C H A P T E R 5

Kernel Methods and Radial-Basis Function Networks

ORGANIZATION OF THE CHAPTER

In this chapter, we study another approach to machine learning : a kernel method based on clustering. After the introductory material presented in Section 5.1, the rest of the chapter is organized as follows:

1. Section 5.2 deals with Cover's theorem on the separability of patterns. This theorem is illustrated by revisiting the XOR problem.
2. Section 5.3 discusses a solution of the interpolation problem that uses radial-basis functions, setting the stage for the construction of radial-basis function (RBF) networks in Section 5.4; this latter section also includes practical considerations pertaining to RBF networks.
3. The K-means algorithm, discussed in Section 5.5, provides a simple, yet highly popular, algorithm for clustering, which is well suited for training the hidden layer in an unsupervised manner. Section 5.6 follows up on the K-means clustering algorithm to describe a recursive implementation of least-squares estimation (discussed in Chapter 2) for training the output layer of the RBF network in a supervised manner. Section 5.7 addresses practical considerations pertaining to this two-stage procedure for designing RBF networks. This procedure is illustrated in the computer experiment presented in Section 5.8, where comparisons are made with the results of the same computer experiment performed in Chapter 4 using the back-propagation algorithm.
4. Section 5.9 examines interpretations of Gaussian hidden units, followed by Section 5.10 on the relationship between kernel regression in statistics and RBF networks.

The chapter concludes with a summary and discussion in Section 5.11.

5.1 INTRODUCTION

The supervised training of a neural network may be approached in several different ways. The back-propagation learning algorithm for multilayer perceptrons, described in Chapter 4, may be viewed as the application of a recursive technique known in statistics as *stochastic approximation.*

In this chapter, we take a completely different approach. Specifically, we solve the problem of classifying nonlinearly separable patterns by proceeding in a *hybrid* manner, involving two stages:

- The first stage transforms a given set of nonlinearly separable patterns into a new set for which, under certain conditions, the likelihood of the transformed patterns becoming linearly separable is high; the mathematical justification of this transformation is traced to an early paper by Cover (1965).
- The second stage completes the solution to the prescribed classification problem by using least-squares estimation that was discussed in Chapter 2.

Through a discussion of the interpolation problem, we first describe an implementation of this hybrid approach to pattern classification by using a *radial-basis function (RBF) network*,[1] the structure of which consists of only three layers:

- The input layer is made up of source nodes (sensory units) that connect the network to its environment.
- The second layer, consisting of *hidden units*, applies a nonlinear transformation from the input space to the hidden (feature) space. For most applications, the dimensionality of the only hidden layer of the network is high; this layer is trained in an unsupervised manner using stage 1 of the hybrid learning procedure.
- The output layer is *linear*, designed to supply the response of the network to the activation pattern applied to the input layer; this layer is trained in a supervised manner using stage 2 of the hybrid procedure.

The nonlinear transformation from the input space to the hidden space and the high dimensionality of the hidden space satisfy the only two conditions of Cover's theorem.

Much of the theory developed on RBF networks builds on the Gaussian function, an important member of the class of radial-basis functions. The Gaussian function may also be viewed as a *kernel*—hence the designation of the two-stage procedure based on the Gaussian function as a *kernel method.*

Speaking of kernels, in the latter part of the chapter, we also discuss the relationship between kernel regression in statistics and radial-basis function networks.

5.2 COVER'S THEOREM ON THE SEPARABILITY OF PATTERNS

When a radial-basis function (RBF) network is used to perform a *complex* pattern-classification task, the problem is basically solved by first transforming it into a high-dimensional space in a nonlinear manner and then separating the classes in the output layer. The underlying justification is found in *Cover's theorem* on the *separability of patterns*, which, in qualitative terms, may be stated as follows (Cover, 1965):

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.

From the work we did on single-layer structures in Chapter 1 through 3, we know that once we have linearly separable patterns, the classification problem is easy to solve. Accordingly, we may develop a great deal of insight into the operation of an RBF network as a pattern classifier by studying the critical issue of separability of patterns.

Consider a family of surfaces where each naturally divides an input space into two regions. Let $\mathscr{X}$ denote a set of N patterns (vectors) $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, each of which is assigned to one of two classes $\mathscr{X}_1$ and $\mathscr{X}_2$. This *dichotomy* (binary partition) of the points is said to be separable with respect to the family of surfaces if a surface exists in the family that separates the points in the class $\mathscr{X}_1$ from those in the class $\mathscr{X}_2$. For each pattern $\mathbf{x} \in \mathscr{X}$, define a vector made up of a set of real-valued functions $\{\varphi_i(\mathbf{x}) | i = 1, 2, \ldots, m_1\}$, as shown by

$$\boldsymbol{\phi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \ldots, \varphi_{m_1}(\mathbf{x})]^T \tag{5.1}$$

Suppose that the pattern $\mathbf{x}$ is a vector in an m_0-dimensional input space. The vector $\boldsymbol{\phi}(\mathbf{x})$ then maps the points in the m_0-dimensional input space into corresponding points in a new space of dimension m_1. We refer to $\varphi_i(\mathbf{x})$ as a *hidden function*, because it plays a role similar to that of a hidden unit in a feedforward neural network. Correspondingly, the space spanned by the set of hidden functions $\{\varphi_i(\mathbf{x})\}_{i=1}^{m_1}$ is referred to as the *feature space.*

A dichotomy $\{\mathscr{X}_1, \mathscr{X}_2\}$ of $\mathscr{X}$ is said to be φ *separable* if there exists an m_1-dimensional vector $\mathbf{w}$ such that we may write the following (Cover, 1965):

$$\begin{aligned} \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) &> 0, \qquad \mathbf{x} \in \mathscr{X}_1 \\ \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) &< 0, \qquad \mathbf{x} \in \mathscr{X}_2 \end{aligned} \tag{5.2}$$

The hyperplane defined by the equation

$$\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) = 0$$

describes the separating surface in the $\boldsymbol{\phi}$-space (i.e., feature space). The inverse image of this hyperplane, that is,

$$\mathbf{x}: \quad \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) = 0 \tag{5.3}$$

defines the *separating surface* (i.e., decision boundary)in the input space.

Consider a natural class of mappings obtained by using a linear combination of r-wise products of the pattern vector coordinates. The separating surfaces corresponding to such mappings are referred to as *rth-order rational varieties.* A rational variety of order r in a space of dimension m_0 is described by an rth-degree homogeneous equation in the coordinates of the input vector $\mathbf{x}$, as shown by

$$\sum_{0 \le i_1 \le i_2 \le \cdots \le i_r \le m_0} a_{i_1 i_2 \ldots i_r} x_{i_1} x_{i_2} \ldots x_{i_r} = 0 \tag{5.4}$$

where x_i is the ith component of input vector $\mathbf{x}$ and x_0 is set equal to unity in order to express the equation in a homogeneous form. An rth-order product of entries x_i of $\mathbf{x}$—that is, $x_{i_1} x_{i_2} \ldots x_{i_r}$—is called a *monomial.* For an input space of dimensionality m_0, there are

$$\frac{m_0!}{(m_0 - r)!\, r!}$$

monomials in Eq. (5.4). Examples of the type of separating surfaces described by Eq. (5.4) are *hyperplanes* (first-order rational varieties), *quadrices* (second-order rational varieties), and *hyperspheres* (quadrics with certain linear constraints on the coefficients).

These examples are illustrated in Fig. 5.1 for a configuration of five points in a two-dimensional input space. In general, linear separability implies spherical separability, which implies quadric separability; however, the converses are not necessarily true.

In a probabilistic experiment, the separability of a set of patterns becomes a random event that depends on the dichotomy chosen and the distribution of the patterns in the input space. Suppose that the activation patterns $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are chosen independently, according to a probability measure imposed on the input space. Suppose also that all the possible dichotomies of $\mathscr{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ are equiprobable. Let $P(N, m_1)$ denote the probability that a particular dichotomy picked at random is φ separable, where the class of separating surfaces chosen has m_1 degrees of freedom. Following Cover (1965), we may then state that

$$P(N, m_1) = \begin{cases} (2^{1-N}) \displaystyle\sum_{m=0}^{m_1-1} \binom{N-1}{m} & \text{for } N > m_1 - 1 \\ 1 & \text{for } N \leq m_1 - 1 \end{cases} \tag{5.5}$$

where the binomial coefficients composing $N - 1$ and m are themselves defined for all integers l and m by

$$\binom{l}{m} = \frac{l!}{(l-m)!m!}$$

For a graphical interpretation of Eq. (5.5), it is best to normalize the equation by setting $N = \lambda m_1$ and plotting the probability $P(\lambda m_1, m_1)$ versus λ for various values of m_1. This plot reveals two interesting characteristics (Nilsson, 1965):

- a pronounced *threshold effect* around $\lambda = 2$;
- the value $P(2m_1, m_1) = 1/2$ for each value of m_1.

Equation (5.5) embodies the essence of *Cover's separability theorem* for random patterns.[2] It is a statement of the fact that the cumulative binomial distribution corresponding to the probability that $(N - 1)$ flips of a fair coin will result in $(m_1 - 1)$ or fewer heads.

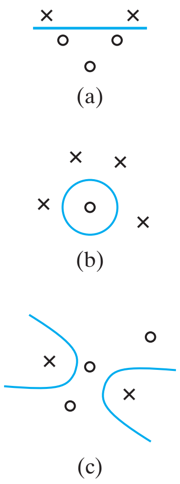


FIGURE 5.1 Three examples of φ-separable dichotomies of different sets of five points in two dimensions: (a) linearly separable dichotomy; (b) spherically separable dichotomy; (c) quadrically separable dichotomy.

Although the hidden-unit surfaces envisioned in the derivation of Eq. (5.5) are of a polynomial form and therefore different from those commonly used in radial-basis-function networks, the essential content of the equation has general applicability. Specifically, the higher we make the dimension m_1 of the hidden space, the closer the probability $P(N, m_1)$ will be to unity. To sum up, Cover's theorem on the separability of patterns encompasses two basic ingredients:

1. *nonlinear formulation of the hidden function defined by* $\varphi_i(\mathbf{x})$, *where* $\mathbf{x}$ is *the input vector and* $i = 1, 2, \ldots, m_1$;
2. *high dimensionality of the hidden (feature) space compared with the input space, where the dimensionality of the hidden space is determined by the value assigned to* m_1 *(i.e., the number of hidden units).*

In general, as stated previously, a complex pattern-classification problem cast in high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space. We emphasize, however, that in some cases the use of nonlinear mapping (i.e., point 1) may be sufficient to produce linear separability without having to increase the dimensionality of the hidden-unit space, as illustrated in the following example.

EXAMPLE 1 The XOR Problem

To illustrate the significance of the idea of φ separability of patterns, consider the simple, yet important, XOR problem. In the XOR problem, there are four points (patterns)—$(1, 1)$, $(0, 1)$, $(0, 0)$, and $(1, 0)$—in a two-dimensional input space, as depicted in Fig. 5.2a. The requirement is to construct a pattern classifier that produces the binary output 0 in response to the input pattern $(1, 1)$, or $(0, 0)$, and the binary output 1 in response to the input pattern $(0, 1)$ or $(1, 0)$. Thus, points that are closest in the input space, in terms of the Hamming distance, map to regions that are maximally apart in the output space. The *Hamming distance* of a sequence as defined is the number of changes from symbol 1 to symbol 0, and vice versa, that are found in a binary sequence. Thus, the Hamming distance of both 11 and 00 is zero, whereas the Hamming distance of both 01 and 10 is one.

Define a pair of Gaussian hidden functions as follows:

$$\varphi_1(\mathbf{x}) = \exp(-\|\mathbf{x} - t_1\|^2), \qquad \mathbf{t}_1 = [1, 1]^T$$
$$\varphi_2(\mathbf{x}) = \exp(-\|\mathbf{x} - t_2\|^2), \qquad \mathbf{t}_2 = [0, 0]^T$$

We may then construct the results summarized in Table 5.1 for the four different input patterns of interest. The input patterns are mapped onto the (φ_1, φ_2) plane as shown in Fig. 5.2b. Here, we now see that the input patterns $(0, 1)$ and $(1, 0)$ are linearly separable from the remaining input patterns $(1, 1)$ and $(0, 0)$. Thereafter, the XOR problem may be readily solved by using the functions $\varphi_1(\mathbf{x})$ and $\varphi_2(\mathbf{x})$ as the inputs to a linear classifier such as the perceptron. ■

In this example, there is no increase in the dimensionality of the hidden space compared with the input space. In other words, nonlinearity exemplified by the use of Gaussian hidden functions is sufficient to transform the XOR problem into a linearly separable one.

