Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2
Radial basis functions, competitive learning and self-organisation

Chiachen Ho, Angelos Stais and Ruxue Zeng

September 29, 2021

1 Main objectives and scope of the assignment

Our major goals in the assignment were

• to build an RBF network for either classification or regression purpose

• to apply different methods for initializing the structure and learning the weights in an RBF network

• to use vector quantization in a neural network context

• to recognize and implement different components in the SOM algorithm

• to discuss the role of the neighborhood and its effect on the self-organization in SOMs

To use SOM network to fold high-dimensional spaces and cluster data

In this lab, we have used an RBF network to approximate one- and two-dimensional functions. And we have developed a competitive learning algorithm to automate the process of RBF unit initialization. Furthermore, We have implemented the core algorithm of SOM and used it for three different tasks.

2 Methods

We have used Python in this lab with libraries below:

- Numpy, Math, Random, Matplotlib, Spicy

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

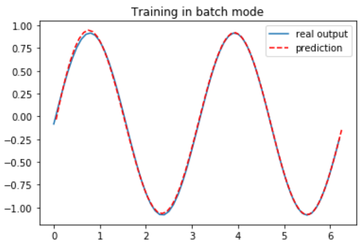


Figure 1: Prediction of sin(2x) data.

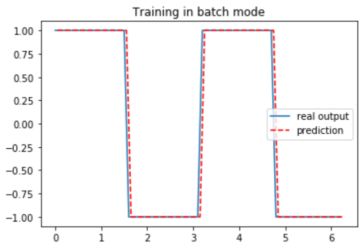


Figure 2: Prediction of square(2x) data

3.1 Batch mode training using least-squares - supervised learning of network weights

Function	Error Threshold	Absolute Residual Error	Hidden Nodes
sin(2x)	0.1	0.06823	6
sin(2x)	0.01	0.0095	8
sin(2x)	0.001	0.000914	11

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

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 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 to all 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 Regression with noise

For this part, noisy data was used. The online learning using the delta rule was done in 10.000 epochs with a 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 widths 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.

		Noisy data		Clean data	
Function	Hidden Nodes	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 for different numbers of hidden nodes with $\sigma = 1$ for the RBF

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. From Table 2, it can easily be noted that the error is consistently smaller for the online training using the delta rule for the *sin(2x)* function. For the *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 ARE decreases, especially significant for the sin(2x) data when training in batch mode.

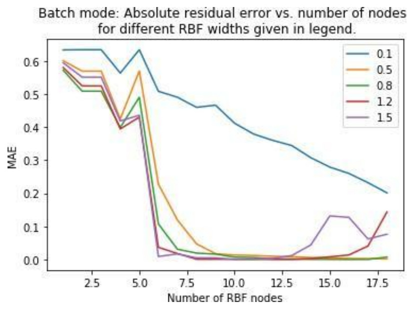


Figure 3: The absolute residual error for a different number of RBF nodes with different widths, RBF network trained in batch mode.

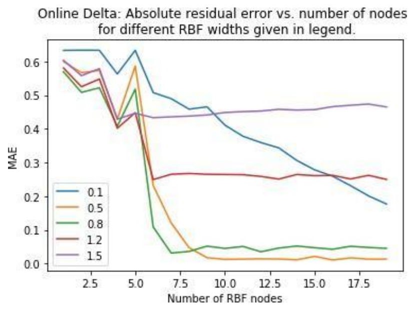


Figure 4: The absolute residual error for a different number of RBF nodes with different widths, RBF network trained online with delta rule.

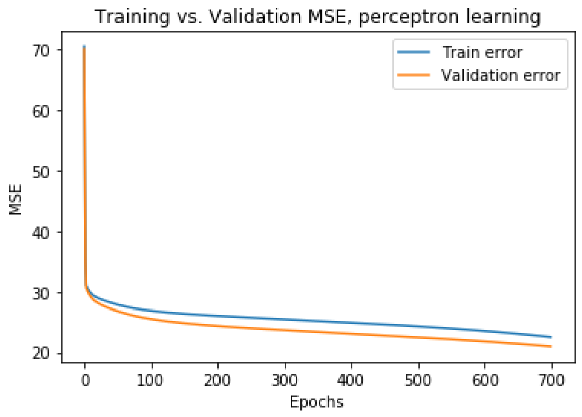


Figure 5 : Training and validation error obtained with training the sin(2x) with noise using Perceptron Learning.

Figure 5 plots the learning curve for training and validation of the sin(2x) data, using perceptron learning for the first 700 epochs. However, 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 produces a smaller ARE for prediction.

