Radial Basis Function Networks: Algorithms

Introduction to Neural Networks : Lecture 13

1. The RBF Mapping
2. The RBF Network Architecture
3. Computational Power of RBF Networks
4. Training an RBF Network
5. Unsupervised Optimization of the Basis Functions
6. Finding the Output Weights

The Radial Basis Function (RBF) Mapping

We are working within the standard framework of function approximation. We have a set of N data points in a multi-dimensional space such that every D dimensional input vector $\mathbf{x}^p = \{x_i^p : i = 1,..., D\}$ has a corresponding K dimensional target output $\mathbf{t}^p = \{t_k^p : k = 1,..., K\}$. The target outputs will generally be generated by some underlying functions $g_k(\mathbf{x})$ plus random noise. The goal is to approximate the $g_k(\mathbf{x})$ with functions $y_k(\mathbf{x})$ of the form

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})$$

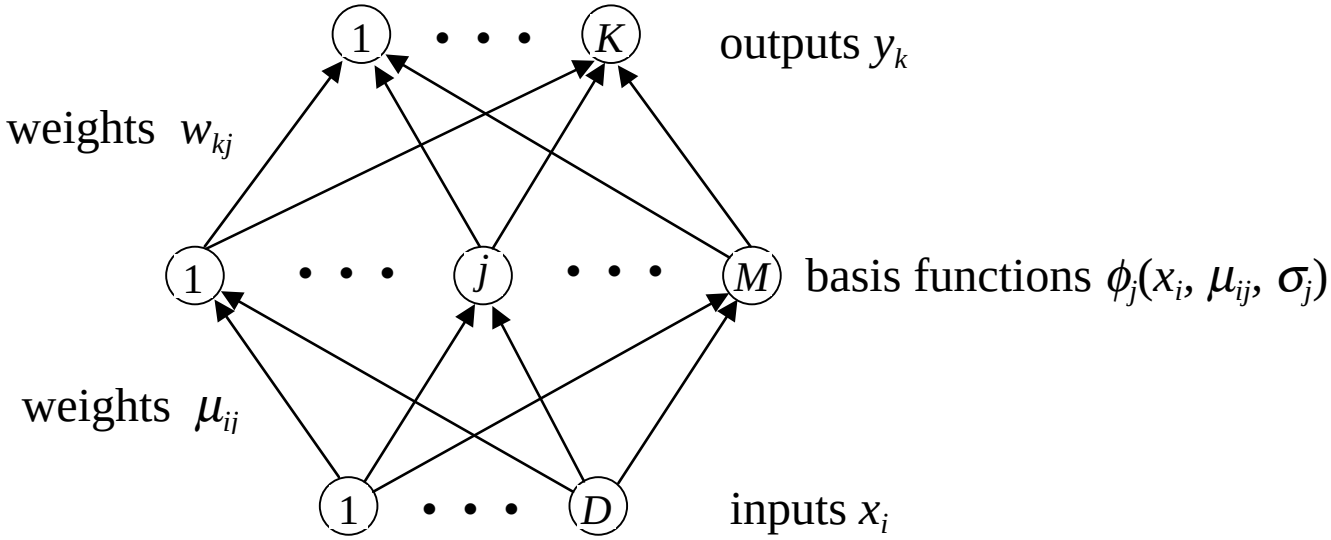
We shall concentrate on the case of Gaussian basis functions

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right)$$

in which we have basis centres $\{\boldsymbol{\mu}_j\}$ and widths $\{\sigma_j\}$. Naturally, the way to proceed is to develop a process for finding the appropriate values for M, $\{w_{kj}\}$, $\{\mu_{ij}\}$ and $\{\sigma_j\}$.

The RBF Network Architecture

We can cast the RBF Mapping into a form that resembles a neural network:



The hidden to output layer part operates like a standard feed-forward MLP network, with the sum of the weighted hidden unit activations giving the output unit activations. The hidden unit activations are given by the basis functions $\phi_j(\mathbf{x}, \boldsymbol{\mu}_j, \sigma_j)$, which depend on the "weights" $\{\mu_{ij}, \sigma_j\}$ and input activations $\{x_i\}$ in a non-standard manner.

