Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Alice Karnsund, Elin Samuelsson and Irene Natale

September 18, 2018

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to build and train an RBF network for regression purposes
- to implement the core algorithm of SOM and use it for different tasks
- to find a low-dimensional representation of higher-dimensional data

2 Methods

We used `Matlab` throughout in this lab without any use of toolboxes.

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)*

We used the absolute residual error throughout this section. n denotes number of RBF nodes and σ their standard deviation. Plots with errorbars have been generated by either 10, 50 or 100 training sessions.

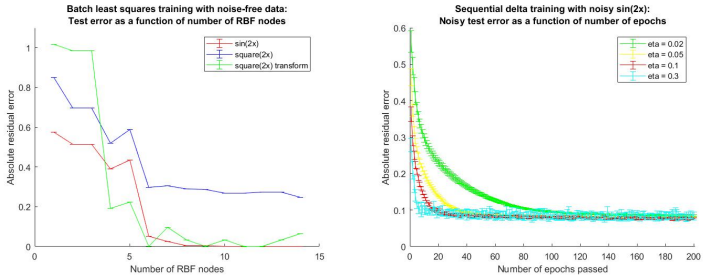

Figure 1: Left: Test error as a function of n. Batch training and testing on clean data, σ=0.9. Right: Test error as a function of number of epochs. Sequential training and testing on noisy sin(2x), n=10, σ=1.2.

3.1.1 Batch mode training using least squares

Figure 1 Left shows that n = 6, 8 and 10 was needed to fulfill the error thresholds for the sin data. None of them was fulfilled for the square data. However, when rounding the square output to 1 or -1, only n=6 were needed for zero error. This transformation is convenient for classification. For decreasing σ and/or increasing n, the normal equations eventually result in *Matrix is close to singular or badly scaled.*

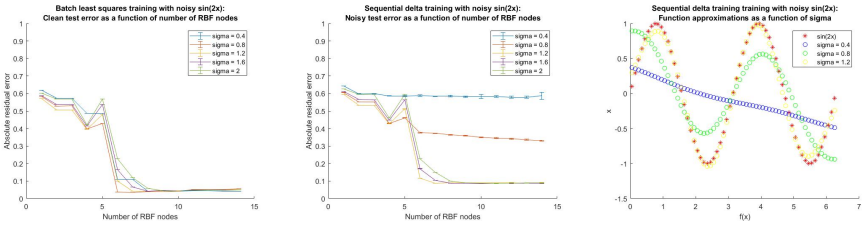

Figure 2: Left and Middle: Test error as a function of n. Batch (left) vs sequential (middle) training and testing with noisy sin(2x). Right: Function approximations of sin(2x). Sequential training with noisy data, n=9, η=0.02.

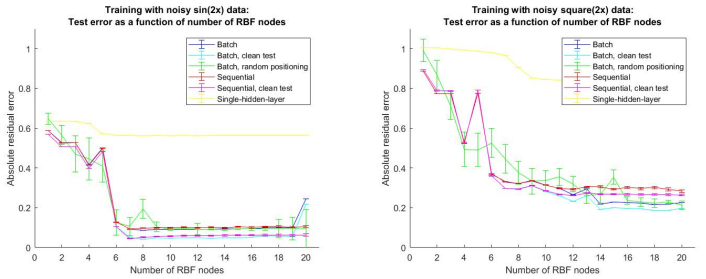

Figure 3: Test error as a function of n, η=0.05, α=0.5, σ=1.2 (left) or 2.0 (right).

3.1.2 Regression with noise

Figure 1 shows that for sequential training, increasing η leads to a faster, but less stable algorithm (i.e. larger std). Batch learning does not depend on η, the optimal $\mathbf{w}$ is found in one step. Figure 2 shows that best results for sin(2x) are obtained when n $\geq$ 8. Sequential training improves significantly when $\sigma \geq 1.2$, while batch learning is almost unaffected. For too small σ, the RBFs are too broad to make a good function approximation.

We used uniform positioning, which perform well except for n=5 since all RBFs then are centered where f(x)=0. Figure 3 show that random positioning gives less stability, but comparable results on average. Testing on clean data improves the results more for sin(2x) than square(2x), thus sin(2x) gives better generalisation performance. The single-hidden-layer perceptron performs worse than the RBF network, both in terms of generalisation performance and training time (batch converges in one step).

