MSc Machine Learning

DD2437 Artificial Neural Networks and Deep Architectures

Lab2 Report: Radial basis functions, competitive learning and self-organisation

Authors:

Alexandros Ferles
ferles@kth.se

Pantelis Myriokefalitakis
pmy@kth.se

George Zervakis
zervakis@kth.se

Professor:

Pawel Herman
Assistant Professor

Contents

2 Background

2.1 Radial-basis function networks

2.1.1 Computing the weight matrix

What is the lower bound for the number of training examples, N ?

The number of training examples N must not be lower than the number of hidden units n[4]. As a result, its lower bound value is equal to n.

What happens with the error if $N = n$? Why?

When the size of the training samples N is equal to the size of the hidden units n, we are dealing with *exact interpolation*[4]: the predicted values at the points for which the data values are known will be the known values. This result in minimum training error in comparison with other network sizes, as the hidden units approximate optimally the input space[7]. In fact, it is strongly advised to have a considerably larger number of input points, compared to the number of RBF nodes in the hidden layer[1].

Under what conditions, if any, does (4) have a solution in this case?

In order to find a compact solution for (4), $\mathbf{\Phi}$ has to be square ($N = n$). Thus, taking $\mathbf{\Phi}|\mathbf{f}$ and applying Gaussian Elimination:

$$\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) & f_1 \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) & f_2 \\ \hdotsfor{5} \\ \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_n(x_N) & f_N \end{bmatrix} \rightarrow \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) & f_1 \\ 0 & \widehat{\phi_2}(x_2) & \dots & \widehat{\phi_n}(x_2) & \widehat{f_2} \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \dots & \widehat{\phi_n}(x_N) & \widehat{f_N} \end{bmatrix}$$

Since $\phi_i(x_k) \neq 0, \forall i, \forall k$ we can calculate $w_n = \frac{\widehat{f_N}}{\widehat{\phi_n}(x_N)}$ and use to find w_{n-1} and so on till we find all the values for the weights. Hence, the solution for $\mathbf{w}$ will be unique.

Another case, is when $n > N$. Here the system will have infinite solutions, since some weights will act as “free” parameters. The solution would then be a familly of solutions and, in fact, it will be a linear combination of those parameters with some vectors.

During training we use an error measure defined over the training examples. Is it good to use this measure when evaluating the performance of the network? Explain!

As in most Machine Learning techniques, the neural network learns to adapt into the samples from the input space during the training process. Thus, we expect a very good performance in predicting the true values of the samples after a few steps in the training process. As in most cases, in order to evaluate the performance of the trained neural network, a validation set must be used.

3 Assignment - Part I

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

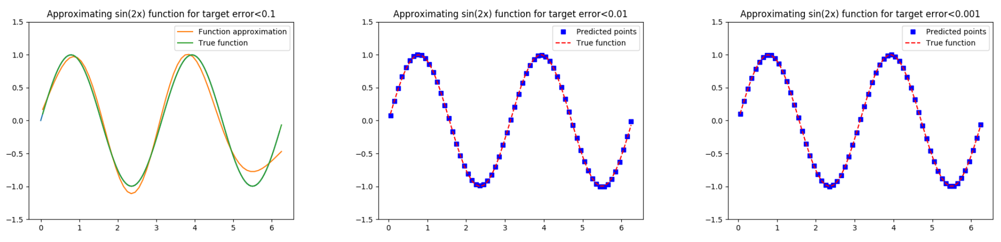
Try to vary the number of units to get the absolute residual error below 0.1, 0.01 and 0.001 in the residual value (absolute residual error is understood as the average absolute difference between the network outputs and the desirable target values). Please discuss the results, how many units are needed for the aforementioned error thresholds?

We employ random datapoints from the input space to be used as centers of the RBFs in the hidden layer. A fixed value of $\frac{d}{\sqrt{2M}}$, where d is the maximum distance between two RBF centres and M is the number of nodes in the hidden layer is used[3][6]. This is the simplest practice in choosing RBF centres, and as a result we may need a greater number of hidden nodes to approximate the solution efficiently[2], compared with a hybrid model that involves *unsupervised learning* methods for computing the RBF centers, such as the *k-means* algorithm[6].

