Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Patrik Ekman, Eric Hartmanis, Martin Koling and Daniel Wass

September 23, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to increase our knowledge in the fundamental dynamics of Radial Basis Functions (RBF).
- to increase our knowledge in the SOM-algorithm and the Competitive Learning (CL) method of initialization of RBF units.
- problems arose mainly from practical coding issues.

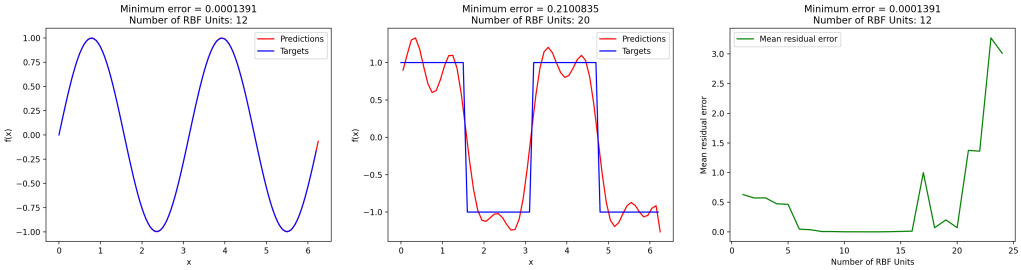
2 Methods

All code have been built in Python3 using Microsoft's developer environment Visual Studio Code. Plots have been created in Matplotlib. When requested, noise was added to the targets, i.e. not to the input data.

3 Results and discussion - Part I: RBF networks and Competitive Learning

3.1 Function approximation with RBF networks

We positioned the RBF units evenly in input space with a standard deviation (std) of $\sigma = d/\sqrt{2M}$, where d is the longest distance between nodes in input space and M is the number of nodes - which is a common choice for the unit width (Marsland, 2015, 119). The batch mode of the network (i.e. least squares approximation for the unsolvable equation) performed well in approximating the sinus function, but not so well for the square function. This is reasonable as the square function values are strictly 1 or -1, whereas the sinus values are, as well as the Gaussian RBF units, continuous between 0 and 1, see *Fig 1a* and *1b*. However, by applying the square function on the output values y (i.e. transforming them into -1 for $y < 0$ and 1 for $y >= 0$) we receive zero error on the square function targets with 9 units. Such a transform could be useful in binary classification tasks, where the output should simply just be 1 or -1. The number of nodes required for an error below 0.1, 0.01 and 0.001 were, for the sinus curve, 6, 8 and 9 respectively, see *Fig 1c*. 12 nodes performed best in terms of mean residual error: 0.00014. For the square function the lowest error was 0.21, given by 20 units.



(a) Sinus approximation. *(b) Square approximation.* *(c) No. of nodes, sinus.*

Figure 1: Least squares approximation with the RBF units evenly placed in input space.

3.2 Function approximation with RBF networks and noise

A comparison of the effect of the number of RBF units and their width for the two learning approaches and the Sinus function specifically, can be seen in *Fig. 2*. The result of 1 to 20 RBF units are displayed because of earlier results. It is evident that sequential learning performs better with smaller RBF widths while batch learning performs better with larger RBF widths. This makes sense since batch learning takes all data points into account, and hence the RBF Units need to cooperate to be able to learn the patterns of both negative and positive values. The sequential learning on the other hand, only takes one data point into account, and so the RBF units can be narrower and each learn different patterns corresponding to the area where their specific data points exist in input space.

While batch learning is very robust with standard deviation 0, sequential learning (with shuffling after each epoch) was also considered robust and the maximum standard deviation did not exceed 0.035.

The number of RBF units that performed best, considering both learning schemes, was between 12 and 20, and the best performing σ between 0.25 to 1. Thus, 15 RBF units and σ equal to 0.5 was considered the best overall model.

