International Journal of Pure and Applied Mathematics
Volume 46 No. 2 2008, 191-197

Invited Lecture Delivered at
Forth International Conference of Applied Mathematics
and Computing (Plovdiv, Bulgaria, August 12–18, 2007)

NUMERICAL SOLUTION OF WEAKLY SINGULAR INTEGRAL EQUATIONS BY A GROWING RBF NETWORK

A. Golbabai[1] §, S. Seifollahi[2], R. Gholami[3]
[1,2,3]Department of Mathematics
Islamic Azad University (Karaj Unit)
Rajae Shahr, Karaj, IRAN
[1]e-mail: golbabai@iust.ac.ir

Abstract: This article presents a new method to solve linear weakly singular Volterra integral equations using radial basis function (RBF) networks. The approximate solution is written as a linear combination of radial basis functions. This method employs a growing RBF network, as the basic approximation element, whose parameters are obtained by optimization of the appropriate error functional composed of the integral equation residue. The simulation results are investigated for the viability of efficiently determining the solution of the integral equation.

AMS Subject Classification: 45G05, 45E10, 45L10
Key Words: weakly singular integral equation, radial basis function networks, unconstrained optimization

1. Introduction

The objective of this paper is to solve linear integral equation of the form:

$$f(x) - \lambda \int_a^x \frac{k(x,t,f(t))}{\sqrt{x-t}} dt = g(x)\,, \tag{1}$$

where the parameter λ and the functions g and k are given and f is the unknown function to be determined [2].

Received: August 17, 2007

§Correspondence author

In the recent years, there has been a growing interest in applying artificial neural networks to many areas of science and engineering, such as control systems, pattern recognition, etc. Many different structural architectures of neural network model have been proposed; however, RBF networks are especially interesting since they are commonly used, because of their simple but adaptive structure and effective learning ability. These networks were brought to widespread attention by Broomhead and Lowe [1] in 1988. The architecture and training algorithms for RBF networks are simple and they train more quickly than do multiple layered perceptron networks.

It has been proved that RBF networks have the property of universal approximation [9]. Li, Jianyu et al [5] have employed the gradient type algorithm for training and configuring of the growing RBF network and then they have extended the method to solve partial differential equations [6]. For the purpose of integral equations, the RBF networks have been recently applied for solving the linear integral equations [3]. Here, we restrict our attention to the solution of the univariate weakly singular integral equations of Volterra type by using the RBF network. The present method uses a growing-network paradigm in which the number of hidden units is not given a priori and a new hidden node is inserted only after a certain criterion has been met.

The paper is organized in the following way. In the next section, approximation of function using a RBF network is reviewed, then RBF network method based on BFGS backpropagation (BFGS-BP) for solving the integral equation is illustrated. To show the effects of the method, two numerical examples are given in Section 5. Section 6 contains conclusions and directions for future research.

2. Approximation Representations of the RBF Networks

The basic architecture for a RBF is a 3-layer network. The input layer is simply a fan-out layer and does no processing. The second or hidden layer performs a non-linear mapping from the input space into a (usually) higher dimensional space in which the patterns become linearly separable. The final layer therefore performs a simple weighted sum with a linear output [4]. A RBF neural network with one output unit can be formulated as

$$f_p(x) = \sum_{i=}^{m} w_i \phi_i(x)\,, \tag{2}$$

where $x \in \mathbb{R}^d$ is the input vector (in this paper $d = 1$), m is the number of RBFs (or hidden nodes), $\{w_i\}_{i=1}^m$ are the output weights of the network and the function ϕ is the RBF that depends upon the radial distance between x and c_i:

$$\phi_i : \mathbb{R}^d \to \mathbb{R}, \quad \phi_i(x) = \phi(\|x - c_i\|)\,,$$

where $\|.\|$ denotes the Euclidean norm, and $c_i \in \mathbb{R}^d$ are the centers of RBFs. In practical applications the number of the hidden nodes is usually much less than the number of training samples, i.e. $m \ll n$.

There have been a number choices for the function ϕ which can be either globally supported or compactly supported. In our experiment results, we will concentrate on the use of globally supported functions. The most prominent choices are Gaussian, multiquadric (MQ) and Inverse MQ functions of the forms, respectively,

$$\phi_i(x) = e^{-\frac{r^2}{2a_i^2}}\,, \tag{3}$$

$$\phi_i(x) = (r^2 + a_i^2)^{\frac{1}{2}}\,, \tag{4}$$

$$\phi_i(x) = (r^2 + a_i^2)^{\frac{-1}{2}}\,, \tag{5}$$

where $r = \|x - c_i\|$ and a_i is a real constant and is called the shape parameter or width of the RBF that must be specified by the user. These function due to having the parameter a_i have exponentially convergence.

