Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Beata Johansson, Elin Saks and Simon Larspers Qvist

October 1, 2023

1 Main objectives and scope of the assignment

Our major goals in the assignment were:

- to explore a method for representing high dimensional data in a low dimensional space.
- to see how SOM networks can be used in practice.
- to get a thorough understanding of RBF networks and their implementation.
- to implement and test out Competetive Learning.

Our scope is limited to two methods for learning weights for the RBF networks but there are many more to explore. For SOM networks we have used Euclidean distance for the similarity measure but there could be outer measurements there as well.

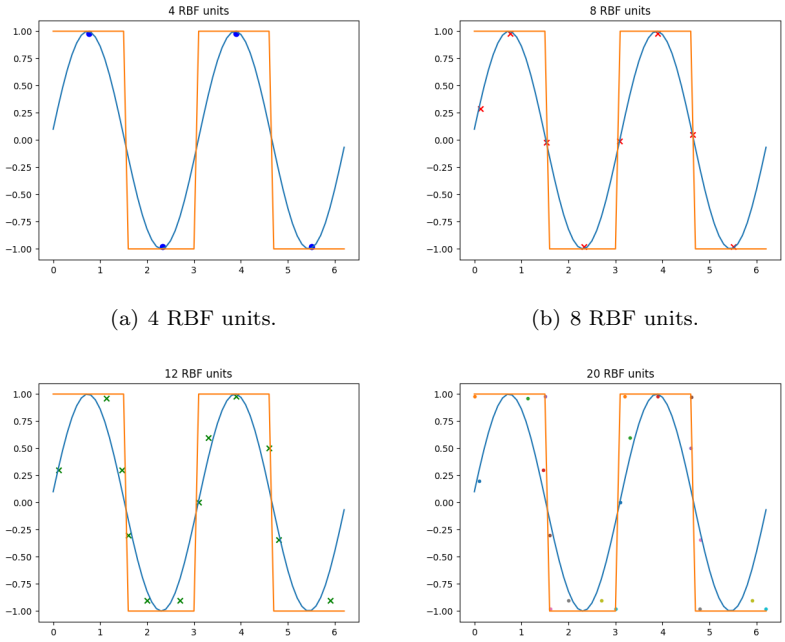
2 Methods

The assignment was done in python, using numpy for vectors, matrices and some mathematics, and matplotlib for plotting. No libraries for neural networks were used.

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

3.1 Function approximation with RBF networks

The choice of the RBF units was done manually and they are shown in figure 1. These four cases was applied to both the sin(2x) and box(2x) data. The RBF units were placed in such a way that we thought they could represent the two curves in the best way. We started with the fewest RBF units, the one with only four got its units placed in the extreme points. When increasing the number, we started placing the units in the spaces in between the previously placed units.



(a) 4 RBF units.

(b) 8 RBF units.

(c) 12 RBF units.

(d) 20 RBF units.

Figure 1: RBF network unit selection for four RFB networks.

We tried different variances for the different numbers of RBF units. As seen in tables 1 and 2 we achieve residual error below 0.1 when we have 20 units and variance 0.1, for the other number of units we need about 0.6 variance. Regarding getting the residual error below 0.001, we seem to only achieve this when we have 20 units and a variance of 2.0. The tendency is that more units and higher variance tend to make better-fitting curves. Experimentation showed us that increasing the variance too much won't help in lowering the residual error. For example, setting the variance to 1000 for the different numbers of RBF units will result in all of them having a residual error of about 0.0005, which isn't that much better than what is shown in the tables 1 and 2. This is because

increasing the "radius" of the RBF units too much will not automatically make them improve the regression.

The number of RBF units and their widths are dependent on each other. Fewer nodes require a wider area each unit covers, i.e. a larger variance. If you have many nodes, a smaller variance can be used. It is also possible to have different variances for the nodes and adjust the placement of nodes so they gather in different places in the space, depending on where the data is located.

To get the residual error to 0 for $square(2x)$ one can apply a step function to it.

