Computers and Mathematics with Applications 57 (2009) 1651–1658

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Solving a system of nonlinear integral equations by an RBF network☆

A. Golbabai [b], M. Mammadov [a,*], S. Seifollahi [b]

[a] School of Information Technology and Mathematical Science, Ballarat University, Ballarat VIC 3350, Australia

[b] Department of Mathematics, Iran University of Science and Technology, Narmak, Tehran 16844, Iran

ARTICLE INFO

Article history:
Received 16 April 2008
Received in revised form 16 March 2009
Accepted 18 March 2009

Keywords:
RBF network
Newton's method
Gradient method
System of nonlinear integral equations
Continuous optimization

ABSTRACT

In this paper, a novel learning strategy for radial basis function networks (RBFN) is proposed. By adjusting the parameters of the hidden layer, including the RBF centers and widths, the weights of the output layer are adapted by local optimization methods. A new local optimization algorithm based on a combination of the gradient and Newton methods is introduced. The efficiency of some local optimization methods to update the weights of RBFN is studied in solving systems of nonlinear integral equations.

1. Introduction

The radial basis function (RBF) networks have attracted the attention of many researchers due to their simple architecture, computational efficiency, powerful generalization capability and learning schemes. These networks were first brought to widespread attention by Broomhead and Lowe [1] in 1988. Moody and Darken [2], Renals and Rohwer [3], and Girosi and Poggio [4] among others made major contributions to the theory, design, and application of RBF networks. The RBF networks have been shown to be universal approximators; that is, theoretically, any continuous function defined on a compact set can be approximated to a given accuracy by increasing the number of hidden nodes (see [4–7]).

The RBF network is a class of feed forward neural networks and in its most basic form involves three entirely different layers. The first layer is an input layer of which each node corresponds to an attribute of an input sample, and passes directly to the hidden layer without weights, i.e., the weight connection is unity. The nonlinear responses of the hidden nodes are weighted in order to calculate the final outputs of the network in the third (output) layer. The transformation from the input layer to the hidden layer is nonlinear, whereas the transformation from the hidden layer to the output layer is linear. Mathematically, given an input vector x, the output of the jth node in the output layer, that implements a sum of arbitrary basis functions defined on its inputs, can be expressed as:

$$\hat{u}_j(x) = \sum_{i=1}^{m} w_{ij}\phi_i(x), \tag{1}$$

where m is the number of hidden nodes, w_{ij} is the weight from the ith hidden node to the jth output node and $\phi_i(x)$ is an activation function which is a locally radial symmetric function. This function is characterized by its center, which is a

☆ This work was supported by the Australian Research Council Discovery Project Grant DP0556685.

* Corresponding author.
E-mail addresses: golbabai@iust.ac.ir (A. Golbabai), m.mammadov@ballarat.edu.au (M. Mammadov), seif@iust.ac.ir (S. Seifollahi).

0898-1221/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2009.03.038

vector with dimension equal to the number of inputs to the node. A popular choice, which is also used in this work, is the multiquadric function:

$$\phi_i(x) = (\|x - c_i\|^2 + a_i^2)^{1/2} \tag{2}$$

where c_i and a_i are the center and the width of the ith hidden node respectively, and $\|.\|$ is the Euclidean norm [8].

An RBF network is trained in many different ways that can be categorized into one-, two- and three-stage learning schemes. In this work, we use a two-stage type learning scheme. First the RBF parameters, including the centers and the widths, are set, and then the weights of the output layer are adapted by a local optimization method. In addition, from the variety of learning constructions for the RBF network in the literature, such as the OLS algorithm [9], the resource allocation network [10], and various implementations of adaptively growing and pruning algorithms (see [11–14]), we adopt a growing based structure algorithm.

The gradient descent training has proven to be more efficient than some other conventional methods [15]; however, it is easy plunging into local minima. There are some other methods that can be used to accelerate training speed and to avoid spurious local minima. One way, to overcome these difficulties, is the use of descent direction methods involving combination of different local methods. In recent years, there has been a growing interest in applying different combination methods for the optimization task. Among several existing combinations in the literature, the combination of steepest descent and Newton's methods seems to be more promising for unconstrained optimization problems [16]. It is shown that this method is globally convergent and at the same time has a high convergence rate. However, in numerical experiments, it is often observed that, in some iterations the gradient direction could be a better choice than Newton's direction. Keeping in mind this, we propose a modification of the algorithm presented in [16], in which the contribution of the gradient method is more considerable.