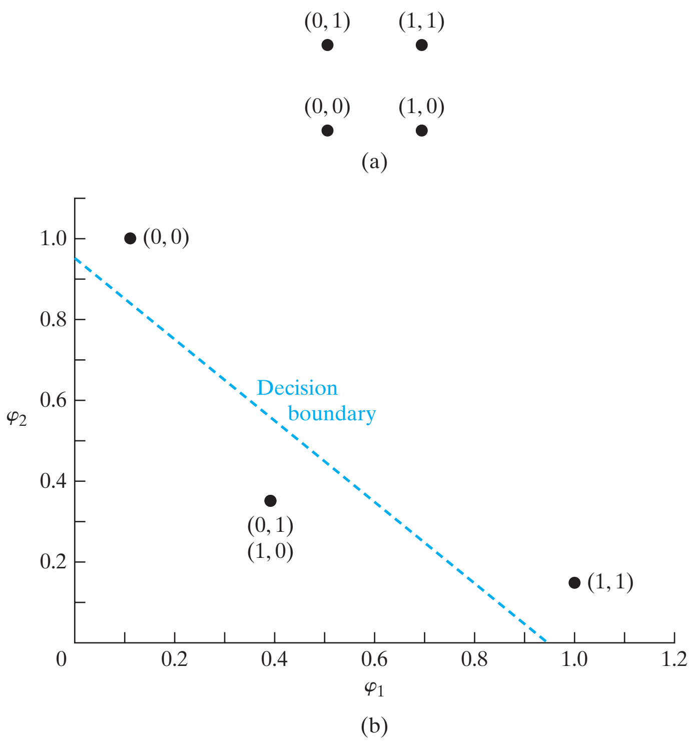


FIGURE 5.2 (a) The four patterns of the XOR problem; (b) decision-making diagram.

TABLE 5.1 Specification of the Hidden Functions for the XOR Problem of Example 1

Input Pattern $\mathbf{x}$	First Hidden Function $\varphi_1(\mathbf{x})$	Second Hidden Function $\varphi_2(\mathbf{x})$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(0,0)	0.1353	1
(1,0)	0.3678	0.3678

Separating Capacity of a Surface

Equation (5.5) has an important bearing on the expected maximum number of randomly assigned patterns that are linearly separable in a multidimensional space. To explore this issue, let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be a sequence of random patterns (vectors) as previously described. Let N be a random variable defined as the largest integer such that this sequence is φ separable, where φ has m_1 *degrees of freedom*. Then, from Eq. (5.5), we deduce that the probability that $N = n$ is given by

$$\begin{aligned}\text{Prob}(N = n) &= P(n, m_1) - P(n+1, m_1) \\ &= \left(\frac{1}{2}\right)^n \binom{n-1}{m_1 - 1}, \quad n = 0, 1, 2, \ldots\end{aligned} \tag{5.6}$$

For an interpretation of this result, recall the definition of a *negative binomial distribution.* This distribution equals the probability that k failures precede the rth success in a long, repeated sequence of *Bernoulli trials.* In such a probabilistic experiment, there are only two possible outcomes for each trial—success or failure—and their probabilities remain the same throughout the experiment. Let p and q denote the probabilities of success and failure, respectively, with $p + q = 1$. The negative binomial distribution is defined by the following (Feller, 1968):

$$f(k; r, p) = p^r q^k \binom{r+k-1}{k}$$

For the special case of $p = q = \frac{1}{2}$ (i.e., success and failure are equiprobable) and $k + r = n$, the negative binomial distribution reduces to

$$f\left(k; n-k, \frac{1}{2}\right) = \left(\frac{1}{2}\right)^n \binom{n-1}{k}, \quad n = 0, 1, 2, \ldots$$

With this definition, we now see that the result described in Eq. (5.6) is just the negative binomial distribution, shifted by m_1 units to the right, and with parameters m_1 and $\frac{1}{2}$. Thus, N corresponds to the "waiting time" for the m_1-th failure in a sequence of tosses of a fair coin. The expectation of the random variable N and its median are, respectively,

$$\mathbb{E}[N] = 2m_1 \tag{5.7}$$

and

$$\text{median}[N] = 2m_1 \tag{5.8}$$

We therefore have a corollary to Cover's theorem in the form of a celebrated asymptotic result that may be stated as follows (Cover, 1965):

> *The expected maximum number of randomly assigned patterns (vectors) that are linearly separable in a space of dimensionality m_1 is equal to $2m_1$.*

This result suggests that $2m_1$ is a natural definition of the *separating capacity* of a family of decision surfaces having m_1 degrees of freedom. In a loose-sense, it can be argued that Cover's separating capacity is related to the VC-dimension, discussed previously in Chapter 4.

5.3 THE INTERPOLATION PROBLEM

The important point that emerges from Cover's theorem on the separability of patterns is that in solving a nonlinearly separable pattern-classification problem, there is usually practical benefit to be gained by mapping the input space into a new space of high enough dimension. Basically, a nonlinear mapping is used to transform a nonlinearly separable classification problem into a linearly separable one with high probability. In a similar way, we may use a nonlinear mapping to transform a difficult nonlinear filtering problem into an easier one that involves linear filtering.

Consider then a feedforward network with an input layer, a single hidden layer, and an output layer consisting of a single unit. We have purposely chosen a single output unit to simplify the exposition without loss of generality. The network is designed to perform a *nonlinear mapping* from the input space to the hidden space, followed by a *linear mapping* from the hidden space to the output space. Let m_o denote the dimension of the input space. Then, in an overall fashion, the network represents a map from the m_o-dimensional input space to the single-dimensional output space, written as

$$s: \mathbb{R}^{m_0} \rightarrow \mathbb{R}^1 \tag{5.9}$$

We may think of the map s as a *hypersurface* (graph) $\Gamma \subset \mathbb{R}^{m_0+1}$, just as we think of the elementary map $s: \mathbb{R}^1 \rightarrow \mathbb{R}^1$, where $s(x) = x^2$, as a parabola drawn in $\mathbb{R}^2$ space. The surface Γ is a multidimensional plot of the output as a function of the input. In a practical situation, the surface Γ is unknown and the training data are usually contaminated with noise. The training phase and generalization phase of the learning process may be respectively viewed as follows (Broomhead and Lowe, 1988):

- The training phase constitutes the optimization of a fitting procedure for the surface Γ, based on known data points presented to the network in the form of input–output examples (patterns).
- The generalization phase is synonymous with interpolation between the data points, with the interpolation being performed along the constrained surface generated by the fitting procedure as the optimum approximation to the true surface Γ.

Thus, we are led to the theory of *multivariable interpolation* in high-dimensional space, which has a long history (Davis, 1963). The interpolation problem, in its *strict* sense, may be stated as follows:

Given a set of N different points $\{\mathbf{x}_i \in \mathbb{R}^{m_0} | i = 1, 2, ..., N\}$ *and a corresponding set of N real numbers* $\{d_i \in \mathbb{R}^1 | i = 1, 2, ..., N\}$, *find a function* $F: \mathbb{R}^N \rightarrow \mathbb{R}^1$ *that satisfies the interpolation condition:*

$$F(\mathbf{x}_i) = d_i, \qquad i = 1, 2, ..., N \tag{5.10}$$

For strict interpolation as specified here, the interpolating surface (i.e., function F) is constrained to pass through *all* the training data points.

The *radial-basis-functions* (RBF) technique consists of choosing a function F that has the form

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \varphi(\|\mathbf{x} - \mathbf{x}_i\|) \tag{5.11}$$

where $\{\varphi(\|\mathbf{x} - \mathbf{x}_i\|) | i = 1, 2, ..., N\}$ is a set of N arbitrary (generally nonlinear) functions, known as *radial-basis functions*, and $\|\cdot\|$ denotes a *norm* that is usually Euclidean (Powell, 1988). The known data points $\mathbf{x_i} \in \mathbb{R}^{m_o}$, $i = 1, 2, ..., N$ are taken to be the *centers* of the radial-basis functions.

Inserting the interpolation conditions of Eq. (5.10) into Eq. (5.11), we obtain a set of simultaneous linear equations for the unknown coefficients (weights) of the expansion $\{w_i\}$ given by

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix} \tag{5.12}$$

where

$$\varphi_{ij} = \varphi(\|\mathbf{x}_j - \mathbf{x}_j\|), \quad i, j = 1, 2, ..., N$$

Let

$$\mathbf{d} = [d_1, d_2, ..., d_N]^T$$
$$\mathbf{w} = [w_1, w_2, ..., w_N]^T$$

The N-by-1 vectors **d** and **w** represent the *desired response vector* and *linear weight vector*, respectively, where N is the *size of the training sample.* Let $\boldsymbol{\Phi}$ denote an N-by-N matrix with elements φ_{ij}:

$$\boldsymbol{\Phi} = \{\varphi_{ij}\}_{i,j=1}^{N} \tag{5.14}$$

We call this matrix the *interpolation matrix.* We may then rewrite Eq. (5.12) in the compact form

$$\boldsymbol{\Phi}\mathbf{w} = \mathbf{x} \tag{5.15}$$

Assuming that $\boldsymbol{\Phi}$ is nonsingular, and therefore that the inverse matrix $\boldsymbol{\Phi}^{-1}$ exists, we may go on to solve Eq. (5.15) for the weight vector **w**, obtaining

$$\mathbf{w} = \boldsymbol{\Phi}^{-1}\mathbf{x} \tag{5.16}$$

The vital question is: How can we be sure that the interpolation matrix $\boldsymbol{\Phi}$ is nonsingular?

It turns out that for a large class of radial-basis functions and under certain conditions, the answer to this basic question is given in an important theorem discussed next.

Micchelli's Theorem

In Micchelli (1986), the following theorem is proved:

Let $\{\mathbf{x}_i\}_{i=1}^{N}$ be a set of distinct points in $\mathbb{R}^{m_0}$. Then the N-by-N interpolation matrix $\boldsymbol{\Phi}$, whose ij-th element is $\varphi_{ij} = \varphi(\|\mathbf{x}_i - \mathbf{x}_j\|)$, is nonsingular.

There is a large class of radial-basis functions that is covered by Micchelli's theorem; it includes the following functions that are of particular interest in the study of RBF networks:

1. *Multiquadrics:*

$$\varphi(r) = (r^2 + c^2)^{1/2} \quad \text{for some } c > 0 \text{ and } r \in \mathbb{R} \tag{5.17}$$

2. *Inverse multiquadrics:*

$$\varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \quad \text{for some } c > 0 \text{ and } r \in \mathbb{R} \tag{5.18}$$

3. *Gaussian functions:*

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \text{for some } \sigma > 0 \text{ and } r \in \mathbb{R} \tag{5.19}$$

The multiquadrics and inverse multiquadrics are both due to Hardy (1971).

For the radial-basis functions listed in Eqs. (5.17) to (5.19) to be nonsingular, the points $\{\mathbf{x}_i\}_{i=1}^N$ must all be different (i.e., distinct). This is all that is required for nonsingularity of the interpolation matrix $\mathbf{\Phi}$, whatever the values of size N of the data points or dimensionality m_o of the vectors (points) $\mathbf{x}_i$ happen to be.

The inverse multiquadrics of Eq. (5.18) and the Gaussian functions of Eq. (5.19) share a common property: They are both *localized* functions, in the sense that $\varphi(r) \rightarrow 0$ as $r \rightarrow \infty$. In both of these cases, the interpolation matrix $\mathbf{\Phi}$ is positive definite. By contrast, the multiquadrics of Eq. (5.17) are *nonlocal*, in that $\varphi(r)$ becomes unbounded as $r \rightarrow \infty$, and the corresponding interpolation matrix $\mathbf{\Phi}$ has $N-1$ *negative* eigenvalues and only one positive eigenvalue, with the result that it is *not* positive definite (Micchelli, 1986). What is remarkable, however, is that an interpolation matrix $\mathbf{\Phi}$ based on Hardy's multiquadrics is nonsingular and therefore suitable for use in the design of RBF networks.

What is even more remarkable is that radial-basis functions that *grow* at infinity, such as multiquadrics, can be used to approximate a smooth input–output mapping with greater accuracy than those that yield a positive-definite interpolation matrix. This surprising result is discussed in Powell (1988).

5.4 RADIAL-BASIS-FUNCTION NETWORKS

In light of Eqs. (5.10) through (5.16), we may now envision a *radial-basis-function* (RBF) network in the form of a layered structure, as illustrated in Fig. 5.3; specifically, we have three layers:

1. *Input layer*, which consists of m_o source nodes, where m_o is the dimensionality of the input vector $\mathbf{x}$.
2. *Hidden layer*, which consists of the same number of computation units as the size of the training sample, namely, N; each unit is mathematically described by a radial-basis function

$$\varphi_j(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{x}_j\|), \qquad j = 1, 2, ..., N$$

The jth input data point $\mathbf{x}_j$ defines the center of the radial-basis function, and the vector $\mathbf{x}$ is the signal (pattern) applied to the input layer. Thus, unlike a multilayer perceptron, the links connecting the source nodes to the hidden units are direct connections with *no* weights.

3. *Output layer*, which, in the RBF structure of Fig. 5.3, consists of a single computational unit. Clearly, there is no restriction on the size of the output layer, except to say that typically the size of the output layer is much smaller than that of the hidden layer.

