Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Romina Arriaza, Clara Tump, Konstantinos Saitas - Zarkias

February 7, 2019

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to understand RBF networks and be able to implement them
- to get experience with using competitive learning techniques such as vector quantization for RBF initialisation
- to understand the SOM algorithm and be able to apply it to different problems such as data ordering, path finding and data visualisation

In the first part of the assignment we implement an RBF network, and use the competitive learning technique vector quantization (VQ) to initialize it. In the second part, we implement the SOM algorithm and use it for three different tasks: ordering, cyclic path finding and data representation.

2 Methods

All the implementations were written using python 3.4. All the models are implemented by hand, without the use of machine learning toolboxes.

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

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

For this part of the assignment, we implemented from scratch a RBF Network and tested it on different settings and data. The data used were $sin(2x)$ and $square(2x)$ (as mentioned in the assignment) in the range $[0, 2\pi]$. Firstly, we used the least squares learning method to update the weights of the network by using batch learning. In order to find the optimal model we tested different number of nodes as show in 1. We set the values of μ of the nodes by sampling evenly in the range $[0, 2\pi]$, so they were evenly distributed across the input space of our data. The σ of the nodes was fixed to 0.1 for all the nodes in this case.

# nodes	2	5	10	20	30	40	50	63(N)
$sin(2x)$: Test error	0.6	0.2	0.02	0.002	0.0001	0.0001	4.4e-5	9.6e-5
$square(2x)$: Test error	0.9	0.4	0.3	0.2	0.15	0.14	0.14	0.13

Table 1: Table of error comparison with different number of nodes

As we can see from 1 the error decreases as the number of nodes increases and we find the optimal model (minimum test error) when # samples (N) = # nodes (n). This can be justified as the system of least squares is well determined when $N = n$ and every RBF node corresponds to a data point. When $n < N$ the network can still approximate a solution but it is not optimal since one node might be responsible to approximate a larger area in the output space. On the other hand, when $N > n$ the system is overdetermind and we cannot solve it using the least squares (Gaussian elimination) approach. Moreover, we can see that for the square function our accuracy is still not very good, one simple way to fix this would be to pass the predicted output through the same *filter* (1 for $y >= 0$, -1 for $y < 0$) used to create this function. Then we can see that even with $n = 30$ nodes we can reach a test error of 0. This type of transformation could be used in 2-class problem (binary classification) and the output of our network needs to go through a threshold function (step function).

3.2 Regression with noise

In this part instead of using the least square approach to update the weights we used delta rule. This approach is based on finding the values of the weights that minimise the error to the target values by following the steepest path on the error space (*gradient descend method.* Furthermore, to train the network with delta rule, we used online (sequential) learning in which the network's weight are being updated sample by sample. Additionally, a small amount of Gaussian noise ($\mu = 0$, $\sigma = 0.05$) was added to the data so we can observe the generalisation performance of the algorithms. In order to make a fair comparison between the two learning approaches we will focus on the test error which is produced by introducing unseen data to the model and observe how it performs.

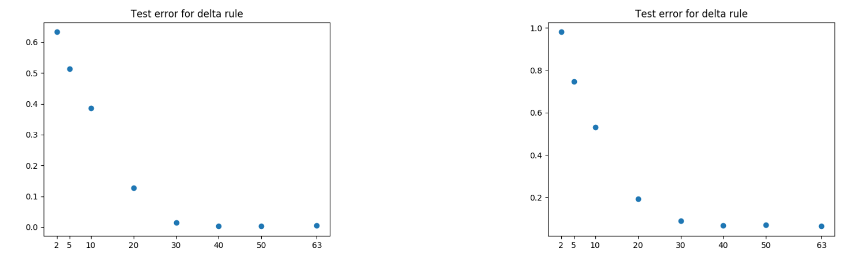


Figure 1: Comparison of the test error with different number of RBF nodes using delta rule (sin-left, square-right).

