Radial Basis Function Networks: Algorithms

Neural Computation : Lecture 14

The Radial Basis Function (RBF) Mapping

We are working in the standard regression framework of function approximation, with a set of N training data points in a D dimensional input space, such that each input vector $\mathbf{x}^p = \{x_i^p : i = 1, \ldots, D\}$ has a corresponding K dimensional target output $\mathbf{t}^p = \{t_k^p : k = 1, \ldots, K\}$. 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

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

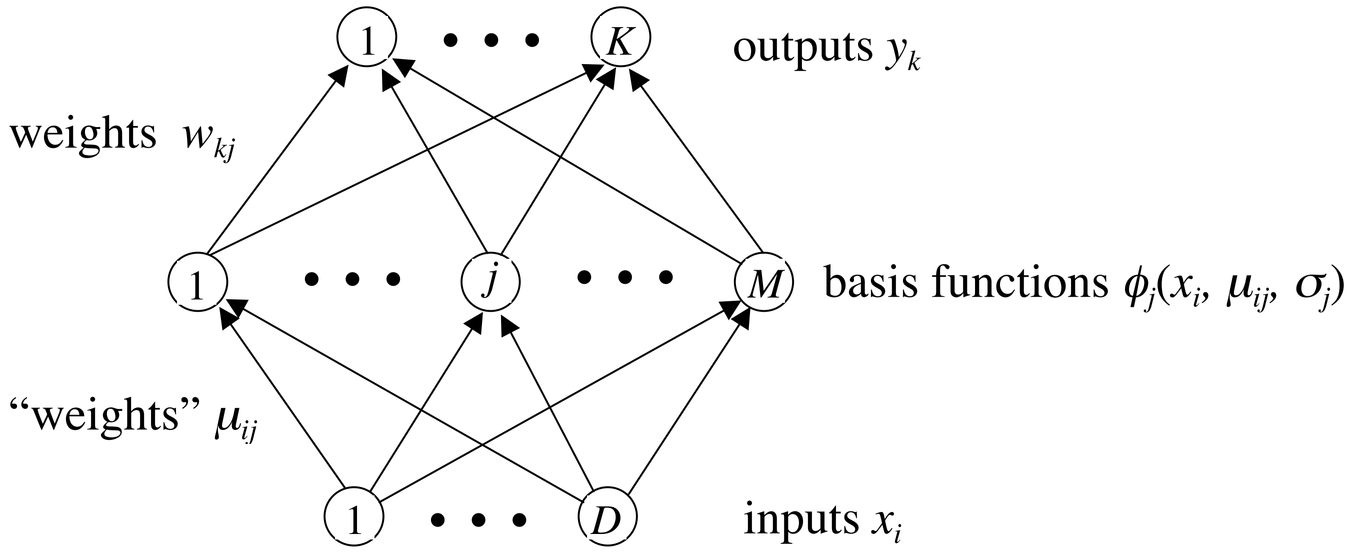
We shall concentrate on the case of Gaussian basis functions

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

which have 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

The RBF Mapping can be cast 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, it is not difficult to 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 tell us nothing about how to find values for all its parameters/weights $\{M, w_{kj}, \boldsymbol{\mu}_j, \sigma_j\}$.

Unlike in MLPs, the hidden and output layers in RBF networks operate in very different ways, 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 one of several possible unsupervised learning techniques.

Then, after the input to hidden "weights" are found, they are kept fixed while the hidden to output weights are learned. This second stage of training only involves a single layer of weights $\{w_{jk}\}$ and linear output activation functions, and we have already seen how the necessary weights can be found very quickly and easily using simple matrix pseudo-inversion, as in Single Layer Regression Networks or Extreme Learning Machines.

Basis Function Optimization

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

1. Fixed centres selected at random
2. Clustering based approaches
3. Orthogonal Least Squares

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). Later we shall look at how one might try to get better results by performing a full supervised non-linear optimization of the network instead.