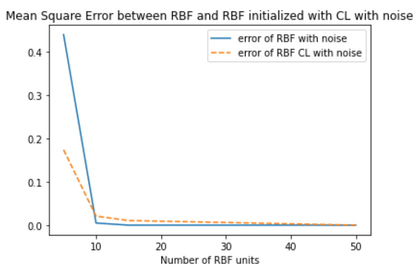
Function	Absolute Residual Error RBF	Absolute Residual Error Perceptron	Hidden Nodes
sin(2x)	0.07455	0.9238	8
sin(2x)	0.00947	0.721	12
sin(2x)	0.01386	0.698	20
square(2x)	0.3448	11.203	8
square(2x)	0.3053	12.654	12
square(2x)	0.3781	15.340	20

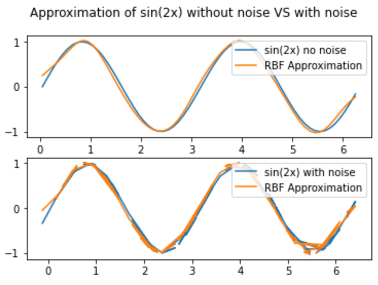
Table 3: Mean square error for the sin(2x) and square(2x) data obtained by the RBF network using competitive learning and perceptron learning, with the same number of hidden units for both algorithms for comparison.

3.3 Competitive learning for RBF unit initialization (η =0.01)

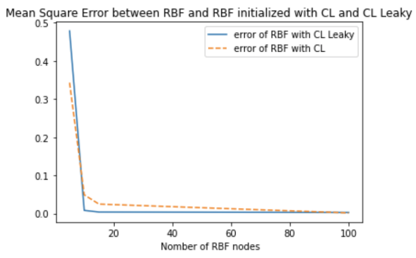
3.3.1 One-dimensional function approximation

In this part, we have used the delta rule to implement the RBF network and a version of competitive learning for Vector Quantization. We have compared the CL-based approach with our earlier RBF network where we manually positioned RBF nodes in the input space. We have made this comparison for both noise-free and noisy approximation of *sin(2x)*.





According to the figure on the right-hand side, with a number of RBF units small, the model initialized with CL algorithm has a lower mean square error, i.e. it has a better performance than the RBF network with manually positioned nodes.

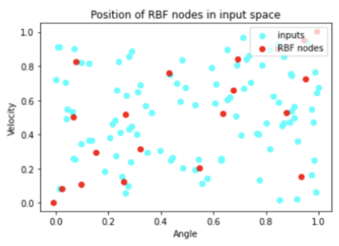


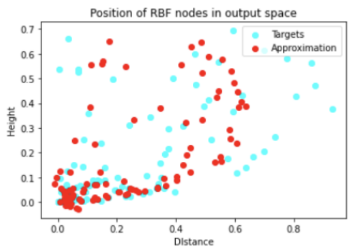
But from numbers of RBF nodes superior to 10, both models have similar performance, for noise-free and with noise. We conclude that The RBF network with automated initialization has a better performance.

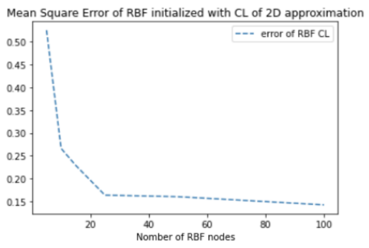
Furthermore, the CL using leaky learning in order to avoid dead units has a better performance than our simple CL algorithm when the number of units gets bigger.

3.3.2 Two-dimensional function approximation

An RBF network with the use of CL could also be used to positioning the RBF units to approximate a two-dimensional function. In this subsection, we have as training examples noisy data from ballistical experiments.







With the first figure above, we observe that the distribution of 20 RBF units in the input space is homogeneously distributed. After training the RBF network, we observe in the 2nd figure that the accuracy of approximation depends on the density of nodes of each area. The approximation of an area with lots of data is satisfactory, but it gets worse when we have little data. The third figure above confirms that this accuracy is affected by the number of RBF units. But we still can't attend an error equals to zero because there still has a dead units problem and overfitting problem when the number of units gets too big.

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

4.1. Topological ordering of animal species

The training epoch is set to 20. The learning rate η is set to 0.2 for the winner node and also for nodes in the neighborhood. The range for neighborhoods decreases as the epoch increases. We set the range = 50 - epoch*2.5, so it goes from 50 to 0.

- **The results of ordering by SOM (one list)**

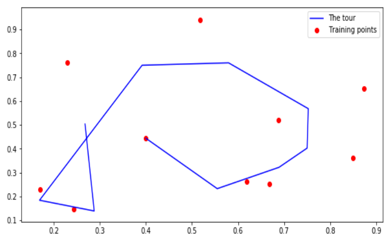
```
beetle
grasshopper
dragonfly
butterfly
housefly
moskito
spider
duck
pelican
penguin
ostrich
frog
seaturtle
crocodile
walrus
bear
hyena
dog
lion
ape
cat
skunk
rat
bat
elephant
rabbit
kangaroo
antelop
horse
pig
giraffe
camel
```

As shown on the left-hand side, insects are grouped together. E.g. from beetle to spider. And animals living in both water and on land are also group together. E.g. from duck to walrus. Finally, four-legged mammals such as dogs, cats, horses, and so on are grouped together.

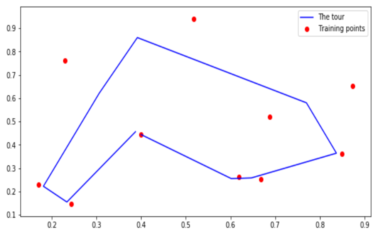
4.2. Cyclic tour

In this task, we also use SOM to solve traveling salesman problem. The difference is the way we define the neighborhood. Here the neighborhood is circular, and the suggested range is from 2 to 1 and finally to 0. We set epoch to 50, hidden nodes to 100, and learning rate to 0.2 for both winner node and neighbor nodes.

