Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Alex Norlin, Athanasios Charisoudis and Giovanni Castellano

February 15, 2022

1 Main objectives and scope of the assignment

Our major goals in the assignment were to understand how RBF networks work, approximate functions using RBF networks, understand the differences between the least squares methods and the delta rule, study how noise affects the learning, understand how SOMs work and to cluster data with a SOM.

2 Methods

We decided to use the programming language Python to implement and test the two algorithms. Thanks to the library Numpy we could easily perform mathematical operations efficiently. Moreover, we used matplotlib to plot results.

3 Part I: RBFs and Competitive Learning

3.1 Function approximation with RBF networks

To begin, we approximated the Sine and the Square functions using the least square training method. For both the functions the final mean absolute error (MAE) changes considerably as we increase the number of hidden nodes. In particular, we noticed that for the Sine function the error decreases smoothly, for the square function instead, the error decreases too but with some oscillations. Moreover, the smallest error that we obtained for the first function is much lower (the Square function is harder to approximate). Analysing the results, we observed that for the sine function, we need respectively 10, 13 and 27 nodes to lower the error below 0.1, 0.01 and 0.001. For the square function instead, we cannot get an error below 0.09 (38 nodes); however, it is possible to low such error to zero mapping positive output of the network to +1 and negative ones to -1. This can be very useful when we know that our function can assume only two values (such as a binary digital signal). Subsequently, we studied how the learning rate and the width of the RBFs affect the delta learning on the noisy data (Figure 1). We concluded that the network was very sensible to the learning rate; in fact we get an unstable behavior with learning rates higher than 0.03. Regarding the width, we can observe that also this parameter affects the speed of convergence; in particular, when it is too high or too low the speed decreases and the error could also converge to an higher value. Afterwards,

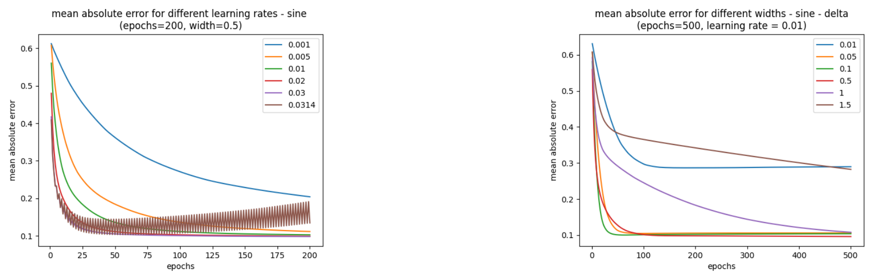

Figure 1: Delta learning curve for different parameters (a) Learning rate (b) Width

we compared the optimal RBF model with a two-layers-MLP with the same number of hidden nodes; In particular, we trained both the networks using batch learning. Figure 2 illustrates the mean square errors; we can observe that the curve is flat for the RBF network, this is because using least squares we can train the network in one epoch. The MLP network instead needs much more time, moreover, the curve is very unstable and depends on the initial conditions. It is important to remember that we could get better results with MLP networks changing the number of hidden nodes. Finally, we studied how the positions of

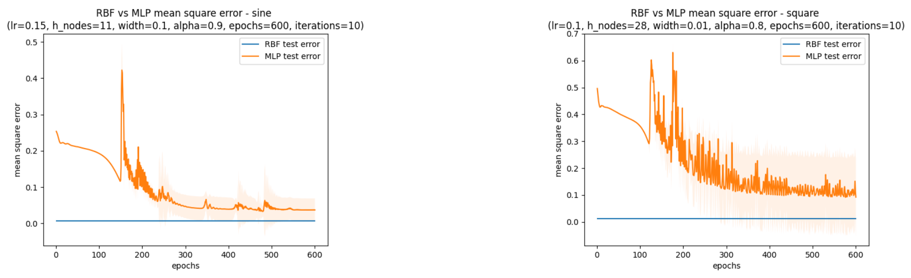

Figure 2: Comparison between RBF network and two-layers-MLP

the RBFs affect the learning; We found that the best solution is to place the centers evenly spaced in the domain of the function (MAE: 0.086). Another approach could be to randomly draw the positions. Starting from a uniform distribution with sample space between $[0; 2\pi]$ and decreasing it to $[0; \pi/3]$ we noticed that the performance decrease as such interval decreases. In particular the mean absolute error was 0.15 for the largest interval ($[0; 2\pi]$) and 0.24 for the smallest one ($[0; \pi/3]$). These errors regard the approximation of the noisy sine using 10 hidden nodes.