With all these approaches, 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 for 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

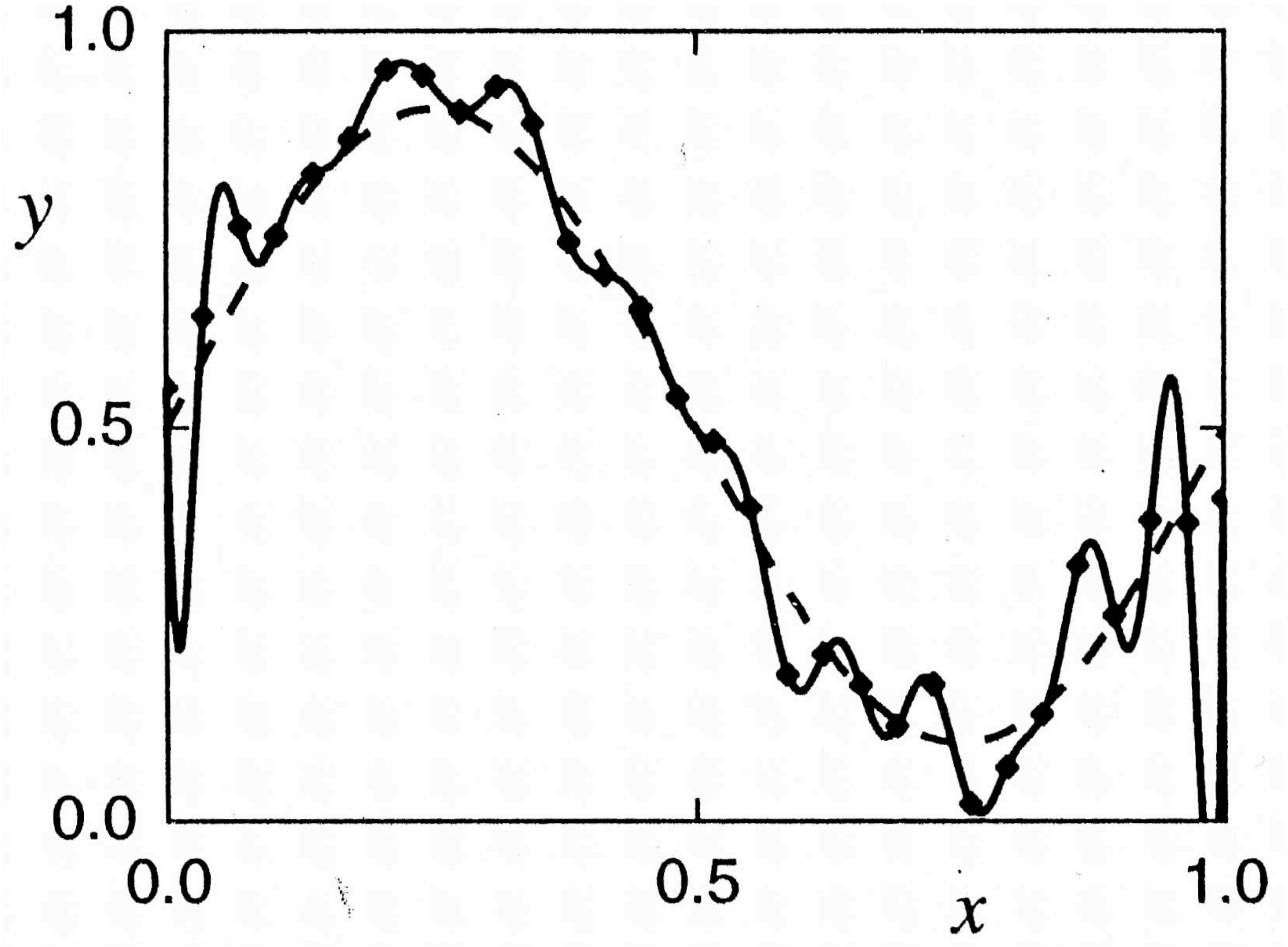
$$\phi_j(\mathbf{x}) = \exp\left(-\frac{\left\| \mathbf{x} - \boldsymbol{\mu}_j \right\|^2}{2\sigma_j^{\,2}} \right) \qquad \text{where} \quad \{\boldsymbol{\mu}_j\} \subset \{\mathbf{x}^p\}$$

