7

Radial Basis Functions

	global	local
empirical	KL	LKL
analytical	DFT	DWT

Fig. 7.1 Summary of types of linear mappings discussed to this point.

7.1 FUNCTION APPROXIMATION

In the previous chapters we have been concerned with determining linear mappings between points in $\mathbb{R}^n$ and points in $\mathbb{R}^m$. We have considered both global and local approaches of both an empirical and analytical nature, see Table 7.1 for a summary. Now we turn to the approximation of nonlinear mappings between vector spaces of potentially high (and possibly unequal) dimensions via the method of *radial basis functions* (RBF) [12, 67, 58]. We will see that RBFs possess a number of interesting properties which distinguish them from the multil-layer perceptrons (MLP) based on sigmoidal transfer functions to be described in Chapter 8.

The chief attraction of RBF-type networks is their flexibility. Basis functions may be chosen to be either local or global and they may or may not incorporate shape parameters which can be tuned to reflect the nature of the data. Another major attraction of such expansion functions is that, in the presence of clustering routine, the error function is quadratic in the learning parameters, a fact which has a dramatic effect on learning rates. Centers may also be adapted using a nonlinear optimization problem, leading to what is referred to as a *partially linear* optimization problem [70].

7.1.1 The Interpolation Problem

The approximating function is estimated by solving the general interpolation problem. The raw data for determining the interpolating function is a collection of input-output pairs $\{(\mathbf{x}^{(\mu)}, \mathbf{f}^{(\mu)})\}$. This involves constructing an approximation function $\tilde{\mathbf{f}}(\mathbf{x})$ capable of mapping a collection of input vectors (*patterns*) to a set of associated output vectors.

Given an ensemble of P input vectors $\{\mathbf{x}^{(\mu)}\}_{\mu=1}^P$, with each $\mathbf{x}^{(\mu)} \in \mathbb{R}^n$, and an associated ensemble of P output vectors, $\{\mathbf{f}^{(\mu)}\}_{\mu=1}^P$, with each $\mathbf{f}^{(\mu)} \in \mathbb{R}^m$, find a function $\tilde{\mathbf{f}} : \mathbb{R}^n \to \mathbb{R}^m$ such that the interpolation condition

$$\tilde{\mathbf{f}}(\mathbf{x}^{(\mu)}) = \mathbf{f}^{(\mu)} \tag{7.1}$$

is satisfied for all $\mu = 1 \dots P$.

Thus, we see that to determine an approximating, or *interpolating* function, we require raw data in the form of a collection of input-output pairs $\{(\mathbf{x}^{(\mu)}, \mathbf{f}^{(\mu)})\}$. The function $\tilde{\mathbf{f}}(\mathbf{x}^{(\mu)})$ may be viewed as approximating an unknown function $\mathbf{f}(\mathbf{x}^{(\mu)})$ which maps

$$\mathbf{f}(\mathbf{x}^{(\mu)}) = \mathbf{f}^{(\mu)} \tag{7.2}$$

for $\mu = 1, \ldots, P$.

Typically, we do not expect to obtain equality in equation (8.15), but rather attempt to satisfy the condition to within some acceptable tolerance, i.e.,

$$\|\tilde{\mathbf{f}}(\mathbf{x}^{(\mu)}) - \mathbf{f}^{(\mu)}\| < \text{tolerance} \tag{7.3}$$

Example 7.1. Consider the polynomial interpolation problem

$$\tilde{f}(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

For each available data point we may apply the interpolation condition

$$\mathbf{f}^{(\mu)} = w_0 + w_1 x^{(\mu)} + w_2 (x^{(\mu)})^2 + w_3 (x^{(\mu)})^3$$

$$\begin{pmatrix} \mathbf{f}^{(1)} \\ \vdots \\ \mathbf{f}^{(P)} \end{pmatrix} = \begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & (x^{(1)})^3 \\ \vdots & & & \vdots \\ 1 & x^{(P)} & (x^{(P)})^2 & (x^{(P)})^3 \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_3 \end{pmatrix}$$

For $P = 4$ we anticipate there will be a unique solution to the interpolation problem. However, if $P > 4$ then the linear system is over-determined and, in general, an approximate solution must be sought. As we see in the next example, the number of parameters required to define the interpolating polynomial increases dramatically with the input dimension.

Example 7.2. Now consider the case $\mathbf{x} \in \mathbb{R}^3$.

$$\tilde{f}(\mathbf{x}) = \sum_{i=1}^{3} w_i x_i + \sum_{j=1}^{i} \sum_{i=1}^{3} w_{ij} x_i x_j + \sum_{k=1}^{j} \sum_{j=1}^{i} \sum_{i=1}^{3} w_{ijk} x_i x_j x_k$$

In general, the number of distinct parameters associated with a polynomial of degree d and domain dimension n has

$$\text{number of parameters} = \frac{(n+d)!}{d!n!} \approx n^d$$

So for Example 7.2 above, there are $6!/(3!3!) = 20$ parameters. However, for $n = 1000$ and $d = 3$ there are already $167,668,501$ parameters. Note that at least this many data points are required to obtain a useful solution. This explosion of parameters is a by-product of the so-called *curse of dimensionality*

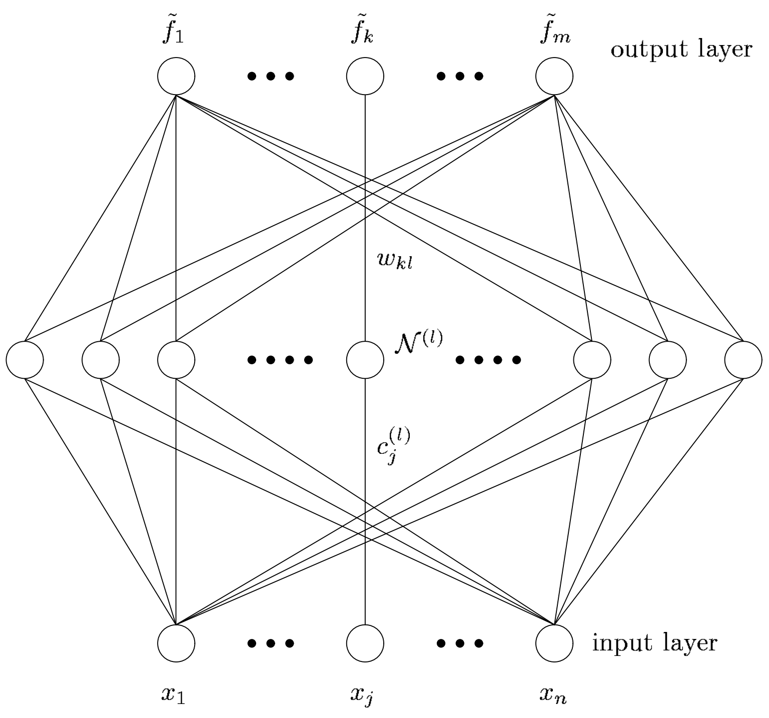

Fig. 7.2 A partial representation of a fully connected radial basis function network. The j'th input variable is linked to the l'th RBF basis function via the RBF center component $c_j^{(l)}$. The k'th output component of the network is related to the l'th RBF via the weight w_{kl}.

which impacts representations of mappings in terms of a fixed set of basis functions.

The problem arises because the fixed functions are designed to construct mappings from all of the domain to all of the range. In fact, all fixed sets of basis functions can be shown to suffer from this curse of dimensionality. Adaptive basis expansions, however, have been shown to resolve this curse, i.e., the number of basis functions required to solve the interpolation problem depends on the complexity of the data and not the dimension of the domain [5, 6].

$\phi(\xi)$	type	local or global	parameter
$\exp(-\xi/r^2)$	Gaussian	local	r
ξ	linear	global	none
ξ^3	cubic	global	none
$\xi^2 \ln \xi$	thin plate spline	global	none
$(\xi^2 + r^2)^{1/2}$	multiquadric	global	r
$(\xi^2 + r^2)^{-1/2}$	inverse multiquadric	local	r

Fig. 7.3 A list of types of radial basis functions; see [67] for further details.

7.2 THE GENERAL RBF EXPANSION

A *radial basis (vector) function expansions* are either local or global expansions of the form

$$\tilde{\mathbf{f}}(\mathbf{x}) = \mathbf{w}_0 + \sum_{m=1}^{N_c} \mathbf{w}_m \phi(\|\mathbf{x} - \mathbf{c}_m\|) \tag{7.4}$$

with $\tilde{\mathbf{f}}(\mathbf{x}) \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$, and N_c is the number of RBFs in the expansion. The set of *centers* $\{\mathbf{c}_m\}$, sometimes referred to as seed points, of the RBF $\phi(\xi)$ define the locations of the basis functions and are selected to represent the distribution of the data.

A standard type of RBF is a Gaussian of the form

$$\phi(\xi) = \exp(-\frac{\xi}{r^2})$$

RBFs of this type produce *local* represenations and the effective domain is determined by r; note that these may be taken to all be the same, or, the may be optimized for each individual RBF.