Error	RBF Units																		
Std Dev of RBF	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0,1	0,675	0,676	0,677	0,610	0,678	0,547	0,537	0,533	0,545	0,522	0,470	0,437	0,389	0,400	0,391	0,347	0,386	0,372	0,330	0,311
0,25	0,665	0,649	0,648	0,526	0,653	0,450	0,385	0,352	0,316	0,299	0,270	0,262	0,260	0,266	0,268	0,264	0,265	0,266	0,264	0,263
0,5	0,639	0,589	0,587	0,480	0,589	0,331	0,266	0,264	0,260	0,259	0,259	0,260	0,259	0,265	0,265	0,266	0,267	0,266	0,264	0,263
1	0,611	0,542	0,541	0,447	0,491	0,269	0,262	0,265	0,260	0,259	0,266	0,266	0,263	0,265	0,266	0,265	0,782	0,272	0,322	0,273
1,5	0,631	0,584	0,584	0,468	0,478	0,262	0,259	0,264	0,261	0,259	0,267	0,270	0,270	0,266	0,265	0,286	0,313	3,655	0,279	0,319
3	0,654	0,603	0,603	0,528	0,529	0,293	0,295	0,261	0,255	0,258	0,256	0,262	0,253	0,270	0,256	0,257	0,268	0,261	0,256	0,252

(a) Batch learning

Error	RBF Units																		
Std Dev of RBF	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0,1	0,685	0,695	0,694	0,614	0,700	0,598	0,561	0,550	0,548	0,519	0,533	0,516	0,426	0,410	0,425	0,383	0,400	0,380	0,364	0,337
0,25	0,688	0,674	0,673	0,527	0,676	0,477	0,414	0,389	0,293	0,291	0,295	0,293	0,267	0,279	0,280	0,280	0,288	0,292	0,290	0,285
0,5	0,680	0,633	0,635	0,499	0,635	0,365	0,295	0,279	0,280	0,270	0,276	0,270	0,275	0,274	0,275	0,284	0,268	0,279	0,274	0,292
1	0,666	0,585	0,586	0,480	0,536	0,406	0,405	0,406	0,396	0,391	0,367	0,373	0,367	0,372	0,352	0,348	0,353	0,360	0,330	0,349
1,5	0,676	0,616	0,616	0,523	0,535	0,523	0,531	0,518	0,516	0,520	0,514	0,508	0,506	0,507	0,509	0,512	0,504	0,502	0,499	0,500
3	0,686	0,637	0,635	0,637	0,630	0,634	0,639	0,639	0,635	0,641	0,636	0,643	0,634	0,639	0,632	0,631	0,639	0,639	0,632	0,634

(b) Sequential learning

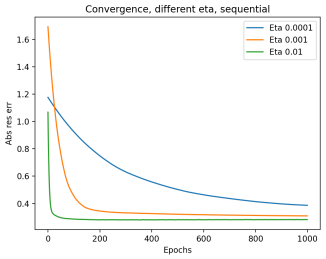
Figure 2: Effect of the number of RBS units and their width, average over 10 simulations on the Sinus-dataset.

The main point that differs between the different learning schemes is the rate of convergence. Batch learning is amazingly fast and does not require the decision of a proper learning rate, while sequential learning is a bit slower. The rate of convergence and its dependence on the learning rate eta for sequential learning can be seen in figure 3a.

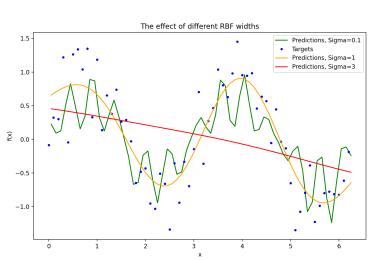
The main effect of changing the width of RBFs can be seen in *Fig. 3b*. A σ being too low (green curve) makes the RBFs too narrow, so that they can only predict values at the specific points where the RBFs are placed in input space, i.e. increasing variance too much at the loss of generalisation. This makes the predicted function curve very "wobbly" and input points in-between the RBFs cannot be predicted very well. A σ too large on the other hand (red

curve) flattens the predicted function curve in an evident way and provides some kind of function average rather than imitating the actual underlying pattern. This implies that too many RBFs cooperate to decide the predicted function curve, some predicting values as negative and some as positive, leaving the final prediction somewhere inbetween, i.e. reducing variance too much at the loss of generalisation. Thus, choosing a proper RBF width seems crucial. The orange curve with σ equal to one seems to simulate the intended sinus curve well, incorporating just the right amount of variance to understand the underlying pattern, thus ignoring most of the noise and generalises well.