Henceforth, we focus on the use of a Gaussian function as the radial-basis function, in which case each computational unit in the hidden layer of the network of Fig. 5.3 is defined by

$$\begin{aligned}\varphi_j(\mathbf{x}) &= \varphi(\mathbf{x} - \mathbf{x}_j) \\ &= \exp\left(-\frac{1}{2\sigma_j^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right), \qquad j = 1, 2, ..., N \end{aligned} \tag{5.20}$$

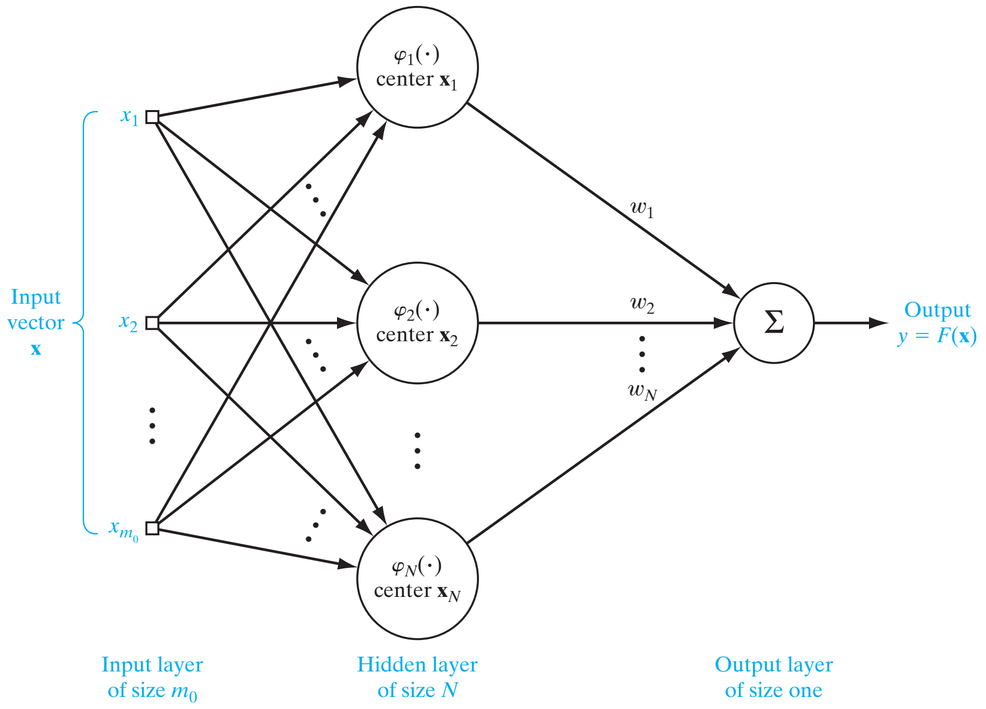


FIGURE 5.3 Structure of an RBF network, based on interpolation theory.

where σ_j is a measure of the *width* of the jth Gaussian function with center $\mathbf{x}_j$. Typically, but not always, all the *Gaussian hidden units* are assigned a common width σ. In situations of this kind, the parameter that distinguishes one hidden unit from another is the center $\mathbf{x}_j$. The rationale behind the choice of the Gaussian function as the radial-basis function in building RBF networks is that it has many desirable properties, which will become evident as the discussion progresses.

Practical Modifications to the RBF Network

Formulation of the RBF network of Fig. 5.3 via interpolation theory is rather neat. In practice, however, we find that the training sample $\{\mathbf{x}_i, d_i\}_{i=1}^{N}$ is typically *noisy*, be that in the context of pattern classification or nonlinear regression. Unfortunately, the use of interpolation based on noisy data could lead to misleading results—hence the need for a different approach to the design of an RBF network.

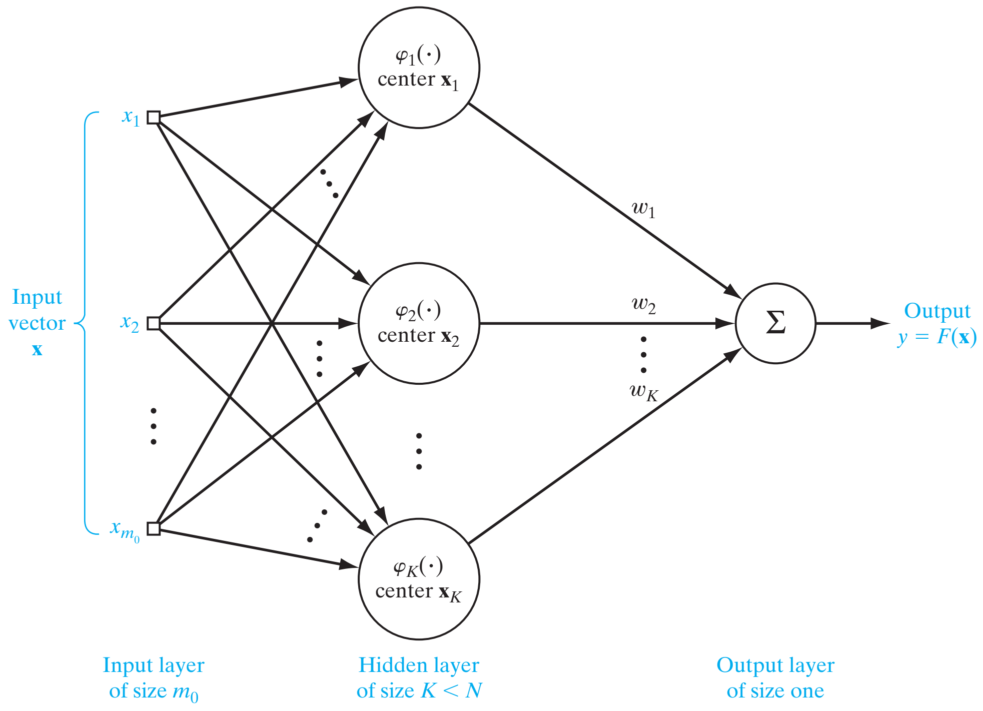
There is another practical issue that needs attention: Having a hidden layer of the same size as the input layer could be wasteful of computational resources, particularly when dealing with large training samples. When the hidden layer of the RBF network is specified in the manner described in Eq. (5.20), we find that any correlation existing between adjacent data points in the training sample is correspondingly transplanted into adjacent units in the hidden layer. Stated in another way, there is *redundancy* of neurons in the hidden layer when they are chosen in accordance with Eq. (5.20) by virtue of the redundancy that may inherently exist in the training sample. In situations of this kind, it is therefore good design practice to make the size of the hidden layer a fraction of the size of the training sample, as illustrated in Fig. 5.4. Note that although the RBF networks of Figs. 5.3 and 5.4 are indeed different, they do share a common feature: Unlike the case for a multilayer perceptron, the training of an RBF network does *not* involve the back propagation of error signals.

Moreover, the approximating function realized by both of these two RBF structures has the same mathematical form,

$$F(\mathbf{x}) = \sum_{j=1}^{K} w_j \varphi(\mathbf{x}, \mathbf{x}_j) \tag{5.21}$$

where the dimensionality of the input vector $\mathbf{x}$ (and therefore that of the input layer) is m_0 and each hidden unit is characterized by the radial-basis function $\varphi(\mathbf{x}, \mathbf{x}_j)$, where $j = 1, 2, \ldots, K$, with K being smaller than N. The output layer, assumed to consist of a single unit, is characterized by the weight vector $\mathbf{w}$, whose dimensionality is also K. The structure of Fig. 5.3 differs from that of Fig. 5.4 in two respects:

1. In Fig. 5.3, the dimensionality of the hidden layer is N, where N is the size of the training set, whereas that of Fig. 5.4 is $K < N$.



$\varphi_1(\cdot)$ center $\mathbf{x}_1$
x_1
Input vector $\mathbf{x}$
x_2
x_{m_0}
$\varphi_2(\cdot)$ center $\mathbf{x}_2$
$\varphi_K(\cdot)$ center $\mathbf{x}_K$
w_1
w_2
w_K
Σ
Output $y = F(\mathbf{x})$
Input layer of size m_0
Hidden layer of size $K < N$
Output layer of size one

FIGURE 5.4 Structure of a practical RBF network. Note that this network is similar in structure to that of Fig. 5.3. The two networks are different, however, in that the size of the hidden layer in Fig. 5.4 is smaller than that in Fig. 5.3.

2. Assuming that the training sample $\{\mathbf{x}_i, d_i\}_{i=1}^N$ is noiseless, the design of the hidden layer in Fig. 5.3 is solved simply by using the input vector $\mathbf{x}_j$ to define the center of the radial-basis function $\varphi(\mathbf{x}, \mathbf{x}_j)$ for $j = 1, 2, ..., N$. On the other hand, to design the hidden layer in Fig. 5.4, we have to come up with a new procedure.

The material covered in the next section addresses this latter point in a practical way for the case when the hidden units use Gaussian functions.

5.5 *K*-MEANS CLUSTERING

In designing the RBF network of Fig. 5.4, a key issue that needs to be addressed is how to compute the parameters of the Gaussian units that constitute the hidden layer by using *unlabeled* data. In other words, the computation is to be performed in an unsupervised manner. In this section, we describe a solution to this problem that is rooted in *clustering*, by which we mean the following:

> *Clustering is a form of unsupervised learning whereby a set of observations (i.e., data points) is partitioned into natural groupings or clusters of patterns in such a way that the measure of similarity between any pair of observations assigned to each cluster minimizes a specified cost function.*

There is a plethora of clustering techniques to choose from. We have chosen to focus on the so-called *K-means algorithm*,[3] because it is simple to implement, yet effective in performance, two features that have made it highly popular.

Let $\{\mathbf{x}_i\}_{i=1}^N$ denote a set of multidimensional observations that is to be partitioned into a proposed set of K clusters, where K is smaller than the number of observations, N. Let the relationship.

$$j = C(i), \qquad i = 1, 2, ..., N \tag{5.22}$$

denote a many-to-one mapper, called the *encoder*, which assigns the ith observation $\mathbf{x}_i$ to the jth cluster according to a rule yet to be defined. (The alert reader could wonder why it is we have chosen the index j to refer to a cluster when the logical choice would have been k; the reason for this choice is that the symbol k is used to refer to a kernel function that will be discussed later in the chapter.) To do this encoding, we need a *measure of similarity* between every pair of vectors $\mathbf{x}_i$ and $\mathbf{x}_{i'}$, which is denoted by $d(\mathbf{x}_i, \mathbf{x}_{i'})$. When the measure $d(\mathbf{x}_i, \mathbf{x}_{i'})$ is small enough, both $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ are assigned to the same cluster; otherwise, they are assigned to different clusters.

To optimize the clustering process, we introduce the following cost function (Hastie et al., 2001):

$$J(C) = \frac{1}{2}\sum_{j=1}^{K}\sum_{C(i)=j}\sum_{C(i')=j} d(\mathbf{x}_i, \mathbf{x}_{i'}) \tag{5.23}$$

For a prescribed K, the requirement is to find the encoder $C(i) = j$ for which the cost function $J(C)$ is minimized. At this point in the discussion, we note that the encoder C is unknown—hence the functional dependence of the cost function J on C.

In *K*-means clustering, the squared Euclidean norm is used to define the measure of similarity between the observations $\mathbf{x}_i$ and $\mathbf{x}_{i'}$, as shown by

$$d(\mathbf{x}_i, \mathbf{x}_{i'}) = \|\mathbf{x}_i - \mathbf{x}_{i'}\|^2 \tag{5.24}$$

Hence, substituting Eq. (5.24) into Eq. (5.23), we get

$$J(C) = \frac{1}{2}\sum_{j=1}^{K}\sum_{C(i)=j}\sum_{C(i')=j}\|\mathbf{x}_i - \mathbf{x}_{i'}\|^2 \tag{5.25}$$

We now make two points:

1. The squared Euclidean distance between the observations $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ is *symmetric*; that is,

$$\|\mathbf{x}_i - \mathbf{x}_{i'}\|^2 = \|\mathbf{x}_{i'} - \mathbf{x}_i\|^2$$

2. The inner summation in Eq. (5.25) reads as follows: For a given $\mathbf{x}_{i'}$, the encoder C assigns to cluster j all the observations $\mathbf{x}_{i'}$ that are closest to $\mathbf{x}_i$. Except for a scaling factor, the sum of the observations $\mathbf{x}_{i'}$ so assigned is an *estimate of the mean vector* pertaining to cluster j; the scaling factor in question is $1/N_j$, where N_j is the number of data points within cluster j.

On account of these two points, we may therefore reduce Eq. (5.25) to the simplified form

$$J(C) = \sum_{j=1}^{K}\sum_{C(i)=j}\|\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j\|^2 \tag{5.26}$$

where $\hat{\boldsymbol{\mu}}_j$ denotes the "estimated" mean vector associated with cluster j.[4] In effect, the mean $\hat{\boldsymbol{\mu}}_j$ may be viewed as the *center* of cluster j. In light of Eq. (5.26), we may now restate the clustering problem as follows:

> *Given a set of N observations, find the encoder C that assigns these observations to the K clusters in such a way that, within each cluster, the average measure of dissimilarity of the assigned observations from the cluster mean is minimized.*

Indeed, it is because of the essence of this statement that the clustering technique described herein is commonly known as the *K-means algorithm*.