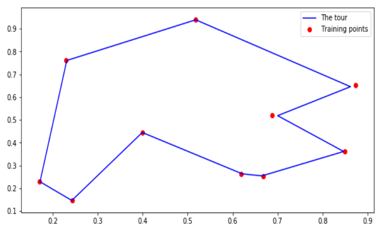
- **The results of clustering by SOM**



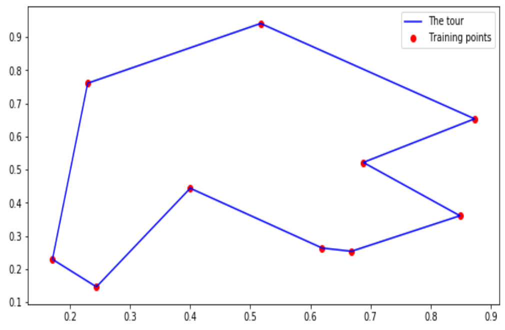
epoch 10



epoch 20



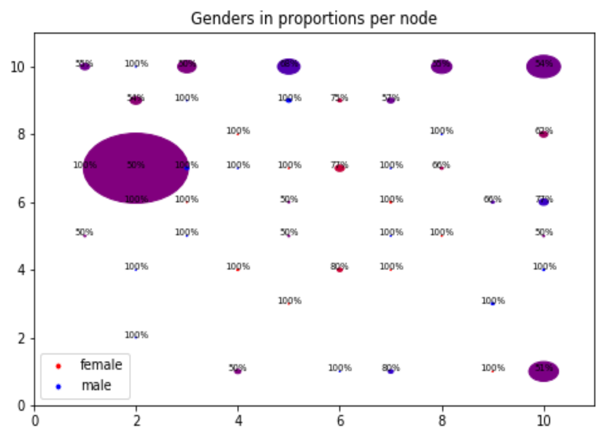
epoch 30



epoch 50

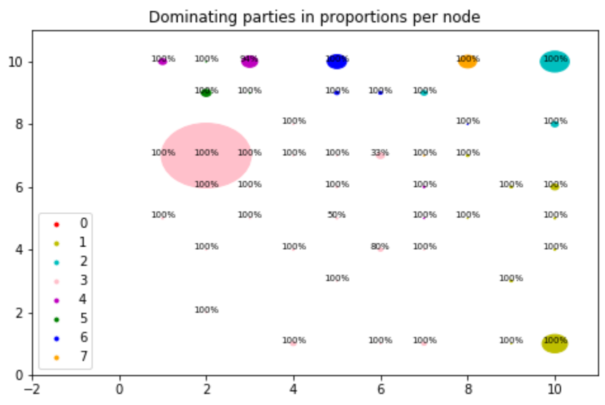
4.3. Clustering with SOM

The training epoch is set to 20 and the range of neighborhood is gradually decreasing from 20 to 0. The learning rate is the same for the winner node and also the neighbor nodes. We have tried to set η to half of the η of the winner node for neighbor nodes but didn't seem much difference in the results.



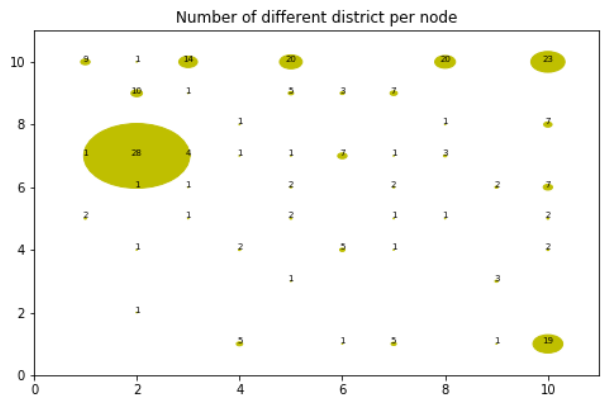
Gender:

Red represents women and blue represents men. The color for each node is proportional to the percentage of a different gender. The more red, the more women. The more blue, the more men. A perfect purple shows that it has equal numbers of women and men. The diameter of the circle is proportional to the number of that particular node among 100 hidden nodes. The big clusters have a mix of genders which doesn't seem to show any patterns of how different genders vote differently.



Party:

There are 8 parties. Each color in the plot is the dominant party for that particular node. And the percentage is the dominant party's percentage for that particular node. The diameter is the same (proportional to the number of that particular node among 100 hidden nodes) Compared with the previous plot, the data points are better clustered. Most people in the same cluster are of the same party.



District:

There are many districts (29) so we don't use a different color for each. The number shown for each node is the number of different districts and the diameter is proportional to the number of that particular node among 100 hidden nodes. As shown in the plot, the biggest cluster has 28 different districts, suggesting districts have little to do with the votes either.

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 networks for regression, ordering and clustering. It was also interesting to interpret those results.