Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Magnus Tronstad, Teo Jansson Minne and Quintus Roos

February 10, 2021

1 Main objectives and scope of the assignment

The main objective with this assignment is to further extend our knowledge of neural networks by implementing and training Radial Basis Functions Networks (RBF) using both batch and online learning, Competitive Learning (CL) and Self Organizing Maps (SOM).

2 Methods

This assignment was written in Python 3 together with the following libraries: NumPy, SciPy, Pandas and Matplotlib.

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

3.1 Function approximation with RBF networks

Regression was performed with RBF units to approximate two functions: $\sin(2x)$ and $\text{square}(2x)$. For the sin(2x) function, using a node width of 1, we needed 6 nodes to reach a mean absolute error (MAE) of 0.001, 8 nodes to reach 0.01, and 11 nodes to reach 0.001. For the square function, we were unable to reach below 0.01. However, when we used a "threshold" to transform any negative RBF function prediction to -1 and any positive RBF function prediction to +1, the we were able to obtain 0 error.

3.1.1 Comparative analysis of batch versus on-line learning schemes

In Figure 1 we see the results from comparing RBF network performance fitting to the $\sin(2x)$ function for varying RBF-widths σ and number of nodes, N, where the nodes have been spread out equidistant from each other on the interval $[0, 2\pi]$.

For large RBF widths, online learning struggled to converge to the least square solution. Another noteworthy trend is that for very small widths (e.g $\sigma = 0.1$), many nodes are required to provide a good fit, which makes sense since highly localized RBF-units "react" only to a very limited subset of the input space, which means that the final linear combination of these RBF-units generalizes poorly to inputs in-between these narrow RBF "peaks".

When the best network (8 nodes, variance 1) is used with an equidistant distribution of nodes, the test error is 0.0093. When the positions of the nodes are instead randomly drawn from a uniform probability distribution on $[0, 2\pi]$, the corresponding error is 0.0502 $\pm$0.003 (averaged over 100 initializations).

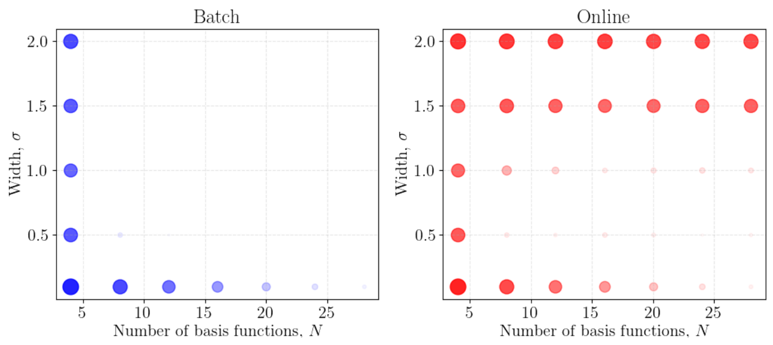


Figure 1: *Plot illustrating the test set performance for different learning schemes. Dots are proportional in size and intensity to the mean absolute error (MAE). For the online learning scheme, the error was recorded at 3000 epochs*

3.1.2 Performance on noisy and clean data sets

Comparing the performance of the RBF network when dealing with clean functions and the same functions with added gaussian noise we found that, as expected, the mean error increased when the network was trained upon data with noise, the differences were quite small for a low number of epochs but once the network converged there was a stark difference. This is shown in table 1 where 40 simulations were run for each problem.

