Handout 6: Radial Basis Functions & Alternative Topologies

Mark Gales
mjfg@eng.cam.ac.uk
October 2001
**Introduction**

In the previous lectures we have looked at multi-layer perceptrons where the units of the network are based on non-linear transforms of a weighted sum of the input vectors. The other major class of network uses units based on the distance between the input vector and a *prototype* vector. These are called *radial basis functions*.

Radial basis functions may be viewed as a two layer network. These are sometimes called **radial basis function (RBF) networks**. This lecture will examine:

- attributes of radial basis functions;
- the form of RBF networks;
- RBF training;
- a comparison of RBF networks with MLP.

In addition some alternative topologies that are sometimes used will be described:

- Kohonen networks (self-organising maps);
- projection pursuit;
- multi-variate adaptive regression splines (MARS).
Exact Interpolation

RBF networks have their origin in techniques for performing exact interpolation. Consider the supervised training case where we need to learn some mapping $h(x)$ such that for each training example

$$h(x_p) = t(x_p)$$

Consider a basis function of the form $\phi(||x - x_p||)$, where $\phi()$ is some non-linear function and $||x - x_p||$ is a distance of the vector $x$ from the prototype vector $x_p$.

For the case of $n$ training examples, we can define the mapping $h(x)$ as

$$h(x) = \sum_{p=1}^{n} w_p \phi(||x - x_p||)$$

From the exact mapping requirement

$$\Phi w = t$$

where

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}, \quad t = \begin{bmatrix} t(x_1) \\ \vdots \\ t(x_n) \end{bmatrix}$$

$$\Phi = \begin{bmatrix} \phi(||x_1 - x_1||) & \cdots & \phi(||x_1 - x_n||) \\ \vdots & \ddots & \vdots \\ \phi(||x_n - x_1||) & \cdots & \phi(||x_n - x_n||) \end{bmatrix}$$
Exact Interpolation (cont)

Provided that the inverse $\Phi^{-1}$ exists then for exact interpolation

$$w = \Phi^{-1}t$$

It has been shown that for a large class of functions $\phi()$ if the set of points $x_1, \ldots, x_n$ is distinct then $\Phi^{-1}$ exists.

With the weights defined above the function $h(x)$ represents a continuous function that passes through all the target points, $t(x_p)$.

There are number of standard forms of $\phi()$ used:

| Non-linearity          | Form $(z = (||x - x_p||)/\sigma)$ |
|------------------------|------------------------------------|
| Gaussian               | $\exp(-z^2)$                       |
| Exponential            | $\exp(-z)$                         |
| Quadratic              | $z^2 + \alpha z + \beta$          |
| Inverse Quadratic      | $\frac{1}{1+z^2}$                  |
| Thin plate spine       | $z^\alpha \log(z)$                 |
| Trigonometric          | $\sin(z)$                          |

$\sigma$ is a variable that controls the smoothness of the interpolation function.

Some of the functions are localized basis functions. These have the property that $\phi(z) \to 0$ as $z \to \infty$. 
RBF Networks

The generalisation of exact interpolation to RBF networks:

- the number of basis functions, \( M \), does not need to be the same as the number of points, \( n \);
- the centers of the basis functions do not have to occur at the input values;
- the value of \( \sigma \) is not constrained to be the same for all basis functions;
- a bias value is added to the summation to compensate for average differences of the activation functions and the output.

An RBF network may be drawn as a multi-layer network.
RBF Training

RBFs are usually trained in a two stage process. The two stages are:

1. **Basis functions**, determine $M$ prototypes, $\mu_1, \ldots, \mu_M$, and smoothing parameters, $\sigma_1, \ldots, \sigma_M$. These may be trained in an *unsupervised* fashion using $x_1, \ldots, x_n$.

2. **Weights**, determine output layer weights $\tilde{W}$. These are trained in a *supervised* fashion. The basis functions are assumed to be fixed. If there are fewer basis functions than data points ($M < n$) then exact interpolation cannot generally be achieved.

