A Greedy Incremental Algorithm for Universal Approximation with RBF Newtorks

Xing Wu and Bogdan M. Wilamowski

Department of Electrical and Computer Engineering
Auburn University, Alabama, USA
{xzw0015,wilambm}@tigermail.auburn.edu

Abstract. Radial basis function(RBF) networks have been proved to be a universal approximator when enough hidden nodes are given and proper parameters are selected. Conventional algorithms for RBF networks training, including two-stage methods and gradient-based algorithms, cost much computation and have difficulty to determine the network size. In this paper, a new greedy incremental(GI) algorithm is proposed which constructs the RBF network by adding hidden node one by one; Each added hidden node is trained once and then fixed. The parameters of each added hidden node are trained in a greedy way to approximate the local area around the pattern with the biggest error magnitude. The center and weight are determined by local regression, the width is tuned iteratively with a simple rule. The proposed greedy incremental algorithm is tested on some practical experiments and compared with other popular algorithms. The experiments results illustrated the GI algorithm could approximate the function universally with high efficiency and robustness.

Keywords: radial basis function networks, constructive learning, greedy incremental algorithm.

1 Introduction

Because of the simple topology structure and the ability to approximate complex nonlinear mappings from the input-output data, radial basis function(RBF) network was broadly used in classification and function approximation area [1,2,3]. It has been proved to be a universal approximator for any continuous target function when sufficient hidden nodes are provided [4]. As analyzed and compared with other neural networks and fuzzy systems, RBF networks have better generalization ability and tolerance to input noiseand perform better in regular function approximation [5,6]. Based on these properties and the simple topology structure, RBF networks are widely applied for solving various industrial application problems, such as fault diagnosis [7,8] and image processing [9,10].

A typical RBF network consists of a hidden layer with non-linear RBF activation function and a linear output layer (Fig. 1). Adjustable parameters of RBF networks include hidden layer parameters (centers and widths) and output weights connecting hidden layer and output layer.

J. Fodor and R. Fullér (eds.), *Advances in Soft Computing, Intelligent Robotics and Control,* Topics in Intelligent Engineering and Informatics 8,
DOI: 10.1007/978-3-319-05945-7_9,

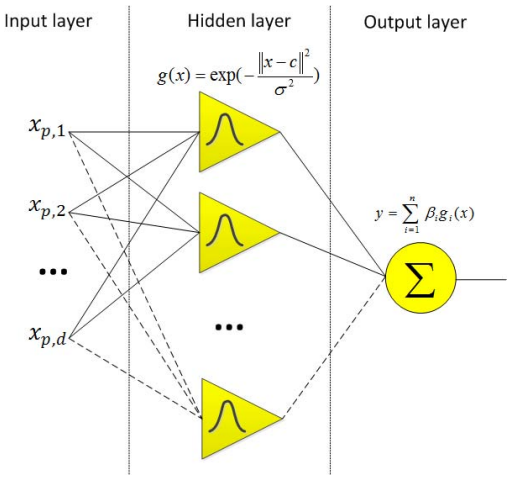

Fig. 1. Architecture of RBF network

Original approach for training RBF networks is to take all the training samples as centers and preset the width of all the hidden nodes. Only the output weights are trained with least squares regression or gradient-based methods. However, this algorithm could lead to overfitting and is also not practical for real world approximation problems. In order to achieve high accuracy with compact RBF networks, centers and widths are further selected or adjusted with different algorithms. Moody and Darken [2] used unsupervised self-organized selection to determine the centers and widths. Support vector machines [11] selected support vectors from training sets as centers. However, the search space is too restricted since the widths are fixed and centers are subset of the discrete training samples.

Alternative method is to train all the parameters simultaneously. Some gradient based optimization algorithms were proposed for RBF networks training [12,13]. To improve the slow convergence rate of first order gradient, recently, T. Xie et al. presented an improved second order(ISO) algorithm [15] based on an improved Levenber-Marquardt algorithm [14]. Though more compact network is reached with this algorithm, it costs more time due to expensive gradient computation.

