ELSEVIER

Neurocomputing 48 (2002) 455–475

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Training radial basis neural networks with the extended Kalman filter

Dan Simon

Department of Electrical and Computer Engineering, Cleveland State University, Stilwell Hall Room 332, 1960 East 24th Street, Cleveland, OH 44115-2425, USA

Received 26 February 2000; accepted 24 May 2001

Abstract

Radial basis function (RBF) neural networks provide attractive possibilities for solving signal processing and pattern classification problems. Several algorithms have been proposed for choosing the RBF prototypes and training the network. The selection of the RBF prototypes and the network weights can be viewed as a system identification problem. As such, this paper proposes the use of the extended Kalman filter for the learning procedure. After the user chooses how many prototypes to include in the network, the Kalman filter simultaneously solves for the prototype vectors and the weight matrix. A decoupled extended Kalman filter is then proposed in order to decrease the computational effort of the training algorithm. Simulation results are presented on reformulated radial basis neural networks as applied to the Iris classification problem. It is shown that the use of the Kalman filter results in better learning than conventional RBF networks and faster learning than gradient descent.

Keywords: Radial basis function (RBF); Training; Optimization; Gradient descent; Kalman filter

1. Introduction

A radial basis function (RBF) neural network is trained to perform a mapping from an m-dimensional input space to an n-dimensional output space. RBFs can be used for discrete pattern classification, function approximation, signal processing, control, or any other application which requires a mapping from an input to an output. RBFs were first used for neural networks in [4]. Many RBF papers and

E-mail address: d.j.simon@csuohio.edu (D. Simon).

0925-2312/02/$ - see front matter
PII: S0925-2312(01)00611-7

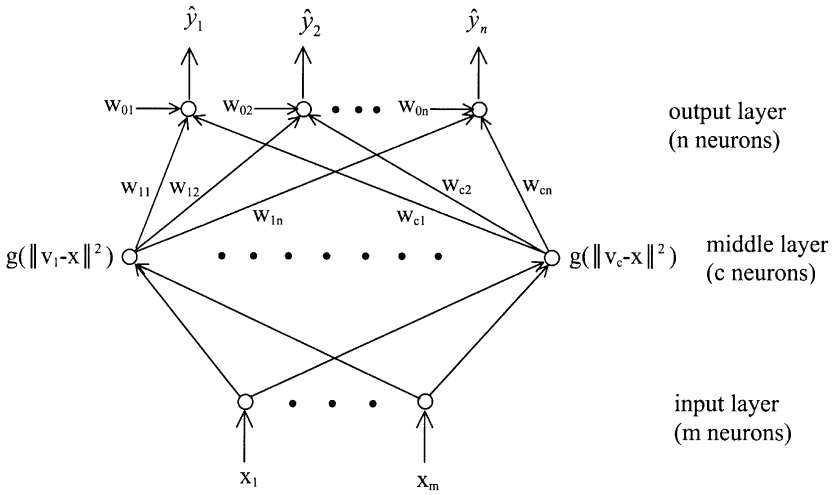

Fig. 1. Radial basis function network architecture.

references can be found in the recent *Neurocomputing* special issues on RBF networks [15,16].

An RBF consists of the m-dimensional input x being passed directly to a hidden layer. Suppose there are c neurons in the hidden layer. Each of the c neurons in the hidden layer applies an activation function which is a function of the Euclidean distance between the input and an m-dimensional prototype vector. Each hidden neuron contains its own prototype vector as a parameter. The output of each hidden neuron is then weighted and passed to the output layer. The outputs of the network consist of sums of the weighted hidden layer neurons. Fig. 1 shows a schematic of an RBF network. It can be seen that the design of an RBF requires several decisions, including the following:

1. How many hidden units will reside in the hidden layer (i.e., what is the value of the integer c);
2. What are the values of the prototypes (i.e., what are the values of the v vectors);
3. What function will be used at the hidden units (i.e., what is the function $g(\cdot)$);
4. What weights will be applied between the hidden layer and the output layer (i.e., what are the values of the w weights).

The performance of an RBF network depends on the number and location (in the input space) of the centers, the shape of the RBF functions at the hidden units, and the method used for determining the network weights. Some researchers have trained RBF networks by selecting the centers randomly from the training data [4]. Others have used unsupervised procedures (such as the k-means algorithm) for selecting the RBF centers [11]. Still others have used supervised procedures for selecting the RBF centers [9].

Several training methods separate the tasks of prototype determination and weight optimization. This trend probably arose because of the quick training that could result from the separation of the two tasks. In fact, one of the primary contributors to the popularity of RBF networks was probably their fast training times as compared to gradient descent training (include backpropagation). Reviewing Fig. 1, it can be seen that once the prototypes are fixed and the hidden layer function $g(\cdot)$ is known, the network is linear in the weight parameters w. At that point training the network becomes a quick and easy task that can be solved via linear least squares. (This is similar to the popularity of the optimal interpolative net that is due in large part to the efficient noniterative learning algorithms that are available [18,19].)

Training methods that separate the tasks of prototype determination and weight optimization often do not use the input–output data from the training set for the selection of the prototypes. For instance, the random selection method and the k-means algorithm result in prototypes that are completely independent of the input–output data from the training set. Although this results in fast training, it clearly does not take full advantage of the information contained in the training set.

Gradient descent training of RBF networks has proven to be much more effective than more conventional methods [9]. However, gradient descent training can be computationally expensive. This paper extends the results of [9] and formulates a training method for RBFs based on Kalman filtering. This new method proves to be quicker than gradient descent training while still providing performance at the same level of effectiveness.

Training a neural network is, in general, a challenging nonlinear optimization problem. Various derivative-based methods have been used to train neural networks, including gradient descent [9], Kalman filtering [20,21], and the well-known backpropagation [7]. Derivative-free methods, including genetic algorithms [6], learning automata [12], and simulated annealing [10], have also been used to train neural networks.

Derivative-free methods have the advantage that they do not require the derivative of the objective function with respect to the neural network parameters. They are more robust than derivative-based methods with respect to finding a global minimum and with respect to their applicability to a wide range of objective functions and neural network architectures. However, they typically tend to converge more slowly than derivative-based methods. Derivative-based methods have the advantage of fast convergence, but they tend to converge to local minima. In addition, due to their dependence on analytical derivatives, they are limited to specific objective functions and specific types of neural network architectures.

In this paper we formulate a training method for RBFs that is based on Kalman filtering. Kalman filters have been used extensively with neural networks. They have been used to train multilayer perceptrons [17,20,21] and recurrent networks [13,14]. They have also been used to train RBF networks, but so far their application has been restricted to single-output networks with exponential functions at

the hidden layer [3]. In the present paper we extend the use of Kalman filters to the training of general multi-input, multi-output RBF networks.