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

Figure 4 shows results for the function approximation of $f(x) = \sin(2x)$ over $[0, 2\pi]$. Adding the SOM algorithm to the delta learning gives better performance for both noisy and clean data. If we update the first and second closest weights together with the "winner", the performance increases. For more neighbors, we risk moving some RBFs in not-optimal positions.

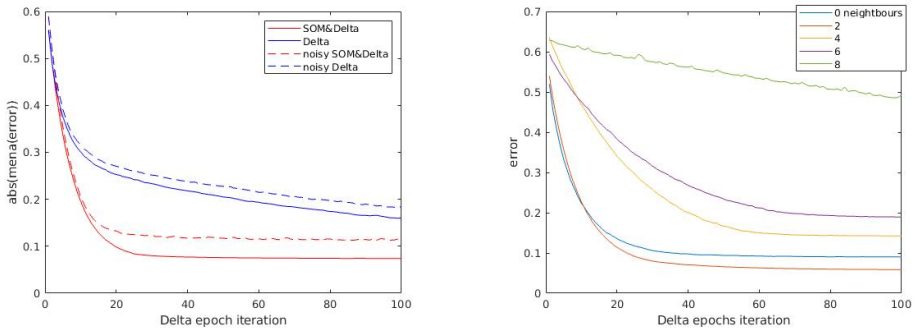

Figure 4: Left: Error value of Delta learning with and without SOM, on noisy and clean dataset. Right: Error value for different number of neighbours.

Figure 5 shows results for 2-dim input and 2-dim output data. The first image shows an example of the final 2-dim RBF means generated by the CL algorithm, using Manhattan distance and including neighbours with index difference ≤ 4. The right figure shows the test error over the Delta algorithm for the 2 output dimensions. The decreasing behaviour implies convergence.

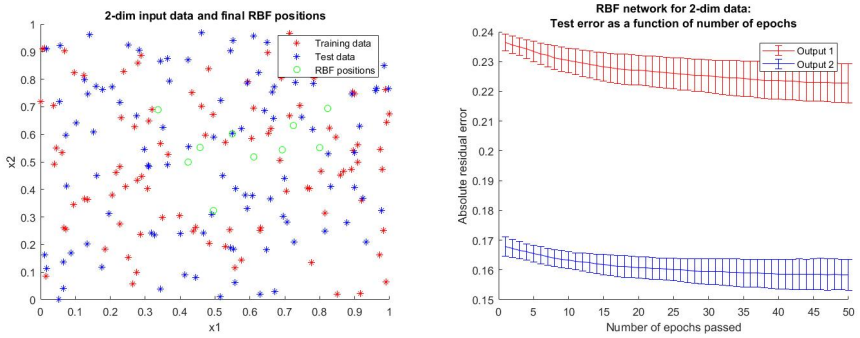

Figure 5: Left: An example of the final RBF positions after CL, neighborhood threshold = 4, $\eta_C L$=0.2. Right: The test error as a function of number of epochs passed in the sequential delta training, σ=20, $\eta_d elta$=0.02.

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

4.1 Topological ordering of animal species

The size of the neighbourhood is depending on the epoch loop variable. Starting with a size of 50 and ending with a size 0. The result clearly depends on how drastically this size is decreasing. In the beginning the neighbour size may decrease pretty fast but for the last epochs, ($\sim$ 10), the size has to decrease very slowly, and mostly keep the same size for more than one epoch. Otherwise more than one pattern may be mapped to the same node in the output grid, and thus the network will not output the full list of animals. For the neighbourhood size we made use of only the following sizes: $[25, 19, 16, 9, 4, 3, 2, 1, 0]$. With 20 epochs and $\eta = 0.2$ we got the following neighbourhood sizes and result.

Neighbourhood: [25, 25, 19, 19, 16, 16, 9, 9, 4, 4, 3, 3, 2, 2, 1, 1, 1, 1, 0, 0, 0]

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

No collisions to the same output node occurred whenever the number of epochs was ≥ 10.