	Mean error	Mean error	Mean error
Function	(20 epochs)	(200 epochs)	(2000 epochs)
Clean sin(2x)	0.90 $\pm$ 0.19	0.30 $\pm$ 0.06	0.061 $\pm$ 0.011
Noisy sin(2x)	0.90 $\pm$ 0.13	0.39 $\pm$ 0.07	0.269 $\pm$ 0.017
Clean square(2x)	1.00 $\pm$ 0.18	0.41 $\pm$ 0.06	0.184 $\pm$ 0.012
Noisy square(2x)	1.10 $\pm$ 0.21	0.49 $\pm$ 0.06	0.343 $\pm$ 0.024

Table 1: *Clean vs noisy data comparison of RBF regression performance 8 nodes used for sine and 20 for square. (MAE $\pm$ SEM)*

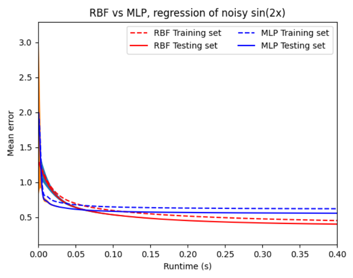
3.1.3 Comparison with performance of MLP and example plots

A comparison between the RBF and the MLP network with one hidden layer was also made, they were both used for approximation of the noisy functions with 8 and 20 nodes/hidden neurons for the sine and square function respectively. When comparing results as a function of epochs the RBF network is significantly faster than the MLP but each epoch also takes longer computational time, in the comparisons found in figure 2.a and 2.b the learning curves are therefore plotted against computational time rather than epochs as it is of more practical interest. Note that this of course has a dependency on the particular implementation of the algorithms. An interesting trade off can be seen here, as the MLP converges faster but not to as good a final approximation as the RBF network which clearly outperforms the single hidden layer MLP.

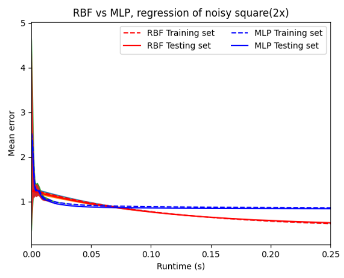
Finally the approximations of the noisy functions by the RBF network can be seen in figure 2.c and 2.d respectively. Worth noting is that the noisy square function approximation becomes sinusoidal when only a few, uniformly spaced, nodes are used.

Figure 2

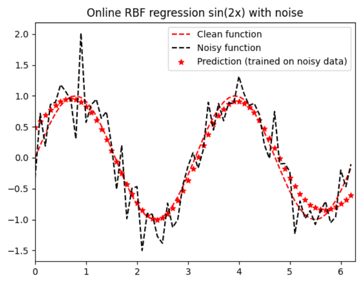
(a) *Mean error as function of runtime, noisy sin(2x)*



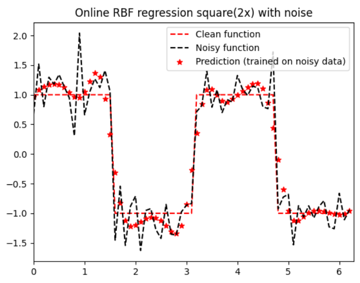
(b) *Mean error as function of runtime, noisy square(2x)*



(c) *RBF function approximation, sin(2x)*



(d) *RBF function approximation square(2x)*



3.2 Competitive learning for RBF unit initialisation

The standard CL algorithm was employed as a means to position the node centres. The results are shown in Figure 3. Note that when we manually placed the node centres in the earlier part of the lab, we used equidistantly spaced points on the interval $[0, 2\pi]$. Since this is what the CL algorithm ends up with, we did not observe any great improvements in performance. However it did outperform the random initialization, indicating its usefulness when it is not intuitively obvious how to manual place nodes.

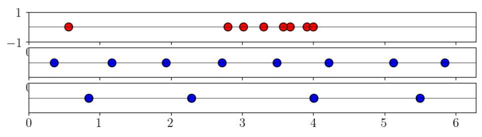


Figure 3: *Effects on node placement using random initialization (top plot), CL with 1 winner (middle) CL with 2 winners (bottom).*

3.2.1 2D Ballistic Regression Problem

Figure 4 visualizes the RBF network's performance when applied to the 2D ballistic regression problem, using an online learning scheme.