For an interpretation of the cost function $J(C)$ defined in Eq. (5.26), we may say that, except for a scaling factor $1/N_j$, the inner summation in this equation is an estimate of the *variance* of the observations associated with cluster j for a given encoder C, as shown by

$$\hat{\sigma}_j^2 = \sum_{C(i)=j}\|\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j\|^2 \tag{5.27}$$

Accordingly, we may view the cost function $J(C)$ as a measure of the *total cluster variance* resulting from the assignments of all the N observations to the K clusters that are made by encoder C.

With encoder C being unknown, how do we minimize the cost function $J(C)$? To address this key question, we use an *iterative descent algorithm*, each iteration of which involves a two-step optimization. The first step uses the nearest neighbor rule to minimize the cost function $J(C)$ of Eq. (5.26) with respect to the mean vector $\hat{\boldsymbol{\mu}}_j$ for a given encoder C. The second step minimizes the inner summation of Eq. (5.26) with respect to the encoder C for a given mean vector $\hat{\boldsymbol{\mu}}_j$. This two-step iterative procedure is continued until convergence is attained.

Thus, in mathematical terms, the K-means algorithm proceeds in two steps:[5]

Step 1. For a given encoder C, the total cluster variance is minimized with respect to the assigned set of cluster means $\{\hat{\boldsymbol{\mu}}_j\}_{j=1}^{K}$; that is, we perform, the following minimization:

$$\min_{\{\hat{\boldsymbol{\mu}}\}_{j=1}^{K}} \sum_{j=i}^{K} \sum_{C(i)=j} \|\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j\|^2 \qquad \text{for a given } C \tag{5.28}$$

Step 2. Having computed the optimized cluster means $\{\hat{\boldsymbol{\mu}}_j\}_{j=1}^{K}$ in step 1, we next optimize the encoder as follows:

$$C(i) = \arg\min_{1 \le j \le K} \|\mathbf{x}(i) - \hat{\boldsymbol{\mu}}_j\|^2 \tag{5.29}$$

Starting from some initial choice of the encoder C, the algorithm goes back and forth between these two steps until there is no further change in the cluster assignments.

Each of these two steps is designed to reduce the cost function $J(C)$ in its own way; hence, convergence of the algorithm is assured. However, because the algorithm lacks a global optimality criterion, the result may converge to a local minimum, resulting in a *suboptimal* solution to the clustering assignment. Nevertheless, the algorithm has practical advantages:

1. The K-means algorithm is computationally *efficient*, in that its complexity is *linear* in the number of clusters.
2. When the clusters are compactly distributed in data space, they are faithfully *recovered* by the algorithm.

One last comment is in order: To initialize the K-means algorithm, the recommended procedure is to start the algorithm with many different random choices for the means $\{\hat{\boldsymbol{\mu}}_j\}_{j=i}^{K}$ for the proposed size K and then choose the particular set for which the double summation in Eq. (5.26) assumes the smallest value (Hastie et al., 2001).

The *K*-Means Algorithm Fits within the Framework of Cover's Theorem

The K-means algorithm applies a *nonlinear transformation* to the input signal $\mathbf{x}$. We say so because the measure of dissimilarity—namely, the squared Euclidean distance $\|\mathbf{x} - \mathbf{x}_j\|^2$, on which it is based—is a nonlinear function of the input signal $\mathbf{x}$ for a given cluster center $\mathbf{x}_j$. Furthermore, with each cluster discovered by the K-means algorithm defining a particular computational unit in the hidden layer, it follows that if the number of clusters, K, is large enough, the K-means algorithm will satisfy the other requirement of Cover's theorem—that is, that the dimensionality of the hidden layer is high enough. We therefore conclude that the K-means algorithm is indeed computationally powerful enough to transform a set of nonlinearly separable patterns into separable ones in accordance with this theorem.

Now that this objective has been satisfied, we are ready to consider designing the linear output layer of the RBF network.

5.6 RECURSIVE LEAST-SQUARES ESTIMATION OF THE WEIGHT VECTOR

The K-means algorithm performs computation in a recursive manner. It would therefore be desirable to recast the method of least squares—discussed in Chapter 2—for computing the weight vector in the output layer of the RBF network to perform this computation in a recursive manner, too. With this objective in mind, we recast Eq. (2.33) in the form

$$\mathbf{R}(n)\hat{\mathbf{w}}(n) = \mathbf{r}(n), \qquad n = 1, 2, ..., \tag{5.30}$$

where all three quantities are expressed as functions of discrete time n. In writing this equation, called the *normal equation* in statistics, we have introduced three terms:

1. the K-by-K *correlation function* of the hidden-unit outputs, which is defined by

$$\mathbf{R}(n) = \sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}^T(\mathbf{x}_i) \tag{5.31}$$

where

$$\boldsymbol{\phi}(\mathbf{x}_i) = [\varphi(\mathbf{x}_i, \boldsymbol{\mu}_1), \varphi(\mathbf{x}_i, \boldsymbol{\mu}_2), ..., \varphi(\mathbf{x}_i, \boldsymbol{\mu}_K)]^T \tag{5.32}$$

and

$$\varphi(\mathbf{x}_i, \boldsymbol{\mu}_j) = \exp\left(-\frac{1}{2\sigma_j^2}\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2\right), \qquad j = 1, 2, ..., K \tag{5.33}$$

2. the K-by-1 *cross-correlation vector* between the desired response at the output of the RBF network and the hidden-unit outputs, which is defined by

$$\mathbf{r}(n) = \sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i)d(i) \tag{5.34}$$

3. the *unknown weight vector* $\hat{\mathbf{w}}(n)$, which is optimized in the least-squares sense.

The requirement is to solve the normal equation in Eq. (5.30) for the weight vector $\mathbf{w}(n)$. Of course, we could do this computation by first inverting the correlation matrix $\mathbf{R}(n)$ and then multiplying the resulting inverse matrix $\mathbf{R}^{-1}(n)$ by the cross-correlation vector $\mathbf{r}(n)$, which is what is done in the method of least squares. However, when the size of the hidden layer, K, is large, which is often the case, computation of the inverse matrix $\mathbf{R}^{-1}(n)$ for $n = K$ can be a demanding task. The proposed use of a recursive implementation of the method of least squares takes care of this computational difficulty. The resulting algorithm is called the *recursive least-squares (RLS) algorithm*,[6] the derivation of which is discussed next.

The RLS algorithm

We begin the derivation of the RLS algorithm by reformulating Eq. (5.34) for the cross-correlation vector $\mathbf{r}(n)$, as shown by

$$\begin{aligned}\mathbf{r}(n) &= \sum_{i=1}^{n-1} \boldsymbol{\phi}(\mathbf{x}_i)d(i) + \boldsymbol{\phi}(\mathbf{x}_n)d(n) \\ &= \mathbf{r}(n-1) + \boldsymbol{\phi}(\mathbf{x}_n)d(n) \\ &= \mathbf{R}(n-1)\hat{\mathbf{w}}(n-1) + \phi(\mathbf{x}_n)d(n) \end{aligned} \tag{5.35}$$

where, in the first line, we isolated the term corresponding to $i = n$ from the summation in Eq. (5.34), and in the last line we used Eq. (5.30), replacing n with $n-1$. Next, we add the term $\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\hat{\mathbf{w}}(n-1)$ to the right-hand side of Eq. (5.35) in a purposeful way and subtract the same term from it in another part of the equation, leaving the equation itself unchanged; we thus write (after factoring common terms)

$$\mathbf{r}(n) = [\mathbf{R}(n-1) + \boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)]\hat{\mathbf{w}}(n-1) + \boldsymbol{\phi}(n)[d(n) - \boldsymbol{\phi}^T(n)\hat{\mathbf{w}}(n-1)] \tag{5.36}$$

The expression inside the first set of brackets on the right-hand side of Eq. (5.36) is recognized as the correlation function

$$\mathbf{R}(n) = \mathbf{R}(n-1) + \boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n) \tag{5.37}$$

For the expression inside the second set of brackets on the right-hand side of Eq. (5.36), we introduce the new term

$$\begin{aligned}\alpha(n) &= d(n) - \boldsymbol{\phi}^T(n)\mathbf{w}(n-1) \\ &= d(n) - \mathbf{w}^T(n-1)\boldsymbol{\phi}(n)\end{aligned} \tag{5.38}$$

This new term is called the *prior estimation error*, where the use of "prior" is intended to emphasize the fact that the estimation error $\alpha(n)$ is based on the old estimate $\hat{\mathbf{w}}(n-1)$ of the weight vector—that is, "before" the weight estimate was updated. The $\alpha(n)$ is also referred to as the *innovation*, because the input vector $\mathbf{x}(n)$ embedded in the $\boldsymbol{\phi}(n)$ and the corresponding desired response $d(n)$ represent "new" information available to the algorithm for estimation at time n.

Returning to Eq. (5.36), we may now make use of Eqs. (5.37) and (5.38) to simplify matters as follows:

$$\mathbf{r}(n) = \mathbf{R}(n)\hat{\mathbf{w}}(n-1) + \boldsymbol{\phi}(n)\alpha(n) \tag{5.39}$$

Accordingly, the use of this equation in Eq. (5.30) yields

$$\mathbf{R}(n)\hat{\mathbf{w}}(n) = \mathbf{R}(n)\hat{\mathbf{w}}(n-1) + \boldsymbol{\phi}(n)\alpha(n) \tag{5.40}$$

which may be expressed in the desired form for *updating* the weight vector, as shown by

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{R}^{-1}(n)\boldsymbol{\phi}(n)\alpha(n) \tag{5.41}$$

where we have multiplied both sides of Eq. (5.40) by the inverse matrix $\mathbf{R}^{-1}(n)$. To do the update in a computationally efficient manner, however, we need a corresponding formula for computing the inverse matrix $\mathbf{R}^{-1}(n)$, given its past value $\mathbf{R}^{-1}(n-1)$; this issue is discussed next.

Recursive Formula for Computing $\mathbf{R}^{-1}(n)$

Referring back to Eq. (5.37), we see that we do have a formula for recursively updating the correlation matrix $\mathbf{R}(n)$. We capitalize on this recursion for the purpose of a recursive formula for the inverse matrix $\mathbf{R}^{-1}(n)$ by using the matrix inverse lemma, which was discussed in Section 4.14.

To recap, consider the matrix

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{CD}C^T \tag{5.42}$$

where it is assumed that matrix $\mathbf{B}$ is nonsingular and matrix $\mathbf{B}^{-1}$ therefore exists. Matrices $\mathbf{A}$ and $\mathbf{B}$ are of similar dimensions, matrix $\mathbf{D}$ is another nonsingular matrix with different dimensions, and matrix $\mathbf{C}$ is a rectangular matrix of appropriate dimensions. According to the *matrix inversion lemma*, we have

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{BC}(\mathbf{D} + \mathbf{C}^T\mathbf{BC})^{-1}\mathbf{C}^T\mathbf{B} \tag{5.43}$$

For the problem at hand, we use Eq. (5.37) to make the following identifications:

$$\begin{aligned}\mathbf{A} &= \mathbf{R}(n)\\ \mathbf{B}^{-1} &= \mathbf{R}(n-1)\\ \mathbf{C} &= \boldsymbol{\phi}(n)\\ \mathbf{D} &= 1\end{aligned}$$

Accordingly, the application of Eq. (5.43) to this special set of matrices yields

$$\mathbf{R}^{-1}(n) = \mathbf{R}^{-1}(n-1) - \frac{\mathbf{R}^{-1}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{R}^{-1T}(n-1)}{1 + \boldsymbol{\phi}^T(n)\mathbf{R}^{-1}(n-1)\boldsymbol{\phi}(n)} \tag{5.44}$$

where, in the second term on the right-hand side of the equation, we have made use of the *symmetry* property of the correlation matrix; that is,

$$\mathbf{R}^T(n-1) = \mathbf{R}(n-1)$$

To simplify the formulation of the RLS algorithm, we now introduce two new definitions:

1. $\mathbf{R}^{-1}(n) = \mathbf{P}(n)$

Hence, we may reformulate Eq. (5.44) as

$$\mathbf{P}(n) = \mathbf{P}(n-1) - \frac{\mathbf{P}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{P}(n-1)}{\mathbf{P}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{P}^T(n-1)} \tag{5.45}$$

where the denominator on the right-hand side of the equation is a quadratic form and therefore a scalar.

To provide an interpretation for $\mathbf{P}(n)$, consider the linear regression model

$$d(n) = \mathbf{w}^T\boldsymbol{\phi}(n) + \varepsilon(n)$$

as the *generative model* for the desired response $d(n)$, with $\boldsymbol{\phi}(n)$ as the regressor. The additive noise term $\varepsilon(n)$ is assumed to be white with zero mean and variance σ_ε^2. Then, viewing the unknown weight vector $\mathbf{w}$ as the *state* of the model and $\hat{\mathbf{w}}(n)$ as the estimate produced by the RLS algorithm, we define the *state-error covariance matrix* as follows:

$$\mathbb{E}[(\mathbf{w} - \hat{\mathbf{w}}(n))(\mathbf{w} - \hat{\mathbf{w}}(n))^T] = \sigma_\varepsilon^2\mathbf{P}(n) \tag{5.46}$$

Verification of this result is addressed in Problem 5.5.

