Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 2

Radial basis functions, competitive learning and self-organisation

Adrian Campoy Rodriguez, Nimara Doumitrou-Daniil and Gustavo T. D. Beck

February 12, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were: (1) to get familiar with the RBF network structure and perform training of an RBF network for classification and regression; (2) to be able to know and apply different initialization methods and understand the importance of variable initialization; (3) to understand the concepts of vector quantisation, SOMs, and how to implement and apply them.

The first part of this assignment involved building and studying RBF networks in one and two dimensions function approximations both with and without noise. The second part of the assignment was mainly focused on SOMs and their application over different data sets: classification of animals, finding a circular tour over a set of cities and finally creating a map with data regarding Sweden politicians. The main goal in these three cases was to find a low dimensional representation of the high dimensional data provided.

2 Methods

During this project the programming language Python 3 was used in three different IDEs such as Jupyter Notebook, VS Code and Spyder. Some particular toolboxes were used for specific tasks. For instance, SciKit Learn was used as an additional support to develop different metrics applied along the project. Finally, the GitHub repository was used to store and manage different versions and pieces of code.

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

RBF networks offer an interesting alternative to the traditional Artificial Neural Networks that we observed in the previous lab. They are shallow networks that nonetheless manage to capture complicated and intricate functions and decision boundaries, by first transforming their $\mathbf{R}^m$ input feature vectors utilizing N RBF function $f_i(x) = e^{\frac{-||x-w_i||^2}{2\sigma_i^2}}$. At its core, it attempts to project our data in a sparser high dimensional space, where classification or regression is more manageable.

There are two main approaches for this problem, **least squares** (batch mode) and **delta rule** (sequential). In principle, least squares is inadequate for online learning and lacks robustness, since it requires the inversion of an often numerically unstable matrix $\Phi = [f_i(x_j)]_{i\in[N],\ j\in[n],\ m\leq N\leq n}$. We will analyze and compare the robustness and efficacy of these two alternatives in the following sections.

3.1 Function approximation with RBF networks

During the first section of the assignment, we focused on working with supervised learning methods. In particular, we applied a batch learning algorithm using least squares to compute the weights. With least squares, we aim at minimizing the total error $= \|\mathbf{\Phi w} - \mathbf{f}\|^2$ where $\mathbf{f}$ is the true underlying function output (the goal). This is done by applying the formula $\mathbf{\Phi}^\top \mathbf{\Phi w} = \mathbf{\Phi}^\top \mathbf{f}$ and solving it for $\mathbf{w}$. The goal was to approximate the functions $sin(2x)$ and $square(2x)$. The results were the following:

Number of Nodes $sin(2x)$	Absolute Residual Error $sin(2x)$	Number of Nodes $square(2x)$	Absolute Residual Error $square(2x)$
6	0.07	60	0.0811
8	0.009	-	-
11	0.0009	-	-

Table 1: Progression of Absolute Residual Error with respect to number of nodes (for $sin(2x)$, $\sigma = 1$ in the RBF nodes. For $square(2x)$ $\sigma = 0.05$ in the RBF nodes.)

From table 1 we can observe that it is quite easy to reach a low level of error for the sinusoidal wave with a few nodes and the function is very well approximated (see figure 1). On the other hand, for the square wave it was not possible to obtain an error below 0.01 even though the number of nodes was significantly higher in this case. This can be due to the fact that for RBF nodes it is quite difficult to grasp the required information to represent the sharp changes in the square wave. It has some fluctuations in its approximation that forces the error to be higher than in other functions, such as the sinusoidal, which is more easily approximated by the RBF network thanks to its curvature. If more data points are used for training, the performance for the RBF network over the square wave slightly improves however we are still not able to reach Absolute Residual Error values below 0.02 (with 6000 training data points and 200 nodes).
Furthermore, in order to obtain 0 error in the square wave, it is possible to add an output node with a step function as activation function that sets values higher than 0 to 1 and values below 0 to -1. This can be interesting in the fields of signal processing, transmission or for classification.