Positioning the RBF nodes in the input space is very important, especially since well placed nodes can reduce the chance of dead units and overfitting. Up until now, the RBFs have been placed evenly in the input space. From our tests, an even placement seems to be better than random positioning, as can be seen in the quantitative evidence in *Fig. 3c.*



(a) Rate of convergence, different eta



(b) Different RBF widths

Sequential

Error

	Even	Random
Sinus	0,275	0,32
Square	0,404	0,474

STD

	Even	Random
Sinus	0,007	0,025
Square	0,012	0,029

Batch

Error

	Even	Random
Sinus	0,265	0,306
Square	0,363	0,373

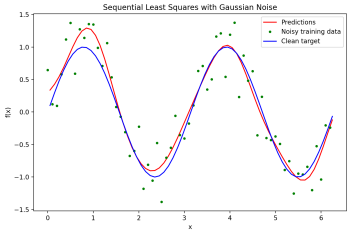
STD

	Even	Random
Sinus	0	0,009
Square	0	0,033

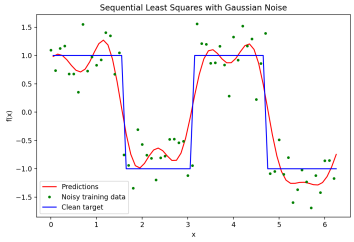
(c) Comparison of even and random placement of RBFs, 10 simulations

Figure 3: 15 RBFs, $\sigma = 0.5$.

Training the model on noisy data and testing it on clean data showed how well the model generalised and could learn the underlying pattern even though noise was indeed incorporated in training. The predicted function curves (batch mode) and quantitative results of doing this can be seen in figure *Fig. 4.* It is striking both visually and quantitatively how well the model have learnt the true underlying pattern (clean data) even though it was trained on noisy data.



(a) Sinus function



(b) Square function

Sequential

Error

	Noisy	Clean
Sinus	0,275	0,127
Square	0,404	0,300

STD

	Noisy	Clean
Sinus	0,007	0,0105
Square	0,012	0,018

Batch

Error

	Noisy	Clean
Sinus	0,265	0,091
Square	0,363	0,244

STD

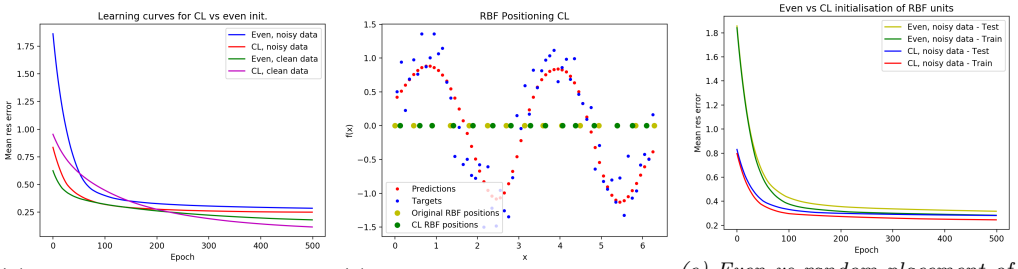
	Noisy	Clean
Sinus	0	0
Square	0	0

(c) Comparison of clean vs noisy test data, 10 simulations

Figure 4: Results of noisy training and clean testing. 15 RBFs, $\sigma = 0.5$.

Lastly, the optimal RBF network trained in batch mode was compared with a single-hidden-layer perceptron (TLP) trained with backprop and with the same number of hidden units (15). This was done only for the noisy case. The mean residual error of the TLP was 0.491 for the sinus function and 0.741 for the square function. This is to be compared with the values in *Fig. 4c.* It becomes clear that the RBF network performs a lot better, in relative terms around twice as good for both functions. On top of this, the RBF network trains very quickly, as it only requires one epoch.

