

## Tutorial for Introduction to Computational Intelligence in Winter 2015/16

Günter Rudolph, Vanessa Volz Lecture website: https://tinyurl.com/CI-WS2015-16

Sheet 2, Block I Due date: 25 November 2015, 2pm Discussion: 26/27 November 2015

12 November 2015

## Exercise 2.1: Radial Basis Function Nets (6 Points)

Implement an RBF network with Gaussian basis function in R. Alternatively, find, download, understand and describe a public domain version. Use that implementation to model the data set given in data.csv (first two columns input, third column class).

- Elaborate on your choice for the number of neurons q, the radii  $\sigma$  and the center  $c_k$ .
- Visualise (in one or more plots):
  - $\Phi(x;c)$  for all neuron center  $c_k$  and  $\forall (x,y) \in [-2,2] \times [-2,2]$
  - $-\,$  classification results and errors
- Using your visualisations, analyse your classification results and try to explain any shortcomings.

## Exercise 2.2: Weights for RBF (8 Points)

The optimal weights  $\mathbf{w}$  for an RBF net can be determined from the solution of the matrix equation  $P\mathbf{w} = \mathbf{y}$  via the pseudo inverse of P.

a) Show formally that the optimal weights can be determined via minimizing  $||P\mathbf{w} - \mathbf{y}||^2 = (P\mathbf{w} - \mathbf{y})'(P\mathbf{w} - \mathbf{y}) \rightarrow \min!$ 

Use differential calculus.

b) If the training examples lead to an ill-conditioned matrix P the numerical process can be made more stable if we minimize the objective function  $||P\mathbf{w} - \mathbf{y}||^2 + \mathbf{w}' D\mathbf{w} \to \min!,$ 

where  $D = \text{diag}(d_1, \ldots, d_q)$  is a diagonal matrix with positive diagonal entries  $d_i > 0$ .

Derive the expression for the optimal weights via differential calculus.

## Exercise 2.3: Hopfield Nets for Error Correction (6 Points)

Assume a Hopfield net with 9 neurons, aranged in a  $3\times 3$  grid, which results in a weight matrix with 81 entries.

Using associative memory, we want to store two different patterns (visualised in figure 1) in the network. The goal is the following: if a slightly modified version of either pattern is fed to the network, after a few iterations, the network displays the corresponding stored pattern, thus *correcting* the input.

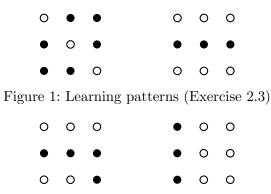


Figure 2: Test patterns (Exercise 2.3)

Storing the patterns can be achieved by using the *Hebb rule* (equation 1) to initialise the network weights. This way, the stored patterns are attractors (stable states) in the resulting energy landscape and can thus be retrieved.

Let *m* be the number of patterns to be stored in the network and *n* be the number of neurons in the network.  $x_i^{\mu}$ , with  $\mu \in [1, m], k \in [1, n]$ , is the state of neuron *i* (either 1 or -1) in pattern  $\mu$ .

According to Hebb's rule, the initial weights should then be:

$$W_{ij} = \frac{1}{n} \sum_{\mu=1}^{m} x_i^{\mu} x_j^{\mu}$$
(1)

(For more background information, look up Hopfield Nets and Hebbian learning)

- Implement the Hopfield Net described above and store the patterns in figure 1 by initialising the weights according to equation 1. Print the weight matrix.
- Now use the patterns depicted in figure 2 as input. In how many iterations does the network reach a stable state? What pattern does it retrieve? Explain the network's behaviour.
- What is the effect on the energy function if for each pattern  $\mu$  a weight  $\epsilon_{\mu} \in \mathbb{R}$  is factored into the weight initialisation in equation 1, such that:

$$W_{ij} = \frac{1}{n} \sum_{\mu=1}^{m} \epsilon_{\mu} x_i^{\mu} x_j^{\mu} \tag{2}$$