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

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

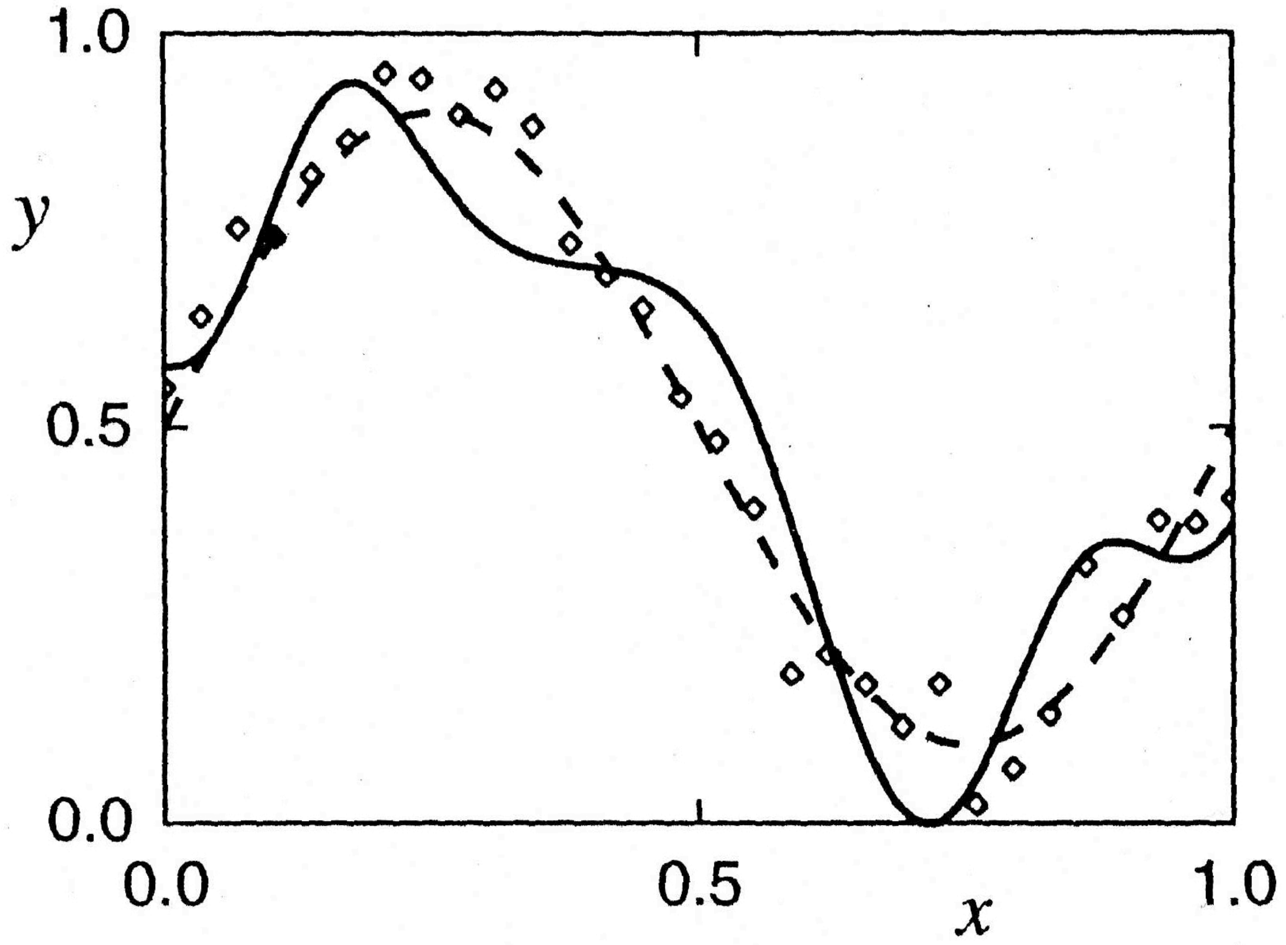
which ensure that the individual RBFs are neither too wide, nor too narrow, for the given training data. For large training sets, this approach gives reasonable results.

