*IUST International Journal of Engineering Science,*
**Vol. 19, No.5-2, 2008, Page 1-7**

# A NEW STRATEGY FOR TRAINING RBF NETWORK WITH APPLICATIONS TO NONLINEAR INTEGRAL EQUATIONS

**A. Golbabai, M. Mammadov & S. Seifollahi**

***Abstract:*** *A new learning strategy is proposed for training of radial basis functions (RBF) network. We apply two different local optimization methods to update the output weights in training process, the gradient method and a combination of the gradient and Newton methods. Numerical results obtained in solving nonlinear integral equations show the excellent performance of the combined gradient method in comparison with gradient method as local back propagation algorithms.*

***Keywords:*** *RBF network, Gradient method, Newton's method, Nonlinear integral equations*



## 1. Introduction

Radial basis functions (RBF) neural 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. RBF networks provide an attractive approach for function approximation because of their simple architecture, computational efficiency, powerful generalization capability, and learning schemes. These networks as presented earlier have been shown to be universal approximators, i.e. theoretically, any continuous function defined on a compact set can be approximated to a prescribed degree of accuracy by increasing the number of hidden nodes (see [4], [5], [6] and [7]).

In training RBF network, decisions are made on the number of hidden nodes, the widths of the functions, and the output layer training. The number of hidden nodes and the widths are generally decided in advance by examining the vectors in the training data. Then some prespecified optimization methods, for example the gradient based methods or other techniques in the literature, can be adopted to update the weights connecting the hidden layer and the output layer which measure the strength of the connections between nodes. Of course, we still can use the optimization method to update all parameters of the network.

The steepest descent method (gradient method) is commonly used in neural network training. However, this method suffers from the slow training speed and is easy plunging into local minima. In order to accelerate training speed and avoid spurious local minima, many methods have been used to overcome these difficulties [8]. 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 problems [9]. It is shown that this method is global convergent and at the same time has a good convergence rate. This method can be used instead of the gradient method in backpropagation phase of the network training to overcome the above mentioned difficulties.

In recent years, RBF network has emerged as an important type of method for the numerical solution of differential equations and more recently it was used for solving linear integral equations (see [10] and [11]). In the present paper, we solve nonlinear Fredholm integral equations by using RBF network. From the variety of learning algorithms for the RBF network in the literature such as the OLS algorithm [12], the resource allocation network [13], and various implementations of adaptively growing and pruning algorithms (see [14], [15], [16] and [17]), we consider a simple structure of growing algorithm. In the implementation of this algorithm, a trial solution of the integral equation is given 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 residua. This error function can be easily expressed as an unconstrained optimization problem. Different numerical techniques can be applied to solve

***Paper first received July. 15, 2008, and in revised form May. 17, 2009.***
***A. Golbabai*** is with the Department of Mathematics, Iran University of Science and Technology, Tehran, Iran. golbabai@iust.ac.ir
***M. Mammadov*** is with the School of Information & Mathematical Science, Ballarat University, Ballarat VIC, Australia. m.mammadov@ballarat.edu.au
***S. Seifollahi*** is with the Department of Mathematics, University of Mohaghegh Ardabili, Ardabil, Iran, sattarseif@gmail.com

it. In this study, after the RBF parameters are fixed, the output weights by using a local optimization method is updated. We adopt two different local optimization methods, the gradient method and the combined gradient method. The performance of these methods is compared in solving nonlinear Fredholm integral equations.

## 2. RBF Network Architecture

The basic architecture of a RBF network is consist of three entirely different layers. The nodes within each layer are fully connected to the previous layer. The first layer is an input layer in which each node corresponds to an attribute of an input sample, and pass directly to the hidden layer without weights, i.e., the weight connection is unity. The second layer is a hidden layer that serves a different purpose from that of the output layer. The third layer is an output layer responding to the input samples. 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. An activation function for a hidden layer node is a locally radially symmetric function. A popular choice of the RBF is as follows:

$$\phi_i(\mathbf{x}) = \exp(-\|\mathbf{x}-\mathbf{c_i}\|^2 / 2a_i^2), \quad (1)$$

where $\mathbf{x}$ is an input vector, $\mathbf{c_i}$ and $a_i$ are the center and the width of $i$-th hidden node respectively, and $\|\cdot\|$ is the Euclidean norm.

