Proceedings of the 35th Conference on Decision and Control Kobe, Japan • December 1996

WM18 2:10

Hybrid Training of RBF Networks with Application to Nonlinear Systems Identification[1]

Youmin Zhang[2] X. Rong Li
Department of Electrical Engineering
University of New Orleans, New Orleans, LA 70148
ymzee@uno.edu *xrlee@uno.edu*

Abstract

The feedforward neural networks, including multiple layer perceptron (MLP) and radial basis function network (RBFN), are the most widely used networks due to their rapid training, generality, and simplicity. For RBFN, a traditional training algorithm consists of two stages: first learning in the hidden layer, which is typically performed using an unsupervised method such as a clustering algorithm; this is followed by a supervised learning in the output layer, such as recursive least squares (RLS) algorithms. Such a training algorithm has several drawbacks, including improper selection of RBF centers and over-size problem of the network in the first stage and ill-condition in the second. This paper proposes a new clustering algorithm based on constructing an augmented vector consisting of both input and output, and for training the RBFN using a U-D factorization based RLS algorithm which is superior to the standard RLS algorithm in convergence rate, numerical stability and accuracy of the training. The performance between RBFN and MLP with a sigmoidal function is also compared via simulation examples.

1. Introduction

The feedforward neural networks (FNNs) have attracted a great deal of interest due to its rapid training, generality, and simplicity. It has been satisfactorily used in many engineering fields, such as data classification, nonlinear system modeling, identification and control [11], [8], [2]. A radial basis function (RBF) model, which can be viewed as a three-layer FNN with a linear output mapping, is equally capable of modeling a wide class of nonlinear functions with arbitrary accuracy [7]. It is linear in network parameters provided that a separate method used for selecting a suitable set of basis function centers. Once the centers are chosen, the RBFN can be uniquely determined by a linear estimation technique. In contrast, the learning schemes of a conventional FNN consist of a nonlinear least squares optimization process, and convergence to the correct minimum error solution can not be guaranteed. Haykin [7] has given a meaningful comparison between the RBFN and perceptron. However, the performance of an RBFN depends critically on the choice of the basis function centers. In practice, a popular approach for selecting RBFN centers is the so-called "K-means-clustering" algorithm, which minimizes the sum of distance squares from each data point to the nearest center. This algorithm has been applied and modified by several researchers [10], [5]. Its main drawback is that the selection of the RBF centers depends only on the distribution of input training vectors without taking into account how output varies with the variation of the input. Thus, where there exhibits a severe variation in the output while the corresponding input is relatively smooth, the performance of the RBFN deteriorates.

To solve this problem we present a new algorithm which constructs an augmented vector consisting of both input and output for selecting the RBF centers. In the second phase, an RLS algorithm is usually used for supervised learning. But the conventional RLS algorithm may suffer from ill-condition and can be biased under poor-excitation [9], [13]. Ill-condition frequently occurs in the case of a near linear dependence caused by, for example, some centers being too close. To improve the numerical stability and convergence, several techniques, such as using singular value decomposition (SVD) technique, have been investigated for the optimization and training of the RBFN. An SVD-based optimization and recursive least squares (SVD-RLS) training algorithm for RBFN was developed recently by the authors [15]. However, this algorithm is computationally demanding due to the complexity of the singular value decomposition. The U-D factorization technique, on the other hand, provides an alternative way to overcome numerical ill-condition and other drawbacks in the training of the RBFN.

2. Feedforward Neural Networks

FNNs have been the subject of intensive research efforts in recent years because of their interesting learning and generalization properties and their applicability in a variety of classification, approximation, modeling, identification, and control problems. The commonly used networks are MLP and RBFN, which are described briefly as follows.

2.1. Radial Basis Function Network (RBFN)

An RBFN consists of one input layer, one output layer and one hidden layer. A schematic of the RBFN with L inputs and M outputs is depicted in Fig.1, which looks like a common feedforward topology. Such a network implements a mapping $f: R^L \rightarrow R^M$. The ith output of the network is computed according to the linear equation

$$y_i = f_i(\mathrm{x}) = \sum_{k=1}^{K} w_{ik}\varphi_k, \; i = 1, ..., M \qquad (1)$$

[1]Project partially supported by NNSFC (grant 69274015), ASFC (grant 94E53186) and NSF (grant ECS-9409358).

[2]Y. Zhang is on leave from Northwestern Polytechnical University.

$$\varphi_k = \exp(- \| \mathbf{x} - \mathbf{c}_k \|^2 / \sigma_k^2) \qquad (2)$$

where $\mathbf{x} \in R^L$ is the input vector; w_{ik} is the weight from the kth output of the hidden layer φ_k to the ith output; $\| \cdot \|$ denotes the Euclidean norm; k is the index of the hidden nodes, $k = 1, 2, ..., K$, and K is the number of the RBFs, which is smaller than the number of available training data points N; $\mathbf{c}_k$ is the kth center; and σ_k stands for the width of the kth Gaussian function. Usually the nonlinear mapping of the hidden layer is carried out with the Gaussian function.