For linear dynamic systems with white process and measurement noise, the Kalman filter is known to be an optimal estimator. For nonlinear systems with colored noise, the Kalman filter can be extended by linearizing the system around the current parameter estimates. This algorithm updates parameters in a way that is consistent with all previously measured data and generally converges in a few iterations. In the following sections we describe how the extended Kalman filter can be applied to RBF optimization. We demonstrate its performance on the Iris classification problem and compare it with RBF optimization using gradient descent.

The next section provides an overview of reformulated RBFs, and Section 3 discusses how gradient-based methods can optimize the weights and prototype vectors of an RBF. Section 4 shows how an extended Kalman filter can optimize the parameters of an RBF, and Section 5 proposes a modification of the Kalman filter in order to decrease the computational effort for large problems. Section 6 contains simulation results and a comparison of the Kalman filter method with the gradient descent method, and Section 7 contains some concluding remarks and suggestions for further research.

2. Reformulated radial basis functions

There have been a number of popular choices for the $g(\cdot)$ function at the hidden layer of RBFs [5] (see Fig. 1). The most common choice is a Gaussian function of the form

$$g(v) = \exp(-v/\beta^2), \tag{1}$$

where β is a real constant. Other hidden layer functions that have often been used are the thin plate spline function

$$g(v) = v \log \sqrt{v} \tag{2}$$

the multiquadric function

$$g(v) = (v^2 + \beta^2)^{1/2} \tag{3}$$

and the inverse multiquadric function

$$g(v) = (v^2 + \beta^2)^{-1/2}, \tag{4}$$

where β is a real constant.

It has recently been proposed [9] that since RBF prototypes are generally interpreted as the centers of receptive fields, hidden layer functions should have the following properties:

1. The response at a hidden neuron is always positive;
2. The response at a hidden neuron becomes stronger as the input approaches the prototype;

3. The response at a hidden neuron becomes more sensitive to the input as the input approaches the prototype.

With these desired properties in mind, an RBF's hidden layer function should be of the general form

$$g(v) = [g_0(v)]^{1/(1-p)} \tag{5}$$

and one of two sets of conditions on the so-called generator function $g_0(v)$ should hold. If p is a real number greater than 1, then the generator function $g_0(v)$ should satisfy the following set of conditions:

1. $g_0(v) > 0 \ \forall v \in (0, \infty)$;
2. $g_0'(v) > 0 \ \forall v \in (0, \infty)$;
3. $\frac{p}{p-1}[g_0'(v)]^2 - g_0(v)g_0''(v) > 0 \ \forall v \in (0, \infty)$.

The last two conditions ensure that $g'(v) < 0$ and $g''(v) > 0 \ \forall v \in (0, \infty)$. If p is a real number less than 1, then the generator function $g_0(v)$ should satisfy the following set of conditions:

1. $g_0(v) > 0 \ \forall v \in (0, \infty)$;
2. $g_0'(v) < 0 \ \forall v \in (0, \infty)$;
3. $\frac{p}{p-1}[g_0'(v)]^2 - g_0(v)g_0''(v) < 0 \ \forall \ v \in (0, \infty)$.

The last two conditions again ensure that $g'(v) < 0$ and $g''(v) > 0 \ \forall v \in (0, \infty)$. It can be shown that the functions of Eqs. (1) and (4) satisfy these conditions, but the functions of Eqs. (2) and (3) do not.

One new generator function that satisfies the above conditions [9] is the linear function

$$g_0(v) = av + b, \tag{6}$$

where $a > 0$ and $b \geqslant 0$. If $a = 1$ and $p = 3$, then the hidden layer function reduces to the inverse multiquadric function of Eq. (4). In this paper, we will concentrate on hidden layer functions of the form of Eq. (5) with special emphasis on the inverse multiquadric function of Eq. (4).

3. Derivative-based optimization of radial basis functions

The response of an RBF of the form of Fig. 1, where the hidden layer functions $g(\cdot)$ have the form of Eq. (5), can be written as follows:

$$\hat{y} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1c} \\ w_{20} & w_{21} & \cdots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n0} & w_{n1} & \cdots & w_{nc} \end{bmatrix} \begin{bmatrix} 1 \\ g(||x - v_1||^2) \\ \vdots \\ g(||x - v_c||^2) \end{bmatrix}. \tag{7}$$

We will use the following notation as shorthand for the weight matrix on the right-hand side of Eq. (7)

$$\begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1c} \\ w_{20} & w_{21} & \cdots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n0} & w_{n1} & \cdots & w_{nc} \end{bmatrix} = \begin{bmatrix} w_1^{\mathrm{T}} \\ w_2^{\mathrm{T}} \\ \vdots \\ w_n^{\mathrm{T}} \end{bmatrix} = W. \tag{8}$$

If we are given a training set of M desired input–output responses $\{x_i, y_i\}$ $(i = 1, \ldots, M)$, then we can augment M equations of the form of Eq. (7) as follows:

$$[\hat{y}_1 \quad \cdots \quad \hat{y}_M] = W \begin{bmatrix} 1 & \cdots & 1 \\ g(||x_1 - v_1||^2) & \cdots & g(||x_M - v_1||^2) \\ \vdots & \vdots & \vdots \\ g(||x_1 - v_c||^2) & \cdots & g(||x_M - v_c||^2) \end{bmatrix}. \tag{9}$$

We will introduce the following notation for the matrix on the right-hand side of Eq. (9).

$$h_{0k} = 1 \quad (k = 1, \ldots, M), \tag{10}$$

$$h_{jk} = g(||x_k - v_j||^2) \quad (k = 1, \ldots, M), \quad (j = 1, \ldots, c) \tag{11}$$

in which case we can write the matrix on the right-hand side of Eq. (9) as

$$\begin{bmatrix} h_{01} & \cdots & h_{0M} \\ h_{11} & \cdots & h_{1M} \\ \vdots & \vdots & \vdots \\ h_{c1} & \cdots & h_{cM} \end{bmatrix} = [h_1 \quad \cdots \quad h_M] = H. \tag{12}$$

In this case we can rewrite Eq. (9) as

$$\hat{Y} = WH. \tag{13}$$

Now, if we want to use gradient descent to minimize the training error, we can define the error function

$$E = \tfrac{1}{2}||Y - \hat{Y}||_{\mathrm{F}}^2, \tag{14}$$

where Y is the matrix of target (desired) values for the RBF output, and $||\cdot||_{\mathrm{F}}^2$ is the square of the Froebinius norm of a matrix, which is equal to the sum of the squares of the elements of the matrix. It has been shown [9] that in this case

$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^{M} (\hat{y}_{ik} - y_{ik}) h_k \quad (i = 1, \ldots, n), \tag{15}$$

$$\frac{\partial E}{\partial v_j} = \sum_{k=1}^{M} 2g'(||x_k - v_j||^2)(x_k - v_j) \sum_{i=1}^{n} (y_{ik} - \hat{y}_{ik}) w_{ij} \quad (j = 1, \ldots, c), \tag{16}$$

where $\hat{y}_{ik}$ is the element in the ith row and kth column of the $\hat{Y}$ matrix of Eq. (13), and y_{ik} is the corresponding element in the Y matrix. Now we can optimize the RBF with respect to the rows of the weight matrix W and the prototype locations v_j by iteratively computing the above partials and performing the following updates:

$$\begin{aligned} w_i &= w_i - \eta \frac{\partial E}{\partial w_i} \quad (i = 1, \ldots, n), \\ v_j &= v_j - \eta \frac{\partial E}{\partial v_j} \quad (j = 1, \ldots, c), \end{aligned} \tag{17}$$

where η is the step size of the gradient descent method. This optimization stops when w_i and v_j reach local minima.