2.
$$\begin{aligned}\mathbf{g}(n) &= \mathbf{R}^{-1}(n)\boldsymbol{\phi}(n) \\ &= \mathbf{P}(n)\boldsymbol{\phi}(n)\end{aligned} \tag{5.47}$$

The new term $\mathbf{g}(n)$ is called the *gain vector* of the RLS algorithm, because, in light of Eq. (5.41), we may view the prior estimation error $\alpha(n)$ multiplied by $\mathbf{g}(n)$ as the correction needed for updating the old estimate $\hat{\mathbf{w}}(n-1)$ to its new value $\hat{\mathbf{w}}(n)$, as shown by

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{g}(n)\alpha(n) \tag{5.48}$$

Summary of the RLS Algorithm[7]

With Eqs. (5.45), (5.47), (5.38) and (5.48) at hand, in that order, we may now summarize the RLS algorithm as follows:

Given the training sample $\{\boldsymbol{\phi}(i), d(i)\}_{i=1}^N$, do the following computations for $n = 1, 2, \dots, N$:

$$\begin{aligned}\mathbf{P}(n) &= \mathbf{P}(n-1) - \frac{\mathbf{P}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{P}(n-1)}{1 + \boldsymbol{\phi}^T(n)\mathbf{P}(n-1)\boldsymbol{\phi}(n)} \\ \mathbf{g}(n) &= \mathbf{P}(n)\boldsymbol{\phi}(n) \\ \alpha(n) &= d(n) - \hat{\mathbf{w}}^T(n-1)\boldsymbol{\phi}(n) \\ \hat{\mathbf{w}}(n) &= \hat{\mathbf{w}}(n-1) + \mathbf{g}(n)\alpha(n)\end{aligned}$$

To initialize the algorithm, set

$$\hat{\mathbf{w}}(0) = \mathbf{0}$$

and

$$\mathbf{P}(0) = \lambda^{-1}\mathbf{I}, \qquad \lambda \text{ is a small positive constant}$$

Note that the λ used in the initialization of the algorithm performs the role of a regularizing parameter in the cost function

$$\mathscr{E}_{\text{av}}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(d(i) - \mathbf{w}^T\boldsymbol{\phi}(i))^2 + \frac{1}{2}\lambda\|\mathbf{w}\|^2$$

When λ is chosen relatively small, which is typically the case, then, indirectly, we are confirming confidence in the quality of the training sample $\{\mathbf{x}(i), d(i)\}_{i=1}^N$.

5.7 HYBRID LEARNING PROCEDURE FOR RBF NETWORKS

Equipped with the K-means clustering algorithm described in Section 5.5 and the recursive least-squares (RLS) algorithm derived in Section 5.6, we are now ready to describe a *hybrid learning procedure*[8] for the RBF network of Fig. 5.4. The K-means algorithm for training the hidden layer is applied first; it is then followed by the RLS algorithm for training the output layer. Hereafter, we refer to this hybrid learning procedure as the *"K-means, RLS" algorithm*, aimed at training an RBF network with the following composition:

Input layer. The size of the input layer is determined by the dimensionality of the input vector $\mathbf{x}$, which is denoted by m_0.

Hidden layer.

1. The size of the hidden layer, m_1, is determined by the proposed number of clusters, K. Indeed, the parameter K may by viewed as a *degree of freedom* under the designer's control. As such, the parameter K holds the key to the model-selection problem and thereby controls not only the performance, but also the computational complexity of the network.
2. The cluster mean $\hat{\boldsymbol{\mu}}_j$, computed by the K-means algorithm working on the unlabeled sample $\{\mathbf{x}_i\}_{i=1}^N$ of input vectors, determines the center $\mathbf{x}_j$ in the Gaussian function $\varphi(\cdot,\mathbf{x}_j)$ assigned to the hidden unit $j = 1, 2, ..., K$.
3. To simplify the design, the same width, denoted by σ, is assigned to all the Gaussian functions in accordance with the spread of the centers discovered by the K-means algorithm, as shown by

$$\sigma = \frac{d_{\max}}{\sqrt{2K}} \tag{5.49}$$

where K is the number of centers and $d_{\max}$ is the maximum distance between them (Lowe, 1989). This formula ensures that the individual Gaussian units are not too peaked or too flat; both extreme conditions should be avoided in practice.

Output layer. Once the training of the hidden layer is completed, the training of the output layer can begin. Let the K-by-1 vector

$$\boldsymbol{\phi}(\mathbf{x}_i) = \begin{bmatrix} \varphi(\mathbf{x}_i, \boldsymbol{\mu}_1) \\ \varphi(\mathbf{x}_i, \boldsymbol{\mu}_2) \\ \vdots \\ \varphi(\mathbf{x}_i, \boldsymbol{\mu}_K) \end{bmatrix}$$

denote the outputs of the K units in the hidden layer. This vector is produced in response to the stimulus $\mathbf{x}_i$, $i = 1, 2, ..., N$. Thus, insofar as the supervised training of the output layer is concerned, the training sample is defined by $\{\boldsymbol{\phi}(\mathbf{x}_i), d_i\}_{i=1}^N$, where d_i is the desired response at the overall output of the RBF network for input $\mathbf{x}_i$. This training is carried out using the RLS algorithm. Once the network training is completed, testing of the whole network with data not seen before can begin.

An attractive feature of the "K-means, RLS" algorithm is its computational efficiency, which follows from the fact that the K-means and RLS algorithms are both computationally efficient in their own individual ways. The only questionable feature of the algorithm is the absence of an overall optimality criterion that combines the training of the hidden and output layers, assuring the whole system of optimality in some statistical sense.

5.8 COMPUTER EXPERIMENT: PATTERN CLASSIFICATION

In this section, we use a computer experiment to evaluate the pattern-classification performance of the "K-means, RLS" algorithm used to train an RBF network. The experiment uses data obtained by randomly sampling the double-moon configuration of Fig. 1.8. A specific aim of the experiment is to compare the performance of the RBF network trained in this way with that of the multilayer perceptron (MLP) trained using the back-propagation algorithm, which was the focus of attention in the experiment performed in Section 4.7.

The hidden layer of the RBF network was chosen to consist of 20 Gaussian units, thereby making it of the same size as the hidden layer of the MLP that was investigated in Section 4.7. For training of the RBF network, 1,000 data points were used; and for testing, 2,000 data points were used. In a manner similar to that for the MLP experiments, the RBF experiments were performed for two different settings of the double-moon figure, $d = -5$ and $d = -6$, with the latter setting being the more difficult one of the two.

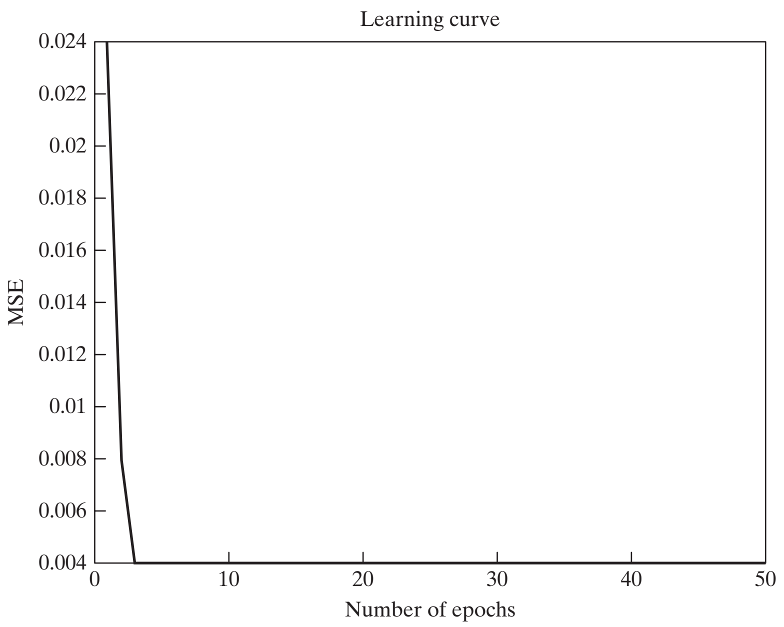
(a) *Vertical separation:* $d = -5$

For this vertical separation between the two moons, $K = 20$ was assigned as the number of clusters (i.e., number of hidden units). By applying the K-means algorithm to the unlabeled part of the training sample, the centers of the clusters, and therefore the centers of the Gaussian units in the hidden layer, were determined. With the spread of the centers known, the formula of Eq. (5.49) was then used to compute the common width $\sigma = 2.6$ assigned to all the Gaussian units. The design of the hidden layer of the RBF network was thus completed. Finally, the RLS algorithm was used to train the output layer and thereby compute the decision boundary, preparing the way for the testing session.

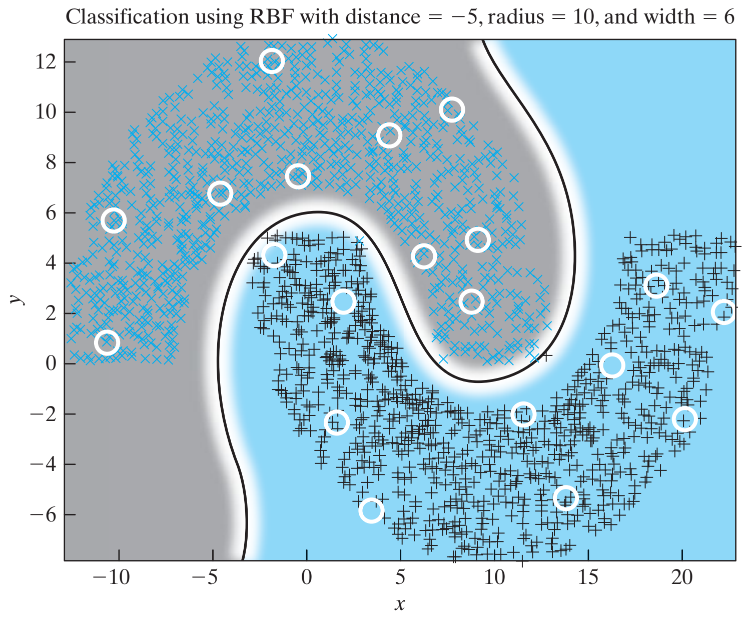
The results of the first part of the experiment are presented in Fig. 5.5. Part (a) of the figure shows the learning curve of the RLS algorithm and part (b) shows the decision boundary learned by the RBF network. As can be seen from Fig. 5.5a, within two epochs of training, the design of the output layer is completed. Part (b) of the figure confirms almost perfect separability of the two moon-shaped patterns by the RBF network.

(b) *Vertical separation:* $d = -6$

The pattern-classification experiment on the RBF network was then repeated for this more difficult setting of the double-moon configuration of Fig. 1.8. This time, the common width $\sigma = 2.4$ was assigned to the 20 Gaussian units, the assignment again being performed in accordance with the formula of Eq. (5.49).



(a) Learning curve



(b) Testing result

FIGURE 5.5 RBF network trained with *K*-means and RLS algorithms for distance $d = -5$. The MSE in part (a) of the figure stands for mean-square error.

The results of the second part of the experiment are presented in Fig. 5.6, with part (a) of the figure showing the learning curve of the RLS algorithm, and part (b) showing the decision boundary learned by the RBF network as a result of training under the "K-means, RLS" algorithm. A total of ten classification errors out of 2,000 test data points were recorded, yielding a classification error rate of 0.5%.

For both parts (a) and (b) of the experiment, the classification threshold was set at zero at the single output of the RBF network.

Comparison of the MLP and RBF Results

Comparing the results of experiments (a) and (b) performed on the RBF network in this section with those of the corresponding experiment performed on the MLP in Section 4.7, we draw the following conclusions:

1. The RBF network trained with the "K-means, RLS" algorithm outperforms the MLP trained with the back-propagation algorithm. Specifically, the MLP failed perfect classification for the setting $d = -5$ of the double moon, whereas the RBF network reported almost perfect classification. The misclassification rate of 0.5% produced by the RBF network for the difficult setting $d = -6$ was slightly greater than the 0.15% produced by the MLP for the easier setting $d = -5$. Surely, the MLP design could have been improved; however, the same thing could also be said for the RBF network.

2. The training process performed on the RBF network was significantly faster than that performed on the MLP.

5.9 INTERPRETATIONS OF THE GAUSSIAN HIDDEN UNITS

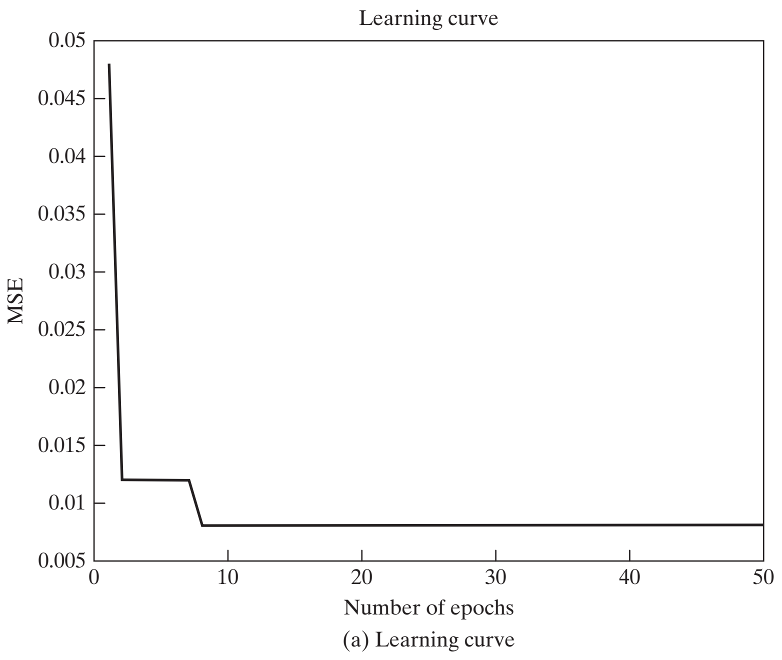
The Idea of Receptive Field

In a neurobiological context, a *receptive field* is defined as "that region of a sensory field from which an adequate sensory stimulus will elicit a response" (Churchland and Sejnowski, 1992). What is interesting to realize is that the receptive fields of cells in higher areas of the visual cortex tend to be much larger than those of cells in the earlier stages of the visual system.

