Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Filip Husnjak, Ivan Klabucar and Corentin Royer

September 26, 2021

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to implement RBF neural network for function approximation,
- to study the results when using different number of RBF nodes and different node initialization techniques,
- to implement SOM algorithm for topological ordering of animal species, solving the traveling salesman problem and visualising the members of the parliament, their political orientation and their gender.

2 Methods

We used python for all our experiments. For the first part we added the numpy library for matrices computation, and we used the pinv function to compute the pseudoinverse of the $\Phi^T\Phi$ matrix during the least squares batch learning method. For the second part, we coded the SOM algorithm from scratch. We used either VS Code or PyCharm as IDE. We used github for versioning and a simpler sharing of the tasks.

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)*

Error threshold	0.1	0.01	0.001
$sin(2x), \sigma = 1.5$	6	8	10
$square(2x), \sigma = 0.112$	28	-	-

Figure 1: Nodes needed in RBF network to achieve absolute residual error on validation set beneath a certain threshold. Cells marked with '-' indicate a certain threshold could not be met.

To begin our exploration of RBF networks we first tested some architectures by hand to see how they performed. Our goal was to satisfy certain error thresholds. The error measure we used was the average absolute residual error on the validation set. In Table 1 we can see the results of this initial testing. The RBF networks did quite a good job approximating the sin(2x) function and reached the desired thresholds with a relatively small increase in the number of nodes. On the other hand, the RBF networks had much more trouble approximating the square(2x) function, and it turned out we couldn't even find a single architecture that scored below 0.01. To explain this discrepancy we can examine Figure 2 to see how increasing the number of nodes affects the predicted square function.

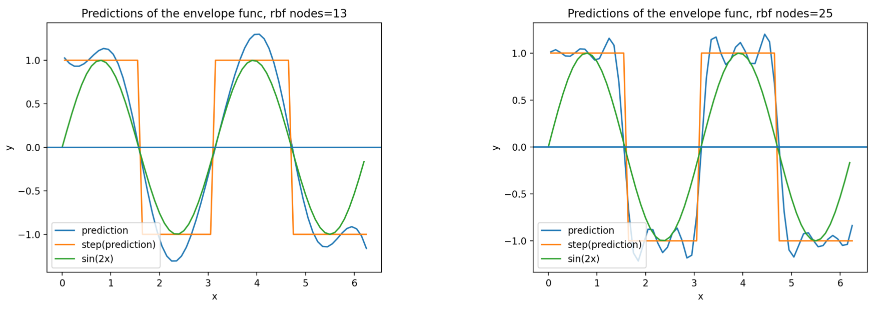
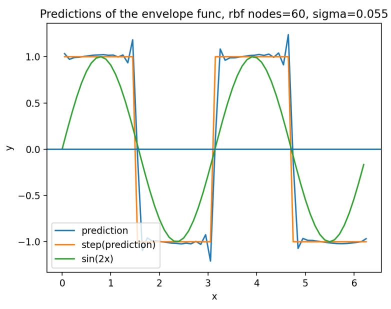
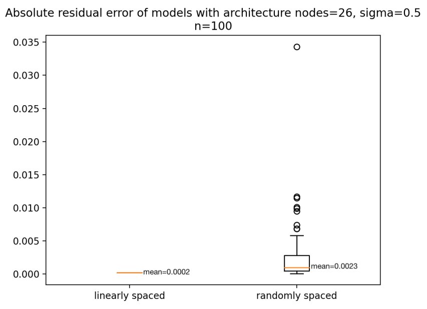


Figure 2: Effects of using more nodes for approximating the square(2x) function.

It turns out the RBF network approximates the straight line by oscillating around it. The more nodes we add the denser the oscillations become. However, this doesn't mean we can get a great approximation by just increasing the number of nodes because as we can see on Figure 3a this approach works well for the middle of the plateaus, but on the edges where the function changes signs dense nodes still cause oscillation. This can be remedied, however, by transforming the output of the RBF network with a step function. This approach reduces the error to zero on the validation set, because after the step transform it only matters that the sign of our prediction was correct. This transformation is particularly useful for classification problems.



