Artificial Neural Networks and Deep Architecture, DD2437

SHORT REPORT

Lab Assignment 2 : Radial basis functions, competitive learning , and self -organisations.

Author :

Putra, Ramadhani Pamapta

Rahgozar, Parastu

Setiawan, Stanley

1. Aim and Objectives

After the lab completion, students are expected to be able to :

- know how to build the structure and perform training of an RBF network for either classification or regression purposes.
- be able to comparatively analyse different methods for initialising the structure and learning the weights in an RBF network.
- know the concept of vector quantisation and learn how to use it in NN context.
- be able to recognise and implement di_erent components in the SOM algorithm.
- be able to discuss the role of the neighbourhood and analyse its effect on the self-organisation in SOMs.
- know how SOM-networks can be used to fold high-dimensional spaces and cluster data.

2. Scope

In this exercise students will implement the core algorithm of SOM and use it for three different tasks. The first is to order objects (animals) in a sequential order according to their attributes. The second is to second a circular tour which passes ten prescribed points in the plane. The third is to make a two-dimensional map over voting behaviour of members of the Swedish parliament. In all three cases the algorithm is supposed to find a low-dimensional representation of higher- dimensional data.

3. Tools Used

- MATLAB 2017a

4. Results/Findings

4.1 Lab 2 assignment - Part I

4.1.1 Batch mode training using least squares – supervised learning of network weights

Sin(2x) Testing and configuration

Using least squares of batch mode training, we found that to achieve absolute residual error below 0.1, 0.01, 0.001 we can use the configuration below:

Residual error requirement	Input space pattern:	Number of input space dimension	Value of actual residual error
Below 0.1	xi = [0; (1/3)π; (2/3)π; (1/1)π; (4/3)π; (5/3)π; 2π]	7	0.0618
Below 0.01	xi = [0; (1/4)π; (2/4)π; (3/4)π; (1/1)π; (5/4)π; (6/4)π; (7/4)π; 2π]	9	0.0018
Below 0.001	xi = [0; (1/5)π; (2/5)π; (3/5)π; (4/5)π; (1/1)π; (6/5)π; (7/5)π; (8/5)π; (9/5)π; 2π]	11	0.0000596
	Width/Distance = 0.4		

Table 1. Input space configuration for certain error threshold.

The difference between each unit in input space is kept at same value (in other words, input space pattern is evenly distributed). The value of width/distance of RBF units is kept at 0.4 . From Table 1 we can see that to achieve lower residual error, we need to increase the number of input dimension.

To enforce this argument of node amount – residual error relationship, we generate testing function of sin(2x) and its relation with changing input and residual error as shown in Figure 1and below :

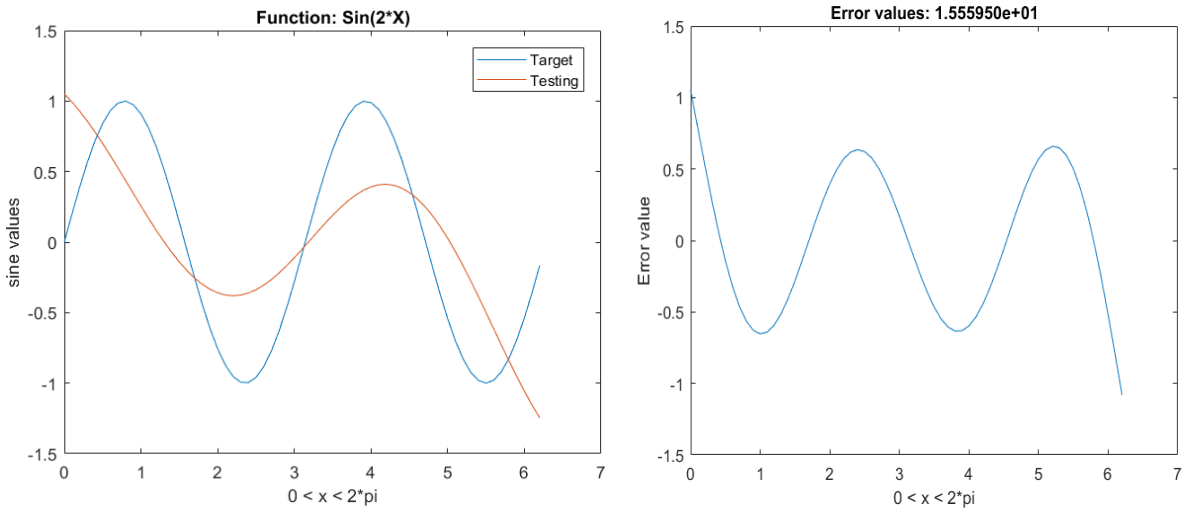

Figure 1.1 Graph showing target and testing of sin(2x) function (left) with the resulting residual error values along 0-2π range (right) using 5 nodes.

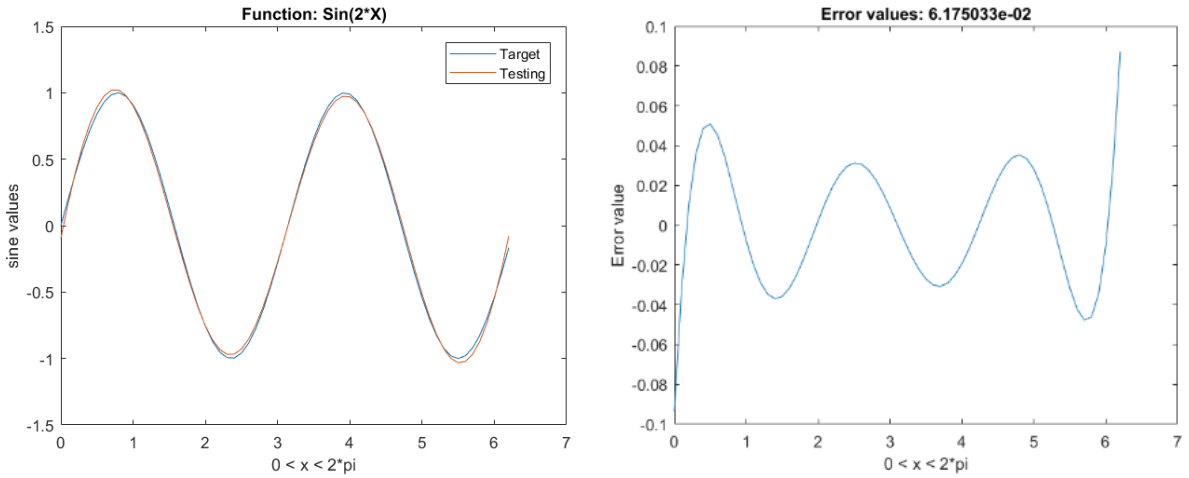

Figure 1.2 Graph showing target and testing of sin(2x) function (left) with the resulting residual error values along 0-2π range (right) using 7 nodes.

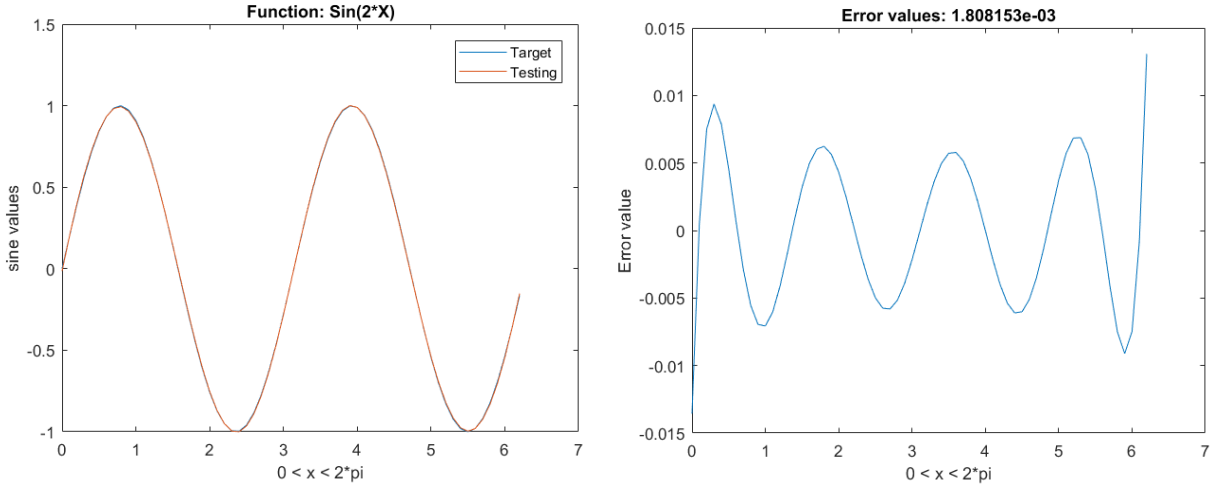

Figure 1.3 Graph showing target and testing of sin(2x) function (left) with the resulting residual error values along 0-2π range (right) using 9 nodes.

