Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Emil Hed, Ivar Karlsson and Elias Kollberg

February 6, 2025

1 Main objectives and scope of the assignment

Our primary goals in this assignment were:

- To implement radial basis function (RBF) networks for function approximation and analyze their performance under different conditions.
- To explore competitive learning as a method for optimizing RBF unit initialization and compare it to traditional approaches.
- To develop and train self-organizing maps (SOMs) for tasks such as topological ordering and clustering.
- To apply SOMs to approximate solutions for the Traveling Salesman Problem (TSP) by generating cyclic tours.

The assignment focused on understanding how these neural network models learn from data and adapt to different problem types. We examined RBF networks in both one-dimensional and two-dimensional function approximation, investigating how noise and initialization strategies impact performance. Additionally, we explored how SOMs can identify meaningful structures in high-dimensional data, applying them to biological species ordering and TSP optimization.

2 Methods

For this assignment we have used Python and Jupyter Notebooks as our primary programming tools, with Visual Studio. We used libraries including Numpy for numerical operations, Matplotlib for data visualization.

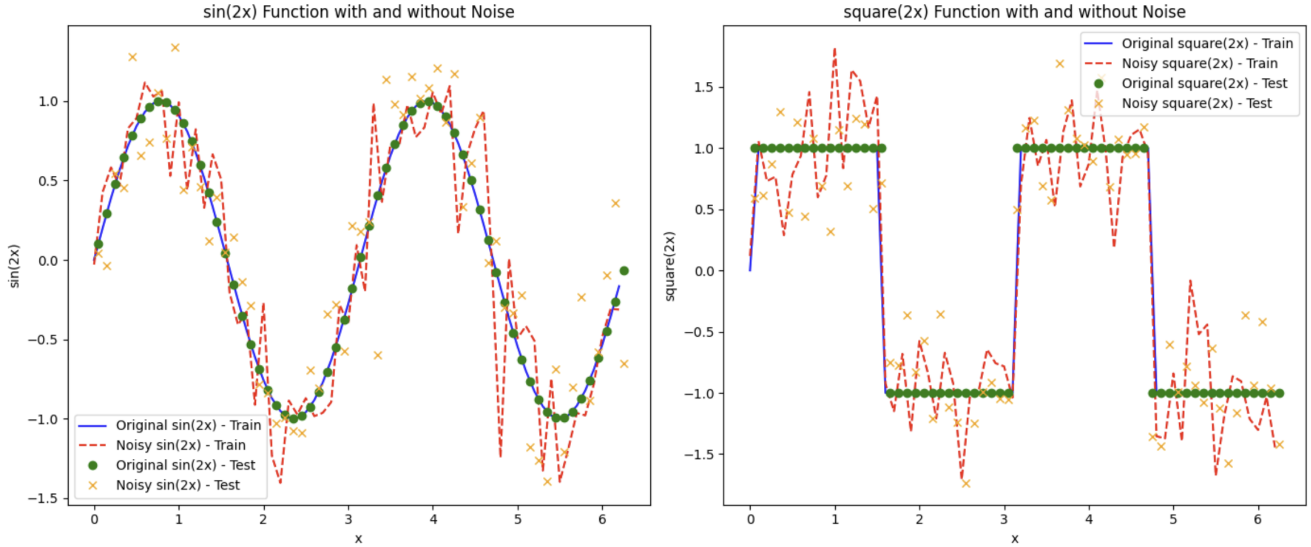
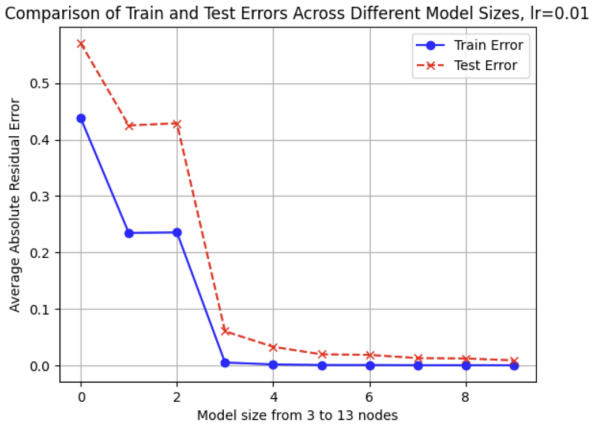
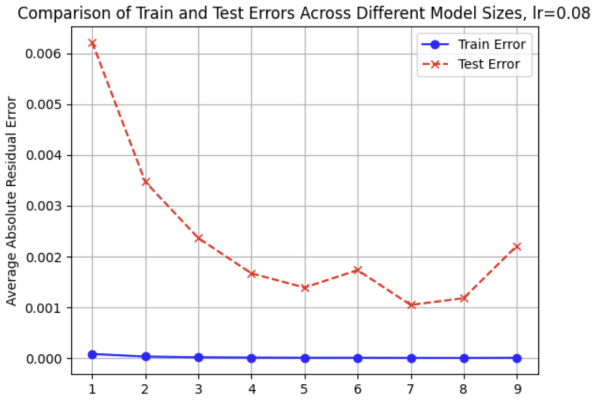