While most algorithms have to specify the size of the RBF network before training, it is difficult to determine the network size. To determine the network size during training, some constructive algorithms were proposed by adding hidden node one by one, or batch by batch. T.-Y. Kwok and D.-Y. Yeung trained each added hidden node with modified Quickprop algorithm by minimizing some objective functions [21]. Huang et al. proposed a family of extreme learning machines (ELM) by adding random hidden nodes and training weight only. The algorithms were shown to be much faster than other general algorithms. However, much larger network was achieved with these ELMs.

In this paper, a new greedy incremental (GI) algorithm is proposed to construct the RBF network by adding hidden nodes one by one. Each added hidden node is trained once and then fixed. The parameters of each added hidden node are trained in a greedy way to approximate the local area around the pattern with the biggest error magnitude. The center and weight are determined by local regression; the width is tuned iteratively with a simple rule. The proposed algorithm is very simple and can construct a compact RBF network for approximation speedily.

The paper is organized as following. In Section 2, computation fundamentals of RBF networks are briefly introduced. Section 3 presents the proposed GI algorithm for the RBF networks training. Section 4 gives several practical benchmarks for function approximation. Experiment results are compared with other popular algorithms. In section 5, a brief conclusion is given and future work is introduced.

2 Computational Fundamentals

Since an approximation problem with multiple outputs can be divided into several independent approximation task with unique output, in this paper, we are focusing on the function approximation with single output. Before describing the algorithm details, several common indices and notations are introduced. Assume the algorithms discussed in this paper are all aiming at approximating a training set $\{(\mathbf{x}_p, y_p) \mid \mathbf{x}_p \in R^D, y_p \in R, p = 1, 2, \ldots, P\}$, where there are P training patterns with D-dimension input and scalar output, $(\mathbf{x}_p, y_p)$ denote the p_{th} input and output.

As shown in figure 1, a conventional RBF network has fixed architecture with three layers: an input layer, a hidden layer with RBF activation function and a linear output layer. Activation function of RBF networks can be different radial basis functions, including multiquadric, inverse quadratic function, etc. In this paper, we used the popular gaussian function as activation function of the RBF network.

$$g(x) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{\sigma^2}\right), \qquad \mathbf{x}, \mathbf{c} \in R^D, \sigma \in R^+ \tag{1}$$

in which, $\|\cdot\|$ represents Euclidean distance, $\mathbf{c}, \sigma$ are center and width of the RBF node.

The output of the RBF network is calculated by summing multiplication of hidden layer and output weights. The output of a RBF network with n hidden nodes can be described as,

$$y = \sum_{i=1}^{n} \beta_i g_i(\mathbf{x}) \tag{2}$$

in which, β_i is the weight connecting the i_{th} hidden node and the output node.

3 The Greedy Incremental Algorithm

In this section, the proposed Greedy Incremental(GI) algorithm is introduced.

3.1 Constructive algorithm

The proposed GI algorithm is a constructive algorithm for RBF network training. The constructive algorithm is a general method for a single layer feedforward network (SLFN) with any kernels. One advantage of constructive algorithms is that the network size can be determined while training. The algorithm starts from a SLFN with zero hidden neuron and constructs the SLFN by adding hidden nodes one by one. Each hidden node is trained supervised according to errors of previous SLFN and then fixed. Thus the training of whole SLFN is simplified into a sequence of single neuron training.

Assume the current SLFN has n hidden nodes, the errors of current SLFN are $\mathbf{E} = [e_1, e_2, \ldots, e_P]^T$. To approximate the errors with the new added neuron $g_{n+1}(\alpha, \mathbf{x})$, one is trying to tune its parameters α and output weight β to minimize the sum squared error(SSE).

$$S(\alpha, \beta_{n+1}) = \sum_{p=1}^{P} (e_p - \beta_{n+1} g_{n+1}(\alpha, \mathbf{x}_p))^2 \tag{3}$$

in which, $S(\cdot)$ is the SSE, α are parameters of the neurons to be tuned.

