Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Isabella Rositi, Gustav Thorén and Nicolas Guy Albert Wittmann

15 Febraury, 2023

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to build and train of an RBF network for classification and regression purposes and analyse different initialization methods
- to be able to implement different components in the SOM algorithm and analyse the role of the neighbourhood in the self-organisation

We firstly construct and analyse an RBF network in order to solve function approximation problems using a CL method to automate the initialization of RBF units. Later, we implement the SOM method to work for three different problems in order to find a low-dimensional representation of higher-dimensional data.

2 Methods

The code was implemented in Python with the use of the *Numpy* and *Matplotlib* libraries.

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

We implemented RBF Networks on two different clean and noisy functions: $sin(2x)$ and $square(2x)$ sampled in the interval $[0, 2\pi]$.

3.1 Function Approximation with RBF Networks

We ran the algorithm using both a random (averaged over 50 runs) and an equal-spaced initialization of the RBF units. Moreover, the algorithm was ran for a varying number of units: 4 to 15 for $sin(2x)$ and 10 to 60 for $square(2x)$. An absolute residual error was calculated, which is visible in Figure 1.

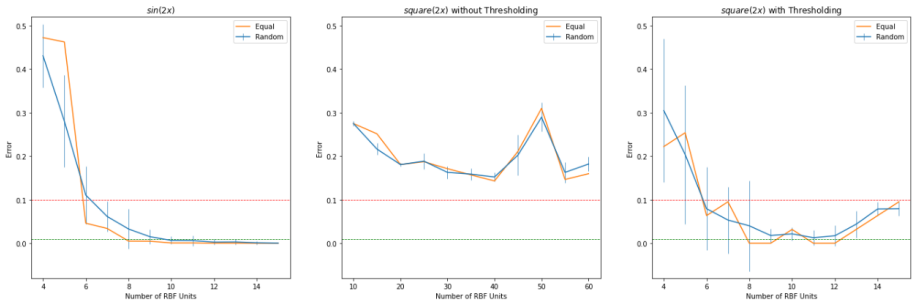

Figure 1: Plots of errors as the number of RBF units increases. Both random and equal-spaced initializations are employed.

As the number of RBF increases, the error overall decreases. Furthermore, there isn't a big difference between the two initialization methods, however the equal-spaced approach produces smaller errors for lower numbers of RBF units, an overall slightly better result. We can observe however, that for $square(2x)$ the error is never lower than 0.1, so we decided to threshold the predictions: if the prediction is ≥ 0, then it's transformed into 1, otherwise it becomes -1. This decreases the error considerably, as can be observed in the last plot of Figure 1 and it can be very useful especially for classification tasks.

3.2 Function Approximation with RBF Networks on Noisy Data

We then added some Gaussian noise to the functions and as expected, the errors provided are clearly higher than the errors provided by the clean data set and the initialization techniques have the same impact on these data sets that they have on the clean data. Moreover, we observed the behaviour of the error as the variance of the noise changed. The two functions presented the same trend, and in Figure 2 the results for $sin(2x)$ can be observed, for both batch and online training. When trained with batch learning, the different variances produce different errors mostly for lower numbers of units, and as the number increases both the error and the difference decrease. However for online training, the different variances have very different effects on the predicting power of the model: the lower variances produce more accurate results, whereas the highest variances don't allow the model to make accurate predictions even as the number of units increases.

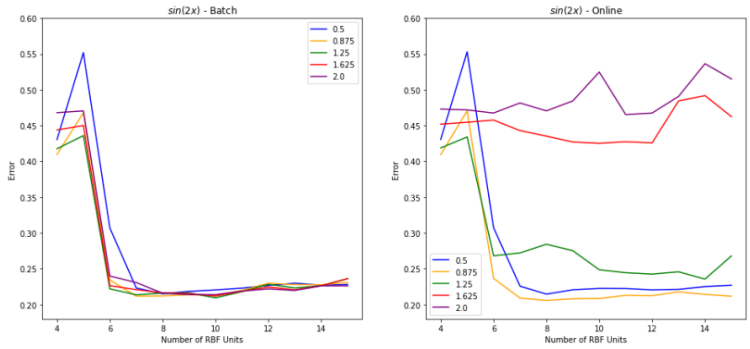

Figure 2: Error for different variances and varying number of RBF units for $sin(2x)$. Equal-spaced initialization is implemented.

Furthermore, for very high and very low learning rates (0.1 and 0.0005) the network produces incredibly high errors, whereas for the values of η in between the performance doesn't differ from one another, as it doesn't the running time.