It is possible to train the basis functions and weights in a single, coherent, fashion. Supervised training is used in this case. We have already seen that RBFs may be viewed as a multi-layer network. Thus it is quite natural to use gradient descent training and similar cost function as used for MLPs. The same problems, and possible solutions, as used for MLPs can be used for RBFs.

Though the use of the two-stage training scheme is suboptimal it is commonly used as it requires less supervised training data. Also standard training schemes for each of the two stages may be used.
Weight Training

We could use any of the standard training criteria, however it is convenient to use a squared error criterion. The output may be written as

\[ y_k(x) = \sum_{m=0}^{M} \hat{w}_{km} \phi_m(x) \]

where we define the bias basis function \( \phi_0(x) = 0 \). In matrix form

\[ y(x) = \tilde{W} \phi(x) \]

The error function is

\[ E = \frac{1}{2} \sum_{p=1}^{n} \sum_{k=1}^{K} (y_k(x_p) - t_k(x_p))^2 \]

This is an extension of the least mean squares estimation to multiple target values. Minimising the error function yields

\[ \Phi' \Phi \tilde{W}[1]' = \Phi' T \]

where

\[ \Phi = \begin{bmatrix} \phi_1(x_1) & \cdots & \phi_M(x_1) \\ \vdots & \cdots & \vdots \\ \phi_1(x_n) & \cdots & \phi_M(x_n) \end{bmatrix} \]

\[ T = \begin{bmatrix} t_1(x_1) & \cdots & t_K(x_1) \\ \vdots & \cdots & \vdots \\ t_1(x_n) & \cdots & t_K(x_n) \end{bmatrix} \]

This can be solved using the pseudo-inverse

\[ \tilde{W}[1]' = (\Phi' \Phi)^{-1} \Phi' T \]
6. Radial Basis Functions & Alternative Topologies

**K-Means Clustering**

One unsupervised clustering scheme is *K-means* clustering. This aims to find the set of prototypes that minimises

\[
E(\mu_1, \ldots, \mu_M) = \sum_{p=1}^{n} ||x_p - \mu_{z_p}||^2
\]

where observation \( p \) is assigned to cluster \( z_p (z_p \in \{1, \ldots, M\}) \).

The basic algorithm is

1. Start with a set of prototype vectors \( \mu_1[0], \ldots, \mu_M[0], \tau = 0 \);
2. Assign all data points to one of the \( M \) prototypes using

\[
z_p[\tau] = \arg \min_m (||x_p - \mu_m[\tau]||)
\]

3. Estimate the centers given the assignments

\[
\mu_j[\tau + 1] = \frac{\sum_{z_p[\tau] = j} x_p}{\sum_{z_p[\tau] = j}}
\]

4. If the assignment of all the points is unchanged, \( z_p[\tau] = z_p[\tau - 1] \ \forall p \); else set \( \tau = \tau + 1 \) and goto 2.

Two common initialisation schemes are:

- **centroid splitting**: start with all the data in one cluster. Perturb the cluster to form two clusters, perform K-means. Keep repeating until \( M \) prototypes are generated.

- **random points**: randomly select \( M \) points from the \( n \) training points. Use these as the initial values for the prototype vectors.
RBFs for Posteriors

Consider a classification task with $K$ classes, $\omega_1, \ldots, \omega_K$. We will model the $K$ class conditional PDFs

$$p(x|\omega_1), \ldots, p(x|\omega_K)$$

using a set of $M$ basis distributions

$$p(x|\lambda_1), \ldots, p(x|\lambda_M)$$

where $\lambda_m$ denotes the $m^{th}$ basis distribution, as

$$p(x|\omega_j) = \sum_{m=1}^{M} p(x|\lambda_m) P(\lambda_m|\omega_j)$$

The unconditional PDF may be written as

$$p(x) = \sum_{m=1}^{M} p(x|\lambda_m) P(\lambda_m)$$

where

$$P(\lambda_m) = \sum_{j=1}^{K} P(\lambda_m|\omega_j) P(\omega_j)$$

The posterior probability of a particular class, $\omega_j$, is

$$P(\omega_j|x) = \frac{p(x|\omega_j) P(\omega_j)}{p(x)}$$

$$= \frac{\left[ \sum_{m=1}^{M} p(x|\lambda_m) P(\lambda_m|\omega_j) \right] P(\omega_j)}{\sum_{k=1}^{M} p(x|\lambda_k) P(\lambda_k)}$$
RBFs for Posteriors (cont)

