Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Majd Dawli, Anna Melnichenko and Yohan Pellerin

February 8, 2024

1 Main objectives and scope of the assignment

Our major goals in the assignment were to

- learn how to use RBF networks with different initialization methods for function approximation.
- learn how to implement different SOM algorithms and analyze the effect of neighborhood.
- Enhance our understanding of non-linearly separable data and how to separate them using the methods mentioned above.

2 Methods

The primary programming environment used in this lab was Python, along with the NumPy and Matplotlib libraries. NumPy provides helpful mathematical functions along with support for large, multi-dimensional arrays and matrices, which helped represent the weight and input matrices and perform the dot product. Matplotlib was used to visualize our approximation function against the real function and also to plot error curves. Furthermore, we used the math library to get access to the sin(x) function.

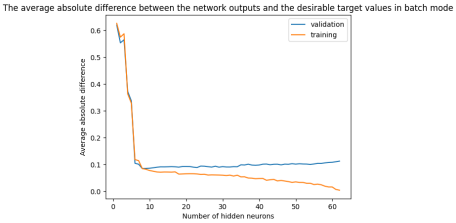
Additionally, Jupyter Notebook was used to make the workflow easier by allowing only certain functions to run separately, which helped us run the different tasks in this assignment easily.

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

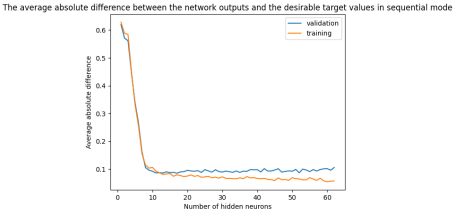
3.1 Function approximation with RBF networks

3.1.1 RBF approaches importance units (number and width)

By training our RBF network on function approximation of $\sin(2x)$ in batch mode with different number of hidden neurons, we plotted the average absolute difference for the training and validation data sets shown in 1. We observe that the for both the training and the validation sets, the error rapidly decreases until the number of hidden nodes reaches 7 and the validation error start to increase, while the error for the training set continues to decrease but in a slower fashion.



Figur 1: batch mode (with sin(2x)

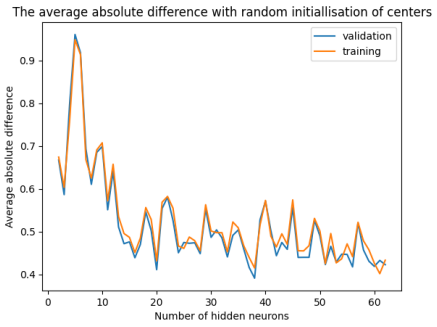


Figur 2: sequential mode (with sin(2x))

Figure 2 is the corresponding figure for sequential mode training. Similarly we observe a rapid decrease in the error as the number of hidden nodes grows until it reaches a value of 7 where the decrease is not as noticeable as before. When reaching 12 nodes, the training error continues to have some decrease in contrary from the validation error that starts to raise. Also compared to the batch mode training, we observe that the difference between training and validation error is smaller so the batch mode approach leads to more overfitting. Indeed, we can see the training goes very close too zero for a high number of nodes.

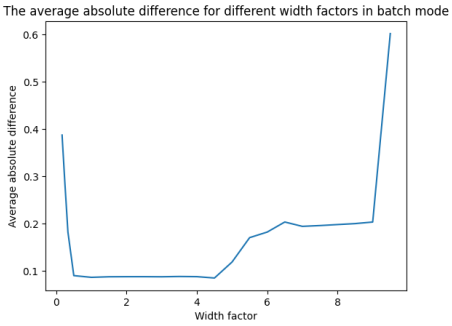
The way of choosing the units for sin(2x) is similar for square(2x).

To determine the initialization of the various units, we opted to position them at regular intervals. This approach yielded positive outcomes, as demonstrated by the preceding graph. However, upon examining the average absolute difference during random initialization of centers (as depicted in Fig. 3), we notice oscillations in the error rate. Nevertheless, despite these fluctuations, the error tends to decrease with an increase in the number of hidden neurons. This observation is logical, as it echoes the patterns observed in the previous graph. Additionally, it aligns with the notion that a higher number of units implies a more dispersed distribution, thus contributing to the reduction in error.

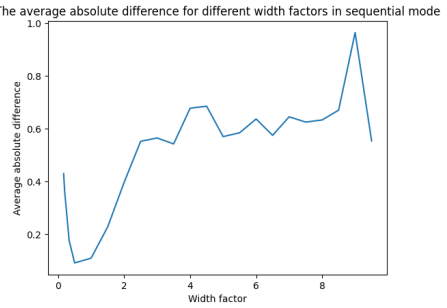


Figur 3: Caption

Next, we turn our attention to the uniform width assigned to each unit, set to be equal to the distance between two units. To validate this assumption, we conducted comparisons with different width (different multiplication of this initial intuition) for both batch mode (refer to Fig. 4) and sequential mode (refer to Fig. 5). Notably, the batch mode yields optimal results with width factors ranging between approximately 0.5 and 4.9; beyond this range, the error rate begins to increase. Conversely, for the sequential mode, the optimal width factor is approximately 1. Hence, we can conclude, firstly, that the initial intuition appears to be correct, given that the best results are indeed observed around this initial width. Secondly, it's worth noting that the batch algorithm appears to be less sensitive to changes in width compared to the sequential mode.



