Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Ravi Bir—ravib@kth.se
Bharat Sharma—bsharma@kth.se
Qiao Ren—qiaor@kth.se

February 19, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to implement and compare batch and sequential RBF networks and investigate how changing their parameters affects their performance
- to implement competitive learning with the RBF network and investigate how it affects its performance
- to implement SOM algorithm for ordering and clustering problems and learn a few ways to visualize the output for different types of input sets.

2 Methods

Jupyter Notebook, Numpy, Matplotlib

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

3.1 Function approximation with RBF networks

First we created the training and testing datasets for the sin(2x) and square(2x) functions. We then applied the batch RBF learning algorithm on the sin(2x) and square(2x) training datasets. We set the RBF means at equal distances along the input space. We then tested different values of the variance and the number of units to see what the minimum number of units needed was to obtain an absolute residual error below 0.1, 0.01 and 0.001. The results are in Tables 1 and 2.

Threshold	Absolute Residual Error Achieved	Number of units	Variance
0.1	0.0997	5	0.695
0.01	0.0099	13	0.080
0.001	0.00099	56	0.007

Tabell 1: Parameters needed to meet required thresholds for sin(2x)

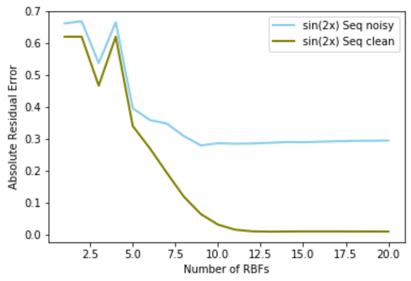
Threshold	Absolute Residual Error Achieved	Number of units	Variance
0.1	0.0958	35	0.015
0.01	-	-	-
0.001	-	-	-

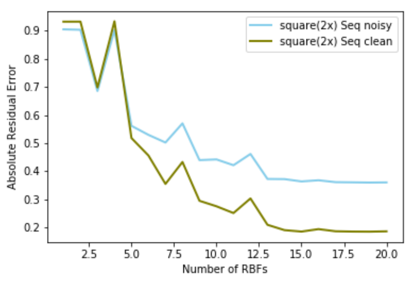
Tabell 2: Parameters needed to meet required thresholds for square(2x)

To reduce the residual error to 0 for the square(2x) function, we could use a sign activation function on the output. This would round each output to 1 or -1. This is useful in binary classification, where a point could belong to one of two classes.

Next we added zero mean Gaussian noise to the training and test datasets. We then implemented an on-line version of the RBF delta rule and investigated how changing different factors would affect its performance. We used the absolute residual error as our error estimate. First we investigated how the number of RBFs and the widths affected the performance. Learning rate was set to 0.01 and epochs was set to 500. Figure 1 shows how the number of RBFs affects the performance on the noisy sin(2x) and clean sin(2x) datasets. Figure 2 shows it for the noisy square(2x) and clean square(2x) datasets. A fixed variance of 0.1 was used.

We can see that, for both the noisy and clean datasets for both functions, as the number of RBFs increases, the error decreases. Increasing the number of RBFs stops significantly decreasing the error around 10 RBFs for the sin(2x) function and 13 RBFs for the square(2x) function.

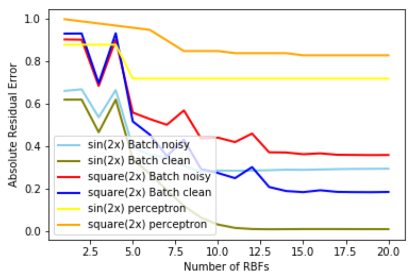




Figur 1: sin(2x) Number of RBFs vs Error

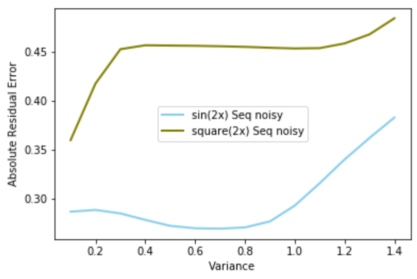
Figur 2: square(2x) Number of RBFs vs Error

Figure 3 shows the results for batch learning. The results are similar to sequential learning. We also compared the batch RBF network with the batch single-hidden-layer perceptron. The perceptron performs worse than the RBF network and has a slower training time.



Figur 3: Batch Number of RBFs vs Error

Next we tested how the variance affects the performance. Figure 4 shows how the width of the RBFs affects the performance on the noisy sin(2x) and noisy sin(2x) datasets. The on-line version of the RBF delta rule was used. A fixed number of 10 RBFs was used.