In recent years, the RBF network has emerged as an important type of methods for the numerical solution of differential equations and more recently it was used for solving linear integral equations (see [17,18]). This paper extends the results of [18] and formulates a training method for a multi-output RBF network with applications in system of nonlinear integral equations. In the implementation of this network to solve the system, a trial solution of the system is presented by the neural network of incremental architecture with a set of unknown parameters. Then, these parameters are trained by minimizing an appropriate error function composed of the problem residue. Different numerical techniques can be applied to solve it. In this study, we fix the centers and the widths of the RBFs and then determine the output weights by applying a local optimization method. We mainly adopt two local optimization algorithms. The first one is the algorithm developed in [16] that will be refereed as ShA (Shi's Algorithm). The second algorithm, developed in this paper, can be considered as a modified version of ShA and, therefore, will be refereed as MShA. These two algorithms are based on a combination of the gradient and Newton's methods that aimed to speed up the training process of the RBF network. The performance of the RBF network by using these local optimization methods is compared in solving systems of nonlinear integral equations.

The rest of the paper is structured as follows. Local optimization methods, ShA and MShA, are described in Section 2. In Section 3, a training algorithm for the RBF network is presented. The application of the RBF network to system of nonlinear integral equations is explained in Section 4. Numerical results obtained by the RBF network using two different local optimization methods and their comparison are given in Section 5. Conclusions are reported in Section 6.

2. Local optimization strategies

The most commonly used error function in the RBF network is the sum square error (SSE). For a multi-output RBF network, with n_o nodes in the output layer, the SSE minimization problem can be written as

$$\text{Minimize: } f(z) = \sum_{i=1}^{n_s} \sum_{j=1}^{n_o} e_{i,j}^2; \tag{3}$$

where $e_{i,j}$ is the jth output residual in the presence of the ith input sample, n_s is the number of input samples and z stands for the unknown network parameters to be determined.

In nonlinear case, the minimization of the error function is usually carried out using iterative methods. This is basically due to the fact that there are no analytical methods to determine the optimal values of parameters. To find a solution to this problem, among the variety of exiting methods, the descent direction method is the most commonly used technique due to its fast convergence property.

Descent direction framework

Denote $\nabla f(z)$ by $g(z)$. Given an initial point $z_1 \in R^q$ and an error tolerance $\varepsilon > 0$, each iteration $k = 1, 2, \ldots$ of a descent method contains the following steps:

- if $\|g_k\| < \varepsilon$, then stop;
- compute a descent direction d_k at z_k satisfying

$$g_k^{\mathrm{T}} d_k < 0; \tag{4}$$

- determine an appropriate step length $\alpha_k > 0$;
- set $z_{k+1} = z_k + \alpha_k d_k$, and go to the next iteration.

Depending on the choice of d_k and α_k, where d_k is a descent direction and α_k is a line search factor, different descent direction methods have been developed. There are some criterions for accepting α_k as an admissible step length such as backtracking method, Armijo, Goldestain and Wolfe line search methods. In Wolfe's case, the step length $\alpha_k > 0$ is determined by an inexact line search along the direction d_k satisfying

$$f(z_k + \alpha d_k) \leq f(z_k) + \rho_1 \alpha g_k^{\mathrm{T}} d_k, \tag{5}$$

$$g_{k+1}^{\mathrm{T}} d_k \geq \rho_2 g_k^{\mathrm{T}} d_k, \tag{6}$$

where $\rho_1 \in (0, 1/2)$ and $\rho_2 \in (\rho_1, 1)$ are fixed parameters. By considering Wolfe conditions (5) and (6), it is shown that the above algorithm is globally convergent.

Theorem 1 (*[19]*). *Let α_k in the above descent algorithm be defined by (5) and (6). Let also d_k satisfy*

$$\cos(\theta_k) \geq \delta \tag{7}$$

for some $\delta > 0$ and for all k, where θ_k is the angle between d_k and $-g_k$. If $g(z)$ exists and is uniformly continuous on the level set $\Omega = \{z : f(z) \leq f(z_1)\}$, then either $g_k = 0$ for some k, or $g_k \to 0$.

A simple method satisfying condition (7) is the gradient method in which $d_k = -g_k$ for all k. This method is globally convergent and can be used to find a local optimal solution to problem (3). Unfortunately, although the method is of global convergence property and usually works well in some early steps, as a stationary point is approached, it may descend very slowly [19].

In order to cope with the above-mentioned problem, Newton based methods with superlinear convergence property can be used. At the kth iteration, the classical Newton's direction is the solution of the following system

$$H_k d = -g_k, \tag{8}$$

where H_k is the Hessian matrix at z_k. If H is positive definite then Newton's direction is a descent direction and consequently the system has a unique solution. Even when H is positive definite, it is not guaranteed that Newton's method will be globally convergent. Consequently, although Newton's method generally converges faster than the gradient method, it depends on a starting point.

On the other hand, the application of Newton's method to the learning of neural networks is expensive for large structures. A number of techniques avoiding the direct computation of H may be used, for example quasi-Newton methods. These techniques are based generally on suitable approximations of the Hessian. Other alternative approaches are the combination methods which have been attracted extensive attention in recent years. In these approaches, in most cases the search direction is defined as a combination of different vectors. ShA, introduced in [16], is one of the most successful algorithms of this group that uses a combination of the gradient method and Newton's method. We present this algorithm below.