Example 1 : $M = N$, $\sigma = 2d_{ave}$



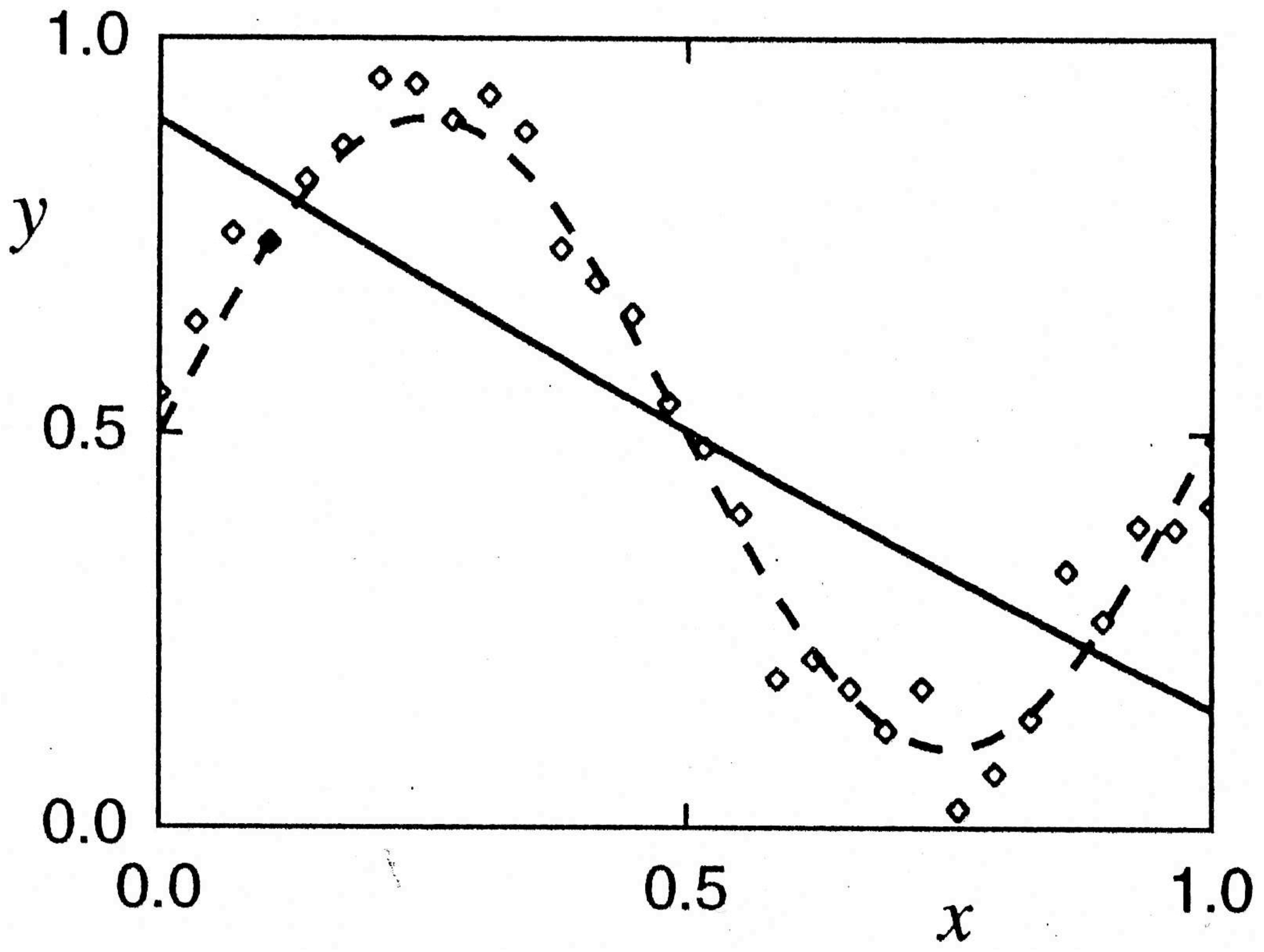
From: Neural Networks for Pattern Recognition, C. M. Bishop, Oxford University Press, 1995.