4. Radial basis function optimization using the Kalman filter

Alternatively, we can use Kalman filtering to minimize the training error. Derivations of the extended Kalman filter are widely available in the literature [1,8]. In this section we briefly outline the algorithm and show how it can be applied to RBF network optimization. Consider a nonlinear finite dimensional discrete time system of the form

$$\begin{aligned} \theta_{k+1} &= f(\theta_k) + \omega_k, \\ y_k &= h(\theta_k) + v_k, \end{aligned} \tag{18}$$

where the vector θ_k is the state of the system at time k, ω_k is the process noise, y_k is the observation vector, v_k is the observation noise, and $f(\cdot)$ and $h(\cdot)$ are nonlinear vector functions of the state. Assume that the initial state θ_0 and the noise sequences $\{v_k\}$ and $\{\omega_k\}$ are Gaussian and independent from each other with

$$\mathscr{E}(\theta_0) = \bar{\theta}_0, \tag{19}$$

$$\mathscr{E}[(\theta_0 - \bar{\theta}_0)(\theta_0 - \bar{\theta}_0)^{\mathrm{T}}] = P_0, \tag{20}$$

$$\mathscr{E}(\omega_k) = 0, \tag{21}$$

$$\mathscr{E}(\omega_k w_l^{\mathrm{T}}) = Q\delta_{kl}, \tag{22}$$

$$\mathscr{E}(v_k) = 0, \tag{23}$$

$$\mathscr{E}(v_k v_l^{\mathrm{T}}) = R\delta_{kl}, \tag{24}$$

where $\mathscr{E}(\cdot)$ is the expectation operator and δ_{kl} is the Kronecker delta. The problem addressed by the extended Kalman filter is to find an estimate $\hat{\theta}_{n+1}$ of θ_{k+1} given y_j $(j = 0, \ldots, k)$.

If the nonlinearities in Eq. (18) are sufficiently smooth, we can expand them around the state estimate $\hat{\theta}_k$ using Taylor series to obtain

$$f(\theta_k) = f(\hat{\theta}_k) + F_k \times (\theta_k - \hat{\theta}_k) + \text{higher order terms},$$
$$h(\theta_k) = h(\hat{\theta}_k) + H_k^{\mathrm{T}} \times (\theta_k - \hat{\theta}_k) + \text{higher order terms}, \tag{25}$$

where we have introduced the notation

$$F_k = \left.\frac{\partial f(\theta)}{\partial \theta}\right|_{\theta=\hat{\theta}_k},$$
$$H_k^{\mathrm{T}} = \left.\frac{\partial h(\theta)}{\partial \theta}\right|_{\theta=\hat{\theta}_k}. \tag{26}$$

Neglecting the higher-order terms in Eq. (25), the system in Eq. (18) can be approximated as

$$\theta_{k+1} = F_k\theta_k + \omega_k + \phi_k,$$
$$y_k = H_k^{\mathrm{T}}\theta_k + v_k + \varphi_k, \tag{27}$$

where ϕ_k and φ_k are defined as

$$\phi_k = f(\hat{\theta}_k) - F_k\hat{\theta}_k,$$
$$\varphi_k = h(\hat{\theta}_k) - H_k^{\mathrm{T}}\hat{\theta}_k. \tag{28}$$

It can be shown that the desired estimate $\hat{\theta}_n$ can be obtained by the recursion

$$\hat{\theta}_k = f(\hat{\theta}_{k-1}) + K_k[y_k - h(\hat{\theta}_{k-1})],$$
$$K_k = P_kH_k(R + H_k^{\mathrm{T}}P_kH_k)^{-1},$$
$$P_{k+1} = F_k(P_k - K_kH_k^{\mathrm{T}}P_k)F_k^{\mathrm{T}} + Q. \tag{29}$$

K_k is known as the Kalman gain. In the case of a linear system, it can be shown that P_k is the covariance matrix of the state estimation error, and the state estimate $\hat{\theta}_{k+1}$ is optimal in the sense that it approaches the conditional mean $\mathscr{E}[\theta_{k+1}|(y_0, y_1, \ldots, y_k)]$ for large k. For nonlinear systems the filter is not optimal and the estimates are only approximately conditional means.

Inspired by the successful use of the Kalman filter for training non-RBF neural networks [14], we can apply a similar technique to the training of RBF networks. In general, we can view the optimization of the weight matrix W and the prototypes v_j as a weighted least-squares minimization problem, where the error vector is the difference between the RBF outputs and the target values for those outputs. Consider the RBF network of Fig. 1 with m inputs, c prototypes, and n outputs. We use y to denote the target vector for the RBF outputs, and $h(\hat{\theta}_k)$ to denote

the actual outputs at the kth iteration of the optimization algorithm.

$$y = [y_{11} \quad \cdots \quad y_{1M} \quad \cdots \quad y_{n1} \quad \cdots \quad y_{nM}]^{\mathrm{T}},$$
$$h(\hat{\theta}_k) = [\hat{y}_{11} \quad \cdots \quad \hat{y}_{1M} \quad \cdots \quad \hat{y}_{n1} \quad \cdots \quad \hat{y}_{nM}]_k^{\mathrm{T}}. \tag{30}$$

Note that the y and $\hat{y}$ vectors each consist of nM elements, where n is the dimension of the RBF output and M is the number of training samples. In order to cast the optimization problem in a form suitable for Kalman filtering, we let the elements of the weight matrix W and the elements of the prototypes v_j constitute the state of a nonlinear system, and we let the output of the RBF network constitute the output of the nonlinear system to which the Kalman filter is applied. The state of the nonlinear system can then be represented as

$$\theta = [w_1^{\mathrm{T}} \quad \cdots \quad w_n^{\mathrm{T}} \quad v_1^{\mathrm{T}} \quad \cdots \quad v_c^{\mathrm{T}}]^{\mathrm{T}}. \tag{31}$$