Algorithm ShA (*[16]*). Let δ, η, ρ_1 and ρ_2 be four parameters so that $0 < \delta < 1, 0 < \eta < 1, 0 < \rho_1 < 1/2$ and $\rho_1 < \rho_2 < 1$, then the steps of the algorithm are as follows.

0. Choose a starting point $z_1 \in R^q$, and an error tolerance $\varepsilon > 0$.
1. For $k = 1, 2, \ldots$;
1.0 If $\|g_k\| < \varepsilon$, then stop.
1.1 Compute Newton's direction d_1 at z_k, that satisfies (8), and then go to step 1.2. If d_1 is not computable due to the singularity of H_k, then compute the gradient direction d_2 at z_k, set $d_k = d_2$ and go to step 1.9.
1.2 If $k = 1$ or if $\|g_k\| \leq \|g_{k-1}\|$ then go to step 1.3, otherwise go to step 1.5.
1.3 Set $\bar{z} = z_k + d_1$.
1.4 If $f(\bar{z}) < f(z_k)$ and $\|g(\bar{z})\| \leq \eta \|g_k\|$, then set $z_{k+1} = \bar{z}$ and go to the next iteration, otherwise go to step 1.5.
1.5 Compute the gradient direction d_2 at z_k.
1.6 If $d_1^{\mathrm{T}} d_2 \geq \delta \|d_1\| . \|d_2\|$ then set $d_k = d_1$ and go to step 1.9, otherwise go to step 1.7.
1.7 If $d_1^{\mathrm{T}} d_2 < 0$, then set $d_k = d_2$ and go to step 1.9, otherwise compute $\bar{\lambda}$ such that

$$\bar{\lambda} = \min \left\{ \lambda; 0 \leq \lambda \leq 1, \frac{((1-\lambda)d_1 + \lambda d_2)^{\mathrm{T}} d_2}{\|(1-\lambda)d_1 + \lambda d_2\| . \|d_2\|} \geq \delta \right\},$$

1.8 Set $d_k = (1 - \bar{\lambda})d_1 + \bar{\lambda} d_2$.
1.9 Use the line search rules (5) and (6) to determine a step length $\alpha_k > 0$ along the direction d_k.
1.10 Set $z_{k+1} = z_k + \alpha_k d_k$ and go to the next iteration.

The above algorithm is an efficient algorithm for training and updating the unknown parameters of the RBF network due to its global convergence and superlinear convergence rate. This method is very close to Newton's method. Practical implementations show that, in some cases the gradient method can be a more suitable choice than Newton's method. For instance, when the difference of the function values, in two previous iterations, is large and the value of the gradient in the previous iteration is also large enough, the gradient method may work better than Newton's method.

In this paper, we introduce the following modification of this algorithm named MShA.

Algorithm MShA. Let δ, ρ_1 and ρ_2 be the parameters as defined in Algorithm ShA. Take any positive constants $\upsilon > 1$, $\nu \gg 1$, $\xi > 0$, $b_1 \in (0, 1)$, $b_2 \in (1, \frac{1}{\delta})$ and $b_3 > 1$ and initialize γ_0 by 1.

0. Choose a starting point $z_1 \in R^q$, and an error tolerance $\varepsilon > 0$.

1. For $k = 1, 2, \ldots$;

1.0 If $\|g_k\| < \varepsilon$, then stop.

1.1 If Newton's direction d_1 at z_k is not computable, due to the singularity of H_k, or if $k \neq 1$ and if $|g_k| > \upsilon$ and $|f_k - f_{k-1}| > \nu$, then compute the gradient direction d_2 at z_k, set $d_k = d_2$ and go to step 1.8.

1.2 Set $\bar{\delta} = \delta$ and $\gamma_k = \gamma_0$. If $k \neq 1$ and if $\|g_k - g_{k-1}\| < \xi$, adjust $\bar{\delta} \leftarrow b_1\bar{\delta}$, otherwise $\bar{\delta} \leftarrow b_2\bar{\delta}$.

1.3 Compute the gradient direction d_2 at z_k and compute Newton's direction d_1 at z_k, that satisfies (8).

1.4 If $d_1^{\mathrm{T}} d_2 \geq \bar{\delta}\|d_1\|.\|d_2\|$ then set $d_k = d_1$ and go to step 1.8, otherwise go to the next step.

1.5 If $d_1^{\mathrm{T}} d_2 < 0$, then set $d_k = d_2$ and go to step 1.9, otherwise set $\bar{\gamma} = \gamma_k$ and compute $\bar{\lambda}$ as follows:

$$\bar{\lambda} = \frac{1}{\bar{\gamma} + \|g_k\|}. \tag{9}$$

1.6 Set $d(\bar{\lambda}) = (1 - \bar{\lambda})d_2 + \bar{\lambda}d_1$.

1.7 If $d(\bar{\lambda})^{\mathrm{T}} d_2 < \bar{\delta}\|d(\bar{\lambda})\|\|d_2\|$, set $\gamma_k \leftarrow b_3\gamma_k$ and go back to step 1.5, otherwise set $d_k = d(\bar{\lambda})$ and go to the next step.