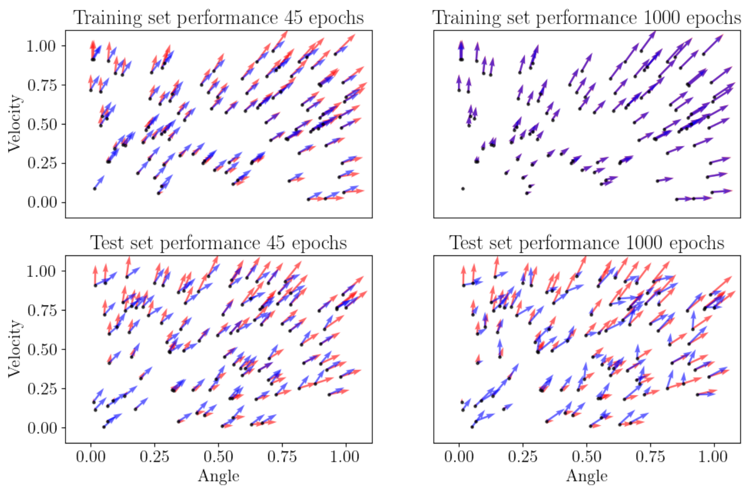


Figure 4: *Visualization of network performance on the 2D regression problem. The target variables, distance and height, are plotted as vectors in the input space. Red vectors are the targets, and the blue vectors are the RBF function approximation.*

16 RBF units with width $\sigma = 1.5$ was chosen through trial and error by comparing convergence plots and corresponding vector plots. The visualization helps illustrate the phenomenon of overfitting: the mean square test error has a minimum at 45 epochs. It then starts to increase again and at 1000 epochs we can actually see the performance has worsened. Conversely, looking at the upper row, we see that the network provides a near perfect fit to the training data.

To avoid "dead units", two ideas were combined: first, the weights were initialized at a random choice of input vectors. Second, during the CL algorithm, when a given RBF unit is sufficiently close to an input vector (set by a tolerance, adjusted through trial and error), that weight is taken out of the update scheme, so that it only considers the remaining RBF units. The algorithm runs until all positions have been "removed" in this way. Overfitting was avoided by monitoring test-set curves to find the minimum test set error during training.

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

4.1 Topological ordering of animal species

The output of the implemented SOM produced reasonable groupings, i.e. the animals are group into insects, flying animals, winged animals, water animals, small mammals, medium mammals, then 'long' animals. The grouping stayed the same after multiple runs, indicating a correct implementation of the SOM algorithm.

The SOM was implemented according to the lab instructions. The initial weight matrix contains random weights which after iterating through all 32 animals with 20 epochs were adjusted to fit the input data. After mapping all data, the tables below shows the resulting grouping sorted after their weight index. The animals with the lowest weight index start on the table to the left and gradually finishes on the table to the right.

	Animal	Index
15	grasshopper	0
4	beetle	0
5	butterfly	0
21	moskito	0
17	housefly	0
10	dragonfly	0
30	spider	0
24	penguin	1
23	pelican	1
22	ostrich	1
11	duck	1

	Animal	Index
28	seaturtle	2
8	crocodile	3
13	frog	5
31	walrus	8
3	bear	10
18	hyena	12
9	dog	15
20	lion	19
7	cat	23
1	ape	28
12	elephant	34

	Animal	Index
2	bat	39
27	rat	45
29	skunk	51
19	kangaroo	57
26	rabbit	63
0	antelop	70
14	giraffe	77
6	camel	84
25	pig	91
16	horse	99

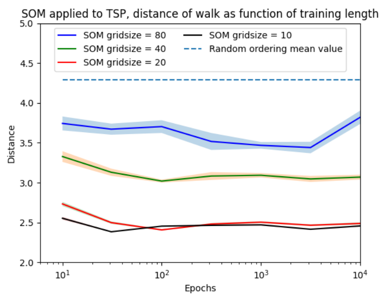
4.2 Cyclic tour

The SOM algorithm was used to optimize the travelling salesman problem, it is suitable for this due to it's topology preservation. 10 cities with normalized x and y coordinates were given and the goal was to find the shortest route passing through each city once, this is not a trivial problem as 10! = 3,628,800 possible routes exist.