In order to minimize the objective function SSE, the neuron's parameters α and its output weight β can be optimized independently. From equation (3), one can observe that SSE is a convex function of output weight β, it achieves its minima while α is fixed and,

$$\beta_{n+1} = \frac{\sum_{p=1}^{P} e_p g_{n+1}(\mathbf{x}_p)}{\sum_{p=1}^{P} g_{n+1}^2(\mathbf{x}_p)}. \tag{4}$$

For other parameters (α) tuning, T.-Y. Kwok and D.-Y. Yeung [21] proposed a modified Quickprop algorithm to minimize the objective function. However, gradient computatoin still cost much time. Huang et al. presented an incremental extreme learning machine (I-ELM) [17] which generated random parameters for the single neuron and only determined output weight using (4). Though it is very fast, the I-ELM algorithm also resulted in a very large SLFN. Taking advantage of local property of RBF node, the proposed GI algorithm tuned the new added node in a greedy way to approximate a local area around the pattern with biggest error magnitude. The parameters of the added node and the output weight are all tuned in simple method.

3.2 Center and Weight

The proposed GI algorithm is a greedy process. In order to make each added new hidden node contribute most to the network, each time the new node was attempt to approximate the local area around the pattern with biggest error magnitude. The center and weight of the new node can be determined by simple regression with the local training sets.

Filter the local sets. While approximating the previous RBF network's residual error $\mathbf{E} = [e_1, e_2, \ldots, e_P]^T$ with a single RBF node, it is not necessary to consider all the training patterns. Because of the local property of gaussian function, one only need consider a local area in the input space for the single RBF training. The GI algorithm in this paper focuses on the local area around the pattern with biggest residual error magnitude. To filter these local sets, one can preset parameters K and τ and do the following process:

1. find the pattern with biggest error magnitude $|e_k|$, note the pattern as $A(\mathbf{x}_A, e_A)$;
2. find the subset(T) of the training set, whose elements are K nearest neighbors of A.
3. In subset T, filter a subset S, whose residual errors are in the range $(\tau e_A, e_A)$.

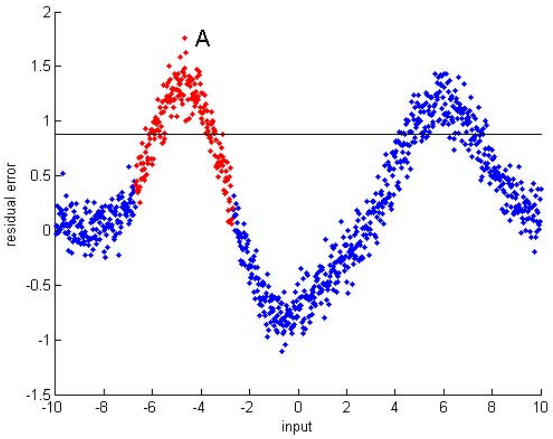

Fig. 2. A 1 dimension example of filtering the local set, $K = 200, \tau = 0.5$. (1) Find point with biggest error magitude A. (2) Filter K nearest neighbors of max point A.(red points) (3) Among red points, filter the points with residual error bigger than τe_A. (above the horizon line)

Local Regression. After above 3 steps, local set around the max point is filtered as S, assume the indices of S are $\{j_1, j_2, \ldots, j_S\}$. Using these patterns, we can determine optimal center and weight for the new node directly with local regression.