Number of RBF units	**Var = 0.1**	**Var = 0.6**	**Var = 1.0**	**Var = 2.0**
4 units	0.214	0.053	0.074	**0.074**
8 units	0.204	0.043	0.058	**0.054**
12 units	0.172	0.062	0.058	**0.032**
20 units	0.073	0.008	0.004	**0.001**

Table 1: Absolute residual error after evaluation on the test set for sin(2x) in one forward pass.

Number of RBF units	**Var = 0.1**	**Var = 0.6**	**Var = 1.0**	**Var = 2.0**
4 units	0.473	0.127	0.082	**0.045**
8 units	0.410	0.126	0.082	**0.036**
12 units	0.137	0.024	0.011	**0.003**
20 units	0.067	0.001	0.0002	**4.125e-05**

Table 2: Absolute residual error after evaluation on test set for square(2x) in one forward pass.

In the next part of the assignment we added Gaussian noise to the training data and evaluated the RBF networks on the test data. In Figure 2 the predictions from a 4, 8, and 12 RBF network are shown. The placement of these RBF nodes is the same as before, see Figure 1. All networks were trained with the on-line learning method. We can see that the predictions are close to a $sin(2x)$ curve.

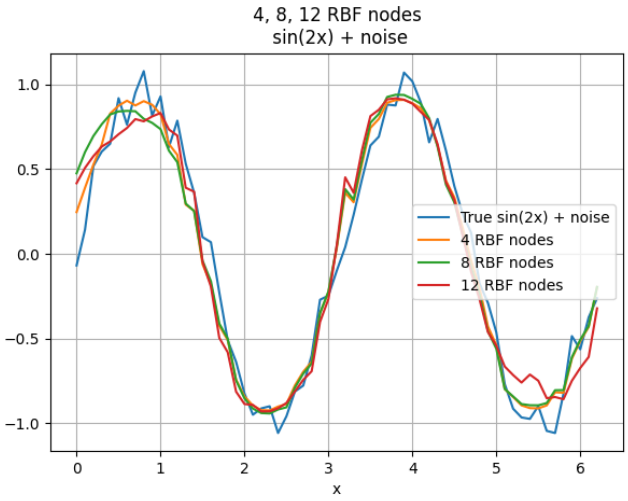
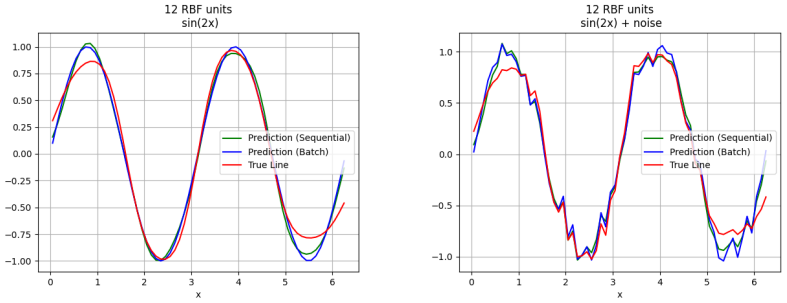


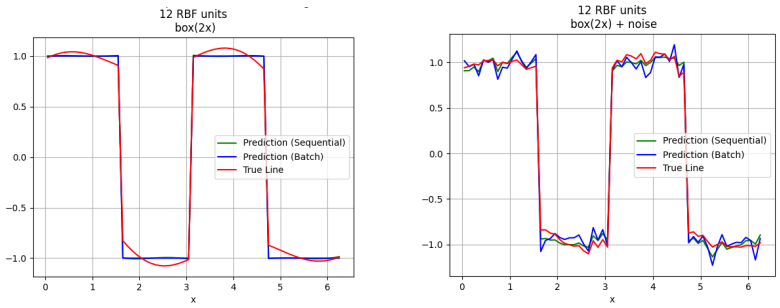
Figure 2: Prediction on test data (sin(2x) + noise) with Batch learning on different RBF network configurations (4, 8, 12 nodes).

For 12 RBF nodes, predictions on noisy and clean data was made in Figure 3 and 4.