Figur 4: batch mode (with sin(2x)

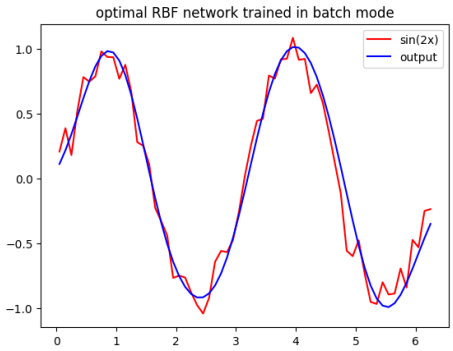


Figur 5: sequential mode (with sin(2x))

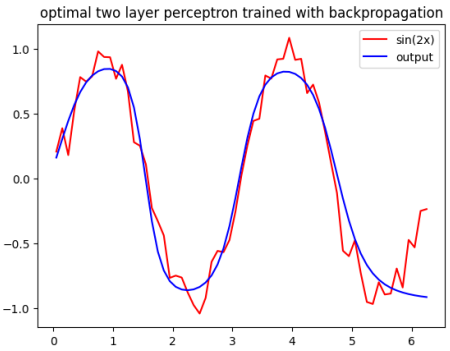
Too conclude, the best network topology for the batch mode is given by 10 hidden neurons and with a width factor between 0.5 and 4.9. For the sequential mode topology, we get 11 hidden neurons and a width factor of 1 for getting the minimal error.

3.1.2 RBF vs MLP

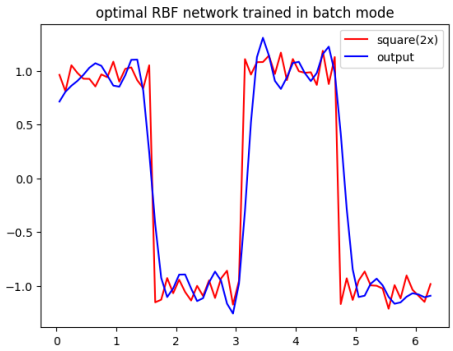
Now, we can compare the MLP method used in the previous lab with the RBF method using optimal parameters. As depicted in the figure above, for the sinus function, the two graphs exhibit remarkable similarity. However, the average absolute difference is approximately 0.15 for the perceptron and slightly lower at 0.09 for the RBF method. In terms of computational time, the RBF algorithm also outperforms, requiring less than 1 second compared to around 3 seconds for the perceptron. While this difference may seem insignificant, it can significantly impact larger datasets. Conversely, for the square function, both the RBF algorithm and the perceptron yield similar results, with an error rate of around 0.18.



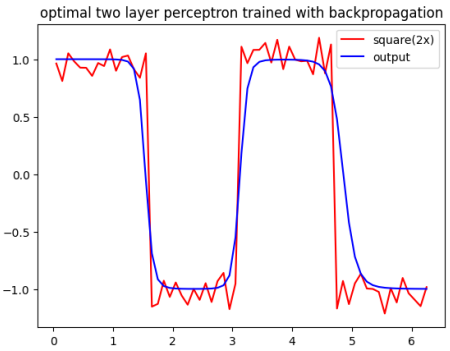
Figur 6: RBF (with sin(2x))



Figur 7: square (with sin(2x))



Figur 8: RBF (with square(2x))



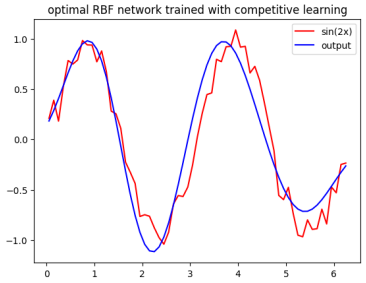
Figur 9: perceptron (with square(2x))

3.2 Competitive learning for RBF unit initialisation

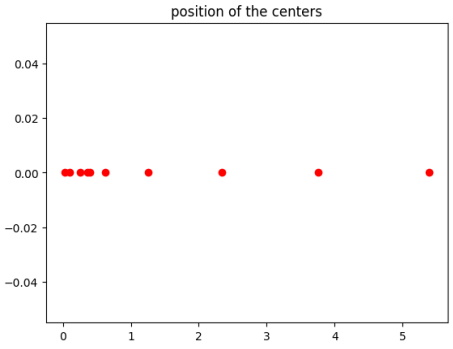
Let's proceed by comparing the CL-based approach with the previous method of manually positioning RBF nodes in the input space. It's clear that we're

achieving relatively satisfactory results, with an average absolute error of approximately 0.12. However, this performance slightly lags behind the previous outcome. Additionally, upon analyzing the distribution of the various centers, it's evident that it lacks uniformity. This uneven distribution likely contributes to the suboptimal results we're experiencing.