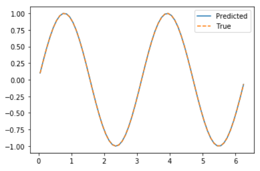


Figure 1: Sinusoidal Approximation with 11 nodes (RBF $\sigma = 1$)

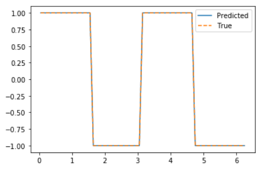


Figure 2: Square Approximation with 11 nodes (RBF $\sigma = 1$) and extra node with step function.

The second part of this task required to apply an RBF network for function approximation ($sin(2x)$) in the presence of Gaussian Noise in the input data ($\mu = 0, \sigma = 0.1$). The effect of the number of nodes and RBF width was studied both applying least squares and delta rule. The results can be seen in figures 3 to 6.

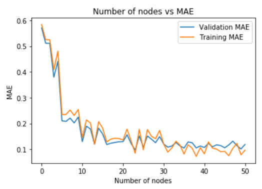


Figure 3: $\eta = 0.001$, $width = 1$, $epochs = 1000$ (Delta Rule).

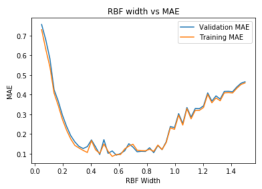


Figure 4: $\eta = 0.001$, $nodes = 10$, $epochs = 1000$ (Delta Rule).

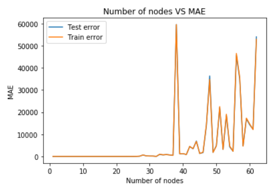


Figure 5: $width = 0.6$ (Least Squares).

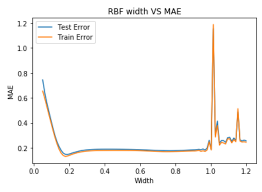


Figure 6: $nodes = 15$ (Least Squares).

From the previous figures, it can be observed that the error decreases with a higher number of nodes as they are more capable of fitting the training data. It can also be observed that around 35 nodes, in figure 3, the validation error starts to slightly increase whereas the training error continues decreasing. This is caused by overfitting. It is expected that, by running the algorithm with a larger number of nodes, the overfitting would worsen the results even more. Moreover, the RBF width produces higher error for very small width values (0.001 to 0.2) or very large width values (> 0.9). The reason behind it is that a small width will produce overfitting and will also ”learn” the noise that is introduced in the data. On the other hand, a big width will produce a simpler model (high bias), consequently the error increases. Therefore, we expect that the training error would be very low with low RBF values. However, it should be taken into account that the plot on figure 4 was taken with less nodes than data points (10 nodes for 63 data points). If we have the number of nodes equal to the number of data points, we will have a very low training error for small values of RBF width.

Regarding the learning rate and how it affects convergence, we could observe that the larger the η a smaller number of epochs was required for convergence (see figures 7 to figure 9). The reason is that a small learning rate will take ”shorter” steps and thus more steps to reach a minimum whereas a larger learning rate will reach it in fewer steps. Note in figures 7, 8 and 9 that by epoch number 200 the error is around 6, 0.03 and 0.012 for $\eta = 0.0001, 0.005$ and 0.01 respectively. Moreover, it can be noticed that a larger learning rate implies that the error will fluctuate more around the minimum as it can be observed in figure 9.

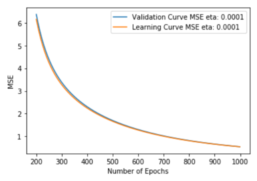


Figure 7: $width = 0.7$, $\eta = 0.0001$ and 20 nodes.

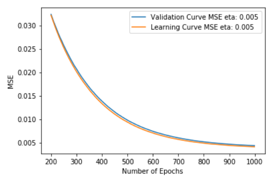


Figure 8: $width = 0.7$, $\eta = 0.005$ and 20 nodes.