4.2 Cyclic tour

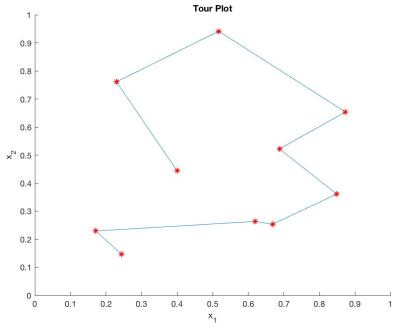

Figure 6: The SOM-algorithm finds a fairly short route which passes through all cities. But it is not guaranteed to always find the shortest route.

In this part the output grid is only 10 nodes, i.e. the cities, all of them has to be in the result for the travelling salesman to fulfill his task. With only 10 outputs, different input patterns may end up at the same node in the output grid. Therefor we sort the `pos` vector in the following way:

```
result_indices(pos)
    sort_pos = pos vector in increasing order
    result_indicies = []
    for every element in sort_pos, find its index in pos
        and add it to result_indices
```

In this way, no repetitions will occur and the algorithm converges in the sense that all cities are included in the output.

4.3 Clustering with SOM

Figure 7 shows the result of the topological mapping from the 31-dimensional `votes.dat` dataset onto the 10×10 output grid, with respect to three different attributes; gender, districts and parties. From the figure we can draw the conclusion that it is mainly the party membership that affect the way the MPs vote. The other two attributes seem to have extremely little or no impact on the voting scheme. This might though play a part if the data was considered for the whole country and not only the parliament.

Just as in the previous task, some inputs mapped to the same output node. To make the display of the results clearer, we simply added some noise to the resulting coordinates in the output grid. Due to this the axes in 7 are not strictly set to go from 0 to 10.

In this part we sat the neighbourhood to represent a square rotated 45 degrees, with the winner node in the middle. The width, distance from winner node to one of the corner nodes in the rotated square, decreased from 5 to zero. If some part of the neighbourhood ended up outside the grid, the neighbourhood were set to be the resulting part.

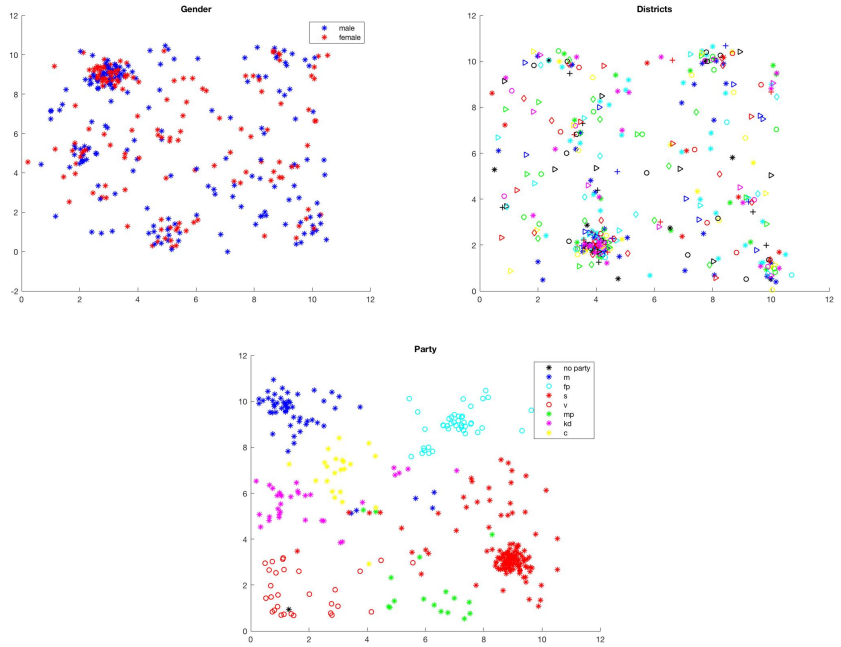

Figure 7: Results of the SOM-algorithm, 20 epochs and $\eta = 0.2$, on the `votes.dat` dataset with respect to Genders, Districts and Parties respectively.

5 Final remarks *(max 0.5 page)*

For the one dimensional function approximation case, the neighbourhood size has a strong influence on the convergency rate. A few neighbours can speed up the decrease of the error, while too many neighbours can lead the convergence to slow down.

In the second part of the assignment it was clear that the neighbourhood size had a great impact on the result. A main discovery was that the neighbourhood size must decrease very slowly for the last epochs to make the detailed positioning correct. If the size was not monitored carefully towards the end, task 4.1 for example, responded by positioning an animal among others with no similarity.