Approximation of Function by Adaptively Growing Radial Basis Function Neural Networks

Li,Jianyu^{1,2}, Luo,Siwei¹, Qi,Yingjian¹

¹ Computer Science Dept., Northern Jiaotong University, Beijing, P.R. China 100044

² Information and Engineering College, Beijing Broadcasting Institute, Beijing, P.R. China

Abstract. In this paper a neural network for approximating function is described. The activation functions of the hidden nodes are the Radial Basis Functions (RBF) whose parameters are learnt by a two-stage gradient descent strategy. A new growing radial basis functions-node insertion strategy with different radial basis functions is used in order to improve the net performances. The learning strategy is able to save computational time and memory space because of the selective growing of nodes whose activation functions consist of different radial basis functions. An analysis of the learning capabilities and a comparison of the net performances with other approaches have been performed. It is shown that the resulting network improves the approximation results.

Keywords: Radial Basis Function Neural Networks; Sum Squared Error; Mean Squared Error; Multiquadrics

1. Introduction

Some of the common types of RBF used in this paper are as follows.

- 1. Linear function: $\phi^{(k)} = 1 + r_k$.
- 2. Duchon radial cubics: $\phi^{(k)} = r_k^3$.
- 3. Radial quadratic plus cubic $\phi^{(k)} = 1 + r_k^2 + r_k^3$.
- 4. Thin plate splines: $\phi^{(k)} = r_k^2 \ln r_k$
- 5. Hardy multiquadrics: $\phi^{(k)} = (r_k^2 + C^2)^{n/2}$ with C being a user specified constant and *n* being a positive integer.
- 6. Inverse multiquadrics: $\phi^{(k)} = 1/\sqrt{(r_k^2 + C^2)}$
- 7. Gaussian: $\phi^{(k)} = \exp(-r_k^2 / C^2)$

where the Euclidean distance function r_k is defined in 2D as:

 $r_{k} = r_{k}(x, y; x^{(k)}, y^{(k)}) = \sqrt{(x - x^{(k)})^{2} + (y - y^{(k)})^{2}}$

In the last ten years, Researches on the RBFs have become very attractive and several different strategies have been proposed. However, all these strategies have a same weakness: the information of all the parameters is not made good use of and the structure of the RBF neural network is not optimal.

A more recent strategy uses a growing-network paradigm in which the number of hidden units is not give a priori and a new hidden node is inserted only after a certain criterion has been met (Borghese,N.A. and Ferrari,S. [1]; Vinod,V. and Ghose, S. [2]; A. Esposito, M. Marinaro, D. Oricchio, S. Scarpetta [3]). In their approaches, part of the parameters (the centers, the variances or the weights) of the added nodes are also fixed through heuristic considerations, and only the variances/weights of the radial basis functions are trained through a two-stage learning strategy, which includes a local optimization of the variances/weights of each added neuron followed by a global

optimization of the variances/weights of all the neurons.

The approach reported in the present paper still uses a growing-network paradigm. Our main goal is to use only as few nodes as necessary (selected according to some quality criterion) to achieve a desired accuracy for the fit. It includes two more algorithms: one for adapting the networks parameters and another for inserting new neurons adaptively. The main differences from other approaches are that all parameters are trained and the activation function of the added nodes can be one of different radial basis functions.

For the parameter adaptation we follow the idea of two-step iterative gradient method (Li Jianyu, Luo Siwei, Qi Yingjian [4]). In this sense, the parameters of centers and width and the parameters of the output weights are trained respectively and interactively. New neurons are inserted when the global error doesn't decrease any more by adjusting the parameters. The whole learning procedure to obtain the appropriate parameter values also consists of two steps. The former is a local tuning, i.e. only the parameters of the new node are learnt. The latter is a fine tuning of all the parameters of the radial basis functions in order to further minimize the global error.

The paper is organized as follows. We formulate the two-stage gradient learning algorithm in section 2, and specializes the discussion to adaptive least squares approximation using node insertion in section 3. Section 4 and section 5 report the net performance on several functions and the comparisons of results obtained to those already reported in literature. We close the paper with conclusions.