For the case of the Euclidean norm we have

$$\phi(\|\mathbf{x}^{(\mu)} - \mathbf{c}_m\|) = \exp(-\frac{\sum_{k=1}^{n}(x_k^{(\mu)} - c_k^{(m)})^2}{r^2})$$

Using the multiplicative property of the exponent,

$$\phi(\|\mathbf{x}^{(\mu)} - \mathbf{c}_m\|) = \prod_{k=1}^{n} \exp(-\frac{(x_k^{(\mu)} - c_k^{(m)})^2}{r^2}) \tag{7.5}$$

From the above calculation it is apparent that the Gaussian radial basis function has a specific domain of influence which is centered around the RBF center and extends in size proportional to the radius r. Outside of this domain, or *receptive field*, i.e., where one of the factors in Equation (7.5) is small, the Gaussian RBF does not contribute significantly to the expansion. These feature, leads to the *exclusivity* of the local RBF expansion, i.e., each weight is associated with a region of the input space and is relatively unaffected by data outside of its receptive field [?].

In contrast with the local Gaussian RBF it is possible to employ the global RBF function cubic

$$\phi(\xi) = \xi^3$$

This form of RBF is attractive (for appropriately scaled data) given the global nature of the functions which tend to require less data for training. In addition, the lack of radius parameter simplifies the fitting procedure.

There are many additional possibilities for RBFs as listed in Table 7.1.1. In an appendix to [67], A.L. Brown proves that if $\mathcal{D}$ is a compact subset of $\mathbb{R}^n$, then every continuous, real-valued function on $\mathcal{D}$ may be approximated uniformly by linear combinations of RBFs with centers in $\mathcal{D}$. In addition, Brown specifies that for functions ϕ to satisfy the completeness theorem, it is sufficient for $\tilde{\phi} : \mathbb{R}^+ \to \mathbb{R}$ defined by $\tilde{\phi}(\rho) = \phi(\sqrt{\rho})$ to be differentiable with completely monotonic derivative [67].

7.2.1 Computing the RBF Weights

In this section we consider the case where the function to be interpolated is a scalar field, i.e., where $f(x) \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^n$. For each input-output pair $(x^{(\mu)}, f^{(\mu)})$, we apply the interpolation condition given by Equation 7.2, i.e.,

$$f^{(\mu)} = w_0 + \sum_{m=1}^{N_h} w_m \phi(\|x^{(\mu)} - \mathbf{c}_m\|)$$

where $\mu = 1 \ldots P$. Writing as a linear system,

$$\begin{pmatrix} f^{(1)} \\ f^{(2)} \\ \vdots \\ f^{(P)} \end{pmatrix} = \begin{pmatrix} 1 & \phi_1^{(1)} & \phi_2^{(1)} & \ldots & \phi_{N_c}^{(1)} \\ 1 & \phi_1^{(2)} & \phi_2^{(2)} & \ldots & \phi_{N_c}^{(2)} \\ 1 & \vdots & & \ddots & \\ 1 & \phi_1^{(P)} & \phi_2^{(P)} & \ldots & \phi_{N_h}^{(P)} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{N_c} \end{pmatrix}$$

where

$$\phi_i^{(j)} = \phi(\|\mathbf{x}^{(i)} - \mathbf{c}_j\|)$$

We can write this more compactly as

$$\mathbf{f} = \mathbf{\Phi w} \tag{7.6}$$

where the *interpolation matrix* $\mathbf{\Phi}$ is defined as

$$(\mathbf{\Phi})_{ij} = \begin{cases} 1 & j = 1, \\ \phi_{j-1}^{(i)} & j > 1 \end{cases} \tag{7.7}$$

Fig. 7.4 The optimal solution to a least squares problem $\mathbf{\Phi w} = \mathbf{f}$ is provided by orthogonally projecting $\mathbf{f}$ onto the range of $\mathbf{\Phi}$.

with

$$\mathbf{w} = (w_1, w_2, \ldots, w_{N_c})^T$$

and

$$\mathbf{f} = (f^{(1)}, f^{(2)}, \ldots, f^{(P)})^T.$$

Thus, we see that solving for the weights is equivalent to solving a system of P linear equations in N_c unknowns.

In summary, to construct a RBF model it is necessary to determine the following:

- weights
  - direct methods for least squares
  - iterative methods for least squares
- centers
  - clustering
  - nonlinear optimization
- RBF type and parameter values

The type of RBF to use for a given set of data is still an active area of research. The parameter values may be determined via a nonlinear optimization problem, or simply fixed uniformly.

7.3 DIRECT METHODS FOR COMPUTING THE WEIGHTS

If there are more points than centers, i.e., $P > N_c$, then we have a classical least squares problem for an over-determined system

$$\mathbf{f} = \mathbf{\Phi w} \tag{7.8}$$

In general, $\mathbf{f}$ will not reside in the range of $\mathbf{\Phi}$, i.e., $\mathbf{f} \notin \mathcal{R}(\mathbf{\Phi})$. Hence we seek a solution to Equation (7.8) which will minimize the error

$$E = \|\mathbf{f} - \mathbf{\Phi w}\|$$

7.3.1 The Normal Equations

By the orthogonal projection theorem 2.1, the residual $\mathbf{r} = \mathbf{f} - \mathbf{\Phi w}$ will have minimum norm if it is orthogonal to the range of $\mathbf{\Phi}$. Thus we require

$$\mathcal{R}(\mathbf{\Phi}) \perp \mathbf{r}$$

This may be achieved by requiring that

$$(\mathbf{\Phi y})^T(\mathbf{f} - \mathbf{\Phi w}^*) = 0$$

since, for arbitrary $\mathbf{y}$, $\mathbf{\Phi}y \in \mathcal{R}(\mathbf{\Phi})$. Hence

$$\mathbf{y}^T\mathbf{\Phi}^T(\mathbf{f} - \mathbf{\Phi w}^*) = 0$$

After multiplying out, this becomes

$$\mathbf{\Phi}^T\mathbf{\Phi w}^* = \mathbf{\Phi}^T\mathbf{b} \tag{7.9}$$

These are the *normal* equations associated with $\mathbf{f} = \mathbf{\Phi w}$.

The normal equations may be solved efficiently via the *Cholesky factorization* method which determines a factorization for $\mathbf{\Phi}^T\mathbf{\Phi}$ in terms of R^TR where R is upper triangular. Now the solution $\mathbf{w}^*$ may be found by solving a lower triangular system followed by an upper triangular system.

If $\mathbf{\Phi}$ has full rank ($P > N_c = \text{rank}(\mathbf{\Phi})$) then the inverse $(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}$ exists and we may solve for the leasat squares solution

$$\mathbf{w}^* = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{b} \tag{7.10}$$

The matrix

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

is referred to as the pseudo-inverse of $\mathbf{\Phi}$.

Interpretation as a projection

We see from Figure 7.2.1 (and the orthogonal projection theorem) that a minimum error is achieved by orthogonally projecting the point $\mathbf{b}$ onto $\mathbf{\Phi}$. Thus we are solving

$$\mathbf{\Phi}\mathbf{w}^* = P\mathbf{b}$$

By applying $\mathbf{\Phi}$ to both sides of Equation (7.10) we observe that our previous calculation for $\mathbf{w}^*$ produces the orthogonal projection matrix

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

Observe that our full rank assumption is equivalent to the requirement that the columns of $\mathbf{\Phi}$ be independent. Thus we have obtained a formula for a projection onto the range of a collection of independent eigenvectors. Compare this formula with the one we obtained for a projection matrix onto the range of a collection of o.n. vectors $U = [\mathbf{u}^{(1)}| \dots |\mathbf{u}^{(N_c)}]$, i.e.,

$$P = UU^T$$

The fact that the orthogonal projection provides the key to the solution will lead us to several alternative methods solving Equation 7.8.

7.3.2 The SVD Solution

We have already found an extermely useful method for computing the range of a matrix via the SVD. It is exactly the span of the left singular vectors. If the reduced SVD of $\mathbf{\Phi}$ is

$$\mathbf{\Phi} = \hat{U}\hat{\Sigma}V^T$$

then the orthogonl projection matrix onto $\mathcal{R}(\mathbf{\Phi})$ is

$$P = \hat{U}\hat{U}^T$$

Given the solution to the least squares problem is

$$A\mathbf{w}^* = P\mathbf{b}$$

it follows

$$\hat{U}\hat{\Sigma}V^T\mathbf{w}^* = \hat{U}\hat{U}^T\mathbf{b}$$

Since $\hat{U}^T\hat{U} = I$ it follows

$$\hat{\Sigma}V^T\mathbf{w}^* = \hat{U}^T\mathbf{b}$$

Multiplying by $\hat{\Sigma}^{-1}$, then V we obtain

$$\mathbf{w}^* = V\hat{\Sigma}^{-1}\hat{U}^T\mathbf{b}$$

Now the expression for the pseudoinverse takes on the form

$$\mathbf{\Phi}^\dagger = V\hat{\Sigma}^{-1}\hat{U}^T$$

7.3.3 Gram-Scmidt (QR)

Given a collection of independent, but non-orthogonal, vectors, i.e., the columns of $\mathbf{\Phi}$, we may compute an o.n. set by applying the Gram-Schmidt (GS) procedure (or, preferably the numerically stable modified GS technique [84]). A standard result from linear algebra is that the resulting o.n. set $Q = [\mathbf{q}^{(1)}| \ldots |\mathbf{q}^{(N_c)}]$ provides a convenient matrix factorization

$$\mathbf{\Phi} = QR$$

where R is upper triangular and Q is orthogonal. Forming a matrix $\hat{Q}$ which consists of the columns of Q which span $\mathcal{R}(\mathbf{\Phi})$ we have a new projection matrix,