1.8 Use the line search rules (5) and (6) to determine a step length $\alpha_k > 0$ along the direction d_k.

1.9 Set $z_{k+1} = z_k + \alpha_k d_k$ and go to the next iteration.

The above algorithm for selecting d_k is based on Theorem 1. The following theorem establishes a global convergence property of the algorithm MShA.

Theorem 2. *Consider using the algorithm MShA to solve problem (3). If $g(z)$ exists and is uniformly continuous on the level set $\Omega = \{z : f(z) \leq f(z_1)\}$, then either $g_k = 0$ for some k, or $g_k \to 0$.*

Proof. Let assume that $g_k \neq 0$ for all k. Here, we show that the direction d_k obtained by the algorithm satisfies condition (7) of Theorem 1. Denote $\delta^* = b_1\delta$. Clearly, $\delta^* \in (0, 1)$. We show that condition (7) holds for δ^*; that is,

$$\cos(\theta_k) \geq \delta^* \tag{10}$$

for all k, where θ_k is the angle between d_k and $d_2 = -g_k$.

Take any k. We have one of the following cases.

Case 1. The direction d_k is obtained at step 1.1 or step 1.5. In this case $d_k = -g_k$ and, consequently, it is easy to see that $\cos(\theta_k) = 1 > \delta^*$; that is, (10) holds.

Case 2. The direction d_k is obtained at step 1.4 in the form $d_k = d_1$. In this case, $\bar{\delta} = b_1\delta$ or $\bar{\delta} = b_2\delta$ which means that $\bar{\delta} \geq b_1\delta = \delta^*$ as $b_1 < 1 < b_2$.

Thus, from step 1.4, we have

$$\cos(\theta_k) \geq \frac{d_k^{\mathrm{T}} d_2}{\|d_k\|.\|d_2\|} \geq \bar{\delta} \geq \delta^*; \tag{11}$$

that is, (10) holds.

Case 3. The direction d_k is obtained at step 1.7. According to steps 1.5–1.7, the number $\bar{\lambda}$ is chosen so that the inequality $d(\bar{\lambda})^{\mathrm{T}} d_2 \geq \bar{\delta}\|d(\bar{\lambda})\|\|d_2\|$, holds, where $\bar{\delta} = b_1\delta$ or $\bar{\delta} = b_2\delta$. The existence of such a number $\bar{\lambda}$ follows from the fact that $d(\bar{\lambda}) \to d_2$ as $\bar{\lambda} \to 0$ ($\bar{\gamma} \to \infty$). Then, similar to *Case* 2 we have (10).

In all the above cases, the obtained direction, d_k, satisfy in the assumption of Theorem 1, hence the remainder proof is similar to the proof of Theorem 1 given in [19]. □

3. The proposed training algorithm

We consider a multi-output RBF network and propose an algorithm for the training of the network based on a growing based architecture. For the sake of simplicity, in the present algorithm we define the centers and the widths of the RBFs fixed in advance. Thus, the output weights are only the unknown parameters of the network which must be determined.

The values of the widths affect significantly on the accuracy of results and the determination of these parameters is still a challenging problem. These parameters control the amount of overlapping of RBFs as well as the network generalization. Small values yield a rapidly decreasing function whereas larger values result in a more gently varying function. In the present study, we define the width of ith hidden node as follows:

$$a_i = \beta/i, \tag{12}$$

where β is a positive constant. The meaning of this definition is that when the number of hidden nodes increase, we want to be sure that the values of the widths (corresponding to the added nodes) decrease.

RBFN Learning Algorithm. The learning algorithm of the RBF network is briefly summarized below. This algorithm is a growing based algorithm in which the network is allowed to grow step by step. If we denote the set of centers by $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, the set of validation data by $X = \{x_1, \ldots, x_n\}$ and the set of training data by X_t that here is considered to be a subset of X, then the steps of the algorithm are summarized as follows:

Step 0. Initialize a tolerance $tol > 0$, take m_0 centers from $\mathcal{C}$ and m_0 constants for the widths as in (12), initialize $w_{i,j}$, $i = 1, \ldots, m_0, j = 1, \ldots, n_o$, take a sample from X, to be added to X_t, and set $k = 1$.

Step 1. Apply a local optimization method to determine the unknown parameters (weights) of the network.

Step 2. If the validation error, calculated on X, is less than tol or all of the validation data have been already incorporated in the training phase, then stop, otherwise go to the next step.

Step 3. If $k \neq 1$ and the validation error has increased, then insert a new node to the hidden layer ($m_0 = m_0 + 1$), select its center from $\mathcal{C}$, set its width as in (12) and set the new output weights by 0.