(a) sin(2x). (b) sin(2x) + noise data.

Figure 3: Prediction on sin(2x) test data for RBF network (trained with batch and sequential learning).



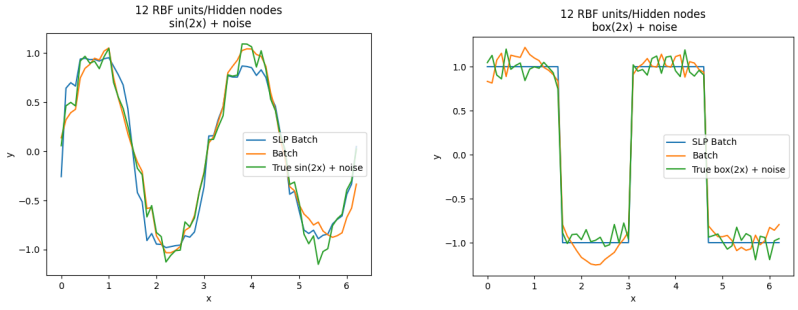
(a) box(2x). (b) box(2x) + noise data.

Figure 4: Prediction on box(2x) test data for RBF network (trained with batch and sequential learning).

In Figure 5 we compared the average residual error on the test set for the online and batch learning methods on noisy data. Even though the differences in errors are small, it seems like a higher variance and more RBF nodes lower the error. The learning methods seem to give approximately the same results.

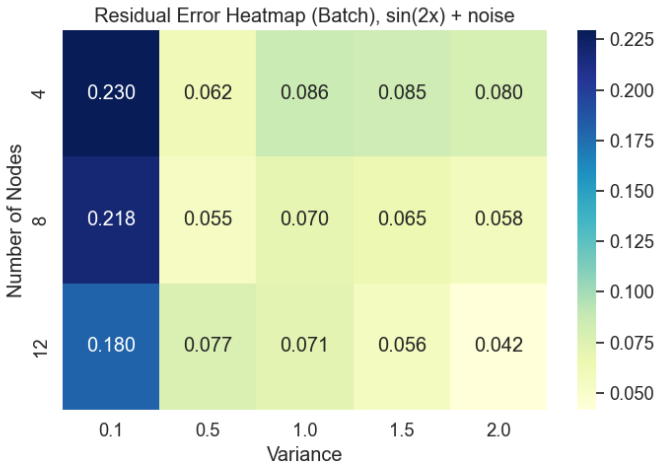
In Figure 6, the effect of the learning rate on the sequential learning convergence was measured with the average residual error per epoch on the test set. The figure shows that a learning rate of about 0.05 or 0.1 leads to the fastest convergence at approximately 40 epochs. A too small of a learning rate, like 0.001, does not let the network converge.

In Figure 7 we compared the predictions on noisy data from RBF network with 12 nodes to our Single Layer Perceptron (SLP) network from the previous labs. The SLP was trained with the generalised delta rule, and the RBF was trained with batch learning with least squares method. Both networks had similar predictions on the sin(2x) data, but on the box(2x) data it seems like the SLP had closer predictions to the targets. The learning rate was 0.1 and 30 epochs.

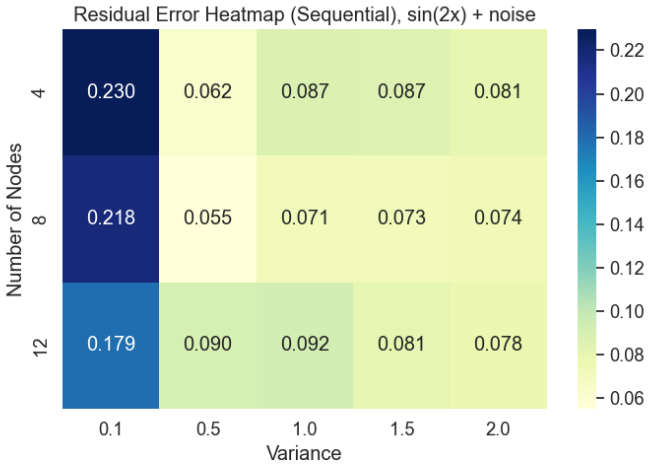


(a) sin(2x) + noise data. (b) box(2x) + noise data.

