Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Fernando Garcia, Lucas Q. Gongora and Maria Bjelikj

February, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to to build and analyse properly an RBF network
- to apply the network to different initialization conditions
- to learn the concept of vector quantisation and its implementation
- to implement and understand the SOM algorithm and implications for multiple datasets.

2 Methods

Python version 3 was used for this assignment. The Numpy toolbox was used, as well as Matplotlib for plotting. For metrics, Sklearn's mean square error and mean absolute error were used.

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

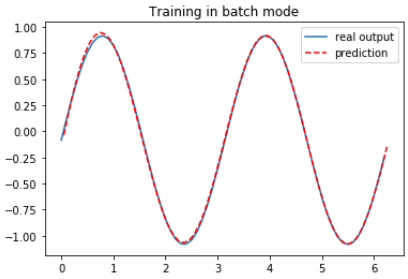

Figure 1: Prediction of sin(2x) data.

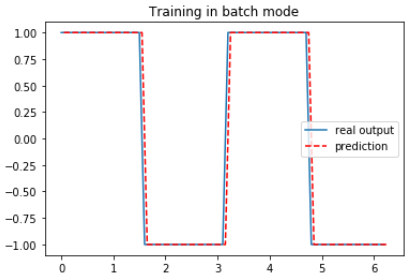

Figure 2: Prediction of square(2x).

Function	Error Threshold	Absolute Residual Error	Hidden Nodes
sin(2x)	0.1	0.06815	6
sin(2x)	0.01	0.0093	8
sin(2x)	0.001	0.000911	11

Table 1: Absolute residual error thresholds and number of used hidden nodes.

3.1. All the results were obtained with width $\sigma = 1$, training in batch mode. Table 1 shows the number of hidden nodes needed to cross each of the three absolute residual error thresholds for the $\sin(2x)$ function. For the $\text{square}(2x)$ however, using linear activation, an absolute residual error lower than 0.1 could not be obtained. To reduce the error to 0.0, sign activation can be used instead, i.e. thresholding the output: a value of 1 is assigned to all outputs ≥ 0, and a value of -1 is assigned for all the negative outputs. An error of 0.0 was first achieved for 9 hidden nodes. This kind of output from the RBF is useful for transforming the problem into classification. Another application is transforming analog values into digital, which is useful in e.g. signal processing.

3.2. For this part, noisy data was used. The online learning using delta rule was done in 10 000 epochs and learning rate $\eta = 0.1$. The results in Table 2 are obtained with variance $\sigma = 1$. Changing the variance i.e. the width, for $\sigma = 0.5$ the absolute residual error goes down to 0.1489 for batch mode with 24 hidden nodes and down to 0.2459 for the online delta rule, which wasn't the case for $\sigma = 1$, where the error started to go up when 19 hidden nodes or more were used. To observe the effect of different width on the absolute residual error, from Figure 3 it can be seen that $\sigma = 0.5$ and $\sigma = 0.8$ yield the lowest error when learning in batch mode, and from Figure 4, $\sigma = 0.5$ is best for online learning with delta rule. For most different widths, the error is almost identical, except for $\sigma = 1.5$ and $\sigma = 1.2$; the larger learning rates are not suited for the delta rule learning.

Looking at Table 2, it can easily be noted that the error is consistently smaller for the online training using delta rule for the $\sin(2x)$ function. For the $\text{square}(2x)$ function, there isn't a big difference between the two methods, which makes sense as the prediction doesn't quite capture the square shape without the application of thresholding for the outputs. When training on noisy data but applying the network to clean data, the MAE decreases, especially significantly for the sin(2x) data when training in batch mode.

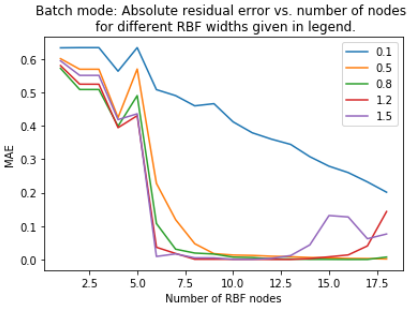

Figure 3: The absolute residual error for a different number of RBF nodes with different widths, RBF network trained in batch mode.

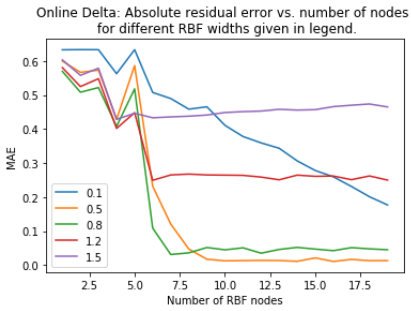

Figure 4: The absolute residual error for a different number of RBF nodes with different widths, RBF network trained online with delta rule.