The vector θ thus consists of all $(n(c+1)+mc)$ of the RBF parameters arranged in a linear array. The nonlinear system model to which the Kalman filter can be applied is

$$\theta_{k+1} = \theta_k,$$
$$y_k = h(\theta_k), \tag{32}$$

where $h(\theta_k)$ is the RBF network's nonlinear mapping between its parameters and its output. In order to execute a stable Kalman filter algorithm, we need to add some artificial process noise and measurement noise to the system model [14]. So we rewrite Eq. (32) as

$$\theta_{k+1} = \theta_k + \omega_k,$$
$$y_k = h(\theta_k) + v_k, \tag{33}$$

where ω_k and v_k are artifically added noise processes. Now we can apply the Kalman recursion of Eq. (29). $f(\cdot)$ is the identity mapping and y_k is the target output of the RBF network. (Note that although y_k is written as a function of the Kalman iteration number k, it is actually a constant.) $h(\hat{\theta}_k)$ is the actual output of the RBF network given the RBF parameters at the kth iteration of the Kalman recursion. H_k is the partial derivative of the RBF output with respect to the RBF network parameters at the kth iteration of the Kalman recursion. F_k is the identity matrix (again, a constant even though it is written as a function of k). The Q and R matrices are tuning parameters which can be considered as the covariance matrices of the artificial noise processes ω_k and v_k, respectively. It is shown in Appendix A that the partial derivative of the RBF output with respect to the RBF network parameters is given by

$$H_k = \begin{bmatrix} H_w \\ H_v \end{bmatrix}, \tag{34}$$

where H_w and H_v are given by

$$H_w = \begin{bmatrix} H & 0 & \cdots & 0 \\ 0 & H & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & H \end{bmatrix}, \tag{35}$$

$$H_v = \begin{bmatrix} -w_{11}g'_{11}2(x_1 - v_1) & \cdots & -w_{11}g'_{m1}2(x_m - v_1) & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ -w_{1c}g'_{1c}2(x_1 - v_c) & \cdots & -w_{1c}g'_{mc}2(x_m - v_c) & \cdots \\ -w_{n1}g'_{11}2(x_1 - v_1) & \cdots & -w_{n1}g'_{m1}2(x_m - v_1) & \\ \vdots & \vdots & \vdots & \\ -w_{nc}g'_{1c}2(x_1 - v_c) & \cdots & -w_{nc}g'_{mc}2(x_m - v_c) & \end{bmatrix}, \tag{36}$$

where H (with no subscript) is the $(c+1) \times M$ matrix given in Eq. (12), w_{ij} is the element in the ith row and jth column of the W weight matrix (see Eq. (8)), $g'_{ij} = g'(||x_i - v_j||^2)$ (where $g(\cdot)$ is the activation function at the hidden layer), x_i is the ith input vector, and v_j is the jth prototype vector. H_w in Eq. (35) is an $n(c+1) \times nM$ matrix, H_v in Eq. (36) is an $mc \times nM$ matrix, and H_k in Eq. (34) is an $[n(c+1)+mc] \times nM$ matrix. Appendix B gives a derivation of $g'_{ij}(\cdot)$ for the case when $g(\cdot)$ is generated by the linear generator function. Now that we have the H_k matrix, we can execute the recursion of Eq. (29), thus using the extended Kalman filter in order to determine the weight matrix W and the prototypes v_j.

5. Decoupling the Kalman filter

The Kalman filter described in the previous section can be decoupled in order to save computational effort. This is similar to the decoupling that has been performed for Kalman filter training of recurrent neural networks [14]. For a large RBF network, the computational expense of the Kalman filter could be burdensome. In fact, the computational expense of the Kalman filter is on the order of AB^2, where A is the dimension of the output of the dynamic system and B is the number of parameters. In our case, there are nM outputs and $n(c+1)+mc$ parameters, where n is the dimension of the RBF output, M is the number of training samples, c is the number of prototypes, and m is the dimension of the RBF input. Therefore, the computational expense of the Kalman filter is on the order of $nM[n(c+1)+mc]^2$.

The Kalman filter parameter vector can be decoupled by assuming that certain parameter groups interact with each other only at a second-order level. For instance, it can be seen from Appendix A that the H_k matrix contains a lot of zeros, showing that the interaction between various parameter groups can be neglected. In particular, the H_k matrix consists of $n+1$ decoupled blocks. These $n+1$ blocks correspond to the n sets of w weights that affect the n output components, and

the set of prototypes. This is intuitive because, for example, the $c+1$ weights that impinge on the first component of the output are completely independent of the $c+1$ weights that impinge on the second component of the output (see Fig. 1). Let us use the notation that θ_k^i refers to the ith group (out of $n+1$ total) of parameters estimated at time step k. Then we have

$$\begin{aligned}
\theta_k^1 &= w_1 \\
&\vdots \\
\theta_k^n &= w_n \\
\theta_k^{n+1} &= [v_1^{\mathrm{T}} \quad \cdots \quad v_c^{\mathrm{T}}]^{\mathrm{T}}.
\end{aligned} \tag{37}$$

We will use the notation that H_k^i refers to the submatrix of H_k corresponding to the ith group of parameters.

$$\begin{aligned}
H_k^1 &= H \\
&\vdots \\
H_k^n &= H \\
H_k^{n+1} &= H_v,
\end{aligned} \tag{38}$$

where H (with no subscript) is the $(c+1)\times M$ matrix given in Eq. (12), and H_v is given in Eq. (36). We will use the notation that y_k^i refers to the elements of the target output of the RBF network that are affected by the ith group of parameters.

$$\begin{aligned}
y_k^1 &= [y_{11} \quad \cdots \quad y_{1M}]^{\mathrm{T}} \\
&\vdots \\
y_k^n &= [y_{n1} \quad \cdots \quad y_{nM}]^{\mathrm{T}} \\
y_k^{n+1} &= [y_{11} \quad \cdots \quad y_{1M} \quad \cdots \quad y_{n1} \quad \cdots \quad y_{nM}]^{\mathrm{T}}.
\end{aligned} \tag{39}$$

Similarly, we use the notation that $h^i(\hat{\theta}_{k-1})$ refers to the elements of the actual output of the RBF network that are affected by the ith group of parameters.

$$\begin{aligned}
h^1(\hat{\theta}_{k-1}) &= [\hat{y}_{11} \quad \cdots \quad \hat{y}_{1M}]_k^{\mathrm{T}} \\
&\vdots \\
h^n(\hat{\theta}_{k-1}) &= [\hat{y}_{n1} \quad \cdots \quad \hat{y}_{nM}]_k^{\mathrm{T}} \\
h^{n+1}(\hat{\theta}_{k-1}) &= [\hat{y}_{11} \quad \cdots \quad \hat{y}_{1M} \quad \cdots \quad \hat{y}_{n1} \quad \cdots \quad \hat{y}_{nM}]_k^{\mathrm{T}}.
\end{aligned} \tag{40}$$