Choosing the number of hidden neurons is an important issue when applying the RBF networks. Redundant hidden neurons often lead to over-fitting of the noisy training data and make the network computationally demanding. On the other hand, too small a hidden layer size may make the network unable to model the training data. In order to cope with this problem, we use an incremental architecture of the network [7].

3. Problem Formulation

First, let us begin to replace equation (1) with the following one:

$$\widehat{f} - \lambda K \widehat{f} = g, \tag{6}$$

where

$$(\mathcal{K}\widehat{f})(x) = \int_a^{x-\delta} \frac{k(x, t, \widehat{f}(t)}{\sqrt{x - t}} dt\,,$$

where δ is a small number, say $\delta = 10^{-15}$. In the above equation, the second term of integration was dropped.

In order to obtain the solution of the integral equations (1), we use the least squares design of the equation (4). In other words, determine the parameters of the network such that the following objective function

$$F(p) = \sum_{j=1}^{n_1} \left[\Big(f_p - \lambda K f_p - g \Big)(x_j) \right]^2 \tag{7}$$

is minimized, where n is the number of training data points, and the subscript p refers to the network parameters, $p = (w, c, \sigma)$.

A two stage training procedure for the RBF network is adopted. In the first stage, we fix c_i and a_i and the purpose is to find w_i that minimizes the $F(p)$. We then keep w_i fixed and find c_i and a_i during the second stage of training that minimize the $F(p)$.

The performance of RBF network can critically depend on the choices of RBF centers and their widths, specially the small or large values of the widths make the response of neuron too peaked or too flat, respectively, and therefore should be avoided. In this work, the centers are selected from the data. One way to choose a new node from the preselected nodes is to use its error component as follows [6]:

$$e^{(j)} = \left[\Big(\widehat{f}_p - \lambda K \widehat{f}_p - g \Big)(c_j) \right]^2, \tag{8}$$

where c_j is a new unselected center with maximum weight error (8). Also, the widths are determined according to the following relation:

$$a_i = \beta \min_{i \neq j} \{ c_i - c_j | c_i, c_j \in C \}, \qquad i = 1, 2, \ldots, n_c \,, \tag{9}$$

where β is a positive scalar and n_c is the number of preselected nodes. Since the centers and the widths are fixed, we can view the RBF network as a linear least square problem.

In order to minimize equation (7), any backpropagation algorithm can be used. In the present work, the BFGS-BP training algorithm was applied. The BFGS-BP algorithm is a standard numerical optimization technique, designed to approximate the Hessian matrix. The k-th iteration of BFGS-BP algorithm is as follows [8]:

$$w_{k+1} = w_k + \alpha_k d_k \,, \tag{10}$$

where α_k is the step size and d_k is the search direction which has the following form:

$$d_k = -B_k^{-1} g_k \,, \tag{11}$$

where B_k, a symmetric positive definite matrix, is an approximation to the Hes-

sian matrix that is updated at every iteration by means of a low-rank formula.

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, \tag{12}$$

where

$$s_k = w_{k+1} - w_k, \quad y_k = g_{k+1} - g_k.$$

The updating matrix B_k is required to satisfy the usual quasi-Newton equation

$$B_{k+1} s_k = y_k \tag{13}$$

so that B_{k+1} is a reasonable approximation to H_{k+1}.

To calculate the step size α_k, instead of the exact line search, we can obtain it as analytically given by:

$$\alpha_k = -\frac{d_k^T g_k}{d_k^T H_k d_k}, \tag{14}$$

where H_k is the Hessian matrix of the error function $\mathcal{F}(w)$ at w_k.

4. Training Process

During the neural network training, we will use two subsets of data points. The first subset is the training set which is used for computing the gradient and updating the network weights. The second subset is the validation set. The error on the validation set is monitored during the training process, but not used to adjust the weights.

$$E(p^*) = \sum_{j=1}^{n_2} \Big[\Big(f_{p^*} - \lambda K f_{p^*} - g \Big)(x_j) \Big]^2, \tag{15}$$

where n_2 is the number of the validation data and p^* refers to the parameters of the network which are initiated at current iteration. The validation error will normally decrease during the phase of training, as does the training set error. However, when the network begins to over-fit the data, the error on the validation set will typically begin to rise. If the validation error increases for a specified number of iterations, the training is stopped and the weights at the minimum of the validation error are returned.