Figure 1shows how changes in number of RBF nodes may affect the amount of overall residual error. Figure 1.1 show that using 5 RBF nodes, the testing function seems approximated badly toward the target function. By increasing the number of nodes to 7, as in Figure 1.2, we can see that the testing function becomes better-aligned with its target counterpart. The amount of residual error also seems to decreased significantly. And when we apply 9 RBF nodes, as in Figure 1.3, we see that testing function aligned much better than the previous amount of RBF nodes. We can't even see the difference clearly in left graph because the testing function is

almost aligned perfectly with the target function. However, we still can see the difference in right graph, and it confirms the smaller residual error value.

Square(2x) Testing

In the testing of square(2x) function, we cannot reach threshold residual error of 0.1 , 0.001, and 0.001 using RBF width and node number tuning. However, using comparation of testing and target function we can see how the configuration of width can affect the shape the testing function.

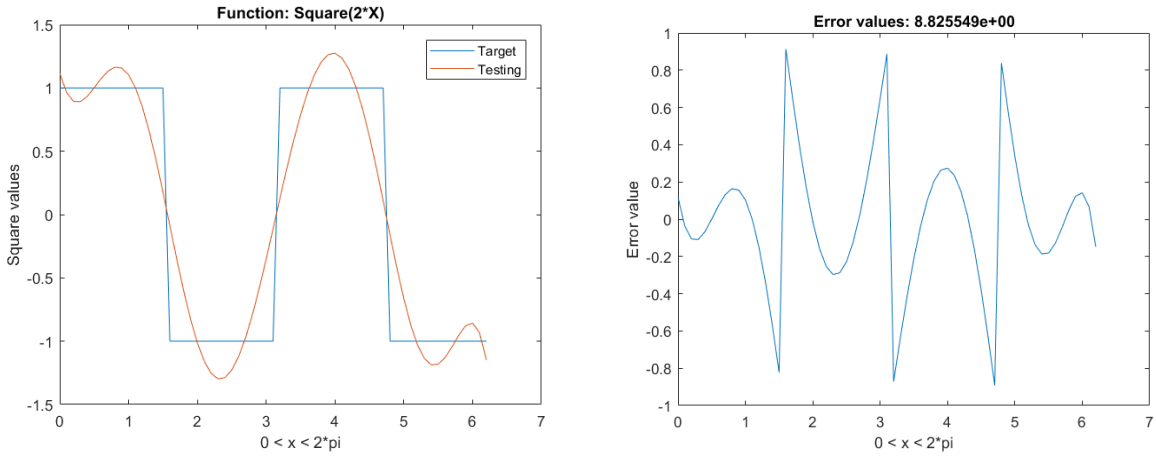

Figure 2.1 *Graph showing target and testing of square(2x) function (left) with the resulting residual error values along 0-2π range (right) using 11 nodes and RBF width of 0.4.*

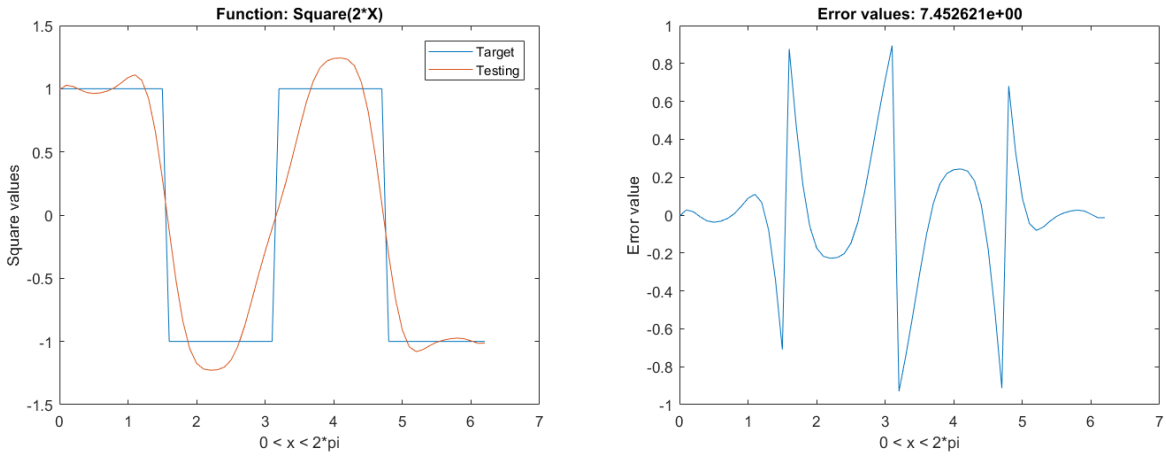

Figure 2.2 *Graph showing target and testing of square(2x) function (left) with the resulting residual error values along 0-2π range (right) using 11 nodes and RBF width of 10.*

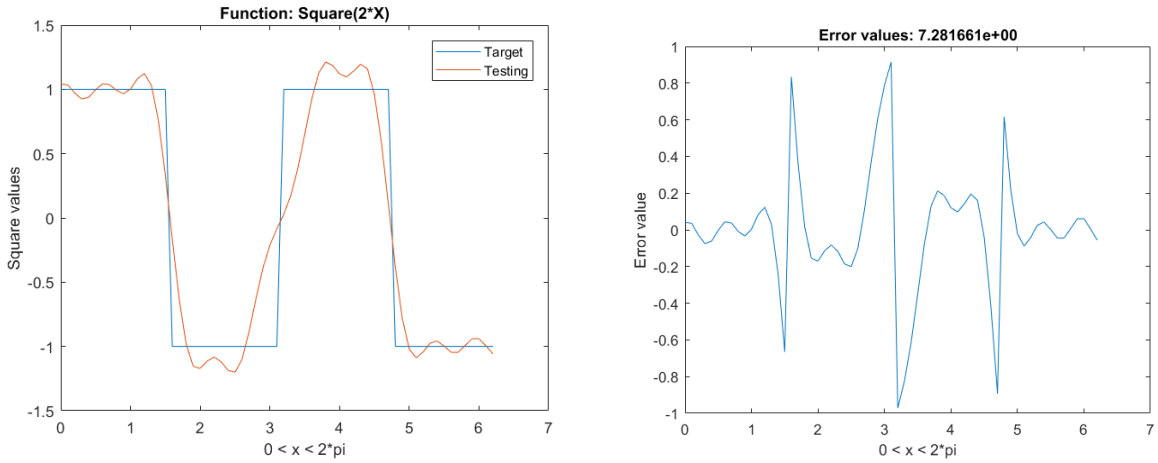

Figure 2.3 *Graph showing target and testing of square(2x) function (left) with the resulting residual error values along 0-2π range (right) using 11 nodes and RBF width of 15.*

From Figure 2 above, we can observe that using manipulation in RBF width, we can change the shape of the testing function. In Figure 2.1 , RBF width of 0.4 is not sufficient enough to make testing function aligned well with its target function. Then, we increased the RBF width to 10, as in Figure 2.2. The alignment of testing function seems better than the last one. And we increased the RBF width again to 15, making testing function seems to follow the pattern of target function better. However, there is a problem with signal in higher frequency. Theoretically, if we apply low pass filter in output/testing signal, the higher frequencies of testing signal will attenuate and the residual error should be lowered.

4.1.2 Regression with noise

Compare the two learning approaches

By comparing two learning approaches : incremental RBF and backpropagation for MLP, we can observe the results as in Figure 3 below :

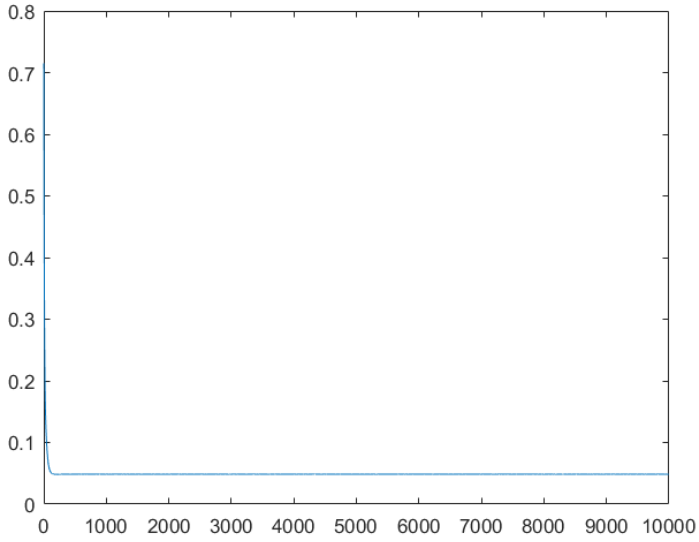

Figure 3.1 Error convergence rate with incremental RBF, 9 nodes.