The decoupled Kalman recursion for the ith parameter group, modified from Eq. (29), is then given by

$$\begin{aligned}
\hat{\theta}_k^i &= f(\hat{\theta}_{k-1}^i) + K_k^i[y_k^i - h^i(\hat{\theta}_{k-1}^i)],\\
K_k^i &= P_k^i H_k^i (R^i + (H_k^i)^{\mathrm{T}} P_k^i H_k^i)^{-1},\\
P_{k+1}^i &= F_k(P_k^i - K_k^i (H_k^i)^{\mathrm{T}} P_k^i) F_k^{\mathrm{T}} + Q^i.
\end{aligned} \tag{41}$$

As before, $f(\cdot)$ is the identity mapping and F_k is the identity matrix. The above recursion executes $n+1$ times. The first n times, the recursion consists of M outputs and $(c+1)$ parameters. The last time, the recursion consists of nM outputs and mc parameters. So the computational expense of the Kalman filter has been reduced to the order of $nM[(c+1)^2 + (mc)^2]$. The ratio of the computational expense of the standard Kalman filter to the decoupled Kalman filter can be computed as

$$\frac{\text{Standard KF Expense}}{\text{Decoupled KF Expense}} = \frac{n^2(c+1)^2 + m^2c^2 + n(c+1)mc}{(c+1)^2 + m^2c^2}. \tag{42}$$

The computational savings will be most significant for large problems, i.e., problems where n (the dimension of the output) is large, m (the dimension of the input) is large, or c (the number of prototypes) is large. Note that this complexity analysis applies only to the Kalman recursion and does not include the computational expense required for the calculation of the partial derivatives (see Appendix A). Also note that for both the standard and decoupled Kalman filters, the computational expense increases linearly with M (the number of training samples).

6. Simulation results

In this section we describe and illustrate the use of Kalman filter training for the parameters of an RBF network. We tested the algorithms of the previous sections on the classical Iris classification problem [2]. Each Iris exemplar has four features and is classified into one of three categories. The Iris data contains 50 exemplars from each category for a total of 150 patterns. We randomly divided the patterns into training and test sets, each containing 25 exemplars from each category. The input data were normalized by replacing each feature value x by $\bar{x} = (x - \mu_x)/\sigma_x$, where μ_x and σ_x denote the sample mean and standard deviation of this feature over the entire data set. The networks were trained to respond with the target value $y_{ik} = 1$, and $y_{jk} = 0 \quad \forall j \neq i$, when presented with an input vector x_k from the ith category. The reformulated RBF networks were trained using the hidden layer function of Eq. (5) with the linear generator function of Eq. (6) with $a = 1$ and $b = 1$. The exponential parameter p in Eq. (5) was varied between 2 and 4 in the simulation results presented in this section. The training algorithms were initialized with prototype vectors randomly selected from the input data, and with the weight matrix W set to 0.

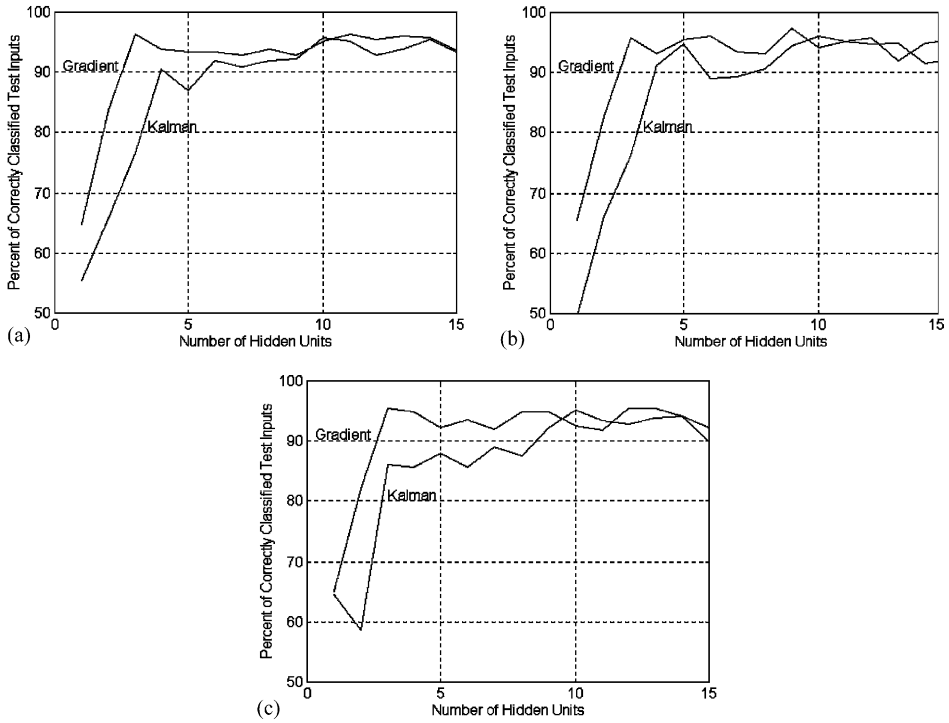

Fig. 2. Average RBF performance on the Iris test data with linear generator functions $g_0(v) = v + 1$ and $g(v) = [g_0(v)]^{1/(1-p)}$: (a) $p = 2$; (b) $p = 3$; (c) $p = 4$.

After some experimentation, it was concluded that gradient descent worked best with $\eta = 0.01$ (see Eq. (17)). The gradient descent optimization algorithm was terminated when the error function of Eq. (14) decreased by less than 0.1%.

The Kalman filter parameters of Eq. (29) were initialized with $P_0 = 40I$, $Q = 40I$, and $R = 40I$, where I is the identity matrix of appropriate dimensions. The Kalman filter recursion was terminated when the error function of Eq. (14) decreased by less than 0.1%.

The decoupled Kalman filter parameters of Eq. (41) were initialized with $P_0^i = 40I$, $Q^i = 40I$, and $R^i = 40I$, where $i = 1, \ldots, (n + 1)$, and where I is the identity matrix of appropriate dimensions.