(a) Using a step function to reduce the absolute residual error to 0.0 for square function predictions.



(b) Error comparison for sin approximating RBFs with different node placements

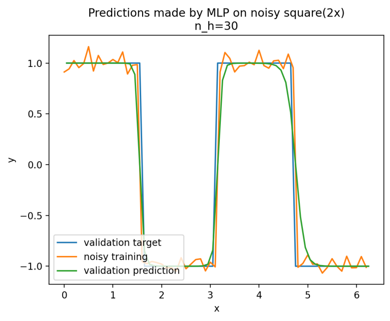
Figure 3

In these experiments we placed the RBF nodes evenly throughout the input space. To see whether this approach really gives us an edge over just assigning the placement of nodes randomly we trained 100 models with different random placement of nodes on the sin(2x) function. The results of this experiment are seen on Figure 3b, and confirms our assumption that placing the nodes uniformly yields better results. Specifically, around 10 times better results in this case.

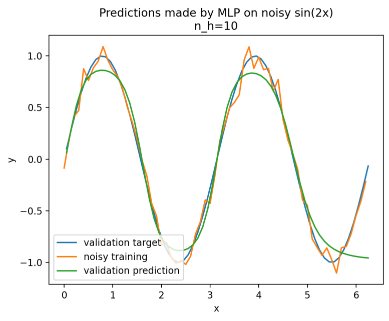
		Non-noisy	Noisy
$sin(2x)$	batch	$err = 4.59 \times 10^{-5}$ $n = 60,\ \sigma = 1.0$	$err = 0.0153$ $n = 10,\ \sigma = 1.0$
	online	$err = 0.0148$ $epochs = 101$ $n = 20,\ \sigma = 0.5$	$err = 0.0229$ $epochs = 101$ $n = 20,\ \sigma = 0.25$
$square(2x)$	batch	$err = 0.105$ $n = 30,\ \sigma = 0.1$	$err = 0.111$ $n = 30,\ \sigma = 0.1$
	online	$err = 0.0993$ $epochs = 101$ $n = 60,\ \sigma = 0.1$	$err = 0.1103$ $epochs = 85$ $n = 60,\ \sigma = 1.0$

Figure 4: Best results achieved during grid search for both noisy and non noisy cases, using two different learning approaches.

To more systematically determine the effects of these parameters on performance, as well as to examine the differences between using least squares batch learning and online delta rule learning, we performed a grid search. We tested out all combinations of number of $nodes \in \{10, 15, 20, 30, 60\}$ and RBF $widths \in \{0.1, 0.25, 0.5, 1, 1.5\}$ in noisy and non noisy conditions for both functions. Results of best architectures for every case can be seen in Figure 4. It is interesting to note that online learning generally performs worse that batch learning, and also takes much longer to converge, usually making it to the set epoch limit of 101 epochs.



(a) Using an MLP to approximate the square(2x) function.



(b) Using an MLP to approximate the sin(2x) function.

Figure 5

We compared our best batch-trained RBF architectures with similar one hidden layer MLP models in noisy conditions.[1] The absolute residual error for sin(2x) was 0.109, and 0.083 for the square(2x). From these numbers and plots of these predictions as seen on Figure 5 we can conclude that while MLPs outperformed RBFs on the square function while the opposite is true for the sine function.

3.2 Competitive learning for RBF unit initialisation

Using competitive learning for RBF node initialization we got interesting results. We used CL version where each iteration closest RBF node is moved towards the point using the formula $w = w + \eta \times (x - w)$. Running multiple iterations we concluded that algorithm converges towards uniform distributions of the RBF nodes because the input data is uniformly distributed as well. Without using any strategy to avoid dead units the performance was not great since a lot of the nodes ended up not being used. To avoid dead units we introduced a method which selects already existing points randomly for initial position of RBF nodes. This method proved to be very successful and that can be seen on the figure below.

By using this method we can also compare this with the previous results when we placed the RBF nodes by hand and that can be seen on the figures 7a and 7b.