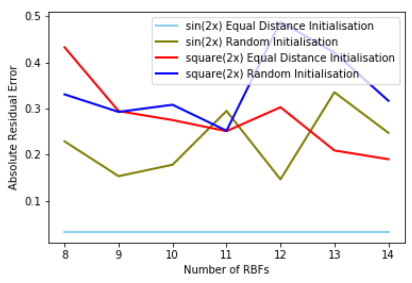


Figur 4: Variance vs Error

For sin(2x), we can see that as the variance increases up to 0.6, the error decreases, but after a variance of 0.8 the error begins to increase again. For square(2x), we can see that as the variance increases, the error increases, but it remain flat between a variance of 0.3 and 1.2.

As the learning rate increased, the rate of convergence for the on-line algorithm would also increase, so it would converge faster. However, when the learning rate was too high, the best solution with the lowest error was not found.

Next we investigated how the positioning of the RBF nodes in the input space affected the performance. Up until this point we have been evenly positioning the RBFs in the interval space, but we will investigate if randomly positioning them in the input space will give a lower error. We investigated the sequential RBF algorithm for both functions, and compared their performance for randomly positioning the RBFs vs positioning them at regular equal intervals. The results are in Figure 5



Figur 5: Investigating RBF Initialisations

We can see that for both functions, at every RBF number, the equal distance RBF initialisation gave a lower error when compared to the random initialisation. The only exception is for square(2x) at 8 RBFs.

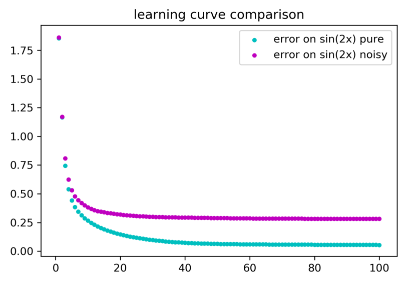
3.2 Competitive learning for RBF unit initialisation *(ca. 1 page)*

After using competitive learning to adjust the position of RBFs, the error has been reduced. We observed this in the cases: pure and noisy sin(2x) (Table 3). The learning curve on pure sin(2x) converges on a lower error than noisy sin(2x). (Figure 6).

To avoid dead units, we chose an approach that updates the positions of all the RBFs. Table 4 shows that updating the positions of all RBFs generates lower error compared with updating only one RBF.

	pure sin(2x)	noisy sin(2x)
evenly distributed RBF	0.0555	0.2925
evenly distributed RBF + CL	0.0547	0.2919

Tabell 3: a comparison between using and not using competitive learning on the pure sin(2x) function and noisy sin(2x) function. (CL means competitive learning. learning rate = 0.01 number of RBF = 13 var of RBF =0.4)

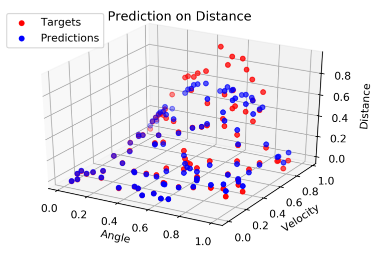


Figur 6: learning curve of pure and noisy sin(2x), when using competitive learning with only one winner

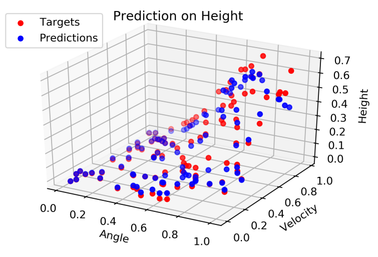
	without CL	CL with one winner	CL with all winners
absolute residual error	0.0555	0.0547	0.0536

Tabell 4: error on pure function sin(2x), in 3 different cases. (CL means competitive learning. learning rate = 0.01. number of RBF=13 var of RBF =0.4)

In the ballistical experiment, we have a two dimensional dataset. We initalize the RBFs by evenly distributing them. We apply competitive learning to adjust the positions of the RBFs. The best performance was achieved when the variance of the RBFs was set to 0.2 and the number of nodes was 10. The error is 0.0304. Figure 7 shows the output distance. Figure 8 shows the output height.



Figur 7: output distance, with two input variables angle and velocity

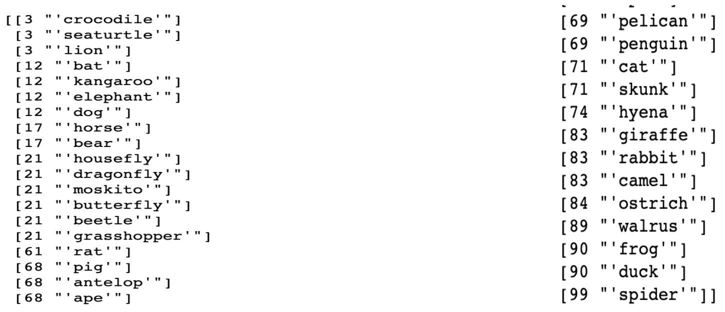


Figur 8: output height, with two input variables angle and velocity

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

4.1 Topological ordering of animal species

Training was made over 20 epochs with learning rate η to be 0.2. We decrease the size of the neighbourhood after each epoch. Below is the result of applying SOM to the animal data -