The performance of each of the training methods was explored by averaging its performance over five trials, where each trial consisted of a random selection of training and test data. Fig. 2 depicts the performance of the RBF network on the test data when the network was trained with gradient descent and when it was trained with the Kalman filter. The number of hidden units in the RBF network was varied between 1 and 15. It can be seen from the figure that, in general, gradient descent training resulted in a better performing network than Kalman filter training. But as the number of hidden units increases, the performances of

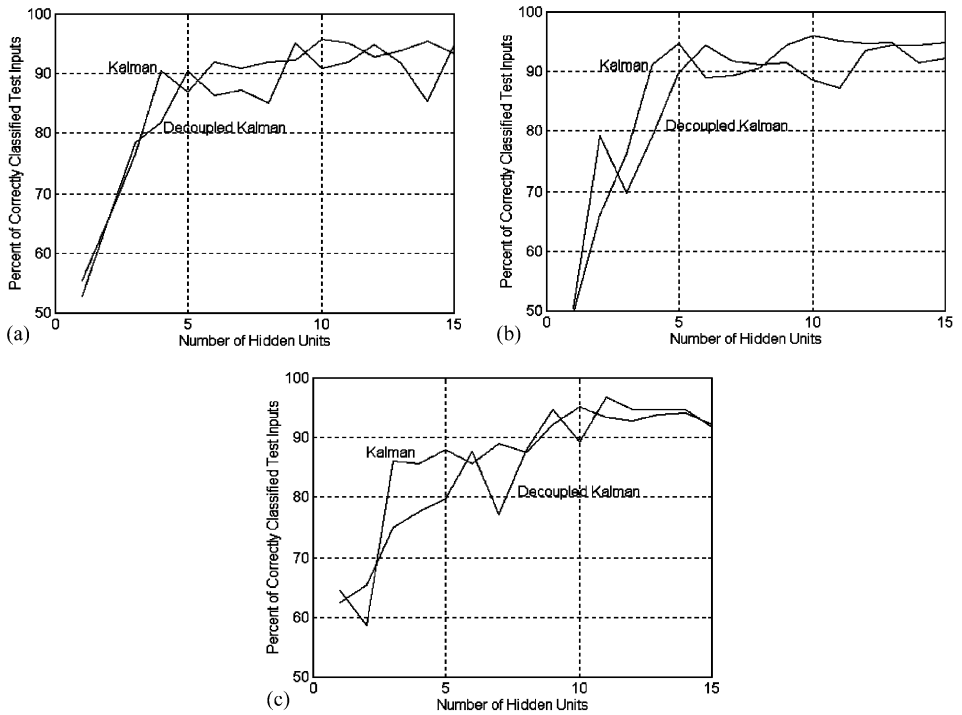

Fig. 3. Average RBF performance on the Iris test data with linear generator functions $g_0(v) = v + 1$ and $g(v) = [g_0(v)]^{1/(1-p)}$: (a) $p = 2$; (b) $p = 3$; (c) $p = 4$.

the two training algorithms are very similar. The RBF network reaches a peak performance of about 95%. This is slightly worse than the results presented in [9]. The discrepancies could be due to differences in the training termination criteria, or differences in other implementation details.

Fig. 3 shows the performance of the RBF network when it was trained with the Kalman filter and when it was trained with the decoupled Kalman filter. The number of hidden units in the RBF network was again varied between 1 and 15. It can be seen from the figure that Kalman filter training resulted in marginally better performance than decoupled Kalman filter training. The decoupling strategy discussed in Section 5 degraded the performance of the RBF network only slightly.

Fig. 4 shows the number of iterations required for convergence for gradient descent training, Kalman filter training, and decoupled Kalman filter training. It can be seen that gradient descent training requires more iterations for convergence than Kalman filter training (both standard and decoupled) by a full order of magnitude. This indicates the computational superiority of Kalman filters over gradient descent. Note that the convergence criterion is identical for all training methods—that is, training was terminated when the error function of Eq. (14) decreased by less than 0.1%.

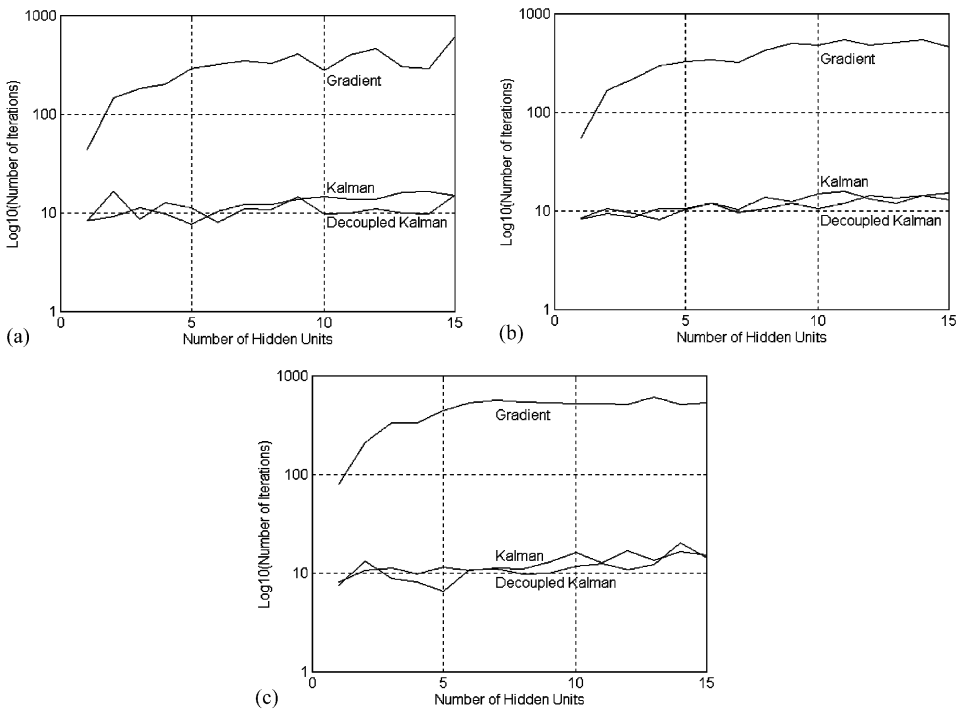

Fig. 4. Average number of iterations required for learning convergence with linear generator functions $g_0(v) = v + 1$ and $g(v) = [g_0(v)]^{1/(1-p)}$: (a) $p = 2$; (b) $p = 3$; (c) $p = 4$.

Fig. 5 compares the CPU time required for convergence for the three training methods. (The CPU time is measured in seconds on a Pentium III 550 MHz CPU running MATLAB.) With just one or two hidden units, the CPU time is comparable for each of the three methods. But as the number of hidden units increases above one or two, the CPU time required by gradient descent reaches a full order of magnitude greater than that required by the Kalman filters. (Recall from Figs. 2 and 3 that the number of hidden units needs to be more than one or two in order to achieve good test performance.) It may be somewhat surprising from Fig. 5 that the decoupled Kalman filter requires about the same amount of CPU time as the standard Kalman filter. However, as the analysis in Section 5 shows, the computational savings achieved by the decoupled Kalman filter will be most significant for problems where the dimension of the output is large, the dimension of the input is large, or the number of prototypes is large. In our Iris classification problem, the dimension of the input is 4, the dimension of the output is 3, and the number of prototypes varies between 1 and 15. So we simply have too small of a problem to see much computational savings. Also, recall that the analysis in Section 5 applies only to the Kalman recursion and does not include the computational