We would like to re-express this in the form of an RBF network.

\[
P(\omega_j|\mathbf{x}) = \sum_{m=1}^{M} \left[ \frac{P(\lambda_m|\omega_j) P(\omega_j) p(\mathbf{x}|\lambda_m)}{\sum_{k=1}^{M} p(\mathbf{x}|\lambda_k) P(\lambda_k)} \right]
\]

\[
= \sum_{m=1}^{M} \left[ \left( \frac{P(\lambda_m|\omega_j) P(\omega_j)}{P(\lambda_m)} \right) \left( \frac{P(\lambda_m)p(\mathbf{x}|\lambda_m)}{\sum_{k=1}^{M} p(\mathbf{x}|\lambda_k) P(\lambda_k)} \right) \right]
\]

\[
= \sum_{m=1}^{M} \left[ P(\omega_j|\lambda_m) \left( \frac{P(\lambda_m)p(\mathbf{x}|\lambda_m)}{\sum_{k=1}^{M} p(\mathbf{x}|\lambda_k) P(\lambda_k)} \right) \right]
\]

Normalised basis functions of the form:

\[
\phi_i(\mathbf{x}) = \frac{P(\lambda_i)p(\mathbf{x}|\lambda_i)}{\sum_{k=1}^{M} p(\mathbf{x}|\lambda_k) P(\lambda_k)} = P(\lambda_i|\mathbf{x})
\]

and weights of the form

\[
w_{ij} = P(\omega_j|\lambda_i)
\]

For an RBF network of this form:

- the basis functions may be viewed as “hidden” classes;
- the output of the hidden layer may be viewed as hidden class posteriors;
- the output of the network may be viewed as a posterior probability.
RBF Network vs MLP

1. **Activation**: the transform associated with a single layer of an MLP is a weighted sum of the input vectors (plus bias) transformed by the activation function. The output is therefore constant for parallel \((d-1)\) dimensional hyperplanes in the input space. In contrast an RBF uses a distance from a set of prototype vectors. The activation of a single basis function is constant in \((d-1)\)-dimensional hyperspheres.

2. **Representation**: MLPs have a distributed representation. Many hidden units will typically contribute to the output value. In RBFs it is expected that the output value depends only on a few local units.

3. **Complexity**: MLPs typically have many weights giving a complex pattern of connectivity. Various different activation functions may be used in the network. RBFs have a simple architecture. The first set of weights are all set to one, the output layer weights only are learnt.

4. **Training**: All the parameters of an MLP are typically updated in a single training epoch. RBFs are usually trained in a two stage process, the basis functions are trained and then the set of weights.
Kohonen Network

A Kohonen network, or *self-organising map* (SOM), is an unsupervised learning procedure. Broadly it projects the high-dimensional \((d\)-dimensional\) input space onto an ordered, usually, 2-dimensional grid known as the map.

For the grid shown above there will be 16 nodes. Associated with each node, \(j\), is:

- a **reference** vector \(\mu_j\). This is in the original \(d\)-dimensional space.

- a **location** vector \(r_j\). This is a vector describing the position in the new 2-dimensional grid.

The idea behind the network is that nodes that are *topographically close* in the grid will activate each other to learn something from the same input vector \(x\).
Kohonen Network (cont)

The basic algorithm is:

1. Start with a set of initialised reference vectors \( \mathbf{\mu}_1[0], \ldots, \mathbf{\mu}_M[0] \), \( \tau = 0 \);

2. Assign all data points, \( x_1, \ldots, x_n \) to one of the \( M \) prototypes using

   \[ z_p[\tau] = \arg \min_m (||x_p - \mathbf{\mu}_m[\tau]||) \]

3. Update the values of the reference vectors using

   \[ \mathbf{\mu}_i[\tau + 1] = \mathbf{\mu}_i[\tau] + h_{iz}[\tau] (x_p - \mathbf{\mu}_i[\tau]) \]

   where \( h_{iz}[\tau] \) is the \textit{neighbourhood function}. It is a function relating node \( i \) to the assigned node \( z \). Set \( \tau = \tau + 1 \).