Example 2 : $M \ll N$, $\sigma \ll 2d_{ave}$



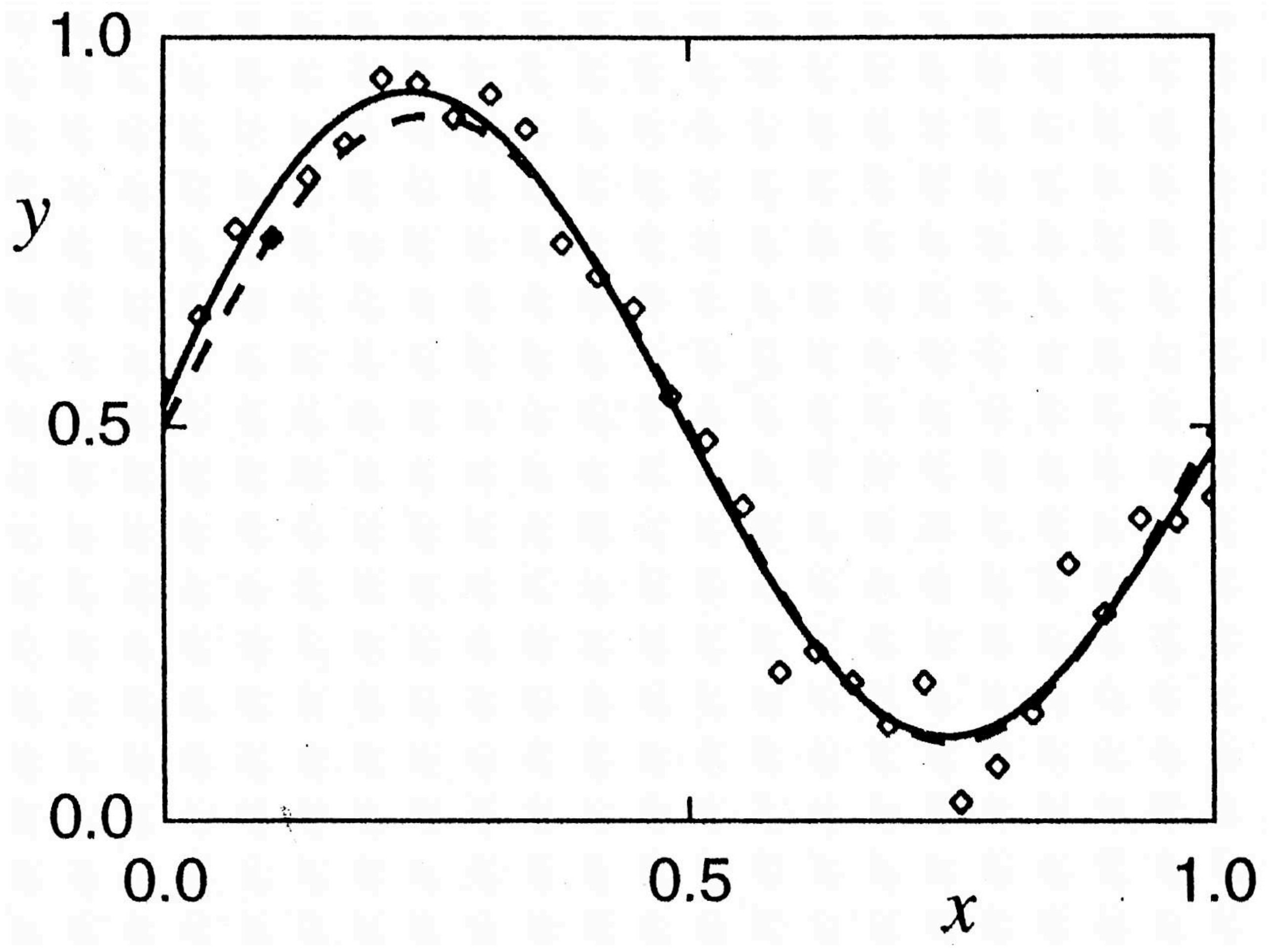
From: Neural Networks for Pattern Recognition, C. M. Bishop, Oxford University Press, 1995.

Example 3 : $M \ll N$, $\sigma \gg 2d_{ave}$



From: Neural Networks for Pattern Recognition, C. M. Bishop, Oxford University Press, 1995.

Example 4 : $M << N$, $\sigma = 2d_{ave}$



From: Neural Networks for Pattern Recognition, C. M. Bishop, Oxford University Press, 1995.