In order to approximate the previous RBF network's residual error $\mathbf{E}_S = [e_{j_1}, e_{j_2}, ..., e_{j_S}]^T$ with (1) multiplied by its output weight β_{n+1}, we are actually solving the following equations,

$$\beta_{n+1} \exp\left(-\frac{\|\mathbf{x}_p - \mathbf{c}\|^2}{\sigma^2}\right) = e_p, \qquad p = j_1, j_2, \ldots, j_S \tag{5}$$

Since the residual errors of local set S are in the range $(\tau e_A, e_A)$, which means they are with the same sign, β_{n+1} should also be the same sign. So for each pattern in the local set S, we can derive (5) as,

$$-\frac{1}{\sigma_2}\sum_{d=1}^{D} x_{p,d}^2 + \frac{2}{\sigma^2}\sum_{d=1}^{D} c_d x_{p,d} - \frac{1}{\sigma^2}\sum_{d=1}^{D} c_d^2 + \ln|\beta_{n+1}| = \ln|e_p| \tag{6}$$

in which, $x_{p,d}$ denotes the d^{th} dimension of the p^{th} pattern, c_d is the d^{th} dimension of the center $\mathbf{c}$. The equation is actually a standard linear regression format,

$$\mathbf{X}\mathbf{w} = \mathbf{b}, \tag{7}$$

in which

$$\mathbf{X} = \begin{bmatrix} \sum_{d=1}^{D} x_{j_1,d}^2 & x_{j_1,1} & x_{j_1,2} & \cdots & 1 \\ \sum_{d=1}^{D} x_{j_2,d}^2 & x_{j_2,1} & x_{j_2,2} & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & 1 \\ \sum_{d=1}^{D} x_{j_S,d}^2 & x_{j_S,1} & x_{j_S,2} & \cdots & 1 \end{bmatrix} \tag{8}$$

$$\mathbf{w} = \left[-\frac{1}{\sigma^2},\ 2\frac{c_1}{\sigma^2},\ 2\frac{c_2}{\sigma^2},\ \ldots,\ 2\frac{c_D}{\sigma^2},\ -\frac{1}{\sigma^2}\sum_{d=1}^{D} c_d^2 + \ln|\beta| \right]^T \tag{9}$$

$$\mathbf{b} = [\ln|e_{j_1}|,\ \ln|e_{j_2}|,\ \ldots,\ \ln|e_{j_S}|]^T \tag{10}$$

For the linear regression (7), we can easily get the optimal $\mathbf{w}$,

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{b} \tag{11}$$

Combined with (9), we can determine all the parameters of the new hidden nodes (center $\mathbf{c}$, width σ and output weight β). However, because the above approximation only used filtered local set S, global convergence of the entire RBF network is not guaranteed. On the other hand, for a gaussian function, parameters determining peak location (center and height) are local parameters while the width is more related to its global character. So the proposed GI algorithm used the center $\mathbf{c}$ and output weight β from above computation and tunes the width in another simple rule.

3.3 Width

As center and weight are determined by local regression, width of the added hidden node is also necessary to be tuned. As presented by N. Benoudjit et al. [22], widths play an important role in the RBF networks approximation. The fast extreme learning machine was also shown to be improved significantly by

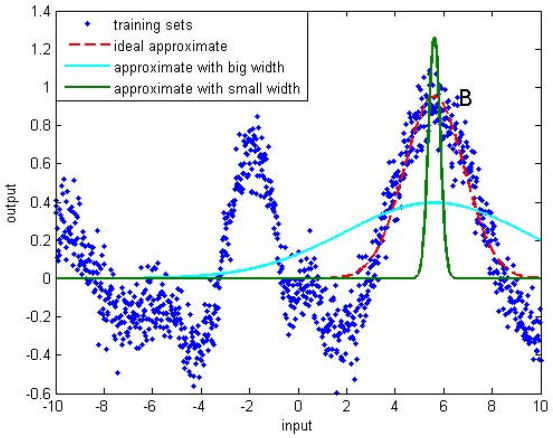

Fig. 3. Approximation result comparison of a 1-dimension example while using different width. With the optimal center determined in previous section, all the three cases use (4) to determine output weight. The red line used the ideal width; the green line selected a small width; the cyan line picked a big width.

tuning width [20]. The proposed GI algorithm tunes the width of the new added RBF node using an efficient iterative rule.

