Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Kevin Dalla Torre, Franco Ruggeri and Fredrik Danielsson

February 12, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were:

- to learn to implement and use RBF neural networks for regression problems.
- to learn to implement and use self-organizing maps (SOMs) for dimensionality reduction and clustering.
- to understand the impact of the parameters and how to select them in both the aforementioned neural networks.

2 Methods

We used *python* for all the tasks with the support of just the *numpy* and *matplotlib* libraries, implementing all the algorithms from the scratch.

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

3.1 Function approximation with RBF networks

Table 3 shows the number of RBF units necessary to get a residual error under a certain threshold. The network was trained with the batch mode and the RBF width fixed using the formula $sigma = \frac{d}{2\ \#rbf}$, where d is the maximum distance between the RBF centres [1]. We can see that the network manages to fit the sinusoide, while does not for the square function. This is because, as a linear combination of RBF kernels, it cannot get the straight lines required. The table shows also that by applying the sign function to the output of the RBF NN, the square function can be fitted perfectly. This is useful in classification problems, which can be thought as regression problems of a square function.

Threshold	sin(2x)	square(2x)	square(2x)*
0.1	6	-	6
0.01	6	-	9
0.001	10	-	9
0	-	-	9

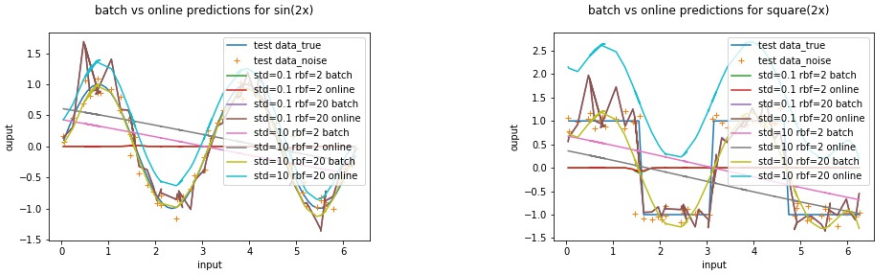
Table 1: * indicates that the output of the RBF NN is transformed using the sign function, - indicates that it is not possible.

In figure 1 there is a comparison between batch and online learning for two significant values of number of RBF units and two of width. For $\eta = 0.1$[1], they both behave similarly and we can draw the following conclusions about the mentioned parameters:

- A large number of RBF units with small (but not too much) width can model well the functions, since each unit will be responsible for a small portion of input space. If the width is too small, the approximate function results in peaks.
- A small number of RBF units with high width result in quite steady functions (in the selected extreme cases we get straight lines).
- A small number of RBF units with small width results in isolated peaks, because there are portions of the input space not covered by any unit and each RBF is very little spread.

The RBF centres were initialized uniformly and the generalization is better than a random initialization according to the results in table 2. Conceptually it is better because with a uniform initialization the RBF nodes occupy as much input space as possible.

[1]The delta rule depends on it, the higher the faster the convergence but a too high value can jump the minimum and continue to oscillate forever



(a) (b)

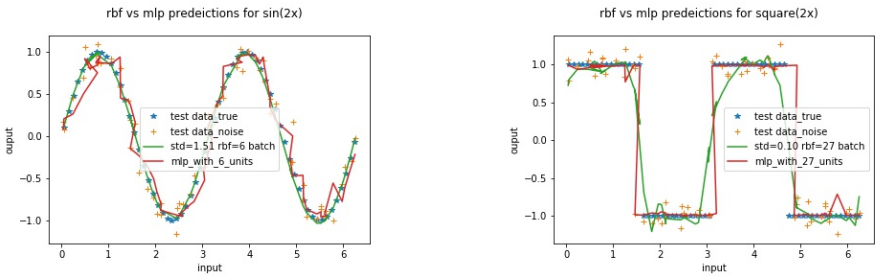
Figure 1: Comparison of batch and online learning for 2 different number of nodes and 2 different widths.

# RBF	Random	Uniform
6	0.50	0.48
20	0.32	0.14
43	0.17	0.09

Table 2: Residual error for random and uniform initializations of RBF, online learning, averaged on 100 runs.

All the models trained with noisy patterns had a lower test error if noise was not added to the test set. This means that we manage to fit the underlying function. For example, for RBF width $\sigma = 0.1$ and 27 RBF units, the residual error was 0.13 on the noisy test set and 0.09 for the original clear one.

When comparing MLP with the RBF NN we can see that the former is better for the square function while the latter for the sinusoide, as shown in figure 2. The RBF network using batch mode is faster in training time. In fact it has a closed-form solution, while the learning algorithm for the MLP is iterative.



(a) (b)

Figure 2: Comparison of MLP and RBF NN for the same number of nodes that the optimal RBF model uses.