Step 4. If the number of training data exceeds an integer N, discard one of the training data and set $k = k - 1$.

Step 5. Select a new sample from X and add it to X_t, set $k = k + 1$ and go to Step 1.

We will apply two different local optimization methods in Step 1. We denote by "RBFN–ShA" the algorithm that uses the algorithm ShA and by "RBFN–MShA" the algorithm that uses the algorithm MShA described in Section 2.

According to the above instruction, the algorithm is terminated if the validation error during a given iteration is less than tol or the number of iteration exceeds n. The validation error will normally decrease during the phase of training, as well as the training error. We start with a few node in the hidden layer, then we continue the training process by incorporating more training samples and more hidden nodes and allow the network to grow. This yields that the network is reconstructed with less complexities. When the network begins to over-fit the data, the error on the validation set may start to rise. To overcome this difficulty and to have a network with less complexity, we apply a new strategy described below. It relates to the selection of new training samples and new nodes and is different from other existing approaches (see [12,17] and references therein).

Selection of new training sample: To select a new training sample, we calculate the error weights on the set $X \setminus X_t$, that is $\sum_{j=1}^{n_o} e_{i,j}^2$, $i = 1, \ldots, length(X \setminus X_t)$. Then, we find the data with the maximum error weight and add it to the training set X_t. Also for the sake of less computational cost and reducing the number of existing training data, after some iteration a new sample is replaced by one sample in the training data. More precisely, the error weights on training set X_t (i.e., $\sum_{j=1}^{n_o} e_{i,j}^2$, $i = 1, \ldots, n_s$) are calculated and the data which has the smallest error weight is discarded from X_t and a new sample from $X \setminus X_t$, which has a maximum distance from X_t, is added to X_t.

Initialization of new node: At the beginning of the learning algorithm, we define a sufficiently large set of centers distributed uniformly. They are selected one by one as a new node is inserted to the network. The center of new node is chosen so that its distance, from the nearest existing center in the network, is maximal among all the other candidate centers not used in the network. The width of new node is defined as in (12).

4. System of nonlinear integral equations

To test the performance of the RBF network with the proposed algorithm presented in Section 3, we consider the following system of nonlinear integral equations

$$u(x) - \int_0^x K(x, t, u(t))\mathrm{d}t = g(x), \tag{13}$$

where

$$u(x) = (u_1(x), \ldots, u_{n_o}(x))^{\mathrm{T}},$$
$$g(x) = (g_1(x), \ldots, g_{n_o}(x))^{\mathrm{T}},$$
$$K(x, t, u(t)) = (K_1(x, t, u(t)), \ldots, K_{n_o}(x, t, u(t)))^{\mathrm{T}}.$$

To obtain a solution to the above system, the collocation method based on RBFs can be used which assumes a discretization of the domain into a set of collocation points. Assume that the centers and the widths are fixed in advance, then we have the following system of nonlinear equations

$$\hat{u}_j(x_i) - \int_0^{x_i} K(x_i, t, \hat{u}(t))\mathrm{d}t - g(x_i) = 0, \tag{14}$$

where $\hat{u}_j$'s are approximate solutions of (13), defined in the form (1). Here, the number of variables (and the number of equations) is $m \times n_o$. Generally, there are no analytical solutions to this system, so it can be solved by iterative methods in terms of the unknown weights. Also, this system may often be very ill-conditioned and it may be impossible to solve accurately. This ill-conditioning is influenced by the number of centers as well as values of the widths. On the other hand, fixing appropriate number of RBFs, which results enough accurate solutions, is a great challenging problem. An alternative to solve (13) is use of a neural network. Here, we use the RBF network with the growing learning structure presented in Section 3. The system arisen in each iteration of the learning algorithm is transformed to a sum square error minimization problem

$$f(z) = \sum_{i=1}^{n_s} \sum_{j=1}^{n_o} e_{i,j}^2, \tag{15}$$

where the term $e_{i,j}$ is constructed as follows:

$$e_{i,j} = \hat{u}_j(x_i) - \int_0^{x_i} K(x_i, t, \hat{u}(t))\mathrm{d}t - g(x_i). \tag{16}$$

It should be noted that the optimal solutions of the neural network with zero value of cost function correspond to solutions of system (14). Also, in implementation of the learning algorithm, we remind the following points:

(1) In each iteration of the algorithm we deal with a system of nonlinear equations with low dimension, but the number of equations are not necessarily the same as the number of variables; the system is reformulated as optimization problem (15) in each iteration of the algorithm.
(2) The number of equations is equal to the number of the nodes in the output layer. More precisely, each node in the output layer state one equation in (13).
(3) The number of equations and the number of variables may vary from one iteration to another one.