As mentioned in section 3.1, output weight is quite easy to determine by (4) once center and width are fixed. Since sum squared error (SSE) shown in (3) is a convex function of weight (β_{n+1}), the optimal weight can also guarantee SSE to be decreasing while adding more hidden nodes. That means, formula (4) could guarantee the constructive algorithm's global convergence. However, given a bad width, approximation result is still far from optimal. Fig. 3 shows a 1-dimension example.

From the figure, one can observe that the optimal approximation results from a proper width which makes the weight calculated by (4) match the optimal weight we get by local regression (point B in the figure). Though the three cases' approximation results look very different, the area under them are similar. In fact, to minimize SSE, a thinner gaussian with a bigger height would always be better than a thinner gaussian with a smaller height; a fatter gaussian with a smaller height would always be better than the one with a bigger height. So the proposed GI algorithm approaches a start width to optimal by equaling its area to the new gaussian with a new width and our optimal weight.

Lemma 1. *The area or integration of a D-dimension gaussian function with width σ, output weight β is,*

$$\text{Area} = \sqrt{\pi^D}\beta\sigma^D \tag{12}$$

Assume in previous section, we get the optimal center $\mathbf{c}_{\text{opt}}$ and weight β_{opt}. Given the width in the t^{th} iteration σ_t, the weight calculated from (4)

using σ_t notes as β_t, for the $(t+1)^{\text{th}}$ iteration, we equal the new gaussian's area whose $\{\text{width}, \text{weight}\} = \{\sigma_{t+1}, \beta_{\text{opt}}\}$ to the old gaussian's area whose $\{\text{width}, \text{weight}\} = \{\sigma_t, \beta_t\}$,

$$\sqrt{\pi^D}\beta_{\text{opt}}\sigma_{t+1}^D = \sqrt{\pi^D}\beta_t\sigma_t^D \tag{13}$$

from which, we can get the update rule of width,

$$\sigma_{t+1} = \sqrt[D]{\frac{\beta_t}{\beta_{\text{opt}}}}\sigma_t \tag{14}$$

While updating width of the added RBF node with (4) (14), it is approaching to the optimal one.

3.4 Pseudo Code

Since all the parameters (center, width, weight) are trained in simple process, the proposed GI algorithm could approximate functions very efficiently. The pseudo code of the whole training process can be seen below.

```
Given a D-dimension training set with P patterns {X,Y}.
Desired error is d, maximum number of RBF nodes is N.
Initialize n = 0.

while n<N and SSE>d
     1. n = n + 1
     2. pick max point A
     3. filter local set around A as S
     4. use points in S following (7)-(11) do regression, get
         optimal center(c) and weight(ww)
     5. fix center as c, initialize width as sgm0, calculate new
         weight ww0 with (4), set a threshold th
         while |ww0-ww|>th
               (1) update new width sgm1 with (14)
               (2) use sgm1 to calculate weight ww1 with (4)
               (3) sgm0 = sgm1
               (4) ww0 = ww1
         endwhile     // optimal width is sgm0
     6. calculate output y of the new RBF node
          (center=c, width=sgm0, weight=ww)
     7. update error: err = err - y, calculate SSE
endwhile
```

4 Experiments

In this section, several highly nonlinear functions with noise are given to test the efficiency of the proposed GI algorithm. The experiments results are compared

with other popular RBF algorithms, including support vector regression (SVR) [11], extreme learning machines (ELM) [17,18,19].

The testing environment consists of: Windows 7 Enterprise 64-bit operating system, Intel® Core™ 2 Quad CPU Q8400 2.67GHz processor, 4.00GB RAM, MATLAB R2012a platform.

4.1 Peaks Function

Peaks function is a popular 2-dimension nonlinear benchmark for approximation test. In this paper, we used normalized format of peaks function (15). Fig. 4 shows the mesh plot of the peaks function.

$$\begin{aligned} z =& (0.3 + 1.8x + 2.7x^2)\exp(-1 - 6y - 9x^2 - 9y^2) - \\ & (0.6x - 27x^3 - 243y^5)\exp(-9x^2 - 9y^2) - \\ & \frac{1}{30}\exp(-1 - 6x - 9x^2 - 9y^2) \end{aligned} \tag{15}$$

In the experiment, 2000 points were generated randomly in the range $[-1, 1]$ as training sets and another 1000 points generated in the same way as testing sets. All the patterns were added a guassian noise with variance Var = 0.01. The RBF network was trained with the given training set by the GI algorithm and some other popular RBF algorithms, including extreme learning machines (ELM) and support vector machine (SVM) for regression.