We start by presenting our results for the approximation of the $sin(2x)$ function:

Absolute residual error	**#Hidden nodes**	**#Epochs**
0.09489386966360751	5	7
0.008884393818372189	7	4
0.0008095762194060115	8	78

The true and approximated functions of each case are presented in the following graphs:



(a) Target error<0.1 (b) Target error<0.01 (c) Target error<0.001

Figure 1: True function of $sin(2x)$ and estimated functions combined

We continue by presenting our results for the approximation of the $square(2x)$ function:

Absolute residual error	**#Hidden nodes**	**#Epochs**
0.09478667091104492	10	50
target < 0.01	-	-
target < 0.001	-	-

We can see from the table above, that in contrast with the $sin(2x)$ function, we can only drop the value of the absolute residual error value under 0.1, but no lower than 0.01, and a minimum of 10 layers in the hidden nodes is requested in order to achieve this result. From the following graph we can compare the similarity between the true and estimated functions of $square(2x)$:

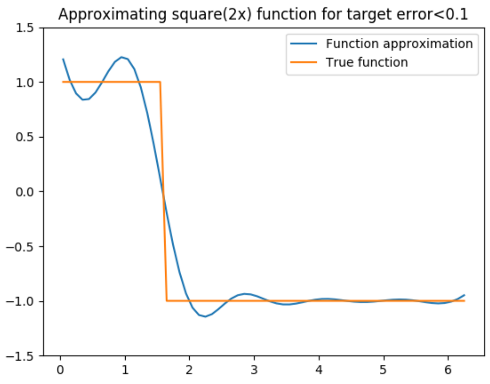


Figure 2: True function of $square(2x)$ and estimated functions combined

We can derive that a good estimation is achieved, but not a perfect one, which can be explained from the complex nature of the problem: while it is a regression problem, it is much more similar to a classification problem instead.

In order to address this matter in full extent, we employ the *k-means* algorithm in order to position the RBF centres in the imput space interval ($[0, 2\pi]$) and compare if a non-random placement will lead to better error performance.

Absolute residual error	**#Hidden nodes**	**#Epochs**
0.09790987862170945	12	2
target < 0.01	-	-
target < 0.001	-	-

Now, the number of hidden layers that is needed in order to drop the absolute residual error value lower than 0.1 is fixed: 12 RBFs are requested and it only takes 2 epochs to do so, in contrast with random placement that could need between 10 and 15 hidden nodes in order to achieve the same result. The 'problem' of not being able to drop this value lower than 0.01 or even 0.001 as in the $sin(2x)$ function approximation case still holds, which leads us to the next question. For full disclosure, we present the function approximation plot for the k-means defined centers:

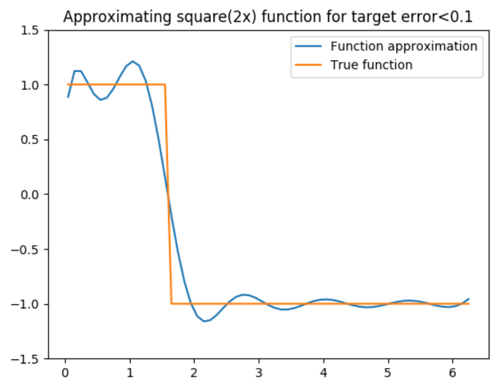


Figure 3: True function of $square(2x)$ and estimated functions combined using k-means defined RBFs

How can you simply transform the output of your RBF network to reduce the residual error to 0 for the square(2x) problem? Still, how many units do you need? In what type of applications could this transform be particularly useful?

A very simple transform, would be a thresholded output: assigning a value equal to 1 for all positive (≥ 0) outputs, and a negative value equal to -1 for all negative (< 0) outputs. Still, a minimum of 4 or 5 nodes in the hidden layer are requested in order to reach a zero error value. We can use such a practice for classification problems, where we have to separate between two distinct classes of datapoints.

3.2 Regression with noise

Compare the two learning approaches in terms of the number of epochs and the number of nodes needed to obtain the requested performance levels. How does it compare, particularly with respect to the number of RBF nodes, to the batch mode training of sin(2x) without noise, as in the previous assignment task? Compare the rate of convergence for different learning rates.