Without loss of generality, we consider a RBF network with only one output. Mathematically, for input vector $\mathbf{x}$ the output of RBF network, which implements a sum of arbitrary basis functions defined on its inputs, is potentially a flexible and efficient structure for approximating arbitrary nonlinear functions, which is expressed as:

$$u(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}), \quad (2)$$

where $m$ is the number of hidden nodes and $w_i$ is the weight from $i$-th RBF node to the output node. For more details about RBF and RBF network, we refer to the recent books by Buhmann [18] and Haykin [19].

## 3. Optimization Strategies

The error function most commonly used in the RBF network is the sum square error (SSE). For a RBF network with one output node the SSE can be written

$$l(z) = \sum_{p=1}^{n} e_{p,m}^2, \quad (3)$$

where $e_{p,m}^2$ is the output residual in the presence of the $p$-th input sample, $m$ is the number of hidden nodes, $n$ is the number of input samples to be trained, and $z = (w, c, a)$ stands for the network parameters where $w = \{w_1, ..., w_m\}$, $c = \{c_1, ..., c_m\}$ and $a = \{a_1, ..., a_m\}$ used in (1)-(3).

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 find the optimal parameters. To find a solution of this problem, among the variety of the existing methods, the descent direction methods are the most commonly used techniques in this area due to their fast convergence property. If we denote $\nabla l(z_k)$ by $g_k$, then the steps of the general descent direction methods for solving problem (3) are as follows:

### 3.1. Algorithm: General Descent Method

0. Choose a starting point $z_0 \in R^q$, and an error tolerance $\varepsilon > 0$.
1. For $k = 0,1,2,\ldots$.
2. If $\|g_k\| < \varepsilon$, then stop.
3. Compute the descent direction $d_k$ at $z_k$ satisfying

$$g_k^t d_k < 0. \quad (4)$$

4. Determine an appropriate step length $\alpha_k > 0$.
5. Set $z_{k+1} = z_k + \alpha_k d_k$, and go to step 1.

The above algorithm is a very general algorithm, and there are different methods correspond to different ways of choosing $d_k$ and $\alpha_k$ where $d_k$ is a general descent direction and $\alpha_k$ is a line search factor. There are some criterions for accepting $\alpha_k$ as an admissible step length such as backtracking method, Armijo, Goldestain and Wolfe line search methods. In Wolfe case, the step length $\alpha_k$ is determined by an inexact line search along the direction $d_k$ satisfying

$$l(z_k + \alpha d_k) \le l(z_k) + \rho_1 \alpha g_k^t d_k, \quad (5)$$

$$g_{k+1}^t d_k \ge \rho_2 g_k^t d_k, \quad (6)$$

where $\rho_1 \in (0, 1/2)$ and $\rho_2 \in (\rho_1, 1)$ are constant parameters. By considering Wolfe conditions (5) and (6) and assuming that the gradient is uniformly continuous on the level set $\Omega = \{z : l(z_k) \le l(z_0)\}$, it is shown that the above algorithm is global convergent [8].

One of the most widely used methods is the steepest descent method (or gradient method), in which

$d_k = -g_k$, for all $k$. This method is global convergent and can be used to find a local solution of (3). Unfortunately, although the method is of global convergent property and usually works well in some early steps, as a stationary point is approached, it descends very slowly with zigzagging phenomena. In fact, it is shown that the convergence rate of the gradient method is at least linear [8].

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

$$H_k d = -g_k, \qquad (7)$$

where $H_k$ is the Hessian matrix at $z_k$. If $H_k$ is positive definite then the Newton's direction is a descent direction and consequently the system has a unique solution. Even when $H_k$ is positive definite, it is not guaranteed that Newton method has the global convergence property. Consequently, although Newton method generally converges in fewer iterations than 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 $H_k$ may be used. 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 considered as a combination of different directions. One of the most successful method of this group is a combination of gradient and Newton based methods, especially, when the direction is made so that as close as possible to the Newton method [15]. If $\delta$, $\eta$, $\rho_1$ and $\rho_2$ be four parameters so that $0<\delta<1$, $0<\eta<1$, $0<\rho_1<1/2$ and $\rho_1<\rho_2<1$, then the combined gradient algorithm is as follows:

### 3.2. Algorithm: Combined Gradient Method

0. Choose a starting point $z_0 \in R^q$, and an error tolerance $\varepsilon > 0$.
1. For $k = 0,1,2,\ldots$.
2. If $\|g_k\| < \varepsilon$, then stop.
3. If (7) is solvable at $z_k$, compute the Newton's direction $d_1$ at $z_k$ from (7) and go to step 4. Otherwise, compute the gradient direction $d_2$ at $z_k$, set $d_k = d_2$ and go to step 8.
4. If $k=1$ or if $\|g_k\| \le \|g_{k-1}\|$, set $\bar{z} = z_k + d_1$ and go to step 5, otherwise go to step 6.
5. If $l(\bar{z}) < l(z_k)$ and $g(\bar{z}) \le \eta\, g(z_k)$, then set $z_{k+1} = \bar{z}$ and go to the next iteration, otherwise go to step 6.
6. If $d_1 d_2 \ge \delta \|d_1\| \|d_2\|$, then set $d_k = d_1$ and go to step 8, otherwise go to step 7.
7. If $d_1 d_2 < 0$, then set $d_k = d_2$ and go to step 8, otherwise compute $\bar{\lambda}$ such that

$$\bar{\lambda} = \min\{\lambda;\ 0 \le \lambda \le 1,\ \frac{\left((1-\lambda)d_1 + \lambda d_2\right)^t d_2}{\|(1-\lambda)d_1 + \lambda d_2\| \|d_2\|} \ge \delta\}$$

and set $d_k = (1-\bar{\lambda})d_1 + \bar{\lambda} d_2$.

8. Determine an acceptable $\alpha_k > 0$ along $d_k$.
9. Set $z_{k+1} = z_k + \alpha_k d_k$ and go to step 1.

The parameters $\delta$, $\eta$, $\rho_1$ and $\rho_2$ used in this algorithm are: $\delta = 0.001$, $\eta = 0.99$, $\rho_1 = 0.001$ and $\rho_2 = 0.9$ (for more details see [9]). In [9], they used the line search rules (5) and (6) to determine an acceptable $\alpha_k$. In this paper, we used the following procedure which aimed to approximate the analytical formula

$$\alpha_k = -\frac{d_k^t g_k}{d_k^t H_k d_k}, \qquad (8)$$

in order to avoid calculating the Hessian matrix $H_k$ at each iteration [20].

- ☑ Calculate $s_k$:

$$s_k = \frac{\nabla l(z_k + \vartheta d_k / \|d_k\|) - \nabla l(z_k)}{\vartheta / \|d_k\|}.$$

- ☑ Calculate $\xi_k = d_k^t (s_k + \mu d_k)$.
- ☑ If $\xi_k \le 0$, set $\xi_k = -\xi_k + \mu \|d_k\|^2$.
- ☑ Calculate step size

$$\alpha_k = -\frac{d_k^t g_k}{\xi_k}. \qquad (9)$$

Typical values for two user dependent parameters, $\vartheta$ and $\mu$, are: $0 < \vartheta \le 10^{-4}$, $0 < \mu \le 10^{-6}$ which, here, are considered as $\vartheta = 10^{-4}$ and $\mu = 10^{-10}$ (for more details see [20] and [21]).

## 4. RBF Network Training

Let us consider a RBF network with only one node in the output layer. Among the variety of learning

methods in the literature, we choose a growing based architecture. For the sake of simplicity, in this paper, we define the set of centers of RBFs fixed in advance. Thus, the determination of the widths and the output weights are 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 first initialize the width of the new node and after the output weights are updated, the widths of all nodes are optimized according to the following procedure given in [15]:

$$a_i(t+1) = a_i(t) + N(0,\sigma)a_i(t),\ 0 \le i \le m \qquad (10)$$

where $N(0,\sigma)$ is a random number chosen from a Gaussian probability distribution with mean zero and variance $\sigma$ and $a_i(t)$ is the $i$-th width in $t$-th iteration of the training process. If the training error on $t$-th iteration is less than the previous one, then the widths are changed according to (10), otherwise, the old values of widths are kept. The value $\sigma$ is initialized to a fixed value and then modified according to the following rule:

$$\sigma(t+1) = \begin{cases} b_1\sigma(t), & if\ SSE\ increased, \\ b_2\sigma(t), & if\ SSE\ decreased, \\ \sigma(t), & if\ SSE not changed, \end{cases} \qquad (11)$$

where $b_1$ and $b_2$ are constants close to $1$ such that $b_1 < 1$ and $b_2 > 1$, and SSE is the training error.