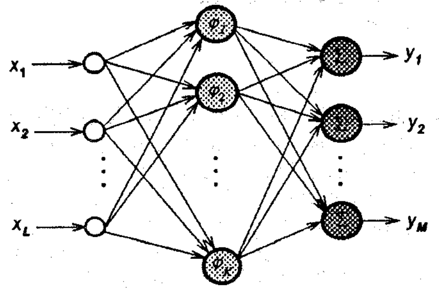

Fig. 1. Radial basis function network (RBFN).

From a design point of view, the requirement is to select suitable values for the parameters of each of the K Gaussian RBFs, namely $\mathbf{c}_k$ and $\sigma_k, k=1, ..., K$, and solve for the weights of the output layer. There are several approaches to computation of these parameters. The simplest approach for the design of an RBFN involves selecting a set of fixed RBFs for the hidden units of the networks. In particular, the locations of the centers may be chosen randomly from the training data set. The main problem with the use of fixed centers is the fact that it may require a large data set for a prescribed level of performance. One way of overcoming this limitation is to use a hybrid learning procedure [10], [4], which combines the steps: 1) self-organized learning algorithm for the selection of the centers of the radial basis function in the hidden layer; 2) supervised learning algorithm for the computation of the weights in the output. Although batch processing can be used to implement these two operations, it is particularly advantageous to take an iterative (adaptive) approach [4], [5].

Based on (1), we intend to use the SVD technique [15] to accomplish the optimization and the U-D factorization technique [1], [13] for the training of the RBFN so that we can resolve the problem of numerical ill-condition as well as avoid the over-size problem of the network. Performance comparison between the RBFN and the MLP by using the proposed U-D based RLS training algorithm is also given.

2.2. MLP and Its Comparison with RBFN

The BP algorithm has become the standard algorithm for training MLP networks. Although successfully used in many cases, the BP algorithm suffers from a number of shortcomings, such as slow convergence rate and local minimal problem. Another serious problem for the application of MLP to system modeling, identification and control, however, is that the sigmoidal activation function usually implemented has a very limited output range, that is [0, 1] or [-1, 1]. This requires a priori knowledge of extrema of all variables to be estimated and does not guarantee the same accuracy for each variable. The problem can be solved simply by using a linearly weighted sum of outputs from the hidden layer to the output layer, rather than the sigmoidal activation function in the output [3], [12]. Fig. 2 shows a typical structure of a single hidden layer MLP.

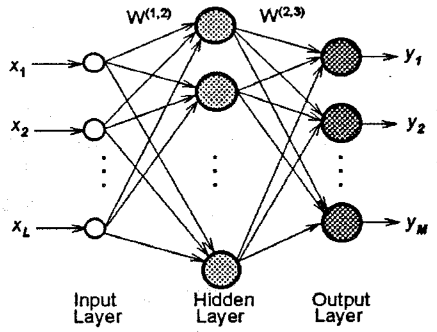

Fig. 2. Multiple layer perceptron (MLP) network.

RBFNs and MLPs are both examples of nonlinear multilayer FNNs that are universal approximators [6], [10], [7]. It is therefore not surprising to find that there always exists an RBFN capable of accurately mimicking a specified MLP, and vice versa. However, these two networks differ from each other in several important respects as outlined in [7]: 1) In general, an RBFN has a single hidden layer, whereas an MLP may have one or more hidden layers; 2) Typically, the hidden layer of an RBFN is nonlinear, whereas the output layer is linear. On th other hand, the hidden and output layers of an MLP used as a classifier are usually all nonlinear; when the MLP is used to solve nonlinear regression, nonlinear system modeling and identification problems, however, a linear layer for the output is usually the preferred choice [3]. In view of this, a linear layer for the output is used in this paper for MLP; 3) The argument of the activation function of each hidden node in an RBFN computes the Euclidean norm (distance) between the input vector and the center of the node. On the other hand, the activation function of each hidden node in an MLP computes the inner product of the input vector and the synaptic weight vector of the node; 4) RBFN construct local approximations to nonlinear input-output mapping due to using exponentially decaying localized nonlinearities (e.g., Gaussian functions), with the result that these networks are capable of fast learning. In many cases, however, in order to represent a mapping to some desired degree of smoothness, the number of RBFs required to span the input space adequately may have to be very large. On the other hand, MLPs construct global approximations to a nonlinear input-output mapping; 5) When continuous learning is required as in the training of a time-varying environment, the use of nested nonlinearities in an MLP makes it difficult to evolve the network in a dynamic fashion. That is, if we want to include a new example, the whole network must be retrained all over again. In contrast, the structure of an RBFN permits its weights of the output layer to be updated without having to recompute them.

3. Clustering Algorithm with Augmented Input

As described in previous section, a major drawback of the traditional clustering algorithm of RBFN is that the selection of the RBF centers depends only on the distribution of input training vectors without taking into account how output varies with the variation of the input.