5. Numerical results

For general nonlinear kernel $K(x, t, u)$, there is no simple way to evaluate analytically the integrals in (13), so a numerical integration scheme is used. In our calculations, we use the Simpson numerical integration scheme with 10 subintervals. To measure the accuracy of solutions, we use the max error given by

$$\mathcal{E}_j = \max_{1 \le i \le n_t} |\hat{u}_j(x_i) - u_j(x_i)|, \quad j = 1, \ldots, n_o \tag{17}$$

where $\hat{u}_j(x_i)$ and $u_j(x_i)$ are the values of the approximate and exact solutions at the point x_i, and n_t is the number of data for testing the network performance.

In the following results, the functions used in all of the hidden nodes are the multiquadratic functions, but a number of alternatives can also be used (see [8]). The validation data used in the training process contains 100 data distributed randomly in the domain and the performance of the network is tested using $n_t = 500$ data distributed uniformly from the domain. In step 4 of the RBFN learning algorithm, when $k > N = 10$, a new training sample is replaced by one of the existing training data. The termination parameters ε and *tol*, used in local optimization methods and in the learning algorithm, are: $\varepsilon = 10^{-8}$ and $tol = 10^{-7}$. The parameters used in Section 2 are: $\delta = 0.001$, $\eta = 0.99$, $\rho_1 = 0.001$, $\rho_2 = 0.9$, $\nu = 100$, $\upsilon = 10$, $\xi = 1$, $b_1 = 0.01$, $b_2 = 100$ and $b_3 = 1.1$. Also, the value of β in (12) is set as $\beta = 10$.

Example 1. Consider the following system of nonlinear Fredholm integral equations of the second kind with exact solutions $u_1(x) = x$ and $u_2(x) = x^2$ (see [20])

$$u_1(x) = x - \frac{5}{18} + \int_0^1 \frac{1}{3}(u_1(t) + u_2(t))\mathrm{d}t,$$
$$u_2(x) = x^2 - \frac{2}{9} + \int_0^1 \frac{1}{3}(u_1^2(t) + u_2(t))\mathrm{d}t.$$

The network begins with one node in the hidden layer and as observations are received, new hidden nodes are added one by one. Algorithm RBFN–MShA is terminated with $m = 5$ nodes. The maximum errors, in this case, are $\mathcal{E}_1 = 2.20 \times 10^{-6}$ and $\mathcal{E}_2 = 4.62 \times 10^{-6}$. Algorithm RBFN–ShA is terminated with $m = 5$ nodes with the maximum errors $\mathcal{E}_1 = 4.39 \times 10^{-6}$

Table 1
$e_j = |\hat{u}_j(x) - u_j(x)|, j = 1, 2$, in some test points x for Example 1.

x	RBFN–ShA		RBFN–MShA	
	e_1	e_2	e_1	e_2
0.0	4.03×10^{-6}	5.42×10^{-6}	2.03×10^{-6}	2.67×10^{-6}
0.1	3.69×10^{-6}	4.97×10^{-6}	1.87×10^{-6}	2.43×10^{-6}
0.2	1.79×10^{-7}	1.28×10^{-6}	8.76×10^{-8}	6.35×10^{-7}
0.3	3.21×10^{-6}	4.23×10^{-6}	1.61×10^{-6}	2.07×10^{-6}
0.4	2.84×10^{-6}	5.04×10^{-6}	1.43×10^{-6}	2.49×10^{-6}
0.5	1.56×10^{-7}	7.48×10^{-7}	7.59×10^{-8}	4.02×10^{-7}
0.6	2.72×10^{-6}	4.50×10^{-6}	1.36×10^{-6}	2.19×10^{-6}
0.7	2.27×10^{-6}	5.11×10^{-6}	1.15×10^{-6}	2.54×10^{-6}
0.8	1.31×10^{-6}	1.40×10^{-6}	6.46×10^{-7}	6.55×10^{-7}
0.9	4.38×10^{-6}	9.07×10^{-6}	2.19×10^{-6}	4.50×10^{-6}
1.0	1.19×10^{-6}	2.47×10^{-6}	5.89×10^{-7}	1.20×10^{-6}

Table 2
$e_j = |\hat{u}_j(x) - u_j(x)|, j = 1, 2$, in some test points x for Example 2.

x	RBFN–ShA		RBFN–MShA	
	e_1	e_2	e_1	e_2
0.0	7.14×10^{-4}	7.54×10^{-5}	4.48×10^{-4}	2.09×10^{-5}
0.1	1.23×10^{-5}	8.54×10^{-6}	1.67×10^{-5}	1.41×10^{-7}
0.2	2.45×10^{-4}	2.28×10^{-5}	1.66×10^{-4}	5.70×10^{-6}
0.3	1.96×10^{-4}	1.04×10^{-5}	1.25×10^{-4}	5.61×10^{-6}
0.4	3.87×10^{-5}	1.98×10^{-6}	1.23×10^{-5}	4.79×10^{-6}
0.5	1.06×10^{-4}	1.83×10^{-6}	8.70×10^{-5}	4.99×10^{-6}
0.6	1.70×10^{-4}	1.16×10^{-5}	1.28×10^{-4}	6.33×10^{-6}
0.7	1.40×10^{-4}	2.99×10^{-5}	1.07×10^{-4}	8.69×10^{-6}
0.8	6.33×10^{-5}	3.80×10^{-5}	6.02×10^{-5}	1.29×10^{-5}
0.9	5.03×10^{-5}	1.61×10^{-5}	6.91×10^{-5}	2.14×10^{-5}
1.0	2.71×10^{-4}	5.82×10^{-5}	2.57×10^{-4}	3.89×10^{-5}