The learning algorithm of RBF network to approximate any function is briefly summarized below. This algorithm is a growing based algorithm in which the network is allowed to grow step by step (see [11], [14] and [15]). If we denote the set of centers by $c = \{c_1,...,c_m\}$, the set of validation data by $X = \{x_1, x_2,..., x_n\}$ and the set of training data by $X_t$ which here is considered as a subset of X, then the algorithm steps are as follows:

### 4.1. Algorithm: RBF Network Learning

0. Given a tolerance $\varepsilon > 0$, $m_0$ centers from $c$ and calculating $m_0$ initial values for the widths, and a starting point $w \in R^{m_0}$ and one sample from $X$, do step 1 to 6.
1. For $k = 0,1,2,...$.
2. Apply the local optimization method to obtain the unknown parameters (weights) of the network.
3. Calculate all of the widths by using (10).
4. If $SSE < \varepsilon$ over the validation data $X$ or $k>n$, then stop, otherwise go to 5.
5. If $k \neq 1$ and current validation error greater than the old one, then add the number of the hidden nodes by one, select its center from $c$ and initialize its width.
6. Select a new sample from $X \setminus X_t$ and go to the next iteration.

According to the above instruction, the algorithm is terminated if the validation error during a given iteration less than a given tolerance $\varepsilon$ or $k > n$. The validation error will normally decrease during the phase of training, as does the training error. At first, this algorithm starts with a few nodes in the hidden layer, then we continue the training process by incorporating more training samples and more hidden nodes one by one 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 be begin to rise. To overcome to this difficulty and having a network with as possible as complexity, selection of new sample and new node can be affect in the training process which is described below and is different from other existing approaches (see [11] and [15]).

### 4.2. Selection of New node

To select a new training sample, we calculate the error weight $e^2_{j,m}$ on the set $X \setminus X_t$. 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 computation and reducing the number of existing training data, after some iteration the new sample is replaced by one of the training data.

More precisely, the error weights on training set $X_t$ are calculated and the data which has the smallest error weight is discarded from $X_t$ and a new training sample from $X \setminus X_t$, which has a maximum distance from $X_t$ and has not been in the network, is selected.

### 4.3. Initialization of New Node

In 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 the maximum among all the other candidate centers not used in the network. We initialize the width of the inserted node as $a_m = \beta / m$, where $\beta$ is a positive constant. The idea behind this initialization is that when the network grows ($m$ increases) the values of widths decreases.

## 5. Application to Integral Equations

In order to show the network performance, we consider nonlinear integral equation of Fredholm type:

$$u(x)-\int_a^b K(x,t,u(t))\,dt = g(x), \qquad (12)$$

where $u(x)$ is an unknown function, $g(x)$, $K(x,t,u)$ are given continuous functions, with $K(x,t;u)$ nonlinear in $u$. This integral equation has received considerable interest in the mathematical applications in different areas of engineering, mechanics, potential theory, electrostatics, and etc.

For implying the learning algorithm to (12) and obtaining an approximate solution, the error function (3) is written as follows:

$$l(z)=\sum_{p=1}^{n}[u_a(x_p)-\int_a^b K(x_p,t;u_a(t))dt - g(x_p)]^2,$$

where $u_a$ is an approximate solution to (12) which here is considered as in (2). Also, the sets $X$ and $C$ used in the learning algorithm are considered as a number of points of $[a,b]$.

## 6. Numerical Results

For general nonlinear kernel $K(x,t;u)$, there is no simple way to evaluate analytically the integral (12), so a numerical integration scheme is used. In our calculations, we use the Simpson numerical integration scheme with 20 subintervals. As a measure of the accuracy of solutions, the max error is given as

$$N_e = \max_{1\le j\le n_t}\left|u_a(x_j)-u_e(x_j)\right|, \qquad (13)$$

where $u_a(x_j)$ and $u_e(x_j)$ are the calculated and exact solutions values at the point $x_j$ 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 Gaussian functions, but a number of alternatives can also be used (see [18] and [19]). Also, when $n>5$, the new training sample is replaced by one of the existing training data.

The number of validation data used in the training process is 200 data distributed randomly in $[a,b]$ and the performance of the network is tested using $n_t=500$ random data of $[a,b]$. The numerical results using the combined gradient method is compared with the results using the gradient method with the same nodes.

The CPU time is measured within MATHEMATICA running on (single user) Windows XP Professional operating system (version SP 2) with a 1.83 GHz Intel Pentium Centrino Duo with 512 MB of RAM.

**Example 1.** Here we solve (12) with $a=0$, $b=1$, $g(x)=x-\pi/8$, $K(x,t;u)=1/2(1+u^2(t))$, and the exact solution $u(x)=x$ (see [22]).