The above-mentioned procedure, briefly, suggests the following algorithm with the number of hidden neurons as small as possible [6], which its implementation is very simple. First choose n_c centers, as a preselected set for hidden nodes, and then calculate the values of the shape parameters. Let training points $X = \{x_1, x_2, \ldots, x_{n_1}\}$ and a tolerance ε be given. Set $m = m_0$ from

preselected nodes at the beginning and choose one pattern randomly from X_1 and do Steps 1-3.

Step 1. Apply BFGS-BP optimization technique to determine the weights of the network in such a way that $E(p^*)$ is minimized.

Step 2. If $n < n_p$ and $E(p^*) > \varepsilon$, choose another pattern from X, similar to the relation (8), and set $n = n + 1$ and return to Step 1; otherwise stop the algorithm.

Step 3. If the algorithm cannot cause any decreasing in $E(p^*)$, then try to insert a new unselected node according to (8) and initiate the weight of the new node with 0. return to Step 1 for training the whole weights of the output layer.

The main attraction of this algorithm is that it starts with a single training data and with a single hidden layer node. Then we continue the training process by incorporating more training patterns and more hidden nodes one by one and allow the network to grow.

5. Numerical Results

We now report on the numerical results of some test problems solved by the proposed method of this article [2]. For testing, 1000 patterns are used to evaluate the root mean square error defined as

$$N_e = \sqrt{\frac{\sum_{j=1}^{n_t}(f(x_j) - f_p(x_j))^2}{n_t}},$$

where $f(x_j)$ and $f_p(x_j)$ are the exact and calculated function values at the point x_j, respectively, and n_t is the number of test data.

Example 1. Consider the following integral equation

$$f(x) - \int_0^x \frac{f(t)}{\sqrt{x-t}}dt = x - \frac{4}{3}x^{\frac{3}{2}}, \quad x \in [0,1],$$

with the exact solution, $f(x) = x$. The error is $N_e = 7.90371 \times 10^{-6}$ by using $m = 5$ Gaussian functions.

Example 2. Consider the following integro-differential equation

$$\frac{df}{dx} - \int_0^x \frac{f(t)}{\sqrt{x-t}}dt = -\frac{1}{w^2} - \frac{1}{\sqrt{w}} \ln \frac{\sqrt{w}+\sqrt{x}}{\sqrt{w}-\sqrt{x}}, \quad x \in [0,1],$$

with the exact solution, $f(x) = \frac{1}{w}$, where $w = x + 1$. The error is $N_e = 2.83643 \times 10^{-5}$ by using $m = 7$ Gaussian functions.

6. Conclusion

Our results show that the weakly singular integral equations can be approximately solved by RBF network with parameters (weights) optimized by a learning method. In the above experiments, the type and the number of radial basis functions, position of the centers and particularly the value of widths have major effect on results. Moreover, our method is quite general and can be used in a wide class of integral and differential equations.

References

[1] D.S. Broomhead, D. Lowe, Multivariate functional interpolation and adaptive networks, *Complex Systems*, **2** (1988), 321-355.

[2] E.A. Galperin, E.J. Kansa, Application of global optimization and radial basis functions to numerical solutions of weakly singular Volterra integral equations, *Computers and Mathematics with Applications*, **43** (2002), 491-499.

[3] A. Golbabai, S. Seifollahi, Numerical solution of the second kind integral equations using radial basis function networks, *Appl. Math. Comput.*, **174** (2006), 877-883.

[4] Simon Haykin, *Neural Networks*, Prentice-Hall, Inc. (1999).

[5] L. Jianyu, L. Siwei, Q. Yingjian, H. Yaping, Interactive gradient algorithm for radial basis function networks, In: *The Sixth International Conference on Signal Processing* (2002), 1187-1190.

[6] 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.

[7] Derong Liu, Tsu-Shuan Chang, Yi Zhang, A Constractive algorithm for feedforward neural networks with incremental training, *IEEE*, **49** (2002).

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

[9] J. Park, I.W. Sandberg, Universal approximation using radial basis function networks, *Neural Computation*, **3** (1991), 246-257.