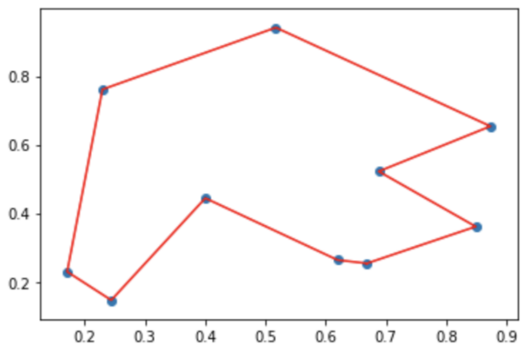
```
[[3 "'crocodile'"]
 [3 "'seaturtle'"]
 [3 "'lion'"]
 [12 "'bat'"]
 [12 "'kangaroo'"]
 [12 "'elephant'"]
 [12 "'dog'"]
 [17 "'horse'"]
 [17 "'bear'"]
 [21 "'housefly'"]
 [21 "'dragonfly'"]
 [21 "'moskito'"]
 [21 "'butterfly'"]
 [21 "'beetle'"]
 [21 "'grasshopper'"]
 [61 "'rat'"]
 [68 "'pig'"]
 [68 "'antelop'"]
 [68 "'ape'"]
```

```
 [69 "'pelican'"]
 [69 "'penguin'"]
 [71 "'cat'"]
 [71 "'skunk'"]
 [74 "'hyena'"]
 [83 "'giraffe'"]
 [83 "'rabbit'"]
 [83 "'camel'"]
 [84 "'ostrich'"]
 [89 "'walrus'"]
 [90 "'frog'"]
 [90 "'duck'"]
 [99 "'spider'"]]
```

Figur 9: Animal ordering

The order varies from one training to another but the general trend remains the same. We can see that insects are grouped together. Sea turtle and crocodile are often together as well as frog and duck as they are water type. 4 legged species are often together.

4.2 Cyclic tour

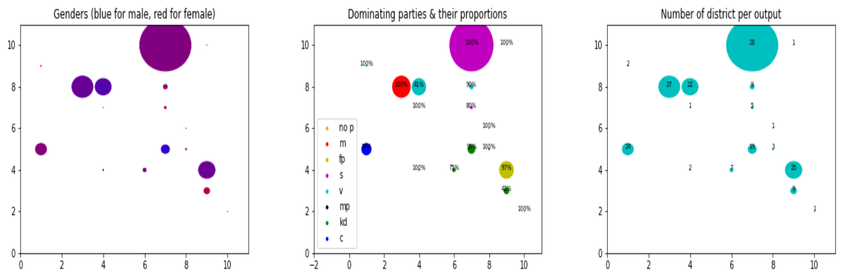


Figur 10: City tour

We had to find a cyclic tour minimizing the distance covered. This is referred to as Travelling sales man problem as well. The only difference between this and the animal problem is that here, we had to find a circular neighbourhood.

The training the done with 20 epochs and a a learning rate η of 0.2. We keep on decreasing the neighbourhood after each epoch. We are able to find a good tour as shown in the figure above but sometimes, the algorithm takes a detour between the two points in the bottom left edge. This might be due to very small neighbourhood or less training.

4.3 Clustering with SOM



Figur 11: Vote mapping with different attributes

- A cluster is blue if mostly men are in it and red if women. If its purple, that means the ratio of men and women is quite similar. Looking at the map, we can say that sex is not a deciding factor in voting.
- Only one party seems to be on the top right of the grip with a huge cluster. Other parties vote differently and are scatter over the cluster. Hence, the type of parties does effect how the MPs vote. Each cluster has most people from the same party.
- The districts in each cluster vary with cluster size. The biggest cluster has 28 districts. The districts seem quite evenly distributed and hence doesn't effect the voting trend much.

5 Final remarks *(max 0.5 page)*

Changing the hyperparamters of an RBF network has a significant effect on its performance. The number of RBFs and the widths of the RBFs needs to be set to their optimum values depending on whether batch or on-line learning is used. We found that positiong the RBFs equally along the interval gave better results when compared to randomly placing them along the interval. Also, competitive learning can be used to improve the performance of RBF networks, and techniques can be used to avoid dead units.
With SOM, it was interesting to see how problems with higher dimensions could be projected on lower dimensional plane. One of the issues faced by us was to

find different kind of neighbourhoods for different types of problems as it wasn't very intuitive to visualize how the neighbourhood world work and change over increasing iterations.