$$P = \hat{Q}\hat{Q}^T$$

Now $\mathbf{\Phi}\mathbf{w}^* = P\mathbf{b}$ becomes

$$\hat{Q}\hat{R}\mathbf{w}^* = \hat{Q}\hat{Q}^T\mathbf{b}$$

Again, since $\hat{Q}^T\hat{Q} = I$ and $\hat{R}$ is invertible, we solve for

$$\mathbf{w}^* = \hat{R}^{-1}\hat{Q}^T\mathbf{b}$$

Thus, the form of the pseudoinverse we obtain from the QR decomposition is

$$\mathbf{\Phi}^\dagger = \hat{R}^{-1}\hat{Q}^T$$

7.4 DESCENT METHODS FOR COMPUTING THE WEIGHTS

Now we assume that our ensemble of input-output pairs lie in $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. Then we can write the error of the expansion for the μth pattern, or the *on-line* error, as

$$E_\mu(W) = \frac{1}{2}\sum_{k=1}^{m}(f_k(\mathbf{x}^{(\mu)}) - \tilde{f}_k(\mathbf{x}^{(\mu)}; W))^2$$

where $(W)_{mn} = w_{mn}$, i.e., the matrix consisting of all the weights in the RBF. The total, or *batch* error is the sum of the error over all the patterns, i.e.,

$$E(W) = \sum_{\mu=1}^{P} E_\mu(W) \tag{7.11}$$

We seek to minimize the function $E(W)$. Writing the matrix of parameters as a vector by concatenating the columns of W, i.e., $\mathbf{w} = \text{vec}(W)$, we recall that the direction of maximum decrease of the function $E(\mathbf{w})$ is in the direction of $-\nabla E(\mathbf{w})$, i.e., the negative gradient of the error w.r.t. the weight vector.

Thus our iteration, known as *steepest descent*, can be constructed as

$$\mathbf{w}^{n+1} = \mathbf{w}^n - \epsilon_n \nabla E(\mathbf{w}^n)$$

The length of the step in the direction of the negative gradient ϵ_n is referred to as the step-size, or using neural network terminology, the *learning* rate. The learning rate may be simply fixed during the iteration. Alternatively, the optimal size for ϵ_n may be calculated analytically at each step. We now develop the theory for this quadradic optimization problem.

The alogrithm of steepest descent requires that the gradient be calculated at each iteration. It can easily be shown that the gradient of the error function for a single pattern is

$$\frac{\partial E_\mu}{\partial w_{km}} = -(f_k(\mathbf{x}^{(\mu)}) - \tilde{f}_k(\mathbf{x}^{(\mu)}))\phi(\|\mathbf{x}^{(\mu)} - \mathbf{c}_m\|).$$

For the ensemble of patterns the total gradient is then

$$\frac{\partial E}{\partial w_{km}} = \sum_{\mu=1}^{P} \frac{\partial E_\mu}{\partial w_{km}}.$$

7.4.1 The Least Mean Square (LMS) Alorithm

In this section we focus on the mapping linear model and the associated least mean square (LMS) algorithm. The LMS algorithm is due to [86]; see also [28]. Here we focus on the connection with the computation of the weights in the RBF problem.

Again, we begin with a collection of input-output data $\{\mathbf{x}^{(\mu)}, f^{(\mu)}\}$. For each pattern $\mathbf{x}^{(\mu)}$ the desired response of the system is $f^{(\mu)}$. The form of the output is modeled by

$$\tilde{f}^{(\mu)} = \mathbf{w}^T \mathbf{x}^{(\mu)}$$

To satisfy the interpolation condition we require that

$$\begin{pmatrix} \mathbf{f}^{(1)} \\ \vdots \\ \mathbf{f}^{(P)} \end{pmatrix} = \begin{pmatrix} - & \mathbf{x}^{(1)} & - \\ & \vdots & \\ - & \mathbf{x}^{(P)} & - \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_{N_c} \end{pmatrix}$$

The RBF problem is a special case of this equation with $\Phi)_{ij} = x_j^{(i)}$.

The mean-square error of the model is defined as

$$\begin{aligned} E(\mathbf{w}) &= \|\Phi\mathbf{w} - \mathbf{f}\|_2^2 \\ &= (\Phi\mathbf{w} - \mathbf{f})^T(\Phi\mathbf{w} - \mathbf{f}) \\ &= (\Phi\mathbf{w})^T\Phi\mathbf{w} - 2\mathbf{f}^T\Phi\mathbf{w} + \mathbf{f}^T\mathbf{f} \\ &= \mathbf{w}^T\Phi^T\Phi\mathbf{w} - 2\mathbf{f}^T\Phi\mathbf{w} + \mathbf{f}^T\mathbf{f} \end{aligned}$$

This error may be simplified to be

$$E(\mathbf{w}) = \mathbf{w}^T C\mathbf{w} - 2\mathbf{w}^T\mathbf{p} + \mathbf{f}^T\mathbf{f} \tag{7.12}$$

where $\mathbf{p} = \Phi^T\mathbf{f}$ is the *cross-correlation* term, and $C = \Phi^T\Phi$ is the ensemble averaged covariance matrix. The optimum linear filtering problem is then to find the unique weight vector $\mathbf{w}^*$ such that the mean square error is a minimum. The solution of this problem is sometimes referred to as the Wiener filter [28].

To produce the minimum mean-square error we differentiate the error term with respect to the free parameters, i,e., the weight vector $\mathbf{w}$.

$$\frac{\partial E}{\partial \mathbf{w}} = 0 + 2C\mathbf{w}^* - 2\mathbf{p} = 0 \tag{7.13}$$

Hence, the weight vector $\mathbf{w}^*$ which determines the minimum error is given by the solution to the problem

$$C\mathbf{w}^* = \mathbf{p} \tag{7.14}$$

which is known as the Wiener-Hopf equation. Note that it see equivalent to the normal equations. In the simplest case, the ensemble averaged correlation matrix will have full rank and we may write the solution simply as

$$\mathbf{w}^* = C^{-1}\mathbf{p}$$

We will see in what follows that it is useful to consider an *adaptive* algorithm for solving this problem. The necessary link is provided by the following theorem:

Theorem 7.1. *If C is a symmetric, positive definite matrix, then the quadratic form*

$$q(\mathbf{w}) = \frac{1}{2}(\mathbf{w}, C\mathbf{w}) - (\mathbf{w}, \mathbf{b})$$

has a global minimum at $\mathbf{w}^*$ where

$$C\mathbf{w}^* = \mathbf{b}$$

Proof. Rewrite $q(\mathbf{w})$ as

$$q(\mathbf{w}) = \frac{1}{2}(\mathbf{w} - \mathbf{x})^T C(\mathbf{w} - \mathbf{x}) + \mathbf{w}^T(C\mathbf{x} - \mathbf{b}) - \frac{1}{2}\mathbf{x}^T C\mathbf{x} \tag{7.15}$$

This expression is true for any $\mathbf{x}$. In particular, let $\mathbf{x}$ be the least squares solution, i.e., $\mathbf{x} = \mathbf{w}^*$ where $C\mathbf{w}^* = \mathbf{b}$. Then

$$q(\mathbf{w}) = \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T C(\mathbf{w} - \mathbf{w}^*) - \frac{1}{2}\mathbf{w}^* C\mathbf{w}^*$$

Since the matrix C is positive definite it follows that

$$(\mathbf{w}-\mathbf{w}^*)^T C(\mathbf{w}-\mathbf{w}^*) \geq 0$$

with equality iff

$$\mathbf{w}=\mathbf{w}^*$$

Hence

$$\min q(\mathbf{w}) = q(\mathbf{w}^*) = -\frac{1}{2}(\mathbf{w}^*)^T A\mathbf{w}^*$$

A Geometrical Picture

We seek to minimize the quadratic function

$$q(\mathbf{w}) = \mathbf{w}^T C\mathbf{w} - \mathbf{w}^T\mathbf{b}$$

Geometrically we may interpret the function $q(\mathbf{w})$ as a high dimensional bowl with minimum value $q(\mathbf{w}^*)$, i.e., $\mathbf{w}^*$ is the point at the bottom of the bowl. This geometry is extremely attractive in view of the fact that there are *no local minima.*

The geometrical picture becomes somewhat clearer if we first apply a transformation which diagonalizes the operator C, i.e., $V^TCV=\Lambda$. Given such a V, introduce the change of coordinates

$$\mathbf{w}=V\mathbf{z}$$

Then the quadratic form becomes

$$q(\mathbf{z}) = (V\mathbf{z})^T CV\mathbf{z} - (V\mathbf{z})^T\mathbf{z}$$

Multiplying out, and setting $\mathbf{b}' = V^T\mathbf{b}$ gives

$$q(\mathbf{z}) = \mathbf{z}^T\Lambda\mathbf{z} - \mathbf{z}^T\mathbf{b}' \qquad (7.16)$$

By completing the square it can be shown that the linear term may be removed from the equation $q(\mathbf{z}) = k$, with k a constant, specifically

$$q(\tilde{\mathbf{z}}) = \tilde{\mathbf{z}}^T\Lambda\tilde{\mathbf{z}} = \tilde{k} \qquad (7.17)$$

Recalling the assumption that $\lambda_i > 0$, i.e., A is positive definite, the equation defining the error

$$q(\tilde{\mathbf{z}}) = \sum_i \lambda_i \tilde{z}_i^2 \qquad (7.18)$$

is seen to describe a convex bowl with global minimum at the point $\tilde{\mathbf{z}}=0$. Furthermore, the level sets defined by

$$\sum_i \lambda_i \tilde{z}_i^2 = \tilde{k}$$

Fig. 7.5 The learning surface (in optimal coordinates) $q(\mathbf{w}) = \mathbf{w}^T A\mathbf{w}$. It has a unique minimum at the origin.

indicate the directions perpendicular to the steepest descent directions. In fact, the optimal step size ϵ_n can be seen to be the distance from the point where the descent direction is orthogonal to the level curve to the point where it is first tangent to a level curve, see Figure 7.5.