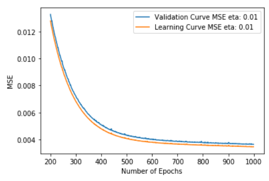
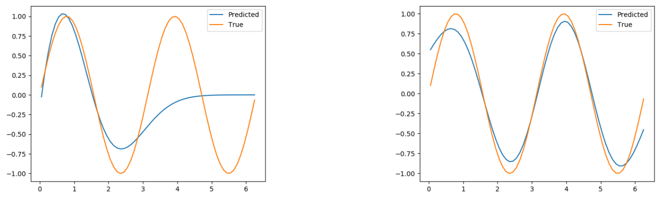


Figure 9: $width = 0.7$, $\eta = 0.01$ and 20 nodes.

It is worth mentioning also the importance of the initial positioning of the RBF nodes. The position of their mean μ_i should be uniformly distributed over the input space in order to approximate and grasp as much information as possible. Failing to do so could cause that we find cases in which we have too many nodes in some areas and to few in others, losing information from the inputs, which can be seen in figure 10.

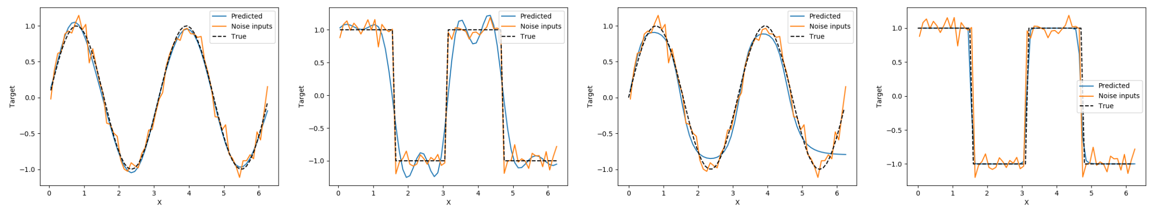


(a) μ initialized as $\mathcal{N}(0, 1)$.

(b) μ equally distributed through X.

Figure 10: Impact of different initialization methods of μ.

In terms of function approximation, the last task assigned for this topic refers to comparing our optimal RBF network trained in batch mode ($width = 0.93$ and $nodes = 14$) with a single-hidden-layer perceptron trained with back-propagation and the same number of nodes. This analysis can be seen in figure 11.



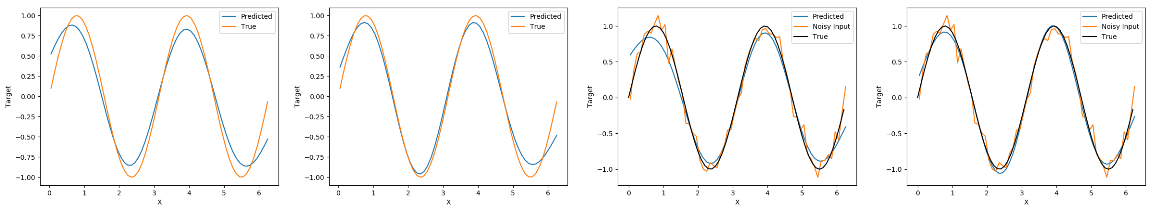
(a) RBF network batch learning *sin(2x)*. (b) RBF network batch learning *square(2x)*. (c) Single perceptron learning *sin(2x)*. (d) Single perceptron learning *square(2x)*.

Figure 11: Comparison between optimal RBF network trained in batch mode and a single-hidden-layer perceptron trained with back-propagation for *sin(2x)* and *square(2x)*.