For the proposed GI algorithm, we set parameters $K = 200$, $\tau = 0.5$ for filtering local set, threshold $th = 0.01$ for width tuning. We used four versions of extreme learning machines: batch Extreme Learning Machine (ELM) [16] and try different architecture, Incremental Extreme Learning Machine (I-ELM)

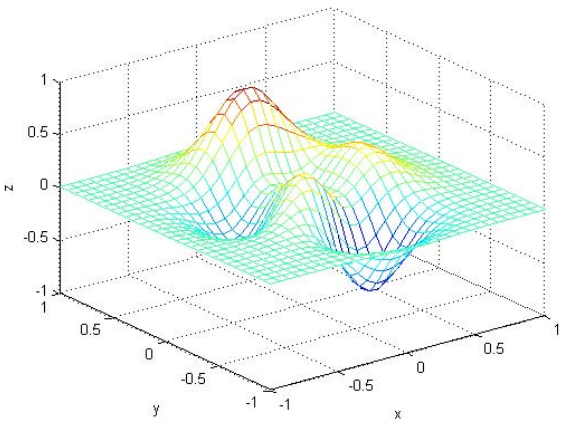

Fig. 4. Peaks function

Table 1. Comparison of RBF algorithms while approximating peaks function

Algorithm	GI	ELM	I-ELM	EI-ELM	CI-ELM	SVR
training error (RMSE)	0.114	0.1117	0.1966	0.1916	0.1963	0.1142
testing error (RMSE)	0.109	0.1089	0.1912	0.2116	0.1908	0.1107
training time (s)	0.0622	0.801	0.094	0.9206	0.1205	193.7296
# hidden nodes	10	48	200	200	200	764

[17], Enhanced Incremental Extreme Learning Machine (EI-ELM) [18], Convex Incremental Extreme Learning Machine (CI-ELM) [19]. All the ELMs used RBF kernel with following format,

$$g(x) = \exp(-\lambda \|\mathbf{x} - \mathbf{c}\|^2), \qquad \mathbf{x}, \mathbf{c} \in R^D, \lambda \in R^+ \tag{16}$$

in which, λ is called impact factor. All the ELMs generated random centers in the range $[-1, 1]$, random impact factors in the range $(0, 0.5]$. For batch ELM, we added hidden node one by one from zero and each time did the pseudo inverse. The EI-ELM algorithm used parameter $k = 20$.

The paper used LIBSVM [26] to train support vector machine for peaks approximation. Parameters of SVR (penalty C, impact factor γ) are grid searched where $C \in \{1, 10, 100, 1000\}$, $\gamma \in \{0.001, 0.01, 0.1, 1\}$. The optimal option was $C = 1000$, $\gamma = 1$, whose result was shown in Table 1. From the comparison table, one can see that the proposed GI algorithm worked very efficient to construct a compact RBF network.

4.2 Control Robot Arm Kinematics

The kinematics problem is a classic industrial application of function approximation [27]. The purpose of this problem is to simulate the movement of robot's end effectors and locate the position when joint angles change. Fig. 5 shows a 2-link planar manipulator.

From the figure, one can calculate the coordinates of the end effector by the following formula,

$$x = L_1 cos(\alpha) + L_2 cos(\alpha + \beta) \tag{17}$$

$$y = L_1 sin(\alpha) + L_2 sin(\alpha + \beta) \tag{18}$$

In this paper, we fix the two arm lengths (L_1, L_2) to be 1 and just give the approximation of the x-coordinate. The y-coordinate can be approximated in a same way. The surface of x-coordinate is shown in fig. 6.