3.2 Competitive learning for RBF unit initialisation

In the previous section, we studied the behaviour of simple RBF NNs, with regards to RBF Gaussian kernel spreads and output weights learning rate. Based on the conclusions drawn there, here we fix the learning rate for Delta Rule to 0.001 and Gaussian σ to 0.1. In what follows, we first experiment on how Competitive Learning (CL) of the RBF means affects the networks performance, then we propose a naive strategy to alleviate the dead RBF units issue and finally we apply RBF NNs in 2D function approximation.

In figures 3 and 4, we give the effect on CL after uniform centers initialization (in the range $[-\pi/2, 5\pi/2]$). We used 11 RBF nodes for the non-noisy sine data and 28 for the sine plus noise (with $\sigma = 0.1$). As can be seen, while CL does helps with test-set performance, most of the RBF units are "dead" (i.e. they do not learn and remain where they were initialized at. To alleviate this situation, we propose the usage of more than one winning nodes. In particular, we found the **k minimum distances and update the all the corresponding nodes but with a linearly decaying learning rate**. To test our modification, we test how the number of winning nodes affects the number of dead units (left) and the final test-set MAE (right), figure 5. What can be seen for these plots, is that using more winning nudes can eventually completely eliminate dead units and also improve the generalization capabilities. Best value for number of winners was found to be 8, and this was increased the model tended to behave worse. Finally, we test our RBF NN and our modified CL to a 2D function approximation. Given the angle and velocity, our network is trained to predict the corresponding maximum height and distance of a ballistic trajectory. After some experiments, we set the number of hidden (RBF) nodes to 20. Subsequently, we searched for the optimum value of the number of winners, which was found to be 7.

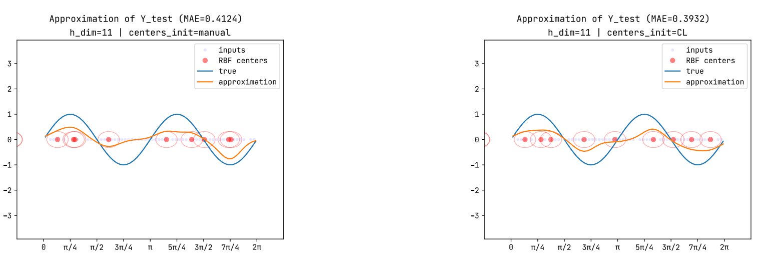

Figure 3: Random centers vs. CL on clean sine data ($h_{dim} = 11$).

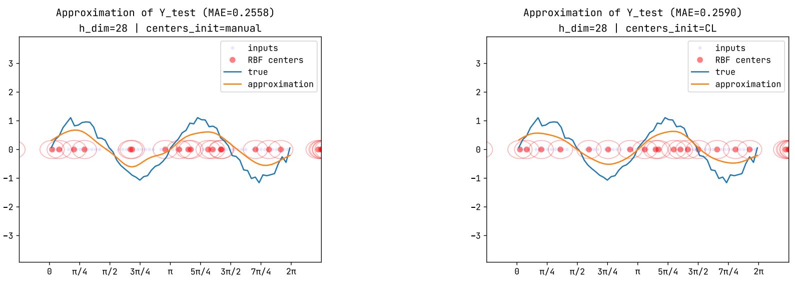

Figure 4: Test-set MAE vs number of winning nodes (clean sine, $h_{dim} = 28$).

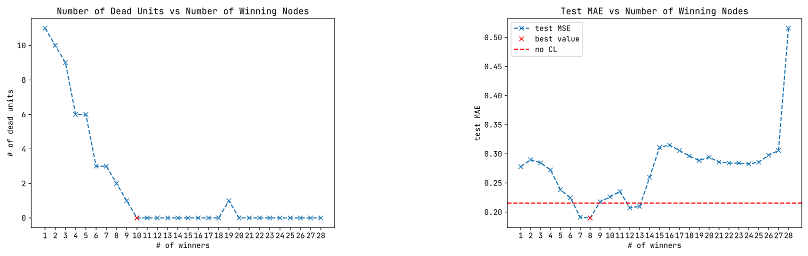

Figure 5: Number of Dead Units & test-set MAE vs. number of winning nodes (clean sine, $h_{dim} = 28$).

For that number of winning nodes in CL, we trained both the RBF kernel means and the output weights to approximate the 2D function. Results are given in figure (see presentation), with the Gaussian $\sigma = 0.1$ and $lr = 0.001$. The number of epochs was held fixed to 100. As can be seen, the model generalizes well, at least of low values on inputs (which is a bit easier to achieve).

