Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Anton Anderzén, Styliani Katsarou and Bas Straathof

September 20, 2019

1 Main objectives and scope of the assignment

Our major goals for the assignment are:

- to understand and implement different RBF networks using batch and sequential learning for approximating functions with or without added noise.
- to explore the differences between manually positioning RBF units and the CL approach.
- to implement SOMs to find low dimensional representations of higher dimensional data, and describe their behavior.

All experiments have been conducted as described in the original assignment. No additional assumptions were made, and no limitations were encountered.

2 Methods

All algorithms are implemented from scratch using Python 3.7. No machine learning or neural network libraries were used in this assignment.

3 Results and discussion - Part I

3.1 Batch mode training using least squares - supervised learning of network weights

The least squares learning method is used to train an RBF network, which entails a batch learning method. A grid search among different variance values and number of hidden nodes was performed, to determine the best possible variance value for the RBF centres, in terms of achieving the smallest error.

Variance values in the range of $[0, 0.5]$ were used to find that the smallest average error can be achieved when the variance is set to 0.25. The variance was the same for all RBF centres and their means were initialized by randomly sampling from a uniform distribution varying within the same limits as the input data. The residual error was subsequently measured by varying the amount of hidden nodes. The error starts to decrease significantly for more than 6 nodes, which corresponds of the sinusoidal wave to be approximated. The smallest error value was obtained by an RBF of 23 nodes (see Figure 1).

Table 1: Error comparison between $sin(2\pi)$ and $square(2x)$ for a varying number of RBF nodes

number of nodes	2	4	6	13	23	63
$sin(2\pi)$	0.320	0.313	0.156	0.002	0.0003	8.314
$square(2x)$	0.046	0.048	0.108	0.041	0.043	0.082

As can be seen in table 1, the square function approximation results were not as good. Nevertheless it was possible to reduce the error by transforming the outputs so that the negative ones would covert to -1 and the positive ones would convert to 1.

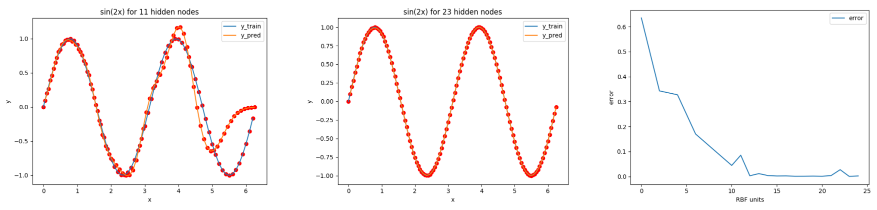

Figure 1: (left) Function approximation for 11 nodes; (middle) optimal function approximation with 23 nodes (right) the residual error vs. the amount of nodes

3.2 Regression with noise

Next, after adding Gaussian noise to the data, the results of the RBF network function approximation were compared: both the batch mode using the least squares algorithm and on-line learning mode using delta rule. In terms of rate of convergence, least squares is faster since it is a batch learning method. Consequently, it does not depend on the learning rate whereas the delta rule is highly affected by varying the learning rate values. Based in the test error as a comparison criterion, the networks would both underperform when the width of the RBF centres were too low (lower than 0.1) or too high (higher than 0.5) possibly because then the nodes would overlap, but the effect of the nodes being too narrow was worse than that of being too wide. Initialization of the RBF nodes is of great importance: placing them randomly in the input space produced ten times bigger error than evenly placing them with equal distances between them. The affection of RBF centres initialization on function approximation performance was even more observable for lower values of variance of the sinusoidal function on noisy data. This might be related to the fact that our model in delta rule mode would overfit when making predictions out of noisy data, since the test error was 4 times bigger than the training error in this case, but no overfitting was observed on non-noisy data (see Figure 2) .

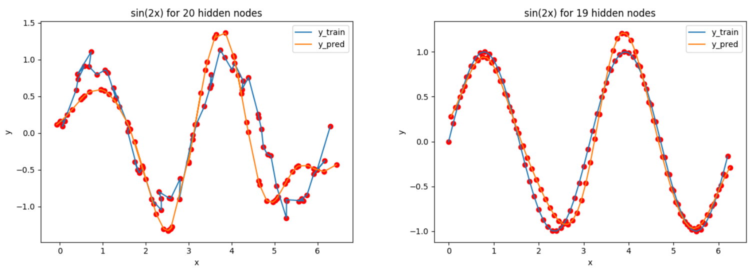

Figure 2: Function approximation of $sin(2x)$ using delta rule on noisy data on the left, and on data without noise on the right

3.3 Competitive learning (CL) to initialise RBF units

The competitive learning is used in the initialization of the RBF-units. In the simple winner takes it all version, the difference between a random data point from the input space is and each of the RBF-unit centers is calculated. The one RBF-unit closest to the randomly selected sample is the winner. After selecting the winning unit, it is moved towards the data sample.