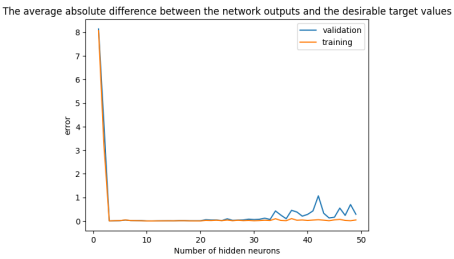
Next, we employ CL to configure an RBF network for approximating a two-dimensional function. We determine the optimal number of nodes, which is approximately 11, yielding an error of 0.005. Upon examining the distribution of the centers, there are no isolated points observed.



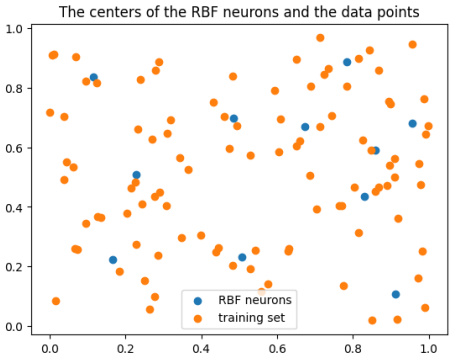
Figur 10: optimal result with competition learning



Figur 11: position of centers



Figur 12: The nodes number



Figur 13: position of centers

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

4.1 Topological ordering of animal species

For the topological ordering of animal species, we used the SOM algorithm to train a network consisting of 84 input nodes and 100 output nodes. The output nodes were arranged in a non-circular order. We started of with a step size of 0.2 and a neighbourhood size of 50 nodes. Througout the 20 epoch used for training of our network, we decreased the neighborhood side to 1 by using neighbours $= 50 \cdot 0.78^{\text{epoch number}}$. As a result, we obtained a one dimensional ordering of the animal species as shown in 1. From that we observe that the algorithm managed to group similar species together.

Tabell 1: Ordering of animal species using SOM

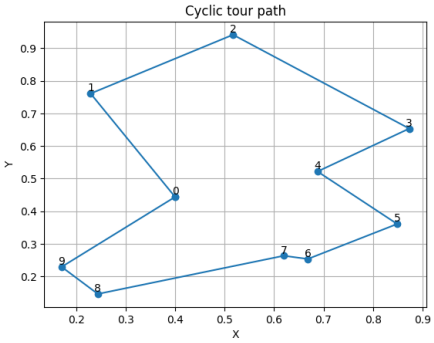
Node #	Species	Node #	Species	Node #	Species	Node #	Species
0	Pig	21	Bat	43	Bear	75	Pelican
2	Camel	24	Rat	46	Walrus	80	Spider
3	Giraffe	27	Ape	51	Frog	84	Housefly
6	Horse	30	Lion	55	Crocodile	88	Moskito
9	Antelop	32	Cat	59	Seaturtle	93	Butterfly
12	Kangaroo	35	Skunk	64	Ostrich	96	Grasshopper
15	Rabbit	38	Dog	68	Penguin	98	Beetle
18	Elephant	41	Hyena	72	Duck	99	Dragonfly

4.2 Cyclic tour

For the cyclic tour problem the adjustments on the previous described algorithm was made by changing the neighbourhood calculation function to being circular, by treating the first an last output nodes as adjacent. The initial size of the neighbourhood was 2 and decreased to 1 after 10 epochs and to 0 after 15 epochs. The resulting routes varied a bit depending on the initial weights, but one of the best results can be seen in 14.

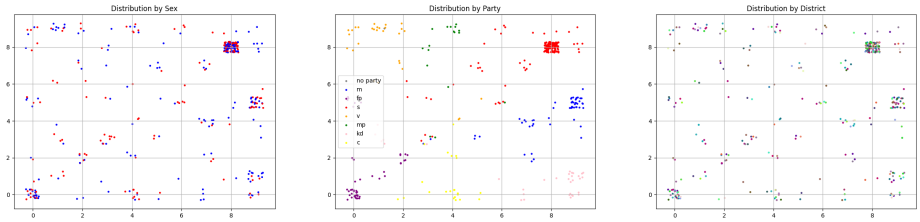
4.3 Clustering with SOM

For the clustering of the votes of MPs, the algorithms neighbour function was changed again. This time manhattan distance was used. Also the output was changed to fit the format of a 10x10 grid, and for a better representation in the visualization, we added small uniform random noise to the resulting coordinates. In 4.3 we can see that the votes of different parties mostly are close to the other votes of the respective party. For the other categories, no specific pattern can be seen as shown in 4.3 and 4.3. An alternative could have been to plot the majority



Figur 14: Cyclic tour path

per the discrete x, y coordinates, that would give us some approximation of the majority classes that corresponds to some specific majority party at the specific coordinate.



Figur 15: Distribution by sex, party and district for voting

5 Final remarks

The lab covered a lot of ground for understanding for the different algorithms and their application as it covered a variety of topics. The theory before each assignment helped with the understanding the topics and was a ground for being able to use the different networks in practice. One question that arises while doing the lab, was how to represent 349 values on a 10x10 grid. Overall we learned a lot from the lab.