Results obtained from *batch* learning:

Absolute residual error	**#Hidden nodes**	**#Epochs**
0.0899170577432	6	1
0.0093784911593	12	8

Results obtained from *sequential* learning:

Absolute residual error	**#Hidden nodes**	**#Epochs**
0.0967834098246	7	1
0.00935787744574	17	1

We can infer from the above results, that the required number of nodes needed to obtain the error performance applying *batch* learning was, in general, less than those we needed for *sequential*. However, in the latter case, we only need 1 epoch to get the intended errors, while in the former, this is achieved after a finite number of epochs.

How important is the positioning of the RBF nodes in the input space? What strategy did you choose? Is it better than random positioning of the RBF nodes? Please support your conclusions with quantitative evidence (e.g., error comparison).

In general, the positioning of the RBF nodes in the input space is actually half part of the training process, so it plays central role to the problem. However, for this particular task, randomly selecting our RBF centers did not result into significant changes, given the nature of our data (every sample is representative). We used *k-means* algorithm to find the optimal position for the RBFs, which gave us slightly better results random positioning the nodes.

Evidence comparison for *batch* learning:

k-means
Total residual error fell under 0.1 for 6 hidden units: 0.0826660483521 after 9 epochs.
minimum residual error on validation set is: 0.0812322750135

Random positioning
Total residual error fell under 0.1 for 6 hidden units: 0.0818267772529 after 14 epochs.
minimum residual error on validation set is: 0.081310162151

Evidence comparison for *sequential* learning:

k-means
Total residual error fell under 0.1 for 7 hidden units and 0.1 learning rate: 0.0772840189382 after 1 epoch.
minimum residual error on validation set is: 0.0889305880261

Random positioning
Total residual error fell under 0.1 for 14 hidden units and 0.1 learning rate: 0.0682289578572 after 1 epoch.
minimum residual error on validation set is: 0.0920641015128

k-means
Total residual error fell under 0.1 for 7 hidden units and 0.01 learning rate: 0.0738202830137 after 1 epoch.
minimum residual error on validation set is: 0.0835565846608

Random positioning
Total residual error fell under 0.1 for 10 hidden units and 0.01 learning rate: 0.0705208321126 after 1 epoch.
minimum residual error on validation set is: 0.0844151060129

k-means
Total residual error fell under 0.1 for 7 hidden units and 0.001 learning rate: 0.0737130720345 after 1 epoch.
minimum residual error on validation set is: 0.0832256993583

Random positioning
Total residual error fell under 0.1 for 10 hidden units and 0.001 learning rate: 0.0703070589528 after 1 epoch.
minimum residual error on validation set is: 0.0837356237535

What are the main effects of changing the width of RBFs?

The width of RBFs, determines the amount of impact that they have to the inputs. Large width in an RBF, results in responses to several inputs while reducing the width, the unit respond to less inputs.

How does the rate of convergence change with different values of eta?

Higher values of η led to faster convergence of the learning process than smaller ones. However, when the learning rate was set to a greater values than a threshold e.g. 0.5, the algorithm did not (most of the times) reach the requested performance. This is reasonable, since big enough values of η would create an error to the weights.

Please compare your optimal RBF network trained in batch mode with a two-layer perceptron trained with backprop (also in batch mode). To nd a suitable candidate two-layer perceptron architecture try to use the same overall number of hidden units as in the RBF network and distribute them in different ways over the two hidden layers (use the existing libraries of your choice for these simulations). Please remember that generalisation performance and training time are of greatest interest.

The following results, were obtained by using an RBF network with 6-hidden units and a 2-layer perceptron with 3 nodes per layer:

	Generalization error	**#Epochs**
RBF	0.081	9
Perceptron	0.422	100

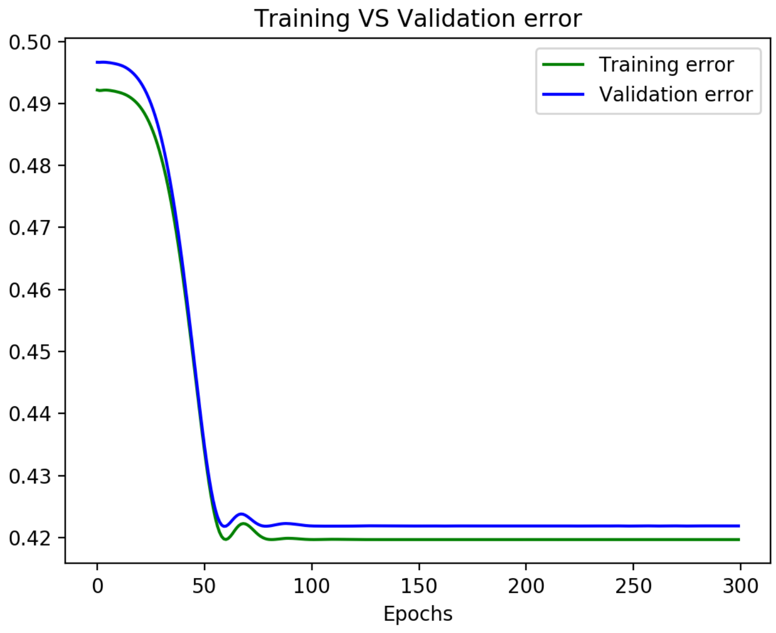


Figure 4: Error of the two-layer perceptron

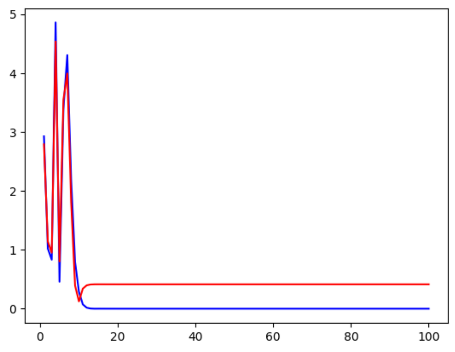
Although the error result from the RBF network might seem pretty bizarre, in fact a simple search[1] might show that the error derived from an MLP network for the sin functions remains in a high standard.

[1]`http://deliprao.com/archives/100`

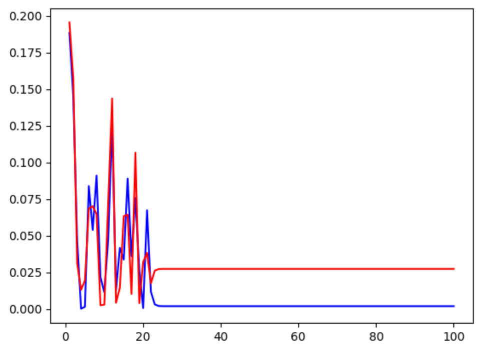
3.3 Competitive learning for the initialisation of RBF units

Compare the CL-based approach with your earlier RBF network where you manually positioned RBF nodes in the input space. Use the same number of units (it could be the number of units that allowed you to lower the absolute residual error below 0.01). Make this comparison for both noise-free and noisy approximation of sin(2x). Pay attention to convergence, generalisation performance and the resulting position of nodes.

We present the training (blue color) and (validation) error of our CL algorithm for both noise-free (left) and noisy (right) data:



(a) Noise-free data



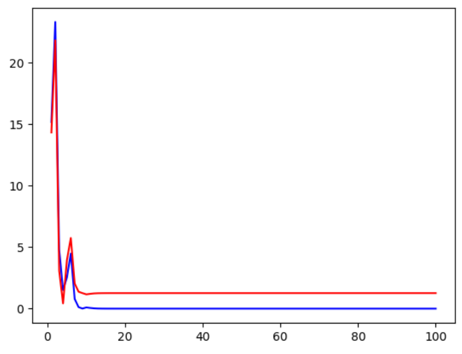
(b) Noisy data

Figure 5: CL training error comparison for noise-free and noisy data

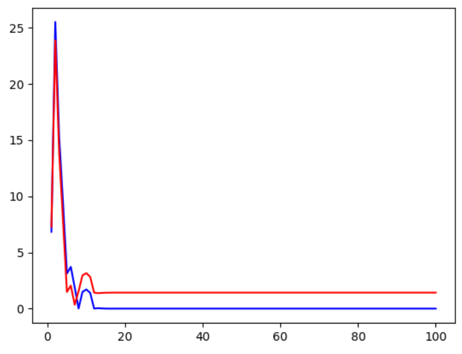
In general, because of the competitive learning approach in re-adjusting the position of the nodes, we can derive a better error-performance compared to a typical RBF network. The randomness in the first number of epochs, can be justified from the fact that the nodes are positioned in the RBF space until an "optimal area" is produced (as we will see in the last task of this subsection, they are readjusting until they form an area that will cover the input space efficiently).