and $\mathcal{E}_2 = 9.30 \times 10^{-6}$. The performance with the gradient method is poor. The errors, in this case, are $\mathcal{E}_1 = 2.27 \times 10^{-3}$ and $\mathcal{E}_2 = 1.58 \times 10^{-3}$, when $m = 8$ and all of the validation data are used. The numerical results on some test data points are shown in Table 1. The results presented in Table 1 show that the performance of RBFN–MShA is better than the RBFN–ShA.

Example 2. Consider the following system of nonlinear Volterra integral equations of the first kind with the exact solutions $u_1(x) = x^2$ and $u_2(x) = x$ (see [21])

$$\int_0^x (1 - x^2 + t^2)(u_1(t) + u_2^3(t))\mathrm{d}t = \frac{1}{3}x^3 + \frac{1}{4}x^4 - \frac{2}{15}x^5 - \frac{1}{12}x^6,$$
$$\int_0^x (5 + x - t)(u_1^3(t) - u_2(t))\mathrm{d}t = -\frac{5}{2}x^2 - \frac{1}{6}x^3 + \frac{5}{7}x^7 + \frac{1}{56}x^8.$$

Similar to Example 1 we start with one node in the hidden layer. Algorithm RBFN–MShA is terminated with $m = 4$ nodes. The maximum errors, in this case, are $\mathcal{E}_1 = 4.48 \times 10^{-4}$ and $\mathcal{E}_2 = 3.89 \times 10^{-5}$. Algorithm RBFN–ShA is terminated with $m = 4$ nodes with the maximum errors $\mathcal{E}_1 = 7.14 \times 10^{-4}$ and $\mathcal{E}_2 = 7.54 \times 10^{-5}$. The errors by using the gradient method are $\mathcal{E}_1 = 6.27 \times 10^{-2}$ and $\mathcal{E}_2 = 6.08 \times 10^{-2}$, when $m = 8$ and all of the validation data are used. The numerical results on some test data points are shown in Table 2.

Example 3. Consider the following system of nonlinear Volterra integral equations of the second kind with the exact solutions $u_1(x) = \sin(x)$ and $u_2(x) = \cos(x)$ (see [22])

$$u_1(x) = \sin(x) - x + \int_0^x \left(u_1^2(t) + u_2^2(t)\right)\mathrm{d}t,$$
$$u_2(x) = \cos(x) - \frac{1}{2}\sin^2(x) + \int_0^x u_1(t)u_2(t)\mathrm{d}t.$$

We start with one node in the hidden layer. Algorithm RBFN–MShA is terminated with $m = 6$ nodes. The maximum errors, in this case, are $\mathcal{E}_1 = 1.50 \times 10^{-6}$ and $\mathcal{E}_2 = 4.40 \times 10^{-6}$. Algorithm RBFN–ShA is terminated with $m = 6$ nodes with the maximum errors $\mathcal{E}_1 = 1.56 \times 10^{-6}$ and $\mathcal{E}_2 = 6.13 \times 10^{-6}$. The errors by using the gradient method are $\mathcal{E}_1 = 5.80 \times 10^{-2}$ and $\mathcal{E}_2 = 2.51 \times 10^{-2}$, when $m = 8$ and all of the validation data are used. The numerical results on some test data points are shown in Table 3.

Table 3
$e_j = |\hat{u}_j(x) - u_j(x)|, j = 1, 2$, in some test points x for Example 3.

x	RBFN–ShA		RBFN–MShA	
	e_1	e_2	e_1	e_2
0.0	2.86×10^{-7}	1.66×10^{-6}	6.94×10^{-7}	7.85×10^{-7}
0.1	4.05×10^{-7}	1.95×10^{-6}	8.33×10^{-7}	9.47×10^{-7}
0.2	1.15×10^{-7}	1.01×10^{-6}	2.85×10^{-7}	4.37×10^{-7}
0.3	2.64×10^{-7}	1.70×10^{-6}	4.35×10^{-7}	7.52×10^{-7}
0.4	2.05×10^{-7}	2.47×10^{-7}	1.58×10^{-7}	1.41×10^{-7}
0.5	8.01×10^{-8}	1.81×10^{-6}	5.03×10^{-7}	8.75×10^{-7}
0.6	4.28×10^{-9}	1.02×10^{-6}	2.67×10^{-7}	5.93×10^{-7}
0.7	5.74×10^{-7}	8.03×10^{-7}	6.69×10^{-8}	6.27×10^{-8}
0.8	8.70×10^{-7}	1.18×10^{-8}	2.15×10^{-7}	6.08×10^{-7}
0.9	3.13×10^{-7}	4.65×10^{-6}	1.16×10^{-6}	3.25×10^{-6}
1.0	1.56×10^{-6}	3.57×10^{-6}	1.23×10^{-6}	3.70×10^{-6}