Figure 1: Original generated data.

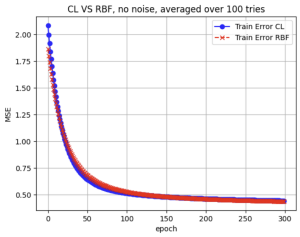
3 Results and discussion - Part I: RBF networks and Competitive Learning normalsize*(ca. 2.5-3 pages)*



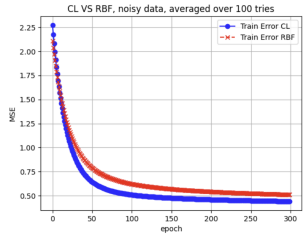
(a) Train/test error across different model sizes, with the amount of nodes in the span of 3,4,5, ... , 13



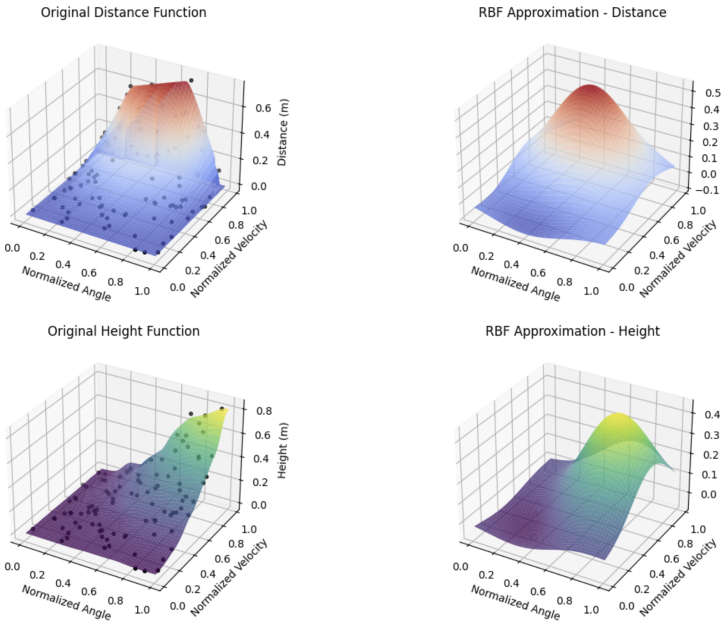
(b) Train/test error across different model sizes, with the amount of nodes in the span of 10,15,20, ... , 60



(c) Convergence of error (in MSE) for the original RBF and the RBF model using CL, no noise on data



(d) Convergence of error (in MSE) for the original RBF and the RBF model using CL, noisy data



(e) Function approximation results, comparison of original data (left side) and approximated function (right side)