This procedure for transforming variables is investigated in more detail in the exercises.

Convergence of Steepest Descent

The new equation for the quadratic form derived above provides some insight into the nature of the convergence of the method of steepest descent. Again, the steepest descent algorithm is based on the iteration

$$\mathbf{w}^{n+1} = \mathbf{w}^n - \epsilon\nabla q(\mathbf{w}^n)$$

Given

$$q(\mathbf{w}) = \mathbf{w}^T\Lambda\mathbf{w}$$

it follows that the gradient

$$\nabla q(\mathbf{w}) = 2\Lambda\mathbf{w}$$

Now, in this coordinate system, the adaption of the weights is carried out using

$$\mathbf{w}^{n+1} = \mathbf{w}^n - 2\epsilon\Lambda\mathbf{w}^n \tag{7.19}$$

$$= (\mathbf{I} - 2\epsilon\Lambda)\mathbf{w}^n \tag{7.20}$$

We may solve this equation as

$$\mathbf{w}^n = (\mathbf{I} - 2\epsilon\Lambda)^n\mathbf{w}^0 \tag{7.21}$$

This matrix equation is uncoupled, i.e., the i'th equation may be written

$$\mathbf{w}^n = (1 - 2\epsilon\lambda_i)^n \mathbf{w}^0$$

The iterations will be bounded if

$$|1 - 2\epsilon\lambda_i| < 1$$

for all i. Thus, the convergence condition may be shown to be

$$0 < \epsilon < \frac{1}{\lambda_{\max}}$$

where $\lambda_{\max}$ is defined to be the largest eigenvalue of A. If this condition is satisfied, we conclude that

$$\lim_{n\to\infty} \mathbf{w}^n = 0$$

which is in fact the optimal solution as discussed above.

7.4.2 Exact Line Search For Steepest Descent

Now we turn our attention computing the global minimum of

$$q(\mathbf{w}) = \mathbf{w}^T C\mathbf{w} - \mathbf{w}^T\mathbf{b}$$

which, by Theorem 7.1, provides the unique solution to $C\mathbf{w} = \mathbf{b}$. Our purpose is to extend the minimization technique of steepest descent described in the previous section by computing the learning rate analytically at each time step. As before, the iteration proceeds

$$\mathbf{w}^{n+1} = \mathbf{w}^n - \epsilon_n \nabla q(\mathbf{w}^n) \tag{7.22}$$

For brevity we write $\mathbf{g}^n \equiv \nabla q(\mathbf{w}^n)$.

We may define ϵ_n to be the value which defines the minimum of

$$\psi(\epsilon) = q(\mathbf{w}^{n+1}) = q(\mathbf{w}^n - \epsilon\mathbf{g}^n)$$

Differentiating

$$\psi'(\epsilon_n) = 0 = (\mathbf{g}^{n+1})^T(-\mathbf{g}^n) \tag{7.23}$$

The gradient has a simple formula in this case, i.e.,

$$\mathbf{g}^{n+1} = C\mathbf{w}^{n+1} - \mathbf{b} \tag{7.24}$$

Substituting this equation into the expression for $\psi'(\epsilon_n) = 0$ produces

$$(C\mathbf{w}^{n+1} - \mathbf{b})^T(-\mathbf{g}^n) = 0$$

Fig. 7.6 Geometrical interpretation of steepest descent. The descent direction is initially orthogonal to the level curve and proceeds until it is tangent to a level curve. The distance between these two points is defined analytically by ϵ_n in Equation (7.25).

Multiplying out and substituting Equation (7.22) for $\mathbf{w}^{n+1}$ gives

$$-C(\mathbf{w}^n - \epsilon_n \mathbf{g}^n)^T \mathbf{g}^n + \mathbf{b}^T \mathbf{g}^n = 0$$

Solving this equation for ϵ_n gives

$$\epsilon_n = \frac{(\mathbf{g}^n)^T \mathbf{g}^n}{(\mathbf{g}^n)^T C \mathbf{g}^n} \tag{7.25}$$

This calculation has produced an analytical solution for the learning rate and is referred to in the literature as an exact line search.

Proposition 7.1. *The method of steepest descent moves in perpendicular steps, i.e.,*

$$(\mathbf{w}^{n+1} - \mathbf{w}^n)^T (\mathbf{w}^{n+2} - \mathbf{w}^{n+1}) = 0$$

Proof. Given $\mathbf{w}^{n+1} = \mathbf{w}^n - \epsilon_n \mathbf{g}^n$ it follows

$$(\epsilon_n \mathbf{g}^n)^T (-\epsilon_{n+1} \mathbf{g}^{n+1}) = 0$$

i.e.,

$$(\mathbf{g}^n)^T \mathbf{g}^{n+1} = 0$$

The previous statement is true from Equation (7.23). □

The geometrical progression of search directions is displayed in Figure 7.6.

Theorem 7.2. *If* $\mathbf{w}^{n+1}$ *and* $\mathbf{w}^n$ *are successive search directions in a sequence defined by steepest descent, and if* $\nabla q(\mathbf{w}^n) \neq 0$, *then*

$$q(\mathbf{w}^{n+1}) < q(\mathbf{w}^n)$$

For a proof see [65].

Theorem 7.3. *Let $\lambda_{\max}$ and $\lambda_{\min}$ be the largest and smallest eigenvalues of C. Then*

$$q(\mathbf{w}^{n+1}) \le \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}\right)^2 q(\mathbf{w}^n)$$

For a proof see [56]. This theorem suggests that if $\lambda_{\min} = \lambda_{\max}$ then there is convergence in 1-step. In this situation the level curves are concentric hyperspheres. On the other hand, if $\lambda_{\min} << \lambda_{\max}$ then convergence is very slow. This is evident by rewriting the convergence factor

$$C = \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}\right)^2 = \left(\frac{\lambda_{\max}/\lambda_{\min} - 1}{\lambda_{\max}/\lambda_{\min} + 1}\right)^2$$

So we see convergence is slowed as $\lambda_{\max}/\lambda_{\min}$ increases.

7.4.3 The Conjugate Gradient (CG) Algorithm

Given a symmetrix matrix C, two vectors $\mathbf{d}, \mathbf{e}$ are said to be *conjugate* w.r.t. C if

$$\mathbf{d}^T C \mathbf{e} = 0$$

A set of vectors is conjugate of all of the pairs of vectors in the set are mutually conjugate. Now we consider an alternative to steepest descent which constructs a set of conjugate search directions. It can be shown that, with infinite precision, this method produces an iteration which converges to the solution $\mathbf{w}^*$ of $C\mathbf{w} = \mathbf{b}$ in n iterations. In practice, numerical roundoff slows actual convergence rates. This section follows [56].

Starting at any $\mathbf{w}^0$ define the first descent direction as

$$\mathbf{d}^0 = -\mathbf{g}^0 = \mathbf{b} - C\mathbf{w}^0$$

The general iteration proceeds as

$$\mathbf{w}^{n+1} = \mathbf{w}^n + \alpha_n \mathbf{d}^n$$

where

$$\alpha_n = -\frac{(\mathbf{g}^n)^T \mathbf{d}^n}{(\mathbf{d}^n)^T C \mathbf{d}^n}$$

and

$$\mathbf{d}^{n+1} = -\mathbf{g}^{n+1} + \beta_n \mathbf{d}^n$$

$$\beta_n = \frac{(\mathbf{g}^{n+1})^T C \mathbf{d}^n}{(\mathbf{d}^n)^T C \mathbf{d}^n}$$

where $\mathbf{g}^n = C\mathbf{w}^n - \mathbf{b}$.

Theorem 7.4. *The conjugate gradient algorithm is a conjugate direction method. If it does not terminate at $\mathbf{w}^n$, then*

1. *span*$\{\mathbf{g}^0, \mathbf{g}^1, \ldots, \mathbf{g}^n\} =$ *span*$\{\mathbf{g}^0, C\mathbf{g}^0, \ldots, C^n\mathbf{g}^0\}$
2. *span*$\{\mathbf{d}^0, \mathbf{d}^1, \ldots, \mathbf{d}^n\} =$ *span*$\{\mathbf{d}^0, C\mathbf{d}^0, \ldots, C^n\mathbf{d}^0\}$
3. $(\mathbf{d}^n)^T C\mathbf{d}^i = 0$ *for* $i \leq n-1$.
4. $\alpha_n = -\frac{(\mathbf{g}^n)^T\mathbf{g}^n}{(\mathbf{d}^n)^T C\mathbf{d}^n}$
5. $\beta_n = \frac{(\mathbf{g}^{n+1})^T C(\mathbf{g}^{n+1})^T}{(\mathbf{g}^n)^T C\mathbf{g}^n}$

7.4.4 The On-Line LMS Alorithm

In many applications, the data set is either not available all at once, or the data is itself nonstationary, i.e., its statistics such as the mean vary with time. In this case, it is desirable to adapt the procedure in the previous section, since it requires that the weight be updated each iteration based on complete knowledge of the patterns. The basis for this new procedure is providing instantaneous estimates for required quantities.