Thus, where there exhibits a severe variation in the output while the corresponding input is relatively smooth, the performance of the RBFN deteriorates. To solve this problem we present a new algorithm which constructs an augmented vector consisting of both input and output for selecting the RBF centers [15]. The algorithm is given briefly as follows:

Let $\{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in R^L, \mathbf{y}_i \in R^M, i = 1, ..., N\}$ be the ith training set, where $\mathbf{x}_i$ and $\mathbf{y}_i$ are input and output training vectors, respectively. Now construct the following vector

$$\bar{\mathbf{x}}_i = \begin{bmatrix} \alpha \mathbf{x}_i \\ \beta \mathbf{y}_i \end{bmatrix}, \; i = 1, ..., N \tag{3}$$

We call $\bar{\mathbf{x}}_i$ an augmented vector, in which $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$ are scaling factors of input and output respectively. This augmented vector is utilized in the K-means-clustering to select RBF centers. One circle of the modified K-means-clustering algorithm may be given as follows:

(a) *Initialization:* Choose K random value for the initial centers $\bar{\mathbf{u}}_j$ (0) from $\{\bar{\mathbf{x}}_i\}$; the only restriction here is that the $\bar{\mathbf{u}}_j$ (0) be distinct for $j = 1, ..., K$.

(b) *Similarity matching:* Find the best-matching (nearest) center $\bar{\mathbf{u}}_j(k)$ at iteration k to $\bar{\mathbf{x}}_i$ and assign $\bar{\mathbf{x}}_j$ to it. For instance, if

$$D_i(k) = \| \bar{\mathbf{x}}_i - \bar{\mathbf{u}}_j(k) \| = \min_l \{ \| \bar{\mathbf{x}}_i - \bar{\mathbf{u}}_l(k) \| \} \tag{4}$$

then $\bar{\mathbf{x}}_i \in S_j(k)$, $\mathbf{x}_i \in R_j(k)$, $\mathbf{y}_i \in Q_j(k)$, where $S_j(k)$, $R_j(k)$ and $Q_j(k)$ are the jth cluster of the augmented vector, input and output, respectively.

(c) *Updating* $\bar{\mathbf{u}}_j(k)$*:* $\bar{\mathbf{u}}_j(k+1) := \frac{1}{K_j} \sum_{\bar{\mathbf{x}} \in S_j(k)} \bar{\mathbf{x}}$, $j = 1, ..., K$, where K_j is the number of vectors in $S_j(k)$.

(d) *Let:* $\mathbf{x}_i := \mathbf{u}_i = \frac{1}{L_j} \sum_{\mathbf{x} \in R_j(k)} \mathbf{x}$; $\mathbf{y}_i := \frac{1}{M_j} \sum_{\mathbf{y} \in Q_j(k)} \mathbf{y}$, $j=1, ..., K$, where L_j and M_j are numbers of vectors in R_j and Q_j respectively. The outcome $\{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, ..., K\}$ is the refined training set.

The above clustering process is stopped when a specified tolerant accuracy is met; otherwise go back to step (b). At last, $\{\mathbf{x}_i \mid i = 1, ..., K\}$ is the ultimate RBF centers.

4. Training: RLS with forgetting factor

An RBFN or MLP is trained by adjusting its weights according to an ongoing stream of input-output observations $\{x_k(t), y_k(t)$: $t=1, ..., N; k=1, ..., p\}$ where p denotes the number of training samples sequences. The objective is to obtain a set of weights such that the neural networks will predict future outputs accurately. This training process is equivalent to the process of parameter estimation. Thus, training RBFN or MLP can be considered as a nonlinear identification problem where the weight values are unknown and need be identified for the given set of input-output vectors. An approach is to concatenate all the network parameters into a vector θ and to map the input to the output assuming this parameter vector is correct. The concatenated parameter vector $\theta \in R^{n_\theta}$ can be defined as:

$$\theta = [W_{1,1}^{(1,2)}, ..., W_{n_2,n_1}^{(1,2)}, b_1, ..., b_{n_2}, W_{1,1}^{(2,3)}, ..., W_{n_3,n_2}^{(2,3)}]^T \tag{5}$$

where

$$n_\theta = (n_1 + 1)n_2 + n_2 n_3; \; n_1 = L, n_2 = K, n_3 = M$$

$W_{j,i}^{(1,2)}$ is the weight from the ith neuron of input layer to jth neuron of hidden layer, and b_i is the threshold of the ith hidden neuron. Hence, the training problem can be posed as a parameter identification and the dynamic equation for the neural network can be cast into the following form for the RLS scheme:

$$\mathbf{y}(t) = \mathbf{f}[\theta(t), \mathbf{x}(t)] + \varepsilon(t) \tag{6}$$

where $\theta(t)$ is the parameter of the FNN at time step t; $\mathbf{y}(t)$ is the observed output vector of the networks; $\varepsilon(t)$ is white Gaussian noises with zero mean.