2. Two-stage gradient learning algorithm

In order to express the algorithm clearly and concisely, we will only use the multiquadrics (MQ) as the RBFs.

The chosen MQ is given by

$$b^{(i)}(||x - c^{(i)}||) = \sqrt{(r^2 + a^{(i)2})}$$
(1)

where $a^{(i)} > 0$, and the parameters $c^{(i)}$, $a^{(i)}$ are the *i* th RBFs' center and width. Let us consider an RBF-network as a function approximator:

$$y(x) = \sum_{i=1}^{m} w^{(i)} \phi^{(i)}(x) = \sum_{i=1}^{m} w^{(i)} \sqrt{(r^2 + a^{(i)^2})}$$
(2)

where m is the number of the radial basis functions (the hidden neurons).

We train the network to approximate an unknown function given a (possibly noisy) training set $TS = \{ (x^{(j)}, f(x^{(j)})) | 1 \le j \le n \}$, where f(x) is the unknown function are to be approximated.

Given the number of basis functions, we can train the whole network by minimizing the sum squared error (SSE) or the mean squared error (MSE)

$$SSE = l(c^{(i)}, a^{(i)}, w^{(i)}) = \sum_{j=1}^{n} (y(x^{(j)}) - u(x^{(j)}))^{2}$$
(3)

$$MSE = \frac{1}{n}SSE = \frac{1}{n}\sum_{j=1}^{n} (y(x^{(j)}) - u(x^{(j)}))^{2}$$
(4)

with respect to the parameters $c^{(l)}, a^{(l)}, w^{(l)}$.

The algorithm is a gradient descent type which is used in two stages, once to

optimize the weights $w^{(i)}$ and once to optimize the centers and the widths $c^{(i)}$, $a^{(i)}$ as follows:

Step 1 fix
$$c^{(i)}, a^{(i)}$$
, the purpose is to find $w^{(i)}$ by minimizing the SSE by the following formula $w^{(i)} = w^{(i)} - n = \frac{\partial l(c_{t-1}^{(i)}, a_{t-1}^{(i)}, w_{t-1}^{(i)})}{\partial l(c_{t-1}^{(i)}, a_{t-1}^{(i)}, w_{t-1}^{(i)})}$ (5)

following formula $w_t^{(i)} = w_{t-1}^{(i)} - \eta_{t-1} \frac{\partial l(c_{t-1}, u_{t-1}, w_{t-1})}{\partial w_{t-1}^{(i)}}$ (5) Step 2 fix $w_t^{(i)}$, the purpose is to find $c^{(i)}$, $a^{(i)}$ that minimizes the SSE by

$$\begin{array}{ccc} p & 2 & \text{fix } W_t^{(i)}, \text{ the purpose is to find } \mathcal{C}^{(i)}, a^{(i)} \text{ that minimizes the SSE} \end{array}$$

$$c_{t}^{(i)} = c_{t-1}^{(i)} - \beta_{t-1} \frac{\partial t(c_{t-1}, u_{t-1}, w_{t-1})}{\partial c_{t-1}^{(i)}}, \qquad (6)$$

$$a_{t}^{(i)} = a_{t-1}^{(i)} - \alpha_{t-1} \frac{\partial l(c_{t}^{(i)}, a_{t-1}^{(i)}, w_{t}^{(i)})}{\partial a_{t-1}^{(i)}}, \qquad (7)$$

where $\eta_{t-1}, \beta_{t-1}, \alpha_{t-1}$ are the learning rates at time t-1. The above two steps are carried on in turn. We call it interactive gradient algorithm.