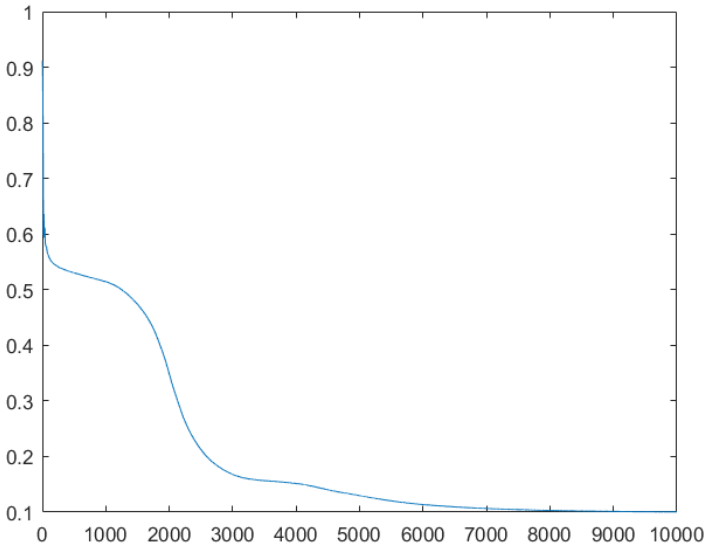

Figure 3.2 Error convergence rate with backpropagation MLP, 4 hidden nodes.

From Figure 3.1 and Figure 3.2 we compare incremental RBF using delta-rule with multilayer perceptron, by equivalent number of nodes (9 nodes for RBF and 4 hidden nodes for MLP). We can see that by using RBF, the error stabilized far earlier than the MLP counterpart. Also, value of error is smaller in RBF. Further observation using larger nodes (11 nodes for RBF and 5 hidden nodes for MLP) can be observed in Figure 3.3 and Figure 3.4 below:

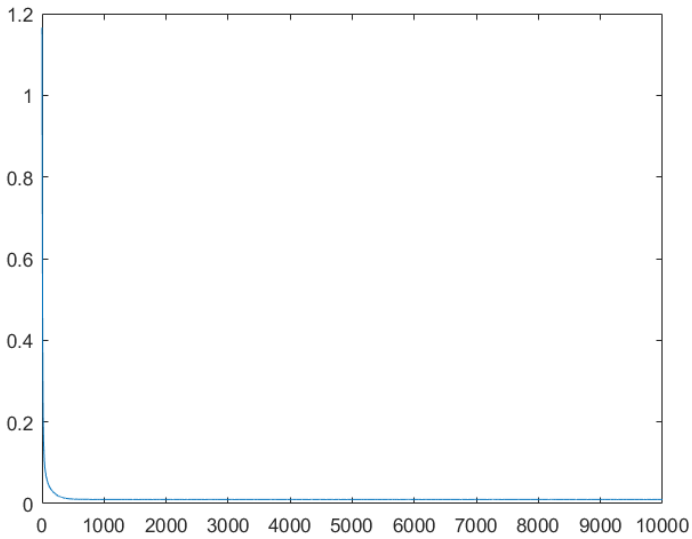

Figure 3.3 Error convergence rate with incremental RBF, 11 nodes.

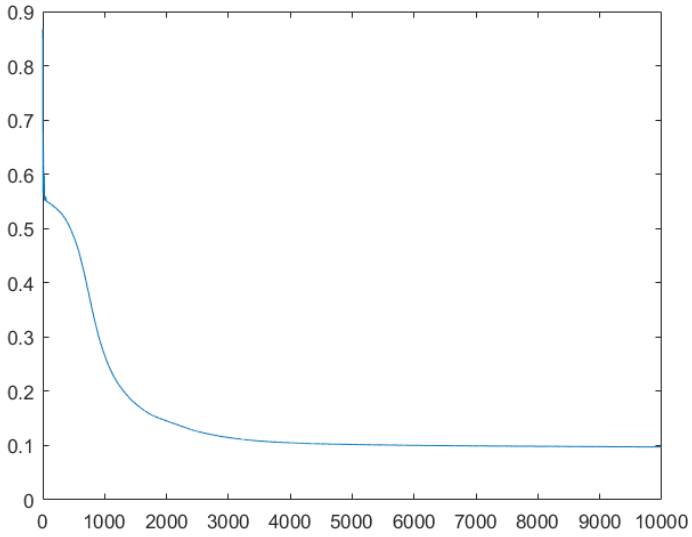

Figure 3.4 Error convergence rate with backpropagation MLP, 5 hidden nodes.

From comparison between Figure 3.3 and Figure 3.4 , we can observe the effect of increased nodes: the main change in MLP is faster error stabilization and no significant error change, while in RBF we see that the error value drops quite significantly but the error stabilized longer.

Configuration of RBF node positioning

Configuration of RBF node positioning in input space can be influential to the residual error . We've tested this using batch mode training with least squares as we had in Table 1 and additional configurations in Table 2 below:

Configuration name	Configuration detail (input space pattern)	Residual error	Input pattern distribution
Config 1	xi = [**0;** (1/4)π; (1/3)π; (2/4)π; (2/3)π; **(1/1)π**; (4/3)π; (5/3)π;**2π];**	0.4787	lower
Config 2	**cc** xi = [**0;** (3/4)π; (2/3)π; (6/5)π; **(1/1)π**; (5/4)π; (4/3)π; (6/4)π; **2π];**	0.4774	middle
Config 3	xi = [**0;** (2/4)π; (2/3)π; **(1/1)π**; (5/4)π; (4/3)π; (6/4)π; (5/3)π; **2π];**	0.4774	upper
Config 4	xi = [**0;** (1/4)π; (2/4)π; (3/4)π; **(1/1)π;** (5/4)π; (6/4)π; (7/4)π; **2π];**	0.4775	even
Number of nodes in input space is kept at 9 nodes. Value of RBF width is kept at 0.4 .			

Table 2. Configuration of RBF nodes in input space with their following residual error.

To show distribution of input pattern more clearly, we can observe Figure 4 below

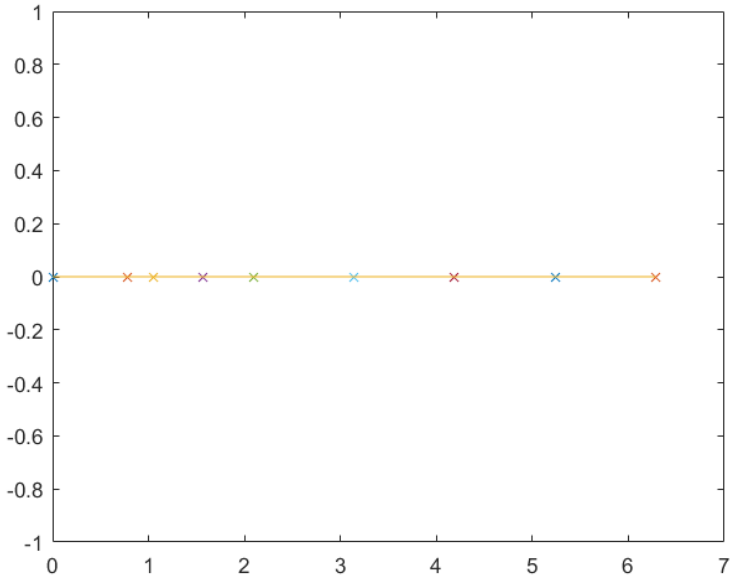

Figure 4 .1 Graph showing Config 1 input pattern distribution from 0 to 2π (6.283), with residual error 0.4787

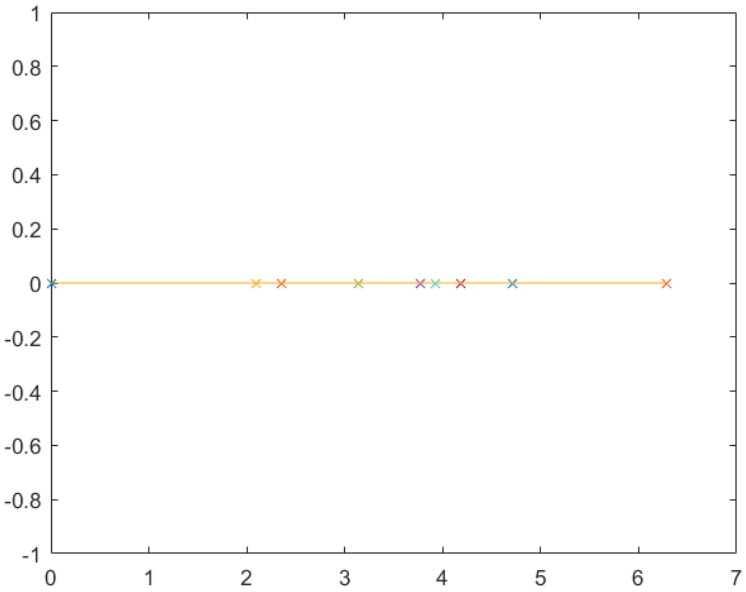

Figure 4 .2 Graph showing Config 2 input pattern distribution from 0 to 2π (6.283), with residual error 0.4774