If we were to compare the performances of a MLP with those of the RBF network, we can see that the RBFs provide a better performance for both functions (0.209 for RBFs and 0.303 for MLP for $sin(2x)$ and 0.361 versus 0.436 for $Square(2x)$) and faster running times (average of 0.027 versus 0.167 seconds).

As shown in Figure 3, the predictions of the RBFs are smoother and less subject to noise then the MLP predictions, which learn the noise of the training data and hence provide less accurate predictions on the test.

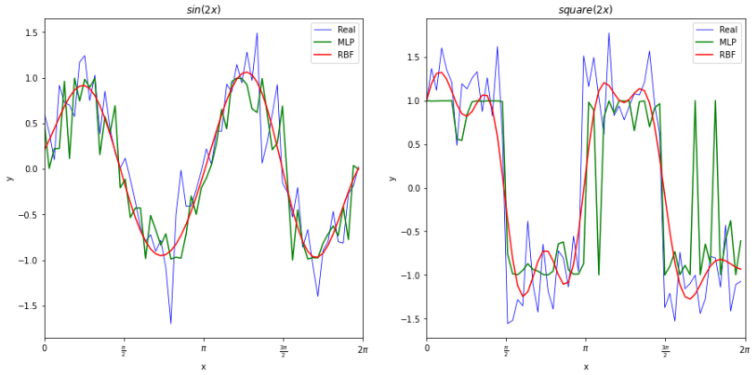

Figure 3: Predictions made by a MLP and a RBF network for both functions.

3.3 Competitive learning for RBF unit initialisation

Competitive Learning (CL) was implemented trough both a "winner takes all" and a "multiple winners" schemes. The sinusoidal data set was generated from an even distribution on the x-axis. Since the distribution is even, there are no natural clusters in the data set which the CL algorithm can find. The winner

takes all algorithm therefore spreads out the nodes so that they become roughly evenly distributed regardless of how they were initialized. As a result, the performance is essentially the same as when using equal-spaced initialization.

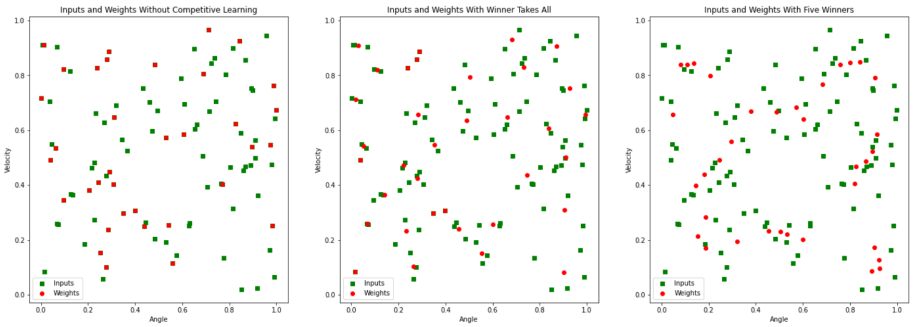

Figure 4: Shows the RBF units placements (red) and the underlying data (green) using zero, 1 and 5 winners. There were 100 data points and 35 RBF units.

Fig. 4 shows the placement of the RBFs and the distribution of the underlying two dimensional training data. The left plot shows how random initialization can fail to efficiently capture the data, there is a over representation on the bottom right and an under representation on the bottom left. The middle plot shows how a "winner takes all" scheme shifts the nodes to better capture the data distribution. Since the initialization was done on existing data points there are no true dead units although some of the nodes will never adapt, but keep their starting position forever, notably the three nodes in the top left. This problem can be avoided with multiple winners, but as can be seen in the right plot, this can lead to other problems. Using multiple winners can lead the algorithm to over represent the denser clusters and forsake the single points, which is often not the desired behaviour.

The algorithm was tested with 35 units and different amount of winners (0,1,2,3,4 and 5) in the CL algorithm. The best model (1 winner) achieved an error rate of 0.030 with a standard deviation of 0.0075. This result makes intuitive sense since the data set does not appear to have any clear clusters and as mentioned earlier the winner takes all scheme serves to even out the weights over the data set.

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

4.1 Topological ordering of animal species

The SOM network is essentially a vector quantization with connections between the elements of the vector. The connections ensure that there will be a coherency of the quantization and the network can be seen as a compression of

the original data. In this instance and in the following example we used a learning rate of 0.2 and decreased the size of the neighbour to be updated linearly.