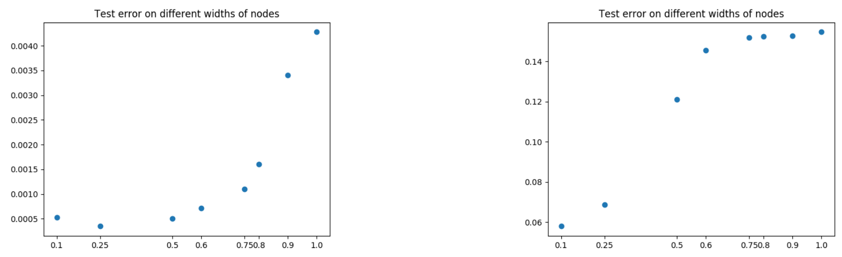
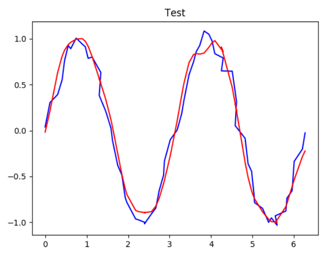


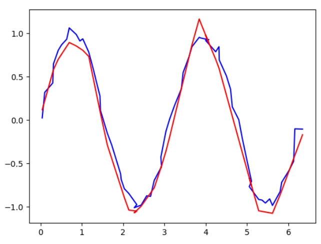
Figure 2: Comparison of the test error with different widths (σ) of nodes using delta rule (sin-left, square-right).

The delta rule it seems to be very dependent on the value of the learning rate and it is very easy for it to overshoot when the learning rate is high (even 0.1) and under-perform when too low. Also, due to the iterative process of sequential learning and the epoch updates of the weights it has a heavier computational cost (time) compared to least squares. Additionally, the least squares learning method is not dependent on a learning rate value. By increasing the width of the nodes we can see that the values of the weights are being decreased dramatically in some areas and are not as equally distributed as with small widths. This is because wider nodes overshadow their neighbours and accumulate bigger weight values. Deciding where to place the RBF nodes in the input space can also be crucial when trying to find a solution with an RBF network. Specifically, when we placed the RBF nodes randomly in the input space the test error was **0.012** whereas by evenly distributing them with equal distances between them the test error was **0.009**.

A noteworthy observation is the performance of the models with least squares and delta rule when we introduce noise to the data. The least squares model with optimized parameters has a **train error: 0.02** and a **test error: 0.15** indicating that it has overfitted to the data. Whereas the delta rule approach has a **train error: 0.05** but **test error: 0.09**. So even though the delta rule

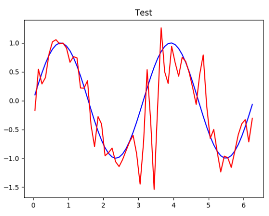


(a) RBF - Test error: 0.08



(b) MLP - Test error: 0.12

method sacrifices accuracy during training, it chooses to converge to a more general and smoother solution which eventually performs much better during testing, compared to least squares. This behaviour is also justified since the least squares method is focusing on solving a specific system of equations based on the training data, without having the chance to explore the error space, thus overfitting to the training data. This beneficial feature of delta rule compared to least squares can be observed even better when we use noisy training data but clean test data, as show in figure 3. In this case more noise was added ($\sigma = 0.1$) resulting in a **test error: 0.05** for delta rule and a **test error: 0.25** for least squares.



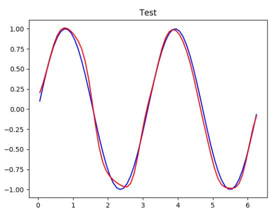
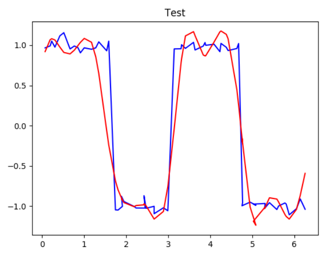


Figure 3: Function approximation of the $sin(2x)$ function with noise only during training using least squares (right) and delta rule (left).