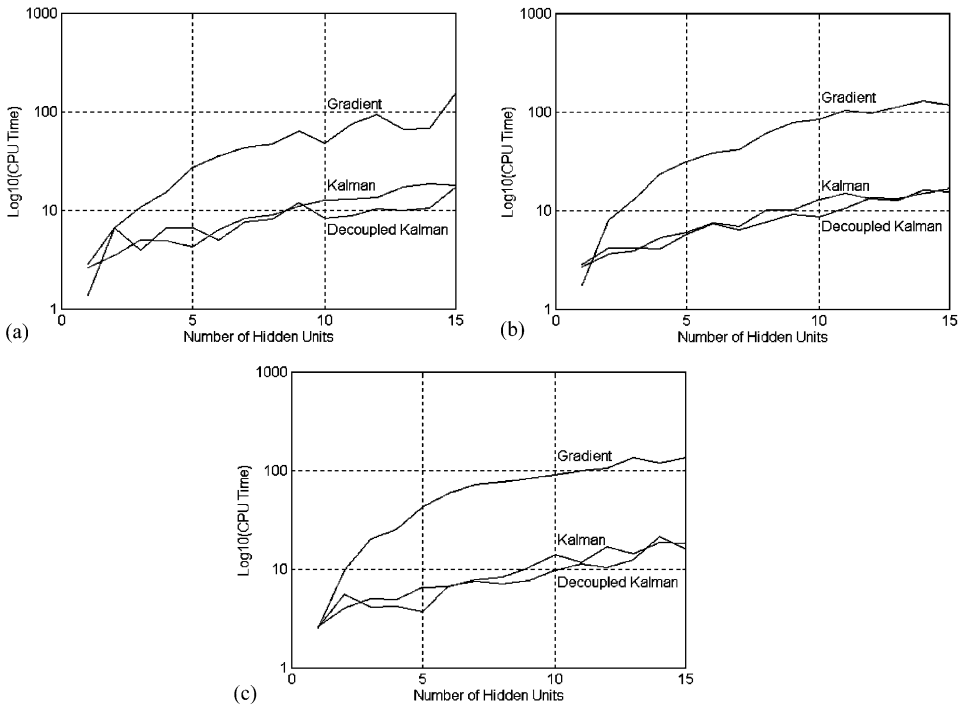

Fig. 5. Average CPU time required for learning convergence with linear generator functions $g_0(v) = v + 1$ and $g(v) = [g_0(v)]^{1/(1-p)}$: (a) $p = 2$; (b) $p = 3$; (c) $p = 4$.

expense required for the calculation of the partial derivatives. This further dilutes the computational savings of the decoupled Kalman filter.

7. Conclusion

The success of a neural network architecture depends heavily on the availability of effective learning algorithms. The theoretical strength of the Kalman filter has led to its use in hundreds of technologies, and this paper demonstrates that RBF network training is yet another fruitful application of Kalman filtering. The experiments reported in this paper verify that Kalman filter training provides about the same performance as gradient descent training, but with only a fraction of the computational effort. In addition, it has been shown that the decoupled Kalman filter provides performance on par with the standard Kalman filter while further decreasing the computational effort for large problems.

Further research could focus on the application of Kalman filter training to RBF networks with alternative forms of the generator function. In addition, the convergence of the Kalman filter could be further improved by more intelligently initializing the training process. (Recall that in this paper the prototypes were all

initialized to random input vectors and the weight matrix was initialized to zero.) Other work could focus on applying these techniques to large problems to obtain experimental verification of the computational savings of decoupled Kalman filter training. Additional efforts could be directed towards effective determination of the Kalman filter tuning parameters (P_0, Q, and R). There are several objectives which could be addressed by further research along these lines. This paper represents just the initial steps in a direction that appears to be promising.

The MATLAB m-files that were used to generate the results presented in this paper can be downloaded from the world-wide web at `http://academic.csuohio.edu/simond/rbfkalman/`. The Iris data files and the m-files for gradient descent training, Kalman filter training, and decoupled Kalman filter training are available for download and experimentation.

Appendix A

Use the notation $\hat{Y}_j$ to denote the output of the RBF network given the jth training input and the current network parameters. Use the notation $\hat{Y}$ to denote the concatenation of all M training outputs.

$$\hat{Y} = [\hat{Y}_1^{\mathrm{T}} \quad \cdots \quad \hat{Y}_M^{\mathrm{T}}]^{\mathrm{T}}. \tag{A.1}$$

From Section 3 we can see that

$$\hat{Y} = \begin{bmatrix} \sum_{i=0}^{c} w_{1i} h_{i1} \\ \vdots \\ \sum_{i=0}^{c} w_{1i} h_{iM} \\ \vdots \\ \sum_{i=0}^{c} w_{ni} h_{i1} \\ \vdots \\ \sum_{i=0}^{c} w_{ni} h_{iM} \end{bmatrix}. \tag{A.2}$$

So if we define w as the first $n(c+1)$ elements of the θ vector in Eq. (31)

$$w = [w_1^{\mathrm{T}} \quad \cdots \quad w_n^{\mathrm{T}}]^{\mathrm{T}} \tag{A.3}$$

then the partial derivative of $\hat{Y}$ with respect to the weights w_{ij} in the Kalman filter parameter vector of Eq. (31) can be derived as

$$\frac{\partial \hat{Y}}{\partial w} = \begin{bmatrix} \partial w_{10} \\ \vdots \\ \partial w_{1c} \\ \vdots \\ \partial w_{n0} \\ \vdots \\ \partial w_{nc} \end{bmatrix} \begin{bmatrix} \sum_{i=0}^{c} w_{1i} h_{i1} & \cdots & \sum_{i=0}^{c} w_{1i} h_{iM} & \cdots \\ & & & \\ \sum_{i=0}^{c} w_{ni} h_{i1} & \cdots & \sum_{i=0}^{c} w_{ni} h_{iM} & \end{bmatrix} \tag{A.4}$$

$$= \begin{bmatrix} H & 0 & \cdots & 0 \\ 0 & H & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & H \end{bmatrix}, \tag{A.5}$$

where H is given in Eq. (12). From Section 3 we can also see that

$$\hat{Y} = \begin{bmatrix} w_{10} + \sum_{j=1}^{c} w_{1j} g(||x_1 - v_j||^2) & \cdots & w_{10} + \sum_{j=1}^{c} w_{1j} g(||x_M - v_j||^2) & \cdots \\ w_{n0} + \sum_{j=1}^{c} w_{nj} g(||x_1 - v_j||^2) & \cdots & w_{n0} + \sum_{j=1}^{c} w_{nj} g(||x_M - v_j||^2) & \end{bmatrix}. \tag{A.6}$$