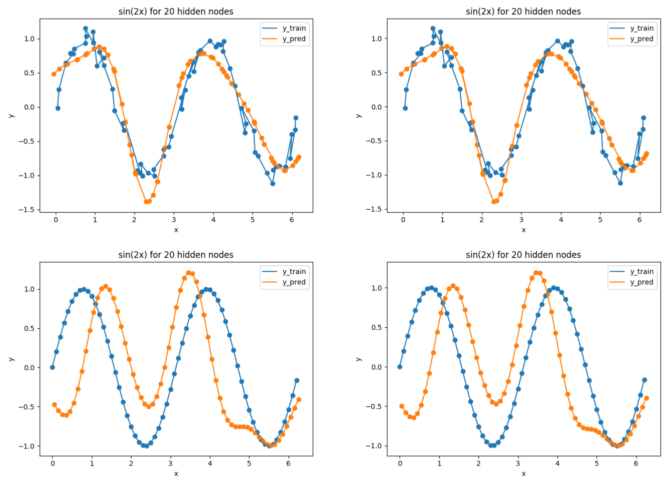

Figure 3: (Top Left) Noisy with CL, TestError = 0.0018; (Top Right) Noisy no CL, TestError = 0.0113; (Bottom Left) No noise, no CL, TestError = 0.0096; (Bottom Right) No noise with CL, TestError = 0.0120

An improved way with which competitive learning can be used to avoid ”dead units” is to not only update the winning unit, but neighbours of the winning unit as well. In our implementation of this concept called leaky learning, the 10 units closest to the winning unit are also updated. The update is stronger for units close to the data sample, and weaker for those further away, see Equation 1. To test this out we increased the RBF unit count to 100 exceeding the input sample space forcing it to be overdetermined. Without leaky learning and 100 RBF units the delta rule converged after 1062 epochs with the test error = 0.0153. With leaky learning and 100 RBF units the delta rule converged after 10 epochs test error = 0.0283.

The effects of this leaky learning can also be seen in Figure 4, here the RBF units are spreading out in the input space. In the top left plot the 10 RBF units are centered around the middle of the input space. This indicates that moving a fixed set of neighbours might not be the optimal way to initialize the units. However, for 20 RBF units the spread gets much better as not all of the units are adjusted for a singe data sample. This gets even more clear for the 30 and 50 RBF versions.

$$\Delta W = \frac{||W_{winner}| - |x||}{||W_{neighbour}| - |x||} * \eta * |x| - |W_{neighbour}| \tag{1}$$

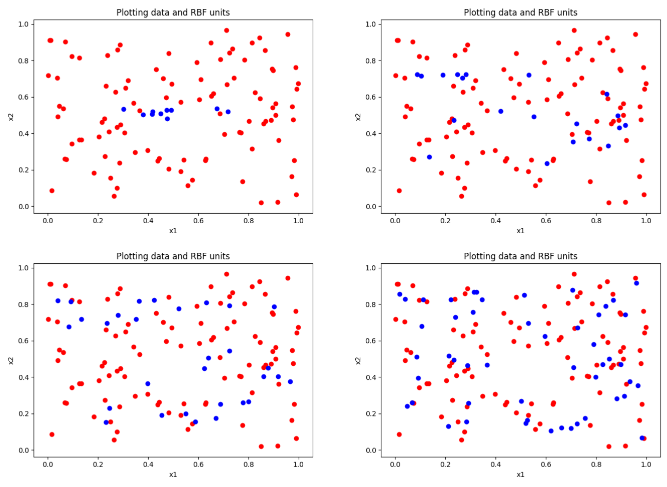

Figure 4: Inputs and RBF Units Errors: (Top Left) H=10: 0.0252, (Top Right) H=20: 0.0092, (Bottom Left) H=30: 0.067, (Bottom Right) H=50: 0.152

4 Results and discussion - Part II

The second part of the assignment consists of building a self-organizing map (SOM) architecture for topological ordering of data. The SOM finds a way to represent the high-dimensional input data in a lower-dimensional space. The inner workings of SOMs are explored through working on three problems: (1) topological ordering of animal species, (2) find a short path in a cyclic tour, and (3) data clustering based on votes of Swedish MPs.