4. Stop if the convergence criterion has been satisfied; else goto (2).

For convergence the neighbourhood function must satisfy

- \( h_{iz}[\tau] \to 0 \) as \( \tau \to \infty \)
- \( h_{iz}[\tau] \to 0 \) as \( ||\mathbf{r}_i - \mathbf{r}_z|| \to \infty \).
Kohonen Network (cont)

Two simple choices of neighbourhood function occur in the literature:

- **Neighbourhood set:**

  \[
  h_{iz}[\tau] = \begin{cases} 
  \alpha[\tau] & i \in N_z[\tau] \\
  0 & i \not\in N_z[\tau]
  \end{cases}
  \]

  where \( N_z[\tau] \) is the set of points in the map that are close to \( z \). This typically gets smaller as \( \tau \) increases.

- **Gaussian function:**

  \[
  h_{iz}[\tau] = \alpha[\tau] \exp\left(-\frac{||r_i - r_z||^2}{2\sigma^2[\tau]}\right)
  \]

  where the parameter \( \sigma^2[\tau] \) defines the width of the kernel.

A reasonable form for \( \alpha[\tau] \) is

\[
\alpha[\tau] = 0.9 \left( 1 - \frac{\tau}{1000} \right)
\]

A similar monotonically decreasing function (with \( \tau \)) may be used for \( \sigma^2[\tau] \).

For large networks if the neighbourhood set is too small (or the kernel width is too small) the network will not converge to a global ordering. Instead a mosaic-like parcellation of the map may be seen. To avoid this the neighbourhood set should be initially set fairly large.
Projection Pursuit Regression

Statistician have developed a variety of techniques that can be regarded as complimentary to the MLP. One very closely related scheme is projection pursuit regression, or simply projection pursuit. This mapping can be described as

\[
y(x) = \sum_{m=1}^{M} w_m \phi_m(u_m x + u_{m0}) + w_0
\]

The parameters that are trained are:

- the \( M \)-dimensional weight vectors, \( w \), and bias \( w_0 \);
- the \( M \times (d + 1) \) matrix

\[
\tilde{U} = \begin{bmatrix}
    u_{10} & u_1 \\
    \vdots & \vdots \\
    u_{M0} & u_M
\end{bmatrix}
\]

These determined the projection of the input onto \( M \) \( d \)-dimensional hyperplanes;

- nonlinear function \( \phi_m() \), this is called the ridge function.

This may be directly compared to a two-layer MLP. The parameters \( u_j \) and \( u_{j0} \) may be viewed as the weights of the hidden layer associated with node \( j \). The node specific activation function \( \phi_j() \) transformed the weighted input which are then weighted to form the output. The training criterion commonly used with this model is least squares estimation. The activation functions used here are node specific and are trained.
Another important model is multivarariate adaptive regression spline. The mapping function may be written as

$$y(x) = \sum_{m=1}^{M} w_m \left[ K_m \prod_{k=1}^{\nu(m,k)} \phi_{mk}(x_{\nu(m,k)}) \right]$$

Here there are a set of one-dimensional spline functions, each of which depends on one of the input feature vector elements, $x_\nu$. The particular input is determined by a label function $\nu(m,k)$. The basis functions are adaptive in that we train:

- $K_m$ the number of factors;
- $\nu(m,k)$ the input vector element;
- $\phi_{mk}()$ the spline function.

MLPs have for the most part supplanted MARS and projection pursuit. Some of the reasons are:

1. error back propagation is simpler that the learning in projection pursuit or MARS;
2. MLPs have a variety of simplifications and regression techniques that may be applied;
3. refining MLPs with additional training data is usually simpler than for projection pursuit and MARS.