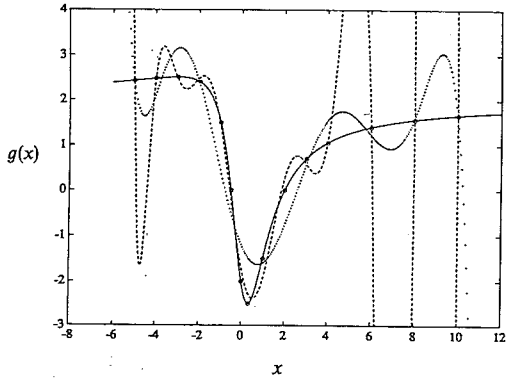

Figure 5.2.2
Polynomial approximation for the function $g(x) = [(x-2)(2x+1)]/(1+x^2)$ (*solid line*) based on the 15 samples shown (*small circles*). The objective of the approximation is to minimize the sum of squared error criterion. The dashed line represents an eleventh-order polynomial. A better overall approximation for $g(x)$ is given by an eighth-order polynomial (*dotted line*).

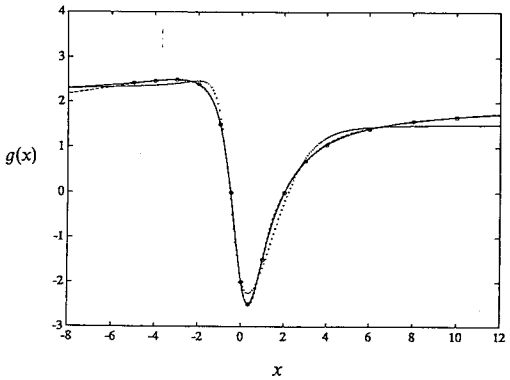

Figure 5.2.3
Neural network approximation for the function $g(x) = [(x-2)(2x+1)]/(1+x^2)$ (*solid line*). The dotted line was generated by a 3-hidden-unit feedforward net. The dashed line, which is shown to have substantial overlap with $g(x)$, was generated by a 12-hidden unit feedforward net. In both cases, standard incremental backprop training was used.

6 Adaptive Multilayer Neural Networks II

The preceding chapter concentrated on multilayer architectures with sigmoidal-type units, both static (feedforward) and dynamic. The present chapter introduces several additional adaptive multilayer networks and their associated training procedures, as well as some variations. The majority of the networks considered here employ processing units that are not necessarily sigmoidal. A common feature of these networks is their fast training as compared with the backprop networks of the preceding chapter. The mechanisms leading to such increased training speed are emphasized.

All networks discussed in this chapter differ in one or more significant ways from those in the preceding chapter. One group of networks employs units with localized receptive fields, where units receiving direct input from input signals (patterns) can only "see" a part of the input pattern. Examples of such networks are the radial basis function network and the cerebellar model articulation controller.

A second group of networks employs resource allocation. These networks are capable of allocating units as needed during training. This feature enables the network size to be determined dynamically and eliminates the need for guessing the proper network size. This resource-allocating scheme is also shown to be the primary reason for efficient training. Examples of networks in this group are hyperspherical classifiers and the cascade-correlation network.

These two groups of networks mainly employ supervised learning. Some of these networks may be used as function interpolators/approximators, while others are best suited for classification tasks. The third and last group of adaptive multilayer networks treated in this chapter has the capability of unsupervised learning or clustering. Here, two specific clustering nets are discussed: the ART1 network and the auto-associative clustering network.

Throughout this chapter, fundamental similarities and differences among the various networks are stressed. In addition, significant extensions of these networks are pointed out, and the effects of these extensions on performance are discussed.

6.1 Radial Basis Function (RBF) Networks

This section describes an artificial neural network model motivated by the "locally tuned" response observed in biologic neurons. Neurons with locally tuned response characteristics can be found in many parts of biologic nervous systems. These nerve cells have response characteristics that are "selective" for some finite range of the input signal space. The cochlear stereocilia cells, for example, have a locally tuned response to frequency that is a consequence of their biophysical properties. The present model is also motivated by earlier work on radial basis functions (Medgassy, 1961) that were used for interpolation (Micchelli, 1986; Powell, 1987), probability

Mohamad H. Hassoun: Fundamentals of Artificial Neural Networks, pp. 285-295, MIT Press, 1995.