beetle	0	hyena	48
grasshopper	0	dog	51
butterfly	1	lion	55
dragonfly	1	cat	58
moskito	2	ape	62
housefly	4	skunk	65
spider	9	elephant	69
duck	14	bat	71
pelican	15	rat	76
penguin	18	rabbit	80
ostrich	21	kangaroo	83
seaturtle	26	antelop	87
crocodile	29	horse	92
frog	34	giraffe	94
walrus	40	pig	96
bear	45	camel	99

Table 2: Topological ordering of animal species as represented by a SOM

4.1 Topological ordering of animal species

There are 32 animal species to be ordered. For each animal, a subset of 84 binary attributes is specified, hence, the size of the input is `(32, 84)`. The topological ordering of the animal species will live in a `(100, 1)` dimensional space. it can be seen that the SOM is able to find a sensible topological representation: insects are grouped together, and so are birds and large mammals (see Table 2). A note on implementation: a linear vectorization was applied such that only two for-loops had to be used instead of the three suggested for-loops in the assignment.

4.2 Cyclic Tour

The data consists of the x- and y-coordinates of ten cities (which, for convenience, have been called A-J), so the input dimensions are `(10, 2)`. The neighborhood computation in this problem is slightly more complicated than in the previous problem. The modulo function was used, to make sure that the neighborhood calculation is circular. From Figure 5 it can be observed that the representation is better when the initial neighborhood is of size 6 than of size 4.

4.3 Data Clustering: Votes of MPs

For this problem, several files were provided. The input data to learn the data clustering exists of the first 31 votes of the 349 Swedish MPs in 2004-2005. For each MP, there are associated files containing the name, sex, district and party of the MP. In this clustering task, the clusters have to be put on a grid of dimensions `10, 10`. To achieve this, a function was used that turns a one-dimensional array of length 100 into a `(10, 10)` grid. To compute the neighborhood for updating the weights, the Manhattan distance between a specific center and all 100 points

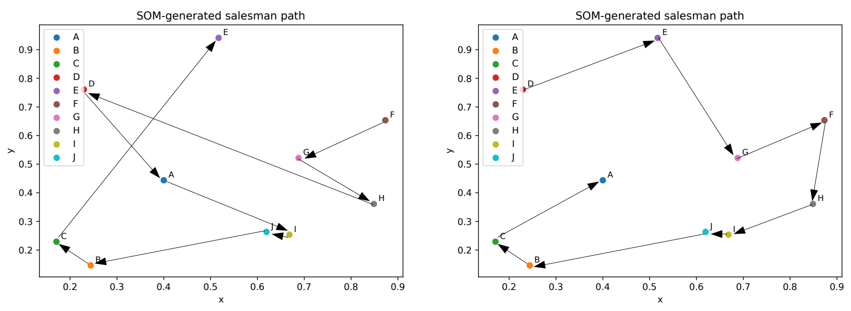

Figure 5: (left) The shortest path when starting out with a neighborhood size of 4 (right) The shortest path when starting out with a neighborhood size of 6

on the grid are computed and sorted by size. See Figures 6 for the data cluster representation of the sex, district and party of the MPs. As it seems, sex does not have a significant influence on the way in which people vote, whereas district and party (obviously) do.

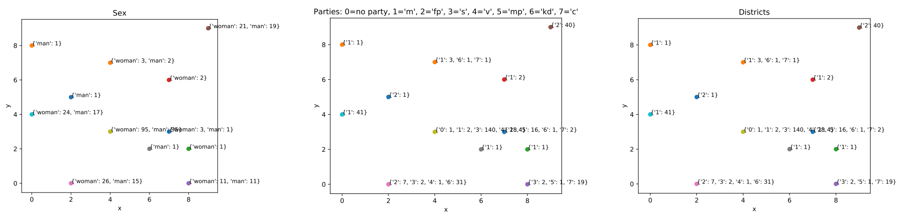

Figure 6: Clustering of MPs: (left) sex; (middle) party; (right) district

5 Final remarks

The assignment is constructed in such a way that it forces one to really think about all the components of RBF-networks and SOMs. It is not possible to just groundlessly implement the mathematical formulas and have an algorithm that works out of the box. What could be improved in future iterations in this lab is that some notational inconsistencies were encountered: in the lecture slides n stands for the dimension of the input space and N stands for the number of RBF patterns – in the assignment this definition is the other way around.