Computational Power of RBF Networks

Intuitively, we can easily understand why linear superpositions of localised basis functions are capable of universal approximation. More formally:

Hartman, Keeler & Kowalski (1990, *Neural Computation*, vol.2, pp.210-215) provided a formal proof of this property for networks with Gaussian basis functions in which the widths $\{\sigma_j\}$ are treated as adjustable parameters.

Park & Sandberg (1991, *Neural Computation*, vol.3, pp.246-257; and 1993, *Neural Computation*, vol.5, pp.305-316) showed that with only mild restrictions on the basis functions, the universal function approximation property still holds.

As with the corresponding proofs for MLPs, these are existence proofs which rely on the availability of an arbitrarily large number of hidden units (i.e. basis functions). However, they do provide a theoretical foundation on which practical applications can be based with confidence.

Training RBF Networks

The proofs about computational power tell us what an RBF Network can do, but nothing about how to find all its parameters/weights $\{w_{kj}, \mu_{ij}, \sigma_j\}$.

Unlike in MLPs, in RBF networks the hidden and output layers play very different roles, and the corresponding "weights" have very different meanings and properties. It is therefore appropriate to use different learning algorithms for them.

The input to hidden "weights" (i.e. basis function parameters $\{\mu_{ij}, \sigma_j\}$) can be trained (or set) using any of a number of unsupervised learning techniques.

Then, after the input to hidden "weights" are found, they are kept fixed while the hidden to output weights are learned. Since this second stage of training involves just a single layer of weights $\{w_{jk}\}$ and linear output activation functions, the weights can easily be found analytically by solving a set of linear equations. This can be done very quickly, without the need for a set of iterative weight updates as in gradient descent learning.

Basis Function Optimization

One major advantage of RBF networks is the possibility of choosing suitable hidden unit/basis function parameters without having to perform a full non-linear optimization of the whole network. We shall now look at several ways of doing this:

1. Fixed centres selected at random
2. Orthogonal least squares
3. K-means clustering

These are all unsupervised techniques, which will be particularly useful in situations where labelled data is in short supply, but there is plenty of unlabelled data (i.e. inputs without output targets). Next lecture we shall look at how we might get better results by performing a full supervised non-linear optimization of the network instead.

With either approach, determining a good value for M remains a problem. It will generally be appropriate to compare the results for a range of different values, following the same kind of validation/cross validation methodology used for optimizing MLPs.

Fixed Centres Selected At Random

The simplest and quickest approach to setting the RBF parameters is to have their centres fixed at M points selected at ***random*** from the N data points, and to set all their widths to be equal and fixed at an appropriate size for the distribution of data points.

Specifically, we can use normalised RBFs centred at $\{\boldsymbol{\mu}_j\}$ defined by

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \qquad \text{where} \quad \{\boldsymbol{\mu}_j\} \subset \{\mathbf{x}^p\}$$

and the σ_j are all related in the same way to the maximum or average distance between the chosen centres $\boldsymbol{\mu}_j$. Common choices are

$$\sigma_j = \frac{d_{\max}}{\sqrt{2M}} \qquad \text{or} \qquad \sigma_j = 2d_{\text{ave}}$$

which ensure that the individual RBFs are neither too wide, nor too narrow, for the given training data. For large training sets, this approach gives reasonable results.

Orthogonal Least Squares

A more principled approach to selecting a sub-set of data points as the basis function centres is based on the technique of ***orthogonal least squares***.

This involves the sequential addition of new basis functions, each centred on one of the data points. At each stage, we try out each potential Lth basis function by using the N–L other data points to determine the networks output weights. The potential Lth basis function which leaves the smallest residual output sum squared output error is used, and we move on to choose which $L+1$th basis function to add.