The parameter identification problem is then defined as the determination of $\hat{\theta}$ which minimizes the sum of squared prediction errors of all prior observations embedded in the function:

$$J(p) = \frac{1}{2} \sum_{t=1}^{p} \{\mathbf{y}(t) - \mathbf{f}\left[\hat{\theta}(t), \mathbf{x}(t)\right]\}^2 \lambda^{p-t}(t) \tag{7}$$

where p denotes the number of training samples. Assume that the nonlinearity of (6) is sufficiently smooth, by a first-order Taylor series expansion of $\mathbf{f}[\theta(t), \mathbf{x}(t)]$ about an estimated parameter $\hat{\theta}(t)$, the estimated output $\hat{\mathbf{y}}(t)$ can be linearized as

$$\hat{\mathbf{y}}(t) = \mathbf{f}\left[\hat{\theta}(t), \mathbf{x}(t)\right] + \left.\frac{\partial \mathbf{f}[\theta(t), \mathbf{x}(t)]}{\partial \theta}\right|_{\theta=\hat{\theta}(t)} [\theta - \hat{\theta}] + \text{HOT} \tag{8}$$

where higher order terms HOT are neglected.

Then the recursive least-squares (RLS) algorithm with forgetting factor can be applied directly:

$$\begin{aligned} \hat{\theta}(t) &= \hat{\theta}(t-1) + \mathcal{K}(t)\{\mathbf{y}(t) - \mathbf{f}[\hat{\theta}(t-1), \mathbf{x}(t-1)]\} & (9) \\ \mathcal{K}(t) &= P(t-1)H^T(t)[H(t)P(t-1)H^T(t) + \lambda(t)I]^{-1} & (10) \\ P(t) &= [P(t-1) - \mathcal{K}(t)H(t)P(t-1)]/\lambda(t) & (11) \end{aligned}$$

where $P(t)$ is the error covariance matrix; $\lambda(t) = \lambda_0 \lambda(t-1) + (1 - \lambda_0)$ is a time-varying forgetting factor, λ_0 and $\lambda(0)$ are design parameters; $H(t)$ is the Jacobian matrix

$$H(t) = \left.\frac{\partial \mathbf{f}[\theta(t), \mathbf{x}(t)]}{\partial \theta(t)}\right|_{\theta(t)=\hat{\theta}(t)}$$

Note that the factor $H(t)P(t-1)$ occurs three times in the computation of $\mathcal{K}(t)$ and $P(t)$. This feature can be utilized in the implementation of the above RLS to reduce computational requirement.

5. U-D Factorization-Based RLS Algorithm

In the above RLS, the computation of the covariance matrix $P(t)$ plays an important role. The finite word length effect (in particular, the truncation errors) of a computer may lead the algorithm to the loss of symmetry and positive definiteness of $P(t)$, or even divergence. Several covariance matrix factorization methods have been developed to improve the computation of $P(t)$. The U-D factorization method is a most popular one [1], [9]. It decomposes $P(t)$ into $P(t) = U(t)D(t)U^T(t)$, where $U(t)$ and $D(t)$ are a unit-upper triangular matrix and a diagonal matrix, respectively. Then, $U(t)$ and $D(t)$, instead of $P(t)$, are updated at every time step. This decomposition guarantees the positive-definiteness and symmetry of $P(t)$, and thus high estimation accuracy and robustness can be attained [1], [9], [13].

The Bierman's U-D factorization algorithm is a sequential process approach, which is especially suitable for single output measurement originally. In order to develop a U-D factorization-based RLS learning algorithm, we first give results for single output case, then extend it to multiple output case.

5.1. Multiple Input Single Output (MISO) Case

In order to carry out the U-D decomposition for $P(t)$, we can first substitute (10) into (11) to get

$$P(t) = \frac{1}{\lambda(t)} \left\{ P(t^-) - \frac{P(t^-)H^T(t)H(t)P(t^-)}{H(t)P(t^-)H^T(t) + \lambda(t)} \right\} \quad (12)$$

where $t-1$ is replaced by t^- to save space. Suppose $P(t^-)$ has been U-D decomposed at time $t-1$:

$$P(t^-) = U(t^-)D(t^-)U^T(t^-)$$

Eq. (12) then becomes

$$\begin{aligned} P(t) &= \frac{1}{\lambda(t)}\{U(t^-)D(t^-)U^T(t^-) & (13)\\ &\quad - \frac{U(t^-)D(t^-)U^T(t^-)H^T(t)H(t)U(t^-)D(t^-)U^T(t^-)}{H(t)U(t^-)D(t^-)U^T(t^-)H^T(t) + \lambda(t)}\} \\ &= \frac{1}{\lambda(t)}U(t^-)\left[D(t^-) - \frac{\mathbf{g}\,\mathbf{g}^T}{\beta}\right]U^T(t^-) & (14) \end{aligned}$$

where

$$\mathbf{e} = U^T(t^-)H^T(t) \quad (15)$$

$$\mathbf{g} = U(t^-)\mathbf{e} \quad (16)$$

$$\beta = \mathbf{e}^T D(t^-)\mathbf{e} + \lambda(t) \quad (17)$$

and $H(t)$, $\mathbf{e}$ and $\mathbf{g}$ are n_θ-vectors for an FNN with single output; and β is a scalar.