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

4.1 Topological ordering of animal species

The result of one trial with the SOM algorithm produced the following list: *camel, giraffe, pig, horse, antelop*[1], *kangaroo, elephant, rabbit, bat, rat, cat, lion, skunk, ape, hyena, bear, dog, walrus, crocodile, seaturtle*[1], *frog, ostrich, penguin, duck, pelican, spider, housefly, moskito*[1], *butterfly, beetle, dragonfly, grasshopper*. As one can see the SOM algorithm has placed all the insects in one group, all the birds in one group, all the creatures closely tied to water

[1]Spelled the same way here as in the document with names provided to us.

in one group and the rest of the animals in one larger group. In the group with animals ranging from camel to skunk there are still similarities between the animals directly next to each other like cat and lion. Depending on exactly what the original data represented one might find a more logical ordering of these animals.

4.2 Cyclic tour

The solution to the cyclic tour problem can be seen in figure 6. There, the blue points are cities and the red crosses and path represent the weights of the SOM and the proposed optimal path between the cities. The SOM algorithm (almost) finds a cyclic tour between the cities. It didn't find a clear solution since one node ended up with two of the cities. However to us humans, we can still see what path we should take to to find the shortest path between all the cities. The solution obtained by the SOM algorithm looks optimal or near optimal to us at least.

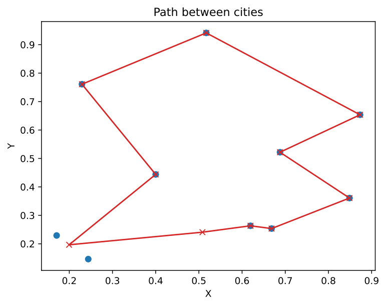

Figure 6: Cyclic tour plot.

4.3 Clustering with SOM

The result of the clustering of the parties can be seen in figure 7. In that figure you can see a clear split down the middle between the two major coalitions in the Riksdag, *Alliansen* and *the Red-Greens*. It's harder to judge what role the vertical dimension plays. One can note that the most popular parties are near the top while the more unpopular are near the bottom. Perhaps that dimension depends on how more extreme in their beliefs the parties are. It's hard to say with certainty.

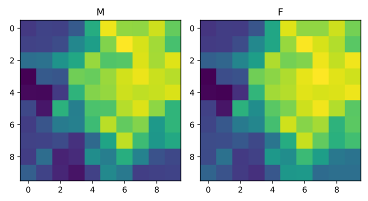

Figure 8: The activation for each node for all MPs depending on gender. Since most MPs vote S the heat map is biased but you can make out very slight differences (yellow corresponds to higher activation; dark purple to a lower).

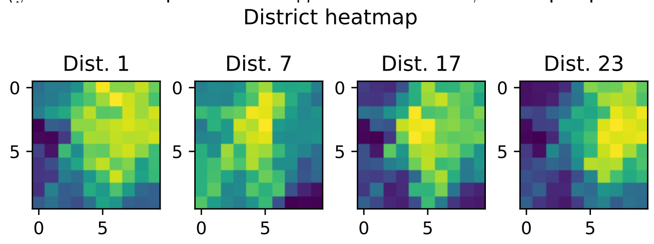

Figure 9: The activation for each node for all MPs in each district. Since most MPs vote S the heat map is biased but it can still be seen that there's a clear difference between some districts. Here yellow corresponds to a higher activation and dark purple t

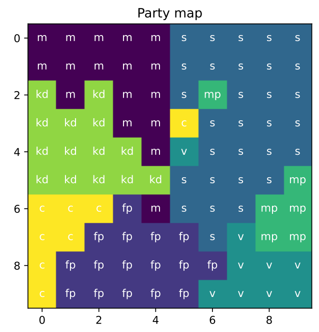

Figure 7: The party with the highest activation at each node. There are some peculiarities in the neighborhood of (3,4) but ignoring those you have a clear clustering of the parties of Sweden. No correlation between colors here.

Moving on to see how the sex and which district they come from affect the result you can from figure 8 see that their sex doesn't change their voting patterns that much. But from figure 9 it's obvious that the district votes differently, on average. Due to how this heat map was constructed it's important to mention that it's biased towards S because they had the most MPs. But you can still see that the node activation is almost the same between the sexes while it differs between districts.

5 Final remarks

It was fun working with RBF things and SOM. Even though in the case with the SOM knowing how good the result was proved troublesome in some cases. There was also a question on how to plot the result of the map.