We generated 40×40 points uniformly in the range $[-\pi, \pi]$ as training set. Another 1000 points were generated randomly in the same range for testing. Both training and testing sets were added white gaussian noise with variance Var = 0.01. All the algorithms were used in a similar setting to construct an RBF network for the approximation task. The comparison results are shown in Table 2. This experiment also illustrated the efficiency of the proposed GI algorithm.

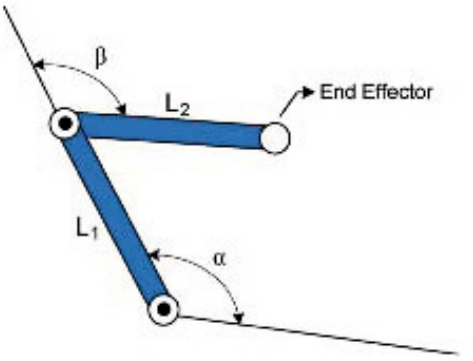

Fig. 5. The 2-link planar manipulator

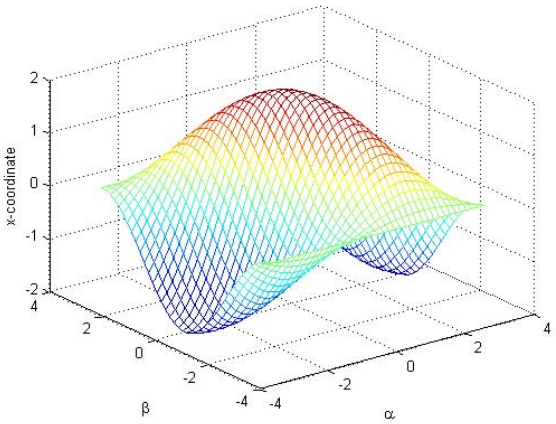

Fig. 6. Mesh plot of x-coordinate versus the two angles α and β

Table 2. Comparison of RBF algorithms while solving kinematics problem

Algorithm	GI	ELM	I-ELM	EI-ELM	CI-ELM	SVR
training error (RMSE)	0.2152	0.1245	0.4925	0.3065	0.5537	0.0796
testing error (RMSE)	0.2074	0.1275	0.4883	0.9984	0.5516	0.1233
training time (s)	0.1581	0.3141	0.1035	0.7888	0.1035	498.0173
# hidden nodes	6	33	200	200	200	738

5 Conclusion

In this paper, a simple greedy incremental(GI) algorithm was proposed for RBF network construction. The proposed algorithm is similar to other constructive algorithm: adding hidden node one by one, each added node is trained once and then fixed. The training of each added hidden node is divided into 2 steps:

1. filter the training set into a local set S around the pattern with biggest error magnitude and do local regression to determine optimal center and weight;
2. tune width iteratively using a simple update rule.

Several highly nonlinear practical experiments were given and presented the efficiency of the proposed GI algorithm, by comparing with other popular RBF algorithms.

Though the GI algorithm could achieve a compact RBF network efficiently, a big disadvantage exists. The mechanism of the construcitve algorithm that training each hidden node once and then fixed is not quite reasonable. The GI algorithm alone can not achieve high accuracy, either. So further research will focus on the fine-tuning of the RBF network after training with GI algorithm.