density estimation (Parzen, 1962; Duda and Hart, 1973; Specht, 1990), and approximations of smooth multivariate functions (Poggio and Girosi, 1989). The model is commonly referred to as the *radial basis function (RBF) network.*

The most important feature that distinguishes the RBF network from earlier radial basis function–based models is its adaptive nature, which generally allows it to utilize a relatively smaller number of locally tuned units (RBFs). RBF networks were independently proposed by Broomhead and Lowe (1988), Lee and Kil (1988), Niranjan and Fallside (1988), and Moody and Darken (1989a, 1989b). Similar schemes were also suggested by Hanson and Burr (1987), Lapedes and Farber (1987), Casdagli (1989), Poggio and Girosi (1990b), and others. The following is a description of the basic RBF network architecture and its associated training algorithm.

The RBF network has a feedforward structure consisting of a single hidden layer of J locally tuned units which are fully interconnected to an output layer of L linear units, as shown in Figure 6.1.1. All hidden units simultaneously receive the n-dimensional real-valued input vector $\mathbf{x}$. Notice the absence of hidden-layer weights in Figure 6.1.1. This is because the hidden-unit outputs are not calculated using the weighted-sum/sigmoidal activation mechanism as in the preceding chapter. Rather, here, each hidden-unit output z_j is obtained by calculating the "closeness" of the input

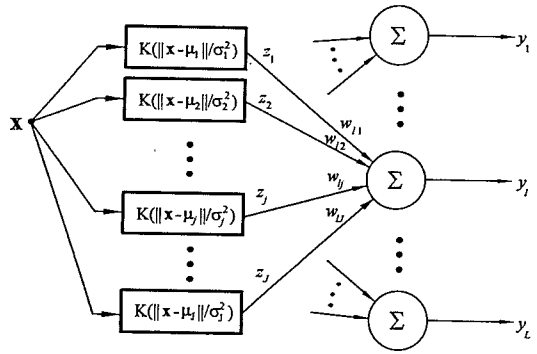

Figure 6.1.1
A radial basis function neural network consisting of a single hidden layer of locally tuned units that is fully interconnected to an output layer of linear units. For clarity, only hidden to output layer connections for the lth output unit are shown.

$\mathbf{x}$ to an n-dimensional parameter vector $\boldsymbol{\mu}_j$ associated with the jth hidden unit. Here, the response characteristics of the jth hidden unit are given by

$$z_j(\mathbf{x}) = K\left(\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|}{\sigma_j^2}\right) \tag{6.1.1}$$

where K is a strictly positive radially symmetric function (kernel) with a unique maximum at its "center" $\boldsymbol{\mu}_j$ and which drops off rapidly to zero away from the center. The parameter σ_j is the "width" of the receptive field in the input space for unit j. This implies that z_j has an appreciable value only when the "distance" $\|\mathbf{x} - \boldsymbol{\mu}_j\|$ is smaller than the width σ_j. Given an input vector $\mathbf{x}$, the output of the RBF network is the L-dimensional activity vector $\mathbf{y}$ whose lth component is given by[1]

$$y_l(\mathbf{x}) = \sum_{j=1}^{J} w_{lj} z_j(\mathbf{x}) \tag{6.1.2}$$

It is interesting to note here that for $L = 1$, the mapping in Equation (6.1.2) is similar in form to that employed by a polynomial threshold gate (PTG), as in Equation (1.4.1). However, in the RBF net, a choice is made to use radially symmetric kernels as "hidden units" as opposed to monomials.

RBF networks are best suited for approximating continuous or piecewise continuous real-valued mappings $f: R^n \rightarrow R^L$, where n is sufficiently small; these approximation problems include classification problems as a special case. According to Equations (6.1.1) and (6.1.2), the RBF network may be viewed as approximating a desired function $f(\mathbf{x})$ by superposition of nonorthogonal bell-shaped basis functions. The degree of accuracy can be controlled by three parameters: the number of basis functions used, their location, and their width. In fact, like feedforward neural networks with a single hidden layer of sigmoidal units, it can be shown that RBF networks are universal approximators (Poggio and Girosi, 1989; Hartman et al., 1990; Baldi, 1991; Park and Sandberg, 1991, 1993).