[1]Number of nodes is the same.

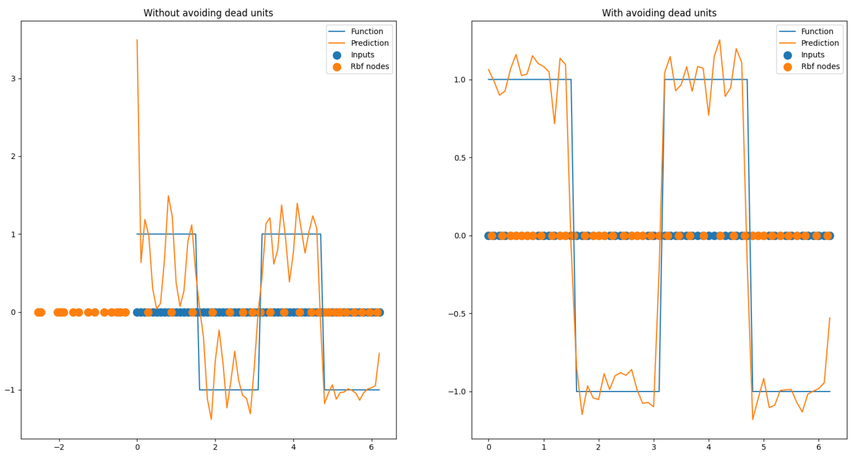
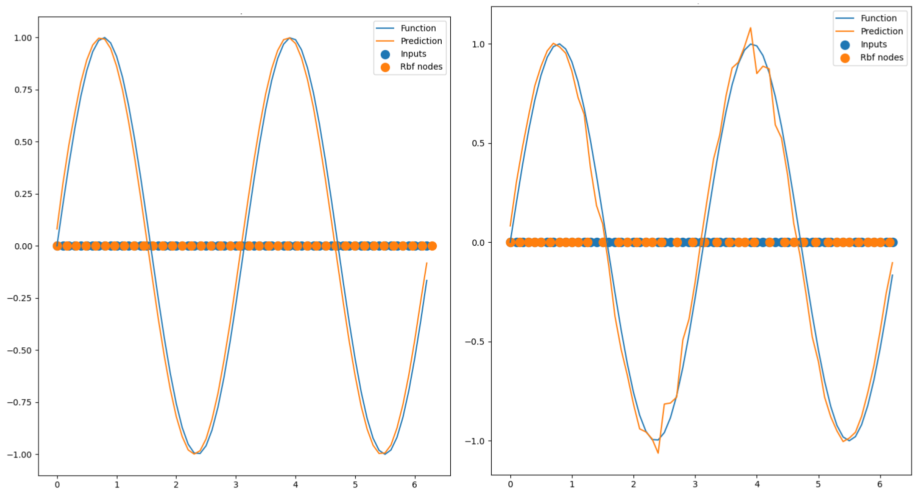


Figure 6: Importance of dead unit avoidance.



(a) Prediction when placing nodes by hand. (b) Prediction when using CL algorithm for node initialization.

Figure 7: Comparison between two methods of RBF node initialization.

From the figures one can observe that placing the nodes by hand in this scenario gives better results. It's because we have uniformly distributed data so the perfect node positioning is uniform distribution over an input space. Although CL algorithm converges towards that solution it will never reach the state of perfect uniform distribution of nodes, so where the nodes are sparse prediction tends to oscillate rapidly.
After that we used CL algorithm for node placement in 2D input space for predicting noisy data from ballistical experiments. Output data was also two dimensional and represented distance and height reached. We performed multiple tests with different number of nodes using test set and concluded that

optimal number of nodes which won't overfit is 30. The predictions using the test set can be seen on the figures below.

