighbourhood in this example was cyclic. With this configuration it was difficult to find convergence, with one node often being 'dead'. To solve this, the learning rate was increased slightly, the neighbourhood reduced slowly from 2 to 1 to 0. Even with this configuration it was identified though that a leaky update would be needed for the node which always ended up stranded as seen in Figure 9, node at position 7.

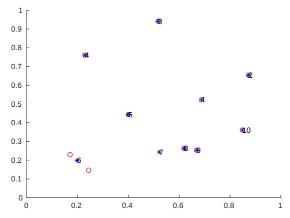

Figure 8: Learning process, dead node p7

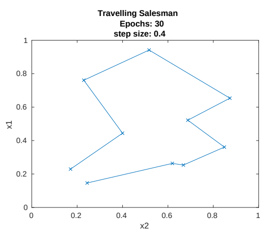

Figure 9: Travelling salesman solution

The result was good approximation as seen in Figure 9.

4.3 Clustering with SOM

The final problem was to cluster 349 MPs based on 31 votes they took in a certain time period, and place them on a 10x10 map. Three types of metadata were used: gender, district and party, to see if they effected the votes of the MPs. 20 epochs were once used again along with a learning rate of 0.2. For the results a small amount of noise was added to better show the MPs distribution rather than all clashing on each node.

Gender and Districts did not seem to have a large effect as seen in Figure 10, while it was clear the party did as seen in Figure 11.

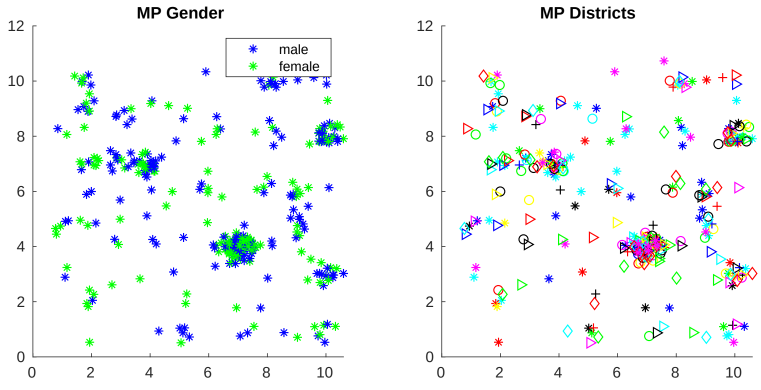

Figure 10: MP clustering by Gender and Districts

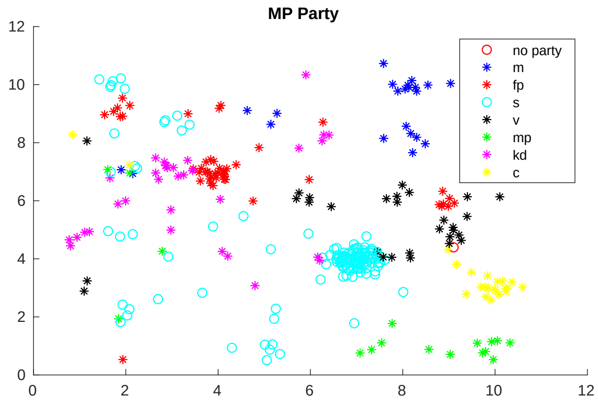

Figure 11: MP clustering by Party

5 Final remarks *(max 0.5 page)*

All learning outcomes were satisfied.

For Part 1 it was interesting to see the effect of varying the number of RBFs, its width and their distribution along the input space according to how data are distributed. It is important to find a right trade off between previous parameters to ensure a good performance in the neural network.

For Part 2 it was insightful to see how such simple networks could produce great results. Careful attention though had to be given to the neighbourhood, step size and epoch number to ensure there were no dead nodes and the network converged.