It has been known the learning rates are very important for the convergence of the network parameters. If it is small, the convergence is slightly; if it is large, the parameters oscillate and don't converge. But how to choose the best learning rates is problem-dependent. η_{t-1} at time of t-1 is determined by minimizing

$$SSE = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} \left(w_{t-1}^{(i)} - \eta_{t-1} \frac{\partial l}{\partial w_{t-1}^{(i)}} \right) \phi^{(i)}(x^{(j)}) - f(x^{(j)})^2 \right)$$
(8)

and

$$\eta_{t-1} = \left(\sum_{j=1}^{n} \left(\sum_{i=1}^{m} w_{t-1}^{(i)} \phi^{(i)}(x^{(j)}) - f(x^{(j)})\right) \left(\sum_{i=1}^{m} \frac{\partial l}{\partial w_{t-1}^{(i)}} \phi^{(i)}(x^{(j)})\right)\right) \left/\sum_{j=1}^{n} \left(\sum_{i=1}^{m} \frac{\partial l}{\partial w_{t-1}^{(i)}} \phi^{(i)}(x^{(j)})\right)^{2} \right)$$
(9)

 $\beta_{t-1}, \alpha_{t-1}$ can be decided by a recurrent procedure. For example, for some positive number a, set $\beta_{t-1} = a$, the proper learning rate β_{t-1} is determined by the following formula:

$$\beta_{t-1} = \begin{cases} \frac{a}{10}, \text{ if } l(c_t^{(i)}, a_t^{(i)}, w_t^{(i)}) > l(c_{t-1}^{(i)}, a_{t-1}^{(i)}, w_t^{(i)}) \\ 10a, \text{ if } l(c_t^{(i)}, a_t^{(i)}, w_t^{(i)}) < l(c_{t-1}^{(i)}, a_{t-1}^{(i)}, w_t^{(i)}) \end{cases}$$
(10)

There are two features in this algorithm compared with the conventional gradient descent algorithm: (a) the learning rates are decided properly, (b) in Eqs. (2) and (3), $c_t^{(i)}$ is determined by $c_{t-1}^{(i)}, a_{t-1}^{(i)}, w_t^{(i)}$, not by $c_{t-1}^{(i)}, a_{t-1}^{(i)}, w_{t-1}^{(i)}; a_t^{(i)}$ is determined by $c_t^{(i)}, a_{t-1}^{(i)}, w_t^{(i)}$, not by $c_{t-1}^{(i)}, a_{t-1}^{(i)}, w_{t-1}^{(i)}; a_t^{(i)}$ is determined by $c_t^{(i)}, a_{t-1}^{(i)}, w_t^{(i)}$.

3. Node Insertion

Let us assume we are given a large number of data and we want to fit them with

different radial basis functions within a given tolerance. The idea is to start with very few nodes which are a subset of the data, and then, as long as the sum square error exceeds the tolerance, repeatedly insert a node at that data location whose sum square error component is largest. Here is the algorithm in detail:

Algorithm: Node insertion

a. Let data points $X = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$, and object data $d^{(i)}, i = 1, 2, \dots, n$ and a

tolerance \mathcal{E} (TOL) be given.

- b. Choose m_0 initial nodes whose centers are some subset of X and let $m = m_0$, and calculate the sum square error.
- c. While SSE>TOL do

"weight" each data point $x^{(i)}$ according to its error component, i.e. let

$$e^{(i)} := |d^{(i)} - y(x^{(i)})|, i = 1, 2, \cdots, n.$$
(11)

d. Find the data point $x^{(\nu)} \notin N$ with maximum weight $e^{(\nu)}$ and insert it as a node, i.e.

$$N = N \bigcup \{x^{(v)}\}, m = m + 1 \text{ and } w_0^{(v)} = 0.$$
 (12)

- e. Train the parameters of the new inserted node by two-stage gradient algorithm.
- f. Train the parameters of all nodes by two-stage gradient algorithm.

4. Numerical Examples

The algorithm was tested, at the beginning, on some simple 1D continuous functions, and its approximation results were compared with the results obtained using other approximation algorithms tested on the same functions (A. Esposito, M. Marinaro, D. Oricchio, S. Scarpetta ;[3];Yong, F. and Chow, T. [5]). The functions are reported below.

$$y = 0.5 \exp(-x)\sin(6x), \qquad x \in [-1,1]$$
 (13)

$$y = \sin(2\pi x), \quad x \in [0,1]$$
 (14)

Table 1 compares our approximation results with other algorithms for (13), Using the same number of training points as used by Yong and Chow [5], and A. Esposito, M. Marinaro, D. Oricchio, S. Scarpetta [3]. The training and testing data were generated randomly in the input domain. The simulation results reported by Y.&C. and A.&E. showed that their network achieves good approximation results. For a net configuration with 25 and 45 neurons of their network, the MSE were 1.1×10^{-6} and 1.6×10^{-6} respectively. Our net gives a more accurate result by learning only 20 neurons (12 MQ and 8 Gaussian RBFs) with the MSE 9.4×10^{-7} . The plot of the testing error of (13) is also reported. It is worth noting that the net is able to achieve a very good generalization.