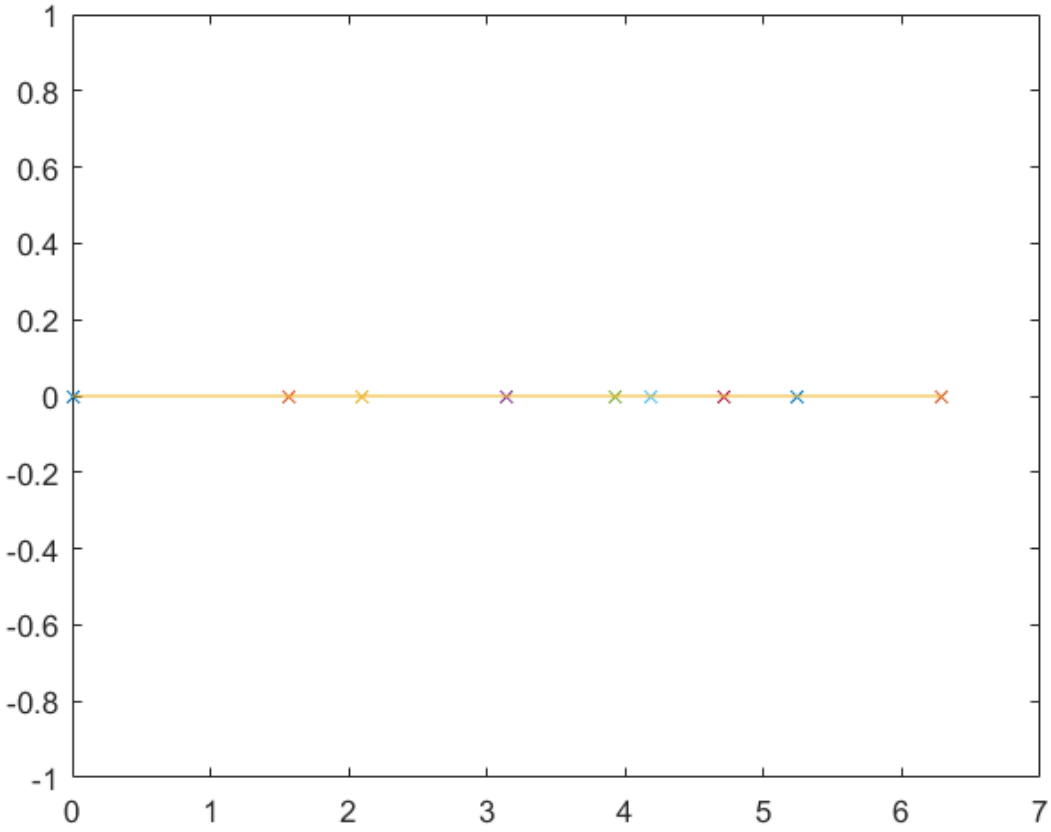

Figure 4 .3 Graph showing Config 3 input pattern distribution from 0 to 2π (6.283), with residual error 0.4774

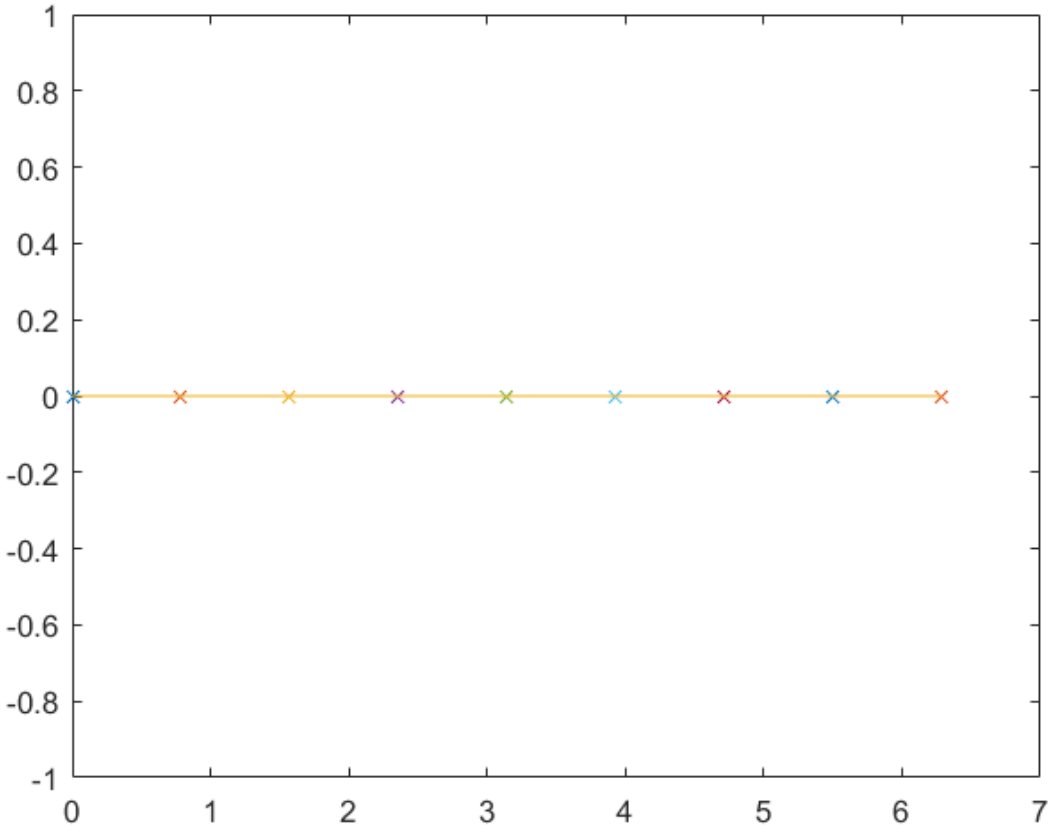

Figure 4 .4 Graph showing Config 4 input pattern distribution from 0 to 2π (6.283), with residual error 0.4775

From Table 2 and Figure 4 above, we can observe that there are 4 different input pattern used. Config 4 is evenly distributed with same difference between each node. Config 1, Config 2 , and Config 3 distribution differs in their focused distribution. Config 1 was set to more likely to distributed in lower level of 0-2π distribution range. Config 2 was set to more likely to distributed in middle of 0-2π distribution range (close to π). Config 3 was set to more likely to distributed in upper level of 0-2π distribution range. These set of pattern in Config 1, 2 and 3 was made manually by hand, and not entirely random. By using different distribution of input pattern, there are only very small differences on the value of residual error. Using Config 1—which has even distribution of input pattern—led to highest residual error than its other three distribution counterparts (0.4787 vs 0.4774, 0.4774, and 0.4775 respectively).