Figure 7: Prediction on test data for Single Layer Perceptron (trained with delta rule) and RBF network (trained with batch learning).



(a) Sequential learning with delta rule.



(b) Batch learning with Least Squares.

Figure 5: Average Residual Error after one forward-pass on test data after Sequential and Batch learning on RBF network for sin(2x) + noise data.

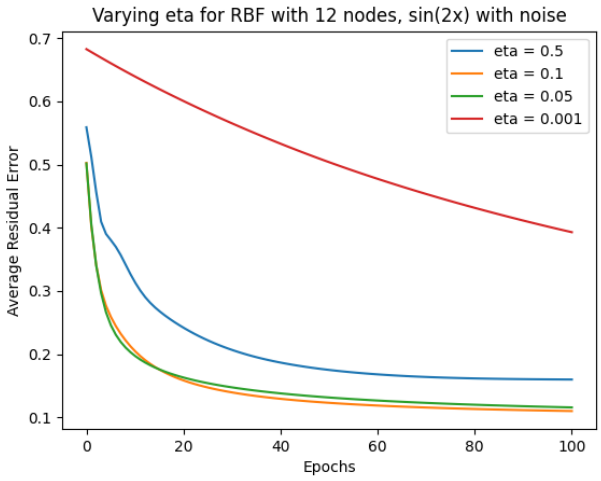


Figure 6: Average Residual Error per epoch on test data (sin(2x) + noise) with Sequential learning on RBF network.

3.2 Competitive learning for RBF unit initialisation

In figure 8 we see how the nodes are placed during the competitive learning. In that figure, we compare how our strategy for avoiding dead nodes performs compared to the one with no strategy. For this comparison, we did sequential learning with noise. In this specific example, we avoided 3 dead nodes with our "share win" strategy. This strategy meant that the winning node's 2 closest neighbours also got a small adjustment: for the closest neighbour we reduced eta to 60% of the original value and for the second closest we reduced it to 40%. This way we get a smaller move for the neighbours while giving more nodes a chance to move, which avoids dead nodes.

For the regression task with ball data, we implemented the RBF with random initialisation and competitive learning for choosing the RBF nodes, and two output nodes were needed. Parameters like variance, learning rate and number of RBF all had an effect on the performance. We found from testing that high number of RBF nodes gave lower errors in general. Having a low learning rate (like 0.01) slowed down the competitive learning process, which made the node updates too small. A very large learning rate (like 3) made the updates too large. We used a variance of 0.5, which gave reasonable results. However, we found from testing that a too-low variance (like 0.1) made predictions worse. The resulting predictions are shown in Figure 9.