Note that, since the product of two unit upper triangular matrices is still a unit upper triangular matrix, to obtain U-D factors for $P(t)$ it is sufficient to obtain U-D factors for the bracketed matrix in (14). In fact, if $\bar{U}(t^-)$ and $\bar{D}(t^-)$ are U-D factors for the matrix $[D(t^-) - \frac{\mathbf{g}\,\mathbf{g}^T}{\beta}]$, then the U-D factors for $P(t) = U(t)D(t)U^T(t)$ are

$$U(t) = U(t^-)\bar{U}(t^-) \quad (18)$$

$$D(t) = \bar{D}(t^-) \quad (19)$$

Let e_i and $g_i, i = 1, ..., n_\theta$ be the elements of $\mathbf{e}$ and $\mathbf{g}$, respectively. Further, let $U_{ij}(t)$ and $D_{ij}(t)$ represent the (i,j)th element of $U(t)$ and $D(t)$, respectively. Then, the RLS algorithm (9)-(11) can be implemented in U-D factorization form as follows:

Step 1. Compute $\mathbf{e} = [e_1, ..., e_{n_\theta}]^T := U^T(t^-)H^T(t)$, $\mathbf{g} = [g_1, ..., g_{n_\theta}]^T := D(t^-)\mathbf{e}$, $\alpha_0 := \lambda(t)$

Step 2. For $j = 1, ..., n_\theta$, compute

$$\left\{ \begin{array}{l} \alpha_j := \alpha_{j-1} + e_j g_j \\ D_{jj}(t) := \alpha_{j-1}D_{jj}(t^-)/\alpha_j\lambda(t) \\ \nu_j := g_j \\ \mu_j := -e_j/\alpha_{j-1} \end{array} \right. \quad (20)$$

For $i = 1, ..., j-1$, compute:

$$\left\{ \begin{array}{l} U_{ij}(t) := U_{ij}(t^-) + \nu_i\mu_j \\ \nu_i := \nu_i + U_{ij}(t^-)\nu_j \end{array} \right. \quad (21)$$

Step 3.

$$\mathcal{K}(t) = [\nu_1, ..., \nu_{n_\theta}]^T/\alpha_{n_\theta} \quad (22)$$

Step 4.

$$\hat{\theta}(t) = \hat{\theta}(t^-) + \mathcal{K}(t)\{y(t) - f[\hat{\theta}(t^-), \mathbf{x}(t^-)]\} \quad (23)$$

5.2. Multiple Input Multiple Output (MIMO)

For MIMO systems, one approach, mentioned by Bierman [1] and Ljung [9], is to transform the problem into a sequence of scalar problems. Based on this idea, a U-D factorization RLS for multiple output can be obtained by processing the M-dimensional output with M sequential applications of the above single output U-D factorization based RLS.

For multiple output networks, the observation equation (6) for the desired observation vector $\mathbf{y}(t)$ can be expressed in term of its elements:

$$y_k(t) = f_k[\theta(t), \mathbf{x}(t)] + \varepsilon_k(t), \quad k = 1, ..., M \quad (24)$$

Initially, set

$$E = [e_{jk}] := U^T(t^-)H^T(t), \; G = [g_{jk}] := D(t^-)E$$

Step 1. For $k = 1, ..., K$, $\alpha_0 := \lambda(t)$, do steps 2-3

Step 2. For $j = 1, ..., n_\theta$, compute

$$\left\{ \begin{array}{l} \alpha_j := \alpha_{j-1} + e_{jk}g_{jk} \\ D_{jj}(t) := \alpha_{j-1}D_{jj}(t^-)/\alpha_j\lambda(t) \\ \nu_j := g_{jk} \\ \mu_j := -e_{jk}/\alpha_{j-1} \end{array} \right. \quad (25)$$

For $i = 1, ..., j-1$,compute:

$$\left\{ \begin{array}{l} U_{ij}(t) := U_{ij}(t^-) + \nu_i\mu_j \\ \nu_i := \nu_i + U_{ij}(t^-)\nu_j \end{array} \right. \quad (26)$$

Step 3. Compute gain

$$\mathcal{K}_k(t) = [\nu_1, ..., \nu_{n_\theta}]^T/\alpha_{n_\theta} \quad (27)$$

Step 4. Compute

$$\hat{\theta}(t) = \hat{\theta}(t^-) + \mathcal{K}(t)\{\mathbf{y}(t) - \mathbf{f}[\hat{\theta}(t^-), \mathbf{x}(t^-)]\} \quad (28)$$

where $\mathcal{K} = [\mathcal{K}_1 \cdots \mathcal{K}_M]$ is an $n_\theta \times M$ dimensional gain matrix; $U_{ij}(t^-)$ represents the (i,j)th element of the matrix $U(t^-)$ and similarly for $D_{ij}(t^-)$.