Main effects of changing the width of RBF node

Main effects of changing the width of RBF can be seen in Figure 5 below

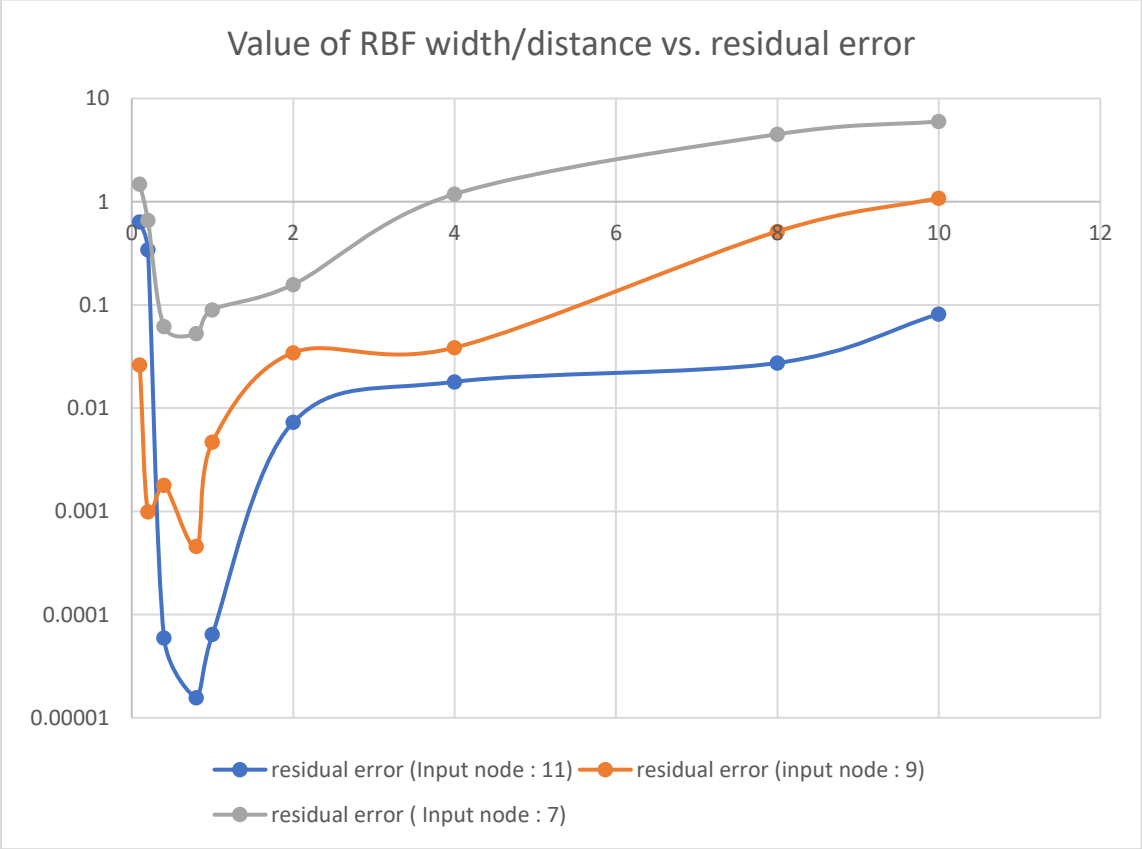

Figure 5 Graph showing influence of larger input pattern amount and RBF width(x-axis) toward value of residual error(y-axis). Y-axis is in logarithmic scale.

From Fig. 5 we can see that by increasing the number of RBF width, the residual error will slowly increase. There is an anomaly which is may caused by programming limitation that shows somewhat high value of residual error in very small RBF width. However, in general we see that RBF width inversely proportional to the amount of residual error. Data from multiple number of input nodes (7 , 9, and 11) confirmed this. The configuration of RBF is using batch mode training with least squares as we had in Table 1. The input is sin(2x) without noise added, for the sake of an ideal condition and stability. Since the residual error can be very low in case of small RBF width, in Figure 5 we use logarithmic scale in y-axis.

Rate of convergence change with different values of `eta`

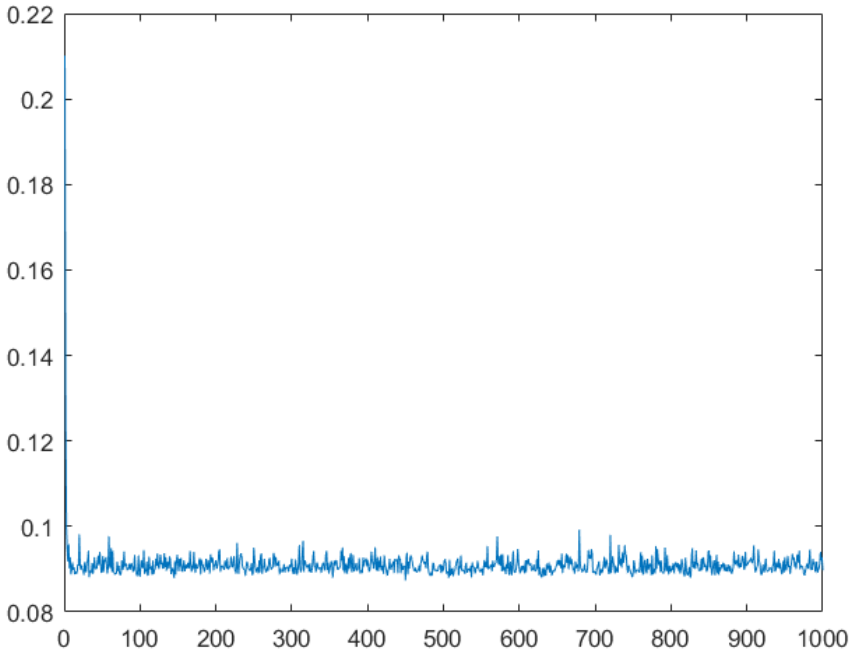

Figure 6 .1 Rate of convergence, using `eta` value of 0.25

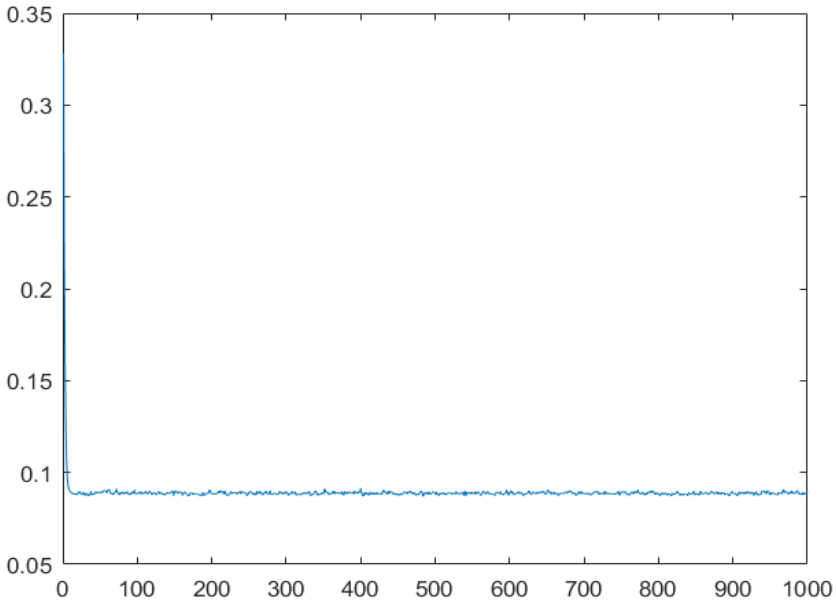

Figure 6 .2 Rate of convergence, using `eta` value of 0.10

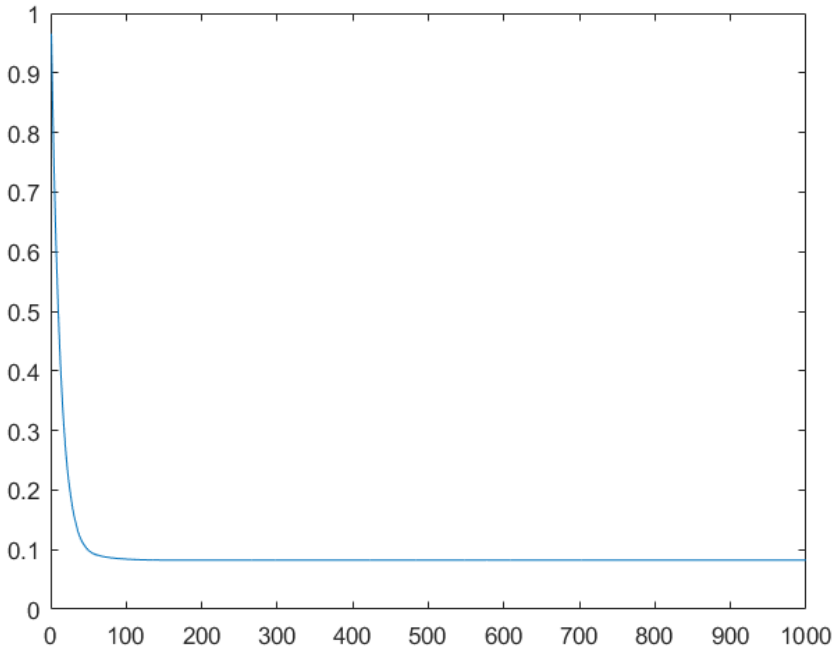

Figure 6 .3 Rate of convergence, using `eta` *value of 0.01*

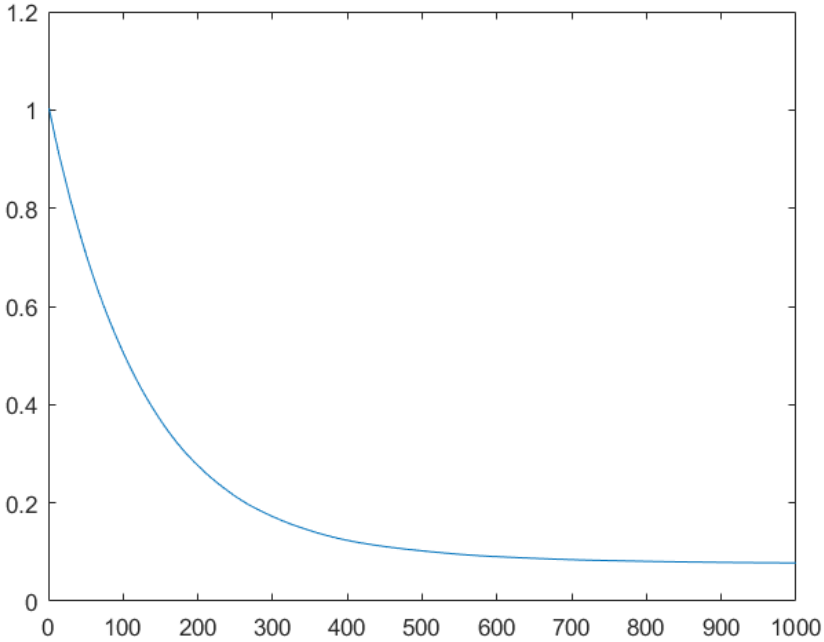

Figure 6 .4 Rate of convergence, using `eta` *value of 0.001*

The Figure 6 shows the generated rate of convergence using different values of eta, by on-line learning. The configuration used is 1000 max epoch and 0.4 epoch. From Figure 6 .1 we can observe that by using eta value of 0.25 the rate of convergence somehow looks noisy where the error value frequently changes but stable. However we can also see that it took less than 25 epoch to for the error reach to reach stability. Figure 6 .2 , using 0.1 eta shows similar result with previous eta value, only with better error frequency. In Figure 6 .3, by using eta value of 0.01 we can see that the error frequency is even better although the error stabilized at later epoch (almost reached 100) than two previous tests. Then,by using eta value of 0.001 we can see that the error stabilized far longer, as shown in Figure 6. 4.