Clustering Based Approaches

An obvious problem with picking the RBF centres to be a random subset of the data points, is that the data points may not be evenly distributed throughout the input space, and relying on randomly chosen points to be the best subset is a risky strategy.

Thus, rather than picking random data points, a better approach is to use a principled clustering technique to find a set of RBF centres which more accurately reflect the distribution of the data points. We shall consider two such approaches:

1. ***K-Means Clustering*** – which we shall look at now.

2. ***Self Organizing Maps*** – which we shall look at next week.

Both approaches identify subsets of neighbouring data points and use them to partition the input space, and then an RBF centre can be placed at the centre of each partition/ cluster. Once the RBF centres have been determined in this way, each RBF width can then be set according to the variances of the points in the corresponding cluster.

K-Means Clustering Algorithm

The ***K-Means Clustering Algorithm*** starts by picking the number K of centres and randomly assigning the data points $\{\mathbf{x}^p\}$ to K subsets. It then uses a simple re-estimation procedure to end up with a partition of the data points into K disjoint sub-sets or clusters S_j containing N_j data points that minimizes 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 $\boldsymbol{\mu}_\mathrm{j}$ is the mean/centroid of the data points in set S_j given by

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

It does that by iteratively finding the nearest mean $\boldsymbol{\mu}_j$ to each data point $\mathbf{x}^p$, reassigning the data points to the associated clusters S_j, and then recomputing the cluster means $\boldsymbol{\mu}_j$. The clustering process terminates when no more data points switch from one cluster to another. Multiple runs can be carried out to find the local minimum with lowest J.

Orthogonal Least Squares

Another principled approach for 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. If we already have L such basis functions, there remain N–L possibilities for the next, and we can determine the output weights for each of those N–L potential networks. Then the L+1th basis function which leaves the smallest residual sum squared output error is chosen, and we then go on to choose the next basis function.

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 at each stage, and the output layer weights can be determined at the same time.

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

Dealing with the Output Layer

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

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

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

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

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

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

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

Computing the Output Weights

The 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

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

and hence the formal solution for the weights is

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

This involves the standard ***pseudo-inverse*** of $\mathbf{\Phi}$ which is defined as

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

and can be seen to have the property $\mathbf{\Phi}^{\dagger}\mathbf{\Phi} = \mathbf{I}$. Thus the network weights can be computed by fast linear matrix inversion techniques. In practice, it is normally best to use Singular Value Decomposition (SVD) techniques that can avoid problems due to possible ill-conditioning of $\mathbf{\Phi}$, i.e. $\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi}$ being singular or near singular.

Supervised RBF Network Training

Supervised training of the basis function parameters may lead to better results than the above unsupervised procedures, but the computational costs are usually enormous.

The obvious approach would be to perform gradient descent on a sum squared output error function as we did for MLPs. That would give the error function:

$$E = \sum_p \sum_k (t_k^p - y_k(\mathbf{x}^p))^2 = \sum_p \sum_k (t_k^p - \sum_{j=0}^{M} w_{kj} \phi_j \big(\mathbf{x}^p, \boldsymbol{\mu}_j, \sigma_j\big))^2$$

and one could iteratively update the weights/basis function parameters using

$$\Delta w_{jk} = -\eta_w \frac{\partial E}{\partial w_{jk}} \qquad \Delta \mu_{ij} = -\eta_\mu \frac{\partial E}{\partial \mu_{ij}} \qquad \Delta \sigma_j = -\eta_\sigma \frac{\partial E}{\partial \sigma_j}$$

This involves all the problems of choosing the learning rates η, avoiding local minima and so on, that arise when training MLPs by gradient descent. Also, there is a tendency for the basis function widths to grow large leaving non-localised basis functions.

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 that different techniques were appropriate for training each of them.
4. We looked at three different unsupervised techniques for optimizing the basis functions, and then at how the corresponding output weights could be computed using fast linear matrix pseudo-inversion techniques.
5. We ended by looking at how supervised RBF training would work.

Reading

1. Bishop: Sections 5.2, 5.3, 5.9, 5.10, 3.4
2. Haykin-2009: Sections 5.4, 5.5, 5.6, 5.7