3.2 Competitive learning for RBF unit initialisation

When initializing the units randomly and then using competitive learning to re-position them the test performance was worse than in the uniform case. If one looks at the final positioning of the nodes when using the CL algorithm one can see that they look uniformly placed. This is because the data is uniformly distributed in the input space. Thatâs why the uniform initialization would be better than this algorithm in this specific case at least. One way to avoid dead units is to have more than one winner. However having more than one winner in this case decreases the test performance ever further. Even more so I one can evenly divide the number of units by the number of winners. If one has for example 9 nodes and 3 winners, then it will look as you only have 3 in the input space so it better to have 2 or 4 winners in this case.

winners	units	CL	error*
1	6	Yes	0.28
1	6	NO	0.24
1	10	Yes	0.031
1	10	NO	0.028

Table 3: * indicates that the output of the RBF NN is transformed using the sign function, - indicates that it is not possible.

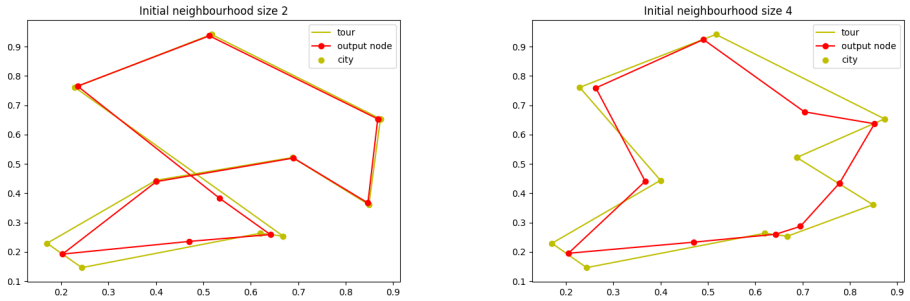
4 Results and discussion - Part II: Self-organising maps

4.1 Topological ordering of animal species

Position	Animal	Position	Animal
0	dragonfly	16	bear
1	grasshopper	17	hyena
2	beetle	18	dog
3	butterfly	19	cat
4	mosquito	20	lion
5	housefly	21	skunk
6	spider	22	rat
7	frog	23	bat
8	sea turtle	24	elephant
9	crocodile	25	kangaroo
10	ostrich	26	rabbit
11	penguin	27	antelope
12	duck	28	horse
13	pelican	29	pig
14	walrus	30	camel
15	ape	31	giraffe

Table 4: Animals sorted in the natural order found by the SOM.

From table 4 we can see that the SOM finds a very reasonable natural order for the 32 animals. In fact, we can notice that animals belonging to the same class are grouped together (e.g. insects are in positions 0-5) and similar animals are close (e.g. cat and dog). This confirms that the SOM, using the suggested parameters ($\eta = 0.2$, 100 nodes, 20 epochs, initial neighbourhood size 50), preserves the topology of the input space and finds a significant 1D representation of the 84D input space.



(a) (b)

Figure 3: Tour and output nodes connected according to the 1D grid used by the SOM, for initial neighbourhood size 2 (a) and 5 (b).

4.2 Cyclic tour

Figure 3 and table 5 show an analysis of the tours found by SOMs with varying initial neighbourhood size. It is easy to see that for values smaller than 5, the SOM misses the global ordering and the resulting tour is not optimal. This confirms that starting from a large neighbourhood size is necessary. We can also notice in figure 3b that few output nodes remain in the middle between two patterns. This happens because their neighbours cause updates in different directions and when the neighbourhood size decreases to 0 they are not updated anymore, since they are not winners of anyone.

Initial neighbourhood size	Tour length
2	3.16
3	2.92
4	2.82
5	2.69
6	2.69
7	2.69
8	2.69

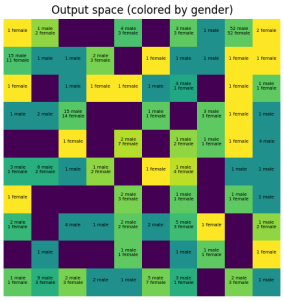
Table 5: Animals sorted in the natural order found by the SOM.

4.3 Clustering with SOM

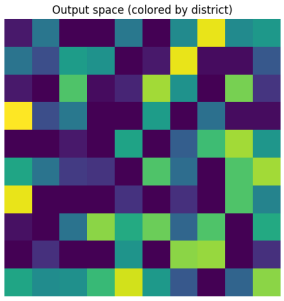
Figure 4 shows that the MPs belonging to the same party vote similarly. It is also possible to observe in the x-axis (horizontal) the traditional left-right scale of the parties (e.g. *MP*, *S*, *V* are left parties, while *M* and *KD* right). A curious discovery is that also the other (vertical) dimension is significant, because it clearly separates MPs according to their parties.

From figure 5 we can notice also small clusters of MPs with the same gender or district, proving that there is a relationship between the votes and these features.

Figure 4: Grid of output nodes colored according to the party of the MPs.



(a)



(b)

Figure 5: Grid of output nodes colored according to the gender (a) and district (b) of the MPs.

5 Final remarks

Concerning SOMs, we found a bit confusing and not so relevant to study both gender and districts in problem 4.3, as they represent conceptually the same thing and the districts are quite hard to visualize (a lot of different districts end up on the same nodes).

References

[1] Stephen Marsland. *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2015.