Following this neurobiological definition of a receptive field, we may envision each hidden unit of a neural network to have a receptive field of its own. Indeed, we may go on to make the following corresponding statement:

> *The receptive field of a computational unit (e.g., hidden unit) in a neural network is, in general, that region of the sensory field (e.g., input layer of source nodes) from which an adequate sensory stimulus (e.g., pattern) will elicit a response.*

This definition applies equally well to multilayer perceptrons and RBF networks. However, the mathematical delineation of the receptive field in an RBF network is easier to determine than that in a multilayer perceptron.



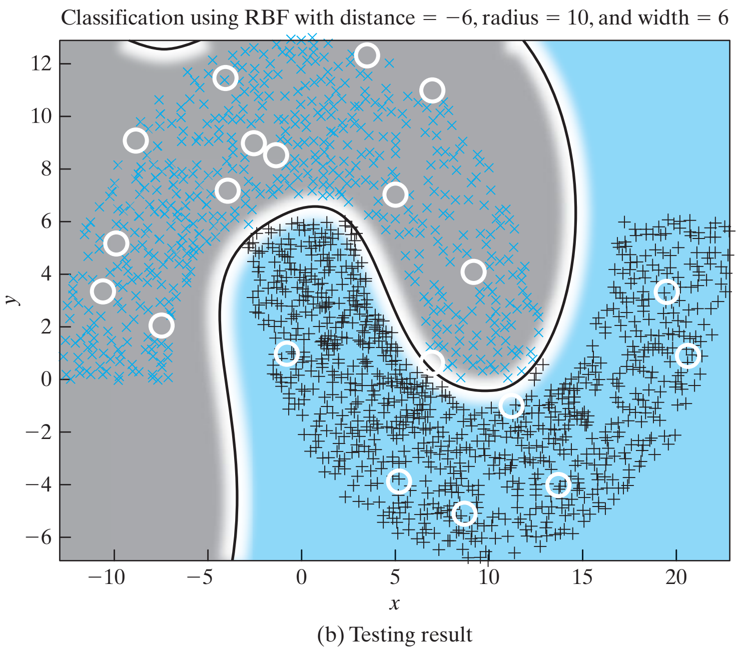


FIGURE 5.6 RBF network trained with *K*-means and RLS algorithms for distanced $d = -6$. The MSE in part (a) stands for mean-square error.

Let $\varphi(\mathbf{x}, \mathbf{x}_j)$ denote the functional dependence of a computational unit on the input vector $\mathbf{x}$, given that the unit is centered on the point $\mathbf{x}_j$. According to Xu et al. (1994), the receptive field of this computational unit is defined by

$$\psi(\mathbf{x}) = \varphi(\mathbf{x}, \mathbf{x}_j) - a \tag{5.50}$$

where a is some positive constant. In words, this equation states that the receptive field of the function $\varphi(\mathbf{x}, \mathbf{x}_j)$ is that particular subset of the domain of the input vector $\mathbf{x}$ for which the function $\varphi(\mathbf{x}, \mathbf{x}_j)$ takes sufficiently large values, all of them being equal to or greater than the prescribed level a.

EXAMPLE 2 Receptive field of a Gaussian hidden unit

Consider a Gaussian computational unit defined by

$$\varphi(\mathbf{x}, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right)$$

According to Eq. (5.50), the receptive field of this unit is

$$\psi(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right) - a$$

where $a < 1$. The minimum permissible value of $\psi(\mathbf{x})$ is zero, for which this equation yields

$$\|\mathbf{x} - \mathbf{x}_j\| = \sigma\sqrt{2\log(\tfrac{1}{a})}$$

It follows therefore that the receptive field of the Gaussian function $\varphi(\mathbf{x},\mathbf{x}_j)$ is defined by a *multidimensional surface*, which is symmetrically centered around the point $\mathbf{x}_j$ in a spheroidal manner. The spheroidal symmetric property of the receptive field is inherited naturally from the Gaussian function itself.

Figure 5.7 depicts two special cases of this surface:

1. *One-dimensional receptive field*, $\psi(x)$, for which the domain of the input x is confined to the closed interval $[(x_i - \sigma\sqrt{2\log(1/a)}), (x_i + \sigma\sqrt{2\log(1/a)})]$, as shown in part (a) of the figure.
2. *Two-dimensional receptive field*, $\psi(\mathbf{x})$, for which the domain of the input $\boldsymbol{x}$ is confined to a circular disc of center $\mathbf{x}_i = [x_{i,1}, x_{i,2}]^T$ and radius $\sigma\sqrt{2\log(1/a)}$, as shown in part (b) of the figure. ■

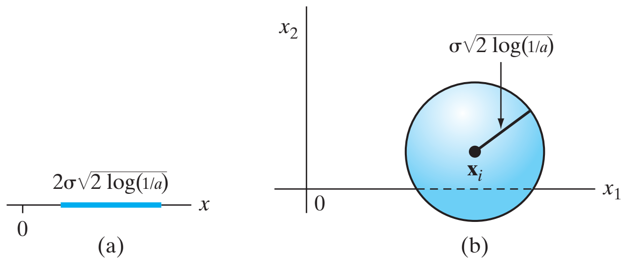


FIGURE 5.7 Illustrating the notion of receptive field for two special cases: (a) one-dimensional, and (b) two-dimensional.

Interpretation of the Gaussian Function as a Kernel

One other important facet of the Gaussian function $\varphi(\mathbf{x}, \mathbf{x}_j)$ is that it may be interpreted as a kernel, a term that is widely used in the statistics literature; it is also being increasingly used in the machine-learning literature.

Consider a function dependent on an input vector $\mathbf{x}$, with its center located at the origin of the Euclidean space. Basic to the formulation of this function as a *kernel*, denoted by $k(\mathbf{x})$, is that the function has properties similar to those associated with the probability density function of a random variable:

Property 1. *The kernel $k(\mathbf{x})$ is a continuous, bounded, and real function of $\mathbf{x}$ and symmetric about the origin, where it attains its maximum value.*

Property 2. *The total volume under the surface of the kernel $k(\mathbf{x})$ is unity; that is, for an m-dimensional* vector $\mathbf{x}$, *we have*

$$\int_{\mathbb{R}^m} k(\mathbf{x})d\mathbf{x} = 1$$

Except for a scaling factor, the Gaussian function $\varphi(\mathbf{x}, \mathbf{x}_j)$ satisfies both of these properties for the center $\mathbf{x}_j$ located at the origin. For a nonzero value of $\mathbf{x}_j$, properties 1 and 2 still hold except for the fact that $\mathbf{x}_j$ replaces the origin.

It is because of the interpretation of the Gaussian function as a kernel that we have used the term "kernel methods" in the title of this chapter.

5.10 KERNEL REGRESSION AND ITS RELATION TO RBF NETWORKS

The theory of RBF networks, presented in Section 5.3, is built on the notion of interpolation. In this section, we take another viewpoint—namely, *kernel regression*, which builds on the notion of density estimation.

To be specific, consider a nonlinear regression model defined by

$$y_i = f(\mathbf{x}_i) + \varepsilon(i), \qquad i = 1, 2, ..., N \tag{5.51}$$

where $\varepsilon(i)$ is an additive white-noise term of zero mean and variance σ_ε^2. To avoid confusion, we have used the symbol y_i (instead of d_i as previously) to denote the model output. As a reasonable estimate of the unknown regression function $f(\mathbf{x})$, we may take the mean of observables (i.e., values of the model output y) near a point $\mathbf{x}$. For this approach to be successful, however, the local average should be confined to observations in a small neighborhood (i.e., receptive field) around the point $\mathbf{x}$ because, in general, observations corresponding to points away from $\mathbf{x}$ will have different mean values. More precisely, we find that the unknown function $f(\mathbf{x})$ is equal to the *conditional mean of the observable y given the regressor* $\mathbf{x}$, as shown by

$$\begin{aligned} f(\mathbf{x}) &= \mathbb{E}[y|\mathbf{x}] \\ &= \int_{-\infty}^{\infty} y p_{Y|\mathbf{X}}(y|\mathbf{x})dy \end{aligned} \tag{5.52}$$

where $p_{Y|\mathbf{X}}(y|\mathbf{x})$ is the conditional probability density function (pdf) of *the random variable Y given that the random vector $\mathbf{X}$ is assigned the value $\mathbf{x}$*.[9] From probability theory, we have

$$p_{Y|\mathbf{X}}(y|\mathbf{x}) = \frac{p_{\mathbf{X}|Y}(\mathbf{x}|y)}{p_{\mathbf{X}}(\mathbf{x})} \tag{5.53}$$

where $p_{\mathbf{X}}(\mathbf{x})$ is the pdf of $\mathbf{X}$ and $p_{\mathrm{x,\,y}}(\mathbf{x}, y)$ is the joint pdf of $\mathbf{X}$ and Y. Hence, using Eq. (5.53) in Eq. (5.52), we obtain the following formula for the regression function:

$$f(\mathbf{x}) = \frac{\displaystyle\int_{-\infty}^{\infty} y p_{\mathbf{X},Y}(\mathbf{x}, y)dy}{p_{\mathbf{X}}(\mathbf{x})} \tag{5.54}$$

Our particular interest is in a situation where the joint probability density function $p_{\mathbf{X},Y}(\mathbf{x}, y)$ is unknown and all that we have available is the training sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. To estimate $p_{\mathbf{X},Y}(\mathbf{x},y)$, and therefore $p_{\mathbf{X}}(\mathbf{x})$, we may use a nonparametric estimator known as the *Parzen–Rosenblatt density estimator* (Rosenblatt, 1956, 1970; Parzen, 1962). Basic to the formulation of this estimator is the availability of a *kernel* $k(\mathbf{x})$. Assuming that the observations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are statistically independent and identically distributed (iid), we may formally define the Parzen–Rosenblatt density estimate of $f_{\mathbf{X}}(\mathbf{x})$ as

$$\hat{p}_{\mathbf{X}}(\mathbf{x}) = \frac{1}{Nh^{m_0}}\sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \qquad \text{for } \mathbf{x} \in \mathbb{R}^{m_0} \tag{5.55}$$

where the smoothing parameter h is a positive number called *bandwidth*, or simply *width*; h controls the size of the kernel. An important property of the Parzen–Rosenblatt density estimator is that it is a *consistent estimator*[10] (i.e., asymptotically unbiased) in the sense that if $h = h(N)$ is chosen as a function of N such that

$$\lim_{N\to\infty} h(N) = 0,$$

then

$$\lim_{N\to\infty} \mathbb{E}[\hat{p}_{\mathbf{X}}(\mathbf{x})] = p_{\mathbf{X}}(\mathbf{x})$$

For this latter equation to hold, $\mathbf{x}$ should be a point of continuity for $\hat{p}_{\mathbf{X}}(\mathbf{x})$.

In a manner similar to that described in Eq. (5.55), we may formulate the Parzen–Rosenblatt density estimate of the joint probability density function $p_{\mathbf{X},Y}(\mathbf{x}, y)$ as

$$\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y) = \frac{1}{Nh^{m_0+1}}\sum_{i=1}^{N} k\left(\frac{\mathrm{x} - \mathrm{x}_i}{h}\right)k\left(\frac{y - y_i}{h}\right) \qquad \text{for } \mathbf{x} \in \mathbb{R}^{m_0} \text{ and } y \in \mathbb{R} \tag{5.56}$$

Integrating $\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y)$ with respect to y, we get the $\hat{p}_{\mathbf{X}}(\mathbf{x})$ of Eq. (5.55), as we should. Moreover,

$$\int_{-\infty}^{\infty} y\hat{p}_{\mathbf{X},Y}(\mathbf{x}, \mathrm{y})dy = \frac{1}{Nh^{m_0+1}}\sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)\int_{-\infty}^{\infty} yk\left(\frac{y - y_i}{h}\right)dy$$

Changing the variable of integration by setting $\xi = (y - y_i)/h$, and using Property 2 of the kernel $k(\cdot)$, we obtain the result

$$\int_{-\infty}^{\infty} y\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y)\, dy = \frac{1}{Nh^{m_0}} \sum_{i=1}^{N} y_i k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \tag{5.57}$$

Thus, using Eqs. (5.57) and (5.55) as estimates of the quantities in the numerator and denominator of Eq. (5.54), respectively, we obtain the following estimate of the regression function $f(\mathbf{x})$, after canceling common terms:

$$\begin{aligned} F(\mathbf{x}) &= \hat{f}(\mathbf{x}) \\ &= \frac{\sum_{i=1}^{N} y_i k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)}{\sum_{j=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_j}{h}\right)} \end{aligned} \tag{5.58}$$

Here, in the denominator, for clarity of presentation, we have used j instead of i as the index of summation.