3.3 Competitive learning for RBF unit initialisation

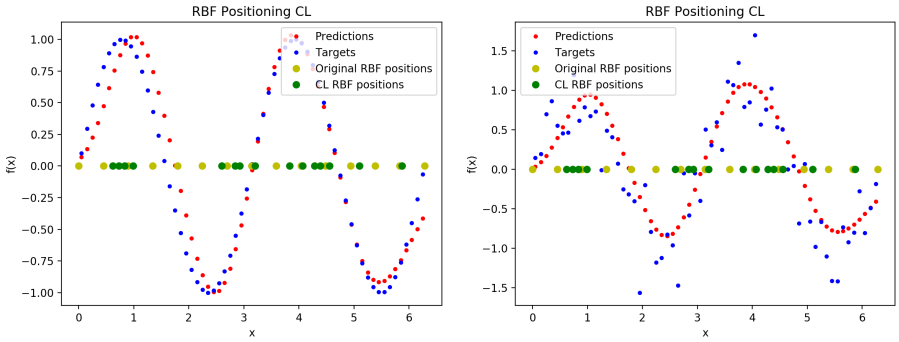


(a) Mean residual error on test data. *(b) Node positioning, even vs CL for noisy data.* *(c) Even vs random placement of RBFs*

Figure 5: 500 epochs of an RBF network with 15 nodes, with CL-initialization for 500 datapoints. $\sigma = 0.5$, $\eta_{CL} = 0.25$.

The CL approach for the nodes initial positioning appeared to make the model learning curve converge faster than the even positioning for the noisy data, as can be seen in *Fig 5a and c.* However, on the clean dataset, the even positioning generated a much faster convergence but at a significantly higher error rate, see *Fig 5a.* The difference in error between train and test data showed to be small for the two initialisation approaches, both on noisy and clean data, which implies good generalisation for both methods. The learning curves on the noisy train and test data for the two methods are presented in *Fig 5c.* It was also clear that the noise in the data is not taken into account when the model predicts the output. This showcases how it is able to generalise and find the underlying pattern even though there are noise involved in the data.

We implemented the "neighbours" technique of the SOM-algorithm to avoid dead units, i.e. not only the weights of the "winning" node was updated, but also its neighbours based on distance in input space. As expected, the RBF units appeared more clustered, see *Fig 6a, 6b* compared to *Fig 5b.* The algorithm was able to properly predict the target sinus function where the RBF units are clustered, but not so well where there are no units - which goes in line with the reasoning in section 3.2.



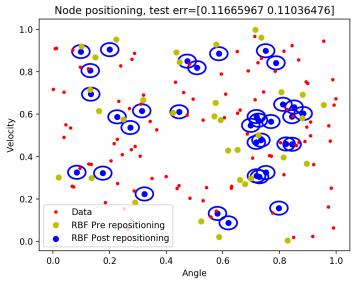
(a) Positioning, clean data. *(b) Positioning, noisy data.*

Figure 6: 500 epochs of an RBF network of 15 nodes, with CL-initialization of 25 iterations of random training vectors of length 20, the neighbourhood radius 2, gradually decreasing by 0.1 to 0, $\sigma = 0.5$ *and* $stepsize_{CL} = 0.05$.

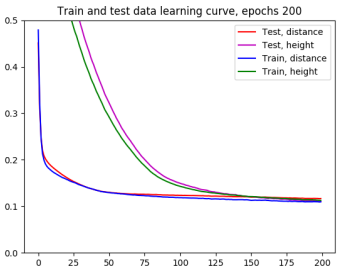
3.3.1 2D to 2D-mapping

When moving the nodes towards the datapoints in input space, a much lower step size is required than previous experiments, as input space is between 0 and 1. We began the RBF units randomly placed in input space, and moved them with the CL-algorithm with a neighbourhood based

on a decreasing neighbourhood to avoid dead units. The results were of course depending on parameters, and from parameter tuning of nodes and standard deviation of the RBF units, the best results were gained from 34 nodes and $\sigma = 0.25$. In *Fig 7a* we see how the nodes have shifted in input space and in *Fig 7b* we see learning curve of 200 epochs. In general for the experiments done, the model performed better on height than on distance. The Cl-initialization had significant impact on performance, and the decreasing neighbourhood proved effective in avoiding dead nodes



(a) Positioning after 25 CL-iterations with vectors of 20 datapoints and stepsize of 0.05 and neighbourhood decreasing from 0.5 to 0.



(b) Learning curve, mean res. error vs epochs.

Figure 7: 2D to 2D mapping, 34 nodes, 200 epochs, $\sigma = 0.25$.