Although, both algorithms managed to represent both curves some insights can be taken from these analysis. For the *sin(2x)*, RBF network performed better than the perceptron learning, since it represented better the true function and the underlying input with noise. However, in the case of the *square(2x)*, it's interesting to notice that the RBF network is weak to noise, probably due to numerical instability of the inverse transformation matrix, while the perceptron filtered better the noise and represented well the true function. Despite from these analysis, it's important to state that RBF network trained in batch takes much less time to be computed, since the perceptron needs many epochs to achieve good representations.

3.2 Competitive learning for RBF unit initialisation

In the previous sections, we initialized our cluster centers randomly in the range of our training examples. This approach, though general, can prove problematic in certain cases, as we are not utilizing all the information provided to us by our data. For example, some centers might lie far away from data, or can get consistently overshadowed by more relevant clusters (dead cells). To deal with this, one can first fit his centers to the data, by a traditional unsupervised procedure, such as EM or k-means. In this section, we will examine one such approach, called Competitive Learning, where we first iterate through the data and update the center of the nearest (winner) center w_i. In order to deal with dead cells, we will utilize **leaky learning** which also updates the centers of the losers (w_j, $j \neq i$) by a smaller coefficient $\eta_{losers} = \eta_{winner}^2$. The result of both approaches can be seen in the following charts:

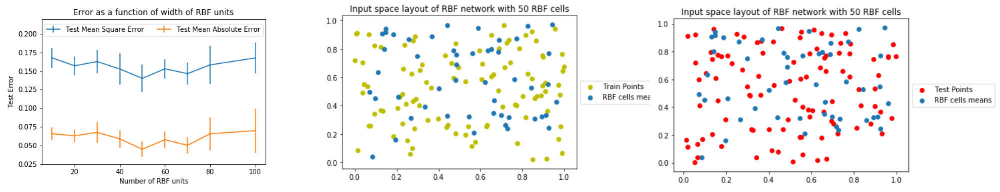


(a) Normal competitive learning without noise. (b) Leaky competitive learning without noise. (c) Normal competitive learning with noise. (d) Leaky competitive learning with noise.

Figure 12: Comparison between normal and leaky learning for competitive learning.

One advantage of these approaches is that one doesn't have to know how the independent variable is distributed to initialize the centers of the RBFs, since the competitive learning tries to lead the RBF's centers towards the correct positions. We can see that the normal competitive learning works well, similarly to 10b, but it still suffers to dead units. Then, leaky learning reduces this problem and the performance of the algorithm is better.

An application of this method can be seen in the subsequent analysis, which accounts a noisy data $\mathbb{R}^2$ (angle and velocity) from ballistical experiments. Figure 13 shows how close the competitive learning method managed to map the noisy input data through the RBF centers (cells means). In addition, it's possible to observe in figure 13a that after 50 RBF units the model starts to overfit and becomes too specific to the training set (generalization error increases).



(a) RBF unit size effect on Test

(b) Input train.

(c) Input space test.

Figure 13: The optimal number of RBF units are 50 (see left). Figure 13b showcases the input space for 50 RBF units, which manages to adequately spread among the region of our training data (note how we have more units in high density areas).

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

4.1 Topological ordering of animal species

The aim of this task was to assign a natural order to animals characterized by a large number of attributes (84 in this case). We do this by letting the SOM create a topological mapping considering the high dimensional features. The animals can intuitively be separated into 5 different clusters (arthropods, birds, amphibians, reptiles and mammals) depending on the type of animal they are. With a step size of 0.2, a initial neighbourhood of 15 nodes by each side of the winner (i.e. a total neighbourhood of 30 nodes) and 20 epochs, the number of neighbours was reduced on 1 (from each side of the winner) on each epoch. The result was the following:

Animal	beetle	grasshopper	dragonfly	butterfly	moskito	housefly	spider
Index	0	0	1	2	4	5	9
Animal	duck	pelican	penguin	ostrich	frog	seaturtle	crocodile
Index	13	14	16	18	21	24	26
Animal	walrus	dog	hyena	bear	lion	cat	ape
Index	29	33	35	36	38	39	41
Animal	skunk	kangaroo	elephant	bat	rabbit	rat	horse
Index	44	47	50	52	57	60	63
Animal	pig	camel	giraffe	antelop			
Index	66	69	71	75			

Table 2: Animals sorted in a 100 size nodes.