A special but commonly used RBF network assumes a Gaussian basis function for the hidden units:

$$z_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \tag{6.1.3}$$

where σ_j and $\boldsymbol{\mu}_j$ are the standard deviation and mean of the jth unit receptive field,

1. A bias can be added to the output units if needed. Here, one may treat the Jth hidden unit as a dummy unit by clamping its output z_J to $+1$. Thus w_{lJ} becomes the weight of the lth output unit associated with the bias input.

respectively, and the norm is the Euclidean norm. Another possible choice for the basis function is the logistic function of the form

$$z_j(\mathbf{x}) = \left[1 + \exp\left(\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|}{\sigma_j^2} - \theta_j\right)\right]^{-1} \tag{6.1.4}$$

where θ_j is an adjustable bias. In fact, with the basis function in Equation (6.1.4), the only difference between an RBF network and a feedforward neural network with a single hidden layer of sigmoidal units is the similarity computation performed by the hidden units. If we think of $\boldsymbol{\mu}_j$ as the parameter (weight) vector associated with the jth hidden unit, then it is easy to see that an RBF network can be obtained from a single-hidden-layer neural network with unipolar sigmoidal-type units and linear output units (like the one in Figure 5.1.1) by simply replacing the jth hidden-unit weighted-sum $net_j = \mathbf{x}^T\boldsymbol{\mu}_j$ by the negative of the normalized Euclidean distance $\|\mathbf{x} - \boldsymbol{\mu}_j\|/\sigma_j^2$. On the other hand, use of the Gaussian basis function in Equation (6.1.3) leads to hidden units with Gaussian-type activation functions and with a Euclidean distance similarity computation. In this case, no bias is needed.

Next, the training of RBF networks is addressed. Consider a training set of m labeled pairs $\{\mathbf{x}^i, \mathbf{d}^i\}$ that represent associations of a given mapping or samples of a continuous multivariate function. Also consider the SSE criterion function as an error function E to be minimized over the given training set. In other words, it is desired to develop a training method that minimizes E by adaptively updating the free parameters of the RBF network. These parameters are the receptive field centers (means $\boldsymbol{\mu}_j$ of the hidden layer Gaussian units), the receptive field widths (standard deviations σ_j), and the output layer weights (w_{ij}).

Because of the differentiable nature of the RBF network's transfer characteristics, one of the first training methods that comes to mind is a fully supervised gradient-descent method over E (Moody and Darken, 1989a; Poggio and Girosi, 1989). In particular, $\boldsymbol{\mu}_j$, σ_j, and w_{ij} are updated as follows: $\Delta\boldsymbol{\mu}_j = -\rho_\mu \nabla_{\boldsymbol{\mu}_j} E$, $\Delta\sigma_j = -\rho_\sigma \dfrac{\partial E}{\partial \sigma_j}$, and $\Delta w_{ij} = -\rho_w \dfrac{\partial E}{\partial w_{ij}}$, where ρ_μ, ρ_σ, and ρ_w are small positive constants. This method, although capable of matching or exceeding the performance of backprop-trained networks, still gives training times comparable with those of sigmoidal-type networks (Wettschereck and Dietterich, 1992).

One reason for the slow convergence of the preceding supervised gradient-descent-trained RBF network is its inefficient use of the locally tuned representation of the hidden-layer units. When the hidden-unit receptive fields are narrow, only a small

fraction of the total number of units in the network will be activated for a given input $\mathbf{x}$; the activated units are the ones with centers very close to the input vector in the input space. Thus only those units which were activated need be updated for each input presentation. The preceding supervised learning method, though, places no restrictions on maintaining small values for σ_j. Thus the supervised learning method is not guaranteed to utilize the computational advantages of locality. One way to rectify this problem is to only use gradient-descent-based learning for the basis function centers and use a method that maintains small σ_j values. Examples of learning methods that take advantage of the locality property of the hidden units are presented below.

A training strategy that decouples learning at the hidden layer from that at the output layer is possible for RBF networks because of the local receptive field nature of the hidden units. This strategy has been shown to be very effective in terms of training speed; however, this advantage is generally offset by reduced generalization ability unless a large number of basis functions is used. In the following, efficient methods for locating the receptive field centers and computing receptive field widths are described. As for the output-layer weights, once the hidden units are synthesized, these weights can be computed easily using the delta rule [Equation (5.1.2)]. One may view this computation as finding the proper normalization coefficients of the basis functions. That is, the weight w_{ij} determines the amount of contribution of the jth basis function to the lth output of the RBF net.