Function	Hidden Nodes	Noisy data		Clean data	
		Batch mode	Online Delta Rule	Batch mode	Online Delta Rule
sin(2x)	3	0.5175	0.5223	0.5683	0.5717
sin(2x)	8	0.129	0.0662	0.04943	0.05293
sin(2x)	19	0.195	0.06409	0.02307	0.03495
square(2x)	3	0.6812	0.6679	0.7872	0.7892
square(2x)	8	0.2953	0.3017	0.3107	0.3168
square(2x)	19	0.261	0.30027	0.21309	0.2885

Table 2: Absolute residual error and number of hidden nodes for $\sigma = 1$.

Table 2 shows the absolute residual error for different numbers of used hidden nodes with width $\sigma = 1$ for the RBF network trained on noisy data. The same network is applied to the clean data for comparison of performance.

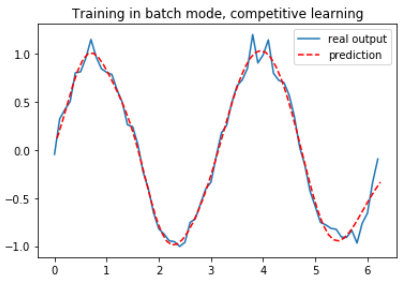

Figure 5: Prediction using competitive learning with 19 RBF nodes, learning rate $\eta = 0.1$ and width $\sigma = 1$.

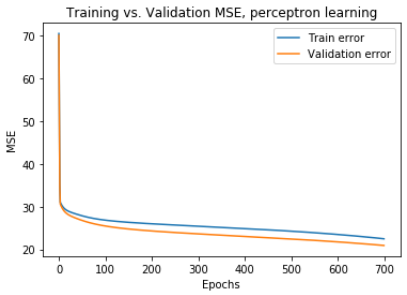

Figure 6: Training and validation error obtained with training the sin(2x) with noise using Perceptron Learning.

Figure 6 plots the learning curve for training and validation of the sin(2x) data, using perceptron learning for the first 700 epochs. However, for the results in

Table 3 were obtained using 10 000 epochs. Looking at the results in Table 3, it can be concluded that the RBF network using competitive learning produces a smaller MSE for prediction.

Function	Absolute Residual Error RBF	Absolute Residual Error Perceptron	Hidden Nodes
sin(2x)	0.07455	0.9238	8
sin(2x)	0.00947	0.721	12
sin(2x)	0.01386	0.698	20
square(2x)	0.3448	11.203	8
square(2x)	0.3053	12.654	12
square(2x)	0.3781	15.340	20

Table 3: Mean square error for the sin(2x) and square(2x) data obtained by the RBF network using competitive learning and perceptron learning, with the same number of hidden units for both algorithms for comparison.

3.3. Figure 5 plots the prediction of the noisy sin(2x) data using competitive learning in batch learning. To compare with Table 2 and the lowest MAE of 0.192 for 19 RBF nodes, for the same configuration in the network except for the completitive learning selection of the RBF nodes, an error of 0.0976 was achieved. To avoid dead units, "Leaky" competitive learning was introduced, where a large learning rate is assigned to the winning node and much smaller learning rate (in this case, η^2 was used) was assigned to the losing node − the node with the largest distance from the training vector.

The RBF network was adapted for two dimensional data. Thus, the RBF nodes are now positioned in the $xy - plane$. The prediction is displayed in Figure 7, which achieved an absolute residual error of 0.01802. The 12 RBF nodes for competitive learning were randomly selected and this resulted in the accuracy in prediction somewhat depending on the selection of the nodes. To illustrate this, let's observe Figure 8, where for all parameters unchanged except for the random selection of RBF nodes for competitive learning, the prediction is not as good as that in Figure 7, achieving a MAE of 0.0869.

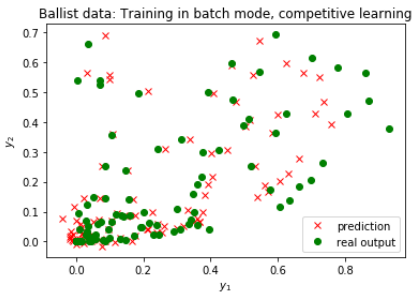

Figure 7: Prediction of balltest data, RBF nodes positioned with competitive learning and fixed widths at $\sigma = 1$.