4 Results and discussion - Part II: Self-organising maps

4.1 Topological ordering of animal species

We implemented the self-organizing map (SOM) learning algorithm from scratch. As described in the instructions, we used an output grid containing 100 nodes, $\eta = 0.2$. The weights were also randomly initalized with values ranging from 0 to 1. We further also used a starting neighbourhood size of 40 and gradually made it smaller, down to a neighbourhood size of 0, within the 20 epochs. The attained results are feasible and seems to make sense. The final order is presented down below and as can be seen, insects are bundled together, as are for example birds, amphibians and felidae.

(horse, 2), (giraffe, 8), (camel, 9), (pig, 15), (spider, 21), (housefly, 27), (beetle, 31), (grasshopper, 34), (moskito, 38), (butterfly, 39), (dragonfly, 40), (pelican, 45), (duck, 49), (ostrich, 50), (penguin, 54), (seaturtle, 58), (crocodile, 60), (frog, 61), (walrus, 66), (hyena, 70), (dog, 71), (bear, 73), (skunk, 77), (ape, 78), (cat, 82), (lion, 84), (bat, 85), (rat, 90), (rabbit, 91), (kangaroo, 95), (elephant, 96), (antelop, 97)

4.2 Cyclic tour

When modifying the SOM algorithm slightly from the previous subsection, a reasonable route between the cities were found. We found that the algorithm only needed 3 epochs to find the shortest path. In this particular scenario, each epoch reduced the neighbourhood size incrementally from 2 down to 0 inclusive. As in the previous task, a learning rate of $\eta = 0.2$ was used. The resulting tour between the cities is presented in *Fig. 8*.

4.3 Clustering with SOM

In order to visualize the clusters of votes seen in *Fig. 9*, noise drawn from a Gaussian distribution with $\mu = 0.5$ and $\sigma = 0.25$ was added to the output when plotting. The SOM learning algorithm

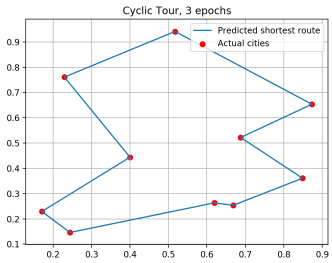
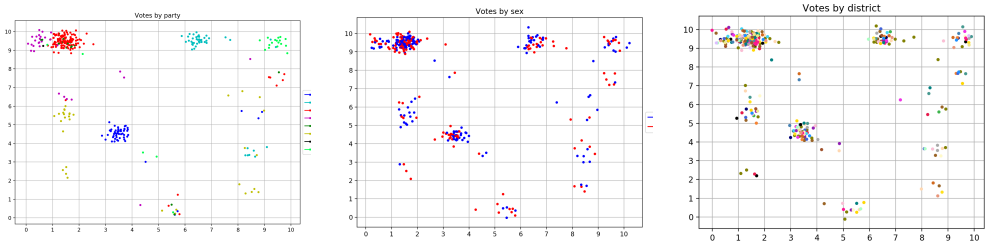


Figure 8: Predicted shortest route through the given cities. Neighbours starting at 2, decreasing by 1 for each epoch to 0.

was implemented using $\eta = 0.2$, 40 epochs and a neighbourhood size starting with 5, which eventually shrunk down to 0, where the neighbouring nodes in the grid were calculated using the Manhattan distance.

In figure $9a$ where each dot represents a member of parliament, we can clearly see that there is a correlation between what party the member belongs to and how they vote. In addition, we found that certain parties vote similarly. This is displayed in the top left hand corner by Vansterpartiet(red) and Socialdemokraterna(purple). This makes sense since the two parties have similar underlying political views.

From figure $9a, 9b$ we did not identify any clear patterns on how sex and district is correlated to voting.



(a) Votes by party. *(b) Votes by sex.* *(c) Votes by district.*

Figure 9: SOM for data clustering of MP votes using 40 epochs, $\eta = 0.2$ and a starting neighbourhood size of 5.

5 Final remarks

The lab was good in such that the level of difficulty was suitable. However, we do believe that many questions are asked in such a way that the range of how thorough an acceptable solution can be is huge. Perhaps this is a problem for ourselves, not wanting to present a question halfway done, rather than a problem of the formulation. Furthermore, much time was put on practical issues rather than theoretical - e.g. for data visualisation. The hints of how the results preferably should be presented were much appreciated.

References

Marsland, S. (2015). *Machine learning: an algorithmic perspective*. CRC press, 2 edition.