As previously, we will use the μ_n to denote the pattern to be trained on at the n'th iteration.

We begin by estimating the error function (in the original coordinates) due to a single point

$$\begin{aligned} E_\mu(\mathbf{w}) &= (f^{(\mu)} - \tilde{f}^{(\mu)})^2 \\ &= (f^{(\mu)})^2 - 2\hat{\mathbf{w}}^T\hat{\mathbf{p}} + \hat{\mathbf{w}}^T\hat{C}\hat{\mathbf{w}} \end{aligned}$$

$$\hat{C} = \mathbf{x}^{(\mu_n)}\mathbf{x}^{(\mu_n)}$$

and the cross-correlation term is estimated as

$$\hat{\mathbf{p}} = f^{(\mu_n)}\mathbf{x}^{(\mu_n)}.$$

The instantaneaous gradient is then defined as

$$\nabla_{\hat{\mathbf{w}}}\hat{E}(\hat{\mathbf{w}}) = 2\hat{C}\hat{\mathbf{w}} - 2\hat{\mathbf{p}}.$$

Steepest descent is then

$$\begin{aligned} \hat{\mathbf{w}}^{n+1} &= \hat{\mathbf{w}}^n + 2\epsilon(\hat{\mathbf{p}} - \hat{C}\hat{\mathbf{w}}^n) \\ &= \hat{\mathbf{w}}^n + 2\epsilon(f^{(\mu_n)}\mathbf{x}^{(\mu_n)} - \mathbf{x}^{(\mu_n)}\mathbf{x}^{(\mu_n)^T}\hat{\mathbf{w}}^n) \end{aligned}$$

Simplifying

$$\hat{\mathbf{w}}^{n+1} = \hat{\mathbf{w}}^n + \epsilon(f^{(\mu_n)} - \tilde{f}^{(\mu_n)})\mathbf{x}^{(\mu_n)}.$$

The factor of 2 has been absorbed into the learning rate ϵ. The least-mean-square algorithm may now be summarized as follows:

LMS Algorithm

```
1. Initialize the weights $\hat{\mathbf{w}}(0) = 0$.
2. Select the $n$'th pattern, $\mathbf{x}^{(\mu_n)}$ at random.
3. Compute the output of the network $\tilde{f}^{(\mu_n)}$
   corresponding to the selected pattern.
4. Compute the instantaneous error $e^n = f^{(\mu_n)} - \tilde{f}^{(\mu_n)}$.
5. Update the weight vector according to the rule
       $\hat{\mathbf{w}}^{n+1} = \hat{\mathbf{w}}^n + \epsilon e^n \mathbf{x}^{(\mu_n)}$.
```

7.5 COMPUTING THE RBF CENTERS

The determination of the weight parameters may be accomplished either via direct or iterative methods for least squares problems. Similarly, there are several options available for determining the placement of the RBFs withing the input domain, each with different characteristics. We consider three basic methods for determining the location of the centers $\{\mathbf{c}_m\}$ of the radial basis functions, i.e.,

1. centers on a lattice
2. random center selection
3. nonlinear optimization
4. clustering

Picking centers uniformly on the input domain is useful in small dimension. This technique is practial for 1, 2, and 3-dimensional lattices if the size of the domain is not to large.

Random center selection operates simply by picking a subset of the training data as locations for the centers. The rationale for this procedure is that the centers should reflect the distribution of the data. Randomly selecting data points is an extremely efficient, if somewhat crude, method for achieving an estimate of the data distribution.

After describing the nonlinear optimization approach in this section, we will begin our discussion of clustering methods. First, however, we consider how knowledge of the cluster centers may be used to determine local paramaters for the RBFs.

Setting Local Parameters

The local RBF expansions require that the size of the domain of the RBF be set. This can be done using nonlinear optimization but this is rather unappealing given the computational expense. Another approach is to estimate a local length scale given a collection of RBF centers, or data clusters, $\{\mathbf{c}_l\}$.

Such a length scale may be derived from the distances between clusters. In particular, the distance from cluster $\mathbf{c}_m$ to the nearest cluster $d^*(\mathbf{c}_m)$, i.e.,

$$d^*(\mathbf{c}_m) = \min_{m \neq n} \|\mathbf{c}_m - \mathbf{c}_n\|$$

provides a measure of separation which can be used to set the widths of Gaussians. The radius can be taken as a multiple α of this distance to the nearest neighboring center, i.e.,

$$r_m = \alpha d^*(\mathbf{c}_m)$$

or, alternatively, a global average may be used

$$r = \frac{1}{N_c} \sum_{m=1}^{N_c} d^*(\mathbf{c}_m)$$

7.5.1 Nonlinear Optimization

As in Section 7.4 which concerned iterative methods for finding weights, the RBF enters may be determined by a gradient descent method. Now we view the error function as

$$E_\mu(\mathbf{c}) = \frac{1}{2} \sum_{k=1}^{m} (f_k^{(\mu)} - \tilde{f}_k(\mathbf{x}^{(\mu)}; \mathbf{c}))^2$$

where $\mathbf{c}$ is the vector consisting of the concatenated centers $\{\mathbf{c}_m\}$. Both the conjugate gradient method and the steepest descent method require the computation of the gradient. It is left to the reader to verify that

$$\frac{\partial E_\mu}{\partial \mathbf{c}_j} = \phi'(\|\mathbf{x}^{(\mu)} - \mathbf{c}_j\|) \frac{\mathbf{x}^{(\mu)} + \mathbf{c}_j}{\|\mathbf{x}^{(\mu)} - \mathbf{c}_j\|} \sum_{k=1}^{m} (f_k^{(\mu)} - \tilde{f}_k(\mathbf{x}^{(\mu)})) w_{kj} \tag{7.26}$$

Partially Linear Optimization

The optimization procedures described above for the centers and in Section 7.4 for the weights may be combined into a concatenated gradient direction. In this setting, the error function is written

$$E_\mu(\boldsymbol{\xi}) = \frac{1}{2} \sum_{k=1}^{m} (f_k^{(\mu)} - \tilde{f}_k(\mathbf{x}^{(\mu)}; \boldsymbol{\xi}))^2$$

where $\boldsymbol{\xi} = (\mathbf{w}^T, \mathbf{c}^T)^T$. The on-line partially linear steepest descent iteration is given by be

$$\boldsymbol{\xi}^{n+1} = \boldsymbol{\xi}^n - \epsilon_n \frac{\partial E_\mu(\boldsymbol{\xi}^n)}{\partial \boldsymbol{\xi}^n}$$

while the batch iteration is given by

$$\boldsymbol{\xi}^{n+1} = \boldsymbol{\xi}^n - \epsilon_n \frac{\partial E(\boldsymbol{\xi}^n)}{\partial \boldsymbol{\xi}^n}$$

where again $E(\xi) = \sum_\mu E_\mu(\xi)$.

7.6 CENTER SELECTION VIA DATA CLUSTERING

As an alternative to selecting RBF centers randomly, or via a nonlinear optimization problem, it is possible to use one of a variety of clustering algorithms which determine the center locations based on the domain data alone. Given the clustering procedure generally takes place independently of the determination of the weights it is often referred to as an *off-line* calculation.

There are many fast and effective approaches for constructing a local covering (partition) of the input patterns. We remark that the requirements for our application are rather forgiving, in particular, we do not require that the covering be unique or even optimal.

Clustering algorithms are typically divided into two distinct classes:

- hierarchical
- non-hierarchical

Hierarchical clustering methods create a small number of clusters which are then clustered, i.e., forming clusters within clusters. Our focus will be on non-hierarchical methods where the data is not labelled according to class membership.

Basic Framework of Clustering Algorithms

- **Initialization:** Choose an initial covering of the patterns in terms of k-clusters.
  - choose the number of clusters k *a priori.*
  - let the algorithm determine the best k.
- **Iteration:** Adapt cluster memberships to improve the the quality of the covering of the patterns.
  - online update : modify cover after one test pattern.
  - batch update : modify cover after testing all patterns.

Fig. 7.7 A Simple Competitive Learning Network

7.6.1 Competitive Learning

Given a collection of data points $\{x^{(\mu)}\}$ in $\mathbb{R}^n$ we seek to identify a collection of centers $\{\mathbf{c}_i\}$ which form a covering of the data.

The first approach we consider is known as *simple competitive learning.* Given an initial set of centers $\{\mathbf{c}_i\}$, typically selected at random, the algorithm updates the centers so that their positions reflect the distribution of the data. Assume the centers are identified by their indices $i \in \mathcal{I}$.