Figure 2

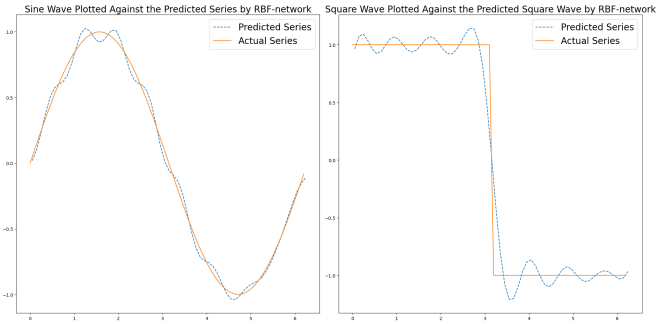
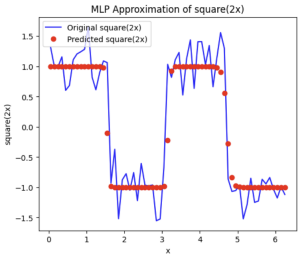
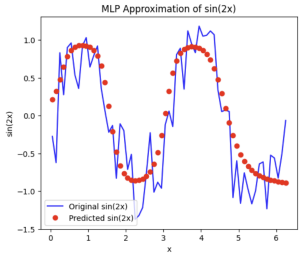


Figure 3: Square and sine wave predicted by rbf networks with 30 nodes



(a) Predictions on the noisy square function by the MLP, taken from lab1



(b) Predictions on the noisy sin function by the MLP, taken from lab1

Variance	Nodes	Residual Error	
		Sine Wave	Square Wave
0.05	10	0.1993	1.0048
0.05	20	0.1556	0.8913
0.05	30	0.1356	0.9049
0.1	10	0.1464	0.8730
0.1	20	0.1540	0.8356
0.1	30	0.1402	0.8533
0.5	10	0.2108	0.7907
0.5	20	0.1345	0.7632
0.5	30	0.2385	0.7354
1.0	10	0.2361	0.9344
1.0	20	0.1906	0.9980
1.0	30	0.2064	0.9629
2.0	10	0.2721	0.6598
2.0	20	0.3225	0.6245
2.0	30	0.3260	0.6049

Table 1: Final residual errors on the test set for different variances and number of nodes.

3.1 Function approximation with RBF networks

In this section, we examine how RBF networks were used to approximate two different functions: $\sin(2x)$ and $\text{square}(2x)$. We evaluated their performance on both noise-free and noisy datasets to understand their adaptability.

To analyze the robustness of RBF networks, we tested the differences of performance between different sized networks, or in other words, different amount of nodes in the hidden layer. Generally speaking, the performance went up when we increased the amount of nodes (as long as we didnt have more amount of nodes than datapoints, which leads to an overdetermined model). However, when we train the model on the noisy data, we can see that overfitting the model becomes a problem with too many nodes. We can see this in Fig2a where the validation-error starts to go up when we use more than 50 nodes in the model. This same figure (along with Fig2b also answers

the question of how many nodes are needed "*to obtain the absolute residual error below 0.1, 0.01 and 0.001 in the residual value (absolute residual error is understood as the average absolute difference between the network outputs and the desirable target values)*". Which according to these figures are around 6 nodes to get the error below 0.1, 9 nodes to get the error below 0.01, and 50 nodes to get the error down to 0.001. This was all done for the sin functions.

Regarding the square function: Interestingly, we found that the RBF network was able to achieve zero error when approximating the square wave function. This is primarily because the square function consists of sharp transitions between discrete values, making it well-suited for RBF units, which can be precisely positioned at these transition points. Unlike the sinusoidal function, which requires a smooth approximation, the square wave benefits from well-placed basis functions that match its step-like nature. One could argue that for this exact span on numbers, only four nodes with the right variance would be needed to get zero validation error, by placing them in the very middle of the four tops and downs.

The number of RBF units and their width play a big role in how well the model performs. More units mean the network can capture more granular details, but too many can lead to overfitting, especially when noise is present. The width of the RBFs is a key factor for a high performing network. If the width is to small, it can lead to gaps in the area that the function is defined in. This causes small responses in all nodes, which leads to reduced capacity to approximate function in these uncovered areas, which leads to global errors in the function approximation. If the width is too large it results in all units firing relatively high for any input, which leads to local inaccuracies.