Several schemes have been suggested to find proper receptive field centers and widths without propagating the output error back through the network. The idea here is to populate dense regions of the input space with receptive fields. One method places the centers of the receptive fields according to some coarse lattice defined over the input space (Broomhead and Lowe, 1988). Assuming a uniform lattice with k divisions along each dimension of an n-dimensional input space, this lattice would require k^n basis functions to cover the input space. This exponential growth renders this approach impractical for a high dimensional space. An alternative approach is to center k receptive fields on a set of k randomly chosen training samples. Here, unless there is prior knowledge about the location of prototype input vectors and/or the regions of the input space containing meaningful data, a large number of receptive fields would be required to adequately represent the distribution of the input vectors in a high dimensional space.

Moody and Darken (1989a) employed unsupervised learning of the receptive field centers $\boldsymbol{\mu}_j$ in which a relatively small number of RBFs are used; the adaptive centers learn to represent only the parts of input space which are richly represented by clusters of data. The adaptive strategy also helps reduce sampling error because it

allows the centers $\boldsymbol{\mu}$ to be determined by a large number of training samples. Here, the k-means clustering algorithm (MacQueen, 1967; Anderberg, 1973) is used to locate a set of k RBF centers that represents a local minimum of the SSE between the training set vectors $\mathbf{x}$ and the nearest of the k receptive field centers $\boldsymbol{\mu}_j$ [this SSE criterion function is given by Equation (4.6.4) with $\mathbf{w}$ replaced by $\boldsymbol{\mu}$]. In the basic k-means algorithm, the k RBFs are initially assigned centers $\boldsymbol{\mu}_j$, $j = 1, 2, \ldots, k$, which are set equal to k randomly selected training vectors. The remaining training vectors are assigned to class j of the closest center $\boldsymbol{\mu}_j$. Next, the centers are recomputed as the average of the training vectors in their class. This two-step process is invoked until all centers stop changing. An incremental version of this batch-mode process also may be used that requires no storage of past training vectors or cluster membership information. Here, at each time step, a random training vector $\mathbf{x}$ is selected and the center $\boldsymbol{\mu}_j$ of the nearest (in a Euclidean distance sense) receptive field is updated according to

$$\Delta\boldsymbol{\mu}_j = \rho(\mathbf{x} - \boldsymbol{\mu}_j) \tag{6.1.5}$$

where ρ is a small positive constant. Equation (6.1.5) is the simple competitive rule that was analyzed in Section 4.6.1. Similarly, learning vector quantization (LVQ) or one of its variants (see Section 3.4.2) may be used to effectively locate the k RBF centers (Vogt, 1993). Generally speaking, there is no formal method for specifying the required number k of hidden units in an RBF network. Cross-validation is normally used to decide on k.

Once the receptive field centers are found using one of the preceding methods, their widths can be determined by one of several heuristics in order to get smooth interpolation. Theoretically speaking, RBF networks with the same σ_j in each hidden kernel unit have the capability of universal approximation (Park and Sandberg, 1991, 1993). This suggests that one may simply use a single global fixed value σ for all σ_j values in the network. In order to preserve the local response characteristics of the hidden units, one should choose a relatively small (positive) value for this global width parameter. The actual value of σ for a particular training set may be found by cross-validation. Empirical results (Moody and Darken, 1989a) suggest that a "good" estimate for the global width parameter is the average width $\sigma = \langle \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\| \rangle$, which represents a global average over all Euclidean distances between the center of each unit i and that of its nearest neighbor j. Other heuristics based on local computations may be used which yield individually tuned widths σ_j. For example, the width for unit j may be set to the distance $\alpha\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|$, where $\boldsymbol{\mu}_i$ is the center of the nearest neighbor to unit j (usually α is taken between 1.0 and 1.5). For classification tasks, one may make use of the category label of the nearest training vector. If that category label is different from that represented by the current RBF unit, it would be advisable to use a smaller width, which narrows the bell-shaped receptive field of the current unit. This leads to a sharpening of the class domains and allows for better approximation.