For each data point in the ensemble, e.g., $\mathbf{x}^{(\mu)}$ the center closest to it is identified and called the *winning* center $\mathbf{c}_{i'}$. For the pattern $\mathbf{x}^{(\mu)}$ the winning center must satisfy the relationship

$$\|\mathbf{x}^{(\mu)} - \mathbf{c}_{i'}\| \leq \|\mathbf{x}^{(\mu)} - \mathbf{c}_i\|, \quad \text{for all}\, i \in \mathcal{I} \tag{7.27}$$

After the winning center is identified it is moved closer to the pattern $\mathbf{x}^{(\mu)}$ for which it was identified as being the winner, i.e.,

$$\mathbf{c}_{i'}^{n+1} = \mathbf{c}_{i'}^{n} + \epsilon(\mathbf{x}^{(\mu)} - \mathbf{c}_{i'}^{n}) \tag{7.28}$$

Simple Competitive Learning Algorithm

1. Initialize a set of center vectors $\{\mathbf{c}_i\}, i \in \mathcal{I}$.
2. Present a pattern $\mathbf{x}^{(\mu)}$ to the network.
3. Determine the winning center vector $\mathbf{c}_i'$.
4. Update the winning center vector only, using

   $$\delta\mathbf{c}_{i'} = \epsilon(\mathbf{x}^{(\mu)} - \mathbf{c}_{i'})$$

   where ϵ is the learning rate.
5. Present another pattern to the network and repeat.

Remarks:

- If any center is initially "far away" it may never win. Update losers (*dead centers*) with smaller learning rate.
- Best results are obtained by normalizing the patterns and centers.
- Eliminate and/or add clusters as algorithm adapts.

We will examine the learning rule both from the geometry of the final centers, and by considering the minimization of an appropriate cost function.

Constant Learning Rate

It is interesting to examine the case where the learning rate is taken to be a constant $\epsilon = \epsilon_0$. Define the sequence

$$\{\mathbf{x}^{(\mu_0)}, \mathbf{x}^{(\mu_1)}, \ldots, \mathbf{x}^{(\mu_{n-1})}\}$$

to be a sequence of input patterns which has the winning centers

$$\{\mathbf{c}^0, \mathbf{c}^1, \ldots, \mathbf{c}^{n-1}\}$$

where each iteration the winner is determined according to Equation (7.27). The competitive learning update gives

$$\mathbf{c}^1 = \mathbf{c}^0 + \epsilon_0(\mathbf{x}^{(\mu_0)} - \mathbf{c}^0)$$

For the n'th iteration

$$\mathbf{c}^n = \mathbf{c}^{n-1} + \epsilon_0(\mathbf{x}^{(\mu_{n-1})} - \mathbf{c}^{n-1})$$

It follows that

$$\mathbf{c}^n = (1-\epsilon_0)^n \mathbf{c}^0 + \epsilon_0 \sum_{j=1}^{n} (1-\epsilon_0)^{j-1} \mathbf{x}^{(\mu_{n-j})} \qquad (7.29)$$

It is interesting to observe that, based on the above equation, the most recent winners contribute more to the centers then past winners. As such, this approach will adjust rapidly to new patterns and is appropriate for non-stationary data.

Example 7.3. *A Problem in 1-dimension.* Let $x^{(1)} = 1$ and $x^{(2)} = 2$. We "randomly" choose the initial centers $c_1^0 = 3$ and $c_2^0 = 0$. To start, we select the pattern $x^{(2)} = 2$ (at random) and determine its winning center. Clearly $c_1^0 = 3$ is the closest center and is updated

$$c_1^1 = c_1^0 + \epsilon_0(x^{(2)} - c_1^0)$$

If we continue to present the same pattern to the network, then the same center is the winner and is updated according to

$$c_1^{n+1} = c_1^n + \epsilon_0(2 - c_1^n)$$

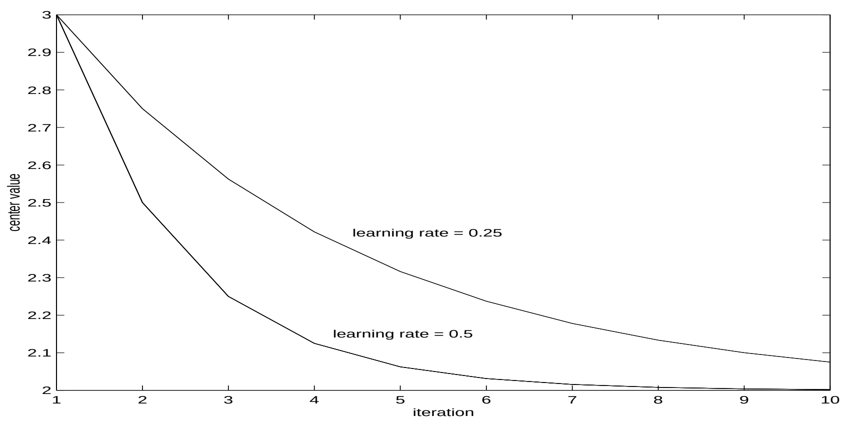

Fig. 7.8 The value of the cluster nearest $x^{(2)} = 2$ as updated by the competitive clustering algorithm.

If we take $\epsilon = .5$, then the competitive learning algorithm produces the sequence

$$c_1 = \{3.0, 2.50, 2.250, 2.1250, 2.06250, 2.031250, \ldots\}$$

which is approaching the data point $x^{(2)}$. This is too simple a case to actually be very interesting. We could have arrived at the result immediately by taking the large learning rate $\epsilon = 1$. However, it does reveal an interesting property of algorithm 7.6.1. The centers tend to migrate to the patterns and eventually reflect their distribution.

k-means centers

The centering of the new patterns in competitive learning may be decreased by decreasing, or *annealing*, the learning rate. A convenient choice for the learning rate is

$$\epsilon_n = \frac{1}{n}$$

An interesting consequence of this choice is that the resulting center vector takes on the value of the mean of the patterns for which it is the winner.

Again, given the sequence of k patterns

$$\{\mathbf{x}^{(\mu_1)}, \mathbf{x}^{(\mu_2)}, \ldots, \mathbf{x}^{(\mu_k)}\}$$

for which the center $\mathbf{c}$ is a winner it follows

$$\mathbf{c}^1 = \mathbf{c}^0 + 1 \cdot (\mathbf{x}^{(\mu_1)} - \mathbf{c}^0) = \mathbf{x}^{(\mu_1)}$$

For the second iteration

$$\mathbf{c}^2 = \mathbf{c}^1 + \frac{1}{2} \cdot (\mathbf{x}^{(\mu_2)} - \mathbf{c}^1) = \frac{\mathbf{x}^{(\mu_1)} + \mathbf{x}^{(\mu_2)}}{2}$$

The claim is that this averaging procedure holds true for each iteration, i.e.,

$$\mathbf{c}^k = \frac{1}{k} \sum_{j=1}^{k} \mathbf{x}^{(\mu_j)} \tag{7.30}$$

This may be established using induction. Assuming that Equation (7.30) is true for the k'th iteration we will show that it is true for iteration $k+1$. By definition,

$$\begin{aligned}
\mathbf{c}^{k+1} &= \mathbf{c}^k + \frac{1}{k+1}(\mathbf{x}^{(\mu_{k+1})} - \mathbf{c}^k) \\
&= (1 - \frac{1}{k+1})\mathbf{c}^k + \frac{1}{k+1}\mathbf{x}^{(\mu_{k+1})} \\
&= \frac{k}{k+1}\frac{1}{k}\sum_{j=1}^{k}\mathbf{x}^{(\mu_j)} + \frac{1}{k+1}\mathbf{x}^{(\mu_{k+1})} \\
&= \frac{1}{k+1}\sum_{j=1}^{k+1}\mathbf{x}^{(\mu_j)}
\end{aligned}$$

which proves Equation (7.30).

Clustering via competitive learning.

It is interesting to examine the regions defined by the winning centers. We shall see that they form a well-defined partition of the input space known as a Voronoi Tessellation.

Consider an input pattern $\mathbf{x}^{(\mu)}$. It's winning center $\mathbf{c}_{i'}$ satisfes (by definition) the relationship

$$\|\mathbf{c}_{i'} - \mathbf{x}^{(\mu)}\| < \|\mathbf{c}_i - \mathbf{x}^{(\mu)}\|$$

for all $i \in \mathcal{I}, i \neq i'$. Now, if instead of taking $\mathbf{x}^{(\mu)}$ as a particular pattern, we consider it to be a variable in the input space, we can ask what is the shape of the region of points $\mathbf{x}$ which have the winning center $\mathbf{c}_i$?