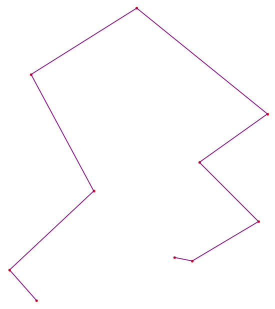
For sake of comparison 100 000 random permutations of the cities were created and the distance of traversing them in that fashion was calculated, the mean value of these is then an approximation of the expected distance of a randomly guessed route. As can be seen in figure 5.a the SOM clearly performs better than this. A decaying neighbourhood as well as global learning rate was used and the decay rates were kept constant as the total grid size was increased, as is also evident in figure 5.a this leads to worse performance, showcasing the importance of finding a good balance between grid and neighbourhood size for a given type of input data. In figure 5.b an example route is shown, found using the best performing grid size 10 x 10.

Figure 5:

(a) *Cyclic tour problem, distance of tour as function of training time and grid size*



(b) *Example route, red dots = cities.*



4.3 Clustering with SOM

The SOM algorithm was applied to voting patterns of members of the Swedish parliament. Weights were initiated in the span of the first two principal components of the data and a learning rate decay scheme $\eta_{k+1} = \alpha\eta_k^{(k/k_{\max})}$ was used (where k is the epoch). The results from running the SOM algorithm for 20 epochs with $\eta_0 = 0.5$, $\alpha = 0.5$, is shown in Figure 6.

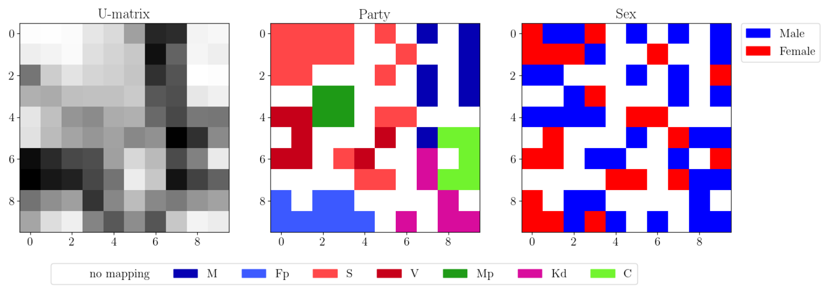


Figure 6: *Output mapping of members of parliament based on voting patterns. The leftmost plot shows the U-matrix of average distances to the neighbours; the middle plot shows the output with respect to party affiliation; the rightmost plot shows the output with respect to sex.*

Firstly, there are no obvious signs of clustering with respect to the `sex` of the parliament members—as expected. However, the appearance of the U-matrix (measuring average distance to each node's 8 surrounding neighbours) (left plot) does indicate clustering of the nodes' weight vectors.

Indeed, if we instead look at *party affiliation* (middle plot) we see the emergence of party-clusters. Looking solely at the middle-plot, we might be tempted to conclude that some left-block parties are in fact located near some right-block parties in the input space! However, the dark areas (corresponding to *large* distance to neighbours) in the U-matrix seem to demarcate between the left and right blocks of the Swedish party system, indicative of the traditional uni-dimensional (left-right) political continuum.

Like `sex`, the dimension `district` also lacked clear signs clustering. Our interpretation, then, is that the `party` dimension is responsible for most of the (obvious) clustering in the input data.

5 Final remarks

The majority of the assignments were interesting and gave valuable insights. For the SOM in particular it felt that the different implementations made some things 'click' which would've been hard just from the lectures. In some cases the results of the assignments are quite obvious and it is a bit unclear what is expected in the report. Coupled with the page limitation this leads to us omitting/shortening down some of the findings we find most interesting.