It has already been noted that the output layer weights w_{ij} can be adaptively computed using the delta rule

$$\Delta w_{ij} = \rho(d_i - y_i)f'(net_i)z_j \qquad i = 1, 2, \ldots, L; \quad j = 1, 2, \ldots, J \tag{6.1.6}$$

once the hidden-layer parameters are obtained. Here, the term $f'(net_i)$ can be dropped for the case of linear units. Equation (6.1.6) drives the output layer weights to minimize the SSE criterion function [recall Equation (5.1.13)] for sufficiently small ρ. Alternatively, for the case of linear output units, one may formulate the problem of computing the weights as a set of simultaneous linear equations and employ the generalized-inverse method [recall Equation (3.1.42)] to obtain the minimum SSE solution. Without loss of generality, consider a single-output RBF net, and denote by $\mathbf{w} = [w_1 \quad w_2 \quad \cdots \quad w_J]^T$ the weight vector of the output unit. Now, recalling Equations (3.1.39) through (3.1.42), the minimum SSE solution for the system of equations $\mathbf{Z}^T\mathbf{w} = \mathbf{d}$ is given by (assuming an overdetermined system, i.e., $m \geq J$)

$$\mathbf{w}^* = \mathbf{Z}^\dagger\mathbf{d} = (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{d} \tag{6.1.7}$$

where $\mathbf{Z} = [\mathbf{z}^1 \quad \mathbf{z}^2 \quad \cdots \quad \mathbf{z}^m]$ is a $J \times m$ matrix, and $\mathbf{d} = [d^1 \quad d^2 \quad \cdots \quad d^m]^T$. Here, $\mathbf{z}^i$ is the output of the hidden layer for input $\mathbf{x}^i$. Therefore, the jith element of matrix $\mathbf{Z}$ may be expressed explicitly as

$$\mathbf{Z}_{ji} = K\left(\frac{\|\mathbf{x}^i - \boldsymbol{\mu}_j\|}{\sigma_j^2}\right) \qquad i = 1, 2, \ldots, m; \quad j = 1, 2, \ldots, J \tag{6.1.8}$$

with the parameters $\boldsymbol{\mu}_j$ and σ_j^2 assumed to have been computed using the earlier described methods.

For "strict" interpolation problems, it is desired that an interpolation function be found that is constrained to "exactly" map the sample points $\mathbf{x}^i$ into their associated targets d^i, for $i = 1, 2, \ldots, m$. It is well known that a polynomial with finite order $r = m - 1$ is capable of performing strict interpolation on m samples $\{\mathbf{x}^i, d^i\}$, assuming distinct vectors $\mathbf{x}^i$ in R^n (see Problem 1.3.4). A similar result is available for RBF nets. This result states that there is a class of radial basis functions that guarantee that an RBF net with m such functions is capable of strict interpolation of m sample points in R^n (Micchelli, 1986; Light, 1992b); the Gaussian function in Equation (6.1.3) is one example. Furthermore, there is no need to search for the centers $\boldsymbol{\mu}_j$; one can just set $\boldsymbol{\mu}_j = \mathbf{x}^j$ for $j = 1, 2, \ldots, m$. Thus, for strict interpolation, the $\mathbf{Z}$ matrix in Equation (6.1.8) becomes the $m \times m$ matrix

$$Z_{ji} = K\left(\frac{\|\mathbf{x}^i - \mathbf{x}^j\|}{\sigma_j^2}\right) \qquad i = 1,2,\ldots,m; \quad j = 1,2,\ldots,m \tag{6.1.9}$$

which we refer to as the *interpolation matrix*. Note that the appropriate width parameters σ_j still need to be found; the choice of these parameters affects the interpolation quality of the RBF net.[2]

According to the preceding discussion, an exact solution $\mathbf{w}^*$ is ensured. This requires $\mathbf{Z}$ to be nonsingular. Hence $\mathbf{w}^*$ can be computed as

$$\mathbf{w}^* = (\mathbf{Z}^T)^{-1}\mathbf{d} \tag{6.1.10}$$

Although, in theory, Equation (6.1.10) always ensures a solution to the strict interpolation problem, in practice, the direct computation of $(\mathbf{Z}^T)^{-1}$ can become ill-conditioned when $\mathbf{Z}^T$ is nearly singular. Alternatively, one may resort to Equation (6.1.6) for an adaptive computation of $\mathbf{w}^*$.