First, consider the simple case in the plane with just 2 centers vectors $\mathbf{c}_1$ and $\mathbf{c}_2$. The locus of points which is equidistant from both centers is a straight line. Therefore, the set of points closest to $\mathbf{c}_1$ is given by the half-plane

$$\{\mathbf{x} \,|\, \|\mathbf{c}_1 - \mathbf{x}\| < \|\mathbf{c}_2 - \mathbf{x}\|\}$$

with the other half of the plane being the set of points which have $\mathbf{c}_2$ as their winning center

$$\{\mathbf{x} \,|\, \|\mathbf{c}_2 - \mathbf{x}\| < \|\mathbf{c}_1 - \mathbf{x}\|\}.$$

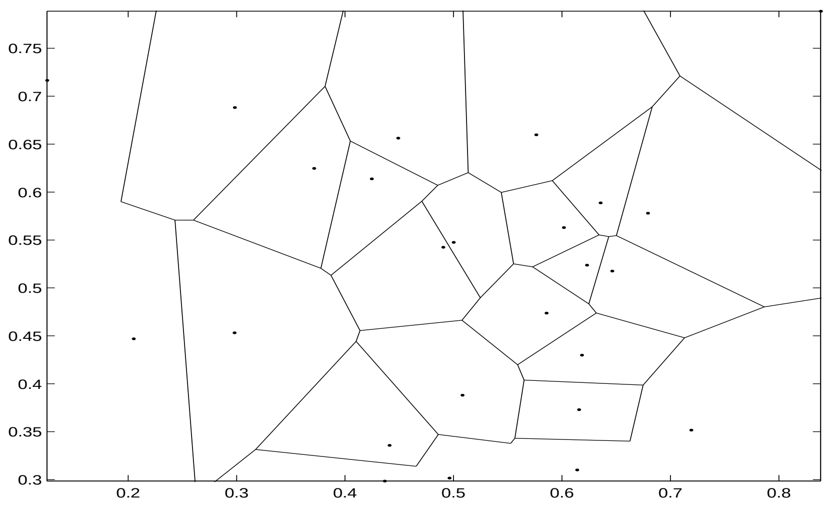

Fig. 7.9 The Voronoi diagram for 25 points in the plane.

Thus, we see that every additional center adds as many new linear boundaries as their are old centers. Hence, the input space is partitioned into polygons known as *Voronoi regions.* This idea may be generalized as follows:

Definition 7.1. *A Voronoi region V_i associated with the center $\mathbf{c}_i$ is the set of points for which $\mathbf{c}_i$ is the nearest center vector, i.e.,*

$$V_i = \{\mathbf{x} \in \mathbb{R}^n \, : \, i = \arg\min_{j \in \mathcal{I}} \|\mathbf{x} - \mathbf{c}_j\|\}$$

where $\mathcal{I}$ is the set of center indices.

A Cost Function for Competitive Learning

To study the convergence properties and to compare the quality of different clusters we introduce the cost function

$$E(c_{mj}) = \frac{1}{2} \sum_{mj\mu} M_m^\mu (x_j^{(\mu)} - c_{mj})^2$$

where

$$M_m^\mu = \{ \begin{array}{ll} 1 & \mathbf{c}_m \text{ is a winner} \\ 0 & \text{else.} \end{array}$$

Definition 7.2. *M_m^μ is referred to as the cluster membership matrix.*

The cluster membership matrix evolves in the course of learning, see [29] for further discussion.

The batch gradient of the cost function is then given by

$$\delta c_{mj} = -\epsilon \frac{\partial E}{\partial c_{mj}} = \epsilon \sum_{\mu} M_m^{\mu} (x_j^{(\mu)} - c_{mj})$$

The individual (on-line) updates are applied to the winning centers only, i.e.,

$$\delta \mathbf{c}_{m'} = \epsilon (\mathbf{x}^{(\mu)} - \mathbf{c}_{m'})$$

which is seen to be the competitive learning rule. In other words, the competitive learning rule is performing a simple gradient descent on the cost function defined above and will converge to a (possibly local) minimum.

To improve convergence, the learning rate ϵ may be annealed as a function of the iteration.

7.6.2 The LBG Algorithm

Recall that the Voronoi set V_i defines the *region* in space which has $\mathbf{c}_i$ as its closest, or winning, center. It is also useful to define the set of points in a data set X for which i is the winning index. This is known as the *Voronoi set.*

Definition 7.3. *A Voronoi set S_i is a subset of points of a data set X for which index i is the winner, i.e.,*

$$S_i = \{\mathbf{x} \in X \, : \, \|\mathbf{x} - \mathbf{c}_i\| \leq \|\mathbf{x} - \mathbf{c}_j\|, \, i \neq j\}$$

The LBG algorithm proceeds by defining a *distortion error* $E(X, \mathcal{I})$ based on the Voronoi sets.

Definition 7.4. *Given a data set X consisting of P points and a set of center indices $\mathcal{I}$, the distortion error is defined by*

$$E(X, \mathcal{I}) = \frac{1}{P} \sum_{i \in \mathcal{I}} \sum_{x \in S_i} \|\mathbf{x} - \mathbf{c}_i\|^2 \tag{7.31}$$

LBG Clustering Algorithm

```
• For each $i \in \mathcal{I}$ compute the Voronoi set $S_i$.
• Move the reference vectors according to

      $\mathbf{c}_i = \frac{1}{|S_i|} \sum_{x \in S_i} \mathbf{x}$

• repeat
```

The quantity $|S_i|$ is the number of points in the Voronoi set S_i.

Proposition 7.2. *A necessary condition for a set of centers $\{\mathbf{c}_i\}$ to minimize the distortion error $E(X, \mathcal{I})$ is that each reference vector satisfy the centroid condition*

$$\mathbf{c}_i = \frac{1}{|S_i|} \sum_{x \in S_i} \mathbf{x}$$

7.6.3 Topology Preserving Mappings

In this section we consider a very powerful nonlinear dimensionality reducing transformation known as the self-organizing feature map, or SOFM. It was introduced by the Finnish scientist Teuvo Kohonen [49] and is based on a simple modification to the competitive learning procedure developed in Section 7.6.1. While the *twist* in the algorithm may seem slight at first glance, an enormous number of papers have been written investigating it.

We begin our discussion of Kohonen's algorithm with a quick review of competitive learning.

1. Initialize a set of center vectors $\{\mathbf{c}_i\}, i \in \mathcal{I}$.
2. Present $\mathbf{x}^{(\mu)}$ to the network.
3. Determine the winning center vector $\mathbf{c}'_i$.
4. Update the winning center vector only, using
$$\delta \mathbf{c}_{i'} = \epsilon(\mathbf{x}^{(\mu)} - \mathbf{c}_{i'})$$
5. Repeat.

Now we make the following observation: If the centers $\{\mathbf{c}_i\}$ are trained according to the above algorithm, then no relationship is induced on the winning indices. In particular, if $\mathbf{x}^{(\mu)}$ has $i(\mu)$ as its winning index and $\mathbf{x}^{(\nu)}$ has $i(\nu)$ as its winning index and the patterns $\mathbf{x}^{(\mu)}$ and $\mathbf{x}^{(\mu)}$ are *near* to each other, there is no inherent connection imposed on the winning indices $i(\mu)$ and $i(\nu)$. In other words, *no information about the distance between* $\mathbf{x}^{(\mu)}$ *and* $\mathbf{x}^{(\mu)}$ *is available based on the distance between* $i(\mu)$ *and* $i(\nu)$. The self-organizing map from the center vectors to the indices changes all that, and provides a powerful tool to examine *nearness* in a high-dimensional space by examining the geometrical relationship, or nearness in a low-dimensional space, i.e., between indices.

The SOFM Algorithm

```
1. Initialize a set of center vectors $\{\mathbf{c}_i\}, i \in \mathcal{I}$.
2. Present $\mathbf{x}^{(\mu)}$ to the network.
3. Determine the winning center vector $\mathbf{c}_{i'}$.
4. Update *all* the center vectors using
       $\delta \mathbf{c}_i = \epsilon h(i - i')(\mathbf{x}^{(\mu)} - \mathbf{c}_{i'})$
5. Repeat.
```

The critical difference between the SOFM and competitive learning is the addition of the multiplicative function $h(x)$. This function is what creates an ordering of the indices based on the nearness of the winning centers. Typically, this function is taken to be a Mexican hat function, or a Gaussian function such as

$$h(x) = \exp(-x^2/2r^2).$$

First, note that

$$h(0) = 1$$

which means that the update rule for the winning index is unchanged. The modification to the learning is that all centers are updated now at a rate proportional to how near their index is to the winning index. If the distance between the indices $d(i, i')$ is small then the $h(\cdot)$ factor is relatively large. Similarly, centers which have indices which are not in the neighborhood of the winning index are update only very slightly, or not at all. I.e., if $d(i, i')$ is large, then the $h(\cdot)$ factor is very small.

Hence, the centers with indices near the winning index $i(\mu)$ will be pulled towards the pattern $\mathbf{x}^{(\mu)}$. This creates a *self-organization* since the centers with neighboring indices are being trained to respond to neighboring input patterns.

In summary, we say a map is self-organizing if neighboring patterns activate winning centers which have neighboring indices. Thus, if the following are true

- $\mathbf{x}^{(\mu)}$ has winning center $\mathbf{c}_{i(\mu)}$
- $\mathbf{x}^{(\nu)}$ has winning center $\mathbf{c}_{i(\nu)}$
- the distance $d(\mathbf{x}^{(\mu)}, \mathbf{x}^{(\nu)})$ is small

and the map is self-organizing, then the distance between the winning indices $d(i(\mu), i(\nu))$ is also small. In particular, it is the map from the pattern vector (to the center vector), to the winning index which is considered self-organizing.

Definition 7.5. *The patterns and the center vectors belong to the input space (or domain). The indices belong to the output space (or range).*

Dimensionality Reduction

The input dimension can be of any size, and it may in fact be high. The output dimension is generally taken to be 1- or 2-D. Hence this map is typically dimensionality reducing. It is of course interesting to consider maps where the input and output dimensions are the same. We note that if the indices are 2 dimensional we generally use the Euclidean metric to measure their separation.

A 1-D to 1-D Example. We now consider a simple illustrative example of the capabilities of the self-organizing feature map. Ten data points between 0 and 1 are selected *out of order.* Kohonen's algorithm then orders the data in the list. Note that there are 2 possible solutions to this problem; the one shown and the reverse ordering.