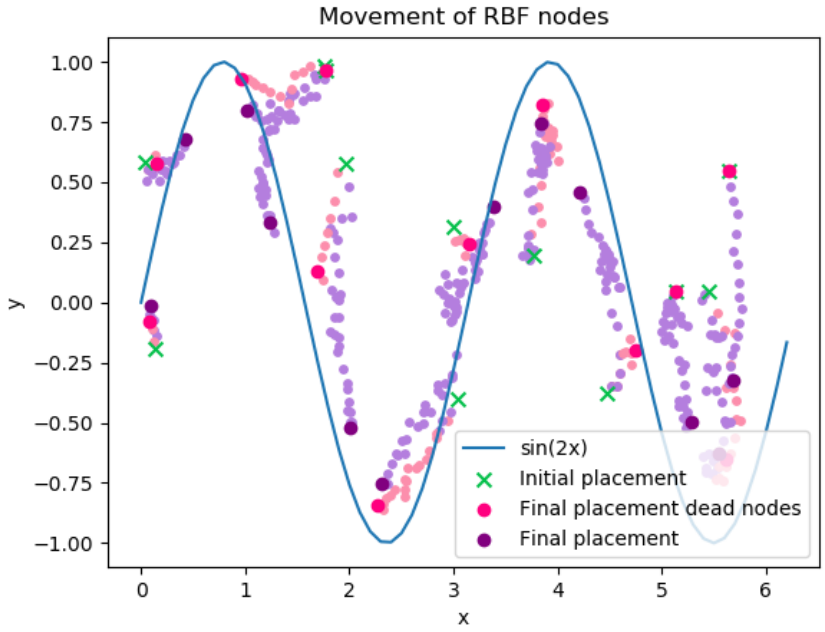
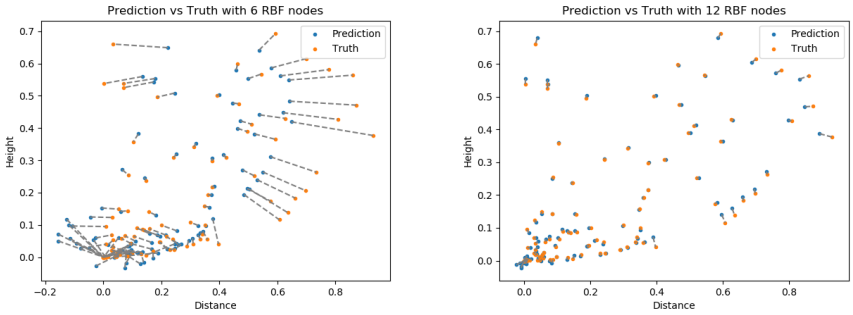


Figure 8: Traces of how the CL algorithm makes the RBF nodes move during learning, sharing win strategy vs no strategy to avoid dead nodes. No noise.



(a) 6 RBF units.

(b) 12 RBF units.

Figure 9: Results from the ball task. Comparison of predictions of (Distance, Height) from 6 and 12 RBF nodes.

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

4.1 Topological ordering of animal species

3 columns with the output ordering of the 32 animals, meaning spider and giraffe are the least similar.

spider	frog	rat
housefly	seaturtle	bat
moskito	crocodile	elephant
butterfly	walrus	kangaroo
dragonfly	dog	rabbit
grasshopper	hyena	antelop
beetle	bear	horse
pelican	lion	pig
duck	cat	camel
penguin	ape	giraffe
ostrich	skunk	

4.2 Cyclic tour

Figure 10 shows the tour the SOM network produced that takes a route close to all the cities.

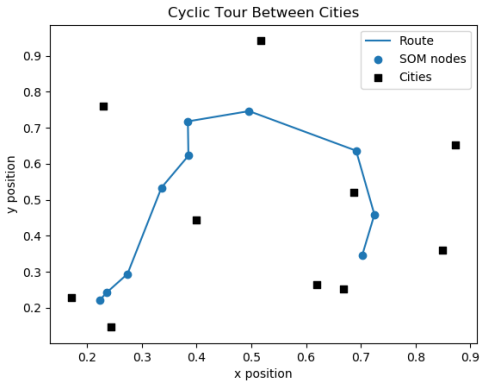
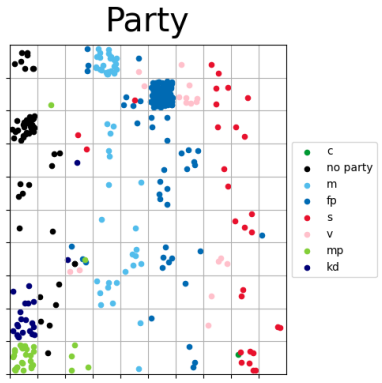


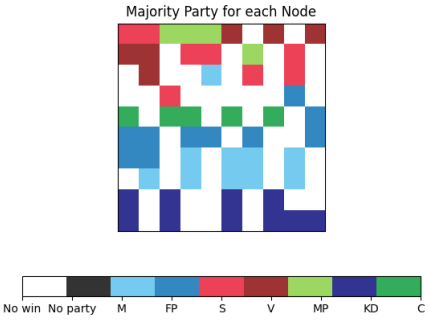
Figure 10: RBF nodes positioned to find a tour of cities.