Receptive field properties play an important role in the quality of an RBF network's approximation capability. To see this, consider a single-input/single-output RBF network for approximating a continuous function $f: R \to R$. Approximation error, due to error in the "fit" of the RBF network to that of the target function f, occurs when the receptive fields (e.g., Gaussians) are either too broad and/or too widely spaced relative to the fine spatial structure of f. In other words, these factors act to locally limit the high-frequency content of the approximating network. According to Nyquist's sampling criterion, the highest frequency that may be recovered from a sampled signal is one-half the sampling frequency. Therefore, when the receptive field density is not high enough, the high-frequency fine structure in the function being approximated is lost. The high-frequency fine structure of f also can be "blurred" when the receptive fields are excessively wide. By employing the Taylor series expansion, it can be shown (Hoskins et al., 1993) that when the width parameter is large, the RBF net exhibits polynomial behavior with an order successively decreasing as the RBF widths increase. In other words, the net's output approaches that of a polynomial function whose order is decreasing in σ. Therefore, it is important that receptive

2. This can be demonstrated as follows: Assume that a smooth nonlinear function is to be approximated from a finite set of noiseless samples. Also, assume that K is Gaussian and that all σ_j values approach zero. This makes the matrix $\mathbf{Z}$ approach the identity matrix, which from Equation (6.1.10) gives the simple solution $\mathbf{w}^* = \mathbf{d}$. Therefore, the RBF net will output the correct value of the function being approximated if the input to the net is one of the training samples. On the other hand, the net will output a near-zero output for all points between the sample points. This is because the output of all hidden units is practically zero, due to the negligible size of their receptive fields. This analysis suggests that successful interpolation requires some degree of overlap between the receptive fields, which can be accomplished by properly setting the σ_j parameters.

field densities and widths be chosen to match the frequency-transfer characteristics imposed by the function f (Mel and Omohundro, 1991). These results also suggest that even for moderately high dimensional input spaces, a relatively large number of RBFs must be used if the training data represent high-frequency content mappings (functions) and if low approximation error is desired. These observations can be generalized to the case of RBF network approximation of multivariate functions.

Example 6.1.1 This example illustrates application of the RBF net for approximating the function $g(x) = [(x-2)(2x+1)]/(1+x^2)$ (refer to the solid line plot in Figure 6.1.2) from the 15 noise-free samples $(x^j, g(x^j))$, $j = 1,2,\ldots,15$, in Figure 6.1.2. We will employ the method of strict interpolation for designing the RBF net. Hence 15 Gaussian hidden units are used (all having the same width parameter σ), with the jth Gaussian unit having its center μ^j equal to x^j. The design is completed by computing the weight vector $\mathbf{w}$ of the output linear unit using Equation (6.1.10). Three designs are generated which correspond to the values $\sigma = 0.5$, 1.0 and 1.5. These networks are then tested with 200 inputs, x, uniformly sampled in the interval $[-8, 12]$. The output

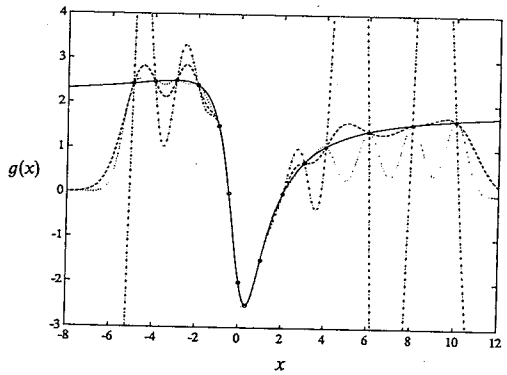

Figure 6.1.2
RBF net approximation of the function $g(x) = [(x-2)(2x+1)]/(1+x^2)$ (*solid line*), based on strict interpolationusing the 15 samples shown (*small circles*). The RBF net employs 15 Gaussian hidden units, and its output is shown for three hidden unit widths: $\sigma = 0.5$ (*dotted line*), $\sigma = 1.0$ (*dashed line*), and $\sigma = 1.5$ (dotted-dashed line). (Compare these results with those in Figures 5.2.2 and 5.2.3.)

of the RBF net is shown in Figure 6.1.2 for $\sigma = 0.5$ (dotted line), $\sigma = 1.0$ (dashed line), and $\sigma = 1.5$ (dotted-dashed line). The value of $\sigma = 1.0$ is close to the average distance among all 15 sample points, and it resulted in better interpolation of $g(x)$ compared with $\sigma = 0.5$ and $\sigma = 1.5$. As expected, these results show poor extrapolation capabilities by the RBF net, regardless of the value of σ (check the net output in Figure 6.1.2 for $x > 10$ and $x < -5$). It is interesting to note the excessive overfit by the RBF net for relatively high σ (compare with the polynomial-based strict interpolation of the same data shown in Figure 5.2.2). Finally, by comparing these results with those in Figure 5.2.3, one can see that more accurate interpolation is possible with sigmoidal hidden-unit nets; this is attributed mainly to the ability of feedforward multilayer sigmoidal unit nets to approximate the first derivative of $g(x)$.