In order to smooth out weight changes and improve the convergence and accuracy of the training algorithm, an "variable forgetting factor" technique can be used[14].

6. Applications to Systems Identification

In this section, simulation results from three problems of nonlinear plant identification are presented. Performance comparison between RBFN and MLP in term of learning accuracy, convergence rate, the generalization ability of the networks, of the proposed UD-RLS, RLS and the BP algorithm is given and analyzed.

Example 1. In this example a training set $\{(x, y)\}$ which consisted of 200 data points was used to test the proposed hybrid training algorithm. The input and output relationship of the system satisfies the following nonlinear mapping.

$$y(k) = 2.0\frac{x(k)}{1.0 + x^2(k)} \quad (29)$$

where $y(t) \in [-1, 1]$ when $x(t) \in [-10, 10]$. $x(t)$ are 200 input data points generated uniformly over [-10, 10]. The UD-RLS, RLS and BP algorithms were employed to train an MLP with an architecture of 1-10-1 to compare with the RBFN. For RBFN, the proposed K-means clustering algorithm with augmented input combining with the UD-RLS training algorithm, is used.

Effect of Input Augmentation: It is obvious that if scaling factors $\alpha=1$ and $\beta=0$, then the augmented vector $\tilde{\mathbf{x}}_i$ depends only on $\mathbf{x}_i$, and therefore, the chosen RBF centers are uniquely determined by the input training data. In this case, the algorithm degenerates into the traditional K-means-clustering method. Similarly, when $\alpha=0$ and $\beta=1$ the RBF centers depends only on output. The desired and the approximate output curves of both cases are shown in Fig. 3. Another case with $\alpha=0.5$ and $\beta=1$ is also added in the figure which indicates that the simultaneous use of the input and output data in the K-means-clustering makes the performance of the RBFN fairly perfect, while in the cases of ($\alpha=1$, $\beta=0$) and ($\alpha=0$, $\beta=1$) the results are not desirable. Better results can be obtained by adjusting the parameters α and β.

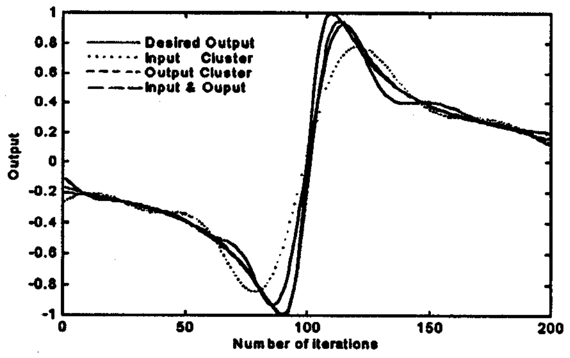

Fig. 3. Effect of input augmentation.

Comparison of RBFN and MLP with different hidden nodes: Table 1 shows the mean and standard deviation (s.d.) of output errors of MLP and RBFN. Fig. 4 illustrates the RMSE and output error curves from the two types of FNNs with different numbers of hidden nodes. It is clear that RBFN has much faster convergence rate than MLP due to the clustering, at the price of bad learning accuracy. The output error of the MLP with 10 hidden nodes has a much higher learning accuracy even than that of the RBFN with 150 hidden nodes. When the number of hidden nodes of RBFN is as many as the that of the input data points, i.e., 200 hidden nodes, very high accuracy were obtained, at the cost of a huge computational requirement.

Table 1. The output error of MLP and RBFN.

# of nodes	Mean	s.d.
MLP-10	$2.9629e^{-6}$	$9.2024e^{-6}$
RBF-10	$1.0959e^{-3}$	$1.5853e^{-1}$
RBF-15	$-3.7522e^{-4}$	$5.0982e^{-2}$
RBF-50	$-2.5554e^{-5}$	$5.1965e^{-3}$
RBF-100	$3.0292e^{-5}$	$1.7659e^{-3}$
RBF-150	$-1.0194e^{-5}$	$7.0486e^{-4}$
RBF-200	$2.1460e^{-18}$	$4.6081e^{-17}$

Example 2. This example compares the learning accuracy of the UD-RLS algorithms while training an RBFN and MLP to model the following nonlinear dynamic equation [4]

$$\begin{aligned} y(k) &= (0.8 - 0.5e^{-y^2(k-1)})y(k-1) - (0.3 + 0.9e^{-y^2(k-1)})y(k-2) \\ &\quad +0.1\sin(3.1415926y(k-1)) + e(k) \end{aligned} \tag{30}$$

where $e(t)$ is a Gaussian white noise sequence with zero mean and variance 0.01. Fig. 5 gives the output error of RBFN and MLP using the UD-RLS algorithm. It is shown that a better accuracy can be obtained by MLP at the price of slow convergence and more training time.