4.1.3 Competitive learning for initialization of RBF units

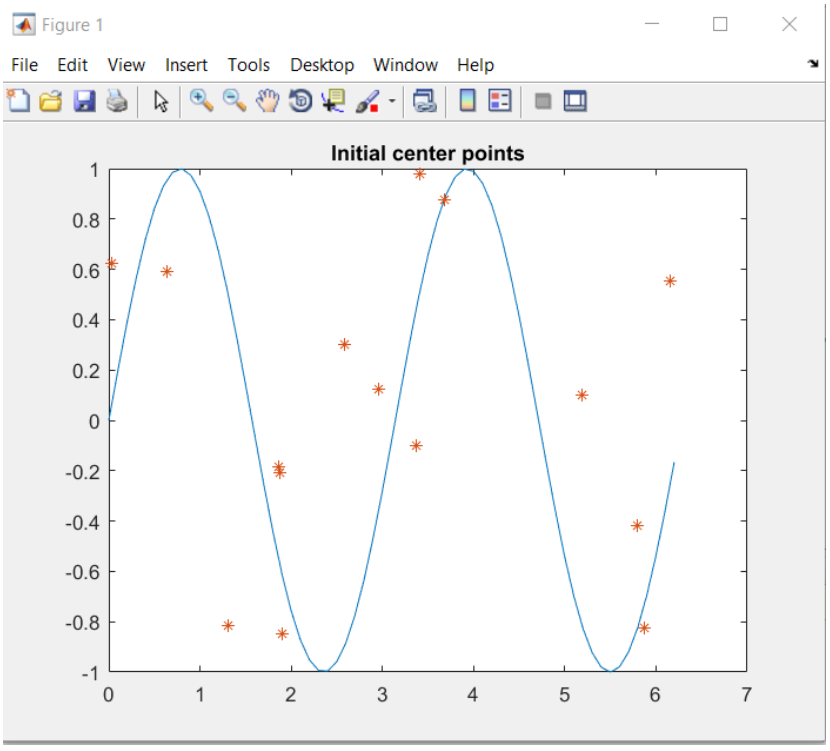

Figure 7.1 Initial center points for competitive learning

This part of the lab, we randomize the location of center points. After we randomize, the algorithm is supposedly to find the location of the best point possible to get the best output for both the noise and the normal data of sin(2x) shown on Figure 7.1.

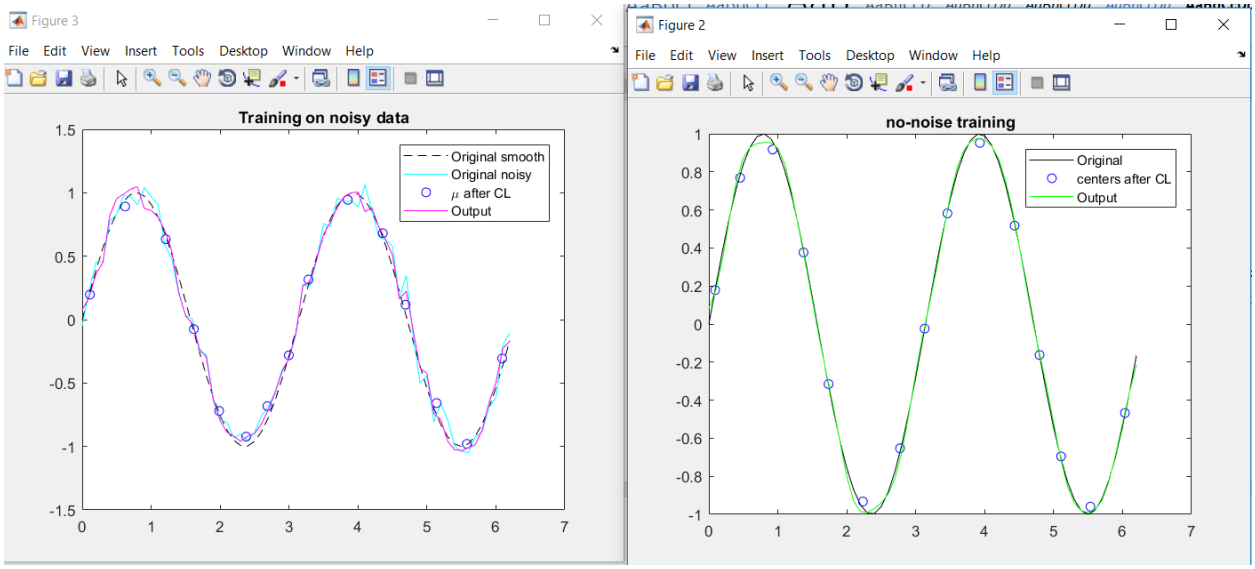

Figure 7.2 (Left) training on noisy data. (Right) training on no-noise data. The centers tried to match with the location of the data. Hence, there are a little change in position of the centers for the two different functions.

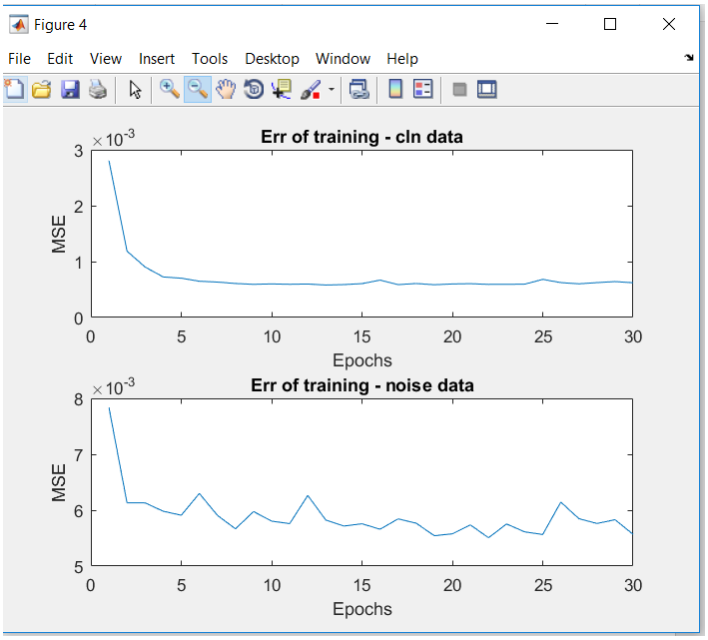

Figure 7.3. Top figure represents the error for clean data ($6 * 10^{-4}$) and bottom represents the error for noisy data ($6 * 10^{-3}$)

After we have the randomized center, we can see on Figure 7.2 that we can manage to set the centers into the best position according to the function. Either it is the ordinary sin(2x) or the one with a noise addition.

What we can see and analyze from these results is that the error for noisy data is more than the clean data. For clean data, we can get $6 * 10^{-4}$ error value, where as for noisy data it fluctuates and have an error value around $6 * 10^{-3}$. These findings can be seen on Figure 7.3.

Do note that we have already tried to set this competitive learning algorithm so that it will reduce the probability of dead nodes. We tackle the "dead node" problem by randomizing the order of centers which we will first use to calculate.