Introduce a strategy to avoid dead units, e.g. by having more than a single winner. Choose an example to demonstrate this effect in comparison with the vanilla version of our simple CL algorithm.

Similarly, we present the results for re-positioning 2 minimum distance winner nodes each time, as a startegy to avoi dead units:



(a) Noise-free data



(b) Noisy data

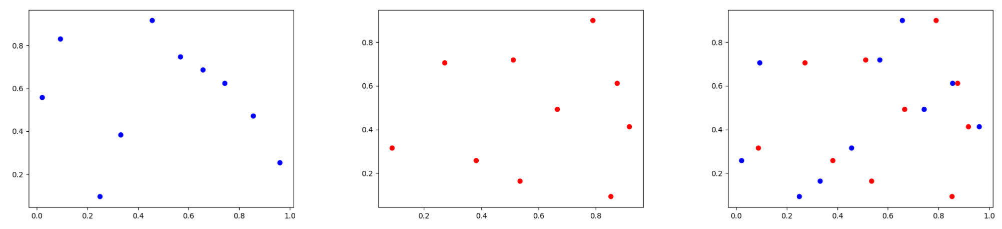
Figure 6: CL training error comparison for noise-free and noisy data

While we do not observe a great difference to the noise-free errors which were already in a pretty low value (although we still managed to drop them), the difference of using 2 winning nodes can be seen in the noisy data, in comparison with the previous errors.

Configure an RBF network with the use of CL for positioning the RBF units to approximate a two-dimensional function, i.e. from R2 to R2.As training examples please use noisy data from ballistical experiments where inputs are pairs: <angle, velocity> and the outputs are pairs: <distance, height> There are two datasets available: ballist for training and balltest for testing. First thing to do is to load the data and then train the RBF network to nd a mapping between the input and out- put values. Please be careful with the selection of a suitable number of nodes and their initialisation to avoid dead-unit and overfitting problems. Report your results and observations, ideally with the support of illustrations, and document your analyses (e.g., inspect the position of units in the input space).

This time, we need to approach a two-dimensional output through a two-dimensional input. We use a vector of RBF positions in the xy-plane, and as an RBF output we produce a mixture of Gaussian by using the position distance from the two-dimensional input. A global Φ matrix is created, and each dimension of output is then used to produce separate weight vector for each dimension. Again, a winner node is readjusted, thus making our algorithm a competitive learning one. By using a simple perceptron we produce the result in a joint output combining the two dimensions.

We present the original positioning of 10 RBF nodes before training (blue color), their final positioning (red color) and a combined plot:



(a) Positioning before training (b) Positioning after training (c) Combined positioning

Figure 7: Positioning of RBF nodes using *Competitive Learning*

As we discussed earlier, the nodes in the RBF space are readjusted in order to form an area that can cover the input space efficiently.

4 Assignment - Part 2

4.1 Topological Ordering of Animal Species

Check the resulting order.Does it make sense? If everything works, animals next to each other in the listing should always have some similarity between them. Insects should typically be grouped together, separate from the different cats, for example.

We follow the procedure described on the assignment instructions. For a total of 20 epochs, neighbourhood size value is initiated at 50, until dropped at 1 at the end of the training phase. Since this is a process that involves randomness, we present the results of 5 different executions in the following table:

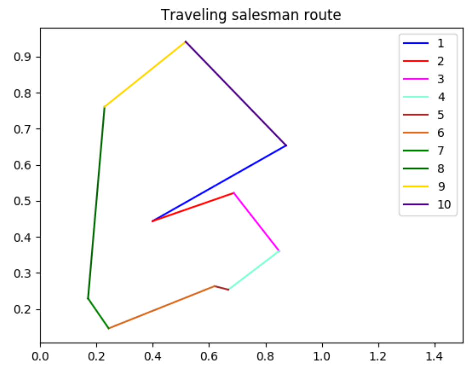
Animal ordering	**1st iteration**	**2nd iteration**	**3rd iteration**
#1	grasshopper	grasshopper	beetle
#2	beetle	beetle	dragonfly
#3	butterfly	butterfly	grasshopper
#4	moskito	dragonfly	butterfly
#5	dragonfly	moskito	moskito
#6	spider	housefly	housefly
#7	housefly	spider	spider
#8	pelican	penguin	penguin
#9	penguin	pelican	pelican
#10	ostrich	ostrich	ostrich
#11	duck	duck	duck
#12	crocodile	seaturtle	seaturtle
#13	seaturtle	frog	frog
#14	crocodile	crocodile	crocodile
#15	frog	walrus	walrus
#16	walrus	bear	hyena
#17	elephant	hyena	bear
#18	horse	dog	dog
#19	pig	rat	skunk
#20	camel	lion	ape
#21	giraffe	cat	lion
#22	cat	skunk	cat
#23	bat	ape	bat
#24	rat	bat	rat
#25	skunk	elephant	rabbit
#26	ape	kangaroo	elephant
#27	bear	rabbit	kangaroo
#28	dog	antelop	horse
#29	hyena	giraffe	antelop
#30	rabbit	camel	giraffe
#31	kangaroo	pig	camel
#32	antilope	horse	pig