There are two ways in which the approximating function $F(\mathbf{x})$ of Eq. (5.58) may be viewed:

1. *Nadaraya–Watson regression estimator.* For the first viewpoint, define the *normalized weighting function*

$$W_{N,i}(\mathbf{x}) = \frac{k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)}{\sum_{j=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_j}{h}\right)}, \qquad i = 1, 2, \ldots, N \tag{5.59}$$

with

$$\sum_{i=1}^{N} W_{N,i}(\mathbf{x}) = 1 \qquad \text{for all } \mathbf{x} \tag{5.60}$$

We may then rewrite the kernel regression estimator of Eq. (5.58) in the simplified form

$$F(\mathbf{x}) = \sum_{i=1}^{N} W_{N,i}(\mathbf{x}) y_i \tag{5.61}$$

which describes $F(\mathbf{x})$ as a *weighted average* of the observables $\{y_i\}_{i=1}^{N}$. The particular form of weighting function $W_N(\mathbf{x}, i)$ given in Eq. (5.61) was independently proposed by Nadaraya (1964) and Watson (1964). Accordingly, the approximating function of Eq. (5.61) is often called the *Nadaraya–Watson regression estimator (NWRE)*.[11]

2. *Normalized RBF network.* For the second viewpoint, we assume *spherical symmetry* of the kernel $k(\mathbf{x})$, in which case we may set

$$k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = k\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|}{h}\right) \qquad \text{for all } i \tag{5.62}$$

where, as usual, $\|\cdot\|$ denotes the Euclidean norm of the enclosed vector (Krzyzak et al., 1996). Correspondingly, we define the *normalized radial basis function*

$$\psi_N(\mathbf{x}, \mathbf{x}_i) = \frac{k\left(\dfrac{\|\mathbf{x} - \mathbf{x}_i\|}{h}\right)}{\displaystyle\sum_{j=1}^{N} k\left(\dfrac{\|\mathbf{x} - \mathbf{x}_j\|}{h}\right)}, \qquad i = 1, 2, ..., N \tag{5.63}$$

with

$$\sum_{i=1}^{N} \psi_N(\mathbf{x}, \mathbf{x}_i) = 1 \qquad \text{for all } \mathbf{x} \tag{5.64}$$

The subscript N in $\psi_N(\mathbf{x}, \mathbf{x}_i)$ signifies the use of *normalization.*

For the regression problem considered under the second viewpoint, we recognize that the "linear weights," w_i, applied to the basic functions $\psi_N(\mathbf{x}, \mathbf{x}_i)$ are simply the observables, y_i of the regression model for the input data $\mathbf{x}_i$. Thus, letting

$$y_i = w_i, \qquad i = 1, 2, ..., N$$

we may reformulate the approximating function of Eq. (5.58) in the general form

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \psi_N(\mathbf{x}, \mathbf{x}_i) \tag{5.65}$$

Equation (5.65) represents the input–output mapping of a *normalized radial-basis-function (RBF) network* (Moody and Darken, 1989; Xu et al., 1994). Note that

$$0 \leq \psi_N(\mathbf{x}, \mathbf{x}_i) \leq 1 \qquad \text{for all } \mathbf{x} \text{ and } \mathbf{x}_i \tag{5.66}$$

Accordingly, $\psi_N(\mathbf{x}, \mathbf{x}_i)$ may be interpreted as the probability of an event described by the input vector $\mathbf{x}$, conditional on $\mathbf{x}_i$.

The basic difference between the normalized radial-basis function $\psi_N(\mathbf{x}, \mathbf{x}_i)$ of Eq. (5.63) and an ordinary radial-basis function is a denominator term that constitutes the *normalization factor.* This normalization factor is an estimate of the underlying probability density function of the input vector $\mathbf{x}$. Consequently, the basis functions $\psi_N(\mathbf{x}, \mathbf{x}_i)$ for $i = 1, 2, \ldots, N$ sum to unity for all $\mathbf{x}$, as described in Eq. (5.64).

Multivariate Gaussian Distribution

A variety of kernel functions is possible in general. However, both theoretical and practical considerations limit the choice. A widely used kernel is the multivariate Gaussian distribution

$$k(\mathbf{x}) = \frac{1}{(2\pi)^{m_0/2}} \exp\left(-\frac{\|\mathbf{x}\|^2}{2}\right) \tag{5.67}$$

where m_0 is the dimension of the input vector $\mathbf{x}$. The spherical symmetry of the kernel $k(\mathbf{x})$ is clearly apparent in Eq. (5.67). Assuming the use of a common bandwidth σ that plays the role of smoothing parameter h for a Gaussian distribution, and centering the kernel on a data point $\mathbf{x}_i$, we may write

$$k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = \frac{1}{(2\pi\sigma^2)^{m_0/2}}\exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2}\right), \qquad i = 1, 2, ..., N \tag{5.68}$$

Thus, using Eq. (5.68), we find that the Nadaraya–Watson regression estimator takes the form

$$F(\mathbf{x}) = \frac{\sum_{i=1}^{N} y_i \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_j\|^2}{2\sigma^2}\right)} \tag{5.69}$$

where the denominator term, representing the Parzen–Rosenblatt density estimator, consists of the sum of N multivariate Gaussian distributions centered on the data points $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ (Specht, 1991).

Correspondingly, using Eq. (5.68) in Eq. (5.63) and then Eq. (5.65), we find that the input–output mapping function of the normalized RBF network takes the form

$$F(\mathbf{x}) = \frac{\sum_{i=1}^{N} w_i \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_j\|^2}{2\sigma^2}\right)} \tag{5.70}$$

In Eqs. (5.69) and (5.70), the centers of the normalized radial-basis functions coincide with the data points $\{\mathbf{x}_i\}_{i=1}^{N}$. As with ordinary radial-basis functions, a smaller number of normalized radial-basis functions can be used, with their centers treated as free parameters to be chosen according to some heuristic (Moody and Darken, 1989), or determined in a principled manner as discussed in Chapter 7.

5.11 SUMMARY AND DISCUSSION

In this chapter, we focused on radial-basis-function (RBF) networks as an alternative to multilayer perceptrons. Like the multilayer perceptron, discussed in Chapter 4, the RBF network is a *universal approximator* in its own right (Sandberg and Xu, 1997a, 1997b). The basic structural difference between them is summed up as follows:

> *In a multilayer perceptron, function approximation is defined by a nested set of weighted summations, whereas in an RBF network, the approximation is defined by a single weighted sum.*

Design Considerations

The design of an RBF network may follow interpolation theory, which, in mathematical terms, is elegant. However, from a practical perspective, this design approach has two shortcomings. First, the training sample may be noisy, which could yield misleading results by the RBF network. Second, when the size of the training sample is large, using an RBF network with a hidden layer of the same size as the training sample is wasteful of computational resources.

A more practical approach to the design of RBF networks is to follow the hybrid learning procedure described in Section 5.5 through 5.7. Basically, the procedure operates in two stages:

- Stage 1 applies the K-means clustering algorithm to train the hidden layer in an unsupervised manner. Typically, the number of clusters, and therefore the number of computational units in the hidden layer, is significantly smaller than the size of the training sample.
- Stage 2 applies the recursive least-squares (RLS) algorithm to compute the weight vector of the linear output layer.

This two-stage design procedure has a couple of desirable features: computational simplicity and accelerated convergence.

Experimental Results

The results of the computer experiment on the double-moon "toy" problem, presented in Section 5.8, reveal that the hybrid "K-means, RLS" classifier is capable of delivering an impressive performance. When the results of this experiment are compared with those of the same experiment using the support vector machine (SVM), to be discussed in the next chapter, we find that the two classifiers perform very similarly. However, the "K-means, RLS" classifier is much faster to converge and less demanding in computational effort than the SVM.

It is noteworthy that Rifkin (2002), in his doctoral thesis, made a detailed comparison between the RLS and the SVM for the classification of linearly separable patterns, using a collection of toy examples. Here is a summary of this part of his experimental results:

- The RLS and SVM classifiers exhibit nearly identical performance.
- They are both susceptible to the presence of outliers in the training sample.

Rifkin (2002) also performed experiments on image classification, using two different data sets:

- The U.S. Postal Service (USPS) handwritten data set, consisting of 7,291 training examples and 2,007 testing examples. The training set contains 6,639 negative examples and 652 positive ones, while the testing set contains 1,807 negative examples and 200 positive ones.
- The MIT recognition set, referred to as **faces**. The training set contains 2,429 faces and 4,548 nonfaces, and the testing set contains 572 faces and 23,573 nonfaces.

For the USPS data set, it is reported that the nonlinear RLS classifier performed as well as or better than the SVM across the entire range of the *receiver operating characteristic* (ROC) curve. The ROC curve plots the *true-positive rate* against the *false-positive rate* for a varying decision threshold when a single network output is used; the term "rate" is another way of referring to the probability of classification. The tests performed on the **faces** set produced mixed results: For one set of design parameters, the SVM performed substantially better than the nonlinear RLS classifier. For another set of design parameters, the performances were close. We should also point out that the strategy used by Rifkin (2002) for designing the hidden layer of the nonlinear RLS classifier was quite different from the K-means clustering algorithm considered in this chapter.

An important message, pertaining to our own double-moon "toy" experiments and the more extensive experiments reported in Rifkin (2002), is twofold:

1. The RLS algorithm has been thoroughly studied in the signal-processing and control literature (Haykin, 2002; Goodwin and Sin, 1984). Unfortunately, it has been almost completely ignored in the machine-learning literature, except for Rifkin's (2002) thesis and a few other publications.
2. There is a need for more extensive experiments, using real-world data sets, to come up with more definitive conclusions on how an RBF network based on the RLS algorithm (for the design of its output layer) and the SVM compare with each other, not only in terms of performance, but also with respect to rate of convergence, as well as computational complexity.

Kernel Regression

The one other important topic studied in this chapter is kernel regression, which builds on the notion of density estimation. In particular, we focused on a nonparametric estimator known as the Parzen–Rosenblatt density estimator, the formulation of which rests on the availability of a kernel. This study led us to two ways of viewing the approximating function defined in terms of a nonlinear regression model: the Nadaraya–Watson regression estimator and the normalized RBF network. For both of them, the multivariate Gaussian distribution provides a good choice for the kernel.

NOTES AND REFERENCES

1. Radial-basis functions were first introduced in the solution of the real multivariate interpolation problem. The early work on this subject is surveyed in Powell (1985). It is now one of the main fields of research in numerical analysis.

   Broomhead and Lowe (1988) were the first to exploit the use of radial-basis functions in the design of neural networks. Another major contribution to the theory and design of radial-basis function networks is due to Poggio and Girosi (1990a). This latter paper emphasizes the use of regularization theory applied to this class of networks as a method for improved generalization to new data; regularization theory is discussed in detail in Chapter 10.
2. The proof of Cover's theorem follows from two basic considerations (Cover, 1965):
   - *Schalfi's theorem*, or the *function-counting theorem*, which states that the number of homogeneously linearly separable dichotomies of N vectors, located in general position in Euclidean-m_1 space, is equal to

   $$C(N, m_1) = 2\sum_{m=0}^{m_1-1}\binom{N-1}{m}$$

     A set of vectors $\mathscr{H} = \{\mathbf{x}_i\}_{i=1}^{N}$ is said to be in "general position" in Euclidean-m_1 space if every subset on m_1 or fewer vectors is linearly independent.
   - *Reflection invariance* of the joint probability distribution of $\mathscr{H}$, which implies that the probability (conditional on $\mathscr{H}$) that a random dichotomy is separable is equal to the

unconditional probability that a particular dichotomy of $\mathscr{H}$ (all N vectors in one category) is separable.

The function-counting theorem has been independently proved in different forms and applied to specific configurations of perceptrons (i.e., linear threshold units) by Cameron (1960), Joseph (1960), and Winder (1961). In Cover (1968), this theorem was applied to evaluate the capacity of a network of perceptrons in terms of the total number of adjustable parameters, which is shown to be lower bounded by $N/(1 + \log_2/N)$, where N is the number of input patterns.

3. Clustering, in general, is discussed in several books, including Theodoridis and Koutroumbas (2003); Duda et al. (2001); and Fukunaga (1990).

The K-means algorithm assumed this name after MacQueen (1967), who studied it as a statistical clustering procedure, including the convergence properties of the algorithm. The idea had been previously described in Foregey (1962) in the context of clustering.

Ding and He (2004) present a very interesting relationship between the K-means algorithm for clustering and principal-components analysis for data reduction. In particular, it is shown that principal components represent a continuous (relaxed) solution of the cluster membership indicators in K-means clustering. In a way, these two views are consistent, in the sense that clustering of data is also a form of data reduction, both of which are, of course, performed in an unsupervised manner. The subject of principal-components analysis is presented in Chapter 8.