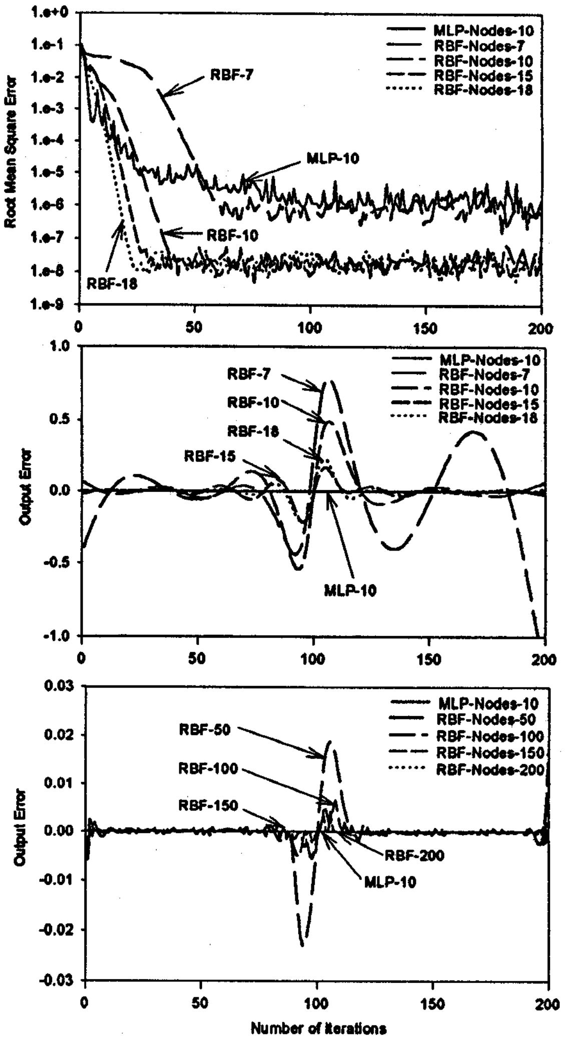

Fig. 4. Performance comparison of RBFN with MLP.

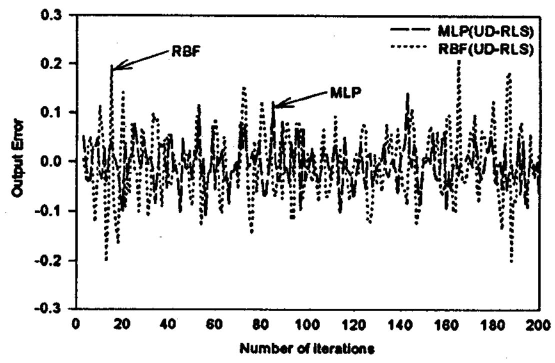

Fig. 5. Comparison of RBFN with MLP.

Example 3. Consider the identification of a MIMO systems given by Narendra [11].

$$\begin{bmatrix} y_1(t+1) \\ y_2(t+1) \end{bmatrix} = \begin{bmatrix} \frac{y_1(t)}{1+y_2^2(t)} \\ \frac{y_1(t)y_2(t)}{1+y_2^2(t)} \end{bmatrix} + \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} \tag{31}$$

An MLP and an RBFN of 4 inputs, 20 hidden nodes and 2 outputs (4-20-2) were used to model and identify the above MIMO system. The prediction response of the three algorithms for a vector input $[\sin(2\pi t/25), \cos(2\pi t/25)]^T$ after the 100th learning iteration given 500 training data are shown in Fig. 6. Table 2 shows the mean and standard deviation (s.d.) of learning and generalization output errors of two types of FNNs. The results of BP were computed by the fast adaptive learning rate BP algorithm, which is much faster than standard BP, given in MATLAB's Neural Network Toolbox 2.0. It is obvious that the MLP has much better prediction accuracy than RBFN with the same number of hidden nodes. Due to the increase in state dimension and poor numerical stability, RLS often exhibits severe oscillation and sometimes even diverges. As such, the double precision has to be used to implement it. The learning results of adaptive learning rate BP algorithms for MLP are also shown in Fig. 6 for comparison with the UD-RLS, with the same zero initial condition as for UD-RLS and RLS. The very slow convergence rate and poor generalization result were obtained by adaptive learning rate BP. If the initial weights were given randomly, a slightly better results may be obtained. This fact also indicates that the BP algorithm is sensitive to initial weights, in contrast to the UD-RLS and RLS.

Table 2. Results comparison with three algorithms.