It can be observed how the SOM has organised the animals in these five groups, locating closer in the nodes those belonging to the same cluster. It is also worth noticing how some animals with similar characteristics have been located together. For instance, it can be observed how *rabbit* and *rat* are close within the mammal cluster as well as *lion* and *cat*.

4.2 Cyclic tour

In the Cyclic tour problem, we were asked to find a path between **N** cities, such that we visit **each one exactly once**, before looping back to our original starting point. As such, this can be seen as a simpler variant of the traveling salesman person (TSP), where we simply wish to find such path, ignoring its cost. One way of tackling this problem in the SOM framework is to attempt to capture this cycle, utilizing a circular topology in the output space, where our weights w_i are linked in a circular pattern ($E = \{(w_i, w_{i+1}) | i = \{1, ..., N\}\} \cup \{(w_1, w_N)\}$). After training for a suitable amount of epochs, our centers w_i will be located near the training samples-cities, and their chain link will showcase the cyclic tour.

4.3 Clustering with SOM

In this final section we will examine the application of SOM in representing high (31) dimensional data in two dimensions, by examining the 10 by 10 grid of clusters in the output space. For

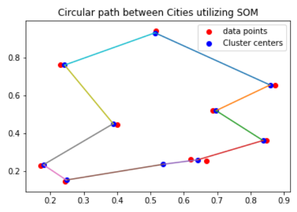


Figure 14: Cyclic Tour solution. As we can see, SOM is able to adequately link our cities. Even though one of the clusters (in between 0.5 and 0.6) is a dead cell, the path is clearly visible.

this representation, we considered 10 × 10 clusters living in the 31 dimensional space of our data (votes), which were linked in grid like structured where every $W_{i,j}$ was linked with $W_{i',j'}$, $|i' - i| + |j' - j| \leq 1$ (each cluster had at most 4 neighbours). By training our SOM, on the MP's votes, we wish to extract a meaningful 2D interpretation and visualization of their distribution. We thus visualized the distribution over biological genders, political parties and districts in the following graphs:

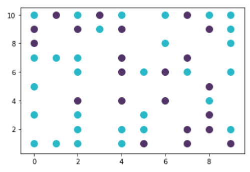


Figure 15: Distribution among genders of MP's votes.

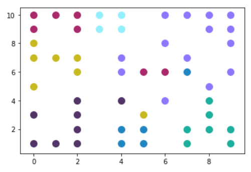


Figure 16: Distribution among parties of MP's votes.

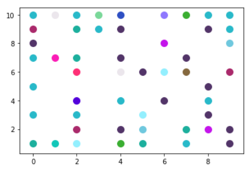


Figure 17: Distribution among districts of MP's votes.

As expected, the votes seem to be evenly split among districts and genders, while we can clearly see clusters forming among members of the same party. Furthermore, we see that certain parties differentiate themselves among the y axis, while others among the x, indicating that the second dimension in the representation was indeed needed.

5 Final remarks

The assignment helped us build a more robust theoretical foundation and bolstered our understanding of RBF networks, competitive learning and SOMs. Similarly to the first assignment, it served as a good exercise to become familiar with the scientific and concise format of reports that is usually requested in the workplace. In retrospect, we would have preferred if equal emphasis was given to SOM, as we feel that it was examined superficially. For instance, we could have examined noisy data sets or were presented with a more systematic framework of approaching such problems (e.g. analysing and comparing the efficacy of modern variations of Competitive learning). In comparison to the previous assignment, this one proposed analysis upon data sets closer to real-life problems. This helped us to comprehend some of the challenges that we may face in our future. In addition, it provided a better sensitivity on how to interpret our outcomes and how they relate to reality.