4.3 Clustering with SOM

In this exercise, we mapped MPs onto a 2D grid, based on their voting patterns, with the help of a SOM network with 100 output nodes. In figure 11 we mapped them based on the party they belong to, visualised two different ways. (The two plots are from different runs.) In 11(a) we scattered all 349 MPs onto the grid, the box they are in is the node whose weight vector their voting pattern was most similar to. In the heat map, we instead coloured each output node after the party which had the largest number of MPs associated with that node. Both in figure 11(a) and in figure 12 a small amount of noise was added to the indices in order to make each individual person visible.

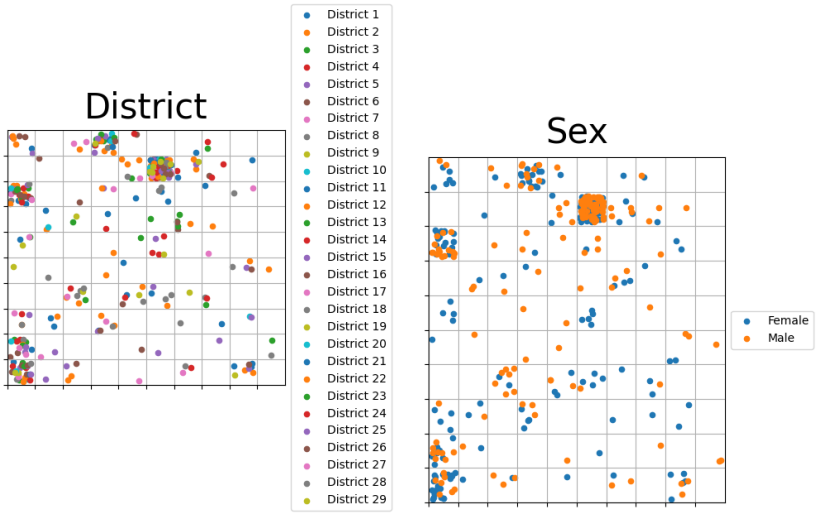


(a) Clustering of parties depending on their votes.



(b) Heat map of parties' votes. Each node takes the colour of the party that is most often associated with it.

Figure 11: Different visualisations of parties' general opinions, from votes.



(a) Clustering of districts depending on their votes.

(b) Clustering of districts depending on their sex.

Figure 12: Clustering on districts and sexes with regards to the votes.

4.4 Discussion

In section 4.1 we can see a list of grouped animals. The assignment was to order the animals with a SOM algorithm. What happened here was that we, with the SOM algorithm, looked at the attributes of each animal and grouped similar ones together. The algorithm seems to have performed in a satisfactory way. Insects and spiders appear first, followed by birds (reasonable since these have wings, and so do most insects). After the birds come reptiles, who also lay eggs. Then come mammals.

In section 4.2 we expand from the one-dimensional list to a two-dimensional space. In this part, we made the neighbourhood cyclical, that the *first node* is neighbour with the *last node*. In figure 10 we can see how the SOM network manages to find a route that passes close by all of the cities.

In section 4.3 we clustered data on politicians and their votes. First, we did this with regards to their party affiliation, then district and lastly sex. We can from figure 11 see that party members tend to agree with each other. They are generally positioned in the same column or row. In the heat map, we can more easily see that the left and right wings tend to agree with each other somewhat as well. However, looking at the images in figure 12 we can not make any clear correlation between either districts or sexes and how the politicians vote. These results are reasonable since party affiliation is most likely the main ideology of the politician, the sex of the politician likely won't affect their political view that much.

5 Final remarks

The lab was interesting. However, it was also more time-consuming than the previous ones (which were also time-consuming). There were a lot of questions and different things to try out, especially in assignment 1. Since there were so many questions in assignment 1, it was very difficult to try to answer them all in just 2.5-3 pages. It felt like assignment 1, task 2 (Regression with noise) was unnecessarily long. However, think that assignment 2 was very interesting and also pretty fun to work on! It was also fun to work with actual data in both assignment 2 as well as assignment 1 - task 3, question 3.