- **Ballist and Balltest Import Part**

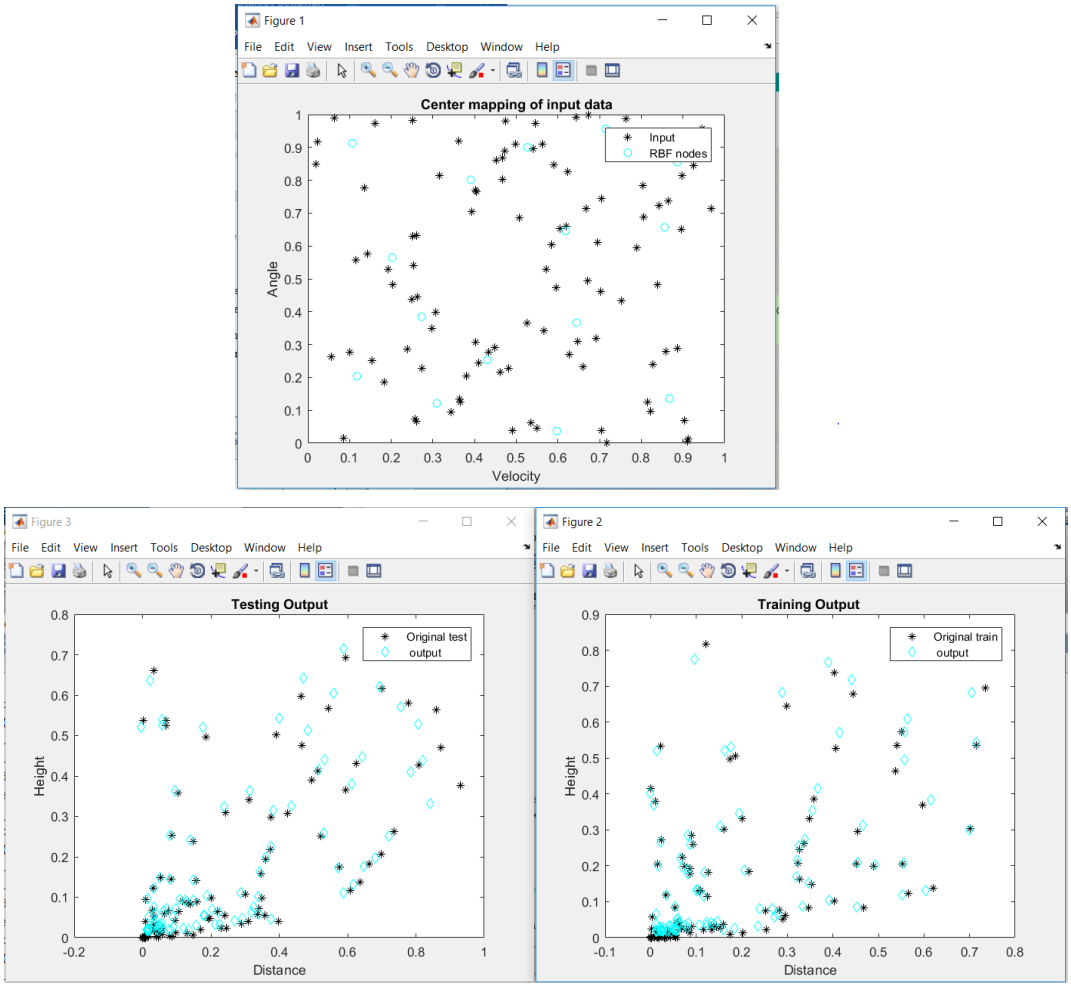

Figure 7.4 Top represents the mapping of input and the randomized RBF nodes. Bottom Left shows the result of testing output and bottom right shows the result of training output. It uses 15 units.

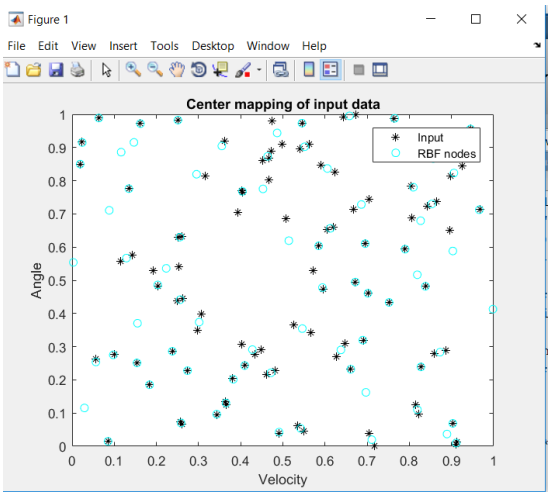

Figure7. 4 Initialization of 80 center points

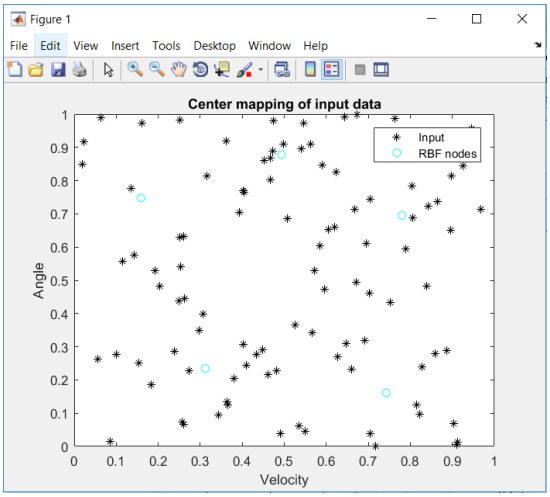

Figure 7.3 Initialization of 5 center points

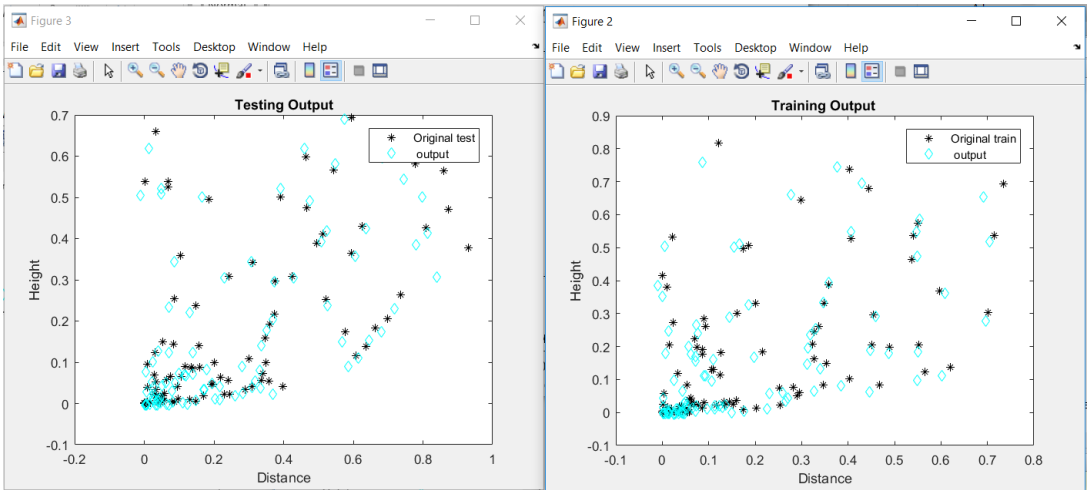

Figure 7.7 Test and Training data for 5 Center points

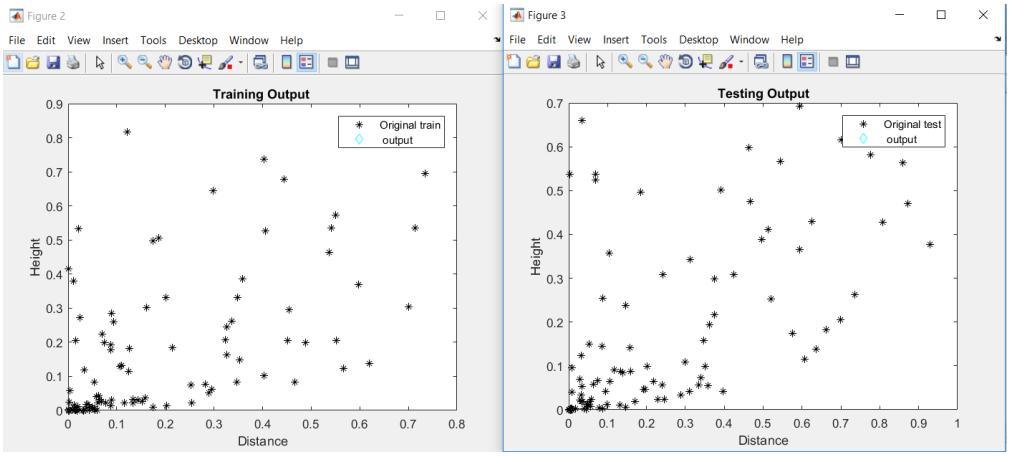

Figure 7.8 Test and Training data for 80 Center points.

On this part of the lab, we tried to load in Ballist.dat and Balltest.dat. our aim was to load the 2-dimensional data and then test it with the competitive learning algorithm. On figure 7.4,

we used 15 units as an example. Testing output means that we use ballist.dat and for training the output, we use balltest.dat.

We tried using 15 center points(nodes), 5 center points(nodes) and 80 center points(nodes). What we found out is that by using 80 center points, there will be no output at all from the result. We strongly believe that it is because of overfitting and then it will not produce any output at all, which can be seen in Figure 7.8. Comparing Figure 7.4 (15 center points) and Figure 7.7 (5 center points), we can say that the results for 15 center points are better than 5 center points because we can see that the output converge more for 15 center points rather than only using 5 center points.