6. Conclusions

The training of the RBF network that uses a combination of gradient and Newton methods as a backpropagation algorithm is described and illustrated. Moreover, this paper proposes a novel approach to optimize the output weights of the RBF network, which is a new modification strategy based on the original algorithm ShA [16].

In applications to systems of nonlinear integral equations, both RBFN–ShA and RBFN–MShA provide only a small number of nodes to achieve good approximations. In particular, the experimental results also illustrate that RBFN–MShA provides almost more accurate solutions than the other one.

In this paper, in order to keep the design of the network simple, the centers and the widths of the RBFs are chosen in advance. However, they can be included in the list of unknown parameters in the optimization procedure, which may require more computational time.

The method developed here is a general method and can be applied to some other types of integral equations.

References

[1] D.S. Broomhead, D. Lowe, Multivariate functional interpolation and adaptive networks, Complex Systems 2 (1988) 321–355.
[2] J. Moody, C. Darken, Fast learning in networks of locally-tuned processing units, Neural Computation 1 (2) (1989) 281–294.
[3] S. Renals, R. Rohwer, Phoneme classification experiments using radial basis function, in: Proceedings of International Joint Conference on Neural Networks, I, 1989, pp. 461–467.
[4] F. Girosi, T. Poggio, Neural networks and the best approximation property, Biological Cybernetics 63 (1990) 169–176.
[5] E. Hartman, J. Keeler, J. Kowalsky, Layered neural networks with gaussian hidden units as universal approximations, Neural Computation 2 (1990) 210–215.
[6] J. Park, I. Sandberg, Universal approximation using radial basis function networks, Neural Computation 3 (2) (1991) 246–257.
[7] J. Park, I. Sandberg, Approximation and radial-basis-function networks, Neural Computation 5 (1993) 305–316.
[8] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice-Hall, New Jersey, 1999.
[9] S. Chen, C.F.N. Cowan, P.M. Grant, Orthogonal least squares learning algorithm for radial basis function networks, IEEE Transactions on Neural Networks 2 (1991) 302–309.
[10] J.C. Platt, A resource-allocating network for function interpolation, Neural Computation (1991) 213–225.
[11] Y.H. Cheng, C.S. Lin, A Learning Algorithm for Radial Basis Function Networks with the Capability of Adding and Pruning Neurons, IEEE ICNN, Orlando, 1994, pp. 797–801.
[12] A. Esposito, M. Marinaro, D. Oricchio, S. Scarpetta, Approximation of continuous and discontinuous mappings by a growing neural RBF-based algorithm, Neural Networks 13 (2000) 651–665.
[13] S. Lee, S. Shimoji, Self-organisation of Probabilistic Network with Gaussian Mixture Model, IEEE ICNN, Orlando, 1994, pp. 3088–3093.
[14] C.S. Lin, Y.H. Cheng, Radial Basis Function Networks for Adaptive Critic Learning, IEEE ICNN, Orlando, 1994, pp. 903–906.
[15] N. Karayiannis, Reformulated radial basis neural networks trained by gradient descent, IEEE Transactions on Neural Networks 3 (1999) 657–671.
[16] Y. Shi, Globally convergent algorithms for unconstrained optimization, Computational Optimization and Applications 16 (2000) 295–308.
[17] L. Jianyu, L. Siwei, Q. Yingjian, H. Yaping, Numerical solution of elliptic partial differential equation using radial basis function neural networks, Neural Networks 16 (2003) 729–734.
[18] A. Golbabai, S. Seifollahi, Radial basis function networks in the numerical solution of linear integro-differential equations, Applied Mathematics and Computation 188 (2007) 427–432.
[19] R. Fletcher, Practical Methods of Optimization, second edition, Wiley, New York, 1987.
[20] E. Babolian, J. Biazar, A.R. Vahidi, The decomposition method applied to systems of Fredholm integral equations of the second kind, Applied Mathematics and Computation 148 (2004) 443–452.
[21] J. Biazar, E. Babolian, R. Islam, Solution of a system of Volterra integral equations of the first kind by Adomian method, Applied Mathematics and Computation 139 (2003) 249–258.
[22] J. Biazar, H. Ghazvini, He's homotopy perturbation method for solving systems of Volterra integral equations of the second kind, Chaos, Solitons and Fractals 39 (2) (2009) 770–777.