In the communications literature dealing with vector quantization, the K-means algorithm is referred to as the *generalized Lloyd algorithm*, which is a generalization of Lloyd's original treatment that appeared in an unpublished 1957 report at Bell Laboratories. Much later, in 1982, Lloyd's report appeared in published form.

4. *Fisher's Linear Discriminant* The cost function defined in Eq. (5.26) is nothing but the *trace* of the so-called within-class covariance (scatter) matrix (Theodoridis and Koutroumbas, 2003).

To understand the meaning of this statement, consider a variable y defined by an inner product as follows:

$$y = \mathbf{w}^T\mathbf{x} \tag{A}$$

The vector $\mathbf{x}$ is drawn from one of two populations $\mathscr{C}_1$ and $\mathscr{C}_2$, which differ from each other by virtue of the mean vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, respectively, and $\mathbf{w}$ is a vector of adjustable parameters. The *Fisher criterion* for discriminating between these two classes is defined by

$$J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{C}_b\mathbf{w}}{\mathbf{w}^T\mathbf{C}_t\mathbf{w}} \tag{B}$$

Where $\mathbf{C}_b$ is the *between-class covariance matrix*, defined by

$$\mathbf{C}_b = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \tag{C}$$

and $\mathbf{C}_t$ is the total *within-class covariance matrix*, defined by

$$\mathbf{C}_t = \sum_{n\in\mathscr{C}_1}(\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^T + \sum_{n\in\mathscr{C}_2}(\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^T \tag{D}$$

The within-class covariance matrix $\mathbf{C}_t$ is proportional to the sample covariance matrix of the training sample. It is symmetric and nonnegative definite and is usually nonsingular if the size of the training sample is large. The between-class covariance matrix $\mathbf{C}_b$ is also symmetric and nonnegative definite, but singular. A property of particular interest is that the matrix product $\mathbf{C}_b\mathbf{w}$ is always in the direction of the difference mean vector $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$. This property follows directly from the definition of $\mathbf{C}_b$.

The expression defining $J(\mathbf{w})$ is known as the *generalized Rayleigh quotient*. The vector $\mathbf{w}$ that maximized $J(\mathbf{w})$ must satisfy the condition

$$\mathbf{C}_b\mathbf{w} = \lambda\mathbf{C}_t\mathbf{w} \tag{E}$$

where λ is a scaling factor. Equation (E) is a generalized eigenvalue problem. Recognizing that, in our case, the matrix product $\mathbf{C}_b\mathrm{w}$ is always in the direction of the difference vector $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$, we find that the solution for Eq. (E) is simply

$$\mathbf{w} = \mathbf{C}_t^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{F}$$

which is referred to as *Fisher's linear discriminant* (Duda et al., 2001).

Taking the trace of the within-class covariance matrix $\mathbf{C}_t$ of Eq. (D), we do find that the cost function of Eq. (5.26) is indeed the trace of this covariance matrix, as already stated.

5. In philosophical terms, the two-step optimization procedure described in the text for the K-means algorithm is similar to the two-step optimization involved in the *EM algorithm*, where the first step is one of expectation, denoted by "E", and the second step is one of maximization, denoted by "M". The EM algorithm was originally developed in the context of maximum-likelihood computation; it is described in Chapter 11.

6. In the literature, the acronym "RLS" is used as the abbreviation for the regularized least-squares algorithm discussed in Chapter 2 as well as the recursive least-squares algorithm discussed in this chapter. From the context of the pertinent discussion, we are usually able to discern which of these two algorithms the acroynm refers to.

7. The essence of the RLS algorithm summarized in Section 5.6, a classic, is described in the books by Diniz (2002); Haykin (2002).

8. Hybrid learning procedures for RBF networks have been variously described in the literature, using different algorithms for the two stages of the procedure; see Moody and Darken (1989) and Lippman (1989b).

9. The conditional mean estimator of Eq. (5.52) is also a minimum mean-square estimator; a proof of this statement is presented in Note 7 of Chapter 14 under the umbrella of Bayes's estimation theory.

10. For a proof of the asymptotically unbiased property of the Parzen–Rosenblatt density estimator, see Parzen (1962) and Cacoullos (1966).

11. The Nadaraya–Watson regression estimator has been the subject of extensive study in the statistics literature. In a broader context, nonparametric functional estimation occupies a central place in statistics; see Härdle (1990) and the collection of papers in Roussas (1991).

PROBLEMS

Cover's Theorem

5.1 As suggested in Section 5.2, the best way to study Eq. (5.5) is to normalize it by setting $N = \lambda m_1$. Using this normalization, plot $P(\lambda m_1, m_1)$ versus λ for $N = 1, 5, 15$, and 25. Hence, validate the two characteristics of Eq. (5.5) described in the section.

5.2 Identify the strong and weak points of Cover's theorem as stated at the beginning of Section 5.2.

5.3 The example given in Fig. 5.1b depicts a spherically separable dictomy. Assume that the four data points outside the separating surface lie on a circle and that the only data point inside lies at the center of the separating surface. Investigate how this sample of data points is nonlinearly transformed, using

(a) the multiquadric

$$\varphi(x) = (x^2 + 1)^{1/2}$$

(b) the inverse multiquadric

$$\varphi(x) = \frac{1}{(x^2 + 1)^{1/2}}$$

K-means Clustering

5.4 Consider the following modification of the cost function defined in Eq. (5.26):

$$J(\boldsymbol{\mu}_j) = \sum_{j=1}^{K} \sum_{i=1}^{N} w_{ij} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$$

In this function, the weighting factor w_{ij} is defined as follows:

$$w_{ij} = \begin{cases} 1 & \text{if the data point } \mathbf{x}_i \text{ lies in cluster } j \\ 0 & \text{otherwise} \end{cases}$$

Show that the minimizing solution of this cost function is

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{i=1}^{N} w_{ij}\mathbf{x}_i}{\sum_{i=1}^{N} w_{ij}}, \qquad j = 1, 2, ..., K$$

How do you interpret the expressions in the numerator and denominator of this formula? Contrast the conclusion from your two answers against that we have learned in the text in the context of clustering.

Recursive Least-Squares Algorithm

5.5 In this problem, we address a statistical interpretation of the matrix **P** defined as the inverse of the correlation matrix **R**.

(a) Using the linear regression model

$$d_i = \mathbf{w}^T \boldsymbol{\phi}_i + \varepsilon_i, \qquad i = 1, 2, ..., N$$

show that the least-square optimized estimate of **w** is expressed as

$$\hat{\mathbf{w}} = \mathbf{w} + (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{\varepsilon}$$

where

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_1^T \\ \boldsymbol{\phi}_2^T \\ \vdots \\ \boldsymbol{\phi}_N^T \end{bmatrix}$$

and

$$\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2, ..., \varepsilon_N]^T$$

Assume that the error ε_i is a sample of a white-noise process of variance σ^2.

(b) Hence, show that the covariance matrix

$$\begin{aligned}\mathbb{E}[(\mathbf{w} - \hat{\mathbf{w}})(\mathbf{w} - \hat{\mathbf{w}})^T] &= \boldsymbol{\sigma}^2\mathbf{R}^{-1} \\ &= \boldsymbol{\sigma}^2\mathbf{P}\end{aligned}$$

where

$$\mathbf{R} = \sum_{i=1}^{N} \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T$$

5.6 Starting with the regularized cost function

$$\mathscr{E}_{\text{av}}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(d(i) - \mathbf{w}^T\boldsymbol{\phi}(i))^2 + \frac{1}{2}\lambda\|\mathbf{w}\|^2$$

do the following:

(a) Show that the addition of the regularization term $\frac{1}{2}\lambda\|\mathbf{w}\|^2$ has no effect whatsoever on the composition of the RLS algorithm, as summarized in Section 5.6.

(b) The only effect of introducing the regularization term is to modify the expression for the correlation matrix of the input data into the form

$$\mathbf{R}(n) = \sum_{i=1}^{n} \boldsymbol{\phi}(i)\boldsymbol{\phi}^T(i) + \lambda\mathbf{I}$$

where $\mathbf{I}$ is the identity matrix. Verify this new formulation of the correlation matrix $\mathbf{R}(n)$, and justify the practical benefit gained by introducing regularization.

5.7 The least-mean-squares (LMS) algorithm for adaptive filtering was discussed in Chapter 3. Compare the advantages and disadvantages of the recursive least-squares (RLS) algorithm with those of the LMS algorithm.

Supervised Training of RBF Networks

5.8 The input–output relationship of a Gaussian-based RBF network is defined by

$$y(i) = \sum_{j=1}^{K} w_j(n)\exp\left(-\frac{1}{2\sigma^2(n)}\|\mathbf{x}(i) - \boldsymbol{\mu}_j(n)\|^2\right), \qquad i = 1, 2, ..., n$$

where $\boldsymbol{\mu}_j(n)$ is the center point of the jth Gaussian unit, the width $\sigma(n)$ is common to all the K units, and $w_j(n)$ is the linear weight assigned to the output of the jth unit; all these parameters are measured at time n. The cost function used to train the network is defined by

$$\mathscr{E} = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

where

$$e(i) = d(i) - y(i)$$

The cost function $\mathscr{E}$ is a convex function of the linear weights in the output layer, but non-convex with respect to the centers and the width of the Gaussian units.

(a) Evaluate the partial derivatives of the cost function with respect to each of the network parameters $w_j(n)$, $\boldsymbol{\mu}_j(n)$, and $\sigma(n)$, for all i.

(b) Use the gradients obtained in part (a) to express the update formulas for all the network parameters, assuming the learning-rate parameters η_w, η_μ, and η_σ for the adjustable parameters of the network, respectively.

(c) The gradient vector $\partial\mathscr{E}/\mu_j(n)$ has an effect on the input data that is similar to *clustering*. Justify this statement.

Kernel Estimation

5.9 Suppose that you are given a "noiseless" training sample $\{f(\mathbf{x}_i)\}_{i=1}^{N}$, and that the requirement is to design a network that generalizes to data samples that are corrupted by additive noise and therefore not included in the training set. Let $F(\mathbf{x})$ denote the approximating function realized by such a network, which is chosen so that the expected squared error

$$J(F) = \frac{1}{2}\sum_{i=1}^{N}\int_{\mathbb{R}^{m_0}} [f(\mathbf{x}_i) - F(\mathbf{x}_i, \boldsymbol{\xi})]^2 f_{\xi}(\boldsymbol{\xi})d\boldsymbol{\xi}$$

is minimum, where $f_\xi(\xi)$ is the probability density function of a noise distribution in the input space $\mathbb{R}^{m_0}$. Show that the solution of this least-squares problem is given as follows (Webb, 1994):

$$F(\mathbf{x}) = \frac{\sum_{i=1}^{N} f(\mathbf{x}_i) f_\xi(\mathbf{x} - \mathbf{x}_i)}{\sum_{i=1}^{N} f_\xi(\mathbf{x} - \mathbf{x}_i)}$$

Compare this estimator with the Nadaraya–Watson regression estimator.

Computer Experiments

5.10 The purpose of this computer experiment is to investigate the clustering process performed by the K-means algorithm. To provide insight into the experiment, we fix the number of clusters at $K = 6$, but vary the vertical separation between the two moons in Fig. 1.8. Specifically, the requirement is to do the following, using an unlabeled training sample of 1,000 data points picked randomly from the two regions of the double-moon pictured in Fig. 1.8:

(a) Experimentally, determine the mean $\hat{\boldsymbol{\mu}}_j$ and variance $\hat{\sigma}_j^2$, $j = 1, 2, ..., 6$, for the sequence of eight uniformly spaced vertical separations starting at $d = 1$ and reducing them by one till the final separation $d = -6$ is reached.

(b) In light of the results obtained in part (a), comment on how the mean $\hat{\boldsymbol{\mu}}_j$ of cluster j is affected by reducing the separation d for $j = 1, 2$, and 3.

(c) Plot the variance $\hat{\sigma}_j^2$ versus the separation d for $j = 1, 2, ..., 6$.

(d) Compare the common σ^2 computed in accordance with the empirical formula of Eq. (5.49) with the trends exhibited in the plots obtained in part (c).

5.11 The purpose of this second experiment is to compare the classification performance of two hybrid learning algorithms: the "*K*-means, RLS" algorithm investigated in Section 5.8 and the "*K*-means, LMS" algorithm investigated in this problem.

As in Section 5.8, assume the following specifications:

Number of hidden Gaussian units: 20
Number of training samples: 1,000 data points
Number of testing samples: 2,000 data points

Let the learning-rate parameter of the LMS algorithm be annealed linearly from 0.6 down to 0.01.

(a) Construct the decision boundary computed for the "*K*-means, LMS" algorithm for the vertical separation between the two moons in Fig. 1.8 set at $d = -5$.

(b) Repeat the experiment for $d = -6$.

(c) Compare the classification results obtained using the "*K*-means, LMS" algorithm with those of the "*K*-means, RLS" algorithm studied in Section 5.8.

(d) Discuss how, in general, the complexity of the "*K*-means, LMS" algorithm compares with that of the "*K*-means, RLS" algorithm.