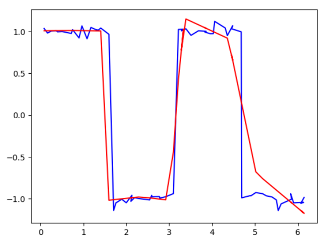
Finally, we compare two fine-tuned models on the noisy data of $sin(2x)$ and $square(2x)$, one trained using the RBF network with delta rule and batch learning and one trained with MLP (1 hidden layer) with generalised delta rule and batch learning. Both networks have the same amount of nodes. Even though both networks used batch learning and similar learning rates the RBF network was able to converge faster to an even better solution than the MLP network.

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

The best architecture found previously was $n = 63$. In table 2 we can see a comparison in the performance in the train and test data for regular RBF and RBF with Competitive Learning. We can see that regular RBF has a better



(a) RBF - Test error: 0.15



(b) MLP - Test error: 0.20

Figure 5: Comparison of performance of a MLP network and a RBF network.

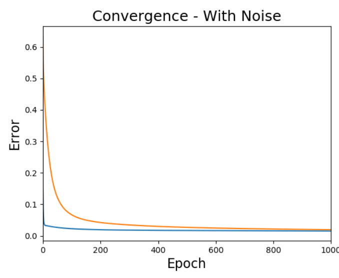
performance.

	Train without noise	Train with noise
Normal	Train: 0.02266 Test:0.00096	Train:0.0208 Test:0.00098
CL	Train:0.05441 Test:0.37454	Train:0.03631 Test:0.35006

Table 2

Figures 6a shows the evolution of the error for RBF with and without Competitive Learning (CL). We can see that the convergence for CL is slower, but the final error is smaller. In Figure 6b is possible to see the distribution of the weights for CL, this time, weights are not uniformly distributed as previously.

(a)



(b)

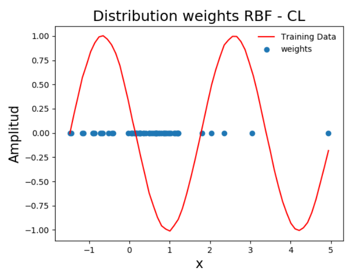


Figure 6: (a) Evolution of error for RBF (blue) and RBF with Competitive Learning (CL). (b) Distribution of nodes for RBF with CL.

As a variant, we can choose more than 1 winner. This time, we picked 3 winners for each iteration in CL. In a vanilla version (1 winner) the train and test error were 0.00436 and 0.36884 respectively. In the case of CL with 3 winners, we got 0.27455 and 0.25865 respectively. The trained error increased but the test error is better, with this we obtained a better generalisation.

3.4 Topological ordering of animal species

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

Figure 7 shows the results of the ordering of animal species using the Self Organising Maps (SOM). This result was achieved with a step size of 0.2, 20 epochs and a 1-dimensional SOM grid of 100 nodes.

One can see that the ordering of the animals is similar to the way we would intuitively order them manually as well. For example, it can be seen that the water birds 'pelikan', 'duck' and 'penguin' are mapped close together. Moreover, in general the animals which are bigger in size are mapped more below and the animals which are smaller are mapped more on top.

