LELEC2870: Exercise session 3
Radial Basis Functions Networks

1 Objectives

The object of this session is the approximation of functions using radial basis functions networks (RBFNs). As you will see, this task requires the application of vector quantization methods (1st session) and of linear methods (2nd session). In addition to discovering a new machine learning model, you will thus have a great opportunity to validate your understanding of the first two exercise sessions of this course. As usual, an archive with data and Python scripts/notebooks is available on the Moodle website.

2 Description of the Session

During this session, you will implement the three learning steps of RBFNs. Thereafter, you will use the RBFN that you have just implemented to approximate the two bivariate functions

$$f : \Re^2 \to \Re : f(\mathbf{x}) = \frac{\sin \|\mathbf{x}\|_2}{\|\mathbf{x}\|_2}$$

$$f : \Re^2 \to \Re : f(\mathbf{x}) = e^{-\frac{1}{2}\|\mathbf{x}\|_2^2}$$

where $\|\mathbf{x}\|_2$ is the Euclidean norm of $\mathbf{x}$. For each function, a dataset is provided which contains a few training instances.

To help you, we provide a quick recap of RBFNs and their learning procedures in the next two sections.

3 Radial Basis Functions Networks

A RBFN model can be seen as an extension of the linear model

$$y = \sum_{i=1}^{D} w_i x_i + w_0$$

where D is the dimensionality of $\mathbf{x}$. Instead of working with the input vectors $\mathbf{x}$, we will work with a non-linear transformation of these inputs: the output is

$$y = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) + w_0$$

where

$$\phi : \mathbf{x} \in \Re^D \rightarrow \phi(\mathbf{x}) \in \Re^M$$

is a non-linear transformation and M is the number of components (or basis functions). Theoretically, any non-linear transformation ϕ could be used. In this session, we will use a Gaussian function of the form

$$\phi_i(\mathbf{x}, \mathbf{c}^i, \sigma^i) = \exp\Big(- \frac{\|\mathbf{x} - \mathbf{c}^i\|^2}{2{\sigma^i}^2} \Big)$$

where the index $i = 1, \ldots, M$ indicates the component of the ϕ function. Two parameters are associated with each function and must be learned: a center $\mathbf{c}^i$, belonging to the same space as $\mathbf{x}$, and a gaussian width σ^i.

4 Learning Procedure

A RBFN contains three different types of parameters:

- the position of the centers $\mathbf{c}^i$;
- the width of the Gaussian kernels σ^i;
- the weights of the linear model w_i.

While it is possible to use a single training procedure to learn all these parameters (e.g. by using gradient descent), it is often preferred to divide the learning in three distinct phases, each dedicated to one type of parameters.

4.1 Position of the centers

In the first phase of the learning process, the aim is to spread the centers in a way that mimics the density of probability of the input data. This is done by applying a vector quantization method. The centroids the method finds become the centers of your RBFN model. To implement this phase of training, you can use your code from the first session or our implementation of the competitive learning method provided in *competitive_learning.py*.

4.2 Widths of the Gaussian Functions

The widths of the gaussian kernels should depend on the density of the data around each center. A simple solution consists in choosing the width as the mean of distances between the data points inside a cluster (or Voronoï region) and its center, multiplied by a smoothing factor h (width scaling factor), common to all basis functions.

4.3 Weights of the linear transformation

Once the centers and widths are fixed, learning the remaining parameters (the weights w_i) becomes a linear problem with inputs $\phi_i(\mathbf{x})$ instead of $\mathbf{x}$. Just like what you have seen for the linear model, it is possible to optimize the weights w_i of the RBFN by optimizing an error criterion. If the least square criterion is chosen, it is possible to show that the optimal weights are given by (cfr. session 2):

$$\mathbf{w} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{t}$$

where transformed inputs are placed in the matrix

$$\Phi = \begin{pmatrix} 1 & \phi(\mathbf{x}^1) \\ 1 & \phi(\mathbf{x}^2) \\ \vdots & \vdots \\ 1 & \phi(\mathbf{x}^P) \end{pmatrix} = \begin{pmatrix} 1 & \phi_1(\mathbf{x}^1) & \phi_2(\mathbf{x}^1) & \cdots & \phi_M(\mathbf{x}^1) \\ 1 & \phi_1(\mathbf{x}^2) & \phi_2(\mathbf{x}^2) & \cdots & \phi_M(\mathbf{x}^2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(\mathbf{x}^P) & \phi_2(\mathbf{x}^P) & \cdots & \phi_M(\mathbf{x}^P) \end{pmatrix} \tag{1}$$

and targets are placed in the column vector

$$\mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_P \end{pmatrix}. \tag{2}$$

The first column of the matrix Φ corresponds to the bias term w_0.

To implement this phase of training, you can use your code from the second session or our implementation of the linear model provided in *linear_model.py*.

5 Questions

Now that RBFNs are implemented, try these models on the two provided datasets and visualise the results (you can use *visualize.py* for this). For a first test, you can use $M = 30$ centers and the width scaling factor $h = 4$.

What is the influence of the number of centers M and of the width scaling factor h ? Can you explain the results that you observe for small, medium and large values of these parameters ? Is there a link with the *overfitting* and *underfitting* phenomena ? Are both datasets as easy to learn?

For each training set, extract a few instances to build a smaller training set, learn a new RBFN and visualise the results. Do you see a difference ? In general, what is the impact of the number of training instances on learning ?