Despite the fact that the results are not the same due to the randomness of the process, we can conclude that they are pretty similar in terms of quality; insects are grouped together and first in order, followed by animals with wings and sea and lake animals afterwards etc. A quality order is preserved, and the derived results make sense.

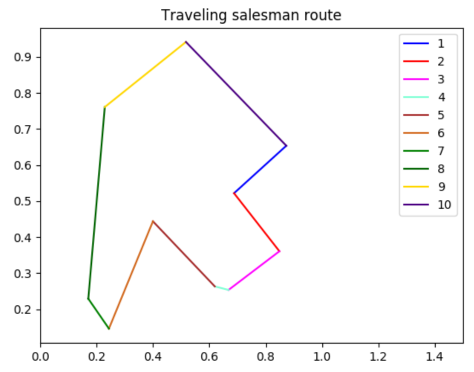
4.2 Cyclic Tour

Please plot both the tour and the training points. Give your interpretation.

In comparison with the previous task, we need to form a cyclic output grid that respects the connection between all the cities (first and last point of the weight grid need to be connected). By following a similar training process, we present the results of two different runs of our algorithm:



(a) First route



(b) Second route

Figure 8: Derived routes for the traveling salesman problem using SOM neural networks

We can see that depending on the starting city, the solution route differs. This is again expected due to the randomness involved in assigning the initial weights. However, what is common on both solutions, is that after the choices that are being made based o the starting point, follow a greedy algorithm solution of the choosing the closest city, which is one of the bibliography practices to approximate a solution for the TSP problem. All in all, we managed to create cyclic tours that differ only in the starting city, thus we can infer that our SOM[5] network puts the inputs in the output grid efficiently.

4.3 Data Clustering: Votes of MPs

Please display the results with respect to different attributes (i.e. party, gender, district) and describe the results, provide your interpretation.

Firstly, we present the votes of the parliament members sorted according to the political party they belong:

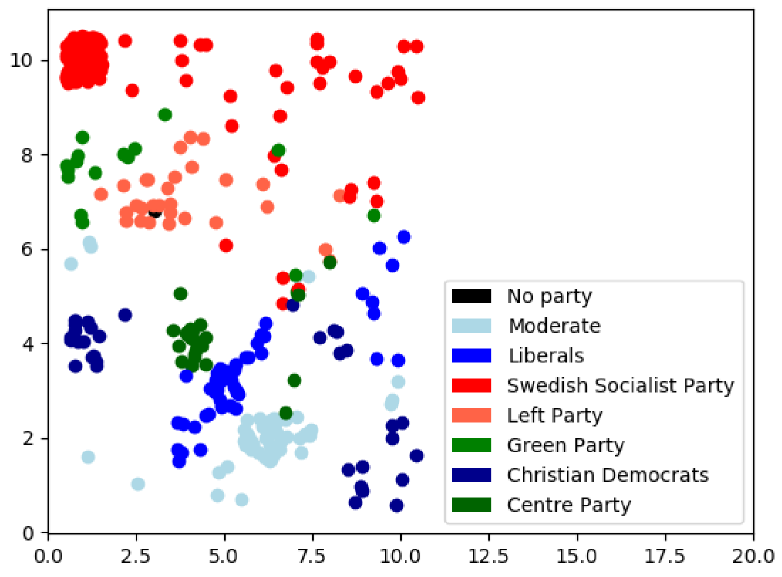


Figure 9: Votes of parliament members according to the political party

We can see that the SOM-based neural network puts efficiently the parliament members in the output grid. Taking under consideration what each party's political point of view, we can see that the votes of members that belong to parties with strict political orientation (e.g Left-wing and Right-wing parties) are accumulated together, while the votes of the Centre party for example can be found in both 'Right-wing' and 'Left-wing areas'. Moreover, political parties with similar political views are grouped together (e.g Socialists and Lefts, Cristian Democrats and Liberals etc) which strengthens our argument. In fact, we have managed to create our own version[2] of *Political Compass*[3] for the Swedish parliament of 2005!