This sounds wasteful, but if we construct a set of orthogonal vectors in the space S spanned by the vectors of hidden unit activations for each pattern in the training set, we can calculate directly which data point should be chosen as the next basis function.

To get good generalization we generally use validation/cross validation to stop the process when an appropriate number of data points have been selected as centres.

K-Means Clustering

A potentially even better approach is to use clustering techniques to find a set of centres which more accurately reflects the distribution of the data points.

The ***K-Means Clustering Algorithm*** picks the number K of centres in advance, and then follows a simple re-estimation procedure to partition the data points $\{\mathbf{x}^p\}$ into K disjoint sub-sets S_j containing N_j data points to minimize the sum squared clustering function

$$J = \sum_{j=1}^{K} \sum_{p \in S_j} \left\| \mathbf{x}^p - \boldsymbol{\mu}_j \right\|^2$$

where $\boldsymbol{\mu}_j$ is the mean/centroid of the data points in set S_j given by

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{p \in S_j} \mathbf{x}^p$$

Once the basis centres have been determined in this way, the widths can then be set according to the variances of the points in the corresponding cluster.

Dealing with the Output Layer

Given the hidden unit activations $\phi_j(\mathbf{x}, \boldsymbol{\mu}_j, \sigma_j)$ are fixed while we determine the output weights $\{w_{jk}\}$, we essentially only have to find the weights that optimise a single layer linear network. As with MLPs we can define a sum-squared output error measure

$$E = \tfrac{1}{2} \sum_p \sum_k \left(y_k(\mathbf{x}^p) - t_k^p \right)^2$$

but here the outputs are a simple linear combination of the hidden unit activations, i.e.

$$y_k(\mathbf{x}^p) = \sum_{j=0}^{M} w_{kj} \phi_j(\mathbf{x}^p) \; .$$

At the minimum of E the gradients with respect to all the weights w_{ki} will be zero, so

$$\frac{\partial E}{\partial w_{ki}} = \sum_p \left(\sum_{j=0}^{M} w_{kj} \phi_j(\mathbf{x}^p) - t_k^p \right) \phi_i(\mathbf{x}^p) = 0$$

and linear equations like this are well known to be easy to solve analytically.

Computing the Output Weights

Our equations for the weights are most conveniently written in matrix form by defining matrices with components $(\mathbf{W})_{kj} = w_{kj}$, $(\mathbf{\Phi})_{pj} = \phi_j(\mathbf{x}^p)$, and $(\mathbf{T})_{pk} = \{t_k^p\}$. This gives

$$\mathbf{\Phi}^{\mathsf{T}}\left(\mathbf{\Phi}\mathbf{W}^{\mathsf{T}} - \mathbf{T}\right) = 0$$

and the formal solution for the weights is

$$\mathbf{W}^{\mathsf{T}} = \mathbf{\Phi}^{\dagger}\mathbf{T}$$

in which we have the standard ***pseudo inverse*** of $\mathbf{\Phi}$

$$\mathbf{\Phi}^{\dagger} \equiv (\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathsf{T}}$$

which can be seen to have the property $\mathbf{\Phi}^{\dagger}\mathbf{\Phi} = \mathbf{I}$. We see that the network weights can be computed by fast linear matrix inversion techniques. In practice we tend to use singular value decomposition (SVD) to avoid possible ill-conditioning of $\mathbf{\Phi}$, i.e. $\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi}$ being singular or near singular.

Overview and Reading

1. We began by defining Radial Basis Function (RBF) mappings and the corresponding network architecture.
2. Then we considered the computational power of RBF networks.
3. We then saw how the two layers of network weights were rather different and different techniques were appropriate for training each of them.
4. We first looked at several unsupervised techniques for carrying out the first stage, namely optimizing the basis functions.
5. We then saw how the second stage, determining the output weights, could be performed by fast linear matrix inversion techniques.

Reading

1. Bishop: Sections 5.2, 5.3, 5.9, 5.10, 3.4
2. Haykin: Sections 5.4, 5.9, 5.10, 5.13