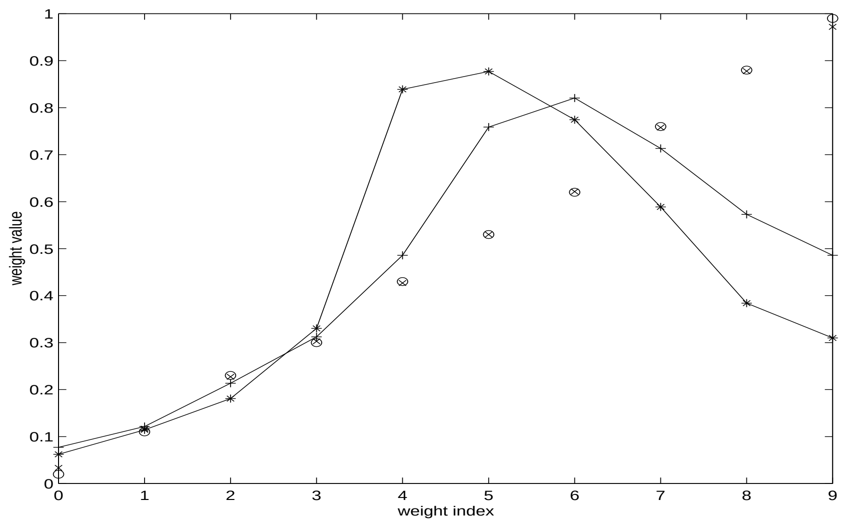

Fig. 7.10 '*' - Initial random center values. '+' - center values after 200 interations; 'x' - center values after 300 interations; 'o' - center values after 2000 interations. After 2000 iterations the centers have converged with machine precision to the initial data.

Training considerations

The function $h(\cdot)$ controls the local/global interaction of the indices for the updates. As such, the value of the width r for this function must be chosen and adapted with some care to avoid spurious solutions.

- *Pinching.* If the Gaussian width is too narrow, then the range of lateral interaction will be too short and self-organization will only occur locally, not globally.

- *Collapse.* If the Gaussian width is too wide, then all the center vectors are adapted in a similar fashion and take on similar values.

To avoid these pitfalls, the general strategy is adopted which reduces the width of the Gaussian as a function of time, so $r = r_n$. There are guideline for adapting both ϵ_n and r_n which ensure convergence (at least in the 1D case).

Kohonen suggests adapting the learning rate as

$$\epsilon_n = .9(1 - n/T)$$

where T is the total number of iterations. This should be maintained until the ordering phase is over. At this stage, he recommends using a value of about .01. The width of the Gaussian may also be reduced in a linear fashion.

patterns	center index	final center
.30	0	0.0200
.62	1	0.1100
.11	2	0.2300
.43	3	0.3000
.76	4	0.4300
.02	5	0.5300
.23	6	0.6200
.88	7	0.7600
.99	8	0.8800
.53	9	0.9900

7.7 ADDITIONAL TRAINING STRATEGIES

7.7.1 Orthogonal Least squares

7.7.2 Resource Allocating Networks

7.8 SUMMARY OF RBFS

Advantages

- locally tuned units (e.g., Gaussians)
- self-organizing receptive fields (unsupervised)
- fast supervised training of expansion coefficients
- stable learning (limited unlearning)

Disadvantages

- technique requires significant data
- number of RBFs gets large fast
- require a separate clustering routine

We have already mentioned the potential advantages of not performing a nonlinear optimization to determine the best set of centers and radii for the Gaussians in the RBF expansion. We now consider what is often referred to as a hybrid learning scheme which determines the RBF parameters in two distinct stages. This first stage computes the centers, and optionally the RBF shape parameterm, and is referred to as *unsupervised* as no target data is used to direct the update of the parameters. The second, *supervised* stage requires the computation of the centers, and proceeds along the lines of the discussion of the preceeding section.

Hybrid Training Schemes

Stage 1: **Unsupervised Stage**

- Determining the centers and radii of the RBFs using an off-line clustering algorithm such as:
  - competitive learning including k-means
  - topology preserving mappings
  - LBG clustering

Stage 2: **Supervised Stage**

- Determination of expansion coefficients (centers) by solving least squares problem.
  - Direct Methods
    - Normal equations with Cholesky
    - SVD
    - QR
  - Descent Methods
    - steepest descent (exact line search on normal equations)
    - conjugate gradient descent
    - least mean squares (LMS)
    - recursive lease squares (RLS)

Problems

7.1 Extend Equation (9.17) for the case that

$$\tilde{\mathbf{f}}(\mathbf{x}) = \mathbf{c}_0 + \sum_{m=1}^{N_c} \mathbf{c}_m \phi(\|\mathbf{x} - \mathbf{c}_m\|) \tag{7.32}$$

i.e., $\tilde{\mathbf{f}}(\mathbf{x}) \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$

7.2 Show that if Φ is an $m \times n$ matrix of full rank with $m \geq n$, then the matrix $A = \Phi^T \Phi$ is nonsingular.

7.3 Verify the identity in Equation (7.15).

7.4 Given

$$q(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T A \mathbf{w} - \mathbf{w}^T \mathbf{b}$$

show that

$$\nabla q(\mathbf{w}^n) = A\mathbf{w}^n - \mathbf{b}$$

Does A have to be symmetric for this to be true?

7.5 Show that the error

$$E(\mathbf{w}) = \|\Phi \mathbf{w} - \mathbf{f}\|_2^2$$

and

$$E(\mathbf{w}) = \langle (f - \tilde{f})^2 \rangle$$

are equivalent (up to a constant).

7.6 Show that Equation (7.14) is equivalent to the normal equations given in Equation (7.9).

7.7 Show that for the optimal weight vector, i.e., the one which satisfies the minimum mean-square error criterion, that the error signal $e^{(\mu)}$ is uncorrelated with the input signals, i.e., show

$$\langle e\mathbf{x} \rangle = 0$$

when $\mathbf{w} = \mathbf{w}^*$.

7.8 Plot representative members of the family of curves given by the equation

$$q(\mathbf{w}) = w_1^2 + w_1 w_2 + 2w_2^2 + w_1 + 2w_2 = k$$

where $k > 0$. Letting $\mathbf{w} = (w_1, w_2)^T$ and $\mathbf{b} = (b_1, b_2)^T$ rewrite this equation as

$$q(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T A \mathbf{w} - \mathbf{w}^T \mathbf{b} + c$$

specifying explicity the values of $A, \mathbf{b}, c$. Further, determine a transformation to diagonalize A and show that in the new primed coordinate system that

$$q(\mathbf{w}') = (\mathbf{w}')^T \Lambda \mathbf{w}' - (\mathbf{w}')^T \mathbf{b}' + c'$$

giving the values of $\Lambda, \mathbf{b}', c'$. Finally, show that the linear terms may be removed by completing the square. Specifically, introduce the new variable w_i'' where $w_i' = w_i'' - w_i/2d_{ii}$ so that

$$q(\mathbf{w}'') = (\mathbf{w}'')^T \Lambda \mathbf{w}'' + c''.$$

Plot the new curve in the $\mathbf{w}''$ coordinates. Lastly, introduce a transformation which will make Λ the identity matrix in the quadratic form.

7.9 Derive Equation (7.17) by determining the appropriate transformation to eliminate the term $\mathbf{z}^T \mathbf{b}'$ from Equation (7.16).

7.10 Verify Equation (7.26).

Computer Projects

7.11 Define the paraboloid

$$q(\mathbf{w}) = w_1^2 + w_1 w_2 + 2w_2^2 + w_1 + 2w_2$$

Consider the method of steepest descent for determining the minimum for $q(\mathbf{w})$.

- Find the maximum constant step size ϵ which guarantees convergence. Compute 10 iterations of the steepest descent algorithm using this maximum step-size.

- Calculate 10 iterations using an exact line search and compare your results with those obtained for a constant line search. Include a plot of the value of the step size as a function of the iteration.

7.12 Consider the paraboloid

$$q(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T C \mathbf{w}$$

where

$$C = \begin{pmatrix} 1 & 0 \\ 0 & \alpha \end{pmatrix}$$

Implement the steepest descent and conjugate gradient algorithms for minimizing $q(\mathbf{w})$ and compare the results of 10 iterations for the values $\alpha = 0.5, 0.1, 0.01$. Plot the sequence of iterations $\mathbf{w}^0, \mathbf{w}^1, \ldots, \mathbf{w}^1 0$ in the (w_1, w_2) plane.

7.13 Consider the logistic map $x_{n+1} = r x_n (1 - x_n)$ with $r = 4.0$. Plot the graphs of

- x_{n+1} versus x_n, i.e., the lagged vectors (x_n, x_{n+1})
- x_n versus n

for $n = 1, 1000$. Using the first 500 points as training data, construct a Gaussian RBF to approximate the mapping

$$f : x_n \rightarrow x_{n+1}$$

Use evenly spaced centers on the domain $[0, 1]$ with varying the number of centers $N_c = 10, 25, 50$. Use either a direct or iterative method to find the RBF weights. To test the results provide

- one-step prediction (domain value known exactly)
- iteration prediction (domain value is the predicted value)

and compare with the actual values of the logistic map for points 500-600.