[2]Although the distinction of the 4 quarters is not an exact political compass representation as it is defined in literature

[3]`https://www.politicalcompass.org`

We proceed by organizing our map according to the gender of the parliament members:

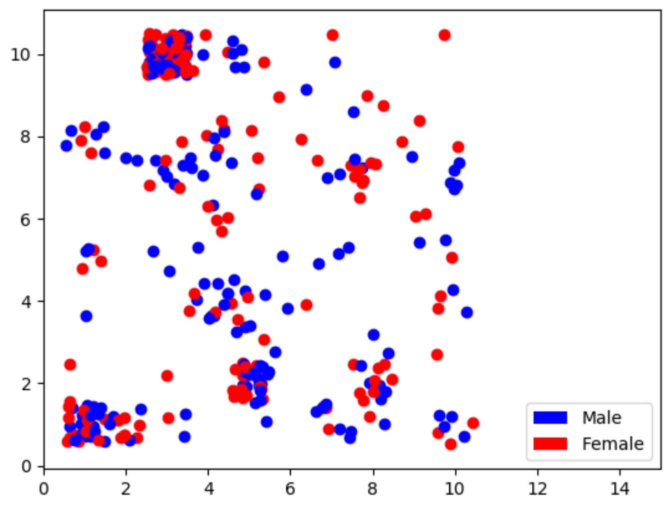


Figure 10: Votes of parliament members according to their sex

And finally, we present the same figure, but this time we have labeled the district that each member belongs:

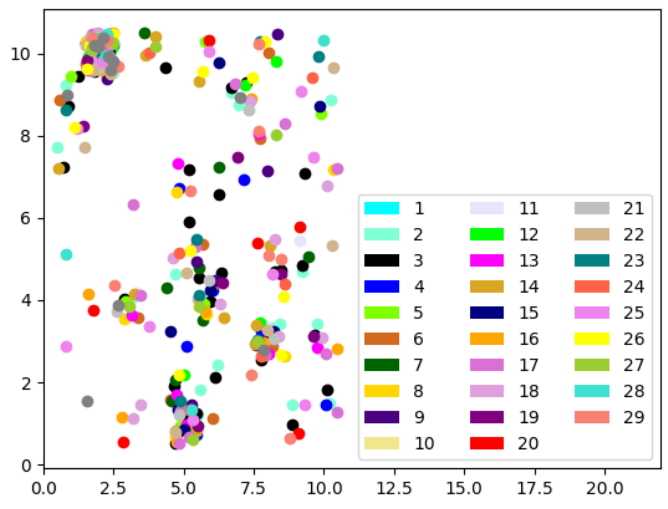


Figure 11: Votes of parliament members according to their sex

The assumption that can be derived from the diagrams above, is that what actually matters is the political party that each member belongs to in order to place their votes in the grid, which was expected. Not even the gender of each member means a fair amount of votes in conflicting matters, so that we can separate the genders in the two-dimensional grid. This, on the other hand, indicates that our society has progressed enough so that women and men have no serious conflicts under a political point of view!

To recapitulate, the total assumption is that 13 years ago, Sweden was an equal-rights country to both genders, following politics defined by the essence of political parties themselves, rather than the interests of districts.

References

[1] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, USA, 1995.

[2] K. Diamantaras. *Artificial Neural Networks.* Klidarithmos, 2007.

[3] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[4] P. Herman. *Artificial Neural Networks and Deep Architectures Course Lecture Slides.* KTH Royal Institute of Technology, 2018.

[5] T. Kohonen. *Self-organization and Associative Memory: 3rd Edition.* Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[6] S. Marsland. *Machine Learning: An Algorithmic Perspective.* Chapman & Hall/CRC, 2nd edition, 2014.

[7] R. Rojas. *Neural Networks : A Systematic Introduction.* Springer, 1996.