So if we define v as the last mc elements of the θ vector in Eq. (31)

$$v = \begin{bmatrix} v_1^{\mathrm{T}} & \cdots & v_c^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \tag{A.7}$$

then the partial derivative of $\hat{Y}$ with respect to the prototypes v_j in the Kalman filter parameter vector of Eq. (31) can be derived as

$$\frac{\partial \hat{Y}}{\partial v} = \begin{bmatrix} \partial v_1 \\ \vdots \\ \partial v_c \end{bmatrix} \begin{bmatrix} \sum_{j=1}^{c} w_{1j} g_{1j} & \cdots & \sum_{j=1}^{c} w_{1j} g_{Mj} & \cdots \\ & & & \\ \sum_{j=1}^{c} w_{nj} g_{1j} & \cdots & \sum_{j=1}^{c} w_{nj} g_{Mj} & \end{bmatrix} \tag{A.8}$$

$$= \begin{bmatrix} -w_{11}g'_{11}2(x_1 - v_1) & \cdots & -w_{11}g'_{M1}2(x_M - v_1) & \cdots \\ \vdots & & \vdots & \vdots \\ -w_{1c}g'_{1c}2(x_1 - v_c) & \cdots & -w_{1c}g'_{Mc}2(x_M - v_c) & \cdots \end{bmatrix}$$

$$\begin{bmatrix} -w_{n1}g'_{11}2(x_1 - v_1) & \cdots & -w_{n1}g'_{M1}2(x_M - v_1) \\ \vdots & & \vdots \\ -w_{nc}g'_{1c}2(x_1 - v_c) & \cdots & -w_{nc}g'_{Mc}2(x_M - v_c) \end{bmatrix}, \quad \text{(A.9)}$$

where $g_{ij} = g(||x_i - v_j||^2)$, $g'_{ij} = g'(||x_i - v_j||^2)$ (where $g(\cdot)$ is the generator function at the hidden layer), x_i is the ith input vector, and v_j is the jth prototype vector. Now, combining Eqs. (A.5) and (A.9), we obtain Eqs. (34)–(36) for the partial derivative of the RBF network output with respect to the Kalman filter parameter vector.

Appendix B

Consider the hidden layer function

$$g(v) = [g_0(v)]^{1/(1-p)} \quad \text{(B.1)}$$

with the linear generator function

$$g_0(v) = v + \gamma^2, \quad \text{(B.2)}$$

where γ is some constant. Note that $g'_0(v) = 1$. Therefore

$$g'(v) = \frac{1}{1-p} g_0(v)^{p/(1-p)} g'_0(v) \quad \text{(B.3)}$$

$$= \frac{1}{1-p} (v + \gamma^2)^{p/(1-p)} \quad \text{(B.4)}$$

$$= \frac{1}{1-p} g^p(v). \quad \text{(B.5)}$$

Recall from Eq. (11) that $h_{jk} = g(||x_k - v_j||^2)$, $(k = 1, \ldots, M)$, $(j = 1, \ldots, c)$. Combining these equations, we can write

$$g'(||x_k - v_j||^2) = \frac{1}{1-p} g^p(||x_k - v_j||^2) \quad \text{(B.6)}$$

$$= \frac{1}{1-p} [g_0(||x_k - v_j||^2)]^{p/(1-p)} \quad \text{(B.7)}$$

$$= \frac{1}{1-p} h_{jk}^p. \quad \text{(B.8)}$$

This provides the required expression for the derivative of the hidden layer function that is needed to calculate the H_v matrix of Eq. (36) and thus perform the Kalman recursion of Eq. (29).

References

[1] B. Anderson, J. Moore, Optimal Filtering, Prentice-Hall, Englewood Cliffs, NJ, 1979.
[2] J. Bezdek, J. Keller, R. Krishnapuram, L. Kuncheva, H. Pal, Will the real Iris data please stand up? IEEE Trans. Fuzzy Systems 7 (1999) 368–369.
[3] M. Birgmeier, A fully Kalman-trained radial basis function network for nonlinear speech modeling, IEEE International Conference on Neural Networks, Perth, Western Australia, 1995, pp. 259–264.
[4] D. Broomhead, D. Lowe, Multivariable functional interpolation and adaptive networks, Complex Systems 2 (1988) 321–355.
[5] S. Chen, C. Cowan, P. Grant, Orthogonal least squares learning algorithm for radial basis function networks, IEEE Trans. Neural Networks 2 (1991) 302–309.
[6] S. Chen, Y. Wu, B. Luk, Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks, IEEE Trans. Neural Networks 10 (1999) 1239–1243.
[7] R. Duro, J. Reyes, Discrete-time backpropagation for training synaptic delay-based artificial neural networks, IEEE Trans. Neural Networks 10 (1999) 779–789.
[8] A. Gelb, Applied Optimal Estimation, MIT Press, Cambridge, MA, 1974.
[9] N. Karayiannis, Reformulated radial basis neural networks trained by gradient descent, IEEE Trans. Neural Networks 3 (1999) 657–671.
[10] S. Kirkpatrick, Cl. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[11] J. Moody, C. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 1 (1989) 289–303.
[12] K. Narendra, M. Thathachar, Learning Automata—An Introduction, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[13] D. Obradovic, On-line training of recurrent neural networks with continuous topology adaptation, IEEE Trans. Neural Networks 7 (1996) 222–228.
[14] G. Puskorius, L. Feldkamp, Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks, IEEE Trans. Neural Networks 5 (1994) 279–297.
[15] V.D. Sánchez, A. (Ed.), Special Issue on RBF Networks, Part I, Neurocomputing 19 (1998).
[16] V.D. Sánchez, A. (Ed.), Special Issue on RBF Networks, Part II, Neurocomputing 20 (1998).
[17] S. Shah, F. Palmieri, M. Datum, Optimal filtering algorithms for fast learning in feedforward neural networks, Neural Networks 5 (1992) 779–787.
[18] D. Simon, Distributed fault tolerance in optimal interpolative nets, IEEE Trans. on Neural Networks, in print.
[19] S. Sin, R. DeFigueiredo, Efficient learning procedures for optimal interpolative nets, Neural Networks 6 (1993) 99–113.
[20] J. Sum, C. Leung, G. Young, W. Kan, On the Kalman filtering method in neural network training and pruning, IEEE Trans. Neural Networks 10 (1999) 161–166.
[21] Y. Zhang, X. Li, A fast U-D factorization-based learning algorithm with applications to nonlinear system modeling and identification, IEEE Trans. Neural Networks 10 (1999) 930–938.

Dan Simon received his BS, MS, and Ph.D. degrees from Arizona State University, the University of Washington, and Syracuse University, all in Electrical Engineering. He has 14 years of industrial experience in the aerospace, automotive, biomedical, process control, and software engineering fields. He has been an Assistant Professor at Cleveland State University since 1999, where he teaches control theory, computational intelligence, and embedded systems. He has over 30 publications in refered journals and conference proceedings.