6.1.1 RBF Networks versus Backprop Networks

RBF networks have been applied with success to function approximation (Broomhead and Lowe, 1988; Lee and Kil, 1988; Casdagli, 1989; Moody and Darken, 1989a, 1989b) and classification (Niranjan and Fallside, 1988; Nowlan, 1990; Lee, 1991; Wettschereck and Dietterich, 1992; Vogt, 1993). On difficult approximation/prediction tasks [e.g., predicting the Mackey-Glass chaotic series of Problem 5.4.1 T time steps ($T > 50$) in the future], RBF networks that employ clustering for locating hidden-unit receptive field centers can achieve a performance comparable with backprop networks (backprop-trained feedforward networks with sigmoidal hidden units) while requiring orders of magnitude less training time than backprop. However, the RBF network typically requires 10 times or more data to achieve the same accuracy as a backprop network. The accuracy of RBF networks may be further improved if supervised learning of receptive field centers is used (Wettschereck and Dietterich, 1992), but then the speed advantage over backprop networks is compromised. For difficult classification tasks, RBF networks or their modified versions (see Section 6.1.2) employing sufficient training data and hidden units can lead to better classification rates (Wettschereck and Dietterich, 1992) and smaller false-positive classification errors (Lee, 1991) compared with backprop networks. In the following, qualitative arguments are given for the preceding simulation-based observations on the performance of RBF and backprop networks.

Some of the reasons for the training speed advantage of RBF networks have been presented earlier in this section. Basically, since the receptive field representation is well localized, only a small fraction of the hidden units in an RBF network respond to any particular input vector. This allows the use of efficient self-organization (clustering) algorithms for adapting such units in a training mode that does not involve the

network's output units. On the other hand, all units in a backprop network must be evaluated and their weights updated for every input vector. Another important reason for the faster training speed of RBF networks is the hybrid two-stage training scheme employed, which decouples the learning task for both hidden and output layers, thus eliminating the need for the slow back error propagation.

The RBF network with self-organized receptive fields needs more data and more hidden units to achieve similar precision to that of the backprop network. When used for function approximation, the backprop network performs global fit to the training data, whereas the RBF network performs local fit. This results in greater generalization by the backprop network from each training example. It also utilizes the network's free parameters more efficiently, which leads to a smaller number of hidden units. Furthermore, the backprop network is a better candidate net when extrapolation is desired. This is due primarily to the ability of feedforward nets with sigmoidal hidden units to approximate a function and its derivatives (see Section 5.2.5). On the other hand, the local nature of the hidden-unit receptive fields in RBF nets prevents them from being able to "see" beyond the training data. This makes the RBF net a poor extrapolator.

When used as a classifier, the RBF net can lead to low false-positive classification rates. This property is due to the same reason that makes RBF nets poor extrapolators. Regions of the input space which are far from training vectors are usually mapped to low values by the localized receptive fields of the hidden units. By contrast, the sigmoidal hidden units in the backprop network can have high output even in regions far away from those populated by training data. This causes the backprop network/classifier to indicate high-confidence classifications to meaningless inputs. False-positive classification may be reduced in backprop networks by employing the "training with rubbish" strategy discussed at the end of Section 5.3.3. However, when dealing with high dimensional input spaces, this strategy generally requires an excessively large training set due to the large number of possible rubbish pattern combinations.

Which network is better to use for which tasks? The backprop network is better to use when training data are expensive (or hard to generate) and/or retrieval speed, assuming a serial machine implementation, is critical (the smaller backprop network size requires less storage and leads to faster retrievals compared with RBF networks). However, if the data are cheap and plentiful, and if on-line training is required (e.g., the case of adaptive signal processing or adaptive control where data are acquired at a high rate and cannot be saved), then the RBF network is superior.