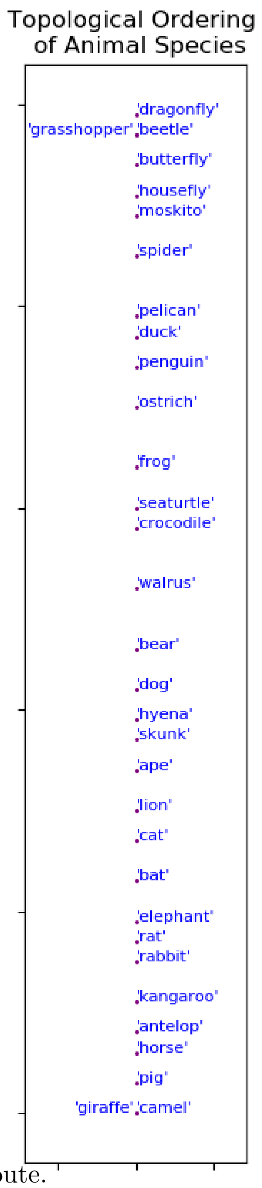


Figure 7: Animal Order using SOM

4.1 Cyclic tour

In this part of the assignment the SOM algorithm was used to find a cyclic tour around 10 different cities. The results are shown in Figure 8. This figure shows that the cyclic tour found by the algorithm matches the shortest cyclic tour that is possible.

This result was achieved with a step size of 0.2, 50 epochs and a one-dimensional SOM grid of a number of nodes matching the number of cities (10 for this example).

The important thing for the cyclic tour is that the neighbourhood is circular. This means that if the neighbouring nodes of the winner is updated, then the first node is assumed to be a neighbour of the last node and vice versa. As a result, the SOM acts as an 'elastic band', trying to minimize the total travelled distance while passing all different cities in a circular route.

4.2 Clustering with SOM

Figure 9 displays the result for clustering using SOM. The three figures show the same distribution of the different parliament members in a two dimensional

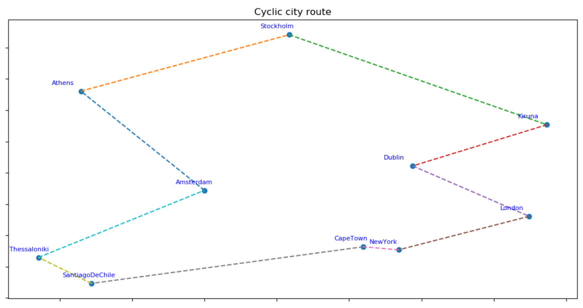
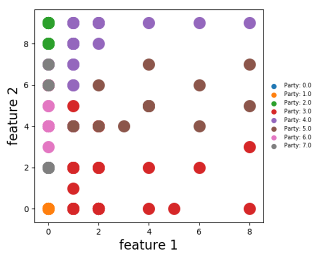


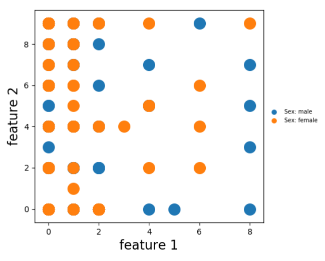
Figure 8: Cyclic route between 10 cities which was found by the SOM algorithm

space, but with a third information. Figure 9a shows how the votes have a relation with the party of each member. The clusters show how members of a party have similar behaviour. Figure 9b shows the relation with the sex of the parlamentarian. Females tend to cluster more than males. In figure 9c we can see that the votes nothing have to do with the district, there is more distribution.

(a)



(b)



(c)

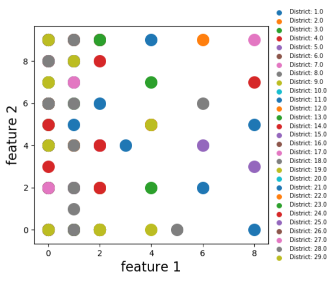


Figure 9: (a) Votes respect the party. (b) Votes respect the sex. (c) Votes respect the district.

5 Final remarks

In this assignment we majorly improved our understanding on RBF networks, competitive learning and the SOM algorithm. It was useful to see how SOM can be applied to so many seemingly different things.

Finally, we are still wondering about the differences between SOM and PCA, because besides that one is a non-linear method and the other is linear, it would be interesting to know which is more fitting in different contexts.

Moreover, this assignment was also very fun. Especially the tasks about ordering the animal species and looking into the features of votes by the Swedish Parliament was nice to do.