4.2 Lab 2 Assignment – Part II

4.2.1 Topological ordering of animal species

Animal	Cluster
'dragonfly'	1
'beetle'	4
'grasshopper'	4
'butterfly'	7
'moskito'	10
'housefly'	13
'spider'	20
'pelican'	27
'duck'	30
'penguin'	34
'ostrich'	37
'frog'	42
'seaturtle'	46
'crocodile'	49
'walrus'	54
'dog'	59
'bear'	62
'hyena'	63
'ape'	67
'lion'	70
'cat'	72
'skunk'	75
'bat'	80
'rat'	80
'rabbit'	83
'elephant'	86
'kangaroo'	91
'antelop'	93
'horse'	96
'pig'	99
'camel'	100
'giraffe'	100

In this part, the data was provided from before and our task is to read 32 names of animals, plus their 84 attributes from another file. By creating a random matrix of weights 100x84, we

calculate the distance from each weight to each input, and then we pick the least distance as the winning vector. Based on that, the weight and neighboring vectors will be updated.

At the end, for sorting out the animals based on mutual features, another loop of 32 times will run. The result is shown in the below table. As it can be seen, for instance, insects are close to each other and their group is further away from mammals or other animals.

4.2.2 Cyclic Tour

For the cyclic tour problem, as it can be seen, we have 10 points with represent the cities and this time, our input is two dimensional. Neighborhood update will be first 2 near ones, then one and then zero. For better observation, we tried both with 10 and 20 iterations. In the second picture, it is visible that not necessarily a high number of repetitions will not give us a better result, and we reach the overfitting problem.

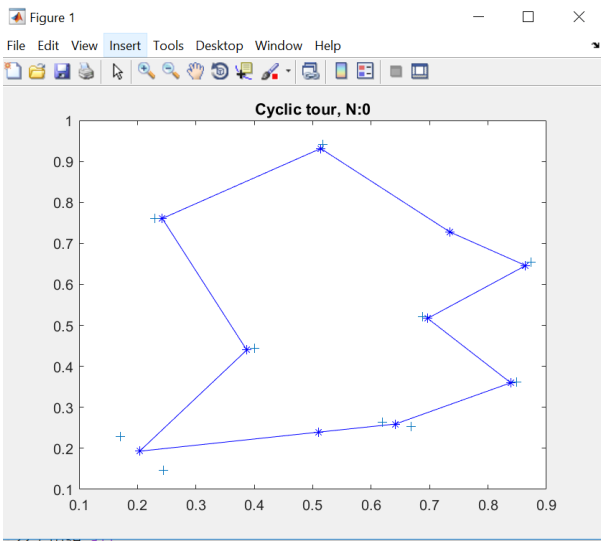

Figure N – Cyclic tour with 10 iterations

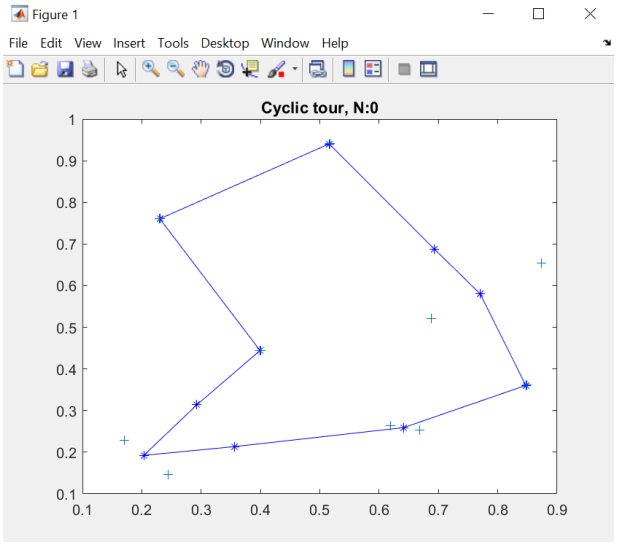

Figure M – Cyclic tour with 20 iterations

4.2.3 Data Clustering: Votes of MPs

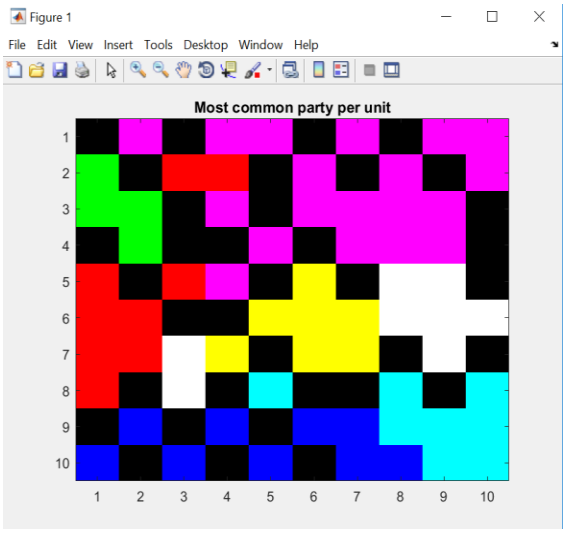

Figure P – Showing the clustering and distribution of parties

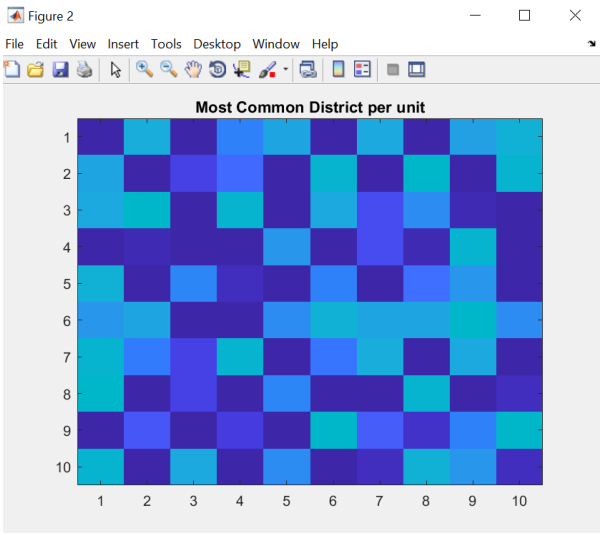

Figure O – Distribution among voting districts

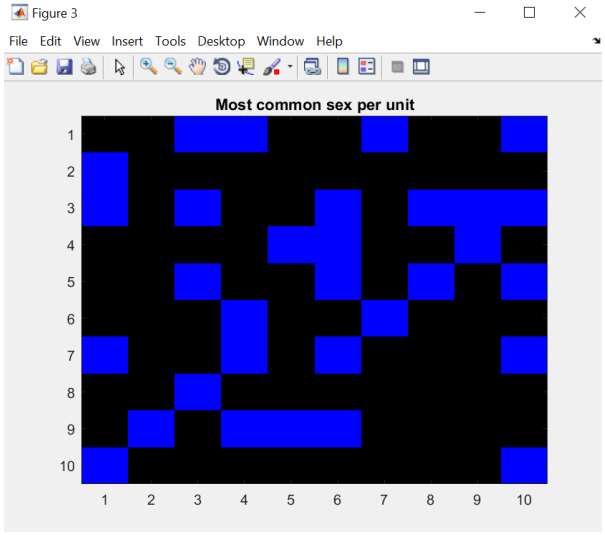

Figure Q – Distribution between Men and Women voters

According to the figures, the position of each party and the surface, matches the data correctly. Also, there is no specific pattern for district distribution, which is logical due to the diversity of votes in the country. Furthermore, the sex also does not play a clear role in the propagation of the votes throughout the country.

5. Reflection

- In RBF, the higher number of nodes in input space tends to lead toward smaller value of residual error.
- Smaller RBF width tends to make smaller value of residual error.
- In noisy signal case, distribution/positioning of RBF nodes does not make any significant difference on residual error.
- By using RBF, to approximate square function accurately, one need to utilize low-pass filter toward the output.
- In adjusting correct eta value, we need to find the balance between error stability and how much time needed for the error to be stable.
- Increasing the number of nodes in MLP is useful faster error stabilization. In RBF, increasing number of nodes is useful in accuracy (lower error value) but with longer error stabilization.

6. Conclusion

- By using RBF, we can suppress the value of residual error using larger node amount and smaller RBF width.
- By using RBF, it is harder to approximate square(2x) function than sin(2x) function.
- By randomizing the centers for competitive learning, the result will have a better outcome.
- We found out that RBF are better than MLP for this lab in respect to convergence rate and the number of iterations it used to reach the minimum error value
- For the competitive learning, we need to see how many number of nodes results the best outcome. Too many nodes will result in overfitting and will not produce any output at all.
- When it comes to comparing different methods of training, SOM has more accurate results due to the fact that it updates the neighboring neurons as well as the main weight vector.