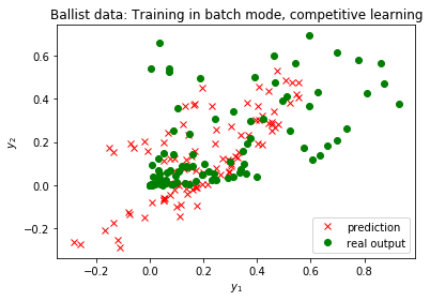

Figure 8: Prediction of balltest data, bad prediction due to bad RBF node selection.

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

4.1. Setting the initial neighbors to 50, with 20 epochs and $\eta = 0.2$, the sorting of the animals makes sense in Table 4, once it is possible to visualize similar animals close to each other and moving to different types of animals as the weight indexes increase. It is possible to divide the data into 5 classes of animals, the first one being the arthropods; followed by the birds; then amphibious (only the frog) and reptiles; lastly, the mammals.

The order depends on the seed and the initialized weights, but the structure is not affected. What may happen is that the mammals will be closer to the Index 0 and the insects at the bottom.

Animal	beetle	dragonfly	grasshopper	butterfly	moskito	housefly	spider
Index	0	0	2	4	8	11	18
Animal	pelican	duck	penguin	ostrich	frog	seaturtle	crocodile
Index	25	26	30	34	38	42	44
Animal	walrus	bear	hyena	dog	lion	cat	ape
Index	50	53	56	59	63	63	67
Animal	skunk	bat	elephant	rat	rabbit	kangaroo	antelop
Index	71	73	76	79	82	85	89
Animal	horse	pig	giraffe	camel			
Index	92	95	98	99			

Table 4: Animals sorted in a 100 size nodes.

4.2. The plot 9 represents the shortest cycling tour route passing through all the cities based on their coordinates.

It is possible to get different paths according to the initial randomized values.

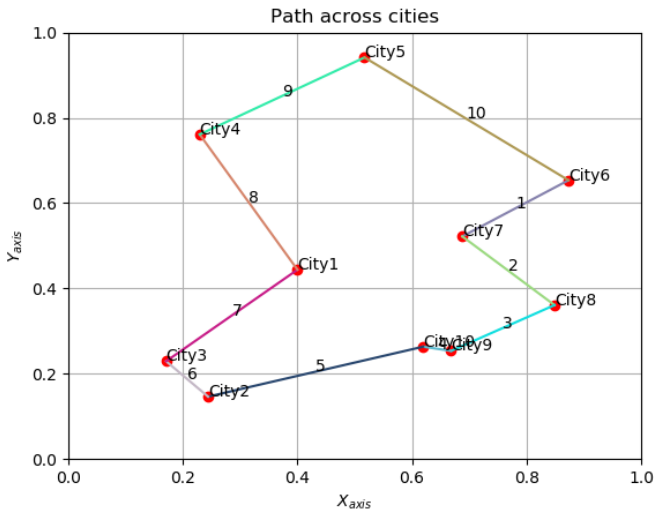

Figure 9: Cycling tour best path

4.3. The plots in Figure 13 represent the votes distributed in a 10x10 output grid.

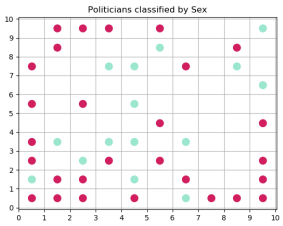

Figure 10: By Gender

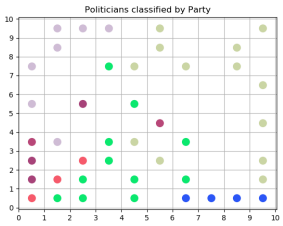

Figure 11: By party

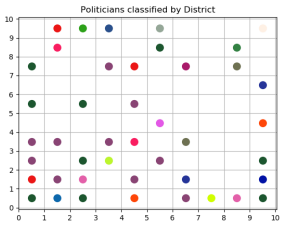

Figure 12: By district

Figure 13: Voting results according to different attributes

By looking at the plots in Figure 10 and Figure 12, it is not possible to visualize any kind of pattern. Most of the points, colored by their attribute, are over each other and spread around the grid. However, for the grid divided into parties, a pattern can be seen; most of the colours are close to each other and not much spread out as in the examples before.

The colour in the grid is the group which had the most repetitions in that specific position. Another solution would be to add Gaussian noise with $\mu = 0$ and $\sigma = 0.3$, to avoid the points in the same location overlapping each other.

5 Final remarks

This lab was a good way to learn how to implement RBF networks as well as the SOM algorithm. It helped us understand how to use these networds for regression, ordering and clustering. It was also interesting to approah problems in more dimension and how to interpret those results.