The network begins with one node in the hidden layer and as observations are received, new hidden nodes are added one by one. By using combined gradient method to update the output weights in step (2) of the learning algorithm, it is terminated with $m=5$ nodes and the maximum error is $N_e=2.38173\times10^{-7}$ on total CPU time $137.15$ seconds used in the training process. By using the gradient method instead of combined gradient method and in order simplify comparison we terminate the algorithm when $m=5$ and the current validation error is greater than the old one.

The maximum error in this case is $N_e=1.91050\times10^{-4}$ on total CPU time $1020.38$ seconds. Also, the numerical results with $m=5$ on some test data points are shown in Table 1.

**Tab. 1.** $|u_a(x)-u_e(x)|$ **in some test data points for example 1**

| $x$ | **Gradient method** | **Combined method** |
|---|---|---|
| 0.0 | $5.40059\times10^{-5}$ | $2.34437\times10^{-8}$ |
| 0.1 | $1.69353\times10^{-5}$ | $2.06671\times10^{-7}$ |
| 0.2 | $3.88651\times10^{-5}$ | $5.86266\times10^{-8}$ |
| 0.3 | $2.97163\times10^{-5}$ | $7.01560\times10^{-8}$ |
| 0.4 | $5.52389\times10^{-6}$ | $6.58316\times10^{-8}$ |
| 0.5 | $1.96329\times10^{-5}$ | $2.07433\times10^{-8}$ |
| 0.6 | $3.36744\times10^{-5}$ | $7.90716\times10^{-8}$ |
| 0.7 | $2.65491\times10^{-5}$ | $2.86902\times10^{-8}$ |
| 0.8 | $9.75199\times10^{-6}$ | $1.07770\times10^{-7}$ |
| 0.9 | $8.11962\times10^{-5}$ | $1.49009\times10^{-7}$ |
| 1.0 | $1.91050\times10^{-4}$ | $2.12532\times10^{-7}$ |

**Example 2.** Here we solve (12) with $a=0$, $b=1$, $g(x)=\sin(\pi x/2)+2x\ln(3)$, $K(x,t,u)=\left(4tx+\pi x\sin(\pi t)\right)/\left(u^2(t)+t^2+1\right)$, and the exact solution $u(x)=\sin(\pi x/2)$ (see [23]).

Similar to example 1 we start with $m=1$ node in the hidden layer and after some iteration, it is terminated with $m=5$ nodes. By using combined gradient method to update the output weights, the maximum error is $N_e=1.73497\times10^{-5}$ on total CPU time $38.5$ seconds. By using the gradient method and similar termination to example 1, the maximum error is $N_e=1.37233\times10^{-3}$ on total CPU time $724.14$ seconds. The numerical results with $m=5$ on some test data points are shown in Table 2.

**Tab. 2.** $|u_a(x)-u_e(x)|$ **in some test data points for example 2**

| $x$ | Gradient method | Combined method |
|---|---|---|
| 0.0 | $5.64785\times10^{-4}$ | $1.73497\times10^{-5}$ |
| 0.1 | $9.04185\times10^{-4}$ | $8.59420\times10^{-6}$ |
| 0.2 | $1.36932\times10^{-3}$ | $6.08925\times10^{-6}$ |
| 0.3 | $1.13305\times10^{-3}$ | $1.15355\times10^{-6}$ |
| 0.4 | $4.87622\times10^{-4}$ | $3.42662\times10^{-6}$ |
| 0.5 | $2.93183\times10^{-4}$ | $5.03367\times10^{-7}$ |
| 0.6 | $9.61713\times10^{-4}$ | $3.08662\times10^{-6}$ |
| 0.7 | $1.30331\times10^{-3}$ | $2.45955\times10^{-6}$ |
| 0.8 | $1.14218\times10^{-3}$ | $3.68792\times10^{-6}$ |
| 0.9 | $3.46185\times10^{-4}$ | $9.73833\times10^{-6}$ |
| 1.0 | $1.16965\times10^{-3}$ | $5.13193\times10^{-7}$ |

## 7. Conclusion

Training of RBF network that use gradient or combined gradient method as a backpropagation algorithm is described and illustrated. 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 networks, which trained with both the gradient and combined gradient methods, provide only a small number of nodes to achieve quite good approximations. The numerical examples show that the network with the combined gradient method provides more accurate solutions than the gradient method (using the same number of nodes). Moreover, in the combined gradient method case, the learning is much faster. For instance, in example 2 with the combined gradient method the learning process takes 38.5 seconds, but in the gradient method case it takes 724.142 seconds and does not achieve as good accuracy as in the case of combined gradient method.