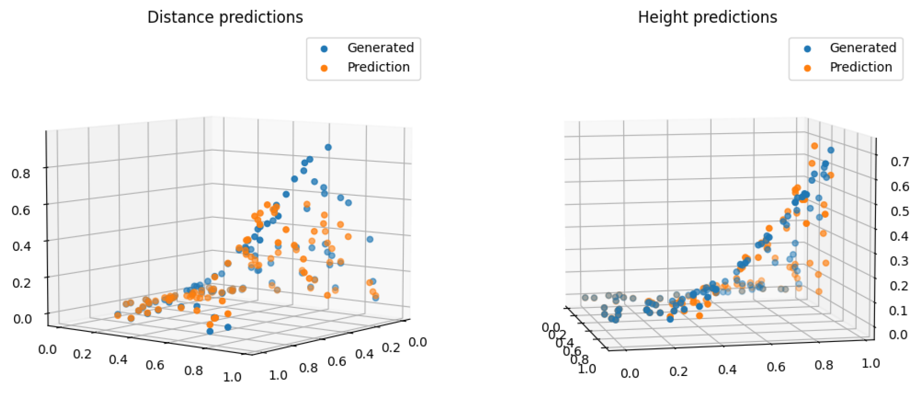


Figure 8: Linear representation of the animal species.

RBF node placement in the input space is shown on the figure 9. For node placement same CL algorithm from previous section was used. It can be seen that the most of the input space is covered by RBF nodes and nodes are concentrated near the bigger clusters of input points, which is expected.

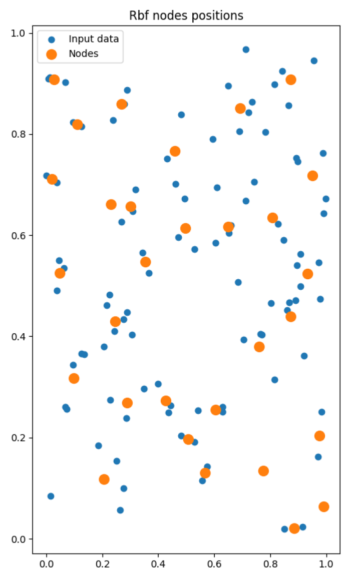


Figure 9: Rbf node initialization using CL.

4 Results and discussion - Part II: Self-organising maps (SOM)

We implemented the SOM as a class where the input and output shapes are parameters so that we could use it for the three datasets. The weights we want to train are of size (input_dim * output_nodes) because each point on the output has a position in the input space.
We confirmed that the SOM implementation worked by testing on a cross shaped cloud of points in 2 dimensions with a grid of 10 by 10 nodes (2 * 100 weights). The nodes correctly moved to characterise as best as possible the distribution.

4.1 Topological ordering of animal species

The first dataset that we analized with the SOM was a set of physical characteristics of 32 animals. The physical features are binary in a 84 dimensions space. The map we wanted to fit to these data was a line of 100 nodes so that we would have a linear ordering of animal species.

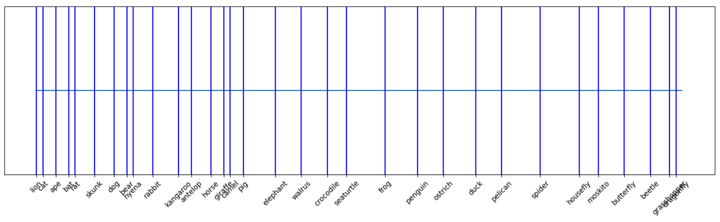
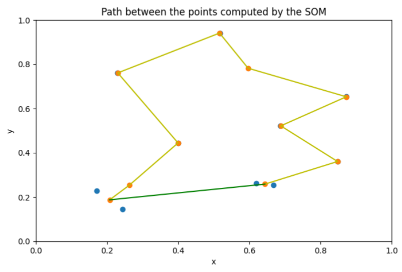


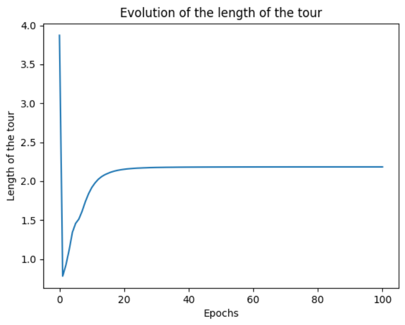
Figure 10: Linear representation of the animal species.