For the online learning, it is possible to use a large learning rate and still reach convergence

The placement of RBF nodes is another vital factor for unlocking performance. If the nodes are randomly placed, some areas might not get covered properly, which significantly lowers performance. A better approach is to place them strategically, using methods like k-means clustering, or placing them according to intuition. A strategic placement leads to lower errors and better function approximation. Comparing performance on clean versus noisy data shows just how much noise affects learning. Models trained on noisy data tend to struggle with both clean and noisy test sets because they can't easily separate noise from the actual function. On the other hand, models trained on clean data perform much better on noiseless test cases, which highlights why techniques like regularization are so important when working with noisy data.

When comparing RBF networks with MLPs trained via backpropagation, the differences are clear. RBF networks train faster because they rely on direct updates, but they're highly sensitive to where the basis functions are placed. MLPs, while slower to train, are more flexible in how they learn representations. When looking at 'sin(2x)' and 'square(2x)', the MLP does a better job generalizing under noisy conditions, whereas RBF networks tend to overfit and struggle with noise, atleast for the case where we used 30 nodes, (the RBF can generalize better if we use less nodes, of course). RBF networks converge faster, while MLPs require careful tuning. Choosing between them depends on what matters more—speed, accuracy, or robustness to noise.

3.2 Competitive learning for RBF unit initialisation

To enhance the initialization of RBF units, we explored competitive learning, which allows for a more structured placement of centers within the input space. More specifically, RBF units were automatically positioned in regions of higher data density, leading to improved approximation accuracy. The results showed some advantages. Firstly, competitive learning led to a reduction in convergence time compared to random initialization. This can be seen in Fig2d. However, when it comes to the data with no noise, there was no considerable difference between the two different models, Fig2c.

Now regarding the two-dimensional function approximation: The goal was to evaluate how well the RBF network could generalize in higher-dimensional input spaces. Our results showed that the network successfully captured complex structures in the data, thought its performance depended on the number of basis functions used. With too few RBF units, the approximation was quite bad, while an excessive number of units led to overfitting. This highlights the importance of choosing an optimal number of hidden units for effective generalization in higher dimensions. The results can be seen in Fig2e.

A common issue in competitive learning is the presence of "dead" units, meaning neurons that never win the competition and therefore fail to contribute effectively to learning. To mitigate this, we implemented a multi-winner strategy, where instead of selecting just one winner per iteration, the top k closest units were allowed to

update their weights. Our implementation sorted all neurons by their distance to the input and selected the top k winners. The update strength was scaled so that the closest unit received the strongest adjustment, while further winners updated with decreasing influence. This led to more robust function approximation. Compared to the standard single-winner competitive learning approach, our multi-winner method resulted in more stable learning and better generalization, particularly in cases where input data was sparse or unevenly distributed.

4 Results and discussion - Part II: Self-organising maps **textit(ca. 2 pages)**

Self-Organizing Maps (SOMs) are an unsupervised neural network technique used for dimensionality reduction and clustering. They achieve this by mapping a high-dimensional input space onto a lower-dimensional output space, while preserving the topological relationships between input patterns. The key idea is that similar input patterns are mapped to neighboring nodes in the SOM's output space.

SOMs belong to the Competitive Learning (CL) family, but unlike classical CL algorithms, which use a strict "Winner Takes All" approach (only updating the best-matching unit, BMU), SOMs also update the weights of neighboring nodes.

The weight matrix dimensions are determined by the relationship between input and output spaces:

- The weight matrix has shape (number of output nodes $\times$ input dimensions), meaning each output node has an associated input-dimensional weight vector.
- For assignment 4.1 and 4.2, where the output space is 1D, the weight matrix is 2D (i.e., [number of nodes $\times$ 84] for animals in 4.1).
- For assignment 4.3, where the output space is 2D (a 10$\times$10 grid), the weight matrix becomes 3D (i.e., [10 $\times$ 10 $\times$ 84]), since each grid position has an input-dimensional weight vector.

The neighborhood function determines how weights are updated during training and must be chosen based on the goal of the mapping:

- Assignment 4.1 (1D Sorting): The 1D topology ensures that similar animals are placed next to each other in a linear sequence.
- Assignment 4.2 (Cyclic Tour): The 1D topology is cyclic, meaning the first and last nodes are treated as neighbors to form a closed loop.
- Assignment 4.3 (2D Clustering): The 2D topology considers both X and Y neighbors, allowing clustering of input patterns in a spatially meaningful way.