References

1. Broomhead, D.S., Lowe, D.: Multivariable functional interpolationand adaptive networks. Complex Syst. 2, 321–355 (1988)
2. Moody, J., Darken, C.J.: Fast learning in networks of locally-tuned processing units. Neural Comput. 1(2), 281–294 (1989)
3. Chen, S., Cowan, C.F.N., Grant, P.M.: Orthogonal least squares algorithm for radial basis funtion networks. Int. J. Control 2(2), 302–309 (1991)
4. Park, J., Sandberg, I.: Universal approximation using radial-basis function networks. Neural Comput. 3, 246–257 (1991)
5. Yu, H., Xie, T.T., Paszczynski, S., Wilamowski, B.M.: Advantages of radial basis function networks for dynamic system design. IEEE Trans. Ind. Electron. 58(12), 5438–5450 (2011)
6. Xie, T.T., Yu, H., Wilamowski, B.M.: Comparison of traditional neural networks and radial basis function networks. In: Proc. 20th IEEE Int. Symp. Ind. Electron., ISIE 2011, Gdansk, Poland, June 27-30, pp. 1194–1199 (2011)
7. Meng, K., Dong, Z.Y., Wang, D.H., Wong, K.P.: A self-adaptive RBF neural network classifier for transformer fault analysis. IEEE Trans. Power Syst. 25(3), 1350–1360 (2010)
8. Huang, S., Tan, K.K.: Fault detection and diagnosis based on modeling and estimation methods. IEEE Trans. Neural Netw. 20(5), 872–881 (2009)
9. Lee, Y.J., Yoon, J.: Nonlinear image upsampling method based on radial basis function interpolation. IEEE Trans. Image Process. 19(10), 2682–2692 (2010)
10. Ferrari, S., Bellocchio, F., Piuri, V., Borghese, N.A.: A hierarchical RBF online learning algorithm for real-time 3-D scanner. IEEE Trans. Neural Netw. 21(2), 275–285 (2010)
11. Vapnik, V.N.: Statistical Learning Theory, 1st edn. Wiley Interscience (1998)

12. Chng, E.S., Chen, S., Mulgrew, B.: Gradient radial basis function networks for nonlinear and nonstationary time series prediction. IEEE Trans. Neural Netw. 7(1), 190–194 (1996)
13. Karayiannis, N.B.: Reformulated radial basis neural networks trained by gradient descent. IEEE Trans. Neural Netw. 10(3), 657–671 (2002)
14. Wilamowski, B.M., Yu, H.: Improved computation for Levenberg–Marquardt training. IEEE Trans. Neural Netw. 21(6), 930–937 (2010)
15. Xie, T., Yu, H., Hewlett, J., Rozycki, P., Wilamowski, B.M.: Fast and efficient second-order method for training radial basis function networks. IEEE Trans. Neural Netw. Learn. Syst. 23(4), 609–619 (2012)
16. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: Theory and applications. Neurocomputing 70(1-3), 489–501 (2006)
17. Huang, G.B., Chen, L., Siew, C.K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Netw. 17(4), 879–892 (2006)
18. Huang, G.B., Chen, L.: Enhanced random search based incremental extreme learning machine. Neurocomputing 71(16-18), 3460–3468 (2008)
19. Huang, G.B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing 70, 3056–3062 (2007)
20. Reiner, P., Wilamowski, B.M.: Nelder-Mead Enhanced Extreme Learning Machine. In: INES 2013, Costa Rica, June 19-21 (2013)
21. Kwok, T.-Y., Yeung, D.-Y.: Objective functions for training new hidden units in constructive neural networks. IEEE Trans. Neural Netw. 8(5), 1131–1148 (1997)
22. Benoudjit, N., Verleysen, M.: On the Kernel widths in radial-basis function networks. Neural Process. Lett. 18, 139–154 (2003)
23. Huang, G.B., Saratchandran, P., Sundararajan, N.: An Efficient Sequential Learning Algorithm for Growing and Pruning RBF (GAP-RBF) Networks. IEEE Trans. on System, Man, and Cybernetics, Part B 34(6), 2284–2292 (2004)
24. Rousseeuw, P., Leroy, A.: Robust Regression and Outlier Detection, 3rd edn. John Wiley & Sons (1996)
25. Blake, C., Merz, C.: UCI repository of machine learning databases. Dept. Inf. Comp. Sci., Univ. California, Irvine (1998), `http://www.ics.uci.edu/~mlearn/MLRepository.html`
26. Chang, C.-C., Lin, C.-J.: $LIBSVM$: a library for support vector machines (2001), `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
27. Malinowski, A., Yu, H.: Comparison of embedded system design for industrial applications. IEEE Trans. Ind. Informat. 7(2), 244–254 (2011)