On fig 10 we can see the results we got. We show here the animal's names on the closest node in the 84 dimensional space (for each species, we compute the closest node). The resulting ordering is close to what a human would have made: the insects are close together on the left (grasshopper, beetle...) and a bit on the right we have birds (pelican, duck...). With the visual features of the dataset, this is a coherent ordering.

4.2 Cyclic tour

The second dataset was a set of 10 points on a 2 dimensional space. These points represented cities' position, the goal was to use the SOM algorithm to find a path that goes through all the cities. As we used the SOM, we know that it will try to minimize the length of the path between all the points (see fig 11b), thus we have an approximation of a solution to the traveler salesman problem in a polynomial complexity.





(a) Representation of the path between the cities points.

(b) Evolution of the length of the cyclic tour

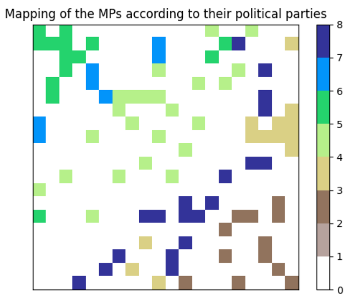
Figure 11: Training of the SOM on the cities data.

The tour we got from the training of the SOM generally went through most cities, however it was often the case that it missed one or two cities that are close together and just put a point in the middle of those two cities (see fig 11a). Indeed there is no hard constraint to put the points directly on the cities so we can end up in situations like that. On fig 11b we can see that the length of the tour first shrinks rapidly and then stabilized at an higher value. This is because at the beginning on the training, the radius to update neighbors of a winning node is large, thus all the points are heavily moved toward the first updated node.

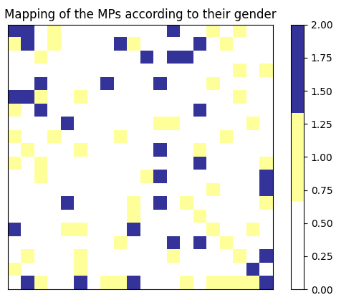
4.3 Clustering with SOM

The last dataset on which we trained the SOM was a result of 31 votes of 349 MPs of the Swedish parliament. We wanted to represent these MPs on a 2 dimensional grid (from the 31 dimensional input space). We can expect to find the traditional left-right dichotomy of the parliament in the representation we got. To verify that, we plotted the MPs in a color that correspond to their party (see fig 12b).

On figure 12a we can see the left right spectrum on the second diagonal. We can see that the brown and dark green are the most "extreme" parties (there are the 'm' and the 'v' parties which are respectively a liberal conservative and a socialist feminist party). The party in blue on the figure corresponds to the Green Party and the dark blue party corresponds to the christian democrats.
We then tried to see if the gender of the MP has an influence on the way he/she votes. On the figure 12b we cannot see any trend in the data. It is safe to assume that the gender is not linked to the political values of a person. The Swedish parliament is known for its gender equality, we could not find information about each parties percentage of woman but it is safe to assume that every party has almost an equal representation of both genders.



(a)



(b)

Figure 12: Visualization of the MPs according to gender and party. **12a)** Distribution of the MPs colored by their party. **12b)** Distribution of the MPs colored by gender.

5 Final remarks

In the first part of the lab it was interesting to see how different the widths of the rbf nodes had to be depending on the shape of the function. When the rbf nodes were wide and overlapped a lot the batch learning algorithm was able to find good solutions by assigning large values to the weights. The online approach couldn't deliver the same solutions to cases where wide rbf nodes were used because of the small incremental nature of the approach. It was good practice to see the difference in training time needed for different learning schemes of rbf networks. The second part of the lab shows how it is possible to represent high dimensional data in a 1 or 2 dimensional space that can be simply interpreted. This is a valuable result for a lot of applications where we have a lot of features on each sample. The SOM can also solve more exotic problems such as the traveler salesman, this problem in particular is important in path finding so finding an efficient solution to it is valuable.