For the function (14), Fig.1 shows our approximation results. Fig. 1a shows the approximation error curve with 15 neurons (9 MQ and 6 Gaussian RBFs) on 50 data points. Fig. 1b displays the training sum square error as a function of the number of

neurons (up to a maximum of 30 neurons) and the errors decrease as the numbers of neurons increases.

Table 1 Comparison between the approximation results for (13) achieved by the present algorithm (L.alg.) and some approximation results reported in literature by Yong and Chow (Y.&C.), and A. Esposito, M. Marinaro, D. Oricchio, S. Scarpetta (A.&E.). The parameter considered for the comparison are: the training mean squared error (Tr. E), the testing mean squared error (Ts.E), the number of parameters (neurons) to be trained (#L. P.), the number of data points used for the training (#Tr.P.) and the number of data points used for the testing (#Ts.P.). * indicates that the corresponding value was not reported in the referred paper.



Fig. 1 Net approximation results for the function (14). 1a) approximation error curve with 15 RBFs (9 MQ and 6 Gaussian) on 50 data points. 1b) shows the relation between the SSE and the neuron numbers.

5. net performance using noisy examples

Another set of functions (reported below) is used in order to test the performance of the present algorithm when the net is trained with noisy examples.

$$f(x, y) = x^{2} + y^{2} \qquad (x, y) \in [-1, 1] \qquad (15)$$

The number of samples used to train the net for the functions reported in Eqs. (15) is 500. The noise added (thereafter indicated with n) to the function samples is randomly generated in the [-0.01, 0.01] intervals. The net is trained on the noisy examples. Using the same number of training points and neurons as used by A. Esposito, M. Marinaro, D. Oricchio, S. Scarpetta [3], the MSE 7.239×10^{-4} of the present algorithm is obviously less than their MSE 10^{-3} .

Fig. 2 shows the approximation results obtained on the function reported in Eq. (15). Figure 2a and 2b shows the noise function and the approximation error curve for the original function of Eq.(15) by using 20 RBFs (10 MQ, 10 Gaussian).

6. Conclusions

In this paper, a new incremental algorithm for growing RBF networks and a two-stage learning strategy for training the net parameters are reported. The novelty of the algorithm has different aspects. First of all, all learning procedure is accomplished by a two-stage gradient method which not only optimizes the weights of the net, but also optimizes the centers and widths of the Neurons. Moreover, the activation function of the added neuron can be different radial basis function. The results show that the strategy is better than the algorithm whose neurons are the same RBF type.

It should be pointed out that there is no mathematical theory to guarantee which RBF type should be chosen when adding a new node. The proper choice of such functions requires some skill and experience on the user's side.



Fig.2. Plots of the function reported in Eq. (15) with noisy version (when $n \in [-0.01, 0.01]$) (see (a)), of the approximation error curve achieved by the net with 20 neurons (b), The size of both the training set and testing set for all the cases considered is 500 samples.

References

[1] Borghese,N.A. and Ferrari,S., Hierarchical RBF networks and local parameter estimate, Neurocoputing, 19(1998), 1-25

[2] Vinod,V. and Ghose, S., Growing nonuniform feedforward networks for continuous mappings, Neurocoputing, 10 (1996), 55-69

[3] 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.

[4] Li Jianyu, Luo Siwei, Qi Yingjian, Interactive Gradient Algorithm for Radial Basis Function Networks, the Sixth International Conference on Signal Processing (2002),

[5]Yong, F. and Chow, T. Neural network adaptive wavelets for function approximation. Internal report (1996). Department of Electrical Engineering, City University of Hong Kong.