Results	y_1		y_2	
Algorithms	Mean	s.d.	Mean	s.d.
MLP	$3.5256e^{-6}$	$7.3684e^{-4}$	$2.3689e^{-6}$	$1.2886e^{-3}$
RBFN	$5.0341e^{-3}$	$1.5104e^{-1}$	$2.2051e^{-3}$	$1.4097e^{-1}$
MLP(BP)	$2.0058e^{-2}$	$1.3416e^{-1}$	$1.3930e^{-3}$	$6.7967e^{-1}$
MLP	$-3.7520e^{-3}$	$1.8192e^{-1}$	$-1.0500e^{-2}$	$1.8923e^{-1}$
RBFN	$-7.5709e^{-1}$	$1.5197e^{-1}$	$1.1484e^{-1}$	$5.9144e^{-1}$
MLP(BP)	$-1.9512e^{-1}$	$8.7109e^{-1}$	$1.2051e^{-1}$	$1.0581e^{0}$

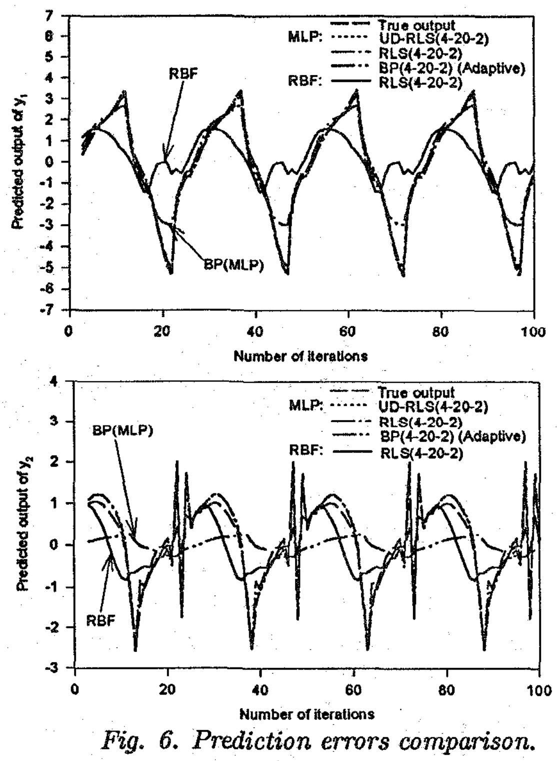

Fig. 6. Prediction errors comparison.

7. Conclusions

In this paper, a hybrid training algorithm for RBFN is proposed, which includes a new K-means clustering algorithm with augmented input for selecting radial basis function centers and a U-D factorization based recursive least squares (UD-RLS) algorithm for weights estimation. The proposed training algorithm is then used for the identification of nonlinear systems. It is also used for the training of MLP. The performance comparison between RBFN and MLP shows that the proposed hybrid algorithm is much faster in training and has smaller mean square errors than the conventional RLS algorithm. The RBFN has much faster convergence and need less training time than MLP, which makes RBFN more suitable to the real-time application of nonlinear system modeling and identification. The MLP can, however, yield much better training accuracy, for the same number of hidden nodes, than RBFN at the cost of more training time.

References

[1] G. J. Bierman. *Factorization Methods for Discrete Sequential Estimation.* Academic Press, New York, 1977.

[2] S. Chen and S. A. Billings. Neural networks for nonlinear dynamical system modeling and identification. *Int. J. Control*, 56(2):319–346, 1992.

[3] S. Chen, S. A. Billings, and P. M. Grant. Non-linear system identification using neural networks. *Int. J. Control*, 51(6):1191–1214, 1990.

[4] S. Chen, S. A. Billings, and P. M. Grant. Recisive hybrid algorithm for non-linear system identification using radial basis function networks. *Int. J. Control*, 55(5):1051–1070, 1992.

[5] C. Chinrungrueng and C. H. Sequin. Optimal adaptive k-means algorithm with dynamic adjustment of learning rate. *IEEE Trans. Neural Networks*, 6(1):157–169, Jan. 1995.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. of Contr., Signals and Syst.*, 2(2):303–314, 1989.

[7] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice-Hall, Englewood Cliffs, NJ, 1994.

[8] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop. Neural networks for control systems — a survey. *Automatica*, 28(6):1083–1112, 1992.

[9] L. Ljung and T. Soderstrom. *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, MA, 1983.

[10] J. Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(1):281–294, 1989.

[11] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks*, 1(1):4–27, Mar. 1990.

[12] Y. M. Zhang, G. Z. Dai, and H. C. Zhang. An extended Kalman filtering learning algorithm for feedforward neural networks. *Information and Control*, 23(1):113–118, 1994.

[13] Y. M. Zhang, Q. G. Li, G. Z. Dai, and H. C. Zhang. A new algorithm for model structure determination and parameter estimation of time-varying systems. *J. of Northwestern Polytechnical Unversity*, 13(1):36–40, 1995.

[14] Y. M. Zhang and X. R. Li. A fast and robust recursive prediction error learing algorithm for feedforward neural networks. In *Accepted by 35th IEEE Conf. on Decision and Control*, Kobe, Japan, Dec. 1996.

[15] Y. M. Zhang, X. R. Li, Z. W. Zhu, and H. C. Zhang. A new clustering and training method for radial basis function networks. In *Proc. of IEEE Int. Conf. on Neural Networks*, pages 311–316, Washington, DC, June 1996.