The method developed here can be applied to some other types of integral equations.

## References

[1] Broomhead, D.S., Lowe, D., "*Multivariate Functional Interpolation and Adaptive Networks*," Complex Systems, Vol. 2, 1988, pp. 321-355.

[2] Moody, J.E., Darken, C.J., *"Fast Learning in Networks of Locally-Tuned Processing Units*," Neural Computation, Vol. 1, 1989, pp. 281-294.

[3] Renals, S., Rohwer, R., *"Phoneme Classification Experiments using Radial Basis Function*," in Proceedings of International Joint Conference on Neural Networks, I, 1989, pp. 461–467.

[4] Girosi, F., Poggio, T., *"Neural Networks and the Best Approximation Property*," Biological Cybernetics, Vol. 63, 1990, pp. 169-176.

[5] Hartman, E., Keeler, J., Kowalsky, J., "*Layered Neural Networks with Gaussian Hidden Units as Universal Approximations,"* Neural Computation, Vol. 2, 1990, pp. 210-215.

[6] Park, J., Sandberg, I.W., *"Universal Approximation Using radial Basis Function Networks*," Neural Comput., Vol. 3(2), 1991, pp. 246-257.

[7] Park, J., Sandberg, I., *"Approximation and Radial-Basis-Function Networks*," Neural Computation, Vol. 5, 1993, pp. 305-316.

[8] Nocedal, J., Wright, S.J., *Numerical Optimization*, Springer-Verlag, New York, 1999.

[9] Shi, Y., *"Globally Convergent Algorithms for Unconstrained Optimization*," Computational Optimization and Applications, Vol. 16, 2000, pp. 295-308.

[10] Golbabai, A., Seifollahi, S., "*Radial Basis Function Networks in the Numerical Solution of Linear Integro-Differential Equations,"* Appl. Math. Comput., Vol. 188, 2007, pp. 427-432.

[11] Jianyu, L., Siwei, L., Yingjian, Q., Yaping, H., "*Numerical Solution of Elliptic Partial Differential Equation Using Radial Basis Function Neural Networks*," Neural Networks, Vol. 16, 2003, pp. 729-734.

[12] Chen, S., Cowan, C.F.N., Grant, P.M., "*Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks*," IEEE Trans. Neural Networks, Vol. 2, 1991, pp. 302-309.

[13] Platt, J.C., *"A Resource-Allocating Network for Function Interpolation,"* Neural Computation, 1991, pp. 213-225.

[14] Cheng, Y.H., Lin, C.S., *"A Learning Algorithm for Radial Basis Function Networks with the Capability of Adding and Pruning Neurons*," IEEE ICNN, 1994, pp. 797-801.

[15] Esposito, A., Marinaro, M., Oricchio, D., Scarpetta, S., *"Approximation of Continuous and Discontinuous Mappings by a Growing Neural RBF-Based Algorithm*," Neural Networks, Vol. 13, 2000, pp. 651-665.

[16] Lee, S., Shimoji, S., *"Self-Organisation of Probabilistic Network with Gaussian Mixture Model*," IEEE ICNN, 1994, pp. 3088-3093.

[17] Lin, C.S., Cheng, Y.H., "*Radial Basis Function Networks for Adaptive Critic Learning*," IEEE ICNN, 1994, pp. 903-906.

[18] Buhmann, M.D., *Radial Basis Functions: Theory and Implementations*, Cambridge University Press, Cambridge, 2003.

[19] Haykin, S., *Neural Networks: a Comprehensive Foundation*, New Jersey: Prentice-Hall, 1999.

[20] Saini, L.M., Soni, M.K., "*Artificial Neural Network-Based Peak Load Forecasting Using Conjugate Gradient Methods*," IEEE Transactions on Power Systems, 17(3), 2002.

[21] Moller, M.F., "*A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*," Neural Networks, Vol. 6, 1993, pp. 525-533.

[22] Yousefi, S., Razzaghi, M., *"Legendre Wavelets Method for the Nonlinear Volterra–Fredholm integral Equations*," Mathematics and Computers in Simulation, Vol. 70, 2005, pp. 1-8.

[23] Maleknejad, K., Karami, M., Aghazadeh, N., "*Numerical Solution of Hammerstein Equations Via an Interpolation Method*," Appl. Math. Comput., Vol. 168, 2005, pp. 141-145.