4.1 Topological ordering of animal species

BMU	Animal	BMU	Animal
0	grasshopper	66	dog
0	beetle	66	lion
0	butterfly	67	hyena
0	moskito	67	cat
0	dragonfly	67	ape
7	housefly	76	rat
15	spider	78	skunk
36	penguin	78	bat
36	pelican	78	rabbit
36	ostrich	89	kangaroo
36	duck	90	elephant
43	frog	98	antelop
44	seaturtle	99	giraffe
46	crocodile	99	camel
57	walrus	99	horse
61	bear	99	pig

Figure 5: Sorted Animals by BMU Index (Ascending Order)

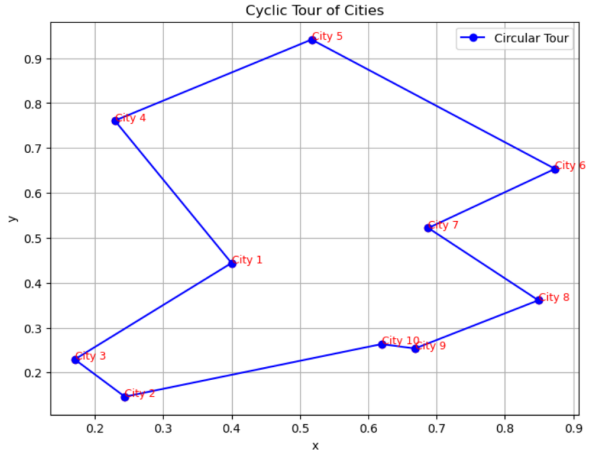
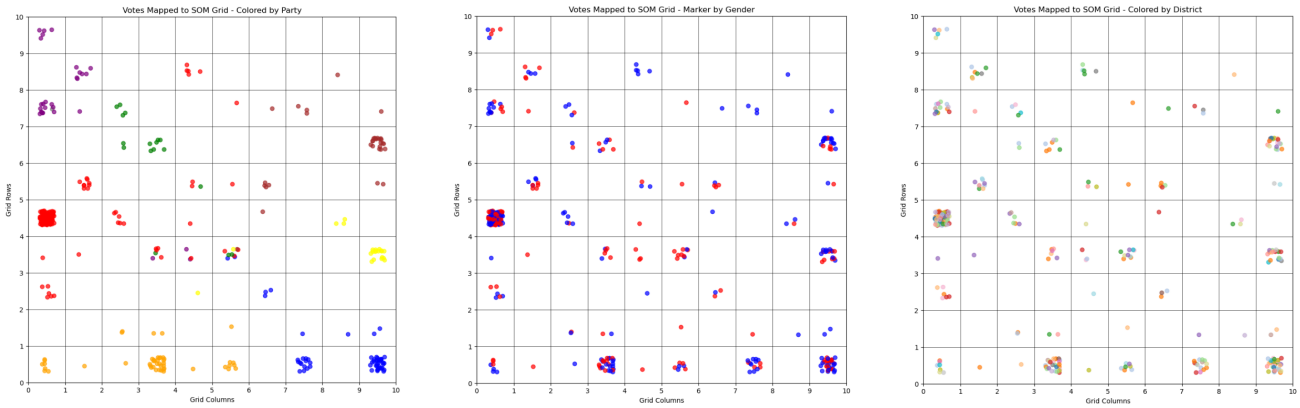


Figure 6: Traveling Salesman Problem (Cyclic Tour of cities).

4.2 Clustering with SOM



(a) Votes by party (b) Votes by gender (c) Votes by district

Figure 7: Visualization of voting patterns based on different attributes.

The results from the clustering shows that the only sorting that gives any valuable insight into the data is when we are sorting by party. This can be expected from this type of data, since the parties are expected to vote similarly, independent of their gender or district.

5 Final remarks *(max 0.5 page)*

This lab offered a great opportunity to explore learning algorithms and model training. Tweaking factors like node count, learning rates, and activation functions showcased how they impact convergence and generalization for RBF networks. It was also very educational for understanding SOMs.