The given data set was 32 animals with 84 unknown traits. Fitting the SOM network to the data with the neighbourhood shape of a line places a line in the 84 dimensional space which represents the animals. The output then sorts the animals into a single line based on their similarity, see Fig. 5, where we can detect some clusters, for example insects, birds amphibians and reptiles and finally mammals.

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

Figure 5: SOMs output placing animals in a line based on their similarity.

4.2 Cyclic tour

In this section, we use a circular neighbourhood in order to map spatial 2 dimensional points to a one dimensional curve (with 10 nodes). Close cities should be close in the mapping. Therefore, we can use the mapping to find a efficient trip by visiting them in the order we get them with the SOM algorithm. It does not guarantee to obtain an optimal solution, but should be quite efficient as we travel from neighbour to neighbour in the mapping to points that are supposed to be close to each others.

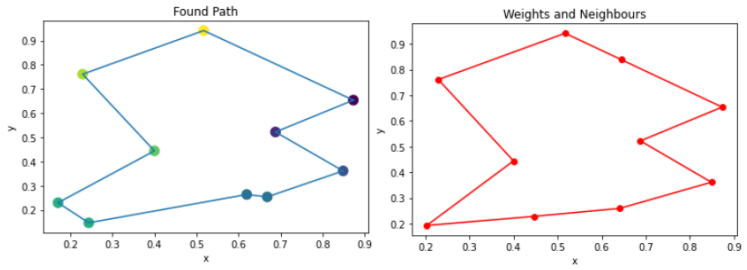

Figure 6: Shows the found path and the weights and neighbourhood of the SOM.

We see in Fig. 6 that the weight obtained after the learning correspond quite well to the actual cities, and we obtain a good travel plan. Some inefficiency might come from the fact that cities can be sent to the same node in the mapping and then we do not have ways to optimize which to visit first among them. Having multiple cities on one node is not a large problem as this can be seen as splitting the problem into multiple sub problems which are easier to solve. Another problem that can occur is that the neighbourhood gets "twisted" when trying to approximate the data which will make the path cross itself and be

inefficient. Both of these problems can be avoided by refitting until all cities are covered by one node and there are no crossing paths.

4.3 Clustering with SOM

In this section, we try to map the 349 members of the parliament in a 10 by 10 grid using their votes. For this we used a square shaped neighbourhood, and Manhattan distance.

All members of the parliament are placed on about 20 different nodes on the grid. In these representation, in order to see every different person, we added a random shift around the position of the person in the 10 by 10 grid. In Fig. 7 we can observe that members of the parliament tend to vote similarly to the other members of their party, as one would imagine. The similarity is not only among the members of the same party, but also between members of "left wing" parties and those of the "right wing" parties. On the contrary, it is quite hard to detect any strong voting pattern depending on the gender and the district they live in.

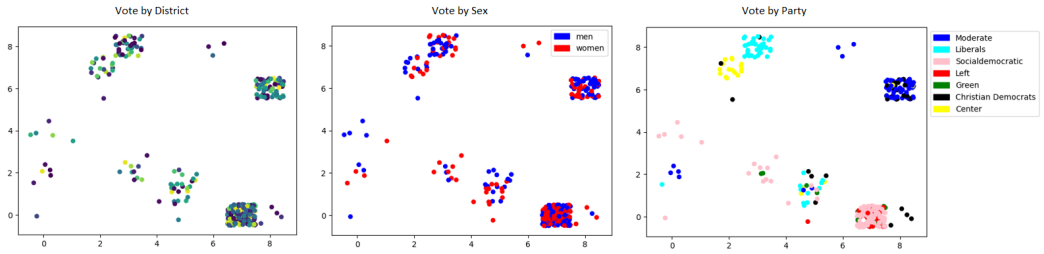

Figure 7: Visualisation of the votes clustered by District, Gender and Political Party on a 10 by 10 grid.

5 Final remarks

When dealing with RBF networks, the initialization of the weights can influence the overall prediction of the model, however if the data doesn't present clusters in the plane (as in this case since the data sets are evenly sampled), then competitive learning can improve the performance implementing the "winner-takes-all" scheme without resulting in dead units. Moreover, when compared to a two-layer perceptron with the same number of hidden nodes as RBF units, the RBF network was found to be more accurate in the approximation and being less susceptible to noise.

SOMs can be used for very different kind of problems. In every instance, it is somewhat sensitive to initialisation of weights. We also need to monitor the size of the neighbourhood throughout the learning process in order to capture the nuances in the data both at a large and at a small scale.